Orientação a Objetos e Java

Sérgio Soares
sergio@dsc.upe.br

Pacotes

Objetivo

Depois desta aula você será capaz de desenvolver sistemas mais **reusáveis** e **extensíveis**, organizando as classes do sistema em "módulos" que podem ser analisados, reusados e modificados isoladamente ou com o auxílio de outros poucos "módulos".

Pacotes

Leitura prévia essencial

• Seções 4.6 e 6.5 do livro *Java: how to program* (de Harvey e Paul Deitel)

Tipos de Módulos em Java

- Classes
  - agrupam definições de métodos, atributos, inicializadores, etc.
  - definem tipos
- Pacotes
  - agrupam definições de **classes** relacionadas
  - estruturam sistemas de grande porte, facilitando a localização das classes
  - oferece um nível mais alto de abstração
    - há mais classes do que pacotes

Pacotes e Diretórios

- As classes de um pacote são definidas em arquivos com o mesmo cabeçalho:

```
package nomeDoPacote;
```

- Cada pacote é associado a um diretório do sistema operacional:
  - os arquivos `.class` das classes do pacote são colocados neste diretório
  - é recomendável que o código fonte das classes do pacote também esteja neste diretório

Nomeando Pacotes

- O nome de um pacote é parte do nome do seu diretório associado: o pacote

```
exemplos.banco
```

deve estar no diretório

```
/home/sergio/exemplos/banco
```

assumindo que o compilador Java foi informado para procurar classes em

```
/home/sergio/
```

classpath

Pacotes e Subdiretórios

- Subdiretórios não correspondem a "subpacotes", são pacotes como outros quaisquer
- Por exemplo, não existe nenhuma relação, além de lógica, entre **`exemplos`** e **`exemplos.banco`**:

```
package exemplos;
public class /*...*/
```

```
package exemplos.banco;
public class /*...*/
```

Pacotes e Visibilidade de Declarações

- **`public`**
  - atributos, métodos, inicializadores e classes
  - declaração pode ser utilizada (é visível) em qualquer lugar
- **`private`**
  - atributos, métodos e inicializadores
  - declaração só pode ser utilizada na classe onde ela é introduzida

Pacotes e e Visibilidade de Declarações

- **`protected`**
  - atributos, métodos e inicializadores
  - declaração só pode ser utilizada no pacote onde ela é introduzida, ou nas subclasses da classe onde ela é introduzida
- Ausência de modificador
  - atributos, métodos, inicializadores e classes
  - declaração só pode ser utilizada no pacote onde ela é introduzida

Reuso de Declarações

- As declarações feitas em um arquivo são visíveis em qualquer outro arquivo do mesmo pacote, a menos que elas sejam **privadas**
- Qualquer arquivo de um pacote pode usar as definições **visíveis** de outros pacotes, através do mecanismo de importação de pacotes...

Importação de Pacotes

- Importando definição de tipo específica:

```
package a.b;
import c.d.NomeDoTipo;
public class X { /*...*/ }
```

- Importando todas definições de tipo públicas:

```
package a.b;
import c.d.*;
public class X { /*...*/ }
```

Importação de Pacotes: Detalhes

- Tanto **`NomeDoTipo`** quanto **`c.d.NomeDoTipo`** podem ser usados no corpo de **`a.b`**
- **`a.b`** não pode definir um tipo com nome **`NomeDoTipo`** caso a importação tenha sido específica
- Importação de pacotes não é **transitiva** nem **distribui** sobre os arquivos de um pacote

Estruturando Aplicações com Pacotes

- Agrupar classes relacionadas, com dependência (de implementação ou conceitual) entre as mesmas
- Evitar dependência mútua entre pacotes:

```
package a.b;
import c.d.*;
/*...*/
```

```
package c.d;
import a.b.*;
/*...*/
```

Sugere problemas de modelagem!

Estruturando Aplicações com Pacotes

- Estruturação típica de um sistema de informação:
  - vários pacotes para as classes da GUI, um para cada conjunto de telas associadas
  - um pacote para a classe fachada e exceções associadas
  - um pacote para cada coleção de negócio, incluindo as classes básicas, coleções de dados, interfaces, e exceções associadas
  - um pacote **`sistema.util`** contendo classes auxiliares de propósito geral

Pacotes da Biblioteca de Java

- Acesso a Internet e WWW (`java.net`)
- Applets (`java.applet`)
- Definição de interfaces gráficas (`java.awt`)
- Suporte a objetos distribuídos (`java.rmi`)
- Interface com Banco de Dados (`java.sql`)
- Básicos: threads e manipulação de strings (`java.lang`), arquivos (`java.io`), utilitários de propósito geral (`java.util`)

Exercícios

- Estruture os exemplos de contas e bancos usando pacotes
- Implemente a classe **`ConjuntoContas`** usando a classe **`Vector`** ou a classe **`Hashtable`** (ambas do pacote **`java.util`**).

