

ESPECIAL

JavaOne 2007

Notícias em detalhe sobre o maior evento Java em uma das mais importantes edições da década



Edição 47 - Ano V - R\$ 11,90

 DevMedia
GROUP

Java ME: Desempenho e Portabilidade

Técnicas para tornar MIDlets mais portáteis e otimizados

JSF com MyFaces e Tomahawk

Conheça e use poderosos componentes JSF da Apache

Testes: Ferramentas e Boas Práticas

Aplicando os principais testes com frameworks e utilitários

Aplicação completa Java EE: EJB 3.0

Conclua a aplicação, adicionando robustez e escalabilidade

Relatórios

Avançados em Java

Crie relatórios crosstab com JasperReports e iReport

ECLIPSE 3.3

Novidades no Núcleo

Explore os principais novos recursos no IDE livre e saiba novidades sobre tecnologias relacionadas como RCP e SWT

Brinde!

Compre esta edição e ganhe **9 vídeo-aulas** sobre JSF

- Primeiros passos em JavaServer Faces – Instalação e configuração do ambiente (1 vídeo)
- Trabalhando com validadores no JavaServer Faces (3 vídeos)
- Conversores do JavaServer Faces (3 vídeos)
- Componentes avançados do JavaServer Faces (2 vídeos)



Suas aplicações em Java EE

são **robustas?**



¡Company® Framework

- JSF 1.2 com "especializações-chave" de componentes Apache Trinidad; **Novo!**
 - Orquestração de Web Beans com especializações do jBoss Seam; **Novo!**
 - EJB 3.0 - Session, Entity e JTA; **Novo!**
 - Validação Invariável via Annotations com especializações do Hibernate Validator; **Novo!**
 - OO ao Máximo: Padrões MVC-P para uso de Herança, Componentes, Relacionamentos N:M e Variáveis de Classe Persistentes; **Novo!**
- Exclusivo componente 'Explorador de Dados': visão treeview -> formulário, do 'tipo Windows Explorer'; **Novo!**
 - Solução de uso de OID para bases legadas em Oracle e DB2; **Novo!**
 - Lógicas de Programação 'por Exceção' via IoC (Inversão de Controle) e ID (Injeção de Dependência);
 - Logging e transações declarativas via AOP (Aspectos), com ou sem EJB;
 - Orquestração MVC simplificada via especializações utilizando Struts e JSF, simultaneamente;
 - Design-Patterns GOF e JEE, com códigos-fonte disponíveis;
 - Ajax automatizado via especializações DOJO e rotinas próprias;
 - Formulários tipo "cliente/servidor", para uso massivo, opcionalmente sem mouse;
 - Aplicações multi-layout via especializações Tiles;
 - Aplicações multi-pele via bibliotecas próprias de CSS;
 - RIA Flash com Laszlo;
 - Validação Variável via XML com especializações do Apache Validator.

E mais de uma centena de funcionalidades decisivas, especializadas de insumos Open-Source.

¡Company® Expert Utils

- Aplicações no padrão Maven 2.0, com diversas especializações de Build e Deploy; **Novo!**
- Eclipse IDE com plugins homologados para Subversion, além de CVS; **Novo!**
- Dezenas de plugins Eclipse 3.2 homologados e próprios;
- DVD para instalação do ambiente completo e funcional.

E muito mais.

Conheça o

¡Company® 5.0



¡Company® Developer Methods

- Metodologia de Construção JavaEE com lógicas MVC-P padrões de alto nível (CRUD, Tabular, Mestre-Detalhe, Mestre-Detalhe SubDetalhe, Relatórios, Consultas, Diálogos de Seleção etc.);
- Roteiros "passo-a-passo" pelos processos, orquestrados por Cheat-Sheets Eclipse;
- Documentação da metodologia e base de conhecimento disponíveis na Ajuda On-Line do Eclipse, para busca textual. **E muito mais.**

¡Company® Doc Generator

- Obtenção do **Manual Técnico da Aplicação** por consequência do processo, automaticamente, sem exigência de um "esforço à parte".

¡Company® PUP Template for EA

- Integração com CASE Enterprise Architect para geração de classes, mapeamento objeto-relacional e Casos de Uso;
- Possível customização para outros ambientes de modelagem.

¡Company® Code Generator

- Plugins Eclipse próprios para geração de artefatos "não-Java": jsp, jsf, mapeamentos, mensagens etc; **Novo!**
- Geração assistida de mapeamentos Hibernate, agora incluindo herança e componentes; **Novo!**
- Geração de aplicações configuradas para Maven, Testes Automatizados, Builds e Deploy imediatos; **Novo!**
- Geração de formulários Struts e JSF completos, multi-pele, multi-layout e internacionalizados;
- Geração sobre framework OO: resultados de produtividade, sem redundância, em horas, não dias! **E muito mais.**

Powerlogic Application Lifecycle Management

¡Company® QA Suite

Ambiente completo de QA incluindo integração contínua; produção simplificada de testes de unidade e funcionais; verificações de padrões corporativos com erros de compilação; além de dezenas de outras averiguações de parâmetros de qualidade, automatizadas.

¡Company® Security

Ambiente de configuração de segurança de acesso em padrão JAAS, com 'conforto visual associado' (modificação dinâmica de campos de formulário ou itens de menu); uso de certificados de cliente para permissão de acessos específicos etc.

¡Company® Monitor

Ambiente de monitoria de aplicações em produção, com coleta de "cliques" JMS, otimizada e não intrusiva. Informações chave para gestão tais como: história da sessão de usuários, perfil (Browser, SO, Resolução Vídeo), auditoria de login, disponibilidade via "agentes de monitoria"; dentre outros.

Powerlogic

Tecnologia Orientada ao Cliente

www.powerlogic.com.br :: plc@powerlogic.com.br

Fone: +55 (31) 3555 0050

¡Company®

Professional Java EE Open-Source

Solution Providers Powerlogic

Elosoft SC 47 2102 4444
Elosoft PR 47 2102 4444
Acrópolis ES 27 3337 5783
DSF SP 11 3142 8811
Plancorp SP 11 6847 4937

eService MG 31 3223 0400
Vinic PE 81 3224 3005
RCA CE 85 3279 7500
Ibrowse RJ 0800 541 4276
Ibrowse RS 51 3463 3131

Prime Informática SP 11 3037 3737
Netra Tecnologia BA 71 3311 7117
Netra Tecnologia DF 61 3962 2923
7COMm SP 11 3358 7700

Avanti RJ 21 3289 4500
PDCase MA 91 3224 5510
PDCase PA 98 3227 1260
Save Consulting MG 31 3234 6402

Conteúdo

Java ME



MINI-CURSO DE JAVA ME, PARTE 4

OSVALDO PINALI DOEDERLEIN

Analisando implementações de Java ME, otimizando código e trabalhando para aumentar a portabilidade e reduzir os efeitos da fragmentação

34

JavaOne 2007



OpenJDK

ABERTURA MÁXIMA E MÚLTIPLAS POSSIBILIDADES

LEONARDO GALVÃO

Analisando os grandes anúncios do maior evento Java – de iniciativas reestruturadoras a demonstrações espetaculares do uso da tecnologia

06



PROJETOS, TECNOLOGIAS – E SUA PRIMEIRA APLICAÇÃO JAVA FX

YARA H. SENGER

Novos projetos, versões e APIs, com uma exploração dos principais pontos da recém-lançada tecnologia JavaFX, e um mini-tutorial

10

Capa



ECLIPSE 3.3: NOVIDADES NO NÚCLEO

OSVALDO PINALI DOEDERLEIN

O essencial do que mudou nas partes centrais do Eclipse (JDT e Platform), e detalhes sobre algumas tecnologias que diferenciam o projeto, como SWT

14

Java EE / Web



APLICAÇÃO COMPLETA JAVA EE, PARTE 4 – EJB 3.0

RENATO BELLIA

Adicione escalabilidade e robustez à aplicação com a tecnologia EJB 3.0, e veja em ação estratégias de convivência entre duas arquiteturas

40



JSF COM MYFACES E TOMAHAWK

FRANCISCO CALAÇA XAVIER

Utilizando componentes JavaServer Faces que acrescentam às suas aplicações web recursos de interface gráfica dignos de aplicações desktop

60

Core



TESTES: BOAS PRÁTICAS E FERRAMENTAS

FERNANDO LOZANO

Examinando os principais tipos de testes, e conhecendo ferramentas e frameworks que apóiam a utilização do TDD e da Integração Contínua

24



RELATÓRIOS CROSSTAB COM JASPERREPORTS E IREPORT

JOSÉ TEIXEIRA DE CARVALHO NETO

Criando relatórios avançados e dinâmicos com agrupamentos e sumarização detalhados, usando o componente CrossTab

52



Ano V • Edição 47 • 2007 • ISSN 1676-8361

Direção

Diretor Editorial Leonardo Galvão

Diretor de Marketing Gladstone Matos

Edição

Publisher e Editor-Chefe

Leonardo Galvão (leonardo@javamagazine.com.br)

Editores adjuntos

Fernando Lozano (lozano@javamagazine.com.br)

Oswaldo Doederlein (osvaldo@javamagazine.com.br)

Colaboraram nesta edição

Fernando Lozano, Francisco Calaça Xavier, José Teixeira Neto, Leonardo Galvão, Oswaldo Pinali Doederlein, Renato Bellia, Yara H. Senger

Arte

Diagramação Vinicius O. Andrade

Ilustrações e Capa: Antonio Xavier

Produção

Gerência de Marketing Kaline Dolabella

Distribuição

Fernando Chinaglia Distribuidora S.A.

Rua Teodoro da Silva, 907, Grajaú - RJ

CEP 20563-900, (21) 3879-7766 - (21) 2577-6362

Atendimento ao leitor

A DevMedia possui uma Central de Atendimento on-line, onde você pode tirar suas dúvidas sobre serviços, enviar críticas e sugestões e falar com um de nossos atendentes. Através da nossa central também é possível alterar dados cadastrais, consultar o status de assinaturas e conferir a data de envio de suas revistas. Acesse www.devmedia.com.br/central, ou se preferir entre em contato conosco através do telefone 21 2283-9012.

Edições anteriores

Adquira as edições anteriores da revista Java Magazine ou de qualquer outra publicação do Grupo DevMedia de forma prática e segura, em www.devmedia.com.br/anteriores.

Publicidade

publicidade@javamagazine.com.br, 21 2213-0940

Anúncios – Anunciando nas publicações e nos sites do Grupo DevMedia, você divulga sua marca ou produto para mais de 100 mil desenvolvedores de todo o Brasil, em mais de 200 cidades. Solicite nossos Media Kits, com detalhes sobre preços e formatos de anúncios.

Reprints Editoriais – Se foi publicado na Java Magazine um artigo que possa alavancar as suas vendas, multiplique essa oportunidade! Solicite a reimpressão da matéria junto com a capa da edição em que saiu, e distribua esse reprint personalizado entre seus clientes.

Encarte de CDs – Faça como nossos maiores anunciantes. Encarte um CD com uma amostra de seus produtos na Java Magazine e atinja um público segmentado e formador de opinião.

Realização



Apoio



Carta ao Leitor

A Java Magazine participou do seu quinto JavaOne, e embora tenham se passado doze anos desde o lançamento da primeira versão do JDK, foi possível confirmar – direto do centro mundial da informática – que a tecnologia continua mostrando-se capaz de inovar e de seguir na direção que pedem os desenvolvedores e usuários. O leitor poderá ver analisadas nesta edição as boas novas que trouxe o JavaOne 2007, e conferir que o Java está se expandindo para alcançar novas plataformas, novas comunidades e novas linguagens.

Além da cobertura do maior evento de desenvolvedores, trazemos seis outros artigos que cobrem as três plataformas do Java. O Eclipse chega à versão 3.3 já contando com uma maturidade invejável no seu núcleo; mesmo assim há um número suficiente de novidades nesta parte do projeto para preencher dez páginas. Veja os detalhes no artigo de capa, onde são filtradas apenas as mais importantes adições e melhorias – para você não perder seu tempo com minúcias.

O mini-curso de Java ME tem aqui sua penúltima parte, abordando aspectos importantes no desenvolvimento de MIDlets: a busca por maior performance e o cuidado com a compatibilidade. A programação para celulares nos transporta por uma espécie de túnel do tempo: precisamos voltar a considerar pequenos detalhes, como técnicas otimizadas de cálculo, e até abdicar de algumas práticas OO que hoje usamos sem pensar duas vezes.

Nesta edição é concluída a aplicação completa, que vem sendo criada com as principais tecnologias do Java EE 5 e da Web 2.0. Além de introduzir o uso de Sessions Beans EJB 3.0, o autor mostra como migrar suavemente de uma arquitetura inteiramente baseada em POJOs, para uma que tira proveito máximo dos recursos oferecidos por servidores Java EE. O resultado será uma estrutura muito flexível, permitindo alternar entre as duas arquiteturas e mantê-las sincronizadas com pouco trabalho adicional.

Com o Java mantendo sua evolução no mercado de sistemas de informação, os relatórios ganham cada dia mais importância. Os usuários também se tornam mais exigentes, solicitando agregações e sumarizações que forneçam uma visão global dos dados. Uma solução popular para isso são os relatórios de tabulação cruzada ou “crosstab”. Você verá como criar esses relatórios em um tutorial avançado, que foca nas questões não-triviais do processo.

O JavaServer Faces vem em dose dupla nesta edição. Além das nove vídeo-aulas sobre o assunto dadas como brinde, temos um artigo que apresenta um dos projetos mais importantes na área – o MyFaces. Nele você conhece as partes mais populares, com atenção especial para o Tomahawk, que traz um conjunto poderoso de componentes visuais para a web.

Temos ainda um artigo aprofundado sobre testes de aplicações Java. A comunidade de desenvolvedores está exigindo mais informações sobre o processo de desenvolvimento e melhores práticas. Esse artigo vem suprir parte dessa demanda, com uma visão geral sobre uma atividade fundamental em qualquer processo de software, apresentando ferramentas que permitem criar, automatizar e acompanhar testes de vários tipos.

Boa leitura!

Leonardo Galvão





Conheça a assinatura Java Magazine Plus – mais conteúdo Java com mais de 110 vídeo-aulas para você e um Curso Online sobre Java ME

www.devmedia.com.br/javaplus

Amigo leitor, o conceito de assinatura da revista Java Magazine mudou. Agora, o novo assinante da revista Java Magazine possui uma assinatura de conteúdo Java Plus:

Java Plus = Revista impressa + vídeo-aulas + cursos online

O pacote Java Plus inclui:

- Recebimento mensal da revista Java Magazine (12 edições)
- Acesso, durante um ano, ao portal JavaPlus, contendo vídeo-aulas e cursos online.

Continuamos estendendo o portal com vídeo-aulas práticas voltadas ao desenvolvedor Java e também para iniciantes na tecnologia. Já são mais de 110 aulas. Acesse o portal JavaPlus e saiba como obter mais esse benefício em sua assinatura!

Na DevMedia, o leitor de banca também ganha!

Na compra desta edição, tenha acesso a 9 vídeo aulas sobre JSF!

- Primeiros passos em JSF – instalação e configuração (1 vídeo)
- Trabalhando com validadores no JSF (3 vídeos)
- Conversores do JSF (3 vídeos)
- Componentes avançados do JSF (2 vídeos)

Para visualizar acesse o link:

<http://www.devmedia.com.br/articles/listcomp.asp?keyword=jm47&codigobanca=borgir>

Confira as últimas novidades em vídeo-aulas

Acessando Banco de Dados utilizando o Spring Framework - Parte 4

Codificando a parte visual de uma aplicação mestre-detalle utilizando Spring e Swing. No final teremos pronta uma grade com todos os pedidos do banco de dados.

Utilizando Ponto Flutuante em CLDC 1.0

Como usar cálculos de ponto flutuante em aplicativos Java ME/CLDC 1.0. É usada a classe MicroFloat e como exemplo é desenvolvido um aplicativo utilizando diversos operadores (raiz quadrada, adição, subtração etc.).

Acessando Banco de Dados utilizando o Spring Framework - Parte 5

Continuamos a montar o esquema mestre-detalle em nossa aplicação Swing. No final teremos um exemplo prático de como implementar uma relação mestre-detalle, muito comum no dia-a-dia.

Introdução ao Wicket - Parte 2

Continuando o estudo sobre o framework Wicket. Segunda de uma série de três aulas.

Spring + JSF: Integrando Spring e JavaServer Faces – Parte 1

Como obter e instalar o ambiente para iniciar a integração entre o Spring e o JSF.

Spring + JSF: Integrando Spring e JavaServer Faces - Parte 2

Segunda parte do desenvolvimento de uma aplicação Java utilizando Spring e JSF.

Spring + JSF: Integrando Spring e JavaServer Faces - Parte 3

Passos finais na criação da aplicação JSF com Spring.

Struts 2: Uma Aplicação Completa - Parte 3

Continuando a série sobre o Struts 2.0.

Struts 2: Uma Aplicação Completa - Parte 1

Dando início à elaboração de uma aplicação de Business Intelligence completa utilizando o Struts Framework versão 2.0.

Struts 2: Uma Aplicação Completa - Parte 2

Segunda parte da elaboração de uma aplicação de BI completa utilizando o Struts 2.0.

Struts 2: Uma Aplicação Completa - Parte 4

Mais uma parte da série de vídeos demonstrando o Struts 2.0.

Integrando JSP e JavaScript

Veja como é simples e eficaz o uso de JavaScript junto com uma aplicação web JSP.

Introdução ao Maven

Apresenta a instalação, configuração e utilização básicas do Maven.

Padrões de Projeto na Prática – Composite

Conheça, através de exemplos práticos, o padrão de projeto Composite.

Padrões de Projeto na Prática – Adapter.

Conheça, através de exemplos práticos, o padrão de projeto Adapter.

JavaOne 2007

Abertura Máxima e Múltiplas Possibilidades

A Java Magazine esteve presente em mais um JavaOne, e aqui apresentamos um resumo do que aconteceu neste importante evento anual – que sempre traz dezenas de novidades, agrega milhares dos maiores experts mundiais em Java e determina os principais rumos da tecnologia.

A esperada General Session inicial trouxe muitas notícias importantes. Em sua pequena abertura, John Gage, Chief Scientist da Sun e o “anfitrião” do JavaOne desde a primeira edição do evento, convocou os mais de 12 mil presentes a participar ativamente de palestras, painéis, sub-eventos e exposições. E mantendo a tradição, falou sua já clássica frase “Don’t be shy” (não sejam tímidos), sem esquecer do lembrete anual: “Sejam brasileiros”.

Os números do crescimento e a maior abertura do Java foram a base do início da sessão. John Gage lembrou a importância dos celulares e realçou a expansão e a onipresença de Java: já há 6 milhões de desenvolvedores Java e 5,5 bilhões de dispositivos Java-enabled. Depois cedeu o palco a Rich Green, vice-presidente execu-

tivo de software e presença marcante em todo o evento. Green assumiu boa parte do restante da General Session.

Finalizada a liberação do Java SE como open source

O grande anúncio do primeiro dia foi o fim da liberação da implementação do Java SE da Sun como open source (sob licença GPL), dentro do projeto OpenJDK. Tudo que podia ser liberado no Java SE foi efetivamente liberado. Restam apenas alguns pacotes e classes que têm problemas de propriedade intelectual – cujos autores ou empresas criadoras não autorizaram a liberação sob condições razoáveis.

As partes que não foram liberadas, ditas *encumbered* (algo como “comprometidas”), constituem cerca de 4% das bibliotecas do JRE: são APIs de processamento de sons, rasterizadores de fontes e de gráficos, e algumas APIs de criptografia e SNMP. Para tornar o OpenJDK totalmente compilável, a Sun fornece implementações binárias dessa partes. Mas isso é só por enquanto.

A Red Hat, por exemplo, já está trabalhando para integrar implementações

livres criadas pela comunidade. A empresa também promete colocar sua equipe em campo para fechar as lacunas que restarem. E a comunidade open source em geral não está parada. Por exemplo, os colaboradores dos projetos Classpath e Kaffe já estão trabalhando para tornar possível uma distribuição com 100% do código disponível sob a GPL.

A disponibilidade de uma implementação livre do Java SE, atualizada, completa e compatível, pode parecer uma tecnicidade, mas não é. Uma implementação do Java SE sob a GPL pode ser incluída com facilidade na maioria das distribuições Linux, o que é essencial para o crescimento contínuo da tecnologia Java nessas plataformas. Com a mudança, você verá muito mais aplicações Java (servidores, aplicações desktop etc.) incluídas no Linux. E certamente crescerá a adoção de Java por desenvolvedores focados nesse sistema.

A licença open source estimula, claro, a colaboração. Uma prova impressionante disso aconteceu durante o evento. No primeiro dia do JavaOne, a equipe do Gentoo obteve o código recém-liberado do Java SE



Grandes entidades do software livre na mesma direção: Eben Moglen (Free Software Foundation); Dalibor Topic (Kaffe.org), Bruno Souza (NetBeans e OSI), Mark Wiegaard (Fundador do Classpath, hoje na Red Hat), Cliff Schmidt (Apache Software Foundation), Ian Murdock (Fundador do Debian, hoje Sun) e Simon Phipps (Chief Open Source Officer, Sun) discutem o futuro do Java open source.

Analizando as grandes novidades do maior evento Java – de iniciativas fundamentais a demonstrações espetaculares

LEONARDO GALVÃO

e colocou no ar um pacote do OpenJDK para essa distribuição – antes de a primeira General Session acabar. E menos de 24 horas depois, um desenvolvedor da Red Hat já tinha enviado patches para suprir parte do pacote de gráficos que não pôde ser tornado open source.

É claro, você também pode colaborar com o projeto OpenJDK. Um primeiro passo (que exigirá boa dose de coragem e paciência) é baixar todo o código do projeto e preparar um ambiente de build. Há um tutorial detalhado sobre isso em nb-openjdk.netbeans.org/get-and-build.html, onde são apresentados os passos necessários para organizar um ambiente completo no IDE NetBeans. Vale o trabalho – nem que seja somente para ter todo o coração do Java acessível na sua máquina, na ponta dos dedos!

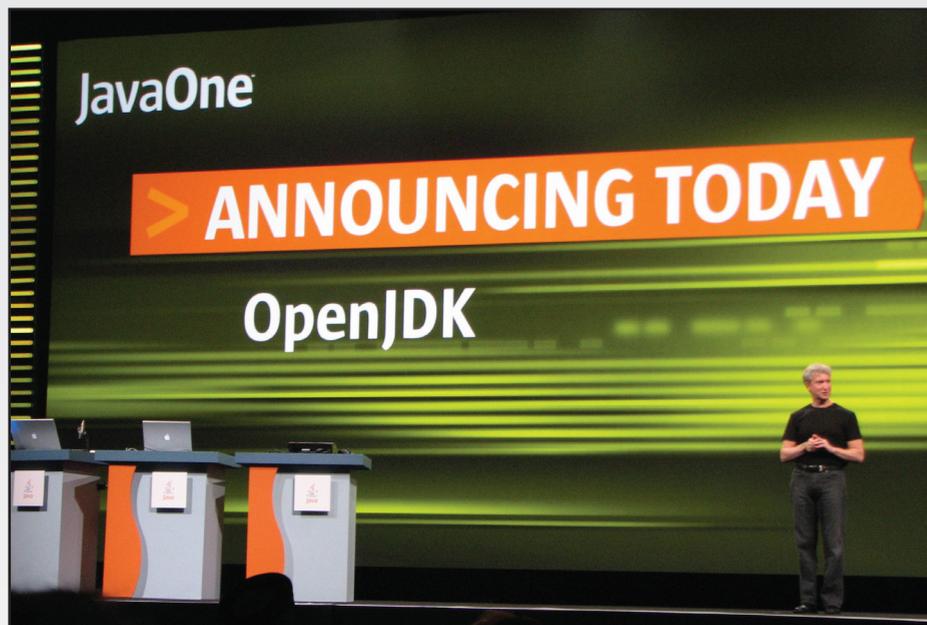
Interatividade com JavaFX

Outra importante notícia é o lançamento da tecnologia JavaFX. O JavaFX, que ainda está em franco desenvolvimento, traz novos recursos de interatividade na web e a plataformas Java móveis. É similar ao Flash com toques de AJAX, mas promete ser mais abrangente que ambas as tecnologias. Veja mais sobre as tecnologias “FX” no artigo de Yara H. Senger, logo depois deste.

Mais mobilidade, menos fragmentação

O JavaOne 2007 teve como um de seus focos principais a mobilidade. “Volume cria oportunidade” foi a frase que deu o tom à General Session da Sun sobre o desenvolvimento para dispositivos móveis. O mercado mundial para aplicações móveis não pára de crescer. É prevista a venda de 2,1 bilhões de aparelhos celulares com suporte a Java somente em 2007, além de 2,5 bilhões de Java Cards.

O padrão MSA (Mobile Service Architec-



OpenJDK completo: Rich Green anuncia a conclusão do processo de liberação como open source do Java SE da Sun

ture – JSR-248) foi concluído no final do ano passado, e neste JavaOne a Nokia anunciou que uma nova série de dispositivos com sua plataforma mais popular – a S40 – terá suporte completo à MSA. Considerando que os telefones baseados na S40 são os mais usados no Brasil, é de esperar a sua chegada aqui assim que forem lançados mundialmente – abrindo novas oportunidades para os desenvolvedores *mobile* nacionais. As ferramentas para desenvolvimento Java ME para a nova plataforma estarão disponíveis em junho.

A presença maciça de dispositivos que atendem às especificações exigentes do MSA (MIDP 2.1, CLDC 1.1; outras APIs; mais memória e maior capacidade de processamento) reduz a fragmentação e simplifica o desenvolvimento de MIDlets. Os desenvolvedores poderão contar com recursos padronizados e focar mais em inovações – em vez de perderem tempo testando aplicações em dezenas de dis-

positivos diferentes com níveis de suporte variados às APIs do Java ME.

A Motorola, a outra gigante no mercado móvel, não está parada. A empresa foi uma das principais patrocinadoras do evento (junto com Intel e Oracle) e divulgou um ambiente completo de desenvolvimento para celulares Motorola, o Motodev Studio. Com esse conjunto de ferramentas, que é baseado no Eclipse, é possível depurar e executar MIDlets em emuladores especializados e instalá-los nos dispositivos reais facilmente.

Consumer JRE

Uma das principais áreas enfocada na construção do futuro Java SE 7 é o suporte à modularização. Esta evolução fundamental na plataforma é importante para as grandes aplicações Java EE (simplificando a componentização) e fundamental para melhorar o processo de distribuição do JRE para aplicações clientes.

Mas o mercado e a comunidade não querem esperar até o Java SE 7 para obter os primeiros benefícios. A resposta da Sun é o “Consumer JRE”, nome dado ao conjunto de novas funcionalidades no próximo update do Java SE 6, previsto para lançamento no início de 2008. A estrutura do runtime está sendo inteiramente revista para permitir a quebra do conjunto de APIs em vários pequenos pedaços, que podem ser carregados sob demanda pelas aplicações. Isso reduzirá grandemente o tempo de download e de inicialização de aplicações desktop distribuídas pela web

(por exemplo, via Java Web Start), pois apenas o mínimo – o chamado “Java Kernel” – será carregado inicialmente. Também será possível configurar exatamente os módulos necessários para a aplicação. Testes preliminares apontam para um ganho de mais de 60% no tamanho dos downloads, para uma aplicação Swing típica.

O Consumer JRE inclui uma série de outros recursos e melhorias. Um *Deployment Toolkit* vai facilitar a detecção e a instalação do JRE, e a experiência de instalação será melhorada, tornando-se mais amigável e rápida. O update trará ainda um novo look-

and-feel chamado Nimbus (baseado no Synth) e grandes melhorias no desempenho de aplicações Swing no Windows, especialmente as que envolvem operações 3D e o processamento avançado de imagens.

Uma dúzia de demonstrações

A última General Session do JavaOne, sempre dirigida pelo criador do Java, James Gosling, tradicionalmente enfoca demonstrações do uso de Java em diversas áreas. Nesta edição, foram duas horas intensas em que foram apresentados mais de dez casos de uso da tecnologia, abrangendo desde novos recursos em ferramentas até a operação de veículos submarinos.

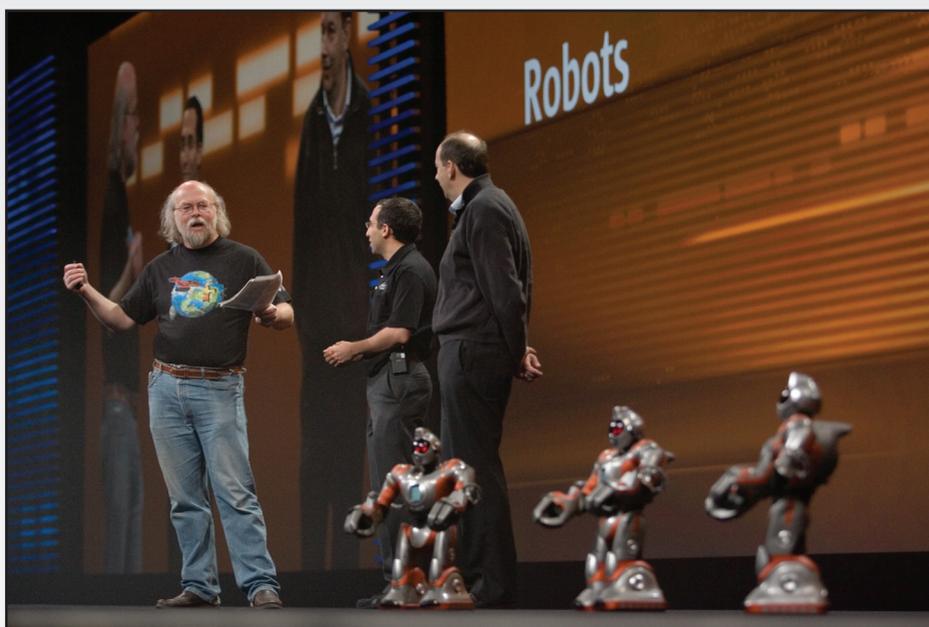
Blu-ray

Uma das demonstrações mais interessantes tratou do uso de Java em discos e equipamentos Blu-ray, a nova geração de discos multimídia projetada para substituir os atuais DVDs. Um único disco Blu-ray (com duas camadas) é capaz de armazenar 50 gigabytes de dados, o suficiente para 9 horas de vídeo em alta definição (HD) ou 23 horas no padrão usado atualmente em DVDs. (Curiosidade: O Blu-ray é o padrão adotado pelo novo PlayStation 3.)

Os tocadores de discos Blu-ray incluem uma JVM de fábrica, que é parte da plataforma BD-J (Blu-ray Disc Java). A BD-J permite implementar menus e muitos outros recursos interativos dos discos usando Java. Isso significa todo um novo mercado para desenvolvedores Java, especialmente os com inclinação multimídia.

O anúncio de Java no Blu-ray foi feito no JavaOne 2005 (quando a tecnologia estava apenas saindo da fase dos protótipos), mas neste ano vários produtos concretos foram mostrados. Filmes usando recursos ricos de interatividade, como jogos de perguntas que se adaptam à cena atual do filme, e mapas indicando a posição geográfica dos protagonistas, foram exibidos na gigantesca tela do Moscone Center.

Os jogos e outras aplicações embutidas nos novos discos podem usar recursos de rede, permitindo interações entre várias pessoas distribuídas geograficamente, assistindo ao filme ao mesmo tempo. Em uma das demonstrações, o filme XMen 3 era exibido com um jogo de perguntas so-



Java no controle de robôs: James Gosling discute a versatilidade dos novos brinquedos de adultos



Interatividade do Blu-ray em ação: jogos contextuais em rede enriquecem a experiência do filme

breposto ao vídeo, com jogadores interconectados em rede. Toda a parte interativa foi criada em Java.

Project Wonderland

A demonstração do projeto Wonderland foi outro destaque da sessão de Gosling. Um dos criadores do projeto mostrou um ambiente virtual em ação, usando um avatar do próprio Gosling para navegar pelas cenas tridimensionais. O objetivo do Projeto Wonderland é fornecer um ambiente multi-usuário virtual que tenha robustez suficiente para permitir a colaboração em nível profissional, de forma aplicável por empresas.

Os participantes do ambiente podem executar colaborativamente aplicações completas, como browsers web e aplicativos de escritório. É possível, por exemplo, exibir apresentações de slides de forma distribuída. Outra aplicação interessante é o trabalho conjunto na criação de aplicações Java: é possível operar um IDE remotamente através do ambiente 3D, e com isso fazer o desenvolvimento colaborativo, trocar idéias sobre bugs, decisões arquiteturais etc.

O Wonderland usa a infra-estrutura do Darkstar, um projeto popular dentro da comunidade de jogos tridimensionais em Java. São também usadas a API Java 3D e uma tecnologia de reprodução de sons do Sun Labs. Você pode fazer o download do código do Wonderland e ver vídeos do projeto em ação no seu site oficial no java.net (veja links).

NetBeans 6

O novo NetBeans, com lançamento previsto para novembro deste ano, teve muito espaço na conferência e foi a base de uma das demonstrações mais ágeis da última General Session. O IDE vem com melhorias significativas, especialmente no editor de código, que foi inteiramente refeito. O suporte a outras linguagens também está evoluindo rapidamente. O projeto Schliemann, por exemplo, permite criar recursos de suporte a linguagens de scripting, com auto-completamento, realce de sintaxe e mais. Há também toda uma nova infra-estrutura de refactoring, baseada no projeto Jackpot.

Na demonstração, foi criada – em minutos – uma aplicação Java com JPA e código Ruby on Rails (este último gerado com o

novo plug-in de suporte a JRuby). Pela velocidade que o código surgia na tela e era manipulado, refatorado e corrigido... ficaram claras as vantagens em produtividade oferecidas pelo novo editor do NetBeans. Vale fazer o download e conferir!

Outras demonstrações

O rol de demonstrações foi muito variado neste ano. Veja um pouco mais sobre o que foi apresentado:

- **Profiling com DTrace** – O DTrace, ferramenta de profiling da Sun que é um dos líderes na área, está cada vez melhor integrada ao Java. Você pode usar o DTrace para analisar o desempenho de aplicações Java minuciosamente. Os resultados incluem gráficos interativos que dão acesso ao ponto exato do código correspondente. Na demonstração, vários aspectos de uma aplicação foram analisados: do acesso a disco em nível de bytes ao uso de memória e processador.

- **Robôs** – Quase todos os anos, é introduzido no JavaOne algum brinquedo sofisticado. Nesta edição, foi a vez dos robôs RoboSapien da empresa Wowee, que deram um show de dança coreografado ao som da música "I'll Survive". Os robôs vêm de fábrica com uma máquina virtual Java e ferramentas para desenvolvimento de aplicações; têm vários sensores e os movimentos possíveis são bastante sofisticados. Outra demonstração relacionada foi de um grande robô de manufatura trazido da Suécia. Apesar de possuir enorme força e velocidade, o robô industrial controlado por Java Real Time mostrou-se capaz de realizar atividades sutis como traçar desenhos de alta precisão.

- **Processamento de vídeo** – A tecnologia Java 2D, que vem sendo usada com frequência para implementar aplicações desktop sofisticadas, foi o foco de uma demonstração visual e interativa. Uma câmera posicionada no palco filmava a platéia e as imagens eram exibidas dentro de um software que permitia fazer ajustes e manipulação do vídeo em tempo real.

- **Veículos autônomos** – Um veículo submersível e um helicóptero (ambos operados por sistemas em Java) foram o tema das últimas demonstrações. Os equipamentos operados pela tecnologia estão ficando cada

vez mais avançados e já estão cumprindo missões importantes em campo.

Conclusões

Java completou doze anos e, mesmo depois de tanto tempo, pudemos ver neste JavaOne 2007 que a tecnologia mantém acelerado seu ritmo de inovação. Agora, com o Java SE open source e o crescimento permanente do Java nos dispositivos móveis, bem como sua presença em vários produtos da eletrônica de consumo, os horizontes estão ainda mais abertos para a tecnologia – e para nós desenvolvedores ●



Leonardo Galvão

(leonardo@javamagazine.com.br)

é bacharel em Ciência da Computação, fundador da Java Magazine e editor-chefe da publicação desde sua primeira edição. É diretor de comunicação do SouJava nacional e um dos líderes da JUGs Community no portal Java.net, além de ter feito parte do Program Committee do JavaOne 2007.



openjdk.dev.java.net

OpenJDK: o projeto com o código GPL do Java SE da Sun.

forum.nokia.com/main/platforms/s40.

Site da Plataforma S40. A quinta edição, com suporte completo ao padrão MSA está chegando.

developer.motorola.com/docstools/motodevstudio

Motodev Studio: conjunto de ferramentas para criação e teste de aplicações Java ME para celulares Motorola.

java.sun.com/developer/technicalArticles/javase/consumerjre

Mais sobre o Consumer JRE.

blueboard.com/bluray

Site informativo sobre a plataforma BD-J, que é baseada no Personal Basis Profile do Java ME.

www.blu-raydisc.com

Tecnologia Blu-ray para consumidores.

wowee.com/robosapien

Robôs sofisticados movidos a Java.

lg3d-wonderland.dev.java.net.

Projeto Wonderland: criação de ambientes tridimensionais colaborativos.



JavaOne 2007

Tecnologias e projetos – e uma primeira

Este ano o JavaOne foi surpreendente. Geralmente não esperamos muitas novidades, devido ao crescente número de blogs, listas de discussão e fóruns, que nos deixam sempre por dentro das últimas notícias – mesmo assim o maior evento Java é sempre um excelente termômetro que indica quais dos planos prometidos no último ano foram cumpridos, quais não saíram do papel e quais estão de fato em andamento. Mas neste ano foi diferente. Voltamos a ter algumas grandes novidades, que estavam “guardadas”, como a família de produtos JavaFX. Tivemos também confirmações de tecnologias importantes, como Web 2.0

e linguagens de scripting, sobre as quais pudemos ver muitas discussões mais maduras neste ano.

Selecionar o que assistir entre as centenas de palestras do JavaOne é sempre uma tarefa complexa. Saímos do evento querendo ter visto mais Sessions e mais BoFs, e desejando ter aproveitado ainda mais as outras atividades que a conferência oferece – entre elas, contatos e discussões com as pessoas que definem a tecnologia.

