

Interfaces, Classes e Objetos

Dalton Serey
© 2004 DSC/UFCG

A aula de hoje...

- Interfaces como contratos
- Classes: descrições de objetos
- Afinal como é um programa em Java?

Mais de um tipo...

- Cada coisa tem pelo menos um tipo
 - O tipo é apenas a caracterização do que a “coisa” pode fazer ou do que podemos fazer com ela em um determinado contexto
- Problema: há **coisas com mais de um tipo!**
- Exemplo: uma pessoa também pode ser
 - um estudante
 - um filho
 - um empregado, etc.
- O tipo relevante depende do **contexto!**

Mais de um tipo...

- Em Java, o mesmo ocorre
 - um único objeto pode ser de vários tipos
 - isto permite que um objeto possa ser manipulado de várias formas diferentes!
- A questão imediata é:
 - **Se os objetos podem ter mais de um tipo, como um programa sabe qual deles deve usar para tratá-los?**

Mais de um tipo...

- A solução é:
 - **Em cada trecho específico, o programa deve “assumir” que os objetos manipulados são de certo tipo.**
 - Dessa forma, um único objeto pode ser tratado pelos vários tipos a que satisfaz...
- E se o programa deve manipular objetos que serão feitos por outros programadores?

Mais de um tipo...

- A solução para este problema é usar **especificações abstratas de tipos!**
- Ou seja, descrições contratuais...
 - que dizem “o quê” um tipo deve poder fazer
 - mas que nada dizem sobre “como” fazer
- Consequências:
 - programas podem abstrair detalhes dos outros
 - programadores ficam mais independentes
 - a modularidade facilita o desenvolvimento

Interfaces

- Em Java, essas especificações abstratas de tipos são chamadas de Interfaces
- Antes de mais detalhes, eis um exemplo:

```
interface Contador {  
    void reset();  
    void conte();  
    int valor();  
}
```

© 2004 Dalton Serey DSC/UFCG

Interfaces

- Esta interface define o tipo “Contador”
 - ela permite usar contadores, mesmo que não saibamos sua real implementação
- Se `cont` é um “Contador”, podemos fazer:

```
cont.reset();  
while ( cont.valor() != 10 )  
    cont.conte();  
System.out.println(cont.valor());
```

© 2004 Dalton Serey DSC/UFCG

Interfaces

- Detalhes de uma interface:
 - palavra-chave `interface`
 - chaves delimitam a interface
 - declarações de métodos
- Declarações de métodos (assinaturas)
 - tipo de retorno
 - nome
 - tipos dos parâmetros

© 2004 Dalton Serey DSC/UFCG

Sobrecarga (overloading)

- É possível ter mais de um método com o mesmo nome...
- O que diferencia dois métodos é a chamada “pegada” do método...
 - nome
 - número de parâmetros
 - ordem dos tipos de parâmetros
- Tipo de retorno **não** conta!
- Normalmente, é usado para métodos com comportamentos **semelhantes**...

© 2004 Dalton Serey DSC/UFCG

Detalhes de interfaces

- Aspectos que interfaces não garantem:
 - o método faz mesmo o que se espera?
 - como o método será implementado?
 - qual será o tempo de resposta dos métodos?
 - há valores de parâmetros inválidos?
- Solução: documentar de maneira informal!
 - comentários em formato javadoc

© 2004 Dalton Serey DSC/UFCG

Detalhes de interfaces

- Após cada declaração de método, é **obrigatória** a colocação de um **ponto-e-vírgula**
- O acesso aos métodos em uma interface é automaticamente **public**
 - Java não reclama se o forms explícitos e colocarmos o `public` lá
- É costume colocar cada interface em um arquivo separado!
 - interface: `Contador`
 - arquivo: `Contador.java`

© 2004 Dalton Serey DSC/UFCG

Classes

- Se interfaces são especificações abstratas de tipos de objetos, como fazemos os **tipos concretos** em Java?
- Classes são **descrições de tipos de objetos**
- Classes descrevem não apenas o quê, mas como os tipos fazem o que devem fazer

Classes e objetos

- Classes são apenas *descrições*
- São os objetos que fazem o trabalho real, durante a execução
- Durante a execução do programa (nos processos), classes servem como **fábricas de objetos**
 - nada impede que se criem vários objetos de uma mesma classe
- E como criamos objetos?

Instanciação

- A criação de objetos é feita através de uma expressão de instanciação
 - palavra-chave "new"
 - nome da classe (tipo) a instanciar
 - parâmetros de criação do objeto

```
Contador cont;  
cont = new MeuContador();
```

Instanciação

- Os parâmetros de criação dependem do tipo específico que será criado
- O tipo deve ser concreto (uma classe)
 - não faz sentido instanciar interfaces
- O retorno do construtor é uma referência para o objeto criado do tipo da classe indicada

Instanciação

- O trecho de código abaixo cria dois contadores do tipo MeuContador

```
Contador cont1;  
Contador cont2;  
cont1 = new MeuContador();  
cont2 = new MeuContador();
```

Classes

- Afinal, vejamos um exemplo de classe
 - palavra-chave (class), nome (MeuContador)
 - corpo (atributos + métodos)

```
class MeuContador {  
    int valor;  
    void reset() { valor = 0; }  
    void conte() { valor++; }  
    int valor() { return valor; }  
}
```

Classes e Interfaces

- Ao instanciarmos um objeto ele é, naturalmente, do tipo da classe dele
 - se temos `"cont1 = new MeuContador()"`
 - então `cont1` é do tipo `MeuContador`
- Como podemos fazer com que ele possa ser reconhecido também como sendo de um outro tipo?
 - como fazemos `cont1` ser visto como um `Contador` e não como um `MeuContador`?