Pacotes

Resumo

- Importância de estruturar um sistema com pacotes
- Cláusula **`package`**
- Cláusula **`import`**, importação específica e importação genérica
- Pacotes e visibilidade de declarações
- Pacotes da biblioteca de Java

Pacotes

Leitura adicional

- Capítulo 5 do livro *Thinking in Java* (de Bruce Eckel)
- Seções 4.5 e 4.9 do livro *A Programmer's Guide to Java Certification* (de Khalid Mughal e Rolf Rasmussen)
- API de Java em http://www.cin.ufpe.br/~java/docs/jdk1.2.2/docs/api

Orientação a Objetos e Java

Sérgio Soares
sergio@dei.unicap.br
http://www.dei.unicap.br/~sergio

Java versus Delphi

ou Java versus Visual Basic

ou Java versus Visual C++

Objetivo

Depois desta aula você será capaz de avaliar melhor que linguagem de programação é mais adequada para projetos específicos em uma determinada empresa.

Aspectos Técnicos

- Portabilidade e Redução de Custos
- Reusabilidade e Produtividade
- Ambientes de Desenvolvimento
- Arquitetura das Aplicações
- Eficiência
- Linguagem de Programação

Portabilidade

- **Em tese**, Java é portável e independente de plataforma, proporcionando redução de custos com migração, instalação, treinamento, etc.
- **Na prática**, ainda é necessário depurar programas (GUI) antes de migrar para outra plataforma
  - com swing isto não é mais necessário

Portabilidade

- Mas é muito mais fácil migrar sistemas desenvolvidos em Java do que em outras linguagens
- Maior rival: Microsoft! Visual J++, J-Direct

Reusabilidade

- Delphi oferece bem mais componentes reusáveis (ActiveX), proporcionando maior **produtividade**
- JavaBeans pode mudar o cenário a médio prazo...
  - maior **produtividade** e **portabilidade**

Reusabilidade

- Delphi favorece um estilo de programação que pode dificultar reuso de código, além de comprometer confiabilidade e extensibilidade
- E componentes ActiveX podem ser usados para desenvolver programas Java também:

*maior **produtividade** que Delphi caso opte-se por comprometer **portabilidade**!*

Ambientes de Desenvolvimento

- Delphi é mais maduro e estável do que os ambientes disponíveis para Java
- Alguns ambientes para Java atingirão o mesmo nível a médio prazo
- Várias alternativas para Java: Visual Café, JBuilder, Java Workshop, J++, Visual Age for Java, ...

Eficiência de Execução

- Java oferece **alternativas**, **compromissos**
- Código do **cliente** pode ser interpretado
  - compiladores JIT aumentam performance
  - independência de plataforma, cliente universal
  - código móvel, carregado pela rede, evitando instalações, conflito entre versões, etc.
- Código do **servidor** pode ser compilado para código de máquina
  - performance um pouco pior do que C++

Arquitetura da Aplicação

- Desenvolvimento de aplicações distribuídas e cliente-servidor de várias camadas
- Java oferece a opção dos clientes serem carregados pela rede (código móvel), e serem executados em várias plataformas:
  - essencial para administrar a distribuição de sistemas para um grande número de usuários

Linguagem de Programação

- Evolução por **remendos**: Turbo Pascal, Pascal OO, Delphi 1, Delphi 2, Delphi 3, Delphi 4, ...
  - incompatibilidades entre versões (até manipulação de strings!)
  - inconsistências entre paradigmas (OO versus imperativo)

Linguagem de Programação

- Java é bem projetada, sem remendos, e integra de forma **consistente** vários avanços na área de linguagens
  - tipos fortes, coleta de lixo, ausência de ponteiros, robustez, etc.
  - grande impacto em **produtividade**

Aspectos Sociais

Em uma empresa típica...

- Vários programadores foram treinados a usar Delphi, e há alguns gurus
- Por não ter base de OO, programadores antigos acham mais fácil aprender Delphi
- Novos programadores são formados em OO
- Programadores querem aprender e usar Java na prática!

Aspectos Econômicos

- Java é implementada por vários fabricantes e disponível em várias plataformas
- Delphi é implementada por apenas um fabricante (com foco em ferramentas) e disponível para poucas plataformas

Aspectos Econômicos

- Borland, depois Inprise, e agora Borland novamente não tem boa reputação no mercado: não continuidade de produtos
- Delphi ainda tem mais adeptos em alguns lugares, mas tendências da indústria de software mostram claramente que Java veio para ficar

Aspectos Comerciais

- Novos diferenciais de mercado dependentes de segurança, robustez e portabilidade:
  - comércio eletrônico
  - serviços e sistemas de informação disponibilizados via Internet e WWW
  - Java devices; smart cards and appliances; wearable computers!

Java versus Delphi

ou Java versus Visual Basic
ou Java versus Visual C++

Resumo

- Java está pronta para ser usada na prática!
- Java leva vantagem em relação a Delphi nos seguintes aspectos: portabilidade, reusabilidade, extensibilidade, robustez, econômicos e comerciais
- Delphi leva vantagem em relação a Java nos seguintes aspectos: produtividade, ambiente de desenvolvimento, sociais