Para os que não foram, e também para os que perderam palestras importantes, os PDFs e o som das Sessões Técnicas são disponibilizados no site do JavaOne. E as General Sessions podem ser vistas na inte-

gra, com vídeo e áudio. Vale a pena assistir pelo menos à abertura do evento, que é sempre a melhor de todas. Veja nos links como acessar esses recursos pós-evento.

Neste artigo apresento uma seleção das novidades do JavaOne 2007, com ênfase em tecnologias e projetos, como JavaFX, GlassFish e Web 2.0 e 3.0.

JavaFX

Um dos grandes anúncios do JavaOne 2007 foi sem dúvida a nova família de produtos chamada JavaFX, composta inicialmente pelas tecnologias JavaFX Script e JavaFX Mobile. As duas aproveitam recursos importantes da plataforma Java como portabilidade, segurança e conectividade com aplicações corporativas – e trazem mais interatividade e novos recursos visuais para aplicações clientes.

Foram mostradas algumas aplicações, que estão disponíveis no site do projeto OpenJFX no java.net. As demonstrações são recheadas de recursos de transparência e movimento (como *fade-in* e *fade-out*), e lembram aplicações criadas com Flash; veja um exemplo na **Figura 1**. As demonstrações são distribuídas como aplicações Java para desktop, via Java Web Start.

JavaFX Script

A linguagem JavaFX Script foi criada para simplificar o desenvolvimento de aplicações de interface rica (RIA – Rich Interface Applications), e apresenta como principais diferenciais o aproveitamento de recursos importantes da plataforma Java como segurança e portabilidade, permitindo acesso a grande parte das APIs Java.

Um diferencial em relação a outras linguagens de script é que JavaFX Script é *statically typed*; ou seja, os tipos são definidos estaticamente em tempo de compilação, facilitando por exemplo a implementação



Figura 1. Aplicação JavaFX em ação: alta interatividade com aplicações ricas

aplicação JavaFX

Um resumo de novas tecnologias apresentadas e futuras versões de produtos, além de um mini-tutorial sobre a JavaFX Script

YARA H. SENER

de auto-complete e outros recursos de produtividade em IDEs. Veja um pouco mais sobre a linguagem no **quadro** "Destaques de sintaxe da JavaFX Script". O **quadro** "Primeira aplicação com JavaFX script" apresenta um mini-tutorial sobre como usar a tecnologia.

JavaFX Mobile

No JavaOne 2006 foi divulgado com grande destaque, um celular especial com sistema operacional baseado em Java e suportando aplicações ricas – tudo o que um programador Java gostaria de ver, na palma da mão. Era fácil visualizar o enorme universo de aplicações que poderia ser criado para o novo dispositivo. O nome do celular era Jasper, e era produzido por uma empresa chamada SavaJe.

Este ano foi apresentado o JavaFX Mobile (veja a **Figura 2**), um sistema operacional e plataforma para aplicações, baseado em Java e Linux. Com o JavaFX Mobile, será possível criar aplicações móveis ricas e interativas, com a capacidade de buscar dados ou serviços pela rede, e com suporte a tecnologias Java que vão além do Java ME.

Qualquer semelhança entre o JavaFX Mobile e o SavaJe não é coincidência. A empresa criadora do SavaJe foi comprada pela Sun, que aproveitou muito da sua tecnologia.

Perspectivas

Quando ouvimos o primeiro anúncio sobre JavaFX, não estava muito claro quais eram os objetivos e os benefícios. Principalmente queríamos saber de onde a tecnologia havia surgido e o seu grau de maturidade. Ao realizar alguns testes com JavaFX Script, ficou claro que já existe algo concreto; o plug-in para o NetBeans funciona, os exemplos estão rodando e a documentação é efetiva para iniciar os estudos. A JavaFX Mobile no entanto me pareceu estar um pouco mais incipiente.

Mas as tecnologias FX devem amadurecer rapidamente e receber muitos investimentos, considerando-se o destaque dado a elas neste JavaOne. Um bom sinal é que o próprio James Gosling está agora trabalhando no projeto. Outro passo que deve acelerar a sua evolução é a futura liberação de todos os produtos JavaFX como open source, prometida pela Sun.

Realizações e planos para o GlassFish

O GlassFish, o servidor Java EE 5 open source suportado pela Sun, foi o primeiro a implementar toda a especificação do novo Java Enterprise Edition. A primeira versão foi finalizada em maio de 2006, e durante este JavaOne foi lançada a versão Beta 2. O GlassFish V2 final está previsto para setembro de 2007. Várias palestras no evento discutiram detalhadamente o projeto.

As principais melhorias para a versão 2 são o suporte a JAX-WS 2.1 e BPEL, clustering, balanceamento de carga e

alta disponibilidade, além da inclusão do Open ESB 2.0 (baseado no padrão JBI). A usabilidade também evoluiu: agora há um download único e menor, múltiplos perfis de usuário, uma central de atualização e novas ferramentas de administração para JSF e Ajax. Um destaque é a melhoria de dez vezes no desempenho do container web do servidor. Outras novidades importantes são o menor tempo de inicialização (o servidor é carregado com o menor número de módulos possíveis), e o suporte integrado a Ajax e a projetos e linguagens de scripting, como jMaki, DynaFaces, WoodStock e JSF Templating.

Muito se falou também sobre a futura versão 3 do GlassFish. Veja alguns planos discutidos:

- O servidor vai ficar ainda menor, mais modular e mais rápido. A meta é que o kernel chegue a menos de 100 Kb, e que o tempo de inicialização seja menor que um segundo.



Figura 2. Rich Green anuncia o JavaFX Mobile

Primeira aplicação com JavaFX Script

Vamos criar um pequeno exemplo com JavaFX Script para matar (ou aumentar!) a sua curiosidade.

Instalando o plug-in de JavaFX no NetBeans

Usaremos o novo plug-in para NetBeans. Embora esse plug-in ainda seja bastante simples (sem auto-complete, refactoring e formatação de código), vale a pena instalá-lo, pois ele simplifica o processo de execução e configuração para deployment.

Instale o plug-in no NetBeans seguindo estes passos:

1. Acesse o comando *Tools|Update Center*, selecione o centro de atualização *NetBeans Update Center Beta* e clique em *Next*.
2. Selecione *JavaFXEditor*, *JavaFXUserLibrary* e *JavaFXLibrary* e clique em *Next* para realizar o download.
3. No menu *Tools*, clique em *Modules Manager* e selecione os plug-ins para instalação.

Criando a aplicação

Já podemos criar o formulário de exemplo. Clique com o botão direito no nome do projeto, selecione *New>File or Folder* e escolha a categoria *Other*. Selecione então o tipo *JavaFX File* e forneça o nome "Formulario". Isso cria o esqueleto de um formulário, no arquivo *Formulario.fx*. A **Listagem Q1** mostra o script do formulário depois de várias adições nossas.

Executando o script no NetBeans

Para executar o exemplo, siga estes passos:

1. Clique com o botão direito sobre o projeto e selecione a opção *Properties*; depois selecione a categoria *Run*.
2. No campo de entrada *Arguments* forneça o nome do script, no nosso caso *Formulario*.
3. Execute o projeto (clcando F6 se o projeto estiver configurado como *Main Project*).
4. O resultado é mostrado na **Figura Q1**.

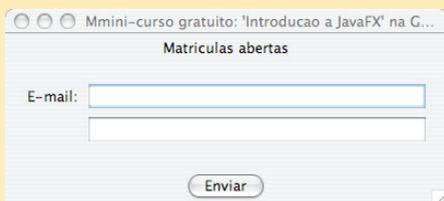


Figura Q1. Primeiro formulário com JavaFX Script

Listagem Q1. Exemplo de formulário em JavaFX Script

```
import javafx.ui.*;

class Formulario {
    attribute email: String;
    attribute nome: String;
}

var model = Formulario {
    email: ""
    nome: ""
};

Frame {
    title: "Campanha Globalcode Open-source education"
    content: BorderPanel {
        top: FlowPanel {
            content: SimpleLabel {
                text: "Mini-curso gratuito: 'Introducao a JavaFX'"
            }
        }
    }

    center: GroupPanel {
        var linhaEmail = Row { alignment: BASELINE }
        var linhaNome = Row { alignment: BASELINE }
        var colunaDescricao = Column { alignment: TRAILING }
        var colunaCampos = Column { alignment: LEADING resizable: true }
        rows: [linhaEmail, linhaNome]
        columns: [colunaDescricao, colunaCampos]
        content:

        [SimpleLabel {
            row: linhaEmail
            column: colunaDescricao
            text: "E-mail:"
        },

        TextField {
            row: linhaEmail
            column: colunaCampos
            columns: 25
            value: bind model.email
        },

        SimpleLabel {
            row: linhaNome
            column: colunaCampos
            text: "Nome:"
        },

        TextField {
            row: linhaNome
            column: colunaCampos
            columns: 25
            value: bind model.nome
        }
    ]

    bottom: FlowPanel {
        content:

        Button {
            text: "Enviar"
            action: operation() {
                /* Poderíamos realizar qualquer operação aqui,
                que seria executada ao clicar o botão */
            }
        }
    }
}

visible: true
};
```

- Haverá suporte a quatro tipos de containers: Java EE, Ruby on Rails (JRuby), Phobos e PHP. O GlassFish com isso deixa de ser um container Java EE e passa a ser um produto muito mais geral.

- O V3 incluirá o projeto SailFin, contribuição da Ericsson. Esse projeto é baseado em Servlets SIP (Session Initiation Protocol) e implementa a JSR-116 (SIP Servlet API), visando também compatibilidade com a JSR-289 (versão 1.1 da mesma API). Os Servlets SIP estão por trás de serviços muito utilizados atualmente, como voz sobre IP (VoIP), mensagens instantâneas e conferências via web. Portanto esta contribuição deve agregar muito valor ao GlassFish.

Você já pode começar a acompanhar o desenvolvimento da nova versão: um preview do GlassFish V3 já está disponível para download.

Web 2.0 e Web 3.0

Um dos grandes temas do JavaOne passado foi a Web 2.0, com ênfase em Ajax. É sempre bom ver planos do ano anterior tornarem-se realidade, e verificar que as tecnologias envolvidas estão ficando mais fortes. Neste ano, houve muitas palestras sobre Web 2.0 e Ajax novamente, mas o assunto foi abordado de forma muito mais madura. Já podíamos ver melhores práticas, comparações de performance, análises de suporte por ferramentas, discussão de questões de segurança e testes, etc.

Embora a Web 2.0 (que enfatiza interatividade e *mashups*) ainda não seja a realidade para a grande maioria das aplicações, já existe infra-estrutura em Java para tornar essas aplicações realidade. Por isso, parte da comunidade (de vanguarda) já começou a discutir a Web 3.0, e há bastante gente trabalhando nisto.

Na popular palestra “Web 3.0: This is the Semantic Web”, os apresentadores previram o domínio da Web 3.0 nos anos entre 2010 e 2020, com a Web 2.0 prevalecendo entre 2000 e 2010. Veja algumas palavras chave da Web 3.0:

- **Web Semântica** – É uma evolução da web, na qual o conteúdo pode ser descrito em linguagem natural e também em formatos que que possam ser interpretados e entendidos por outras aplicações. Permite integrar,

Destaques da sintaxe da JavaFX Script

Aqui apresentamos algumas características da nova linguagem JavaFX Script

- Pode-se importar pacotes, instanciar classes e executar métodos de APIs Java.
- Variáveis são declaradas com **var** – por exemplo, **var variableName : typeName [?,+,*] = initializer;**
- A inserção, remoção e busca em arrays têm notação diferenciada e simplificada em relação a Java.
- Além dos operadores do Java, foram adicionados novos, como o **bind** que suporta a “avaliação incremental”, entre outras funcionalidades.
- A concatenação de Strings com variáveis pode ser feita de forma bastante simples, utilizando o nome da variável entre chaves – ex: **var s = “Hello {name}”.**

- O suporte à formatação de datas e Strings é simplificado em comparação com Java.

- Pode-se criar triggers de criação, inserção e remoção de objetos através da palavra reservada **trigger** e algumas outras. Estas palavras definem os tipos das triggers ao invés de construtores e métodos de acesso (getters e setters). Um trigger de inserção, por exemplo, será executado quando um valor for atribuído a um atributo; um trigger de criação será executado quando um objeto da classe for instanciado, e assim por diante.

- Além das estruturas de controle de fluxo **if, while, try-catch-finally, for, return, throw, continue** e **break**, bastante similares às do Java, é suportada a estrutura **do-later**, para programação assíncrona.

compartilhar e obter mais informações mais facilmente, e com maior precisão.

- **RDF** (Resource Description Framework) – Uma linguagem usada para representar metadados na web, endossada pelo consórcio W3C, com sintaxe baseada em URIs e XML. Para modelar os metadados são utilizadas triplas *assunto-propriedade-objeto*.

- **REST** (REpresentational State Transfer) – É um estilo de arquitetura para sistemas de hipermídia distribuídos. O termo foi definido por um dos criadores do protocolo HTTP, Roy Fielding. É também utilizado para descrever interfaces baseadas em XML sobre HTTP com uma camada adicional, como SOAP.

O trabalho com a Web 3.0 está apenas começando, mas a tecnologia Java já está na dianteira nesta área. Por exemplo, mais de metade das aplicações de Web Semântica disponíveis são baseadas em Java.

Conclusões

Neste JavaOne, vimos como a comunidade de Java permanece ativa, e que continua gerando muitas tecnologias e inovações. Se você estava esperando um momento de paz, sem ter que desbravar muitas coisas novas, estudando apenas uma ou outra API, pense novamente! Há muito de novo para descobrir e explorar em Java e nas suas novas tecnologias relacionadas. ●

openjfx.dev.java.net

Home do projeto OpenJFX no java.net

java.sun.com/javaone/sf/sessions/general

Webcasts com o conteúdo completo das General Sessions

developers.sun.com/learning/javaoneonline/index.jsp

Slides das Technical Sessions

glassfish.dev.java.net

Projeto Glassfish

woodstok.dev.java.net

Projeto Woodstock, biblioteca de componentes JSF

jsftemplating.dev.java.net

JSF Templating, alternativa de templating para JSF

w3.org/RDF

Especificações de RDF da W3C



Yara M. H. Senger

(yara@globalcode.com.br)

é formada em Ciências da Computação na USP em São Carlos, e é especialista em desenvolvimento web; possui as certificações SCJA, SCJP e SCWCD. Atualmente é Instrutora e Diretora Educacional da Globalcode, criadora e coordenadora de diversos cursos das carreiras Academia do Java e Academia do Web Developer.

Eclipse 3.3

Conheça o que há de novo no núcleo do IDE

Chegamos a outro mês de junho e, como já é tradicional, a outro release do IDE Eclipse. Na Edição 37, conhecemos o projeto Callisto, desenvolvimento coordenado do Eclipse 3.2 com um grande número de complementos hoje considerados essenciais. Dando continuidade a este plano, o satélite da vez é o Europa. Imagino se os próximos releases serão Io e Ganymede?¹ Mas como a revista é de software e não de astronomia, vamos deixar essa especulação de lado e examinar as novidades e aperfeiçoamentos do IDE mais usado pela comunidade Java.

Porém, antes de realmente começar, algumas palavras sobre este artigo. Todo ano escrevo sobre o novo Eclipse, e é difícil “encaixar” centenas de novidades (mais discussões de tecnologias e assuntos rele-

vantes) em poucas páginas. Procuo evitar uma repetição de informações presentes no site eclipse.org, como os relatórios *New and Noteworthy*. Minha intenção aqui é favorecer conteúdo novo e importante.

Decidi, dessa vez, eliminar por completo qualquer apresentação de melhorias superficiais. Assim, você não encontrará aqui nem uma única informação como:

- “A GUI está mais bonita: abas de views desabilitadas têm cantos com curvas...”
- “No depurador, o *Step Into* numa expressão complexa é mais fácil: segure Ctrl+Alt e...”
- “O diálogo de configuração de associações de teclas a comandos foi melhorado...”

Sim, o Eclipse 3.3 tem um enorme número de melhorias desse tipo, que tornam funcionalidades preexistentes mais fáceis de usar, mais organizadas, agradáveis,

produtivas, completas. Mas aqui teremos que deixar de lado estes itens “decorativos” ou mesmo incrementais. Cobriremos somente as novas funcionalidades. Exceção apenas para melhorias críticas na facilidade de uso, como os novos refactorings “inline”. Para as minúcias, remetemos o leitor aos relatórios *New and Noteworthy* (cujos links estão disponíveis nas páginas de download de milestones e de versões finais do Eclipse).

Aproveitaremos o espaço aberto por esta abordagem, para não só falar das novidades desta versão, mas também para discutir sobre tópicos interessantes e atuais ligados ao Eclipse. Por exemplo, daremos uma atenção especial à SWT, assunto importante para desenvolvedores que usam o Eclipse RCP (Rich Client Platform).

A evolução do Eclipse

O produto principal da Fundação Eclipse, o JDT (Java Development Tools²) teve desde o começo uma excelente reputação, oferecendo um editor de código e gerenciador de projetos excepcionais, um compilador com capacidades inéditas (incremental e integrado às funcionalidades do IDE como editor ou refactorings), uma arquitetura de plug-ins sofisticada e extensível, um novo toolkit de GUI considerado na época superior ao Swing... Por esse resumo podemos ver que desde o começo o projeto Eclipse não se limitou ao IDE. Para mais detalhes, veja o quadro “Eclipse: o IDE e a tecnologia”.

Por outro lado, o Eclipse era pobre na cobertura de técnicas especializadas de desenvolvimento. Não tinha editores visuais de GUI, suporte a J2EE/Java EE, nem coisas bem mais básicas como um editor de XML. Embora a sua ampla comunidade

² Também conhecido como “Eclipse IDE” ou simplesmente “Eclipse”, termos potencialmente ambíguos, mas que utilizaremos aqui dessa maneira para não complicar.

¹ São os nomes dos satélites “galileanos” de Júpiter: os quatro maiores, descobertos por Galileu em 1610.



e da Plataforma

O essencial do que mudou nas partes fundamentais do Eclipse, e detalhes sobre algumas tecnologias que diferenciam o projeto, como a SWT

OSVALDO PINALI DOEDERLEIN

de desenvolvedores de plug-ins garantisse que nenhuma necessidade ficaria sem suporte, é bom ter plug-ins de terceiros, mas não é bom depender somente deles.

Recursos incorporados aos projetos oficiais da Fundação Eclipse são quase sempre muito superiores aos produzidos por terceiros (a maioria das exceções são produtos comerciais). Criar ferramentas de desenvolvimento modernas exige muitos recursos. Indivíduos ou projetos open source pequenos e sem patrocínio têm pouca chance de competir com equipes grandes, bem estruturadas e sem dúvida bem pagas pela multidão de grandes corporações que sustentam a Fundação Eclipse³.

Há cerca de um ano, com o Callisto, parecia que o Eclipse recuperaria todo o atraso em relação a outros IDEs na abrangência de ferramentas. Mas o mundo não ficou parado enquanto o Eclipse corria atrás de funcionalidades como criação de JSPs e EJBs. Assim, quando o WTP (Web Tools Project) 1.x chegou, a primeira reação pode ter sido de satisfação com as funcionalidades disponíveis. Mas a segunda poderia ser: OK, temos o feijão-com-arroz do J2EE 1.4, mas cadê o resto? Faltava ainda suporte a Struts, JSF, edição visual de GUIs, mapeamento objeto-relacional, e mesmo suporte a Java EE 5 e JPA/EJB 3 (na época ainda em desenvolvimento, mas já contando com suporte inicial de outros IDEs).

O WTP continuou atrás da competição, especialmente após o NetBeans 5.5. E se o suporte para Java EE ainda não era ideal, o que dizer do Java ME, até há pouco sem absolutamente nenhum suporte? Teríamos, então, que esperar mais uma geração da família de ferramentas Eclipse para atingir um status de estado da arte em todas

³ Isso não é exclusivo ao Eclipse. Usando o NetBeans, percebe-se também uma grande diferença de qualidade e funcionalidade entre os recursos embutidos no IDE (ou seus "Packs") e os módulos contribuídos por terceiros.

as funcionalidades agora consideradas essenciais.

O Europa ou Eclipse 3.3 traz grandes atualizações nessas áreas anteriormente defasadas. Veremos se estas melhorias foram suficientes ao longo deste ano, pois o presente artigo é focado apenas no JDT. Aguarde por edições futuras da Java Magazine, onde teremos artigos mais específicos abordando as novidades do WTP e de outras ferramentas do Europa (mas mencionaremos brevemente algumas destas novidades aqui).

Aqui estamos falando do núcleo do Eclipse, que mesmo na liderança, também não ficou repousando sobre suas glórias e continua avançando. Há muitos aperfeiçoamentos na versão 3.3. São coisas cada vez mais evolucionárias, que tornam os recursos que já existiam melhores e mais refinados, ou acrescentam novos recursos cada vez mais discretos. Mas isso não significa que o update não valha a pena. Pelo contrário, o acúmulo de muitos "toques sutis" pode fazer a diferença entre uma ferramenta ótima e uma excepcional – mais inteligente, eficiente e ergonômica.

SWT

A SWT, o toolkit de GUI que compete com os padrões do Java (AWT/Swing/Java 2D), continua firme e forte apesar de toda a reclamação da Sun ou do grande avanço da Swing no Java SE 6 (e fora dele, com Bean Binding, Swing Framework, NetBeans Platform etc.). No release 3.3, o foco foi de suporte cada vez melhor a um número maior de plataformas. Há melhorias para todos: impressão no GTK, suporte ao System Tray do Mac OS X, cursores coloridos em ambos, e novos recursos em geral como um componente de calendário.

Não é possível ignorar, todavia, que o Windows continua sendo o sistema operacional instalado em 90% dos PCs.

E que o grande lançamento do ano foi o Windows Vista, cuja GUI estabelece o alvo que toda a indústria de software para PC irá perseguir a partir de agora. O Vista tem uma nova infra-estrutura gráfica que quebra uma tradição de 22 anos, tornando obsoleta a GDI (API gráfica fundamental do Windows – do 1.0 até o XP) e substituindo-a por algo radicalmente novo.

Aí entra a SWT 3.3. Esta nova versão do polêmico toolkit gráfico da Fundação Eclipse possui um novo porte: a SWT para WPF (WPF é o novo framework gráfico do .NET 3.0 / Windows Vista⁴). Então você, criador de aplicações ricas em Java, que já instalou o Vista e se informou sobre estas novidades já prevendo as futuras expectativas dos seus clientes, não precisa mais se desesperar e trocar o Java pelo .NET. Basta usar a SWT e sua aplicação ficará totalmente alinhada com o novo padrão de funcionalidade estabelecido pelo Windows Vista.

Ainda não se sabe se a Sun dará suporte semelhante com a Swing, mas isso é bem provável. Senão os adeptos da Swing sofreriam séria desvantagem: a SWT, JFace e Eclipse RCP se tornariam (novamente) muito superiores às APIs padrão para aplicações exigindo o máximo em termos de recursos nativos de GUI. Um porte da Swing para WPF não deve ser difícil, uma vez que a Apple já fez algo parecido na sua plataforma: no JDK da Apple (que é um porte do Sun JDK) para o Mac OS X "Tiger", a Swing foi "envenenada" com a tecnologia Quartz Extreme, que é comparável à WPF.

WPF e Java SE 6

É importante diferenciar "suporte total ao Vista e WPF" do "novo Windows look-and-feel". Este último já está disponível no Java

⁴ Embora exista um porte para Windows XP, este nunca terá o mesmo desempenho e qualidade da versão para Vista, devido à falta de elementos cruciais do Vista: DirectX 10 e os novos drivers de vídeo WDDM.

SE 6 e torna aplicações Swing praticamente indistinguíveis de aplicações nativas (mas tradicionais) no Vista. Mas não estou falando de aplicações tradicionais, e sim de uma nova geração de aplicações – que ainda nem chegaram ao mercado. Nem mesmo os produtos mais recentes da própria Microsoft, como o Office 2007, utilizam a WPF. A única exceção que conheço é o visualizador de arquivos XPS do Vista.

Creio que a primeira leva de aplicações WPF incluirá pacotes de edição gráfica, editoração, mídia, CAD e outros que se

beneficiam muito do novo framework de gráficos e GUI. Mas se o passado ensina alguma coisa, os recursos que hoje são exigidos e valorizados somente por especialistas em artes visuais (muitos dos quais estarão bocejando para o WPF porque seus Macs têm o Quartz Extreme desde 2002), em alguns anos se tornarão um requisito banal para qualquer aplicação.

Depuração

A plataforma Java SE 6 traz melhorias importantes nas capacidades de depuração

da JVM⁵, e o novo Eclipse explora estas capacidades.

Cansado de correr atrás de vazamentos de memória? Na view de variáveis do Eclipse, acione o menu de contexto sobre qualquer variável cujo tipo seja um objeto (e não um tipo primitivo como `int`). Com a opção *All References*, você visualiza todos os objetos que referenciam o objeto selecionado. E com a opção *All Instances*, verá

⁵ Ver JVMTI (Java VM Tool Interface): <http://java.sun.com/javase/6/docs/technotes/guides/jvmti/changes6.html>

Eclipse: o IDE e a tecnologia

O Eclipse já tem uma forte tradição de inovar além das fronteiras de IDEs, e para entender o Eclipse, é preciso entender isto. A SWT (e sua extensão MVC, a JFace) foi só o começo. Várias outras tecnologias e frameworks que nasceram para dar suporte ao Eclipse alçaram vôo próprio, passando a ser utilizadas por desenvolvedores de aplicações em geral, e tiveram um profundo impacto na comunidade Java. Podemos adicionar a essa lista, encabeçada pela SWT, frameworks de programação orientada por metamodelos, data binding e XML (EMF, GEF, XSD, SDO, UML2); criação de aplicações ricas (Eclipse RCP); suporte a modularização (Equinox); e inovações no processamento de código-fonte Java (Compilador JDT).

Alguns destes subprodutos do Eclipse criaram polêmica e atrito com outros fornecedores de tecnologia Java, especialmente a Sun, por não se alinharem aos padrões do Java Community Process – e às vezes concorrerem frontalmente com esses padrões (especialmente SWT versus Swing). O fato é que a IBM não agiu assim para competir com a Sun. A IBM tem recursos próprios suficientes para desenvolver qualquer tecnologia que ache necessária e tem suas próprias prioridades. Por que então a empresa deveria se contentar com as inadequações da Swing (lembre que o Eclipse 1.0 é de 2001, na era do J2SE 1.3.1), se tinha recursos para contornar o problema?

Um dos fatos pouco conhecidos da história do Eclipse (apesar do nome sugestivo) é que o alvo da IBM não era concorrer com a Sun – ou com qualquer outro vendedor de IDEs Java. Na época, o VisualAge for Java, da mesma

IBM, já era um dos líderes nesse mercado, no qual a Sun não existia. Não; o alvo da IBM era a Microsoft com o VisualStudio. Já falei isso antes (na seção “Caféina” da Edição 24), citando uma apresentação da EclipseCon 2005. Mas de lá para cá, levantei mais informações sobre essa história, que achei interessante comentar aqui, porque nos traz grande insight sobre a história do Eclipse, bem como a relação da IBM com a Sun e a comunidade Java.

A história secreta da SWT

A guerra dos toolkits de GUI portáteis – componentes nativos (AWT, SWT) versus componentes “leves” (Swing) – já existia no mundo Smalltalk, que era dividido entre três grandes fornecedores: IBM, Digitalk e ParcPlace. As VMs Smalltalk da IBM e da Digitalk usavam componentes nativos e a da ParcPlace utilizava componentes leves. Em 1995 a Digitalk e ParcPlace se fundiram na ObjectShare, que tentou criar uma nova VM Smalltalk unificando os produtos e tecnologias das empresas originais. Este projeto acabou falhando, uma das causas sendo que os engenheiros passaram mais de um ano disputando qual seria a melhor arquitetura de toolkit de GUI: componentes nativos ou leves? Enquanto perdiam tempo com isso, a IBM capturava a maior parte do já decadente mercado de Smalltalk – e a ObjectShare afundou em 1999, após um breve período tentando sobreviver com um novo IDE para Java, o PARTS (na sua época excelente; foi um dos seus usuários).

A maioria dos engenheiros pró-componentes leves da ParcPlace/ObjectShare foi trabalhar para a Sun, e lideraram o projeto Swing, adotando um

código que havia sido prototipado inicialmente pela Netscape. É por isso que a Swing faz a opção por componentes leves. Os “pais do Java” mais famosos, como James Gosling, não tiveram muito a ver com essa decisão.

Enquanto isso, a IBM aposentava a sua velha linha de IDEs VisualAge baseados em Smalltalk, em favor de uma nova linha baseada em Java. O trabalho foi feito pela mesma equipe da IBM que antes desenvolvia os produtos VisualAge. Esse pessoal era da “turma dos componentes nativos”, mas mesmo assim tentaram usar a Swing. Houve um protótipo do WebSphere Studio (jamais visto pelo público) construído com a Swing.

Este protótipo foi submetido a testes com usuários potenciais, sendo comparado com o Microsoft Visual Studio, e a resposta dos usuários foi unânime: acharam a ferramenta da IBM lenta e horrível. (Ninguém ligou para a funcionalidade: GUI ruim é visto como produto ruim – uma história comum em estudos de usabilidade.) Vamos lembrar que isso aconteceu na época do JDK 1.2, com uma Swing primitiva, que demoraria vários anos para adquirir o desempenho e os look-and-feels que a versão atual possui.

A IBM partiu então para o Plano B, criando um toolkit de GUI para Java que seria baseado em componentes nativos como a AWT, mas que ao contrário da AWT seria moderno e completo. Usaram este toolkit primeiramente no VisualAge Micro Edition (um IDE especializado em Java ME) e o resultado foi ótimo. Então decidiram usar o novo toolkit, a SWT, no novo projeto Eclipse, que seria a base da sua nova linha de IDEs. A IBM ainda planejava integrar a SWT à AWT, mas isso



todas as instâncias da mesma classe. O novo recurso é configurável em *Window>Preferences>Java>Debug>Heap Walking*. Veja a Figura 1 para uma demonstração. Além de exibir as referências num pop-up, também há uma nova opção na view de variáveis para criar um nó “referenced from” sob todas as variáveis.

Outro recurso que exige o Java SE 6 é o comando *Force return*. Ao depurar um método, em qualquer ponto pode-se forçar seu retorno imediato (sem executar o código até o final do método ou até o próximo

return). Se o método tiver valor de retorno não-void, pode-se selecionar qualquer variável ou expressão de valor compatível com o retorno, e este valor será retornado “na marra”. É um enorme facilitador, porque permite reproduzir cenários de erro facilmente. Sem este recurso, muitas vezes a alternativa mais fácil seria alterar código temporariamente de forma a induzir a aplicação ao estado no qual se suspeita comportamento incorreto.

Para quem faz profiling (análise de desempenho), outra facilidade importante

é a capacidade de fazer essa análise e a depuração ao mesmo tempo. Você pode lançar a aplicação num profiler (por exemplo, o do TPTP 4.4) e ter a capacidade de usar breakpoints e todos os recursos de depuração. A mesma melhoria possibilita a outros plug-ins contribuir novos “modos de execução” simultâneos.

Refactoring

A capacidade de manipular a estrutura do código ou de grandes projetos de forma automatizada e confiável, ou *refactoring*, é

dependia de colaboração da Sun, que teria que resolver problemas da AWT identificados pela IBM. Mas esta colaboração não aconteceu.

Então a IBM partiu para o confronto, construindo a SWT na sua forma atual, sem qualquer integração com as APIs de GUI padrão (a SWT possui até mesmo suas próprias classes de ponto e retângulo), e evoluindo-a numa competidora da Swing cada vez mais forte.

A Sun reagiu, por exemplo negando ao Eclipse a certificação Java, com o pretexto de o IDE possuir código nativo. Isso embora todos os IDEs avançados possuam algum código nativo. O NetBeans, por exemplo, possui uma biblioteca nativa para integração com o browser do sistema operacional, e outra para a interface do Profiler. Mas pode funcionar sem estas bibliotecas nativas, ainda que com redução de funcionalidade. Já o Eclipse não pode funcionar sem a SWT, o que permite à Sun discriminá-lo.

A questão da portabilidade é controversa: por um lado, a Swing funciona em todas as plataformas que possuem uma JVM Java SE, que é um número muito grande. Por outro lado, a versão mais recente e melhor da Swing é amarrada à última versão da JVM, disponível num número bem menor de plataformas, geralmente só as “primárias” da Sun (Windows, Solaris e Linux). Portes feitos por terceiros para outras plataformas sempre estão um ano ou mais defasados em relação aos releases da Sun.

Já a SWT é capaz de rodar em JVMs mais antigas, portanto, a última versão da SWT sempre suporta um número de plataformas superior ao da última versão da Swing. A SWT também tem um número de plataformas primárias maior

que o da Swing: inclui não só o Windows, Linux e Solaris, mas também o QNX, AIX e Mac OS X. Sem falar na eSWT para Java ME. Mas talvez o jogo vire com o OpenJDK (o projeto que hospeda o JDK open source da Sun).

A colaboração da comunidade open source já ajudou a SWT a ter o número de plataformas suportadas que tem hoje – o porte para Mac OS X foi inicialmente feito por um voluntário de fora, e a comunidade Linux também tem colaborado nos portes para GTK2. Se funcionou para a IBM, esperamos que funcione também para a Sun.

A posição da IBM, claramente, não é de tranqüila aceitação de todos os padrões criados pela Sun ou pelo JCP. No caso dos padrões abertos do Java Community Process isso seria uma atitude, a princípio, condenável. Mas é preciso lembrar que a AWT e Swing são APIs pré-JCP. Foram desenvolvidas unilateralmente pela Sun, num processo totalmente fechado. Portanto a IBM (ou qualquer outra empresa) não teve oportunidade de criticar a AWT/Swing, fazer sugestões, nem de corrigir o que achava errado na época em que era possível fazer isso – quando estas APIs estavam sendo projetadas.

Debates à parte, a SWT e outras tecnologias do Eclipse só existiram devido a lacunas ou deficiências das APIs padrão do Java. Isso teve o efeito de pressionar a Sun para investir mais no aperfeiçoamento destas APIs, especialmente as que habilitam o desenvolvimento de clientes ricos em geral. Podemos agradecer parcialmente ao Eclipse por coisas como:

- As grandes melhorias na Swing, no Java SE 5 e 6.
- A JSR-199 (Java Compiler API) e a JSR-269

(API para Processamento de Anotações Plugável), e o novo framework de processamento de código-fonte Java do NetBeans 6.0 que foi “envenenado” por estas APIs e outras melhorias do *javac*. Todos podem ser vistos como resposta à enorme superioridade do compilador Java do Eclipse JDT, inclusive a forma inovadora como o JDT se integra com o compilador **.

- A JSR-295 (Beans Binding), que será parte do futuro Java SE 7, para sincronizar componentes de GUI com objetos Java automaticamente, com uma linguagem de expressões (EL).

- A JSR-296 (Swing Application Framework), também parte do Java SE 7, mas com suporte ao Java SE 6, padronizando funcionalidades essenciais hoje presentes nas plataformas Rich Client (Eclipse RCP e NetBeans Platform).

- A JSR-277 (Sistema de Módulos para o Java), também parte do Java SE 7 que acrescenta ao coração do Java funcionalidades de programação e execução de módulos com facilidades de descoberta, carregamento e ciclo de vida dinâmicos (ou seja, uma arquitetura de plug-ins). É uma resposta direta ao suporte do Eclipse para OSGi; veja o **quadro** “OSGi versus JSR-277: a guerra dos plug-ins”.

* Mais precisamente, pela equipe da OTI, empresa canadense na época já incorporada à IBM, responsável por toda a sua tecnologia de Smalltalk e depois muito de Java.

** Para quem nunca usou ambientes Smalltalk, onde a VM, a linguagem, o compilador e a aplicação são praticamente uma só coisa. Como foi dessa tradição que vieram os criadores do Eclipse, é fácil ver como se inspiraram.



um dos diferenciais de IDEs modernos, e é uma área de destaque do Eclipse. A versão 3.3 traz mais algumas melhorias para os adeptos desta técnica.

Scripting mais completo

No artigo sobre Eclipse 3.2 apresentamos a capacidade do Eclipse de trabalhar com scripts de refactoring. O recurso funcionava para a maioria dos refactorings, mas não para todos. Faltavam alguns casos que são mais difíceis, porque criam, destroem ou renomeiam recursos do projeto: *Move*, *Copy*, *Paste* e *Delete*. No Eclipse 3.3, mesmo estes refactorings mais difíceis suportam a nova facilidade de scripting. Agora 100% dos refactorings podem ser usados com a tranquilidade oferecida pelo histórico de scripts.

Para completar, operações de arrastar-e-soltar do Project Explorer também geram refactorings com os respectivos scripts. Por exemplo, arrastar uma classe de um package para outro, seja do mesmo projeto ou entre projetos distintos, irá gerar um refactoring *Move*.

Agrupamento de parâmetros

Temos um novo refactoring importante: *Introduce Parameter Object*. Se você tem métodos com vinte e sete parâmetros que deixam seus relatórios de qualidade de código horríveis, seus problemas acabaram! Este refactoring permite selecionar um grupo de parâmetros que você considere fortemente relacionados, e agrupá-los numa nova classe. Todas as invocações ao método antigo serão alteradas para criar uma instância desta nova classe.

Refactorings mais fáceis

Mas a novidade que será a favorita de muita gente é a nova facilidade de refactoring *inline*. Para começar, se você quiser renomear alguma coisa (ex.: uma variável ou método), coloque o cursor sobre o identificador a ser renomeado e pressione *Alt+Shift+R*. O identificador é destacado por uma moldura. Daí basta modificá-lo no próprio editor, sem precisar de nenhum diálogo de refactoring.

E o preview? À medida que você edita, todas as referências para o identificador renomeado que existam no mesmo arquivo serão mudadas simultaneamente. A

Figura 2 demonstra esta facilidade.

