PACOTES E INTERFACES

PACOTES

- Características
  - A linguagem Java permite agrupar classes um uma coleção chamada pacote ou packages.
  - Os pacotes são convenientes para organizar o trabalho e separá-lo de bibliotecas de código fornecidas por terceiros.
  - Um pacote (package) é uma coleção de classes relacionadas e interfaces que provê acesso restrito (protected) e uma forma de gerenciamento por meio de nomes/pastas.
  - Todos os pacotes padrões da linguagem Java estão contidos na hierarquia do pacote java.
  - Exemplo:
    - java.io.* , java.util.*, javax.swing.*, etc...
  - Podem existir quantos níveis de aninhamento quanto se queira.

PACOTES

- Características
  - Para criar um pacote, é necessário incluir uma classe ou interface.
  - Exemplo:

```
package graphics;
public class Circle extends Graphic {
        ...
}
```

PACOTES

- Nomeando pacotes
  - Por convenção: Empresas usam o domínio da internet em ordem inversa em seus pacotes
    - Exemplo: com.empresa.package.
  - Algumas empresas também acrescentam a região a que pertencem ao nome do pacote.
    - Exemplo: com.empresa.regiao.package.

PACOTES

- Importando pacotes
  - Para importar um membro de um pacote, é usada a instrução import no início do arquivo fonte, antes da declaração da classe.
    - Exemplo: import graphics.Circle;
  - Também pode ser importado todo o pacote
    - Exemplo: import graphics.*;
    - Exemplo: import graphics.A*; //NÃO FUNCIONA!

PACOTES

- Características
  - Obviamente, nos projetos mais complexos, deve-se usar pacotes para organizar melhor o código.
  - Normalmente, importar todas as classe do pacote é mais simples. Isso não tem efeito negativo no tempo de compilação ou no tamanho do código, de modo que não há motivo para não fazê-lo.

INTERFACES

- Características
  - A linguagem Java permite o uso de herança simples, mas não permite a implementação de herança múltipla.
  - Para tentar superar essa limitação o Java faz uso de interfaces.
  - As interfaces podem ser vistas como uma "promessa" que certos métodos com características previamente estabelecidas serão implementados, usando inclusive a palavra reservada implements para garantir esta implementação.

INTERFACES

- Características
  - As interfaces possuem sintaxe similar as classes, no entanto apresentam apenas a especificação das funcionalidades que uma classe deve conter, sem determinar como essa funcionalidade será implementada.
    - Apresentam apenas protótipos dos métodos.
  - Dentro de uma interface, os métodos podem ter somente o seu protótipo, não sendo permitida a sua implementação e devem ser públicos.
  - As interfaces podem ter atributos, mas estes devem ser públicos e finais, o que significa que os atributos usados pelas interfaces serão sempre constantes públicas.
  - Uma interface só pode herdar características de outra interface (uso do extends).
  - Uma classe pode implementar uma ou várias interfaces.

INTERFACES

- Objetivo
  - Estabelecer um padrão a ser seguido para cada uma das rotinas de um sistema/projeto.
  - Especialmente no desenvolvimento por vários grupos de programadores.
  - Quanto maior e mais distribuído se torna o projeto, maior a importância no uso de interfaces.
  - Modularidade, reaproveitamento de código, organização, transparência no desenvolvimento, maior facilidade de manutenção do código, etc...

INTERFACES

- Considerações
  - Uma interface Java é praticamente uma classe abstrata para a qual todos os métodos são implicitamente abstract e public, e todos os atributos são implicitamente public e final.
  - Em outros termos, uma interface Java implementa uma "classe abstrata pura".
  - A sintaxe para a declaração de uma interface é similar àquela para a definição de classes, porém seu corpo define apenas assinaturas de métodos e constantes.

INTERFACES

- Exemplo 1

```
public interface Formulario {
	int altura = 400;
	int largura = 500;
	int posicaox = 200;
	int posicaoy = 200;
	String erro = "...";
	void clickconfirmar();
	void clickcancelar();
	boolean dadosvalidos();
}
```

INTERFACES

- Exemplo II

```
public interface Usuario {
	int maxusuarios = 10;
	void alteranome(String nm);
	void alteraemail(String em);
	void cadastra(String nm, int id);
}
```

INTERFACES

- Exemplo III

```
public class DadosUsuario implements Usuario,Formulario{
        public DadosUsuario(){ implementacao... }
        public void cadastra(String nm, int id){ impl... }
        public void alteranome(String nm){ implementacao }
        public void alteraemail(String em){ implementacao }
        public boolean dadosvalidos(){ implementacao...}
        public void clickconfirmar(){ implementacao... }
        public void clickcancelar(){ implementacao... }
}
```

REFERÊNCIAS

- DEITEL, P.J. Java - Como Programar. Porto Alegre: Bookman, 2001.
- MORGAN, Michael. Java 2 para Programadores Profissionais. Rio de Janeiro: Ciência Moderna, 2000.
- HORSTMANN, Cay, S. e CORNELL, Gary. Core Java 2. São Paulo: Makron Books, 2001 v.1. e v.2.
- NIEMEYER, Patrick. Aprendendo java 2 SDK. Rio de Janeiro: Campus, 2000.