Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.2

http://www.dcc.ufrj.br/~fabiom/java

Revisão – Classes e Objetos

- Classes são uma das unidades básicas de um programa Java
- Usamos as classes para construir *objetos*: a classe de um objeto dá a sua forma e seu comportamento
- Uma classe possui *campos* (forma), *métodos* (comportamento) e *construtores* (inicialização)

variáveis globais e funções (static)

- Construtores e métodos podem ser *sobrecarregados*: podemos ter vários construtores, e vários métodos com o mesmo nome, contanto que recebam parâmetros diferentes
- Métodos e construtores sempre operam no contexto de um objeto, o parâmetro implícito `this`

Revisão - Identidade

- Como objetos do mundo real, os objetos computacionais possuem *identidade*
- Cada objeto instanciado com `new` tem sua identidade própria e distinta de todas as outras instâncias de sua classe, mesmo que suas *formas* sejam iguais
- Os operadores == e != testam identidade e não igualdade de forma, cuidado!

```
String a = "a";
String b = "b";
System.out.println("ab" == a + b);         // false
System.out.println("ab".equals(a + b));   // true
```

Revisão - Padrões

- Padrões são formas recorrentes de se organizar um programa, descobertos em diversos programas reais, e documentados
- Framework – um framework é uma aplicação com partes faltando, e o programador fornece essas partes
- Mediador – um objeto que coordena a interação entre outros objetos
- Estado – delegar parte da implementação de um objeto para outros objetos chamados de *estados*, permitindo mudar a implementação mudando o estado

Revisão – Outros Padrões

- Decorador – um objeto que modifica o comportamento de outro objeto (provavelmente implementando a mesma interface)
- Compósito – um objeto composto por outros objetos, com o todo implementando a mesma interface das partes
- MVC – separar uma aplicação interativa em três camadas, de modo a desacoplar a lógica e dados da aplicação da sua interface visual
- Observador – um objeto que se inscreve para receber notificações de que alguma parte de outro objeto mudou

Revisão – Classes Parametrizadas e Coleções

- As coleções da biblioteca padrão Java, como `HashSet` (conjuntos) e ArrayList (listas), são *parametrizadas* pelo tipo do elemento: `HashSet<Tijolo>`, `ArrayList<String>`, `HashSet<Integer>`

- Uma forma simples de percorrer uma coleção é com o laço `for` para coleções:

```
for(int x: vetor) {
   System.out.println(x);
}

for(Tijolo t: tijolos) {
   t.desenhar(tela);
}
```

Revisão – Visibilidade

- Em Java, podemos marcar qual a *visibilidade* de um campo ou um método:
  - `public` indica que o acesso é livre
  - `private` indica que o acesso é restrito apenas às instâncias da classe
- Quando não dizemos nada, temos um campo ou método que é público para quem estiver na mesma pasta, e privado para o resto
- Mesmo construtores podem ter visibilidade restrita! Isso é útil para ter maior controle sobre a criação de objetos (limitar o número de instâncias, por exemplo)

Revisão – Interfaces

- Uma *interface* é uma forma abstrata de descrever um objeto
- A classe fixa a forma de um objeto e as assinaturas e as implementações das suas operações
- Uma interface fixa apenas as operações, e quase sempre apenas suas assinaturas
- Sintaticamente, uma interface tem declaracões de métodos, e todos são públicos; um método pode ter corpo apenas se for marcado como `default`

Revisão – Implementando interfaces

- Se quisermos que uma classe pertença à uma interface precisamos *implementá-la*

- A classe deve usar a palavra-chave `implements`, e ter implementações para *todos* os métodos declarados pela interface que não são `default`

- Uma classe pode implementar quantas interfaces ela quiser!

Revisão – Polimorfismo

- Nem todas as linguagens orientadas a objeto possuem interfaces como as de Java, mas todas elas permitem o *polimorfismo* que obtemos com interfaces

- Polimorfismo é poder operar com objetos diferentes de maneira uniforme, mesmo que cada objeto implemente a operação de uma maneira particular; basta que a assinatura da operação seja a mesma para todos os objetos

- Em programas OO reais, é muito comum que todas as operações sejam chamadas em referências para as quais só vamos saber qual classe concreta o objeto vai ter em tempo de execução

- Quase todos os padrões de programação que vimos dependem do polimorfismo para funcionar

Revisão – Recursão estrutural

- Decoradores e compósitos são exemplos de *recursão estrutural*
- Recursão estrutural aparece sempre que uma ou mais partes de um objeto são similares ao todo
- Chamamos métodos que operam sobre essas partes de *métodos recursivos*
- Os métodos `getValor` das classes `Escala`, `Derivada`, `Soma` e `Composta` são exemplos de métodos recursivos

Revisão – Classes anônimas

- Podemos criar uma *classe anônima* que implementa alguma interface, em qualquer lugar que podemos usar uma expressão

```
new Funcao() {
  public double getValor(double x) { ... }
  public String getFormula() { ... }
}
```

- Uma classe anônima pode ter campos, e outros métodos, mas não poderemos acessá-los de fora da classe, mesmo que sejam públicos

- Dentro de uma classe anônima, podemos usar campos e métodos visíveis naquele ponto do código, assim como parâmetros do método corrente

- Para usar o `this` do ponto onde ela foi criada usamos `NomeDaClasse.this`

Revisão – Referências a métodos

- Uma forma mais recente de criar uma classe anônima é com uma *referência a um método*

- Podemos criar uma classe anônima para qualquer interface que tenha apenas um método usando uma referência

- Podemos usar um método com a mesma lista de parâmetros e tipo de retorno do método da interface que queremos implementar:

```
objeto::metodo
```

```
new Acao() {
  public void executa() {
    objeto.metodo();
  }
}
```

- Também podemos fazer referências a funções (`Classe::funcao`), e até mesmo a construtores (`Classe::new`)