Vários Quick Fixes e Quick Assists⁶ apresentam uma melhoria semelhante. Por exemplo, com o cursor sobre a declaração de uma inner class, clique *Ctrl+2* e selecione o quick fix "Convert anonymous to nested class", cujo efeito é o mesmo de *Refactor>Convert Anonymous Class to Nested*. A nova classe é criada no editor,

⁶ Revisões automáticas de código, acionadas por *Ctrl+1* e *Ctrl+2* respectivamente.

e os elementos configuráveis do código gerado, como o nome e a visibilidade da nova classe, são destacados por uma moldura. Estes elementos podem ser alterados da mesma forma que no caso anterior (usando a tecla *Tab* para alternar entre estes elementos).

O mesmo comportamento é suportado também pelos seguintes quick assists, que são novos no Eclipse 3.3 (e também facilitam o acesso a refactorings de funcionalidade similar): *Extract to local variable*,

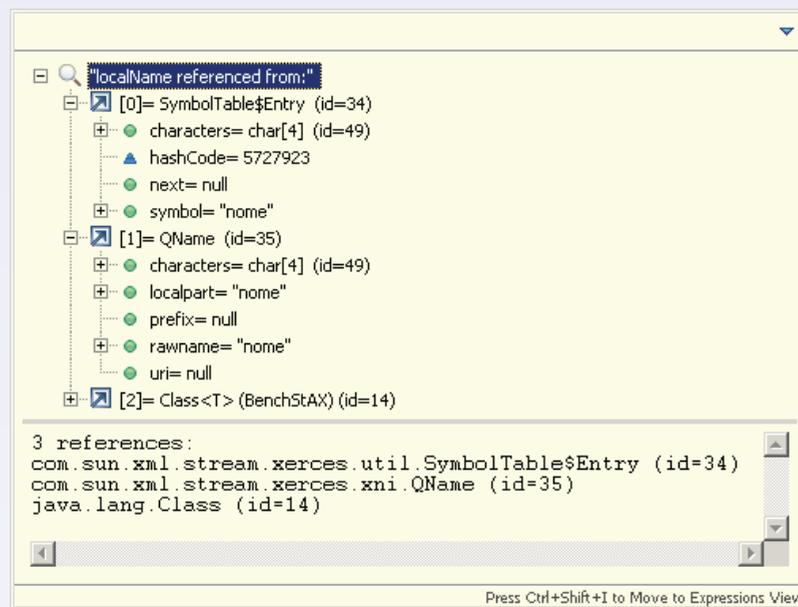


Figura 1. Examinando todas as referências para um objeto (variável local **String localName** do método onde esta funcionalidade foi acionada). Esta string foi gerada por um parser XML (ainda em execução), portanto é referenciada por objetos do parser, como **SymbolTable** e **QName**.

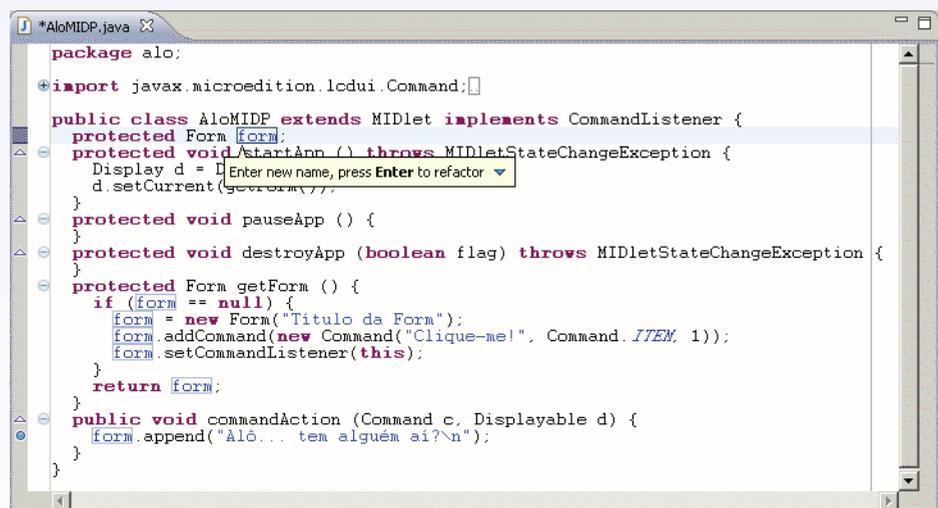


Figura 2. Renomeando inline uma variável "form". Todas as referências a esta variável são destacados por uma moldura e todas são alteradas simultaneamente quando for editada qualquer uma destas referências.

Extract to constant, Inline local variable, Convert local variable to field, Convert anonymous to local type.

Incomodado com a exigência que os refactorings tinham de forçar todos os editores a estarem sincronizados com o disco, gravando as alterações? Esta exigência não existe mais. Os refactorings funcionam mesmo com editores “sujos”, sem gravar as alterações nem antes nem depois.

Todas estas melhorias otimizam o trabalho do desenvolvedor que gosta de usar refactorings. Não é mais preciso “sair” do editor, abrindo menus, caixas de diálogo ou de confirmação que quebram o seu fluxo normal de atenção ao digitar código. Diversos refactorings comuns ficam acessíveis por uma combinação de teclas, e seus parâmetros podem ser modificados no próprio editor de código.

Controle de versões

O Eclipse 3.3 traz melhorias no suporte ao CVS, mas este suporte já é muito maduro e as novidades são poucas e bem incrementais (são do tipo que prometemos não gastar espaço explicando neste artigo: um novo campo de busca na view de histórico etc.) A janela de commit usa o corretor ortográfico do Eclipse, de forma que não mais ficaremos constrangidos por um escorregão como “conçertei o bug 94”, já que os repositórios não permitem alterar revisões já confirmadas. (Veremos mais sobre o corretor ortográfico adiante.)

Mas a melhor novidade do Eclipse 3.3 relacionada ao controle de versões é que o CVS *não* é mais embutido na Eclipse Platform! Nas páginas de download que oferecem arquivos individuais do projeto Eclipse, como Platform, JDT e PDE, você encontrará um novo item, *CVS Client Runtime Binary*, que deve ser instalado à parte se você ainda quiser usar o CVS. Mas quem quer fazer isso? Eu, pelo menos, acho o Subversion (SVN) tão superior ao CVS que até arrisquei converter todo o repositório CVS da minha empresa, com projetos desde 2002, de CVS para SVN. Mas antes de repetir esta experiência correndo, veja o quadro “cvs2svn: vale a pena, mas tenha

cuidado”.

E suporte para o Subversion nós temos no 3.3. O plug-in Subversive, criado pela Polarion, foi proposto e aceito pela Fundação Eclipse como plug-in oficial de Subversion para o Eclipse. No momento em que escrevo, o Subversive ainda é encontrado na “incubadora” da Fundação, oferecendo duas versões: 1.0.x para o Eclipse 3.0 ou 3.1, e 1.1.x para o Eclipse 3.2 ou 3.3. A versão 1.1.x terá um release de produção coincidindo com o Europa. Este plug-in também não será embutido em nenhum dos downloads primários do IDE⁷, mas estará listado nas páginas e site de update do novo Eclipse, e embutido em “distros” integradas feitas por terceiros. A **Figura 3** mostra a configuração de um repositório SVN.

Há alguns aperfeiçoamentos que beneficiam qualquer sistema de versionamento.

7 Mas isso não implica num status inferior ao do plug-in para CVS. É que o suporte para Subversion exige bibliotecas como a JavaHL ou SVNKit, que embora sejam open source, possuem licenças não compatíveis com a EPL do Eclipse.

Em especial, o recurso de comparação (diff) é bem mais rápido. Além disso, ao fazer merges, o IDE irá destacar alterações individuais: caracteres ou palavras isoladas numa linha de texto que não mudou em sua maior parte.

Desenvolvimento Java

Aqui vamos tratar de aperfeiçoamentos genéricos, não discutidos nas seções anteriores, de Depuração ou Refactoring, mas que são específicos ao trabalho com a linguagem Java. Um dos pontos mais fortes do Eclipse sempre foi sua capacidade de compreender e manipular código Java, seja no editor, compilador ou em outras ferramentas. Isso continua avançando.

Tolerância a erros

Você perceberá que o editor de código melhorou (ainda mais) a capacidade de lidar com erros. Por exemplo, numa classe com variáveis locais ou parâmetros de blocos **catch** duplicados, o editor sinalizará

cvs2svn: vale a pena, mas tenha cuidado

Fiz a migração dos meus repositórios CVS para SVN (Subversion) com um script de conversão automática, o cvs2svn, que demorou várias horas para executar e gerou alguns erros que tive que corrigir à mão. Mesmo assim, migrei para o Subversion e nunca mais olhei para trás. Mas é importante observar que só fiz isso após meses de preparação e testes. Em especial, após esperar que tanto este script de conversão quanto os plug-ins de suporte a Subversion para Eclipse atingissem um grau de funcionalidade e qualidade adequados. Para quem estiver considerando agora uma migração semelhante, pelo menos estas pré-condições já estão satisfeitas. Mas há outros pontos a considerar nesta migração (ou em qualquer tipo de migração entre sistemas de controle de versões):

1. Comece usando o Subversion em paralelo, com um repositório pequeno para projetos de prospecção e outros menos críticos. Vá aprendendo a configurar e administrar o servidor SVN – que tipo de repositório você prefere (fsfs

ou bdb), que tipo de acesso por rede (“svn:” ou HTTP via Apache), segurança, backup, etc.

2. Faça uma migração de teste do seu maior repositório CVS, criando o novo repositório SVN somente para validá-lo, mas continuando a usar o antigo. Você verá que no novo repositório os branches e tags são organizados de maneira pouco natural para o SVN, e talvez causando confusão se o repositório servir a muitos projetos com nomes de tag/branch coincidentes. (Isso é uma limitação do script de conversão, não do Subversion, mas pode ser contornada/corrigida de várias formas.) Talvez a conversão falhe para alguns recursos versionados. Isso é geralmente culpa do CVS, sendo possível corrigir o repositório do CVS “na unha” e refazer a conversão.

3. Depois de dominar o processo todo, faça a conversão para valer. Ou se você tiver identificado muitos riscos, adote o Subversion apenas para novos projetos. Declare que CVS é legado e proíba a criação de projetos novos em repositórios CVS.

estes erros, mas mesmo assim todos os recursos do editor e do JDT continuarão funcionando. Vamos supor que você tem, por exemplo, dois loops `for` aninhados com mesma variável de controle:

```
for (int i = ...) {
    x += i; // Usa o 'i' externo
    for (int i = ...) {
        ...
        a -= i; // Usa o 'i' interno
    }
    y *= i; // Usa o 'i' externo
}
```

Neste caso você poderá usar o refactoring de renomeação de variável para corrigir um dos `i` para outro nome. O Eclipse não vai se confundir: por exemplo, ao renomear o `i` externo, o IDE não modificará os usos de `i` no loop interno e vice-versa. Esse tipo de erro não é tão básico quanto parece. É comum, por exemplo, ao copiar e colar código para dentro de um método que, por coincidência, tinha alguma variável ou parâmetro com nome igual ao de alguma variável do código colado.

O completamento de código foi melhorado em vários pontos, inclusive para código com erros. Se você declarar uma variável `List lista`, sendo que o tipo `List` é desconhecido do compilador (pois faltou o seu `import`) e tentar o completamento após "`lista.`", funcionará da mesma maneira. O editor procura todos os tipos com nome simples `List` visíveis no seu classpath, como `java.util.List` e `java.awt.List`, e oferece os métodos e atributos de todos eles. Se você aceitar uma das sugestões de completamento,

OSGi versus JSR-277: a guerra dos plug-ins

Quem acompanha o projeto lembrará que todas as funcionalidades do Eclipse – até as mais elementares – são implementadas por plug-ins. No coração do Eclipse há apenas um "microkernel" que tem a função de configurar, gerenciar, carregar e executar plug-ins. Este microkernel não é uma invenção da Fundação Eclipse. É uma implementação do padrão OSGi (Open Services Gateway Initiative), descrito como "um ambiente orientado a serviço e baseado em componentes para gerenciar o ciclo de vida de softwares". Traduzindo: um sistema de módulos.

O que isto proporciona é a capacidade de construir aplicações complexas a partir de componentes fracamente acoplados e ativados de forma dinâmica. Explicando de outra forma, o que os web services fazem por aplicações distintas conectadas pela rede, um sistema de módulos faz por componentes individuais agregados no mesmo processo.

Outra boa comparação é com a plataforma Java EE, que nos permite instalar e atualizar módulos individuais (EARs/WARs) à quente, ou instalar no mesmo servidor várias versões diferentes da mesma aplicação (que só não podem entrar em conflito em pontos de contato com o mundo externo, como nomes JNDI ou contextos web). O projeto no qual trabalho hoje possui cerca de 30 módulos independentes (são tantos EARs – e tantas *resource references* JNDI – que uma reinstalação total, se não for automatizada por scripts, leva uma hora).

As vantagens de arquiteturas modulares são conhecidas. Por exemplo, se for descoberto um bug, é possível corrigir e reinstalar somente o módulo afetado pelo bug, sem indisponibilidade de

nenhum serviço prestado pelos demais EARs.

O problema é que servidores Java EE são grandes, pesados, complexos. E o modelo de programação Java EE não serve para muitos tipos de aplicações, por exemplo clientes ricos feitos com Swing ou SWT. Precisamos então de uma tecnologia que ofereça somente aquelas facilidades de modularização dos containers, mas sem nos "empurrar" coisas como EJB, JTA, Servlets, conectores, web services e o resto do Java EE.

É aí que entra o OSGi. Mas o OSGi vai além. É uma tecnologia de uso geral, não limitada aos cenários mais estreitos do ciclo de vida de módulos de qualquer container de aplicações específico (como Java EE ou Spring). Por exemplo, um container Java EE não possui a capacidade de disponibilizar várias versões de uma biblioteca compartilhada. Se você quiser que três EARs instalados num servidor usem a biblioteca `Log4j 1.2.14`, e outros dois EARs mais antigos, executando no mesmo processo, continuem usando a `Log4j 1.2.6`, o único jeito é embutir o `log4j.jar` correspondente em cada EAR. Não há como ter duas versões incompatíveis de uma biblioteca instalada globalmente e possibilitando que cada módulo (EAR) decida qual versão prefere usar. Já o OSGi permite fazer isso.

O OSGi não é usado somente pelo Eclipse (cuja implementação deste padrão chama-se Equinox, e no Eclipse 3.3 foi atualizada para o novo OSGi 4.1). A lista de softwares Java estruturados como coleções de *bundles* (plug-ins) para OSGi é cada vez maior. Entre servidores de aplicações, por exemplo, é uma febre: inclui as últimas versões do IBM WebSphere, BEA WebLogic, JBoss, Spring Framework e provavelmente outros.

Ainda mais interessante, o padrão OSGi foi

recentemente referendado pela JSR-291: "Suporte para Componentes Dinâmicos para o Java SE". Esta JSR foi aprovada com voto contrário da Sun. O OSGi é agora um padrão JCP. Idem para a JSR-232: Mobile Operational Management, que é específica à plataforma Java ME, e também adota o padrão OSGi na sua variante para plataformas limitadas.

Enquanto isso, a Sun lidera o esforço da JSR-277, "Sistema de Módulos para Java", a ser incorporada ao Java SE 7. A JSR-277 tem sido criticada, pois a funcionalidade proposta é bem inferior à do OSGi. É um passo para trás, tanto tecnicamente quanto em termos de padronização, pois teremos dois padrões competidores quando já estávamos muito bem servidos pelo padrão de fato, o OSGi.

A Sun afirma que esta funcionalidade será beneficiada por novas sintaxes da linguagem (ex.: os *superpackages*), além de otimizações que serão possíveis ao implementar o recurso no "coração" da JVM. Isso também é verdade e a minha esperança é que a JSR-277 (em andamento) produza uma especificação que possa conviver bem com a OSGi; por exemplo, para permitir uma implementação de OSGi para Java SE 7 que funcione como uma camada sobre a JSR-277, usando e estendendo funcionalidades básicas da última.

Outra desvantagem da JSR-277 é que esta não servirá ao Java ME. Seria muito melhor ter todas as plataformas Java unificadas em torno de um padrão único, e o OSGi já é este padrão; já suporta todas as edições do Java, do ME ao EE, e é uma tecnologia madura e comprovada. O cenário ainda é um pouco confuso; precisamos esperar mais um tempo para ver qual será a estratégia da Sun.

o editor também faz o **import** que faltava. Quer mais facilidade que isso?

Completamento

Também há melhorias de completamento para a sintaxe do Java 5. Para quem é fã de tipos genéricos, mas às vezes se atrapalha com os casos mais complexos, as mensagens de erro para usos incorretos de captura de coringas (ex.: `List<? extends X>`) são bem mais claras.

O completamento para anotações também melhorou. O Eclipse 3.2 já oferecia completamento de nomes de anotações após digitar '@' seguido de pelo menos um caractere. No Eclipse 3.3, pode-se completar o nome após digitar somente o '@'. E o que é mais importante: uma vez dentro dos parênteses da anotação, agora há completamento dos seus parâmetros.

Temos também o completamento de *static imports*. Ao utilizar APIs que disponibilizam atributos ou métodos **static**, como **System**, **Arrays** ou **Collections**, o Java 5 permite usar uma declaração como `import static java.lang.System.*`; com isso é possível escrever somente `out.println()` ao invés de `System.out.println()`. No novo Eclipse, membros **static** importados dessa forma, como `out` ou `arraycopy()` de **System**, serão oferecidos pelo recurso de auto-completamento.

E uma melhoria vale para qualquer versão do Java: ao escrever cláusulas **catch**, o autocomplete do Eclipse prioriza as exceções que são lançadas dentro do **try** mas que ainda não foram capturadas por blocos **catch** anteriores.

Clean-Up

O assistente de "limpeza" de código (*Source>Clean-Up*), lançado no Eclipse 3.2, ganhou algumas novas opções no 3.3, como formatação de código e de comentários. Também está melhor integrado ao sistema de preferências. Pode-se definir perfis de opções de clean-up, associados ao workspace ou a projetos individuais.

A **Figura 4** mostra o diálogo de propriedades de um projeto do Eclipse 3.3, destacando as preferências de clean-up. É uma grande facilidade para projetos em equipe, pois não é necessário que todos fiquem sincronizando configurações dos seus workspaces particulares. Basta configurar

as opções no projeto, e tudo é sincronizado através de um repositório de versões.

E para você não esquecer mais de rodar o Clean-Up Wizard, é possível agora configurar o Eclipse para fazer isso automaticamente toda vez que um arquivo *.java* for salvo.

Visualização de classes sem fontes

Ao abrir um arquivo *.class* para o qual não há código fonte disponível, por exemplo devido a um *Step Into* no depurador, o Eclipse apresenta o bytecode "desassembled". Esta listagem, que é semelhante à saída gerada por `"javap -c NomeDaClasse"`, exige conhecimento do bytecode Java e é bem mais difícil de ler do que fontes Java. Mas é melhor do que não ter nenhum tipo de visualização de classes sem fontes. Veja um exemplo na **Figura 5**.

☞ *Uma questão interessante: o uso deste recurso viola as licenças de uso de bibliotecas de código fechado, que proibam descompilação? A princípio, não: a listagem de bytecodes não é descompilação – não produz uma listagem na sintaxe Java (ou qualquer que seja a linguagem que gerou o bytecode: poderia ser JRuby, Groovy...), ou em qualquer sintaxe de alto nível e facilmente legível. Portanto, não viola nenhuma licença que proíba especificamente "engenharia reversa" ou "obtenção de fontes". Há alguns produtos com licenças mais linha-dura, que proíbem até mesmo um dump de bytecodes ou qualquer depuração interna do código. Estas licenças proíbem a visualização do código em qualquer forma. Mas você já não nem o direito de fazer um "step into" em métodos de bibliotecas protegidas por tais licenças (que eu considero ridículas mesmo para software de código fechado), então o novo recurso do Eclipse*

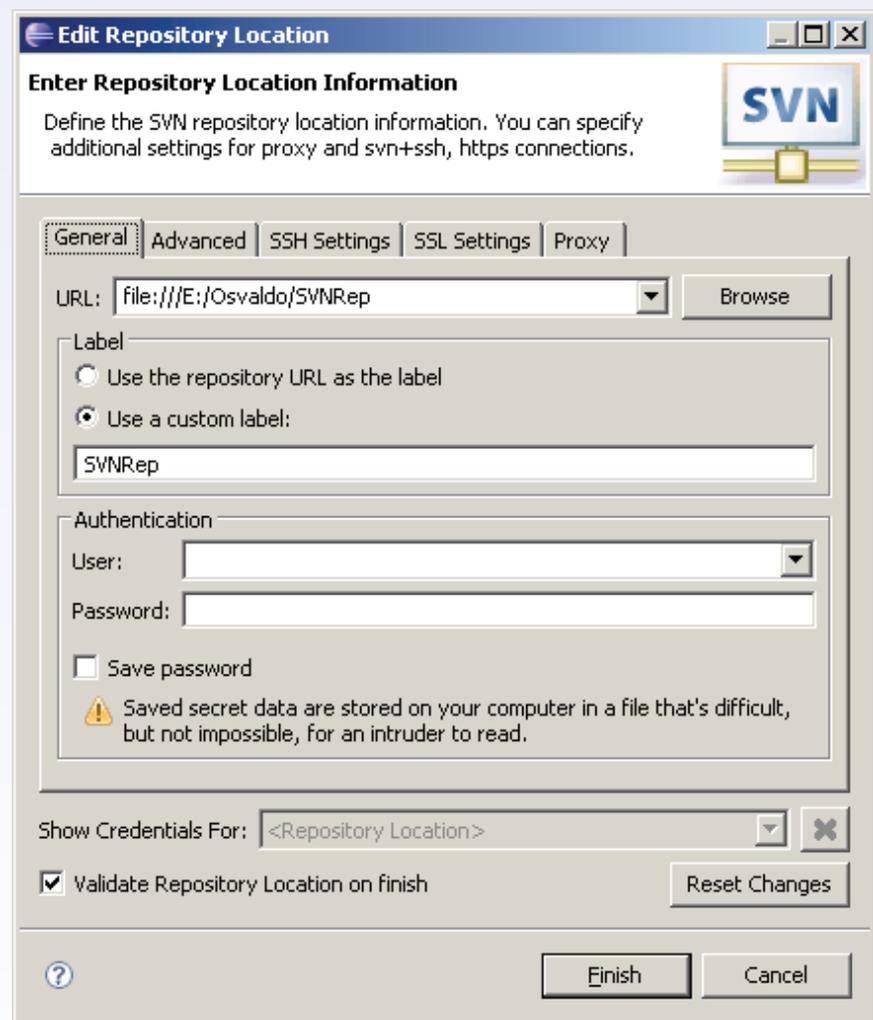


Figura 3. Diálogo de propriedades de uma conexão SVN com o Subversive.

3.3 não muda nada nesse aspecto. Além disso, bibliotecas super-protegidas costumam também ser ofuscadas, o que torna a leitura direta dos seus bytecodes extremamente difícil.

Validações do compilador

O grande número de validações de código feitas pelo compilador do Eclipse sempre foi um dos pontos fortes do IDE, pois não precisamos rodar ferramentas de detecção de bugs como PMD ou FindBugs para detectar um bom número de problemas. No Eclipse 3.3, a detecção de bugs envolvendo valores nulos foi muito melhorada, sendo dividida em três validações independentes:

- **Null reference** – Detecta código que, se executado, certamente gerará uma **NullPointerException**. Esta opção é ativada como *Warning* por default. Mas recomendo alterar o default para *Error*.

- **Potential null reference** – Detecta código que provavelmente possui o mesmo bug, mas talvez não ocorra nunca; por exemplo porque a **NullPointerException** só acontece se determinada condição for verdadeira – mas se por azar esta condição sempre for falsa nos seus testes, o bug passará despercebido. O default é *Ignore*, mas recomendo mudar para *Warning*.

- **Redundant null check** – Detecta código

que verifica desnecessariamente se uma referência é não-nula antes de utilizá-la. Essa validação pode revelar algum engano do programador. Se ele/ela não percebeu que a variável nunca será nula em determinado ponto, basta eliminar a verificação. O default é *Ignore* e também recomendo mudar para *Warning*.

Geral

Listamos aqui recursos da Plataforma Eclipse, que beneficiam qualquer IDE baseado no Eclipse.

- Há vários refinamentos nos Working Sets, que permitem reduzir a bagunça de workspaces complexos e na alternância entre workspaces (por exemplo, pode-se mudar para outro workspace carregando junto o layout do Workbench).

- Temos agora templates mais poderosos, para quem gosta de automatizar a geração de código.

- O suporte ao Ant foi atualizado para o Ant 1.7.0.

- O Eclipse 3.3 inclui dois dicionários para Inglês (americano e britânico), e o corretor pode ser usado em qualquer editor e também em lugares como o diálogo de commit. No editor Java, o corretor só age sobre regiões de texto livre, no caso comentários. Outros editores também estão sendo atualizados para ter a inteligência de fazer correção ortográfica só onde isso faz sentido.

Sobre o último item, já vi gente comentando que “agora o Eclipse tem corretor ortográfico”, mas ele já tinha, desde o 3.0. A questão é que como o IDE não vinha com nenhum dicionário incluído, nem mesmo para inglês, o corretor era desabilitado e passava despercebido.

Infelizmente, o formato de dicionário usado pelo Eclipse – um simples arquivo texto com uma palavra por linha – não é o mesmo dos corretores livres *ispell* e *MySpell*, para os quais há dicionários livres para muitas línguas incluindo a nossa. Por exemplo, tanto o *BrOffice.org* quanto o *Thunderbird* e o *Firefox* utilizam dicionários neste formato. Mas imagino que não seja difícil convertê-los para o formato do Eclipse, e que logo alguém disponibilize os arquivos para *pt_BR*.

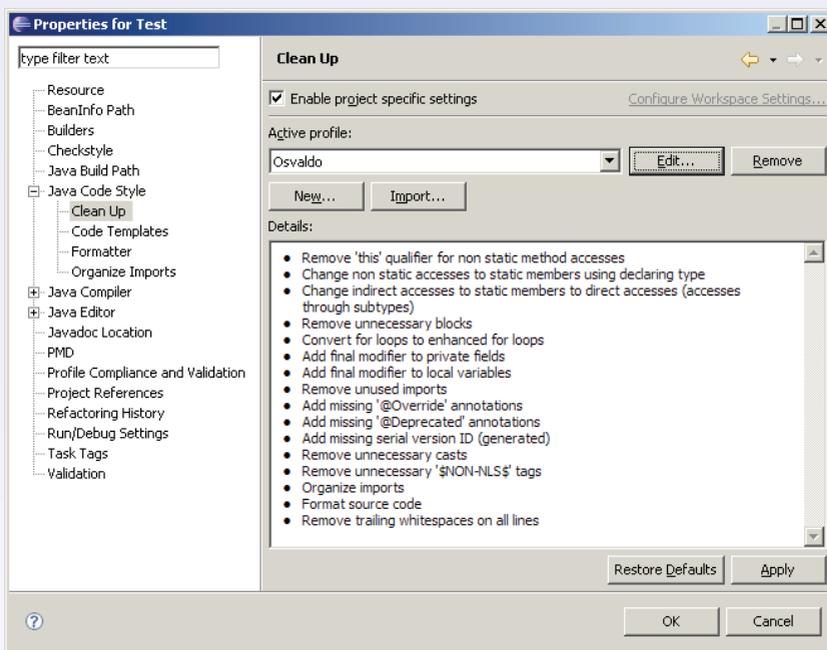


Figura 4. Configurações de clean-up específicas a um projeto.

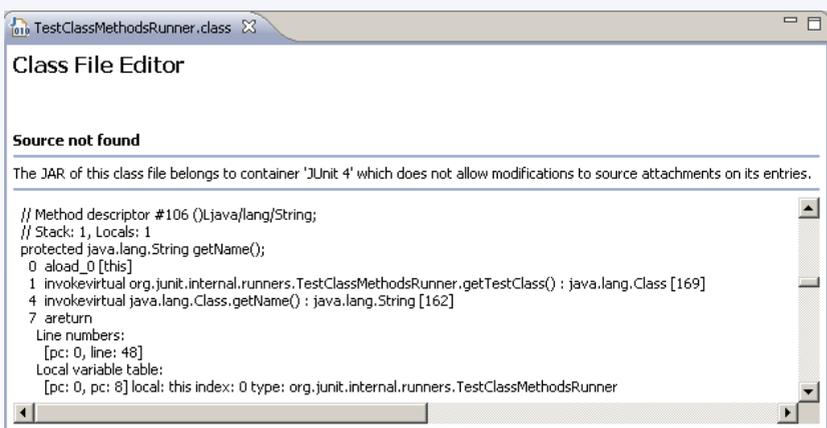


Figura 5. Visualizando uma classe da qual não temos os fontes.

Uma olhada rápida no “resto” do Europa

Neste artigo focamos no núcleo do Eclipse (Platform e JDT), mas junto com ele, o projeto coordenado Europa resultará no release simultâneo de vários outros projetos:

- **AJDT:** Suporte a Aspect Oriented Programming com a linguagem AspectJ.
- **BIRT:** Gerador de relatórios.
- **CDT:** Desenvolvimento de aplicações nativas em C/C++.
- **PTTP:** Teste, análise de logs e profiling de desempenho.
- **WTP:** Suporte a desenvolvimento Java EE/J2EE.
- **DLTK:** Suporte a linguagens dinâmicas, incluindo Ruby, Tcl, Groovy, PHP, e JavaScript.
- **DTP:** Cliente de SGBDs.
- **Corona, ECF:** Ferramentas de colaboração e comunicação, inclusive peer-to-peer.
- **EMF, EMFT, JET, GEF, GMF, EODM, OCL, UML2, MDT:** Frameworks de programação orientada a modelos, geração de código e afins; utilizados por diversos plug-ins do Eclipse.
- **Eclipse Monkey:** Automação do Eclipse com JavaScript.
- **DSDP:** Suporte a desenvolvimento para dispositivos, incluindo Java ME (subprojeto MTJ).
- **Buckminster:** Ferramentas para “build, assemble & deploy” (BA&D); algo como o Maven.
- **BPEL, BPMN, SOA Tools Platform:** Suporte à Service Oriented Architecture.
- **Mylar:** Facilidades de GUI orientada a tarefas. Integra o Eclipse com sistemas de bug tracking, e “otimiza” automaticamente a GUI do Eclipse para focar nas tarefas sendo executadas.

A lista é impressionante; maior que a do Callisto. Alguns destes itens possuem atualizações importantes para o Europa, entre eles:

- O WTP 2.0 tem grandes avanços no suporte a JSF, melhorias de desempenho importantes (por exemplo, publish muito mais rápido), e suporte total ao Java EE 5 incluindo JPA/EJB 3 completos, com validação avançada de anotações.
- O CDT 4.0 é a maior atualização deste IDE C/C++ desde o 1.0. O indexador foi aperfeiçoado (ex.: suporta templates), beneficiando vários recursos (auto-com-

pletamento, novos browsers de includes, hierarquia etc.). Um novo sistema de build (opcional) dispensa Makefiles e traz o C/C++ mais próximo da facilidade de configuração e build de projetos do JDT. No Windows, há suporte melhorado para compiladores MinGW e preliminar para o Visual C++ e Windows SDK.

- Os projetos de suporte a Java ME, inclusive o MTJ (parte do DSDP) para Java ME, preenchem uma lacuna histórica – e enorme – do Eclipse.

- Há vários plug-ins para tecnologias “quentes”, como P2P, linguagens dinâmicas, ou SOA⁸.

O abandono do VE

No meio de tantas novidades e boas notícias, contudo, há uma má notícia. O projeto VE (Visual Editor) está fora da lista. Pior: está parado. Até o momento em que escrevo (logo após o Eclipse 3.3 RC1), não há sequer um *nightly* do VE para o Eclipse 3.3. Acontece que os desenvolvedores do VE (todos funcionários da IBM) foram realocados para outras coisas. E isso não é de agora; há quase dois anos não há quase nenhum desenvolvimento no editor visual de interfaces do Eclipse.

A versão 1.2, que acompanhou o Callisto, já era apenas uma atualização mínima de compatibilidade com o Eclipse 3.2. Para o Europa, nem isso; há meio ano não há ninguém trabalhando no projeto VE. Essa situação revela a fragilidade de projetos open source muito dependentes do patrocínio de grandes corporações.

Seria possível uma nova equipe de voluntários, ou mesmo de terceiros que constroem produtos comerciais baseados no Eclipse, organizar-se para “pegar o bastão”; o ex-líder do projeto VE declarou que não custaria mais que uma semana de trabalho (pelo menos para a equipe original) para, pelo menos, atualizar o VE para compatibilidade com o Eclipse 3.3 (a versão mais recente para o 3.2 pode funcionar, mas sem garantias). Até agora isso não aconteceu, mas acredito que o problema será resolvido no final, pois estes terceiros terão sérios problemas com seus clientes se o Eclipse 3.3 não tiver o

⁸ O AJAX não está faltando; há o ATF (AJAX Toolkit Framework), mas este não está incluído no Europa.

Visual Editor. E enquanto não acontecer, ficaremos simplesmente sem nenhum suporte a desenvolvimento visual de GUIs no Eclipse 3.3 (isso enquanto o NetBeans 6 avança rápido nesta área, com suporte ao Swing Application Framework e ao Beans Binding, entre várias outras melhorias).

Conclusões

Neste artigo você o Eclipse 3.3 introduz um lote considerável de novas funcionalidades. E isso porque aqui focamos apenas no essencial; ao usar o novo Eclipse você descobrirá muitas melhorias mais sutis. Resta saber se há alguma dificuldade na migração. A boa notícia é que, devido ao Eclipse já ser tão maduro, as atualizações “traumáticas” são cada vez mais raras. Em artigos sobre versões anteriores, como a 3.2, já elogiei a ótima compatibilidade com plug-ins preexistentes. Praticamente todos os plug-ins de terceiros feitos para Eclipse 3.2, e mesmo a grande maioria compatível, como 3.0 ou superior, continua funcionando no 3.3. Há vários plug-ins já “otimizados” para o 3.3, mas eles apenas incluem aperfeiçoamentos para explorar novas facilidades do último Eclipse; não são updates obrigatórios de compatibilidade.

O Eclipse 3.3 vai além: a compatibilidade de workspaces é excelente; se você carregar um workspace criado pelo Eclipse 3.2 com o 3.3, o novo Eclipse *não* fará nenhuma alteração no workspace que cause problemas se você tiver que voltar ao 3.2. Isso é importante para projetos em equipe onde nem todos os desenvolvedores atualizam seu IDE ao mesmo tempo. É ótimo poder compartilhar os arquivos de metadados de projeto (tais como “.classpath”, que são normalmente ocultos). Pelo menos, não enquanto você não usar recursos exclusivos da nova versão, que é o cenário de convivência que eu testei, e o mais comum. Essa atualização tranquila deverá significar uma rápida popularização do “Eclipse 2007”. ●



Osvaldo Pinali Doederlein

(opinali@gmail.com)

é Mestre em Engenharia de Software Orientado a Objetos, membro individual do Java Community

Process e trabalha na Visionnaire Informática como arquiteto e desenvolvedor.



Testes: Ferramentas e Muito além do JUnit

A comunidade Java incorporou em larga escala a filosofia do **Test-Driven Development (TDD)**, o desenvolvimento orientado a testes. Prova disso é a inclusão do suporte a JUnit (descendente direto do XUnit, criado junto com o próprio XP) em 100% dos IDEs Java no mercado; e a profusão de frameworks de testes alternativos ou complementares ao JUnit como TestNG, DBUnit, XMLUnit, Cactus, HttpUnit, entre muitos outros.

Os benefícios do TDD são vários, não só no aspecto da qualidade de código mas especialmente em relação à gerenciabilidade do projeto e à manutenibilidade do produto final. Entretanto, é fácil se perder dentro da diversidade de frameworks e ferramentas de apoio ao processo de testes. Também é comum se iludir, acreditando que a bateria de testes construída ao longo do desenvolvimento da aplicação irá garantir uma aplicação em produção sem incidentes.

Neste artigo apresentamos os principais tipos de testes que podem ser desenvolvidos ao longo de um projeto Java e quais ferramentas livres podem ser utilizadas

para facilitar a construção destes testes. Vemos também como inserir os testes dentro de um processo de desenvolvimento de modo a maximizar os resultados, e evitar surpresas desagradáveis na passagem para a produção. Apresentamos ainda algumas melhores práticas para se avaliar a qualidade dos próprios testes e assim evitar que se tornem um peso morto no projeto.

Testes no ciclo de vida do software

Testes formais podem e devem ser definidos em todas as etapas do desenvolvimento de um software. Mesmo no caso de não poderem ser automatizados, defini-los precisamente permite identificar desde cedo problemas mais sérios com os requisitos ou a modelagem de uma aplicação. Podemos dizer que, quando faltam informações para a definição formal de um teste em termos de entradas exatas e de resultados esperados, isso significa que os requisitos ainda não foram suficientemente entendidos e documentados.

Passado o levantamento de requisitos e a modelagem de negócios de uma aplicação,

os testes passam a se focar mais em aspectos particulares da implementação, o que aumenta a demanda pela sua automação. Neste momento, a falta de informações suficientes para se implementar um teste automatizado pode indicar que as especificações de classes, pacotes, componentes, formatos de dados etc. ainda estão incompletas, ou então que surgiram requisitos novos ainda não completamente documentados.

O desenvolvimento orientado a testes traz



Boas práticas

Conheça ferramentas e estratégias para maximizar os benefícios do Test-Driven Development em seus projetos de desenvolvimento Java

FERNANDO LOZANO

o benefício de validar de modo objetivo os requisitos documentados desde o início do ciclo de vida. O TDD expõe erros, omissões e inconsistências que de outra forma só seriam detectados quando já houvesse código executável entregue ao usuário.

A **Figura 1** ilustra um exemplo de processo de desenvolvimento e os tipos de testes que são produzidos em cada etapa. Vale a pena ressaltar aqui que em muitos casos as mesmas tecnologias e ferramentas podem ser utilizadas para construir diferentes tipos de testes. A distinção entre, por exemplo, “testes de unidade” e “testes de sistema” é feita em relação à finalidade de cada teste, e não em relação à sua forma.