© 2004 Dalton Serey DSC/UFCG

Classes e Interfaces

- Para isso, a classe a que o objeto pertence (o tipo natural do objeto) deve indicar isso explicitamente...
- Ou seja, devemos declarar que objetos daquele tipo satisfazem ao contrato especificado por certa interface...
 - lembre: interface = especificação de tipo

© 2004 Dalton Serey DSC/UFCG

Classes e Interfaces

- Eis nosso exemplo, corrigido para indicar que objetos do tipo `MeuContador` também são do tipo `Contador`

```
class MeuContador
implements Contador {
    int valor;
    void reset() { valor = 0; }
    void conte() { valor++; }
    int valor() { return valor; }
}
```

© 2004 Dalton Serey DSC/UFCG

Classes e interfaces

- Somente assim, as atribuições abaixo não serão consideradas por Java como um erro de tipos!

```
Contador cont1;
Contador cont2;
cont1 = new MeuContador();
cont2 = new MeuContador();
```

© 2004 Dalton Serey DSC/UFCG

Classes e Interfaces

- Na verdade, há ainda um último "erro" em nosso código...
- Ao declaramos uma interface, declaramos métodos **públicos**
 - significa que outras classes podem "chamá-los"
- Mas, na classe `MeuContador` os métodos foram deixados sem **especificação de acesso**!
 - isto fará Java reclamar ao tentar compilar
 - métodos sem especificação de acesso são de acesso *restrito ao pacote* (voltaremos ao assunto)

© 2004 Dalton Serey DSC/UFCG

Classes e Interfaces

- Eis a classe corrigida...

```
class MeuContador
implements Contador {
    private int valor;
    public void reset() { valor = 0; }
    public void conte() { valor++; }
    public int valor() { return valor; }
}
```

© 2004 Dalton Serey DSC/UFCG

Classes e Objetos

- O que ocorre quando o código executa a linha `"cont1 = new MeuContador();"?`
- Em detalhes:
 - um espaço na memória é alocado
 - espaço suficiente para manter todo estado do objeto (neste caso, para o campo `valor`)
 - a referência para o objeto é retornada e armazenada na variável `cont1`
- O que ocorre se criarmos outros objetos?
 - quantos campos valor existirão?

© 2004 Dalton Serey DSC/UFCG

Classes e Objetos

- Java cuida dos detalhes de construção
- Se for necessário, o programador pode especificar detalhes de construção
- Essa é a idéia de um **construtor**
- É um trecho de código que é executado logo após a construção básica dos objetos
 - tipicamente, é usado para colocar os objetos em um estado inicial mais adequado
- Como escrevemos construtores?

© 2004 Dalton Serey DSC/UFCG

Construtores

- Um construtor em Java tem sempre o mesmo nome da classe
 - não tem valor de retorno
 - não tem tipo de retorno
 - não pode ser invocado diretamente, somente através de expressões `"new"`
 - mas aceitam parâmetros
- Construtores parecem muito com métodos, mas não são métodos...

© 2004 Dalton Serey DSC/UFCG

Construtores

- Um exemplo de construtor...

```
class MeuContador
implements Contador {
    private int valor;
    MeuContador( int valorInicial ) {
        valor = valorInicial;
    }
    public void reset() { valor = 0; }
    public void conte() { valor++; }
    public int valor() { return valor; }
}
```

© 2004 Dalton Serey DSC/UFCG

Construtores

- E agora o que ocorre com nosso código?
- A lista de parâmetros dos construtores está correta?

```
...
cont1 = new MeuContador();
cont2 = new MeuContador();
```

© 2004 Dalton Serey DSC/UFCG

Construtores

- De fato, o código continua correto!
- Java permite a declaração de múltiplos construtores
 - são diferenciados pela lista de parâmetros
- Em nosso código, java chama o construtor *default*... sem parâmetros...

```
...
cont1 = new MeuContador();
cont2 = new MeuContador();
```

© 2004 Dalton Serey DSC/UFCG

Exercício

- Escreva o código de um novo tipo de objeto chamado `ContadorDecrescente` que conte de forma decrescente até zero, e que também atenda à interface `Contador`.
- Escreva todo o código descrito ao longo desta aula em arquivos de texto.
 - arquivo `Contador.java` (interface)
 - arquivo `MeuContador.java` (classe)
 - arquivo `DoisContadores.java` (main)