É importante observar que na etapa de manutenção temos, na verdade, um novo ciclo de desenvolvimento, com novos requisitos, nova modelagem etc. – e todos os tipos de testes são produzidos novamente, sendo agregados aos testes já existentes. Mas na figura destacamos um novo tipo de teste que surge nesta etapa, o teste de regressão.

Embora a figura sugira, à primeira vista, um processo seqüencial, ela se aplica igualmente a processos iterativos como XP. Nestes processos, cada iteração passa por todas as etapas, desde o detalhamento de requisitos até a homologação pelo usuário, e cria e utiliza todos os tipos de testes.

A efetividade dos testes é diminuída quando eles são executados em uma etapa isolada, que seria uma etapa de “QA” (*Quality Assurance* ou controle de qualidade) realizada no final do processo, ou no final de cada grande fase. Esse adiamento dos testes faz com que o feedback demore mais a chegar, e pode até mesmo levar a se apontarem culpados em vez de se identificarem os reais problemas (afinal, nenhum profissional gosta de dar uma tarefa por completada, para depois alguém vir afirmar que seu trabalho está com problemas).

Integração Contínua

Equipes mais maduras na adoção de processos de software terão incorporado, além do TDD, a prática da Integração Contínua (*Continuous Integration*) ou IC. Esta prática consiste em se ter um ambiente automatizado, que várias vezes ao dia baixa o código mais recente no repositório de código (ou no sistema de controle de versões, como o CVS ou o Subversion¹) e executa todos os testes já definidos. O resultado final é um relatório geralmente publicado em uma intranet, que fornece estatísticas gerais dos testes (quantos passaram, quantos falharam), com quebras por projeto, módulo, subsistema, pacote e classe Java, e detalhes sobre os testes que falharam.

O uso da IC permite tanto os desenvolvedores como o gerente do projeto terem

feedback freqüente sobre o impacto de mudanças recentes no código, em relação ao progresso (que pode ser realimentado para uma ferramenta de gerência de projetos) e também em relação a incidentes. Os desenvolvedores podem resolver problemas, como novos bugs ou regressões, enquanto as causas ainda estão “frescas” em suas cabeças. E efeitos colaterais gerados pelo novo código (como falhas em outros subsistemas teoricamente não-relacionados) são detectados imediatamente, em vez de tardiamente pelo usuário final ou pela equipe de QA.

Daqui em diante serão apresentados mais detalhes sobre cada etapa do processo genérico, focando nos tipos de testes relevantes em cada uma. Dentro de cada seção serão vistas também sugestões de ferramentas e frameworks Java para automatizar e facilitar a criação, a execução e o acompanhamento dos testes.

¹ Alternativas proprietárias seriam o ClearCase da Rational / IBM e o Visual SourceSafe da Microsoft

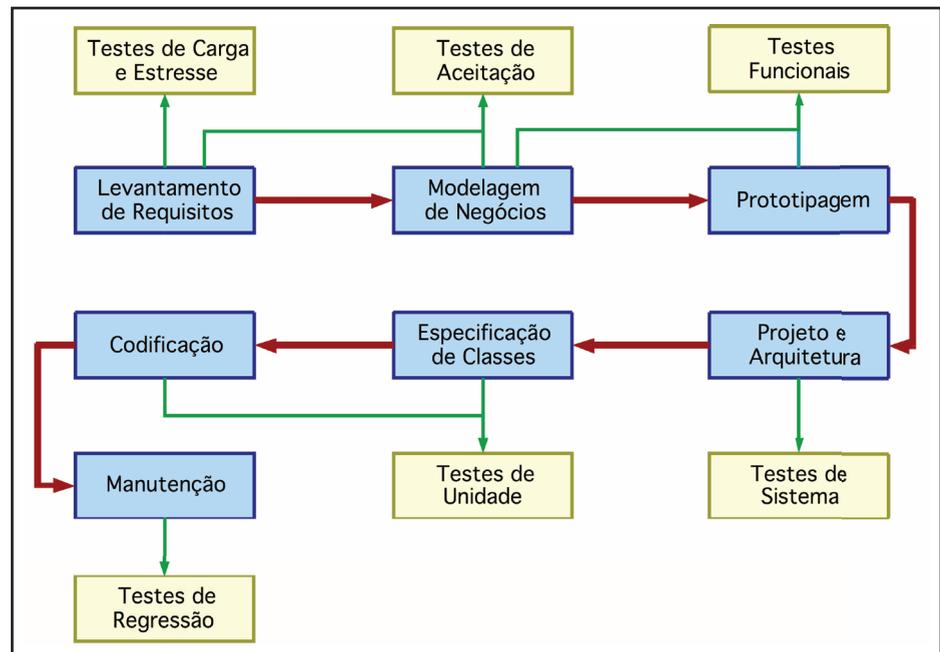


Figura 1. Etapas de um processo de desenvolvimento de software (azul) x tipos de testes (amarelo)

Levantamento de requisitos – testes de carga, de estresse e de aceitação

Como sabemos, todo projeto de desenvolvimento de software é iniciado pelo levantamento e documentação dos requisitos que devem ser cumpridos pelo software. Seguindo a definição clássica, estes requisitos podem ser *funcionais*, referindo-se diretamente ao funcionamento da aplicação sob o ponto de vista do usuário final; ou serem *não-funcionais*, tratando de questões independentes dos processos de negócios, por exemplo alta disponibilidade ou limites sobre tempos de resposta.

Os testes que validam requisitos funcionais estão mais focados nos processos de negócios em si do que na tecnologia utilizada para implementar o software. Eles devem tanto quanto possível ser definidos de modo independente da interface com o usuário ou da formatação final de documentos e relatórios. Isto facilita sua posterior aplicação aos testes de sistema em etapas seguintes.

Os testes que validam requisitos funcionais e não-funcionais são muitas vezes chamados conjuntamente de **Testes de Aceitação**. Isso porque o seu sucesso ou insucesso determina se a aplicação atingiu ou não seus objetivos e condiciona a aceitação do produto final pelo usuário, especialmente se o desenvolvimento é terceirizado.

☞ É importante não confundir os **testes de aceitação**, que validam requisitos funcionais, com os **testes funcionais**, que são focados na interface com o usuário e serão apresentados mais adiante.

Testes de Estresse verificam se a aplicação continua se comportando de modo correto sob um alto volume de transações e de usuários concorrentes, ou sob condições adversas de rede, memória e outras condições de hardware. Eles ajudam a identificar problemas de contenção² e de concorrência³ em uma aplicação, assim como falhas no tratamento de erros.

Já os **Testes de Carga** são focados em determinar a capacidade da aplicação em um ambiente específico, buscando identificar quantos usuários ou quantas transações podem ser sustentados dentro de parâmetros pré-determinados de performance como *throughput* e tempo de resposta. É um erro grave realizar testes de carga sem antes definir parâmetros de performance aceitáveis, incluindo a taxa de erros (por exemplo, timeouts). Afinal, um sistema degradado pode ser tão ruim para o usuário ou para o negócio quanto um sistema inoperante.

Pode acontecer de os testes de aceitação de uma aplicação, sob o ponto de vista do usuário, incluírem testes focados em requisitos tecnológicos bem específicos; por exemplo a capacidade de se processar um arquivo em formato pré-determinado fornecido exigido por uma instituição financeira ou um órgão governamental. Este tipo de teste, embora seja baseado em um requisito real da aplicação, será bastante semelhante a um teste de sistema (visto depois). Daí a importância de se planejar e definir testes não em termos da implementação dos próprios testes, mas em relação aos objetivos de cada teste.

FIT

O framework FIT foi criado partindo do princípio que testes de aceitação têm que ser escritos pelo usuário, e não por um

desenvolvedor. Por isso o FIT parte de ferramentas que todo usuário entende: tabelas em editores de texto ou planilhas eletrônicas. As tabelas relacionam claramente entradas e saídas. Entre as tabelas pode haver também texto livre, associando cada conjunto de entradas e saídas a um caso de uso ou requisito específico.

O uso do FIT permite que os próprios documentos que descrevem requisitos ou casos de uso sejam utilizados como entrada para os testes de aceitação. Basta que eles contenham tabelas em formato predeterminado, para relacionar entradas e saídas de cada situação.

O usuário monta as tabelas na ferramenta de escritório de sua preferência, ou em um editor HTML. Então o framework extrai as tabelas e começa a executar os testes, passando como argumentos os valores de entrada e verificando se as saídas correspondem aos valores descritos nas tabelas. A organização do código Java dos testes e sua execução é propositalmente semelhante ao uso do JUnit. Dessa forma, cada tipo de profissional (usuário e desenvolvedor) lida com as ferramentas que lhe são familiares.

O FIT é independente da tecnologia de interface com o usuário. Um teste FIT pode até comandar a execução de aplicações não-iterativas, como um web service, ou comandar diretamente componentes que representam lógica de negócio, como EJBs session. Na maioria das vezes, o FIT será utilizado em conjunto com outro framework de testes focado em automatizar um tipo particular de tecnologia.

JMeter

O JMeter é provavelmente a mais popular das ferramentas livres para simular usuários e gerar requisições em testes de estresse e de carga. Enquanto a maioria dos softwares livres citados neste artigo são frameworks para a programação dos próprios testes, o JMeter é uma ferramenta que pode em muitos casos ser utilizada sem nenhuma programação.

O JMeter foi originalmente criado para gerar requisições HTTP, para testar a capacidade de um servidor ou aplicação web. Mas logo a flexibilidade do Java permitiu que a ferramenta evoluísse para gerar car-

² Há contenção basicamente quando o código fica parado aguardando pela liberação de algum recurso, limitando assim a capacidade de se atender a requisições simultâneas. Por exemplo, quando um registro de um banco de dados está travado por um `select for update` executado em outra transação ainda não confirmada.

³ Erros de concorrência são *deadlocks*, condições de corrida (*race conditions*) e outras situações onde um código que funciona corretamente quando executado isoladamente passa a gerar erros intermitentes quando executado simultaneamente por vários usuários ou várias transações.

ga para outros tipos de servidores/containers, como EJB, JMS, LDAP, e também para bancos de dados relacionais. Na verdade é possível “plugar” qualquer tipo de componente ao JMeter utilizando *players* customizados. Por exemplo, você pode executar testes JUnit com o player correspondente, reutilizando testes criados originalmente com outros objetivos, e ao mesmo tempo validar o comportamento da aplicação em alto volume de demanda.

O trabalho real do JMeter não é executar os testes ou gerar requisições de rede. É sim seqüenciar um conjunto de requisições, de modo que sejam estatisticamente equivalentes a um conjunto de usuários reais. O JMeter também mede os tempos de cada execução e gera estatísticas como por exemplo o tempo médio, o tempo máximo e o desvio padrão do tempo de resposta. Estas estatísticas podem ser visualizadas em tabelas e gráficos. Também podem ser arquivadas para análise e comparação posterior com outras execuções dos mesmos testes – por exemplo uma execução simulando 100 usuários contra uma simulando 200.

Dessa forma, o JMeter permite validar coisas como “todas as requisições devem ser completadas em menos de 10 segundos, com o tempo médio de 3 segundos e 90% das requisições sendo atendidas em menos de 5 segundos”. Observe como não é possível validar um requisito desse tipo com valores exatos. Deve-se levar em consideração a variação que irá existir no mundo real e definir qual a *faixa* aceitável.

Para lidar com ambientes de alto volume, o JMeter pode ser executado em modo texto (pois toda interface gráfica tem um limite para o volume de atualizações de tela que será possível sustentar). Neste caso os logs dos testes deverão ser processados posteriormente para gerar as estatísticas e gráficos relevantes, e depois se avaliar se o teste foi bem ou mal sucedido.

Outra facilidade embutida no JMeter é o uso de RMI para comandar a execução de vários “servidores” JMeter escravos. Assim, se a capacidade de hardware de um computador for o limite para a geração de carga, é possível agregar vários computadores gerando carga para o mesmo teste, e consolidar os logs de todos eles.

Modelagem de negócio e prototipação – testes funcionais

Muitos processos de desenvolvimento iniciam a modelagem sob um ponto de vista abstrato, independente da tecnologia, de modo a capturar melhor os detalhes dos processos de negócios que serão implementados pelo sistema. De certa forma, a modelagem de negócios é um detalhamento dos requisitos, em especial dos fluxos alternativos e de erros. Portanto essa etapa irá gerar mais testes de aceitação.

É comum que a modelagem de negócios (ou o próprio levantamento de requisitos) utilize a prática de *prototipagem*. Esta prática consiste em se desenhar “rascunhos” ou “simulações” de telas e relatórios que serão parte do sistema, para fornecer aos usuários algo mais concreto com que trabalhar. Isto ajuda muito os usuários a expressarem detalhes sobre o funcionamento esperado das aplicações e dos próprios processos de negócios que eles automatizam.

A prototipagem permite que se inicie a construção dos chamados **Testes Funcionais**, que validam as funções acessíveis através da interface com o usuário. Um teste funcional deve comprovar que foi apresentada a seqüência correta de telas, contendo os dados corretos em cada etapa, e formatados também do modo esperado.

Podem ser criados testes funcionais para componentes não-interativos de um sistema, por exemplo para um web service. Afinal estes componentes também permitem o acesso do mundo externo ao sistema. A idéia é a mesma de um ator em um caso de uso UML, que pode ser um usuário humano ou outro sistema com o qual há troca de informações. Mas aqui os testes funcionais começam a se confundir com os testes de sistema, que serão apresentados mais adiante.

HttpUnit

O HttpUnit fornece um cliente HTTP não-interativo, capaz de enviar e receber respostas de requisições GET e POST e gerenciar cookies. Suporta assim sessões HTTP de um container web Java EE, e até a execução de código JavaScript. (Para este último recurso, é utilizado o Rhino, um interpretador JavaScript escrito inteiramente em Java e mantido pela Mozilla Foundation.)

Com os recursos do HttpUnit, é possível

programar em Java toda a conversação que um usuário, ou melhor, seu navegador, teria com uma aplicação web – mesmo que ela não seja escrita em Java. O HttpUnit fornece ainda várias classes utilitárias para extrair informações de uma página HTML, construir requisições em formulários HTML, e até simular eventos JavaScript.

Para a grande maioria das aplicações, o HttpUnit permitirá uma simulação fiel do comportamento do usuário. Interações mais complexas podem ser simplificadas pelo uso de linguagens de script como o BeanShell. E o uso do HttpUnit em conjunto com o FIT permite simular várias condições diferentes com pouco esforço de programação.

Note que o HttpUnit não simula fielmente um navegador real, com todas as suas nuances (e particularidades) em relação ao tratamento de DHTML e JavaScript. Isso pode ser particularmente grave em aplicações com uso pesado de Ajax, e para desenvolvedores que não estejam habituados com práticas de programação independentes do navegador. Mas o HttpUnit tem a vantagem de ser executado de forma leve, sem necessidade de uma tela gráfica, o que o torna especialmente amigável a ambientes de integração contínua (discutidos adiante).

Selenium

Nos casos em que uma simulação de navegador web como o HttpUnit não atende, a solução é usar um navegador web real. Esta é a proposta do Selenium, que utiliza código Java e JavaScript para comandar a execução de um navegador Mozilla, Firefox ou Internet Explorer em ambientes Linux, Windows ou Mac OS. O mesmo script de testes do Selenium poderia validar se uma aplicação se comporta corretamente com diferentes navegadores web⁴.

⁴ O bom desenvolvedor estará preocupado não apenas com a compatibilidade entre diferentes navegadores, mas também entre diferentes versões do mesmo navegador. Neste caso, usuários Windows estão com problemas sérios, porque não existe uma forma fácil de se ter várias versões do IE instaladas no mesmo computador sem limitar severamente os recursos das versões mais antigas. Veja positioniseverything.net/articles/multiIE.html para mais informações. As soluções são usar software de virtualização como o VMWare ou, surpreendentemente, usar o Linux para rodar o IE; veja como em detalhes no site webexpose.org/2007/01/07/internet-explorer-7-on-linux.

É possível utilizar o Selenium de forma programática em Java; neste caso, ele fica semelhante ao HttpUnit. O Selenium pode também gravar e re-executar uma interação real do usuário com o navegador. Com a ferramenta, é fornecido um IDE JavaScript (um plug-in para o Firefox), que facilita a construção de testes programáticos e gravados, além do acompanhamento da execução desses testes.

Infelizmente o Selenium pode ser pesado e um tanto instável, por causa das idiosincrasias dos próprios navegadores com a execução de JavaScript e DHTML. E seu uso em ambientes de integração contínua exige a disponibilidade de um ambiente gráfico, o que aumenta a demanda de memória e processamento.

Usuários Windows terão problemas para rodar múltiplos testes do Selenium concorrentemente em um mesmo servidor de integração contínua, devido a limitações do próprio sistema operacional. Já usuários Linux e Unix poderão contar com vários recursos como o xvfb⁵ ou o VNC para simular múltiplas sessões interativas em um mesmo servidor – sem para isto necessitar de vários monitores e placas de rede, nem precisar de múltiplas estações de trabalho conectadas a desktops remotos.

JFCUnit

O JFCUnit cumpre papel semelhante ao HttpUnit, mas para aplicações Swing. De modo similar ao Selenium, ele permite a “gravação” de sessões de um usuário real. Essas sessões podem ser posteriormente re-executadas e modificadas por scripts, por exemplo para criar variações nos dados de entrada. Assim como o HttpUnit, o JFCUnit pode se beneficiar muito do uso conjunto com o FIT.

☞ *Usuários Eclipse podem encontrar recursos semelhantes aos do JFCUnit, porém voltados para aplicações SWT, como parte do projeto TPTP.*

Projeto e arquitetura de sistemas – testes de sistema

A modelagem de negócios fornece uma

visão inicial das classes que farão parte do sistema final, incluindo seus principais atributos, relacionamentos e métodos. Mas esta visão de negócios deve ser posteriormente transferida para uma visão de tecnologia, que será diferente (por exemplo) para aplicações gráficas ou web. Essa visão também será grandemente influenciada pela escolha dos frameworks usados para implementar o sistema. A modelagem do sistema passa para a definição de uma *arquitetura*, e dentro desta arquitetura um *projeto físico* irá determinar exatamente quais classes serão construídas e com quais papéis.

Nesta etapa, os testes não serão tão abrangentes quanto testes de aceitação e testes funcionais, podendo se focar em pedaços menores da aplicação. Os testes serão influenciados por nomes de classes, protocolos de rede e outros detalhes de implementação. Mas ainda irão exigir a presença de várias classes; por exemplo um objeto de negócios e vários objetos persistentes. Também poderão ser necessários recursos externos ao código da aplicação, como um servidor de banco de dados.

Esses testes, que podem ser considerados como sendo intermediários entre os testes de aceitação/funcionais e os testes de unidade (que ainda serão apresentados), são chamados de **Testes de Sistema**, sendo também conhecidos como “Testes de Integração” ou “Testes de Componente”.

A principal diferença entre um teste de aceitação e um teste de sistema é que o teste de sistema é condicionado pela arquitetura e pelo projeto tecnológico. Então cada teste de sistema pode se limitar a um subconjunto de um dos fluxos de execução de um caso de uso, ou pode ter exigências diferentes de infra-estrutura.

A execução de testes de sistema dentro de ambientes de integração contínua pode ser complexa, dependendo das tecnologias adotadas e da forma como a arquitetura e design das classes foi definido. Muitos frameworks populares, por exemplo o Struts, estão sendo remodelados de modo a facilitar seu uso junto com práticas de TDD. Outros foram criados explicitamente com a premissa de que serão usados em ambientes de TDD e de integração contínua, como o Spring.

Na verdade, mesmo a execução de testes de sistema sem integração contínua pode ser complicada, pela necessidade de se configurar todo o ambiente de execução, por exemplo o servidor de aplicações, o banco de dados e um diretório LDAP. Isso além do tempo gasto em tarefas como o deployment de pacotes Java EE e a carga de massas de dados de testes no banco de dados. É nessas tarefas que os frameworks especializados em testes de sistema ajudam o desenvolvedor.

Cactus

O Cactus visa facilitar o teste de classes e componentes que funcionam dentro de um servidor de aplicações Java EE. Ele permite executar os testes com maior performance, porque coloca-os dentro do próprio servidor de aplicações. Também permite testar coisas que não seriam testáveis fora do componente, como métodos locais de EJBs.

O Cactus facilita ainda testes de unidade para determinados componentes que são difíceis ou impossíveis de executar de forma isolada, por exemplo taglibs JSP. Em alguns cenários, o Cactus até mesmo elimina a necessidade de um framework especializado para testes funcionais como o HttpUnit.

Existem frameworks que visam facilitar o teste de componentes Java EE pela eliminação dos containers, por exemplo o **MockRunner**. Já alguns frameworks buscam automatizar justamente a configuração do container, facilitando a execução dos mesmos testes em diferentes servidores de aplicações. Este é o caso do **Cargo**. Ambos são relacionados nos links no final do artigo.

DbUnit

A maior dificuldade na construção de testes de sistema é assegurar que estes testes sejam repetíveis, ou seja, que possam ser executados múltiplas vezes sem que repetições seguintes sejam prejudicadas pelas execuções anteriores. Isto significa que os testes têm que ser iniciados em um ambiente completamente conhecido, o que inclui configurações de servidores e repositórios de dados.

O framework DbUnit visa garantir a repetibilidade dos testes em relação ao banco de dados. Ele fornece ferramentas

⁵ O xvfb é um Servidor X (base do ambiente gráfico do Unix) que usa memória RAM para simular uma tela gráfica.

para carregar de forma eficiente uma massa de dados de testes em um banco de dados relacional qualquer. De quebra, permite comparar as situações do banco antes e depois.

XMLUnit

O Framework XMLUnit tem dois usos principais. O primeiro, mais aplicável em testes de aceitação ou em testes de funcionais, é a facilidade em se usar documentos XML para variar os parâmetros de entrada e saída de testes específicos. Neste caso, é uma alternativa ao framework FIT apresentado anteriormente.

O segundo uso é a capacidade de gerar diferentes documentos XML de entrada e validar os documentos XML de saída contra DTDs, XML Schemas ou transformações XSLT. O XMLUnit também fornece APIs para facilitar a extração de dados de documentos XML, o que é particularmente útil para validar a execução de web services.

Struts Test Case

O Struts é talvez o framework de aplicações mais popular no universo Java EE, mas em sua versão 1.x é bem difícil fazer testes de unidade ou de sistema sobre aplicações Struts. Um dos motivos é que várias classes em uma aplicação Struts, por exemplo Actions e Validators, têm que estender classes do próprio Struts.

Inserir os JARs do próprio Struts no ambiente de testes (e de integração contínua) não seria um problema tão grande se as classes do framework, por sua vez, não dependessem de várias classes da API de Servlets. Na verdade, as classes da aplicação que interagem com o Struts também referenciam classes da API de Servlets. Assim, qualquer tipo de teste sobre classes de controle ou de visão da aplicação Struts exigiria um container web completo.

O framework Struts Test Case permite executar classes Action e outras do Struts

fora do container web e sem a necessidade de um cliente HTTP gerando requisições. Ele faz isso fornecendo *objetos mock*⁶ para a API de Servlets.

A lógica de controlador de certas aplicações pode ser bastante complexa e depender significativamente das definições nos arquivos de configuração do Struts. Por isso, ter a capacidade de executar testes sobre classes do Struts de forma rápida e leve pode ser um grande diferencial na qualidade da aplicação final.

Codificação e manutenção – testes de unidade e de regressão

Finalmente chegamos à etapa de codificação de um sistema Java. Foi nesta etapa que se iniciou a revolução do TDD, pela constatação de que não adianta ter toda uma modelagem e documentação detalhados, se o código entregue ao usuário for lento ou instável. Qualquer esforço de QA será inútil se ele não se iniciar garantindo que seja realizada uma codificação de qualidade. Afinal, é claro, o produto que realmente importa ao usuário é o código executável, não os documentos de requisitos ou modelos UML.

O mais simples dos testes orientados a código é o chamado **Teste de Unidade** (algumas vezes chamado “Teste Unitário”). Na sua forma mais pura, este tipo de teste visa verificar o funcionamento de uma classe ou método de forma isolada, independentemente de outras classes e métodos que sejam utilizados pelo código sendo testado.

A maioria dos desenvolvedores Java já ouviu falar ou já experimentou o JUnit ou outro framework para testes

⁶ Um *objeto mock* é basicamente uma simulação de outro objeto. Ele pode usar o mesmo nome, inclusive o pacote do objeto real, mas funciona de modo isolado, sem todas as dependências do objeto original e sem implementar a sua funcionalidade interna.

unitários. Mas é importante lembrar que criar testes de unidade não é suficiente para a adoção do TDD. Os testes de unidade são apenas parte de um processo bem mais amplo de controle efetivo de qualidade de software.

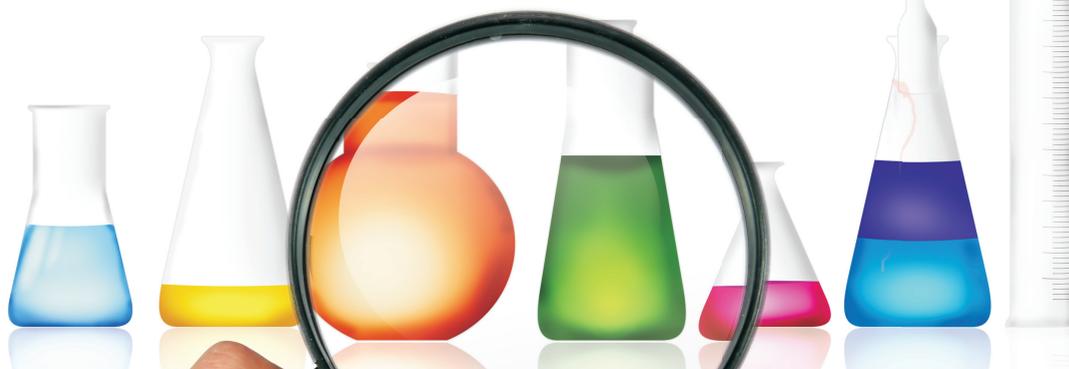
Nem todas as classes de uma aplicação podem ser testadas de modo isolado, seja porque elas necessitam de outras classes da própria aplicação, ou porque precisam de acesso a recursos externos, como bancos de dados relacionais. Testar uma classe juntamente com seus pré-requisitos consiste em um teste de sistema. Então, ou não será possível criar testes de unidade para algumas classes da aplicação, ou então será necessário fornecer “simulações” destas dependências na forma de *objetos mock*.

É importante perceber que usar objetos mock em todas as camadas de uma aplicação, de modo a ter testes de unidade para todas as classes e métodos de uma aplicação, não elimina a necessidade de se ter também os outros tipos de testes.

Ter componentes funcionando corretamente de forma isolada não é a mesma coisa, claro, que ter componentes que funcionam corretamente em conjunto. Afinal, cada componente⁷ é implementado e testado partindo-se do pressuposto que suas interfaces externas foram corretamente e completamente especificadas. Nada impede que um componente tente usar outro componente de forma não prevista pela sua especificação. E este uso incorreto só irá aparecer nos testes de sistema, ou talvez apenas em testes mais abrangentes, como os funcionais ou de aceitação.

Quando qualquer sistema é exposto ao usuário final, sabemos que é inevitável a constatação da existência de erros na

⁷ Para os fins deste artigo, componente e classe podem ser considerados sinônimos.



aplicação. Uma prática bastante efetiva de QA é gerar testes que exercitam cada bug em particular. Estes testes são utilizados para ajudar o desenvolvedor a isolar a causa do erro, e permitem ao gerente de projeto verificar se o bug foi realmente eliminado.

Além disso, se estes testes forem incorporados à bateria de testes da aplicação, e forem re-executados a cada release (ou como parte do processo de integração contínua) evita-se que o bug apareça novamente. Os “bugs recorrentes”, ou *regressões*, são um dos problemas mais comuns com sistemas que já sofreram vários releases ou que sofreram várias mudanças em sua equipe de desenvolvimento. Por isso estes testes são chamados de **Testes de Regressão**, que são nada mais do que testes de unidade, de sistema ou funcionais que foram escritos não em resposta a um requisito, mas em resposta a uma falha observada.

JUnit

O JUnit é provavelmente o framework de testes mais conhecido e utilizado na comunidade Java. Todo IDE Java que se preza fornece algum suporte ao JUnit, e a documentação da maioria das APIs e frameworks populares demonstra como usá-los dentro de testes JUnit. Isso quando eles mesmos não usam o JUnit em seus processos de QA.

O JUnit definiu a organização de testes em casos (**TestCase**) e suítes (**TestSuite**), além dos “executores” (**TestRunner**) que a maioria dos frameworks posteriores imitou. O JUnit ajuda não só a escrever os testes, mas também a executar e tabular os resultados com uma interface gráfica ou numa intranet. Na verdade, a maioria dos frameworks citados neste artigo foram projetados como extensões para o JUnit.

Note entretanto que um teste criado com o JUnit não é necessariamente um teste de unidade. Se a execução do teste depender de outros pré-requisitos além das classes e métodos sendo chamados diretamente, o JUnit terá sido usado para a criação de testes de sistema. E usando o JUnit junto com o HttpUnit ou o JFCUnit, temos testes funcionais.

Existem frameworks alternativos ao JUnit, por exemplo o **TestNG**, mas por mais

que se possa argumentar sobre os méritos técnicos dos concorrentes, o JUnit permanece o mais popular, e sua versão 4 faz uso de novidades do Java SE 5, especialmente anotações, para simplificar ainda mais a escrita de testes.

Qualidade dos testes

Os testes automatizados constituem a mais importante de todas as ferramentas de controle de qualidade de software hoje em dia, e oferecem também a maneira mais confiável de se acompanhar o progresso de uma equipe de desenvolvimento em relação aos requisitos do projeto. Por isso é fundamental avaliar a qualidade e a abrangência dos próprios testes.

Se a documentação dos requisitos está completa, é relativamente fácil verificar se há testes de aceitação e funcionais para cada fluxo de cada caso de uso. Esta verificação estabelece o mínimo para avaliar a qualidade de uma bateria de testes.

Já avaliar a qualidade dos testes de sistema e de unidade é um pouco mais complicado. A modelagem e a especificação de interfaces públicas para cada componente ou pacote fornecem uma maneira para se avaliar o conjunto de testes, mas isso não é suficiente.

Para avaliar a qualidade de testes sobre o código não basta olhar o código “de fora”, como uma caixa preta. É necessário olhar também o código internamente, e testar a implementação específica. Durante muito tempo, foram utilizadas ferramentas de cobertura para avaliar a qualidade dos testes. A idéia é rodar toda a bateria de testes e verificar se existe alguma linha de código que não foi exercitada. Não apenas os testes de unidade deveriam ser avaliados em relação à cobertura, mas também os testes de sistema e especialmente os testes funcionais. Se a execução dos testes passa por todo o código, podemos confiar que tudo o que foi previsto pelo sistema foi testado⁸.

⁸ Veja que um conjunto “completo” de testes, em termos de cobertura, não garante que todos os requisitos foram implementados pelo sistema. Por isso é importante cruzar os testes com os requisitos, e definir formalmente os testes de aceitação junto com o levantamento de requisitos e a modelagem de negócios.

Anti-patterns para testes

Construir e programar testes unitários ou outros tipos de testes pode não ser um exercício tão simples quanto se imagina. Não pela dificuldade da programação em si, mas pela criatividade e experiência necessária para se imaginar testes eficazes e concretos.

Com vistas a orientar os leitores a construir testes mais eficazes, elaborei esta lista com erros comuns – ou anti-patterns – baseada na minha experiência de consultor orientando equipes de desenvolvimento nas práticas de TDD.

Se não deu erro, é por que funcionou

O teste chama algum método de negócios, que não retorna nada, e não verifica se os efeitos colaterais que seriam gerados pelo método foram efetivamente realizados.

Por exemplo, foi testado o envio de uma mensagem de e-mail, mas não se verificou se ela chegou à caixa postal correta. Ou então foram modificados registros no banco de dados, mas estes registros não foram lidos para confirmar que estão com os valores corretos ao fim da transação.

Em suma, caímos nesse erro se considerarmos que o teste só “falha” se houver alguma exceção. Mas não ter ocorrido nenhum problema de rede, nem de sintaxe SQL, por exemplo, não significa que foi gerado o resultado correto.

Note que em geral este tipo de problema ocorre em testes que não são verdadeiros testes de unidade, e sim testes de sistema, devido à dificuldade em se verificar os “efeitos colaterais”. Mas podem ocorrer variações deste anti-pattern até em cálculos simples; por exemplo o teste verifica apenas se o resultado é positivo, em vez de verificar o valor real esperado.

Fazer rollback no final do teste

O teste chama um ou mais métodos cujo efeito é modificar registros no banco de dados, mas para evitar o problema de se ter que “zerar” as tabelas envolvidas (para manter a repetibilidade do teste), é feito um rollback no final.

Especialmente em aplicações transacionais, esta forma de teste não funciona, pois não considera a possibilidade do próprio commit falhar. Na maioria dos casos um banco

relacional irá executar regras de integridade (constraints) e triggers imediatamente, de modo que uma falha no commit seria causada por falta de espaço em disco ou algum outro motivo fora do controle da lógica da aplicação. Mas em situações mais sofisticadas, como um banco replicado, ou com transações distribuídas, pode haver falha apenas no commit, e não nos comandos que inseriram ou modificaram registros.

Além disso, caso sejam usados Entity Beans ou frameworks de persistência objeto-relacional, as atualizações reais sobre o banco de dados podem ser postergadas para o momento do commit, mascarando erros causados por lógica incorreta em métodos executados no início da transação.

Não podemos deixar de considerar que o commit da transação deve ser feito nos momentos corretos. O commit é parte integral da lógica sendo testada. Por exemplo, se o commit estiver fora do método que representa todo o processo de negócio, a camada de apresentação da aplicação poderá até mesmo “esquecer” de fazer o commit.

O próprio método de testes calcula o resultado esperado

Um dado método de negócios implementa um algoritmo com diversas variações. Em vez de criar vários testes, cada qual chamando o mesmo método, porém variando os parâmetros de entrada (e as respostas esperadas), cria-se um teste “genérico” que calcula as respostas esperadas e então chama o método de negócios apropriado.

Além de representar uma duplicação desnecessária de lógica (implementando o algoritmo de cálculo duas vezes, uma no teste e outra no método de negócios), é possível que o programador cometa o mesmo erro nas duas vezes, ou pior, que haja cópia de código do método para o teste, ou vice-versa.

Do jeito que foi apresentado, pode parecer uma situação trivial, fácil de se identificar. Mas o programador pode tomar “atalhos” para produzir rapidamente o teste. Por exemplo, quando o resultado de um método depende de uma tabela de consulta e o teste é codificado para usar a mesma tabela.

Testar apenas o funcionamento correto

O método nunca é chamado com argumentos inválidos, então não se sabe se ele irá gerar as exceções esperadas ou de alguma forma dar o feedback correto para o usuário

Aqui não temos exatamente um erro de um teste em especial, mas um erro na concepção de toda uma suíte de testes. É importante testar se cada método faz sua validação de argumentos e tratamento de erros da forma correta. Especialmente quando um método pode ser chamado usando-se argumentos derivados de fontes externas, como um formulário HTML, um banco de dados ou um arquivo em disco. As falhas de segurança mais comuns em aplicações web decorrem deste anti-pattern.

Testar várias situações diferentes em um único teste

Um único método de negócios pode gerar múltiplas situações de teste, de acordo com apenas os valores dos seus argumentos, e um único teste exercita todas essas variações.

Este anti-pattern ocorre quando o programador acredita que deve haver um teste para cada método (afinal é assim que a maioria dos IDEs geram automaticamente classes de teste a partir de classes da aplicação).

Um teste construído desta forma pode até servir para indicar se o método como um todo funcionou, mas uma falha no teste, no meio de um relatório que pode conter milhares de outros testes, diz pouco sobre exatamente qual situação falhou.

Outra variação deste anti-pattern é na verdade um erro de projeto da aplicação. Se existe uma seqüência de métodos que deve ser chamada na ordem correta para gerar certo resultado, esta seqüência deveria estar encapsulada em um único método de negócios, e é este método que seria testado. Mas ter a seqüência correta dentro de um método de teste não garante que a mesma seqüência foi implementada corretamente dentro da aplicação.

Confundir Testes de Unidade com Testes de Sistema

Cria-se um teste de unidade onde o sucesso ou falha do teste depende de métodos e classes

diferentes do alvo do teste.

Um teste de unidade deve testar uma classe ou um método isoladamente, sem nenhuma dependência externa. A idéia é que uma falha no teste indique precisamente qual método ou classe o programador deve depurar e corrigir, ou completar. Mas, caso o teste dependa de outros métodos chamados em cascata, ou de recursos externos como bancos de dados, uma falha no teste pode significar simplesmente um erro de configuração do ambiente, ou uma falha no trabalho de outro desenvolvedor.

Nem sempre será possível ou viável criar verdadeiros testes de unidade para todas as classes e métodos. Já os testes de sistema devem ser isolados para que, antes da sua execução (ou em caso de falhas), seja fácil verificar falhas de ambiente, ou acionar o desenvolvedor responsável pelo componente que causou a falha. Isso também ajuda a separar testes longos de testes rápidos, e a agendar freqüências de execução diferenciadas para cada um. Em projetos envolvendo uma equipe grande ou muitos casos de uso, pode se tornar inviável executar sempre todos os testes. Assim, será necessário ter agendas separadas para a execução de diferentes tipos de testes.

Construir testes que não podem ser repetidos

O funcionamento do teste depende de o programador ajustar manualmente algum pré-requisito, como inserir dados de teste no banco de dados.

Um teste automatizado deve funcionar sempre. Deve ser possível executá-lo diversas vezes, sem que o programador tenha que realizar tarefas de limpeza, como recarregar uma massa de testes no banco de dados. Falhar em observar estas diretivas leva a vários alarmes falsos, pois faz com que testes falhem quando o código testado está totalmente correto.

Todo o ambiente de execução de um teste deve ser configurado pelo próprio teste (por exemplo, usando comandos SQL delete e insert executados via JDBC), ou pelo script que comanda a sua execução. Para facilitar esta tarefa, existem vários frameworks, alguns dos quais apresentados na seção sobre testes de sistema neste artigo (ex.: Cactus e DbUnit).

☞ *A cobertura medida em linhas de código pode oferecer uma visão errônea da abrangência dos testes. Uma mesma linha pode participar de vários caminhos de execução diferentes, e cada caminho traz diferentes configurações de objetos em memória.*

Hoje já temos ferramentas capazes de cruzar a cobertura medida em termos de linhas exercitadas pelos testes com os caminhos possíveis de execução dentro de um método. Por outro lado, testar todos esses caminhos pode representar esforço excessivo em relação aos benefícios obtidos. Os próprios criadores do JUnit dizem que nem todos os métodos precisam sofrer testes, pois alguns métodos simples não têm como “dar errado” por si só. Um exemplo óbvio são métodos `getXxx()`.

Um meio-termo entre buscar 100% de cobertura e não saber de forma objetiva se temos ou não testes suficientes para garantir a qualidade da aplicação é usar uma ferramenta de análise estática de código para identificar “pontos críticos”, cujos testes devem ser avaliados com mais cuidado por um profissional especializado. Existem várias métricas formalmente definidas para avaliar a complexidade de classes, pacotes e métodos individuais, por exemplo, o acoplamento entre as classes⁹,

⁹ O acoplamento pode ser medido em termos da quantidade de classes das quais uma dada classe depende (acoplamento aferente), ou em termos da quantidade de classes que dependem dela (acoplamento eferente).

e ampla oferta de ferramentas capazes de calcular estas métricas.

Cobertura

O projeto Cobertura utiliza manipulação de bytecodes para instrumentar o código testado, e registra estatísticas sobre quantas vezes cada linha de código foi executada para cada teste. Também é possível gerar relatórios sobre a cobertura de saltos, que é uma aproximação inicial para a quantidade de caminhos possíveis em uma classe.

A utilidade do Cobertura é identificar facilmente classes, métodos e linhas de código que não foram exercitadas pelos testes, fornecendo um primeiro indicador de quais conjuntos de testes podem precisar de expansões e revisões. O Cobertura pode ser usado tanto em aplicações independentes quanto dentro de containers Java EE.

JDepend

O JDepend é uma das muitas ferramentas de análise estática de código Java capazes de gerar métricas de complexidade e outras métricas de qualidade de código Java. Além da sua execução independente e como parte de scripts de automação, o JDepend pode ser integrado aos IDEs Eclipse e NetBeans por meio de plug-ins inclusos na sua distribuição.

Testes e integração contínua

Ter os testes é uma coisa, mas garantir que sejam executados regularmente pelos desenvolvedores, em especial depois de atividades críticas como um commit de código no repositório, é outra história.

Muitas das ferramentas citadas fornecem alguma integração com IDEs Java populares, e virtualmente todas elas se integram com a ferramenta de automação mais utilizada, o Ant. Assim, é possível configurar o buildfile de um projeto, especialmente com IDEs bem integrados ao Ant como o NetBeans, para executar os testes a cada compilação ou a cada commit ou update do CVS.

Algumas empresas adotam a prática de que falhas dos testes são tão graves quanto falhas de compilação, e que um programador não pode entregar código que falhe em qualquer teste.

Em geral, não só a execução dos testes, mas também a geração dos seus relatórios e a publicação destes em uma intranet, são facilmente integradas a ferramentas populares de automação. Daí para ter os testes sendo executados continuamente em um servidor dedicado, caracterizando a prática da integração contínua, é um passo relativamente simples.

Ant

É mais fácil do que se imagina construir um buildfile Ant que faça todo um processo de integração contínua. Para o Ant é simples trazer código do CVS ou SVN, compilar classes Java, empacotar, fazer deployment em um servidor de aplicações e rodar testes. O Ant também pode iniciar e finalizar servidores de aplicações, editar seus arquivos de configuração e inicializar bancos de dados.

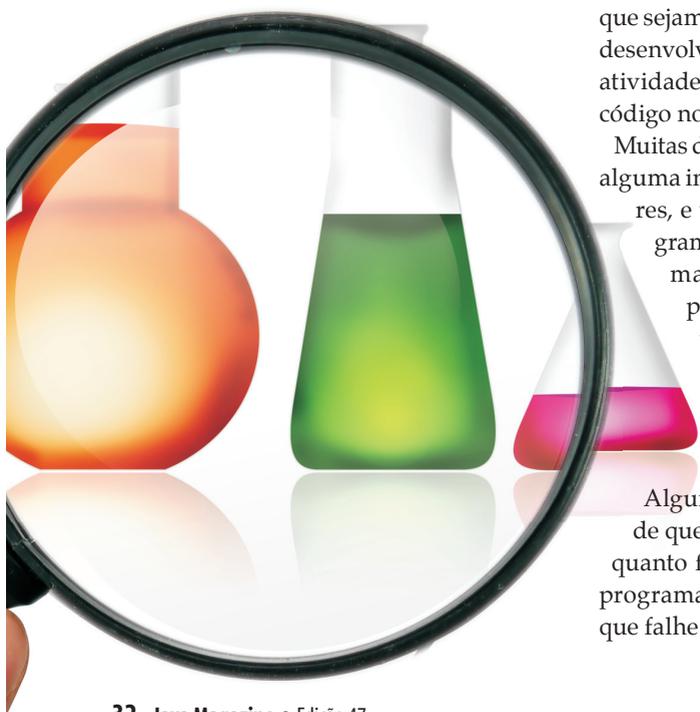
Maven

O Maven é visto por muitos como um “super Ant”. Suas maiores vantagens são o foco em templates de projetos padronizados, de modo que a configuração de layouts de diretórios e a construção de scripts se tornam bem mais simples do que com o Ant, pelo uso de defaults pré-definidos. Outra vantagem do Maven é prever, como parte do processo normal de construção de aplicativos, a geração de relatórios em uma intranet. Os sites de vários projetos da Apache Software Foundation e de outros projetos de software livre são gerados diretamente pelo Maven, sem esforço adicional.

Cruise Control

O Cruise Control é um dos softwares cujo objetivo é automatizar um processo de integração contínua. Ele é configurado com a URL para acesso a um repositório de código, e alguns parâmetros para indicar coisas como o servidor de aplicações ou o banco de dados utilizado. O Cruise Control também pode delegar as tarefas de compilação e execução da aplicação e dos testes para buildfiles Ant ou projetos Maven.

É possível configurar a periodicidade da execução dos testes, ou então configurar sua execução a cada commit no repositório de código. E uma mesma instalação do Cruise Control pode controlar a integração contínua de vários projetos independentes.



xprogramming.com

Site sobre o Extreme Programming (XP).

agiledata.org/essays/tdd.html

Sobre o TDD, uma das principais práticas dos processos ágeis.

agilemodeling.com

Sobre processos ágeis e suas aplicações, junto com processos mais tradicionais como o RUP.

martinfowler.com/articles/mocksArentStubs.html

Sobre o uso de objetos mock com TDD.

jakarta.apache.org/jmeter

JMeter, ferramenta para construção e execução de testes de carga e de estresse, com suporte a vários protocolos Java EE.

openqa.org/selenium

O Selenium é um front-end e um framework para construção de testes de aceitação de aplicações web. Funciona controlando um navegador real, e permite validar características específicas do suporte a DHTML e JavaScript em cada navegador.

jfcunit.sf.net

O JFCUnit simula um usuário clicando e digitando em uma interface Swing.

httpunit.sf.net

O HttpUnit, framework para testes funcionais de aplicações web, simula um usuário em um navegador.

abbot.sf.net

O Abbot é uma alternativa ao JFCUnit que muitos consideram mais simples de utilizar

testng.org

TestNG, alternativa ao JUnit.

strutstestcase.sf.net

O Struts Test Case permite executar classes Action do Struts fora do container web, sem a necessidade de

um cliente HTTP gerando requisições. Assim, torna possível criar testes de unidade (ou de sistema) para validar a lógica de controlador de uma aplicação na arquitetura MVC.

jakarta.apache.org/cactus

Cactus, framework para execução de testes de unidade dentro de um container Java EE. Possibilita testar de modo eficiente Servlets e EJBs, incluindo EJBs que forneçam apenas interfaces locais.

cargo.codehaus.org

Cargo, framework que automatiza a configuração, o deployment e a inicialização/término de servidores Java EE, facilitando testar uma mesma aplicação contra servidores diferentes.

mockrunner.sf.net

O MockRunner simula um servidor Java EE, permitindo o teste de componentes como Servlets e EJBs de forma isolada e mais leve do que em um servidor real.

junit.org

JUnit, o mais popular dos frameworks para testes de unidade em Java (com derivativos para várias outras linguagens como PHP, Python, C# e C++).

fit.c2.com

FIT, ferramenta que permite a confecção de *fixtures* (tabelas de entradas e saídas para testes de aceitação e funcionais) por usuários finais, bem como sua utilização direta por programadores.

jmock.org

O jMock é um framework que simplifica a criação de objetos mock para classes da aplicação ou de APIs externas.

maven.apache.org

Maven, ferramenta de automação de ciclo de vida de projetos de software, com vários recursos especificamente voltados para o TDD.

cobertura.sf.net

Cobertura, ferramenta para análise de cobertura de código, permite avaliar a qualidade dos seus testes de unidade, identificando trechos de código que não foram exercitados por nenhum teste.

darkware.com/software/JDepend.html

JDepend, uma das várias ferramentas para análise estática de código Java, complemento importante para a avaliação da qualidade dos testes de unidade baseada em ferramentas de análise de cobertura.

ant.apache.org

Ant, ferramenta de automação de builds (compilação e empacotamento de componentes e aplicações) que é muito popular para automatizar processos de TDD.

dbunit.sf.net

DbUnit, framework para testes envolvendo bancos de dados relacionais, é capaz de verificar o estado do banco antes e depois de uma transação, e facilita a inicialização do banco com uma massa de testes.

xmlunit.sf.net

XMLUnit, framework de testes unitários para componentes que utilizem documentos XML como formato de dados de entrada e saída. Tem suporte a validações e transformações.

cruisecontrol.sf.net

O CruiseControl é uma ferramenta para automação do processo de Integração Contínua, que pode ser integrada ao Maven e ao Ant, e a várias das ferramentas apresentadas neste artigo.

eclipse.org/tptp

O projeto Testing and Performance Tools integra várias das ferramentas apresentadas aqui (e algumas além do escopo deste artigo, como ferramentas de profiling).



Conclusões

A prática de TDD pode melhorar muito a qualidade e a gerenciabilidade de projetos de desenvolvimento de software, reduzindo incertezas quanto a prazos e custos, e evitando incidentes ao passar o código para o ambiente de produção. Entretanto a aplicação eficiente do TDD envolve reconhecer e utilizar vários tipos diferentes de testes, como foi visto neste artigo.

Também foram apresentadas várias ferramentas e frameworks Java que facilitam

a construção, execução e acompanhamento dos testes, além da aferição de sua qualidade. E o que é melhor, tudo com software livre, que pode ser integrado ao seu IDE preferido ou às ferramentas de automação mais populares, Ant e Maven.

Veja também o **quadro** “Anti-patterns para testes”, que apresenta alguns dos erros mais comuns cometidos pelos desenvolvedores quando iniciam a adoção das práticas de TDD e de integração contínua. ●



Fernando Lozano

(lozano@javamagazine.com.br,
www.lozano.eti.br)

trabalha há mais de 10 anos com desenvolvimento de sistemas de informações e integração de redes, sendo um dos pioneiros do uso do Software Livre no Brasil. Dentro da comunidade, atua como Conselheiro do LPI Brasil, Webmaster da FSF e Community Manager do Java.net. Atualmente é consultor associado à Neki Technologies (www.neki.com.br).



Mini-curso

Programação Java ME

Parte 4: Portabilidade e Desempenho

O leitor que está seguindo este mini-curso verá que nesta edição tivemos que reduzir o ritmo temporariamente, fazendo um artigo mais curto que os anteriores. O motivo é o lançamento próximo do novo Eclipse, evento que sempre pede cobertura num momento oportuno para leitores interessados neste IDE. Teremos pelo menos mais um artigo completo de Java ME neste mini-curso, onde falaremos de outras APIs fundamentais da MIDP, como a RMS e a WMA. (Posteriormente, teremos outros artigos eventuais sobre a plataforma Java ME nesta coluna.)

Todavia, para não deixar o leitor aficionado em Java ME a ver navios, apresentamos nesta edição material concentrado em dois aspectos importantes na criação de aplicações Java ME. O texto não tem dependências fortes com o artigo anterior. O projeto Java ME mencionado em alguns pontos é um gerador de fractais chamado Micromandel, cujos fontes e binários estão disponível no download da edição anterior (e também desta).

A portabilidade das JVMs ME

Como temos visto nas partes anteriores, as JVMs para dispositivos limitados são também limitadas: seus compiladores JIT, Garbage Collectors e outros componentes

são menos avançados que os de JVMs Java SE, pois devem “caber” nos dispositivos. Estas limitações têm uma consequência. APIs, frameworks, aplicações etc. que abusam de design orientado a objetos moderno (abstração, polimorfismo, encapsulamento) e oferecem muita funcionalidade dependem muito de JVMs eficientes. E a falta de determinadas otimizações é ainda mais grave, se lembrarmos que o hardware destes dispositivos é dezenas de vezes inferior ao de PCs.

Código nativo?

Uma solução possível seria explorar mais a opção por código nativo, para implementar com desempenho máximo pelo menos as APIs mais críticas. Isso talvez reduzisse a necessidade de simplificar ou substituir algumas APIs. Parece boa idéia, lembrando



Analizando implementações de Java ME, otimizando código ao máximo e trabalhando para aumentar a portabilidade e reduzir os efeitos da fragmentação

OSVALDO PINALI DOEDERLEIN

que as interfaces nativas proprietárias das JVMs para Java ME são mais eficientes que a nossa conhecida JNI. (Na Java ME, cada JVM define a interface nativa que quiser, que pode ser otimizada para a plataforma. Estas interfaces proprietárias só podem ser usadas pelo runtime Java, jamais por código de aplicações.)

O problema é que o mercado de dispositivos Java ME – basicamente, telefones celulares e PDAs – é ainda muito fragmentado. Quem acha ruim portar código C/C++ entre Windows, Mac OS X e variantes de Linux/Unix, não conhece o caos (tanto na variedade quanto na qualidade) dos sistemas operacionais e SDKs para celulares. Pior ainda, estas plataformas mudam muito rápido, algo típico do mercado de “eletrônica de consumo”. E há um agravante: as JVMs para Java ME precisam ser muito customizáveis, adaptáveis e extensíveis, para se adaptarem não só a diferentes CPUs e sistemas operacionais, mas às decisões específicas a cada fornecedor e a cada produto.

A conclusão é que, longe de poderem ser mais otimizadas para cada plataforma de hardware e sistema operacional, as JVMs ME precisam ser ainda *mais* portáveis que as JVMs SE! Isso é verdade, por exemplo, para a KVM + CLDC HotSpot da Sun (que é licenciada para muitos fornecedores de

dispositivos – e cujo código-fonte já foi totalmente liberado sob licença GPL; por isso podemos examinar). De fato, os fontes da KVM possuem proporcionalmente *menos* código Assembly, *menos* código C/C++, e *menos* código específico para CPUs e sistemas operacionais particulares, do que os fontes do Sun JDK para Java SE. Isso é conseguido com opções de design muito diferentes das que são comuns em JVMs SE.

Design para portabilidade

Por exemplo, a KVM implementa threads por conta própria, sem qualquer ajuda do sistema operacional (portanto, o código que faz isso é 100% portátil). As desvantagens? Praticamente nenhuma, pois (1) o suporte a threads nativos dos sistemas operacionais para dispositivos como celulares varia entre “não suportado” e “péssimo”¹; e (2) estas plataformas nunca têm múltiplas CPUs (multi-core ou SMP), o que seria o maior motivo para se preferir threads nativos.

Pelo contrário, como a JVM é capaz de controlar e modificar facilmente todo o código executável, ela pode inserir no código (interpretado ou compilado) “pontos de *yield*” em locais estratégicos, dando uma chance à JVM de alternar entre vários threads. Em comparação, nas

¹ Tanto o Symbian quanto o BREW só oferecem threads (e multitarefa) “cooperativa”, uma tecnologia dos tempos do Windows 3.1. O Windows CE é melhor, implementando scheduling preemptivo, mas o CE só é capaz de rodar numa variedade de equipamentos muito mais estreita (com requisitos mínimos bem maiores que os celulares médios atuais). Também há alguns fornecedores começando a utilizar o Linux, mas isso é incipiente e ainda não se pode dizer que o Linux esteja bem adaptado para plataformas como celulares.

aplicações nativas estes pontos precisam ser determinados pelo programador, e um trabalho imperfeito pode resultar em problemas que vão desde desperdício de CPU e pequenas “travadas”, até problemas mais sérios como *livelocks*.

Otimizando o bytecode

Quando se trata de economizar bytes, na Java ME nenhuma economia é demais. Ainda mais considerando que muitos dispositivos MIDP se recusam a instalar JARs com mais de 300 Kb. Este é o tamanho máximo cujo suporte é obrigatório pela especificação Mobile Service Architecture (veja a edição anterior).

A compressão dos arquivos JAR ajuda um pouco, pois a especificação fala no tamanho do JAR e não no tamanho normal das classes descompactadas. Ainda assim, para uma aplicação de tamanho realista, 300 Kb não é uma folga muito grande, ainda mais considerando que este limite inclui todos os recursos que possamos querer incluir no JAR, como ícones, imagens e áudio.

Para avaliação, a primeira versão do projeto da edição anterior, com somente código de navegação entre dois forms vazios, gera um JAR de 1.737 bytes (já com compressão ZIP). Não é tão pouco – 0,5% do limite de 300 Kb – para um programa que ainda não faz praticamente nada!

☞ *Leitores mais experientes em Java SE lembrarão do compressor Pack200, que permite reduzir arquivos JAR mais ainda do que qualquer outro compressor de uso geral, e é muito utilizado para gerar instaladores ou em aplicações Web Start. O problema é que o Pack200 é pesado para os padrões dos dispositivos ME, e só é adequado à distribuição por rede. Quando você instala o Sun JDK ou JRE, que é um procedimento bem demorado considerando o tamanho do instalador, a maior parte do tempo é gasta pela descompressão*

Pack200. Mas graças a isso só precisamos fazer um download de 57 Mb para instalar um software de 162 Mb (usando o JDK 6.0u1 como exemplo). Porém, o Pack200 só se justifica pela economia de tempo de transmissão por rede. Na instalação, as classes precisam ser convertidas para o formato normal (arquivos JAR comuns utilizando somente compressão ZIP). Se fossem mantidas em formato .pack, seu carregamento pela JVM seria lento demais. Isso porque este formato é “denso”; não permite ler uma classe arbitrária no meio do arquivo sem antes descomprimir todo o arquivo até aquela posição.

Para reduzir o bytecode ainda mais, utilizamos **ofuscadores de bytecode**. Estas ferramentas (como sugere seu nome) foram originalmente criadas para proteção de propriedade intelectual. Elas distorcem o bytecode, por exemplo substituindo nomes de atributos e métodos por nomes sem significado. Isso torna muito mais difícil o trabalho de algum xereta com um descompilador, que queira fazer a engenharia reversa dos fontes (pois os .class normais são fáceis de ler e de transformar automaticamente em código-fonte razoavelmente próximo ao original).

Mas as pessoas logo descobriram que um “benefício colateral” dos ofuscadores era a redução dos arquivos .class, por exemplo devido à substituição de identificadores longos (como “executaSolicitacaoDoUsuario”) por outros muito menores (como “a”). Os ofuscadores logo se tornaram uma ferramenta essencial para desenvolvedores Java ME, mesmo os que não se preocupem com ataques de piratas.

Tanto o NetBeans Mobility Pack quanto o EclipseME e o MTJ usam o ofuscador open source ProGuard para otimizar

o bytecode. Sugiro o uso das seguintes opções do ProGuard (no NetBeans, ao invés de usar um dos níveis de ofuscação predefinidos, utilize o nível 1 – só opções customizadas):

```
-dontusemixedcaseclassnames
-defaultpackage "
-overloadaggressively
-allowaccessmodification
-keep public class ** extends javax.microedition.midlet.MIDlet
```

Estas opções preservam unicamente o nome da classe principal da MIDlet e os métodos que esta classe redefine. O resultado, para a versão final do mesmo projeto criado na edição anterior, é uma redução de 9.481 bytes (16.708 bytes descompactados) para 6.814 bytes (13.684 bytes descompactados). Trata-se de um corte de 30% comparando os arquivos JAR distribuíveis. Nada mau, mesmo considerando que boa parte da redução é devida à supressão de informações de depuração (como nomes de variáveis locais).

Desempenho: Java ME X Java SE

Eu não poderia perder a oportunidade de medir o desempenho do meu gerador de fractais, agora na versão Java ME! É um benchmark interessante de aritmética em ponto flutuante. Veja os resultados na **Tabela 1**.

Java SE

O primeiro grupo, “Referência”, contém os escores para a versão em Java SE do gerador de fractais, modificado para ter um comportamento semelhante ao do Micromandel, mas rodando no HotSpot Server 6.0. Seu desempenho é de longe o melhor, o que não é nenhuma surpresa:

calculamos 80 fractais de 176x176 por segundo, mesmo com a precisão aritmética máxima (**double**). Isso permitiria até mesmo fazer uma animação de 80 quadros por segundo, com cada quadro consistindo de um fractal gerado em tempo real².

Para gerar este fractal de teste, o programa executa cerca de 3,2 milhões de vezes cada “passo” (fórmula de Mandelbrot, dentro do loop interno de métodos como **calcDouble()**). Um desempenho de 0,012s indica que cada passo executou em 4 nanossegundos. A CPU desktop utilizada tem 3,4 ciclos de clock por nanossegundo³, então cada passo executa em aproximadamente 14 ciclos. Como estas CPUs são superescalares (capazes de executar mais de uma instrução completa por ciclo de clock – até umas cinco instruções por ciclo, na CPU utilizada), este resultado é compatível com a quantidade de operações que temos em cada passo, supondo que temos um compilador JIT que gera um código nativo ideal ou próximo do ideal.

Java ME

No Java ME, vejamos os resultados para precisão **double**. O emulador do WTK 2.5 é 90 vezes mais lento que o HotSpot Server: duas ordens de magnitude pior, apesar de ambos rodarem no mesmo hardware. Esta diferença pode ter duas causas. Primeiro, o HotSpot Server é o que há de melhor em JVMs, gerando código nativo e acelerado por literalmente centenas de otimizações. Em especial, o HotSpot Server⁴ é capaz de explorar as instruções aritméticas MMX/SSE disponíveis no Pentium, entre outras otimizações críticas para o desempenho de cálculos aritméticos. Outro motivo de desvantagem do emulador é que este

Runtime utilizado	Precisão	Tempo (176x176 pixels)
Referência (Java SE 6.0, Server) PC com Pentium-D 3,4GHz	double	0,012s
	float	0,012s
	fixed	0,040s
Emulador do WTK 2.5 PC com Pentium-D 3,4GHz	double	1,125s
	float	0,984s
	fixed	1,046s
Dispositivo real: Sony Ericsson Z550i, ARM9 ~110MHz sem VFP ¹	double	13,588s
	float	7,041s
	fixed	1,266s

Tabela 1. Desempenho do MicroMandel, em várias plataformas e precisões numéricas.

² Se você conseguir uma taxa tão alta num programa Java ME (o que não é difícil com animações simples – sem cálculos matemáticos pesados!), pode limitá-la a um máximo de 30fps, pois o olho humano não percebe variações muito acima dessa frequência. Esta limitação pode ser obtida com timers, e é importante para poupar a CPU, e por consequência a bateria do dispositivo.

³ Apesar de a CPU do PC ser dual-core, isso não beneficia o programa MicroMandel, que não é multithreaded. (Mesmo que fosse, a versão Java ME não seria beneficiada, pois a KVM não é capaz de usar múltiplas CPUs.)

⁴ Assim como outras JVMs SE de topo, como o IBM JDK ou BEA JRockit.

possui um grande overhead devido às suas ferramentas, monitores, capacidades de depuração e profiling. (O HotSpot Server também tem estas capacidades, mas tem uma arquitetura mais avançada, capaz de eliminar praticamente todo o overhead associado a estas funcionalidades quando elas não estiverem sendo usadas.)

O meu telefone celular, que é um modelo recente (mas não topo de linha: ele me custou menos de R\$ 200,00 num plano pós-pago), teve um desempenho 12 vezes pior do que o emulador e cerca de 1.000 vezes pior que o HotSpot Server no PC. Este celular tem uma CPU de 110 MHz⁵: 30 vezes mais lento que o PC em ciclos por segundo. Mas o PC tem outras vantagens, como velocidade da memória, execução superescalar, predicação de desvios etc. Multiplicando os fatores, o celular deve ser no mínimo 100 vezes mais lento que o PC. Ajustando para esta diferença, um escore mil vezes inferior ao do PC se traduz numa JVM cerca de dez vezes menos eficiente.

Parte da desvantagem que resta (após descontada a diferença de hardware) certamente é devida ao compilador JIT muito superior do HotSpot Server. Mesmo assim, ainda achei a diferença alta. Devemos ter esquecido algum fator importante. Não será a implementação de aritmética em ponto flutuante? O Pentium possui uma FPU (unidade de ponto flutuante) que faz estes cálculos em hardware, mas a maioria das CPUs para dispositivos limitados não oferece esta capacidade, obrigando os programas a usar uma biblioteca de emulação que implementa por software a aritmética de ponto flutuante IEEE-754⁶.

Para conferir, escrevi uma segunda versão do método de cálculo (disponível no download) que usa **float** ao invés de **double**. Como podemos ver na Tabela 1, os cálculos

⁵ Estimado pelo programa JBenchmark ACE (www.jbenchmark.com). Os fabricantes destes dispositivos raramente publicam especificações com detalhes como modelo e velocidade de CPU. A tabela de resultados já publicada no site do JBenchmark é uma boa referência para pesquisar antes de comprar o seu próximo dispositivo Java ME.

⁶ Norma seguida pelo Java (e maioria das linguagens e CPUs) que especifica a representação binária de números em ponto flutuante, como os **double** e **float** do Java, e suas operações fundamentais. Ver "Matemática em Java", Edição 19.

com **float** são quase duas vezes mais rápidos que com **double** no celular. Este é o resultado esperado em sistemas sem unidade de ponto flutuante: as implementações em software de operações como multiplicação executam num tempo proporcional ao tamanho dos tipos de dados. Já no PC com o HotSpot Server, esta diferença não existe: **float** tem o mesmo desempenho de **double**. No emulador há uma pequena vantagem do **float**, mas isso é típico de JVMs menos otimizadas (por exemplo, usam duas instruções de 32 bits ao invés de uma de 64 bits para fazer uma atribuição de **double**; não usam MMX/SSE etc.).

Para a prova final, escrevi outra variante

do cálculo que utiliza somente aritmética de inteiros (mais precisamente, de "ponto fixo", que simulam aritmética de ponto flutuante com inteiros divididos logicamente em uma parte inteira e uma parte fracionária⁷). No emulador rodando no PC houve pouca vantagem de desempenho – o desempenho com números **fixed** é

⁷ No MicroMandel, usamos um **int** de 32 bits organizado como 8 bits inteiro + 24 bits fracionário. Isso permite representar números de -128 a +127, com uma precisão decimal de ~7 dígitos, suficiente para gerar o fractal de Mandelbrot (pelo menos com as configurações default). Esta artimanha aritmética tem pouca relação com o assunto de Java ME, mas o leitor curioso encontrará o código bem documentado no download do artigo.

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

COBREBEMX

-  56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
-  GERAÇÃO DE BOLETOS ON LINE;
-  GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
-  MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM WWW.COBREBEM.COM


Tecnologia

até um pouco mais lento que **float**, pois o código de ponto fixo é mais complexo. No HotSpot Server, o **fixed** é bem mais lento, provavelmente porque não se beneficia das instruções MMX/SSE, orientadas a cálculo em ponto flutuante. Já no celular, a vantagem foi enorme – 10,7 vezes mais rápido que a versão com ponto flutuante! Tão bom que ficou no mesmo patamar que o emulador: somente 21% mais lento. E a diferença entre a KVM do celular e o HotSpot Server no PC caiu para somente 100 vezes comparando com **double/float**, ou apenas 30 vezes comparando com o **fixed**. Isso coincide precisamente com a nossa estimativa de diferença de desempenho dos hardwares, na faixa de 30X a 100X.

Podemos tirar daí algumas conclusões:

- A KVM, ainda que tenha um compilador JIT modesto em comparação com as JVMs para Java SE, é realmente capaz de produzir desempenho similar ao de aplicações nativas no dispositivo.

- Infelizmente, o meu celular não possui FPU, o que é a regra em CPUs ARM9 (embutidos em vários celulares disponíveis no Brasil). Os chips ARM mais recentes, geralmente ARM11, costumam possuir as instruções VFP (Vector Floating Point). Mas estes chips serão mais comumente encontrados em dispositivos maiores e de maior custo, como Smartphones e PDAs.

- Os velhos tempos estão de volta. Quem, como eu, começou a programar em micros de 8 bits de 64 Kb de memória total, e há anos achava que nunca mais precisaria de técnicas extremas de otimização de código (como o uso de ponto fixo, ou mesmo de selecionar os tipos de dados de menor precisão capazes de implementar um algoritmo), terá uma sensação de “*déjà vu*”... É hora de desenferrujar a capacidade de programar de forma muito eficiente, utilizando recursos de forma espartana, pois isso poderá significar a diferença entre uma aplicação espetacular e uma medíocre – ou simplesmente, entre uma aplicação que consegue ser instalada na maioria dos

dispositivos, e outra que simplesmente gera erros de “JAR muito grande”.

Para aplicações de tempo real como jogos ou animações, é importante fazer testes com um desempenho aproximadamente igual ao dos dispositivos reais. O emulador do WTK tem a capacidade de emular a velocidade do dispositivo. Executando o seu utilitário *prefs* (pelo NetBeans: *Java Platform Manager>Tools and Extensions>Open Preferences*), vá na página *Performance*, habilite *Enable VM speed emulation* e selecione um valor para *VM speed* que produza um resultado semelhante ao do seu dispositivo real. Para o meu dispositivo e o teste do MicroMandel, configurando 700 bytecodes/milissegundo, obtive um tempo de execução idêntico ao do celular.

Esta emulação de desempenho é bastante aproximada, pois só cria um limite para o número de bytecodes executados por unidade de tempo, desconsiderando as variações de desempenho entre diferentes bytecodes, as transformações de código feitas pelo compilador JIT e outros fatores. Mas já ajuda muito, especialmente para aplicações “real-time” como jogos. Será frustrante passar um mês trabalhando naquela aplicação revolucionária, que parece muito boa num emulador rodando a todo gás, e na hora de testar em dispositivos reais (o que acabamos fazendo pouco porque dá mais trabalho), vemos que o desempenho é péssimo.

Finalmente, os SDKs proprietários de alguns fornecedores de dispositivos incluem facilidades para emulação no dispositivo. Isso pode exigir algum hardware não incluso na aquisição dos dispositivos, como um cabo especial ligado à porta serial do PC, e pode exigir plug-ins de IDE e versões customizadas do WTK. Mas vale a pena, porque realizar o processo de *deploy-teste-debug* direto no dispositivo poderá aumentar a sua produtividade.

Conclusões

Sabemos que dispositivos Java ME, como celulares e PDAs possuem um desempenho muito inferior ao de PCs ou servidores,

mas temos que lembrar que estes últimos têm performance realmente impressionante. Às vezes perdemos a noção de como as CPUs modernas são rápidas, pois temos que usá-las para executar softwares cada vez mais “gordos”, que consomem qualquer capacidade de processamento disponível. Dito isto, ter 1/30 ou 1/100 desta capacidade de processamento numa maquininha que cabe no seu bolso e capaz de trabalhar horas com uma carga de bateria, é outra maravilha da tecnologia.

O software escrito para tais plataformas precisa compensar essa diferença de velocidade de processamento, bem como outras, como capacidade de armazenagem ou de transmissão de dados. Isso valeria para qualquer linguagem ou toolkit de desenvolvimento para dispositivos móveis, mas podemos verificar que a plataforma Java ME apresenta um desempenho muito satisfatório, e que é possível produzir aplicações Java ME com ótimo desempenho e funcionalidade. Isso também exige escrever código extremamente enxuto, uma arte que parece andar um pouco esquecida até entre programadores de sistemas operacionais – mas que volta com todo o gás nos dispositivos “micro”. ●

java.sun.com/javame

Página oficial do Java ME.

proguard.sourceforge.net

ProGuard, o ofuscador de bytecode livre, preferido de três entre três IDEs/extensões para Java ME cobertos nesta série. Também útil para aplicações Java SE/EE que desejem dificultar a ação de larâpios ou crackers, ou que também tenham necessidade de reduzir o tamanho dos seus JARs (ex.: para uso com Web Start).

community.java.net/mobileandembedded

Projeto phoneME (implementação Java ME de código aberto da Sun, incluindo o HotSpot CLDC). O leitor avançado não pode perder os blogs de engenheiros da Sun, como Mark Lam.

www.jbenchmark.com

Benchmarks de VMs Java ME. Inclui versões gratuitas que vale muito a pena consultar antes de fazer um investimento no seu próximo dispositivo de última geração. Os dispositivos mais caros nem sempre são os de melhor desempenho!



Globalcode®

THE DEVELOPERS COMPANY

Vamos entregar no prazo. Vamos documentar os requisitos.
Vamos adotar UML. Vamos melhorar o processo de testes.

Se isso faz parte das suas promessas para 2007, conte com a Globalcode para cumprí-las!



Metodologia Requisitos Estimativas UML Padrões Testes

Curso de Análise de Requisitos ^{AN1}

A análise de requisitos é uma das atividades mais importantes do ciclo de desenvolvimento de software, mas às vezes é relegada ao segundo plano – com conseqüências graves sobre a qualidade e a eficácia do software desenvolvido. Uma das razões alegadas para a não-realização da análise de requisitos é a falta de praticidade, ou o sentimento de que há excesso de “burocracia” nas atividades envolvidas.

Foi por isto que a Globalcode criou este novo curso que procura remover essas barreiras, mostrando na prática técnicas modernas para a análise de requisitos, que permitem especificar de forma prática e eficaz as funcionalidades e características de um sistema.

Tópicos

- Conceitos e técnicas de levantamento de requisitos
- Casos de uso UML
- Documentos de caso de uso
- Unified Process
- Diagrama de atividades UML
- Pontos de caso de uso
- Estudo de caso

Próximos Lançamentos

Desenvolvimento de Testes para Softwares ^{AN2}

O treinamento mais completo sobre testes de software com ênfase na plataforma Java, abordará técnicas, metodologias e ferramentas para testes funcionais e não-funcionais, como stress-testing.

Carga Horária: 32 horas
Lançamento: março / 2007

Análise e modelagem de softwares com UML ^{AN3}

Um curso específico sobre o padrão mais utilizado para especificação e modelagem de softwares, em todos seus ciclos. Você aprenderá todos os diagramas UML e seus componentes nas versões 1.5 e 2.0.

Carga horária: 40 horas
Lançamento: abril / 2007

WWW.GLOBALCODE.COM.BR

0800 - 704 - 5282

Uma Aplicação Java EE

Parte 4: Escalabilidade e Flexibilidade com

Neste último artigo da série de desenvolvimento com Java EE, vamos concluir a aplicação de controle de matrículas que temos construído nas últimas três edições. Até o momento, desenvolvemos a camada de negócios e de persistência com JPA, criamos a camada de componentes web com JSF e incrementamos a interface web com AJAX.

Agora introduziremos a tecnologia EJB 3 para fornecer mais robustez à aplicação. Os requisitos não-funcionais do projeto deixam claro que não deve existir uma dependência forte com essa tecnologia, para que possamos executar a aplicação em duas arquiteturas. Relembramos aqui os requisitos relevantes para esta parte:

- *A aplicação deverá ser facilmente escalável, para que depois possa acomodar os componentes de negócio em um container à parte.*
- *No futuro, a aplicação poderá ser fornecida como um framework a diversas instituições educacionais, que podem ou não utilizar um*

container de componentes de negócio em sua infra-estrutura.

Ao adaptar a aplicação ao uso do EJB 3.0, vamos considerar estratégias de desenvolvimento para um alto nível de reaproveitamento de código. Aplicaremos também alguns design patterns que promoverão a independência solicitada.

Decisões de projeto

Vamos começar discutindo algumas alternativas de projetos para atender os requisitos restantes.

Escalabilidade

A escalabilidade é a capacidade de um sistema de atender um volume crescente de carga de processamento sem comprometer

seriamente a performance ou o tempo de resposta para os usuários. A versão atual da nossa aplicação funciona em um container web, conforme a **Figura 1**. Essa arquitetura é totalmente funcional, mas atualmente concentra todo o processamento em uma única JVM. Se o volume de trabalho sobre o servidor aumentar e quisermos manter um tempo de resposta aceitável para os usuários, temos algumas possibilidades:

- *Aumentar a capacidade do servidor, adicionando mais memória e mais processadores.* Neste caso, estaríamos limitados às possibilidades de expansão do servidor.
- *Criar um cluster de containers web coordenado por um balanceador de carga (por exemplo, utilizando três servidores, teríamos um balanceador de carga e dois servidores para os containers web).* Com essa solução, no entanto, acabaríamos prendendo a arquitetura a algum produto específico, pois o balanceamento de carga de containers web ainda não é padronizado pela especificação Java EE.

Completa

EJB 3.0

Aprenda os fundamentos do desenvolvimento de componentes de negócio EJB 3.0, com técnicas de programação e design patterns que aumentam o reaproveitamento dos componentes em diversas arquiteturas

RENATO BELLIA

• *Distribuir o processamento em camadas físicas de responsabilidade.* Uma abordagem totalmente coerente com a especificação Java EE é alocar um container para processamento web e um segundo container para o processamento de regras de negócio. Vamos seguir esta direção para proporcionar mais escalabilidade à aplicação.

Queremos poder distribuir uma parte da carga de processamento para um container de componentes de negócio, que funciona numa outra JVM em outro servidor corporativo. Há muito tempo, a especificação Java EE propõe a utilização de componentes de negócio distribuídos, capazes de funcionar em containers especializados e independentes da tecnologia de apresentação utilizada. Esses são os conhecidos componentes EJB ou Enterprise JavaBeans.

Um componente EJB pode oferecer uma interface de acesso remoto, através da qual clientes da rede local invocam os métodos de negócio disponíveis no componente, externamente ao servidor. O uso de EJBs com interface remota nos permitirá implantar a arquitetura da **Figura 2**.

Mas não é só a questão da distribuição de processamento que deve ser levada em conta para adotar o uso de EJBs. No **quadro** "EJBs de sessão" apresentamos conceitos básicos e discutimos outras vantagens dessa tecnologia.

Equivalência de regras de negócio

Os requisitos da nossa aplicação exigiram que começássemos criando uma versão web e que só depois criássemos uma versão mais escalável (baseada em containers de negócio). Isso nos leva a conviver com as duas arquiteturas mostradas nas **Figuras 1 e 2**.

Os componentes de negócio devem ser equivalentes em ambas as arquiteturas.

Ou seja, as regras de negócio implementadas nas classes comuns do pacote **jm.matriculas.business.impl.comum** devem ser implementadas de modo equivalente nos Session Beans que iremos criar. Vamos estudar três abordagens para resolver a questão da equivalência das regras de

negócio entre as duas arquiteturas.

Abordagem 1: Famílias paralelas de componentes

A **Figura 3** ilustra uma primeira abordagem, que consiste em manter (trabalhosamente) duas famílias de componentes, uma para cada arquitetura. Neste caso, qualquer

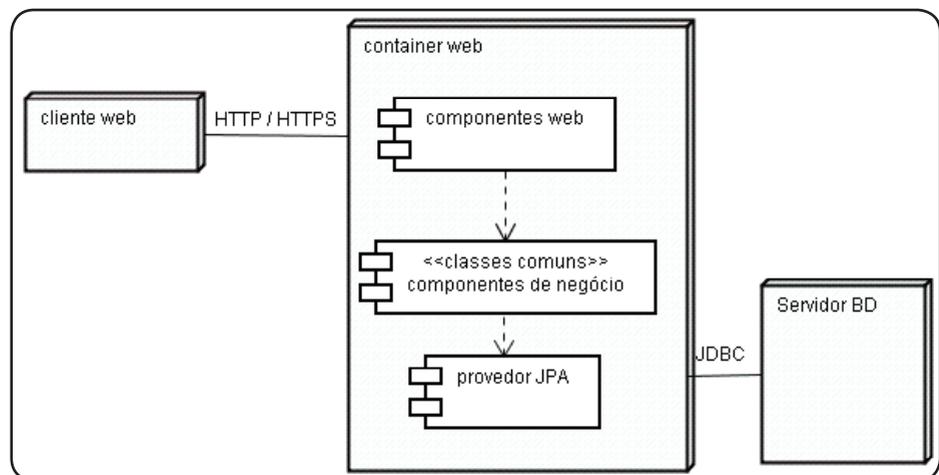


Figura 1. Arquitetura 1: implementação em container web com classes comuns e JPA.

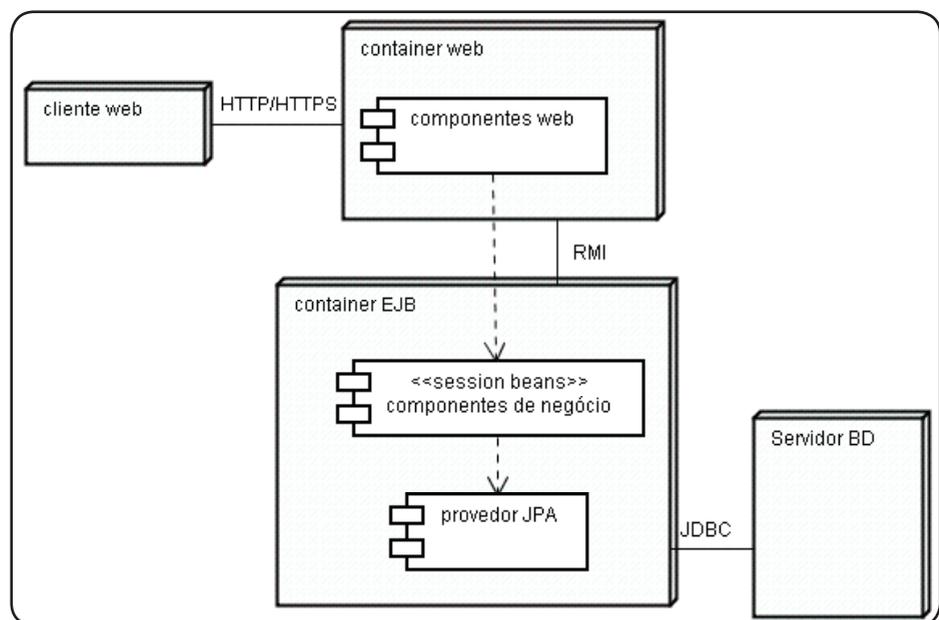


Figura 2. Arquitetura 2: componentes web + componentes EJB 3 + JPA.

EJBs de sessão

Os EJBs, ou Enterprise JavaBeans, são componentes de negócio que ficam hospedados em um container de EJBs. Um dos tipos de componentes definidos pela especificação EJB é o *Session Bean*, ou “EJB de sessão”.

Finalidade

Um Session Bean tem como finalidade representar uma sessão interativa de um processo de negócios, do ponto de vista de um cliente da

aplicação. Por exemplo, um processo de controle de contas a receber pode ser disponibilizado para um cliente através de um Session Bean.

Cada método de uma classe de Session Bean que pode ser acessado por um cliente é chamado de *método de negócio*.

Existem dois tipos de Session Beans:

- O tipo *Stateless*, mais leve, é indicado para processos que não necessitam manter estado (valores de atributos) entre chamadas de métodos.

- O tipo *Stateful*, mais pesado, garante a preservação de estado entre chamadas de métodos de um mesmo cliente.

Vantagens

Por que usar Session Beans no lugar de classes Java comuns? Há diversas vantagens, que são oferecidas pelo container de EJBs.

Sendo hospedado em um container de EJBs, um Session Bean pode funcionar como um componente de negócio distribuído, que disponibiliza o mesmo processo e regras de negócio para vários tipos de clientes. Existem três formas de acessar um Session Bean:

- Através de uma interface remota, para clientes da rede local.
- Através de uma interface local, para clientes localizados na mesma JVM onde está rodando o container de EJBs.
- Através de uma interface *Service End Point*, para clientes que visualizam o Session Bean como um web service.

Além disso, o container oferece uma série de serviços de infra-estrutura, como:

- Controle de segurança e transacional em nível de métodos.
- Gerenciamento de ciclo de vida de instâncias de EJBs, proporcionando escalabilidade através do uso racional da memória do servidor.
- APIs de acesso a recursos corporativos (JNDI, JMS, JCA etc.).
- API de temporização (Timer).

Com toda essa infra-estrutura disponível, o programador da classe de Session Bean pode ficar focado nas regras de negócio, e se desprende de várias responsabilidades técnicas da construção de uma aplicação corporativa robusta.

Desenvolvendo EJBs de sessão

Para que possa assumir as responsabilidades oferecidas como infra-estrutura, o container de EJBs deve mediar todas as chamadas a métodos de negócio de um componente EJB. Conseqüentemente, uma instância de um Session Bean não pode ser manipulada diretamente por um cliente. Um cliente manipula uma “representação” do objeto session bean – um *proxy* – através de uma interface de acesso (local, remota ou *service end*

Listagem Q1. Interface de acesso e implementação de um Session Bean

ManipuladorNumeros

```
package jm.exemplo.ejb;

public interface ManipuladorNumeros {
    public void addNumero(int i);
    public int getMaiorNumero();
    public double getMedia();
    public void resetNumeros();
}
```

ManipuladorNumerosBean

```
package jm.exemplo.ejb;

import java.util.ArrayList;
import javax.annotation.PostConstruct;
import javax.ejb.*;

@Stateful @Remote(ManipuladorNumeros.class)
public class ManipuladorNumerosBean
    implements ManipuladorNumeros
{
    private ArrayList<Integer> numeros;

    public void addNumero(int i) {
        numeros.add(i);
    }

    public int getMaiorNumero() {
        if (numeros == null || numeros.size() == 0) return 0;
        int maior = Integer.MIN_VALUE;
        for(Integer n : numeros) {
            if (n > maior) maior = n;
        }
        return maior;
    }

    public double getMedia() {
        if (numeros == null || numeros.size() == 0) return 0;
        double media = 0;
        for(Integer n : numeros) {
            media += n;
        }
        return media / numeros.size();
    }

    @PostConstruct
    public void resetNumeros() {
        numeros = new ArrayList<Integer>();
    }
}
```

point). Acompanhe o processo passo a passo:

1. O cliente aciona um método da interface de acesso, no proxy.
2. O proxy se comunica com o container, solicitando a invocação do método de negócio correspondente em um objeto Session Bean.
3. O container, nesse momento, cuida de serviços como controle de segurança, controle transacional (iniciando uma transação) e gerenciamento de escalabilidade. (Veja a **Figura Q1**.)
4. O método é invocado no Session Bean.
5. Antes de retornar para o proxy solicitante, o container pode concluir a transação (fazendo commit ou rollback).
6. O proxy recebe o retorno e o repassa para o cliente.

É responsabilidade do desenvolvedor disponibilizar as interfaces de acesso de um Session Bean, além da classe Session Bean propriamente dita (que implementa o bean).

Existem algumas regras que devem ser seguidas pelo “hóspede”, o Session Bean, para não prejudicar a qualidade da infra-estrutura oferecida pelo “hospedeiro” (o container de EJBs). Algumas delas:

- Não abrir ou manipular threads.
- Não abrir servidores de sockets.
- Não manipular o sistema de arquivos local do container.

Exemplo

Apresentamos na **Listagem Q1** uma interface de acesso, e a classe do Session Bean que mantém estado (**@Stateful**), seguindo a especificação EJB 3. A interface é registrada como uma interface remota (**@Remote**). O último método do Session Bean é marcado como o método de inicialização (**@PostConstruct**), que deve ser executado pelo container quando uma instância é requisitada por um cliente.

Compactamos a interface e a classe em um módulo EJB – um arquivo JAR – e implantamos no container EJB. Apresentamos na **Listagem Q2** o cliente remoto.

O container deve fornecer um “catálogo” de Session Beans hospedados, para que os clientes possam localizá-los através de um *serviço de nomes*. No JBoss, por default, um Session Bean com interface remota é registrado no catálogo de nomes seguindo o modelo “nome-simplificado-da-classe/

remote” – neste exemplo o bean será registrado como “ManipuladorNumerosBean/remote”.

Temos que nos conectar ao serviço de nomes do container através de um objeto **InitialContext** da API JNDI (Java Naming and Directory Interfa-

ce). Solicitamos um Session Bean ao container via o método **lookup()**. Lembrando que o que recebemos, na verdade, é um proxy (um objeto que representa o componente), gerado dinamicamente pelo container.

Listagem Q2. Cliente remoto para o Session Bean

```
package jm.exemplo.client;

import java.util.Hashtable;
import javax.naming.*;
import jm.exemplo.ejb.ManipuladorNumeros;

public class ClienteSessionBean {
    public static void main(String[] args) throws NamingException {

        // configurando o serviço de nomes do JBoss
        Hashtable hashtable = new Hashtable();
        hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
            "org.jboss.naming.NamingContextFactory");
        hashtable.put(Context.PROVIDER_URL, "localhost:1099");

        // conectando ao serviço de nomes
        InitialContext initialContext = new InitialContext(hashtable);

        // obtendo o proxy
        ManipuladorNumeros manipuladorNumeros =
            (ManipuladorNumeros) initialContext.lookup(
                "ManipuladorNumerosBean/remote");

        // usando a funcionalidade do EJB
        manipuladorNumeros.addNumero(14);
        manipuladorNumeros.addNumero(56);
        manipuladorNumeros.addNumero(-11);
        System.out.println("maior " + manipuladorNumeros.getMaiorNumero());
        System.out.println("media " + manipuladorNumeros.getMedia());
        manipuladorNumeros.resetNumeros();
        System.out.println("maior " + manipuladorNumeros.getMaiorNumero());
    }
}
```

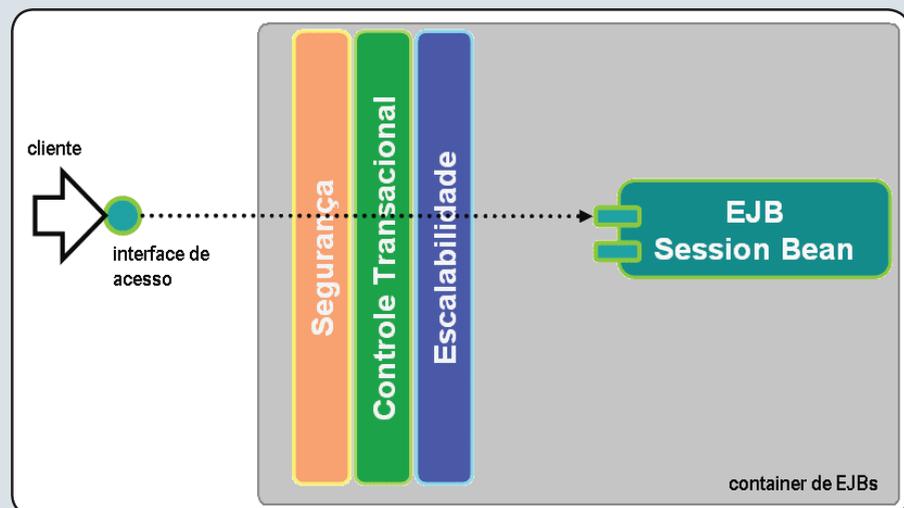


Figura Q1. Intermediação do container de EJBs.

alteração de regras de negócio vai exigir esforço duplicado de manutenção e de testes nas duas famílias de componentes.

Essa abordagem é frágil e não reaproveita os componentes criados até aqui (ou seja, os da Arquitetura 1). Deveria ser considerada no caso de não existir controle sobre o código-fonte das classes comuns de negócio. Também seria uma abordagem válida se essas classes realizassem operações proibidas dentro de um container EJB, ou se usassem recursos transacionais (como conexões com bancos de dados) de maneira inadequada.

Abordagem 2: Reutilização por dependência

Numa segunda abordagem, poderíamos criar componentes EJB de sessão encapsulando as classes da Arquitetura 1, como mostrado na **Figura 4**. Basicamente, os métodos de negócio das

classes comuns seriam invocados pelos métodos disponibilizados pelos Session Beans através de suas interfaces remotas de acesso.

Com essa abordagem, uma alteração nas regras de negócio iria concentrar o esforço de manutenção nos componentes da Arquitetura 1. Note que estamos utilizando o design pattern Java EE conhecido como *Session Façade*. Seguindo este pattern, os Session Beans funcionam como uma fachada (*façade*) baseada na tecnologia EJB, que coordena o acesso aos verdadeiros componentes de negócio.

Abordagem 3: Reutilização por herança

É possível utilizar herança para propagar os comportamentos dos componentes comuns da Arquitetura 1 para os componentes EJB da Arquitetura 2, conforme vemos na **Figura 5**. Como acontece na abordagem

anterior, alterações nas regras de negócio implicam manutenção concentrada nas classes comuns da Arquitetura 1.

Esta é a abordagem que exige menos esforço, mas ela só deve ser considerada quando temos controle sobre o código-fonte das classes comuns de negócio. Isso porque precisamos garantir que todos os parâmetros e retornos dos métodos sejam compatíveis com uma interface remota (mais especificamente, eles devem ser serializáveis). Vamos adotar essa terceira abordagem em nosso projeto.

Componentes multi-arquitetura

No intuito de conviver com os dois cenários (com e sem componentes EJB), o ideal é que os componentes web desenvolvidos sejam reutilizados sem modificação de código-fonte, quando alternarmos entre as Arquiteturas 1 e 2. Para que isso seja

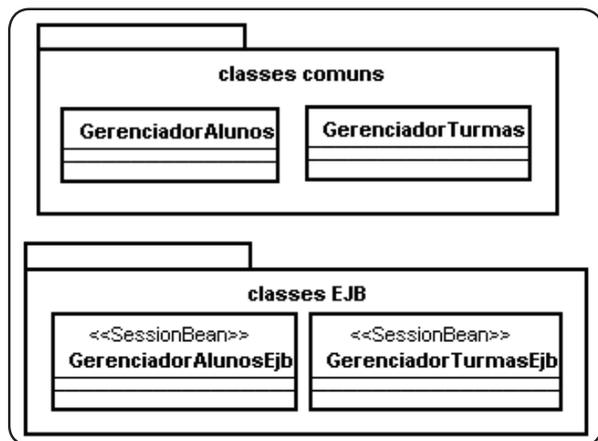


Figura 3. Famílias de componentes paralelas e independentes.

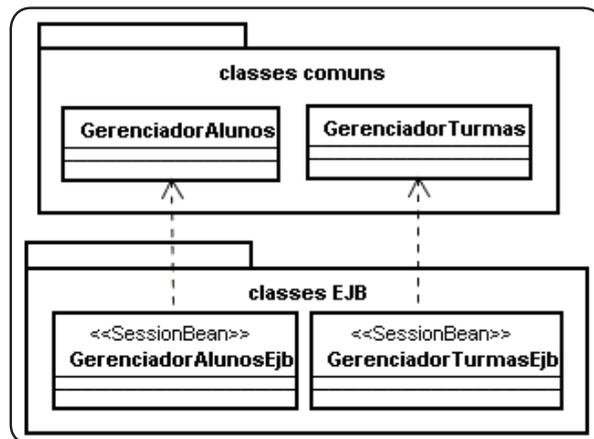


Figura 4. Reutilização por dependência (pattern Session Façade).

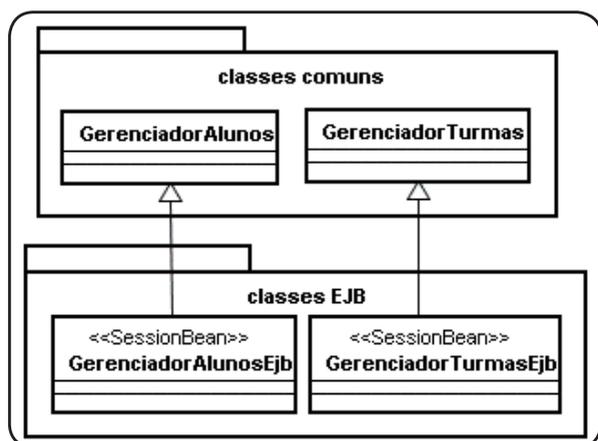


Figura 5. Reutilização por herança.

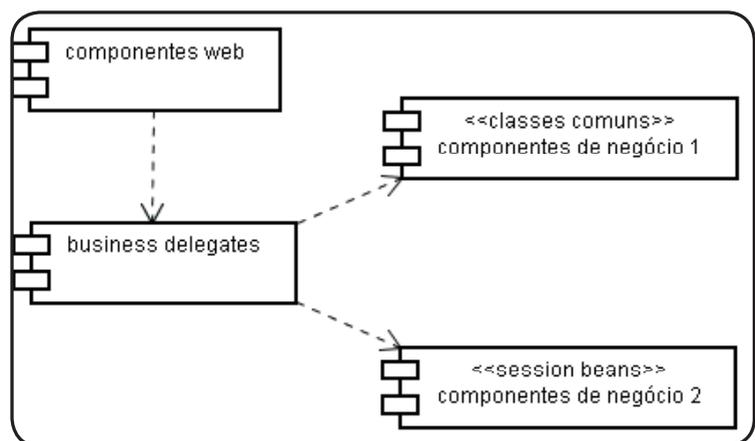


Figura 6. Business Delegate para duas arquiteturas.

possível, os detalhes de comunicação remota com os EJBs de sessão da Arquitetura 2 devem ser resolvidos fora dos componentes web.

Business Delegate

A solução mais praticada nessa situação é proposta pelo design pattern Java EE *Business Delegate*. Dentre outros benefícios, um Business Delegate encapsula os detalhes de utilização dos componentes de negócio, desacoplando os clientes (no caso, os componentes web) das APIs e exceções típicas na manipulação de EJBs ou de outras tecnologias.

☞ *Embora inicialmente proposto para reduzir o acoplamento de clientes de EJBs o Business Delegate pode ser perfeitamente considerado para clientes de outras tecnologias como web services, CORBA etc.*

Podemos pensar em criar uma classe Business Delegate para cada componente de negócio. E para cada método de negócio do componente haverá um método similar na classe Business Delegate (que passaremos a chamar de “delegate” para simplificar).

O método do delegate pode ter uma assinatura diferente do método de negócio para ocultar algum detalhe de tecnologia, principalmente exceções. Cada método de um delegate tem a função de invocar o método de negócios real, adequando parâmetros, convertendo exceções e convertendo tipos de retorno quando necessário.

Para promover a reutilização dos componentes web nas duas arquiteturas apresentadas, poderíamos ocultar dentro dos delegates a utilização dos componentes de negócio, de acordo com a arquitetura vigente – veja a **Figura 6**.

☞ *Um parâmetro de configuração pode ser disponibilizado para os componentes web, por exemplo através do descritor web.xml, e transmitido para os delegates, indicando qual família de componentes de negócio deve ser utilizada.*

Neste cenário de convivência com múltiplas arquiteturas o delegate vai encapsular, em cada método delegado, uma estrutura **if-else**. Essa estrutura

Listagem 1. Interfaces e implementações de gerenciamento de alunos

```
IGerenciadorAlunos
-----
// ... package e imports
public interface IGerenciadorAlunos {
    public abstract Integer salvar(Aluno aluno)
        throws ControleMatriculaException;
    public abstract void excluir(Aluno aluno)
        throws ControleMatriculaException;
    public abstract Aluno getById(Integer id)
        throws ControleMatriculaException;
    public abstract Aluno getByCpf(String cpf)
        throws ControleMatriculaException;
    public abstract List<Aluno> getByNome(String nome)
        throws ControleMatriculaException;
}

GerenciadorAlunos
-----
// ... package e imports
public class GerenciadorAlunos implements IGerenciadorAlunos {
    private EntityManager entityManager = null;

    @PersistenceContext
    public void setEntityManager(EntityManager em) {
        this.entityManager = em;
    }

    public Integer salvar(Aluno aluno)
        throws ControleMatriculaException
    {
        if (aluno.getNome() == null
            || aluno.getEmail() == null
            || aluno.getTelefone() == null)
        {
            throw new ControleMatriculaException(
                "Dados de aluno incompletos");
        }
        if (aluno.getId() == null) {
            entityManager.persist(aluno);
        } else {
            entityManager.merge(aluno);
        }
        return aluno.getId();
    }
    // ... demais métodos de negócio
}

GerenciadorAlunosEjb3
-----
// ... package e imports
@Stateless
@Remote(IGerenciadorAlunos.class)
public class GerenciadorAlunosEjb3 extends GerenciadorAlunos {
}
```

decidirá qual família de componentes deverá ser invocada. Entretanto, se uma terceira arquitetura viesse a ser requisitada (por exemplo, uma baseada em Session Beans de interfaces locais, ou em web services), teríamos que visitar todos os métodos delegados e ampliar a estrutura **if-else**. Também teríamos o problema da quantidade de APIs combinadas dentro das classes delegate – por exemplo, um bloco **if**, manipulando a API JNDI e interfaces de acesso para EJBs, adjacente a um bloco **else** manipulando APIs como JAX-RPC ou JAX-WS

para acessar web services.

Analisaremos, então, uma alternativa melhor, que garante a reutilização dos componentes web nas duas arquiteturas propostas, e que acomodará melhor futuras necessidades de outras arquiteturas.

Interfaces de negócio

Vamos nos valer de uma melhoria introduzida pela especificação EJB 3.0 – a manipulação de interfaces de acesso a Session Beans remotos não mais requer o tratamento de exceções do tipo **RemoteException**.

☞ Anteriormente à especificação EJB 3.0, os métodos de negócio disponíveis na interface de acesso de um *Session Bean* remoto obrigatoriamente deveriam lançar a exceção **RemoteException**. Basicamente, uma **RemoteException** transportava para o cliente remoto uma exceção do tipo **RuntimeException** ocorrida durante a execução de um método de negócios dentro do container. Quando trabalhamos com EJB 2.x, o tratamento dessas exceções normalmente é feito no *Business Delegate*.

No quadro “EJBs de sessão”, vimos que os *Session Beans* invariavelmente necessitam de uma interface de acesso. Como há controle sobre todo o código produzido, nada impede o projetista de utilizar as interfaces de acesso dos *Session Beans* como interfaces de negócio para as duas arquiteturas. Dessa forma, os componentes web precisam apenas obter as implementações das interfaces de negócio, de acordo com a arquitetura utilizada – veja a **Figura 7**.

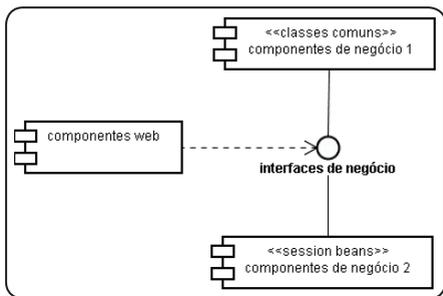


Figura 7. Uso de interfaces de negócio.

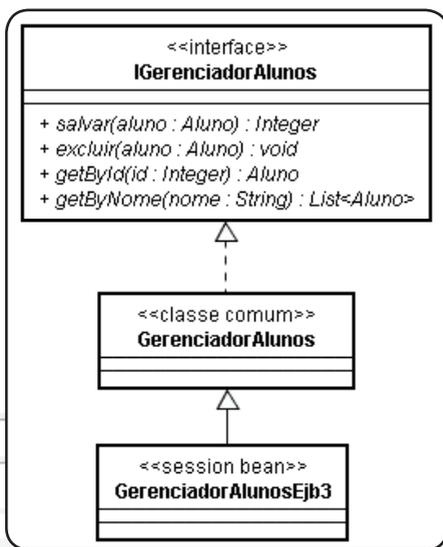


Figura 8. Combinando a reutilização por herança com interfaces de negócio.

Como reaproveitamos os componentes de negócio por herança, temos que uma interface definida para um componente da Arquitetura 1 será definida (por herança) para o componente correspondente da Arquitetura 2. Isso pode ser observado no diagrama mostrado na **Figura 8**.

Vamos analisar o código fonte da **Listagem 1**. **IGerenciadorAlunos** é a interface de negócios, e **GerenciadorAlunos** é sua implementação para a Arquitetura 1. Repare nas declarações da classe **GerenciadorAlunosEjb3** para a Arquitetura 2: a anotação **javax.ejb.Stateless (@Stateless)** define a classe como um EJB de sessão sem estado, e a anotação **javax.ejb.Remote (@Remote)** especifica para o container EJB qual é a interface de acesso remoto.

Na classe Java comum (POJO) **GerenciadorAlunos**, encontramos o método **setEntityManager()**. Através deste método, o componente de negócios receberá um objeto **EntityManager** para que possa realizar as operações de persistência. No caso das classes comuns da Arquitetura 1 (**GerenciadorAlunos**), este **EntityManager** deverá ser fornecido manualmente pela nossa aplicação.

Já na versão EJB para a Arquitetura 2 (subclasse **GerenciadorAlunosEjb3**), ocorrerá o processo de “injeção de dependência”. Ou seja, o container EJB reconhecerá a anotação **@PersistenceContext** presente no método e automaticamente fornecerá (injetará) um objeto **EntityManager** como parâmetro desse método.

☞ A simplicidade introduzida na especificação EJB 3 salta aos olhos neste momento, para quem desenvolve componentes EJB 2.x. Não precisamos de descritores como *ejb-jar.xml*, nem de componentes *Home (EJBHome)* ou de interfaces de ciclo de vida (*SessionBean*).

Fabricando componentes de negócio

Para definir qual família de classes de negócio deverá ser utilizada, podemos contar com a ajuda de uma fábrica de componentes

de negócio. Como as duas famílias podem ser manipuladas pelas mesmas interfaces, podemos considerar a utilização do clássico design pattern *Abstract Factory*.

Cada família de componentes será obtida por uma fábrica específica; portanto teremos duas fábricas. E para que os componentes web não fiquem acoplados a nenhuma das duas fábricas, criamos uma fábrica abstrata, com a capacidade de decidir pela fábrica de componentes que deve ser utilizada – veja a **Figura 9**.

O uso do pattern *Abstract Factory* permite acomodar melhor futuras famílias de componentes de negócio. Para cada nova família bastaria fornecer uma nova implementação de fábrica, e novas implementações das interfaces de negócios.

A **Listagem 2** nos mostra a **BusinessFactory** abstrata e as fábricas de componentes para arquitetura com classes comuns e a baseada em componentes EJB 3.0.

Componentização e controle de transações

O último requisito do nosso projeto define um cenário de longo prazo, em que outras instituições poderiam se basear nesta aplicação para construir novas aplicações de gerenciamento educacional. Os componentes de negócio projetados para os requisitos definidos no início do projeto poderiam ser combinados para suportar processos não previstos até o momento.

Imagine um processo de transferência de matrícula entre turmas, no qual o aluno quer mudar da turma A para a turma B. Não basta alterar o identificador de turma no objeto da matrícula, pois a turma B precisa aceitar o aluno, sem que a quantidade de vagas seja ultrapassada. O processo de transferência pode ser suportado com o cancelamento da matrícula na turma A e a criação de uma nova matrícula na turma B. Este processo deve ocorrer dentro de uma única transação, pois se a turma B rejeitar a matrícula, o aluno deverá permanecer na

turma A (ou seja, seria feito um rollback). Existem duas soluções:

- Acrescentar um método de negócios de transferência em **GerenciadorMatricula**, que deve coordenar todo o processo.
- Acionar em seqüência, a partir de um cliente, os métodos **cancelar()** e **criar()**, já disponíveis em **GerenciadorMatricula**.

A primeira solução é muito prática, pois com ela a operação ficará contida na camada de negócios, dentro de uma transação. Se o requisito de transferência estivesse definido desde o início, poderíamos facilmente incluir o método adicional. Mas para atender a um requisito não definido previamente, como essa da transferência, os futuros desenvolvedores/mantenedores da aplicação precisariam ter acesso ao código-fonte dos componentes de negócio. Ou então teriam que entender toda a proposta de reaproveitamento dos componentes em duas arquiteturas, antes de criar novos componentes de negócio.

Pensando nos futuros usos da aplicação, a segunda solução – a de acionamento em seqüência – poderá ser mais apropriada, conforme vemos na **Figura 10**. Toda a seqüência de chamadas deve acontecer dentro de uma única transação. Isso significa que as transações não podem ser delimitadas pelos métodos de negócio **cancelar()** e **criar()** (e por todos os demais); elas devem ser delimitadas pelos componentes web. Isso já vinha sendo feito nas outras versões da aplicação, apresentadas nas partes anteriores da série.

Mas há a questão do container EJB, que por default controla a transação automaticamente. Uma chamada feita por um cliente externo ao container EJB, a um método de negócios de um Session Bean, pode ser isolada em uma transação. Tal comportamento impede, na Arquitetura 2, o suporte a novos processos pelo acionamento em seqüência dos métodos de negócio disponíveis. A solução é controlar, via programação, nos componentes web, as transações do container EJB através da interface **javax.transaction.UserTransaction**.

Veja a **Listagem 3**, onde o processo de transferência está contido em uma única transação, com exemplos para as Arquiteturas 1 e 2. Na classe **ClienteTransferencia**, um objeto

Listagem 2. Fábricas de componentes de negócio

BusinessFactory

```
package jm.matriculas.business;

public abstract class BusinessFactory {
    private static String businessFactoryClassName;
    public abstract IGerenciadorAlunos getGerenciadorAlunos();
    public abstract IGerenciadorTurmas getGerenciadorTurmas();
    public abstract IGerenciadorMatriculas getGerenciadorMatriculas();
    public abstract void beginTransaction();
    public abstract void commitTransaction(boolean releaseResources);
    public abstract void rollbackTransaction(boolean releaseResources);

    public static BusinessFactory getInstance() {
        try {
            Class _class = Class.forName(businessFactoryClassName);
            return (BusinessFactory) _class.newInstance();
        } catch (Exception e) {
            throw new IllegalArgumentException("businessFactoryClassName invalido");
        }
    }
    // ... getters e setters
}
```

CommonBusinessFactory

```
// ... package e imports
public class CommonBusinessFactory extends BusinessFactory {
    private static EntityManagerFactory entityManagerFactory;
    private EntityManager entityManager;

    public IGerenciadorAlunos getGerenciadorAlunos() {
        GerenciadorAlunos ga = new GerenciadorAlunos();
        ga.setEntityManager(this.entityManager);
        return ga;
    }

    public void beginTransaction() {
        if (entityManager == null || ! entityManager.isOpen())
            entityManager = entityManagerFactory.createEntityManager();
        if (! entityManager.getTransaction().isActive())
            entityManager.getTransaction().begin();
    }
    // ... demais métodos
}
```

Ejb3BusinessFactory

```
// ... package e imports
public class Ejb3BusinessFactory extends BusinessFactory {
    // ... constantes
    private static Properties jndiProperties;
    private InitialContext initCtx;
    private UserTransaction userTransaction;

    public IGerenciadorAlunos getGerenciadorAlunos() {
        try {
            String jndiName = jndiProperties.getProperty(JNDI_PROPERTY_GERENCIADOR_ALUNOS);
            IGerenciadorAlunos ga = (IGerenciadorAlunos) initCtx.lookup(jndiName);
            return ga;
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
    }

    public void beginTransaction() {
        try {
            if (initCtx == null) {
                initCtx = new InitialContext(jndiProperties);
                String jndiName = jndiProperties.getProperty(JNDI_PROPERTY_USER_TRANSACTION);
                userTransaction = (UserTransaction) initCtx.lookup(jndiName);
            }
            if (userTransaction.getStatus() == Status.STATUS_NO_TRANSACTION)
                userTransaction.begin();
        } catch (Exception e) {
            initCtx = null;
            throw new RuntimeException(e);
        }
    }
}
```

UserTransaction foi solicitado ao container EJB, para envolver as chamadas a métodos de negócio em uma única transação. O objeto **UserTransaction** representa uma transação dentro do container, e neste exemplo o cliente está forçando o container a obedecer ao controle manual de transações.

☞ O **InitialContext** utilizado para obter o **UserTransaction** deve localizar todos os componentes EJB de sessão cujos métodos serão envolvidos na mesma transação.

E como os componentes web poderão controlar as transações de negócio de for-

ma transparente, sem criar acoplamento com os tipos **javax.persistence.EntityTransaction** da JPA ou com **javax.transaction.UserTransaction** da JTA? Podemos utilizar as fábricas de componentes para manipulação da API de transações adequada, como vemos nos métodos **beginTransaction()**, **commitTransaction()** e **rollbackTransaction()**, das classes mostradas na **Listagem 2**.

Adotando o controle de transação externo, podemos afirmar que chegamos a um modelo robusto de componentes de negócio multi-arquitetura, que poderá com segurança funcionar como um framework para futuras aplicações.

Construção (build), implantação e testes

No quadro “Checklist” apresentamos todos os elementos necessários para preparar o ambiente deste projeto. O arquivo *script-ant/build.xml*, disponibilizado no site da Java Magazine junto com o código-fonte completo, é capaz de montar rapidamente arquivos de deployment para as duas arquiteturas estudadas. Além disso, o script Ant é capaz de fazer o deployment em um servidor JBoss local.

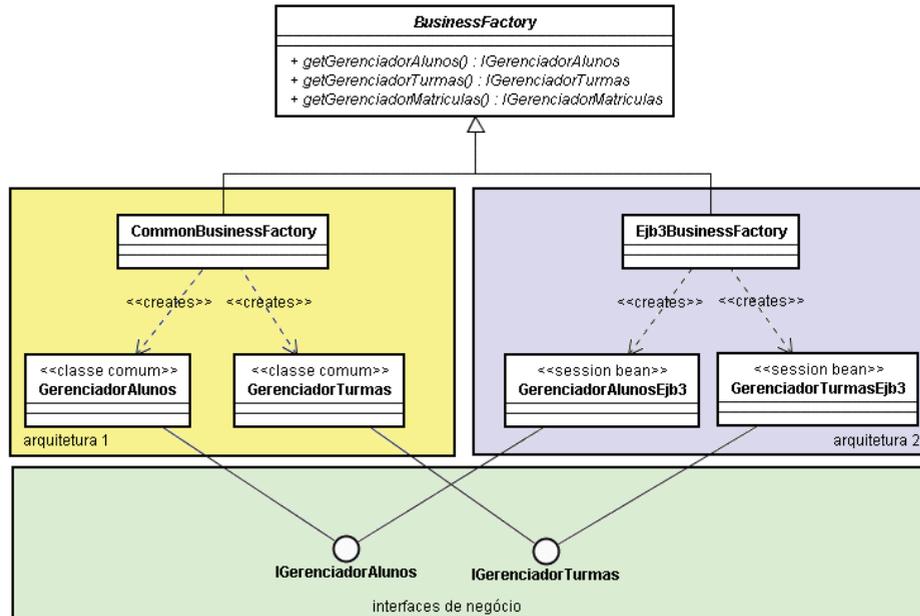


Figura 9. Fábricas de componentes de negócio.

Listagem 3. Controle manual de transações em EJBs – classe ClienteTransferencia

```
// ... package e imports
public class ClienteTransferencia {
    public static void main(String[] args) throws Exception {
        Hashtable hashtable = new Hashtable();
        hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
            "org.jboss.naming.NamingContextFactory");
        hashtable.put(Context.PROVIDER_URL, "localhost:1099");
        InitialContext initialContext = new InitialContext(hashtable);
        UserTransaction userTransaction =
            (UserTransaction) initialContext.lookup("UserTransaction");
        userTransaction.begin();

        try {
            IGerenciadorTurmas gt = (IGerenciadorTurmas)
                initialContext.lookup("GerenciadorTurmasEjb3/remote");
            IGerenciadorMatriculas gm = (IGerenciadorMatriculas)
                initialContext.lookup("GerenciadorMatriculasEjb3/remote");

            Turma t1 = gt.getById(new TurmaId("HTML", 1));
            Matricula m1 = t1.getMatriculas().iterator().next();
            gm.cancelar(m1);
            Matricula m2 = gm.criar(new TurmaId("HTML", 2), m1.getAluno());
            userTransaction.commit();
        }
        catch (Exception e) {
            userTransaction.rollback();
            throw e;
        }
        finally {
            initialContext.close();
        }
    }
}
```

☞ É interessante verificar o diretório etc do projeto, onde é possível encontrar arquivos de configurações importantes para as duas arquiteturas. Também vale a pena examinar os pacotes gerados no diretório archive. Este diretório é criado após a execução dos targets de build.

Comece executando o target **hsqldb.start** para inicializar o banco de dados.

Testando a Arquitetura 1

1. Realize um primeiro teste com o target **teste.business.common** para verificar o funcionamento dos componentes de negócio na Arquitetura 1. Em seguida, inicie o servidor de aplicações executando o target **jboss.start**.

2. Aguarde alguns instantes a inicialização completa do servidor e execute o target **deploy.war.arquitetura1**. Este target cria o arquivo *matriculas.war*, que é a aplicação web baseada em componentes de negócio comuns, e depois implanta esse WAR no servidor.

3. Você já poderá acessar a aplicação via browser com a URL *http://localhost:8080/matriculas*.

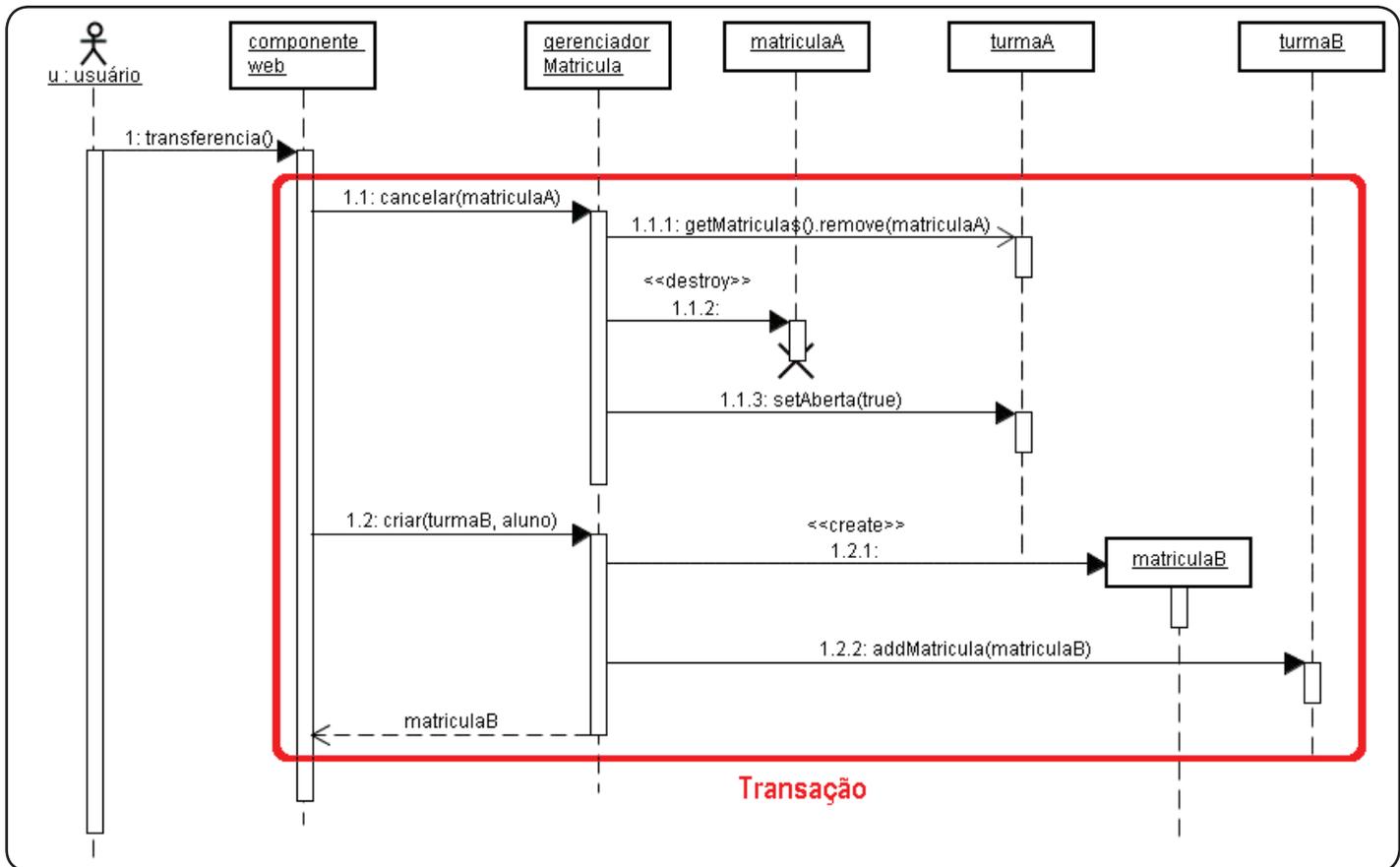


Figura 10. Diagrama de seqüência para transferência de matrículas.

RHEALEZA

INFORMÁTICA

Solicite uma apresentação ou visite-nos

- Consultorias e suporte especializados no desenvolvimento de softwares
- Outsourcing
- Consultoria e mentoring em CMMI
- Manutenção e suporte à infra-estrutura corporativa
- Treinamentos oficiais Borland

Trabalhando ao seu lado.



28
anos
RHEALEZA

oferecendo
as melhores soluções



Av. 7 de Setembro, 4698 . 15º andar . Batel . Curitiba . PR | (41) 2107-4550 | comercial@rhealeza.com.br | www.rhealeza.com.br

Testando a Arquitetura 2

1. Execute o target `undeploy.arquitetura1` para remover a aplicação web implantada anteriormente (a remoção é necessária porque a aplicação web da Arquitetura 1 não está configurada para trabalhar com os EJBs).

2. Execute o target `deploy.ejb.jar.arquitetura2`. Será criado e implantado o arquivo `matriculas-ejb.jar`, que contém os componentes de negócio na forma de EJBs de sessão. Este arquivo contém também as definições para o JBoss de um pool de conexões JDBC para a segunda arquitetura – o arquivo `matriculas-ds.xml`.

3. Execute um teste preliminar para esta arquitetura, disparando o target `teste.business.ejb3`.

4. Execute o target `deploy.war.arquitetura2`. Isso cria a aplicação web (`matriculas.war`) baseada em componentes EJB remotos.

5. Acesse novamente a aplicação via browser com a mesma URL (`http://localhost:8080/matriculas`).

Os testes que descreveremos para a Arquitetura 2 estão previstos para funcionar em um único servidor JBoss. Para testar a aplicação com containers em servidores

separados, são necessárias algumas configurações adicionais, que apresentamos a seguir.

Por comodidade, utilize a máquina onde os testes estão sendo feitos como o servidor hospedeiro de EJBs (digamos, com IP 192.168.1.73), e escolha uma outra máquina na rede local, também com o servidor JBoss, para hospedar os componentes web (digamos, 192.168.1.84).

1. Edite o arquivo `etc/matriculas.jndi.properties`, e na propriedade `java.naming.provider.url` informe o endereço IP do servidor de EJBs, por exemplo: `java.naming.provider.url=jnp://192.168.1.73:1099`

2. Execute o target `build.war.arquitetura2` para gerar o arquivo `archive/matriculas.war`, configurado para acessar o servidor de EJBs.

3. Copie `matriculas.war` para o servidor web escolhido. O arquivo deve ser copiado para a pasta `[instalação-do-jboss]/server/default/deploy` deste servidor.

4. Inicialize o servidor web (`[instalação-do-jboss]/bin/run.sh` ou `run.bat`) e acesse a aplicação web pelo browser, por exemplo usando `http://192.168.1.84:8080/matriculas`.



Checklist

Para experimentar com o projeto criado nesta série, será necessário configurar um ambiente com os softwares a seguir.

- **JDK 5.0 ou 6.0** - Você precisará de um Java Development Kit instalado e de ter a variável de ambiente `JAVA_HOME` configurada, apontado para o diretório de instalação do JDK.

- **Ant 1.6 ou superior** - O Ant será usado para compilar o projeto, instalá-lo no servidor e realizar várias outras operações.

- **JBoss Application Server** - O provedor JPA Hibernate Entity Manager, o banco de dados HSQLDB, a implementação do JSF Apache MyFaces e o Tomcat estão disponíveis no JBoss AS 4.0.5 (e em versões posteriores). Faça o download em labs.jboss.com/portal/jbossas/download. Nesta página localize a versão (estamos usando a 4.0.5 para esta série, mas versões mais recentes devem funcionar sem problemas), e escolha a opção *Download Installer*. O arquivo `jems-installer-1.2.0.jar` será obtido. Este instalador deve ser executado via linha de comando: `java -jar jems-installer-1.2.0.jar`. Durante o processo de instalação, é necessário escolher o profile "ejb3" (necessário apenas para a versão 4.0.5).

- **JBoss Ajax4Jsf e RichFaces** - Para incluir no projeto o uso da biblioteca

RichFaces, devemos fazer o download do arquivo `jboss-richfaces-3.0.0.zip`, disponível em labs.jboss.com/portal/jbossrichfaces/downloads. Realizado o download, descompacte o arquivo, pois ele será utilizado como referência no script Ant do projeto. Veja mais sobre estas bibliotecas na parte anterior (Edição 46).

Download e compilação do exemplo

O pacote de download para este artigo contém todo o código do projeto, além do script Ant para compilação e execução. Descompacte o ZIP e localize o `build.xml` no diretório `script_ant`. Utilize um editor de texto para configurar neste arquivo as seguintes propriedades:

- `jboss.home` - indicando o diretório onde o JBoss foi instalado.
- `richfaces.home` - indicando o diretório onde a distribuição da biblioteca RichFaces foi descompactada.

Faça um teste de compilação via linha de comando, digitando o comando `ant compile` a partir do diretório `script_ant`. Se o comando compilar todos os códigos sem gerar erros (warnings de compilação são esperados), isso significa que o ambiente está configurado corretamente.

Conclusões

Ao longo desta série, trabalhamos com um conjunto de requisitos realista do ponto de vista de funcionalidades, e desafiador da perspectiva de reaproveitamento de componentes de software. Também consideramos várias decisões de projeto e suas conseqüências. Pudemos ainda explorar as vantagens da API de persistência Java (JPA) e o modelo de programação JavaServer Faces, além de introduzir o uso de AJAX.

Nesta última parte, verificamos conceitos fundamentais da tecnologia EJB 3.0 e os seus benefícios. Estudamos o conceito de interfaces de acesso remoto e apresentamos os mecanismos de robustez para uma aplicação corporativa, garantidos por um container de EJBs. Vimos como tirar proveito das facilidades da tecnologia

EJB 3.0 para transformar componentes de negócio comuns em componentes Java EE. Verificamos também estratégias de desenvolvimento para alternar rapidamente entre duas arquiteturas, reaproveitando os componentes Java EE e gerenciando transparentemente as transações, nos dois cenários.

Esperamos poder ter acelerado o proces-

so de assimilação desse poderoso conjunto de tecnologias. O projeto desenvolvido ao longo da série, e disponibilizado no site desta publicação, pode ser usado como fonte de estudo e de experiências para o leitor que quiser consolidar seus conhecimentos, criar seus próprios projetos Java EE e ficar em sintonia com o mercado de desenvolvimento corporativo. ●

jcp.org/aboutJava/communityprocess/final/jsr220/index.html

Download da especificação EJB 3.0 simplificada – escolha a primeira opção de download, na tela seguinte aceite os termos de licença e escolha o link JSR-000220 Enterprise JavaBeans 3.0 Final Release (simplified)



Renato Bellia

(rbellia@globalcode.com.br)

é instrutor de cursos Java da Globalcode, onde colabora com o desenvolvimento de treinamentos.

Atua na área de informática há mais de dez anos, é formado em Engenharia de Computadores pela FEI e possui as certificações SCJP, SCWCD e SCBCD.



Todo profissional precisa do melhor ponto de partida

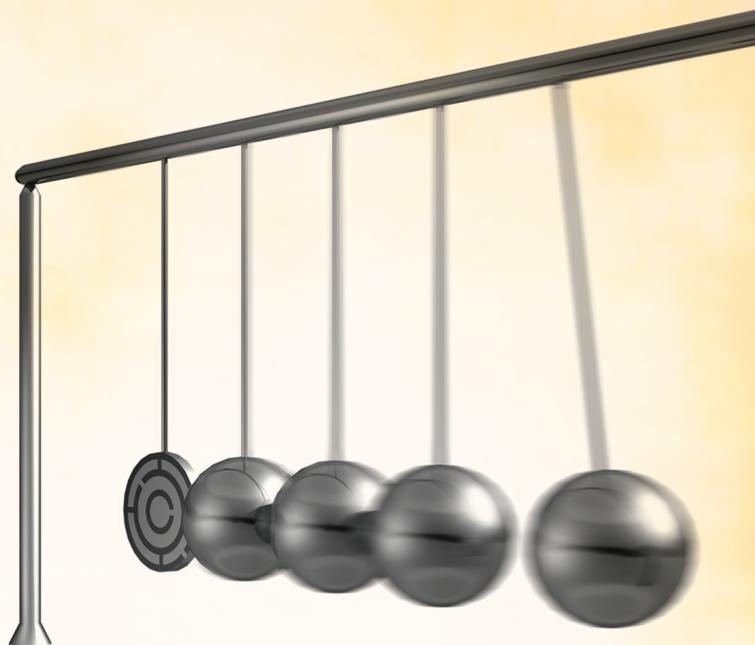
A Pontoclass Treinamentos é o melhor ponto de partida para quem deseja aprender tecnologias relacionadas à Orientação a Objetos - JAVA e UML

Analisar e desenvolver sistemas é muito mais do que conhecer a sintaxe de uma linguagem. Você só estará realmente preparado para desenvolver e analisar sistemas Java e UML se souber "pensar" Orientado a Objetos.

Nossa metodologia de ensino proporciona um ambiente dinâmico ao aprendizado Java e UML, consolidando as bases da Orientação a Objetos através da prática.

Na Pontoclass Treinamentos:

- O iniciante terá o melhor ponto de partida.
- O profissional encontrará o melhor direcionamento.



tel.: 11 3123-3606
www.pontoclass.com.br

Java Fundamentos • Java Elementos • Orientação a Objetos com UML • Java Mobile • Java Web • Certificações Java e UML



pontoclass

Relatórios com Tabula

Usando o componente Crosstab no Jasper

Na construção de aplicações corporativas, é comum encontrar requisitos que envolvem a implementação de relatórios de tabulação cruzada. São relatórios na forma de uma tabela, que podem crescer dinamicamente e terem agrupamentos tanto por linhas quanto por colunas, com sumarizações nas duas direções. Veja a estrutura básica de um desses relatórios na **Figura 1**.

Desde sua versão 1.1.0, o framework JasperReports possibilita a criação de relatórios “crosstab”, (como muitas vezes são chamados), através do componente **JRCrosstab** ou Crosstab. O componente também é suportado pelo designer de relatórios iReport, permitindo a criação interativa de relatórios de tabulação cruzada.

Este artigo apresentará um caso de exemplo e as etapas envolvidas na criação e geração de um relatório crosstab.

Supõe-se que o leitor já possua conhecimentos básicos sobre o iReport/JasperReports. Para o exemplo, será utilizado o banco de dados relacional MySQL, mas qualquer outro SGBD com driver JDBC atualizado pode ser utilizado.

Sobre o exemplo

O exemplo será conceitualmente simples: queremos implementar um relatório que mostra a quantidade de vendas das publicações de uma editora, por edição (mês/ano) em cada região do país, além de sua totalização. Um protótipo do resultado final pode ser visto na **Figura 2**.

Construção do relatório

Podemos criar o relatório de exemplo usando um assistente (wizard) ou de forma “manual”. Neste artigo será dada preferência à forma direta. Assim poderemos apresentar melhor os conceitos, propriedades e ações envolvidas na elaboração de um relatório com tabulação cruzada. Além disso, criando o relatório manualmente veremos como um relatório gerado através do assistente pode ser posteriormente alterado.

Para os que depois optarem pelo uso do assistente, o **quadro** “Assistente Crosstab” descreve os passos necessários.

☞ *As edições 13, 21, 37 e 38 da Java Magazine já abordaram a implementação de relatórios a partir do JasperReports e iReport, tanto em aspectos teóricos como nas mais variadas possibilidades de implementação de relatórios, básicos e avançados.*

Iniciando o relatório

O primeiro passo é criar um novo relatório em branco. Para melhor visualização do resultado, mude a orientação para paisagem: acesse o comando de menu *Edit | Report Properties* e no item *Orientation* escolha a opção *Landscape*. Você precisará também preparar a base de dados e a conexão; separamos estes passos no **quadro** “Configurando o acesso a dados”.

Com o acesso a dados configurado, defina a fonte de dados do relatório usando o comando *Data | Report Query* e acrescentando a seguinte consulta SQL:

```
select pub.idPublicacao, pub.nomePublicacao,
       reg.siglaRegiao, v.qtdVenda, v.mesReferencia
from publicacao as pub
inner join vendas as v on (pub.idPublicacao = v.idPublicacao)
inner join regiao as reg on (v.idRegiao = reg.idRegiao)
order by pub.idPublicacao, v.mesReferencia, reg.siglaRegiao
```

Os campos (*fields*) do relatório já podem ser automaticamente obtidos, usando o botão *Read Fields* nesta mesma tela de definição da consulta.

Conclua a estrutura básica do relatório acrescentando um texto estático à banda *Title*: “Vendas por Região, Mês a Mês”.

Adicionando o Crosstab

O próximo passo será a inserção e a manipulação do próprio componente Crosstab. Para inserir o Crosstab, clique no botão correspondente na barra de ferramentas do iReport (veja a **Figura 3**) e posicione e dimensione o componente na banda *Summary*.

ção Cruzada

Reports e iReport

Crie relatórios avançados e dinâmicos com agrupamentos e sumarização detalhada, usando o conjunto de ferramentas open source mais usado para a criação de relatórios em Java

JOSÉ TEIXEIRA DE CARVALHO NETO

☞ Diferentemente dos relatórios tradicionais, o detalhe de um relatório de tabulação cruzada (que é o próprio componente) não deverá ser posicionado na banda Detail. Se fizermos isso, apesar de não serem relatados erros de compilação, será gerado um relatório incorreto.

O Crosstab se comportará como um sub-relatório do relatório principal, com bandas (denominadas **células**), agrupamentos, parâmetros e campos próprios.

Ao acrescentar o componente ao relatório, o assistente será exibido automaticamente. Mas como decidimos não utilizá-lo, basta clicar em *Cancel*. Ao adicionar o Crosstab, uma nova aba de nome *crosstab-1* surge na janela principal do iReport (usaremos vários elementos desta aba adiante).

Propriedades do Crosstab

É importante conhecer e definir algumas propriedades relacionadas ao componente. Para acessar estas propriedades, dê um duplo clique sobre o Crosstab e selecione a aba *Crosstab* na janela que se abre (Figura 4).

Entre estas propriedades, destacam-se *Repeat column headers* e *Repeat row headers*, que habilitam a repetição dos títulos das colunas e das linhas, respectivamente (caso o relatório tenha mais de uma página). A propriedade *Column break offset* delimita o espaço em pixels, entre duas partes de um Crosstab que exceder a largura da página, por conta do crescimento do número de colunas.

Na aba *Crosstab* há também um espaço onde podem ser definidos parâmetros de entrada para uso exclusivo do componente Crosstab, mas não precisaremos destas configurações para nosso exemplo. (Tais parâmetros se comportam da mesma forma que os parâmetros de entrada do relatório principal.)

Cabeçalho	Coluna 1	Coluna 2	Coluna 3	...	Total
Linha 1					
Linha 2					
Linha 3					
...					
Total					

Figura 1. Layout genérico de um relatório do tipo tabulação cruzada

Vendas publicação/mês	Janeiro - 07					Fevereiro - 07					Total geral
	CO	N	NE	...	Total	CO	N	NE	...	Total	
Publicação 01	00	00	00	...	00	00	00	00	...	00	00
Publicação 02	00	00	00	...	00	00	00	00	...	00	00
Publicação 03	00	00	00	...	00	00	00	00	...	00	00
...
Total	00	00	00	...	00	00	00	00	...	00	00

Figura 2. Protótipo do relatório de exemplo



Figura 3. Barra de ferramentas, com realce em vermelho para o Crosstab

A seguir, é necessário criar os agrupamentos de linhas e colunas, bem como suas *medidas*. Para acessar estas propriedades avançadas do Crosstab, clique no botão *Edit Crosstab Properties*. Será mostrada uma nova janela.

Dataset

A primeira aba nesta janela (*Crosstab data*) possibilita definir a origem dos dados que serão apresentados pelo Crosstab. Os dados podem vir do relatório principal, o que acontece no exemplo, ou de um conjunto de dados secundário (que no iReport é representado por um Subdataset). Como não vamos usar Datasets, a opção *Use a*



Figura 4. Janela de propriedades gerais do componente Crosstab

dataset to fill the crosstab deve estar desmarcada, como na **Figura 5**.

Linhas, colunas e células

Um relatório de tabulação cruzada deve possuir no mínimo um agrupamento de linhas e um agrupamento de colunas. À medida que são adicionados mais agrupamentos, um passa a ser subgrupo do anterior.

Para o exemplo, os agrupamentos de linhas e colunas serão adicionados de acordo com o protótipo visto anteriormente (**Figura 2**). Isso pode ser feito na

segunda aba (*Row and column groups*) das propriedades do Crosstab.

Primeiro criamos os agrupamentos de linhas. Clique no botão *Add* da área *Row groups*, e uma nova janela será aberta para configuração do novo agrupamento. Nesta janela, será necessário dar um nome ao agrupamento no campo *Group Name*. Devemos também configurar a largura para as células do agrupamento, em pixels, no campo *Group Width*. Usamos o nome “publicacao” e definimos a largura como 80 (veja a **Figura 6**).

Na área *Bucket Expression* é escolhida a origem dos dados a serem apresentados nas linhas do agrupamento. Estes dados podem vir de parâmetros, de campos ou de variáveis do relatório principal. No exemplo, os dados que serão apresentados se relacionam aos nomes das publicações da editora, que encontra um campo correspondente em **`$F{nomePublicacao}`**. Este campo foi mapeado da consulta do exemplo, no início do artigo, e deve ser do tipo **`java.lang.String`**.

É necessário também escolher a or-

Assistente Crosstab

O componente Crosstab pode ser adicionado e configurado de maneira rápida (mas limitada) através de um assistente do iReport. A janela do assistente será exibida sempre que for incluído o componente no relatório. Veja a seguir uma descrição resumida dos passos envolvidos.

O primeiro passo é a definição do Dataset do relatório (**Figura Q1**). A opção padrão, já selecionada, é da fonte de dados do próprio relatório. E caso existam Subdatasets configurados, estes aparecerão na lista.

O próximo passo (**Figura Q2**) é a seleção dos parâmetros, campos ou variáveis que corresponderão aos agrupamentos de linhas do Crosstab. (Aqui surge a primeira limitação do assistente: só podem ser criados dois agrupamentos.)

Após selecionar os agrupamentos de linhas, o passo seguinte permite selecionar os agrupamentos de colunas. Aqui também há o limite de dois agrupamentos, apenas.

A seguir deve-se selecionar um dos campos, que será o detalhe dos cruzamentos entre agrupamentos de linhas e agrupamentos de colunas, bem como a função matemática que será aplicada. O campo de detalhe é referenciado no Crosstab com o nome da medida. Veja a **Figura Q3**.

O último passo (**Figura Q4**) envolve a totalização das linhas e colunas do relatório, e detalhes de visualização. É possível decidir pela exibição

ou não dos totais sobre os agrupamentos de linhas e de colunas, com as opções *Add row group totals* e *Add column group totals*. Pode-se também optar pela colocação das linhas de grade nas células do relatório, com a opção *Show grid lines*.

Executados estes passos, basta apenas finali-

zar o assistente e o componente Crosstab estará criado no relatório.

Observe que para relatórios muito básicos, o assistente atenderá satisfatoriamente, mas em casos mais complexos, o desenvolvedor terá que partir para a construção manual do relatório, como fazemos neste artigo.

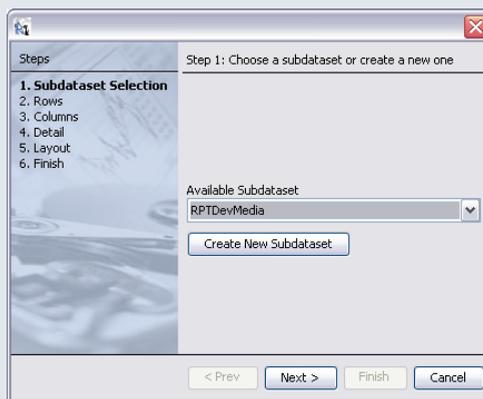


Figura Q1. Definição da origem dos dados do Crosstab

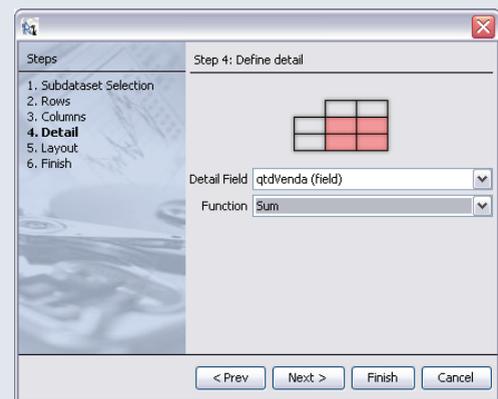


Figura Q3. Seleção do detalhe (medida) do Crosstab

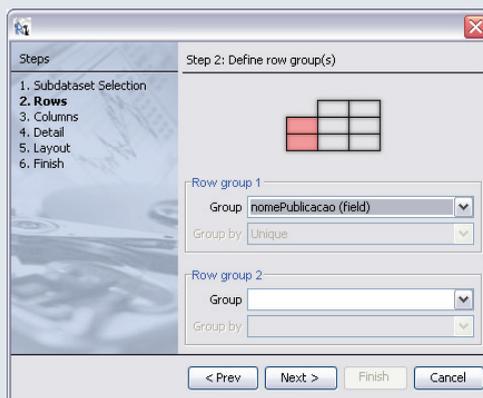


Figura Q2. Definição dos agrupamentos de linhas

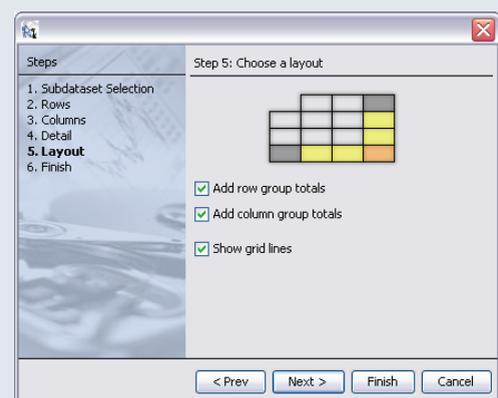


Figura Q4. Adição da exibição dos totais e linhas de grade

denaço dos valores apresentados pelo agrupamento, no campo *Order*. Finalizando, na propriedade *Print the group header on this position*, é possível alinhar o título/cabeçalho das linhas nas posições *Top*, *Bottom* e *Center*. Você pode também usar *Stretch* para expandir em toda a célula.

Pode-se ainda optar por exibir uma totalização das medidas (measures) das linhas que forem apresentadas, na lista *Print the group total on this position*. Para o exemplo, foi escolhida a opção *End*, que exibe a totalização após as linhas de detalhe. Outras opções são *None* (a totalização não é exibida) e *Start* (a totalização é exibida antes das linhas de detalhe do grupo).

Definindo os agrupamentos

Surge agora a necessidade de definir os agrupamentos de colunas, que serão dois. A definição consiste dos mesmos parâmetros utilizados para os agrupamentos de linhas, que podem ser adicionados com o botão *Add* da área *Column groups*.

Crie um agrupamento de nome “mesReferencia”, onde a *Bucket Expression* será ***#{mesReferencia}*** e o tipo, ***java.sql.Timestamp***. Em seguida defina outro agrupamento de nome “regiao”, com expressão ***#{siglaRegiao}*** e tipo ***java.lang.String***. As opções de posicionamento do título/cabeçalho das colunas mudam apenas em relação às opções *Top* e *Bottom*, que são substituídas por *Left* e *Right*. Por fim, em ambos os agrupamentos, a linha de total

deve ser apresentada após os detalhes: escolha a opção *End*, na lista *Print the group total on this position*.

É importante lembrar que a ordem em que os agrupamentos são adicionados será a ordem na qual suas células de informações serão exibidas no relatório (veja a **Figura 7**). No exemplo, serão agrupados e exibidos os meses de referência acima das regiões. Esta ordem pode ser alterada após a criação dos agrupamentos, através dos botões *Up* e *Down*, na área correspondente da tela *Row and column groups*.

Criando e configurando medidas

Com os agrupamentos de linhas e colunas adicionados, o último passo na configuração das propriedades do Crosstab será

Configurando o acesso a dados

Após realizada a instalação e a configuração básica do banco de dados, bem como a ativação do seu serviço (estamos usando o MySQL), deve-se realizar a criação da base de dados (usamos o nome “devmedia”), e suas tabelas e chaves. A modelagem Entidade-Relacionamento desta base de dados pode ser vista na **Figura Q1**.

Os scripts SQL correspondentes à criação destes elementos estão disponíveis para download. Para executar os scripts utilizamos um cliente multiplataforma e com versão gratuita, de nome Aqua Data Studio (aquafold.com/downloads.html). Mas você pode usar a ferramenta com que estiver confortável.

No ambiente do iReport, é preciso acrescentar uma nova conexão/fonte de dados. Selecione *Data|Connections|Data Sources* e na janela que se abre clique no botão *New*. Defina então as propriedades como na **Figura Q2**:

- *Name* não Devmedia (ou um nome qualquer que identifique a conexão);
- *Type of Connection/Data Source* – Database

JDBC Connection;

- *JDBC Driver* – com.mysql.jdbc.Driver (driver JDBC do MySQL e que já faz parte da instalação do iReport);
- *JDBC URL* – jdbc:mysql://127.0.0.1/devmedia (URL supondo o acesso ao MySQL instalado na própria estação);
- *User Name* – root (ou outro usuário criado posteriormente);
- *Password* – (a senha de root definida durante a instalação do MySQL, e que pode não ser mais solicitada pelo iReport se a opção *Save Password for* marcada).

A configuração pode ser testada clicando-se o botão *Test*. Em caso de sucesso, basta salvar a configuração. Pronto: o relatório já poderá acessar e exibir os dados armazenados no BD. Basta selecionar, pelo nome, a conexão recém-criada na barra de ferramentas do iReport.



Figura Q2. Configuração da conexão à base de dados, no iReport

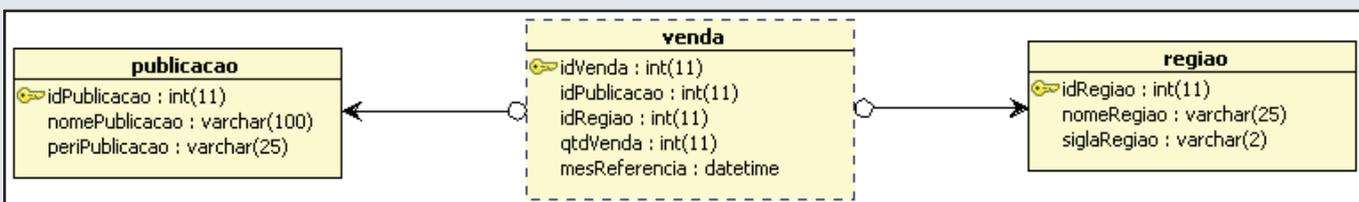


Figura Q1. Modelo ER da base de dados de exemplo

definir suas medidas (*measures*). As medidas, que já citamos algumas vezes acima, são variáveis essenciais ao componente Crosstab, e devem ser sempre utilizadas. Elas representam o cruzamento de um determinado registro entre as informações apresentadas nas linhas e nas colunas.

No nosso exemplo, o relatório terá apenas uma medida, “quantidade”, representando a número de revistas vendidas. Para criá-la, selecione a aba *Measures* e clique no botão *Add*. Na janela de configuração dê um nome à medida: “quantidade”; e defina o seu tipo no campo *Measure Class Type*: `java.lang.Integer`.

Neste ponto, surge a primeira nova opção: escolher um tipo de cálculo para a medida em *Calculation Type*. O cálculo pode ser o próprio valor vindo da expressão (opção *Nothing*); ou ser o resultado de operações sobre os valores. Por exemplo, *Count* faz uma contagem dos registros retornados; *Sum* realiza a soma, e *Average* calcula a média aritmética; *Highest* seleciona o maior valor entre os registros retornados e *Lowest*, o menor.

Vamos usar a opção *Sum*, pois queremos apresentar o total de vendas das publicações por mês e região. É possível ainda exibir a medida como porcentagem do total geral (para um determinado cruzamento de linha/coluna). Para isso, basta selecionar uma opção da lista *Percentage of*. Você pode também implementar sua própria classe de cálculo de porcentagem.

Por fim, assim como fizemos para os agrupamentos, é necessário escolher uma expressão para a medida, em *Measure expression*. A expressão, neste caso, deve corresponder ao campo retornado na consulta que represente a quantidade de vendas: `$F{qtdVenda}`. A criação da medida “quantidade” pode ser vista na **Figura 8**.

Layout do Crosstab

Com as propriedades do Crosstab configuradas, chega a hora de compor o layout do componente. Para fazer isso, clique na aba *crosstab-1* (que surgiu na janela principal do iReport após a inclusão do componente). Na apresentação desta aba, vemos um “raio-x” da distribuição das células no Crosstab após a criação dos agrupamentos e das medidas (veja a **Figura 9**). Surgem também dois novos painéis à direita. O painel *Crosstab objects* exibe o conjunto de agrupamentos (marcador verde) e as medidas do componente¹ (marcador azul); já o painel *Crosstab structure* mostra a estrutura e o conteúdo das áreas do Crosstab.

Atribuição de objetos às células

Para cada uma das células do Crosstab (exceto o cabeçalho e os totais), deve-se

¹ Se durante a criação dos agrupamentos de linhas e colunas tiver se optado por acrescentar a linha de totalização, o próprio iReport se encarregará de criar objetos responsáveis por essas totalizações. Estes objetos são identificados no painel com um marcador azul claro e são referências às medidas do Crosstab.

atribuir um dos objetos visualizados no painel *Crosstab objects*. Para isso basta arrastar o elemento correspondente e soltar na célula sugerida para ele. Cada célula indica o local adequado para um determinado elemento, mas o desenvolvedor poderia optar por atribuir outro valor para a célula (um valor constante, por exemplo). O objeto já será dimensionado de acordo com a área da célula.

Veja a seguir a lista de células e os objetos que devem ser relacionados a elas, para o nosso exemplo (veja também a **Figura 10**):

- *publicacao header* – recebe o objeto **publicacao**, o agrupamento de linhas.
- *mesReferencia header* – recebe o objeto **mesReferencia**, o primeiro agrupamento de colunas.
- *regiao header* – recebe o objeto **regiao**, o segundo agrupamento de colunas.

As seguintes células devem receber o objeto **quantidade**:

- *detail / detail* – a própria medida.
- *detail / regiao* – totalizador das regiões por publicação.
- *detail / mesReferencia* – totalizador dos meses por publicação.
- *publicacao / detail* – totalizador das publicações por região.
- *publicacao / regiao* – totalizador das publicações por todas as regiões, dentro de um determinado mês.
- *publicacao / mesReferencia* – totalizador das publicações por todos os meses.

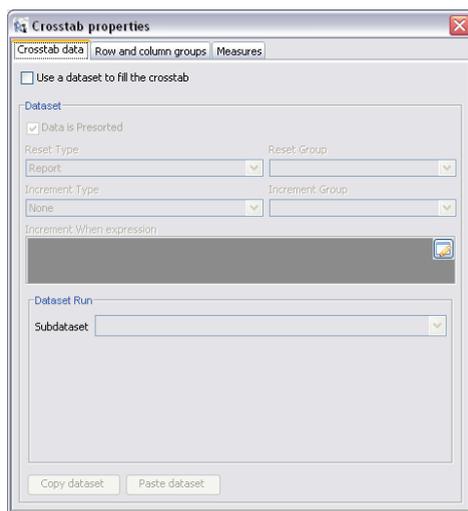


Figura 5. Configuração de propriedades de Datasets do Crosstab



Figura 6. Criação de um agrupamento de linhas

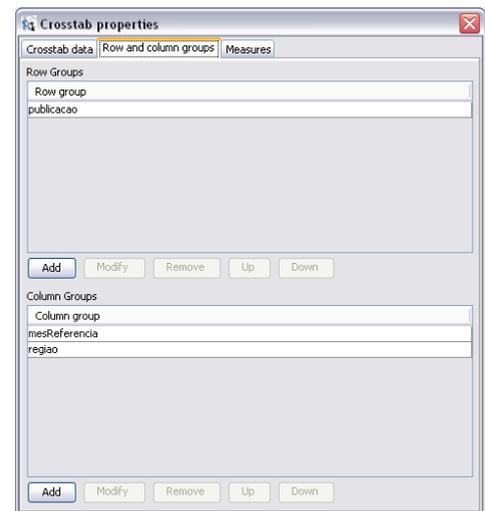


Figura 7. Agrupamentos de linhas e colunas adicionados ao Crosstab

Nas outras células, com conteúdo estático, recomenda-se adicionar um texto que dê significado ao que será apresentado naquela linha/coluna. Estas células são: *Crosstab header*, *publicacao Total header*, *mes Total header* e *regiao Total header*.

Existe ainda uma célula especial que será exibida apenas no caso de o relatório não possuir conteúdo para apresentar. Para editar esta célula, de nome *When-No-data*, clique com o botão direito sobre a área de desenho do Crosstab e selecione o item *Edit When-No-Data default cell*. Uma moldura amarelada surgirá indicando o espaço para sua definição. Normalmente este espaço conterá algum texto estático informando que o relatório não possui dados a apresentar.

Após a adição dos elementos às suas células, já é possível testar o exemplo e obter um resultado concreto. A **Figura 11** mostra o relatório resultante (valores são fictícios).

Ajustes visuais

A parte “bruta” do relatório já está pronta, e as informações já são mostradas corretamente. Nota-se, no entanto, que a

apresentação está um tanto confusa e até certo ponto difícil de interpretar. É preciso ajustar o layout para que o conteúdo geral fique mais claro.

Uma primeira ação é adicionar as linhas de grade das linhas, colunas e células. Selecione todas as células do Crosstab e clique com o botão direito, tomando o cuidado de manter todos os itens selecionados. Depois escolha a opção *Element Properties*. Na nova janela que surge, acesse a aba *Borders* e para cada uma das bordas selecione a opção *Thin*. Outra opção equivalente é através do item , na barra de ferramentas do iReport.

Os próximos aspectos a serem melhorados são destacar em negrito o título do Crosstab (conteúdo da célula *crosstab header*) e as informações exibidas nas linhas e colunas; alterar a cor de fundo das linhas e colunas de total, e centralizar verticalmente o conteúdo de todas as células. O negrito pode ser aplicado selecionando as células e clicando em . Com um clique no botão direito e selecionando a opção *Element Properties* (aba *Common*), é possível alterar a cor do plano de fundo

das células. Finalmente, para centralizar verticalmente as células, selecione todas e clique em .

Precisamos ainda alterar o formato de visualização da célula do mês de referência, já que o formato apresentado (completo com data/hora) não é adequado. Para alterá-lo, selecione a célula correspondente *mesReferencia header* e com um duplo-clique acesse as propriedades do elemento. Na aba *Text Field*, no campo *Pattern*, acrescente **MMMM-yy**. Outros formatos podem ser testados, clicando no botão *Create* ao lado do campo.

As melhorias na apresentação, cujo resultado pode ser visto nas **Figuras 12** e **13** são apenas sugestões (e as mais práticas possíveis). O leitor poderá adaptar e incrementar o relatório de acordo com suas preferências.

Conclusões

A dupla JasperReports/iReport já está definitivamente consolidada entre os desenvolvedores Java. A cada versão, novas e eficientes funcionalidades surgem para tornar essas ferramentas mais completas e

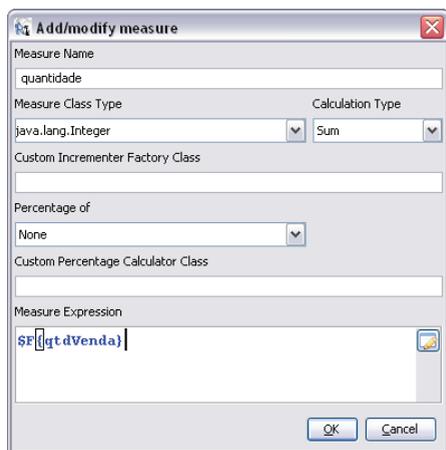


Figura 8. Criação de uma medida

Vendas	\$V		Total
Publicação/Mês	{mesReferencia}	Total	Geral
	{regiao}		
\$V{publicacao}	\$V	\$V	\$V
	{quanti {quantida	{quantida	{quantida
Total	\$V	\$V	\$V
	{quanti {quantida	{quantida	{quantida

Figura 10. Layout inicial do Crosstab, após atribuição dos objetos às células

crosstab header	mesReferencia header		mesReferencia total header
	mesReferencia total header	regiao total header	
publicacao header	detail / detail	detail / regiao	detail / mesReferencia
publicacao total header	publicacao / detail	publicacao / regiao	publicacao / mesReferencia

Figura 9. Visualização das células no layout do Crosstab após sua configuração

Vendas	31/01/07 00:00						28/02/07 00:00						31/03/07 00:00						Total Geral
	Publicação/Mês																		
	CO	N	NE	S	SE	Total	CO	N	NE	S	SE	Total	CO	N	NE	S	SE	Total	
.net Magazine	112	120	125	90	250	697	245	115	0	136	420	916	0	180	70	0	270	520	2133
Java Magazine	2850	2550	2740	2500	4100	14740	2050	2580	2650	4650	3720	15650	3100	2500	2950	3200	4200	15950	46340
SQL Magazine	1500	2300	2240	2250	3910	12200	2500	2450	2500	2450	3020	12920	3000	1950	2200	1900	3750	12800	37920
WebMobile	1180	2250	2250	0	2448	8128	0	0	0	0	0	0	1200	2250	1500	1250	3100	9300	17428
Total	5642	7220	7355	4840	10708	35765	4795	5145	5150	7236	7160	29486	7300	6880	6720	6350	11320	38570	103821

Figura 11. Visão do relatório no iReport JasperViewer

Relatórios com Tabulação Cruzada

profissionais. Neste artigo, demonstramos um componente avançado do JasperReports, integrado ao iReport, para geração de relatórios com tabulação cruzada. A flexibilidade deste tipo de relatório torna seu campo de aplicação vasto. Os relatórios de tabulação

cruzada são muito utilizados na análise de informações gerenciais e no suporte à gestão de negócios, sendo disponibilizados em sistemas de apoio a decisões e ferramentas de Business Intelligence.

Vendas Publicação/Mês	\$V		Total Geral
	{mesReferencia}	Total	
{publicacao}	{regia}	{total}	{total}
Total	{total}	{total}	{total}

Figura 12. Layout final do Crosstab, após ajustes visuais

jasperforge.org/sf/projects/ireport
iReport

jasperforge.org/sf/projects/jasperreports
JasperReports

mysql.org/downloads/
Download do banco de dados MySQL

JasperReports for Java Developers
David Heffelfinger (Packt Publishing)
Referência completa para a implementação de relatórios com JasperReports

Para saber mais

Relatórios Corporativos (Edição 13)

Apresentação do JasperReports e iReport como alternativa para a criação de relatórios profissionais

Relatórios Avançados (Edição 21)

Utilizando parâmetros, imagens e JavaBeans com JasperReports e iReport

Relatórios na Web Passo a Passo (Edição 37)

Criando um relatório mestre-detalhe completo baseado no JasperReports, com a facilidade do iReport

Mais Relatórios Passo a Passo (Edição 38)

Definindo agrupamentos, calculando totais e criando gráficos – também com JasperReports e iReport



José Teixeira de Carvalho Neto

(jtneto@gmail.com)

é graduado em Ciências da Computação e possui MBA em

Tecnologia da Informação, pelo Unipê.

É também Sun Certified Associate e Sun Certified Java Programmer, além de Líder de Desenvolvimento Web em Java, na Simpletec Informática, com foco em soluções de e-Gov.

VENDAS POR REGIÃO, MÊS A MÊS																			
Vendas Publicação/Mês	Janeiro-07						Fevereiro-07						Março-07						Total Geral
	CO	N	NE	S	SE	Total	CO	N	NE	S	SE	Total	CO	N	NE	S	SE	Total	
.net Magazine	112	120	125	90	250	697	245	115	0	136	420	916	0	180	70	0	270	520	2133
Java Magazine	2850	2550	2740	2500	4100	14740	2050	2580	2650	4650	3720	15650	3100	2500	2950	3200	4200	15950	46340
SQL Magazine	1500	2300	2240	2250	3910	12200	2500	2450	2500	2450	3020	12920	3000	1950	2200	1900	3750	12800	37920
WebMobile	1180	2250	2250	0	2448	8128	0	0	0	0	0	0	1200	2250	1500	1250	3100	9300	17428
Total	5642	7220	7355	4840	10708	35765	4795	5145	5150	7236	7160	29486	7300	6880	6720	6350	11320	38570	103821

Figura 13. Visão do relatório final (formatado), no iReport JasperViewer



S U M M A
T E C H N O L O G I E S

Se você for, leve quem você confia!

Conheça os Frameworks Open-Source

genesis

Finalmente é possível desenvolver aplicações corporativas escaláveis de forma produtiva e simples. As funcionalidades do genesis são perfeitas para aplicações desktop por requerer apenas a criação de JavaBeans e o uso de anotações, sem precisar de diversos arquivos de configuração xml. O framework permite também que métodos remotos e transacionais de integração com o servidor sejam implementados com classes Java comuns e anotações. Além disso, o genesis permite que a sua solução evolua naturalmente de uma aplicação local para um rich-client com servidor J2EE sem necessidade de alteração do código fonte.

Greenbox

O Greenbox ajuda a criar padrões de desenvolvimento dentro das companhias, pregando software gerador a partir de inúmeros artefatos, que vão de banco de dados, XML até Annotations (Java 5). Não força com que você trabalhe com frameworks pré-estabelecidos, da mesma forma como você pode usar Hibernate para persistência, você pode usar IBatis, OJB ou até mesmo JDBC se você preferir, e todo esse poder é oferecido pelo poderoso engine de templates baseados em Jakarta Velocity.

Os frameworks Greenbox e genesis foram desenvolvidos pela Summa Technologies e são disponibilizados de forma livre e gratuita (licença open source LGPL).

Para informações, downloads e documentações consulte os sites:
<http://www.summa-tech.com.br/genesis.html>
<http://greenbox.javaforge.com/>

Uma empresa premiada!

Os projetos que contaram com consultoria da Summa Technologies incluem os únicos sistemas brasileiros que já receberam o almejado prêmio **Duke Award**, conferido anualmente durante o **JavaOne Conference**: Projeto Cartão Nacional da Saúde do Datasus (2003), Projeto IRPF Multiplataforma (2004) da Receita Federal e Serpro e Projeto do Sistema Interado de Agendamento de Pacientes da Secretaria de Saúde da Prefeitura de São Paulo (2005).



Consultoria . Arquitetura . Mentoring . Treinamento . Framework . Componentes

Summa Technologies do Brasil . Rua Funchal, 411 11º andar cj. 112 . 04551-060 . São Paulo . SP . +55.11.3846.1622 . www.summa-tech.com

Em busca de novos desafios em sua carreira? Envie seu currículo para: curriculo@summa-tech.com

JSF com MyFaces e

Aprenda a utilizar os recursos do projeto

Uma das implementações do JavaServer Faces mais utilizada atualmente é o MyFaces, um projeto da Apache Software Foundation, que vem crescendo rapidamente e hoje vai muito além da especificação JSF. Além do MyFaces Core, a implementação do JSF em si, há vários subprojetos do MyFaces que fornecem componentes adicionais, trazendo suporte a Ajax e à vinculação com dados, entre outras funcionalidades importantes no desenvolvimento web.

Neste artigo apresentaremos o Apache MyFaces em geral, e criaremos um exemplo que faz uso dos componentes JSF fornecidos pelo mais popular dos seus subprojetos, o Tomahawk.

Projetos do MyFaces

Muitas das funcionalidades que diferenciam o MyFaces estão nos seus subprojetos, que apresentamos brevemente a seguir:

- **Tomahawk** – O Tomahawk é o principal e mais utilizado subprojeto do MyFaces. Ele possui uma grande quantidade de componentes visuais, como menus, tabelas com ordenação interativa, calendários e até um editor de HTML. Já há versões estáveis disponíveis para uso em produção.

- **ADF Faces / Trinidad** – O ADF Faces inclui mais de 100 componentes JSF. Foi desenvolvido originalmente pela Oracle

como uma implementação completa do padrão JSF e depois doado à Apache Software Foundation. Embora os componentes estejam bastante maduros, o projeto está atualmente incubado no projeto MyFaces. Quando sair da incubação, assumirá o nome “Trinidad”.

- **Tobago** – Além de ser um conjunto de componentes JSF que funciona sobre a implementação do MyFaces, o principal objetivo do Tobago é tornar mais ágil o desenvolvimento de aplicações web. O Tobago fornece um gerenciador de layouts que organiza automaticamente os componentes na página, não sendo necessário o uso de tabelas para esse objetivo.

- **Sandbox** – O projeto Sandbox consiste em componentes JSF ainda em fase de desenvolvimento, e que futuramente podem fazer parte do Tomahawk.

A aplicação de exemplo

Para ilustrar o uso do MyFaces, vamos construir uma agenda de contatos, conforme ilustrado na **Figura 1**. O exemplo utiliza vários componentes padrão do JSF (fornecidos pelo MyFaces Core) e também diversos outros incluídos com o Tomahawk.

Na inclusão de novos contatos, é usado um componente de calendário para informar a data de aniversário, conforme

Listagem 1. Classe Contato

```
package br.com.jm.agenda;

import java.util.Date;

public class Contato implements Comparable<Contato> {
    private int codigo;
    private String nome;
    private String telefone;
    private Date aniversario;
    private boolean favorito;

    //... getters e setters omitidos ...
}
```

Tomahawk

Apache MyFaces

Use a implementação do JavaServer Faces da Apache, considerada a mais sólida e confiável, e conheça um conjunto de tags JSF poderoso

FRANCISCO CALAÇA XAVIER

mostra a **Figura 2**. Note que usamos também um componente de menu. Com o Tomahawk é possível construir menus com submenus rapidamente. Outra funcionalidade interessante é a ordenação de colunas em tabelas. Através de cliques nos títulos das colunas da nossa tabela de contatos é possível ordená-las conforme desejado. E à medida que um contato marcado como favorito é adicionado, este aparece no menu *Favoritos*.

Para mantermos o foco na programação JSF, não será feito acesso a banco de dados; todas as informações dos contatos permanecerão em memória (usando coleções). Na **Listagem 1** está o código da entidade **Contato**. Na **Listagem 2**, temos o código da classe **ContatoDao** que é responsável por incluir e consultar dados de contatos. O método **consultarFavoritos()** retorna todos os contatos que possuem a propriedade **favoritos = true**, e **consultarAniversariantes()** obtém os que fazem aniversário no dia e no mês fornecidos como parâmetros.

Adicionando suporte ao MyFaces Tomahawk

O Tomahawk precisa de uma implementação do padrão JSF para funcionar. Em nosso exemplo, usaremos o MyFaces Core, que é por muitos considerada a implementação mais sólida do padrão. Nada impede também que o leitor use os componentes do Tomahawk junto com a implementação de referência da Sun para o JSF, ou ainda com outra opção. Veja detalhes sobre versões e questões de compatibilidade no site myfaces.apache.org.

Para configurar o Tomahawk, deve-se também incluir o filtro **org.apache.myfaces.component.html.util.ExtensionsFilter** no **web.xml** (**Listagem 3**). Observe que, entre outras coisas, definimos a extensão **.faces** como padrão para as páginas baseadas em JSF. Finalmente, para a utilização dos compo-

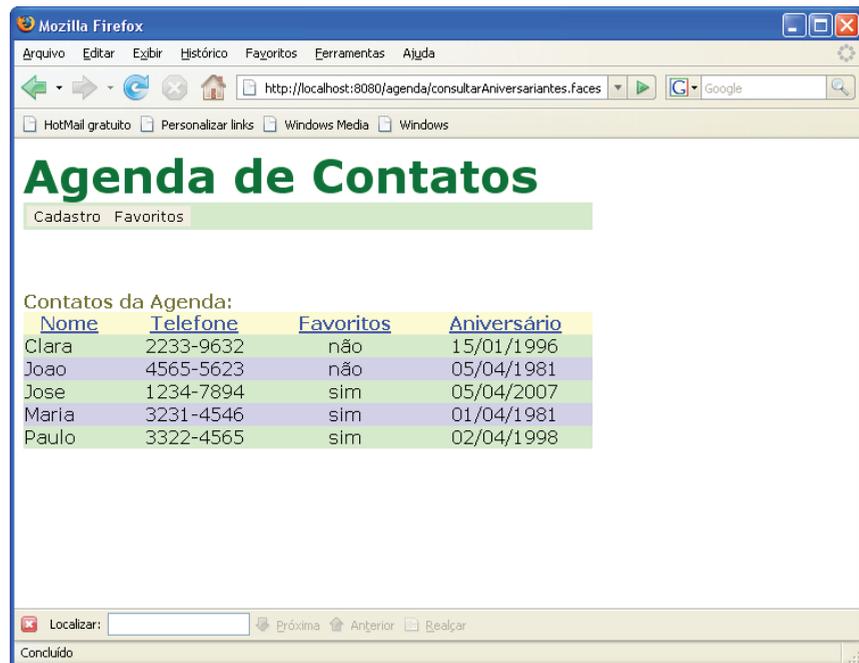


Figura 1. Página principal da aplicação de exemplo

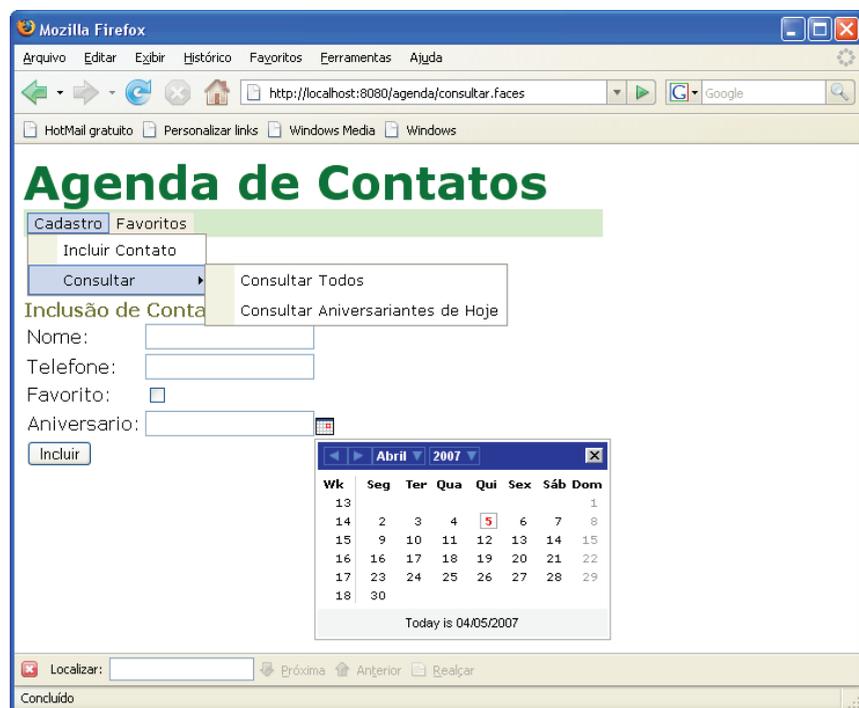


Figura 2. Consulta e inclusão de contatos

Listagem 2. Classe ContatoDao

```
package br.com.jm.agenda;

import java.util.*;

public class ContatoDao {
    private static List<Contato> contatos = new ArrayList<Contato>();

    public void incluir(Contato contato) {
        contatos.add(contato);
    }

    public List<Contato> consultar() {
        return contatos;
    }

    public List<Contato> consultarFavoritos() {
        List<Contato> resultado = new ArrayList<Contato>();
        for (Contato contato : contatos) {
            if (contato.isFavorito()) resultado.add(contato);
        }
        return resultado;
    }

    public List<Contato> consultarAniversariantes(Date data) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(data);
        int dia = cal.get(Calendar.DAY_OF_MONTH);
        int mes = cal.get(Calendar.MONTH);
        List<Contato> resultado = new ArrayList<Contato>();

        for (Contato cont : contatos) {
            Calendar calAniversario = Calendar.getInstance();
            calAniversario.setTime(cont.getAniversario());
            int diaAniversario = calAniversario.get(Calendar.DAY_OF_MONTH);
            int mesAniversario = calAniversario.get(Calendar.MONTH);
            if (diaAniversario == dia && mesAniversario == mes) {
                resultado.add(cont);
            }
        }
        return resultado;
    }
}
```

Listagem 3. web.xml completo

```
<web-app id="WebApp_ID">
    <display-name>agenda</display-name>

    <filter>
        <filter-name>extensionsFilter</filter-name>
        <filter-class>
            org.apache.myfaces.component.html.util.ExtensionsFilter
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>extensionsFilter</filter-name>
        <url-pattern>*.faces</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>extensionsFilter</filter-name>
        <url-pattern>/faces/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
</web-app>
```

mentes do Tomahawk em suas páginas JSP, é necessário o uso da seguinte taglib:

```
<%@taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
```

Componentes web do exemplo

Descreveremos agora os componentes utilizados em nossa aplicação. Veja também o quadro “Mais componentes do Tomahawk”.

Menu

Para menus, utilizamos a tag `<t:jscookMenu>` com o atributo `layout="hbr"`. Isso faz com que o menu seja renderizado horizontalmente (para um menu vertical utiliza-se “vbr”). Usamos também `theme="ThemeOffice"` para que o menu pareça com os do Microsoft Office. Outros temas disponíveis são `ThemeIE`, `ThemeMiniBlack` e `ThemePanel`.

Para criar um menu deve-se utilizar a tag `<t:navigationMenuitem>`, definindo o nome do menu no atributo `itemLabel`. Veja um exemplo na Listagem 4, onde o código que define os menus está realçado em negrito. Observe que os submenus são construídos aninhando-se as tags `<t:navigationMenuitem>`. O resultado deste menu é mostrado também na Figura 2.

Calendário

O calendário usado no exemplo é criado com a tag `<t:inputCalendar>`. Fizemos `renderAsPopup="true"` para que apareça um botão que, ao ser clicado, mostra o calendário. Também definimos `renderPopupButtonAsImage="true"` para que o botão possua uma imagem indicativa. A aplicação da tag `<t:inputCalendar>` pode ser vista na Listagem 5.

Tabela ordenada

O Tomahawk fornece uma extensão da tag `<h:dataTable>` do JSF: `<t:dataTable>`. Entre as funcionalidades adicionais, há a ordenação automática de colunas, o suporte a eventos JavaScript para as linhas e a exibição de tabelas em modo *newspaper* (como um jornal).

Mais componentes do Tomahawk

Aqui exploramos alguns componentes e funcionalidades adicionais do Tomahawk.

Validação

São fornecidos diversos validadores no Tomahawk. O validador `<t:validateEmail>` verifica se o endereço informado é válido sintaticamente. Não sendo válido, é enviada como mensagem de erro a string definida no atributo `detailMessage` da tag. Veja um exemplo onde o campo a ser validado é um `<h:inputText>`:

```
<h:inputText>
  <t:validateEmail detailMessage="Não é um email válido."/>
</h:inputText>
```

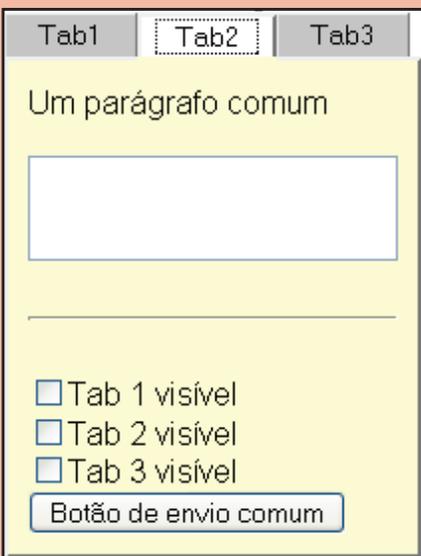


Figura Q1. Exemplo de painel com abas



Figura Q2. Exemplo de uma árvore de dados

O validador de números de cartão de crédito verifica se a quantidade e a estrutura dos números informados são válidos. Veja um exemplo de uso:

```
<h:inputText>
  <t:validateCreditCard detailMessage='
    #{#{0} Não é um cartão de crédito válido.'}/>
</h:inputText>
```

Não existindo um validador predefinido que atenda às suas necessidades, é possível utilizar expressões regulares para definir a regra de validação desejada. O exemplo a seguir exige que o valor do campo tenha entre zero e cinco dígitos:

```
<h:inputText>
  <t:validateRegExpr pattern='\d{5}'
    detailMessage=#{#{0} Campo inválido.'}/>
</h:inputText>
```

Painel com abas

Os painéis com abas são úteis em formulários web com muitos campos de entrada de dados, conforme pode ser visto na Figura Q1.

Para a definição de um painel com abas, são necessárias dois elementos. Uma tag `<t:panelTabbedPane>` cria a região onde serão renderizadas as abas, e tags como `<t:panelTab label="Nome da aba">` definem cada aba. Por exemplo:

```
<t:panelTabbedPane>
  <t:panelTab label="Tab 1">
    <!-- conteúdo da aba Tab1 -->
  </t:panelTab >
  <t:panelTab label="Tab 2">
    <!-- conteúdo da aba Tab2 -->
  </t:panelTab >
</t:panelTabbedPane>
```

Árvore

O Tomahawk possui também o recurso de árvore de dados (*data tree view*), conforme ilustra

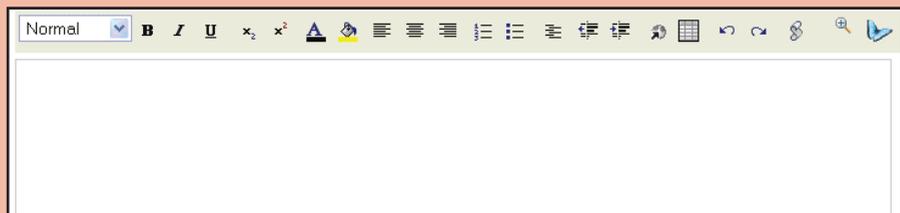


Figura Q3. Exemplo do componente HtmlEditor

a Figura Q2 (extraída do site irian.at/myfaces/tree2HideRoot.jsf).

A tag `<t:tree2 clientSideToggle="false" value="#{managedBean.treeData}">` renderiza uma árvore e obtém os dados de um método chamado `getTreeData()`, implementado no managed bean especificado. Este método é responsável por montar a estrutura de dados da árvore.

Note que, com o atributo `clientSideToggle` da tag `<t:tree2>` ajustado como `false`, a cada clique em um nó da árvore, será feita uma nova requisição para busca dos dados. Ajustando-se `clientSideToggle` para `true`, todos os dados da árvore serão trazidos do servidor no início, e a cada clique em um nó não serão necessárias novas requisições.

O site wiki.apache.org/myfaces/Tree2 fornece mais detalhes sobre as opções disponíveis para árvores e exemplos completos de código.

HtmlEditor

Um dos componentes mais interessantes e sofisticados do Tomahawk é o editor de HTML ilustrado na Figura Q3.

Esse componente possui recursos dignos de qualquer editor de textos básico como suporte a negrito, sublinhado, cores de fontes etc. O seu uso é simples. Basta adicionar a tag `<t:inputHtml>` e ele será renderizado na página.

Outros componentes

Você pode obter mais informações sobre outros componentes do Tomahawk em irian.at/myfaces/home.jsf. Esse site, que é mantido por colaboradores do projeto MyFaces, dá acesso a exemplos utilizando componentes do Tomahawk e do SandBox.

Listagem 4. menu.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<h:outputText styleClass="tituloAplicacao" value="Agenda de
Contatos" />
<h:panelGrid width="500px" style="background-color: #cfc">
  <t:jscookMenu layout="hbr" theme="ThemeOffice">

    <t:navigationMenuItem itemLabel="Cadastro">
      <t:navigationMenuItem itemLabel="Incluir Contato"
        action="incluir" />
      <t:navigationMenuItem itemLabel="Consultar">
        <t:navigationMenuItem itemLabel="Consultar Todos"
          action="consultar" actionListener=
            "#{gerenciadorContato.consultar}" />
    </t:jscookMenu>
  </h:panelGrid>
</f:verbatim>
</br /></br /></br />
</f:verbatim>
```

```
<t:navigationMenuItem
  itemLabel="Consultar Aniversariantes de Hoje"
  action="consultar" actionListener=
    "#{gerenciadorContato.consultarAniversariantes}" />
</t:navigationMenuItem>
</t:navigationMenuItem>

<t:navigationMenuItem itemLabel="Favoritos">
  <t:navigationMenuItems value="#{gerenciadorContato.favoritos}" />
</t:navigationMenuItem>
</t:jscookMenu>
</h:panelGrid>
</f:verbatim>
</br /></br /></br />
</f:verbatim>
```

Listagem 5. incluir.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html><head>
<link href="estilo.css" type="text/css" rel="stylesheet" />

<title></title></head>
<body bgcolor="#ffffff">

<f:view>
  <h:form>
    <%@ include file="menu.jsp"%>
    <h:outputText styleClass="titulo"
      value="Inclusão de Contatos:" />
    <h:panelGrid columns="2">
      <h:outputText value="Nome:" />
      <h:inputText value="#{gerenciadorContato.contato.nome}" />
    </h:panelGrid>
  </h:form>
</f:view>
</body>
</html>
```

```
<h:outputText value="Telefone:" />
<h:inputText value=
  "#{gerenciadorContato.contato.telefone}" />
<h:outputText value="Favorito:" />
<h:selectBooleanCheckbox
  value="#{gerenciadorContato.contato.favorito}" />
<h:outputText value="Aniversario:" />
<t:inputCalendar renderAsPopup="true"
  renderPopupButtonAsImage="true"
  value="#{gerenciadorContato.contato.aniversario}" />
<h:commandButton actionListener=
  "#{gerenciadorContato.incluir}" value="Incluir" />
</h:panelGrid>
</h:form>
</f:view>
</body>
</html>
```

Listagem 6. consultar.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html><head>
<link href="estilo.css" type="text/css" rel="stylesheet" />

<title></title></head>
<body bgcolor="#ffffff">
<f:view>
  <h:form>
    <%@ include file="menu.jsp"%>
    <h:outputText styleClass="titulo"
      value="#{gerenciadorContato.tituloTela}" />
    <t:dataTable cellpadding="0" cellspacing="0" headerClass="tituloTabela"
      rowClasses="linha1, linha2"
      columnClasses="colE, colC, colC, colC" width="500px"
      sortable="true" value="#{gerenciadorContato.contatos}"
      var="contato">

      <t:column>
        <f:facet name="header">
          <h:outputText value="Nome" />
        </f:facet>
        <h:outputText value="#{contato.nome}" />
      </t:column>

      <t:column>
        <f:facet name="header">
          <h:outputText value="Telefone" />
        </f:facet>
        <h:outputText value="#{contato.telefone}" />
      </t:column>

      <t:column>
        <f:facet name="header">
          <h:outputText value="Favoritos" />
        </f:facet>
        <h:outputText rendered="#{contato.favorito}"
          value="sim" />
        <h:outputText rendered="#{!contato.favorito}"
          value="não" />
      </t:column>

      <t:column>
        <f:facet name="header">
          <h:outputText value="Aniversário" />
        </f:facet>
        <h:outputText value="#{contato.aniversario}"
          <f:convertDateTime pattern="dd/MM/yyyy" />
        </h:outputText>
      </t:column>
    </t:dataTable>
  </h:form>
</f:view>
</body>
</html>
```

```
<h:outputText value="Telefone" />
</f:facet>
<h:outputText value="#{contato.telefone}" />
</t:column>

<t:column>
  <f:facet name="header">
    <h:outputText value="Favoritos" />
  </f:facet>
  <h:outputText rendered="#{contato.favorito}"
    value="sim" />
  <h:outputText rendered="#{!contato.favorito}"
    value="não" />
</t:column>

<t:column>
  <f:facet name="header">
    <h:outputText value="Aniversário" />
  </f:facet>
  <h:outputText value="#{contato.aniversario}"
    <f:convertDateTime pattern="dd/MM/yyyy" />
  </h:outputText>
</t:column>
</t:dataTable>
</h:form>
</f:view>
</body>
</html>
```

Fale conosco!

Central de Atendimento

Home ClubeDelphi SQL Magazine MSDN Magazine WebMobile Java Magazine Fórum Cursos

- Alterar Dados
- Renovação
- Entregas
- Quero Assinar
- Minha Assinatura
- Edições Anteriores
- Assinatura Internacional
- Editorial
- Fale Conosco

Bem-vindo à Central de Atendimento

Este é o mais novo serviço do Grupo DevMedia. Agora ficou muito mais fácil acompanhar sua assinatura e retirar qualquer dúvida sobre nossos serviços.

Através da Central do Assinante, você pode consultar o início e o término de sua assinatura, além de acompanhar a data em que suas revistas foram postadas. Pode consultar e alterar, se necessário, seus dados cadastrais. E ainda retirar suas dúvidas sobre renovação, cobranças, edições anteriores e muito mais.

Muito obrigado por ser nosso cliente!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

Dados Cadastrais	
Login:	Ricsilva
Senha:	rsilva2006
Nome:	Ricardo da Silva
Empresa:	Minha Empresa
E-Mail:	ricsilva@email.com
E-Mail Trab:	workgroup@email.com
Endereço:	Rua das Marrecas, 32
Bairro:	Centro
Cidade:	Rio de Janeiro
Estado:	RJ
País:	Brasil
Cep:	12327-500
Tel. Comercial:	21 2654-8697
Tel. Residencial:	21 3879-6123
Tel. Contato:	21 9854-2348

[Editar Cadastro](#)

Assinaturas Ativas	
ClubeDelphi - Revistas Enviadas	
MSDN Magazine - Revistas Enviadas	

Na Central de Atendimento você:

- Consulta a data de envio de cada exemplar
- Consulta as perguntas mais frequentes
- Renova por boleto ou cartão de crédito
- Consulta e altera seus dados cadastrais

DevMedia Sistema de Atendimento On-Line

Boa-tarde Alfredo J. S. Ferreira [Logout](#)

Atendimento

Assunto
Assinatura

Classificação
Dúvida

Área/Setor/Departamento
Atendimento ao Leitor

Descrição
Assinando um ano de ClubeDelphi, quantos exemplares eu teria direito?

Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

Listagem 7. Classe GerenciadorContato

```
package br.com.jm.agenda;

import java.util.*;
import javax.faces.event.ActionEvent;
import javax.faces.model.*;
import org.apache.myfaces.custom.navmenu.NavigationMenuItem;

public class GerenciadorContato {
    private Contato contato = new Contato();
    private DataModel contatos;
    private String tituloTela;

    public List getFavoritos() {
        List resultado = new ArrayList();
        ContatoDao cDao = new ContatoDao();
        List<Contato> favoritos = cDao.consultarFavoritos();
        for (Contato cont : favoritos) {
            StringBuilder label = new StringBuilder(cont.getNome());
            label.append(": ");
            label.append(cont.getTelefone());
            resultado.add(new NavigationMenuItem(label.toString(), ""));
        }
        return resultado;
    }

    public void consultarAniversariantes(ActionEvent e) {
        tituloTela = "Aniversariantes de Hoje";
        ContatoDao cDao = new ContatoDao();
        contatos = new ListDataModel(
            cDao.consultarAniversariantes(new Date()));
    }

    public void consultar(ActionEvent e) {
        tituloTela = "Contatos da Agenda";
        ContatoDao cDao = new ContatoDao();
        contatos = new ListDataModel(cDao.consultar());
    }

    public void incluir(ActionEvent e) {
        ContatoDao cDao = new ContatoDao();
        cDao.incluir(contato);
        contato = new Contato();
    }

    //... getters e setters omitidos
}
```

Listagem 8. faces-config.xml

```
<faces-config>
  <managed-bean>
    <managed-bean-name>gerenciadorContato</managed-bean-name>
    <managed-bean-class>
      br.com.jm.agenda.GerenciadorContato
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

  <navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
      <from-outcome>incluir</from-outcome>
      <to-view-id>/incluir.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>consultar</from-outcome>
      <to-view-id>/consultar.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

Para que seja ativada a ordenação automática, usamos `sortable="true"`. Também fizemos a substituição das tags `<h:column>` por `<t:column>`. O uso das tags de tabelas é demonstrado na **Listagem 6**.

Restante do projeto

Temos ainda um managed bean, **GerenciadorContato**, cujo código está na **Listagem 7**, e o arquivo de configuração do JSF, *faces-config.xml*, mostrado na **Listagem 8**. A classe **GerenciadorContato** possui apenas uma novidade em relação ao JSF padrão. É utilizada a classe **NavigationMenuItem**, no método **getFavoritos()**, para realizar a inclusão dinâmica de contatos no menu *Favoritos*.

Conclusões

O uso de JavaServer Faces torna o desenvolvimento web mais fácil e as aplicações mais ricas e interativas, e a disponibilidade de implementações do JSF com alta qualidade, como o Apache MyFaces Core, aliada ao uso de projetos como o Tomahawk e o Trinidad, aumenta ainda mais esta facilidade. São projetos que trazem para a programação JSF recursos que antes só eram possíveis no mundo das aplicações desktop. ●

myfaces.apache.org

Página oficial do projeto MyFaces

irian.at/myfaces.jsf

Exemplos implementados de componentes do MyFaces



Francisco Calaça Xavier

(francisco@javacerrado.org)

é desenvolvedor Java na Politec (filial Goiânia), instrutor Java certificado pela Borland e fundador da comunidade javacerrado.org. Possui as certificações SCJP, JBuilder Developer e JBuilder Instructor.



Cursos Online



A revista **Java Magazine** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade .

Conheça abaixo o curso já disponível.

Curso em destaque

Introdução ao desenvolvimento para celulares com J2ME

Confira neste curso os principais recursos do **J2ME**. Aprenda também com o passo a passo para criar sua primeira aplicação **J2ME**. Neste curso você irá aprender diversas funcionalidades desta tecnologia para desenvolvimento de dispositivos móveis.

Confira o plano de aula completo:
www.devmedia.com.br/celularesj2me

Assine a **Java Magazine** e comece já seu treinamento!
www.devmedia.com.br/assine

A sua melhor opção de aprendizagem!

Outros cursos disponíveis: www.devmedia.com.br/curso



Mais Informações: www.devmedia.com.br/central - Tel.: 2220-5375 / 2220-5435

Ajax/JPA/EJB3

Mentoring JAVA

Frameworks Open Sources

EJB3
Spring
Struts 2.0
JSF
Laszlo
Ajax4JSF
DWR
Jmeter
JUnit
EMMA
XDoclet
Log4j
Continuum
Cruise Control
Maven
Subversion
Entre outros.

Benefícios:

- Absorção completa dos principais frameworks que aumentam a produtividade
- Utilização dos frameworks em seu ambiente de desenvolvimento
- Combinação das funcionalidades de frameworks distintos
- Total transferência de conhecimento
- Independência total de fornecedores e frameworks proprietários

