Programação Orientada a Objetos

Pacotes, diálogos, construtores e array de objetos

Franciéric Alves – francieric [at] gmail.com

Pacotes

- Utilizados Java para agrupar classes correlatas
- Fornecem uma para o gerenciamento e organização semelhantes a pastas e diretórios
- Evitam conflito de nomes
- Para utilizar classes externas ao pacote atual

Importar pacotes ou classes

- Por padrão, todos os programas Java importam o pacote **java.lang**
  - Exemplos de classes que estão nele: String, Integer, System
- Sintaxe para importar pacotes:

```
import <nomeDoPacote>.<nomeDaClasse>;
import <nomeDoPacote>.<subpacote>. <nomeDaClasse>;
import <nomeDoPacote>.*;
```

- Exemplo:

```
//importa a classe Vector e JOptionPane
import java.util.Vector;
import javax.swing.JOptionPane;
//Importa todas as classes do pacote java.util
import java.util.*
```

3

Criação de pacotes

- Para criar nossos pacotes, escrevemos:

```
package <nomePacote>; // ou
package <nomePacote>.<nomeSubpacote>;
```

- Ex:

```
package br.ifpi.poo.banco;
public class Conta {
    String numero;
    String saldo;
}
```

- Para usar:

```
import br.ifpi.poo.banco.Conta
```

4

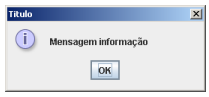
Diálogos

- Diálogos são mensagens que tomam a frente da apliação até que se clique em um botão
- Há mensagens de informação, erro e aviso
- Há diálogos em que só se exibe uma mensagem, outro que exigem tomadas de decisão ou entrada de dados
- A classe javax.swing.JOptionPane possui vários métodos estáticos para criar diálogos

5

Diálogos - Mensagens

- JOptionPane.showMessageDialog(pai, mensagem, titulo, tipoMensagem)
  - Exibem mensagens para o usuário



Ex:

```
JOptionPane.showMessageDialog(null,
    "Mensagem informação",
   "Titulo",
    JOptionPane.INFORMATION_MESSAGE);
```

  - Outros tipos de mensagem:
    - JOptionPane.ERROR_MESSAGE
    - JOptionPane.WARNING_MESSAGE
    - JOptionPane.QUESTION_MESSAGE

6

Diálogos - Confirmação

- JOptionPane.showConfirmDialog(pai, mensagem, titulo, conjuntoBotoes, tipoMensagem)
  - Exibe um diálogo em que o usuário deve clicar em um botão de acordo com o seu interesse
  - Ex:

```
JOptionPane.showConfirmDialog(null,
  "Pergunta?", "Confirme",
  JOptionPane.OK_CANCEL_OPTION,

  JOptionPane.QUESTION_MESSAGE);
```

7

Diálogos - Confirmação

- O showConfirmDialog() retorna um inteiro que corresponde ao botão clicado.
- Há constantes pré-declaradas dentro da própria classe:
  - JOptionPane.CANCEL_OPTION
  - JOptionPane.OK_OPTION
  - JOptionPane.CLOSED_OPTION (fechar a janela sem clicar em nada)

8

Diálogos – Entrada de dados

- JOptionPane.showInputDialog(pai, mensagem, titulo, tipoMensagem)
  - Exibe um diálogo em que o usuário deve preencher uma caixa de texto e clicar em um botão
  - É retornada a String preenchida pelo usuário ou num, caso ele não clique no OK.
  - Ex:

```
JOptionPane.showInputDialog(null,
          "Seu nome:", "Pergunta",
      JOptionPane.QUESTION_MESSAGE));
```

9

Construtor

- É um método especial onde um objeto é inicializado
- É invocado no momento da instanciação após o operador **new**
- Características sintáticas:
  - Possuem o mesmo nome da classe
  - Não retornam valor
  - Podem ter parâmetros

10

Sintaxe

- Construtor comum:

  [modificador] <nomeClasse> (<parâmetros>) {

  <instruções>;

  }

  – Ex:

```
Conta(String numero, double saldoInicial) {
   this.numero = numero;
   saldo = saldo inicial
}
```

11

Construtor padrão

- Construtor padrão:
  - É público e sem argumentos;
  - Pode ser omitido

```
Conta( ){
   ...
}
```

12

Vários construtores

- Podem ser escritos vários construtores
- Isso chama-se sobrecarga e será visto futuramente

```
public class Conta {
    public Conta(String numero){
      this.numero = numero;
    }
    public Conta(String numero, double
      saldoInicial {
       this.numero = numero;
       saldo = saldo inicial
    }
}
```

13

Utilização de construtores

```
...
public static void main(String[] args) {
   Conta c1 = new Conta("1123");
   Conta c2 = new Conta("1123", 200.00);
   Conta c3 = new Conta();
   ...
}
...
```

14

Inicialização de atributos

- Como visto anteriormente, variáveis locais devem ser inicializadas obrigatoriamente antes do uso
- Atributos não inicializados explicitamente, são inicializados pela JVM:
  - Números recebem valores 0
  - Strings e char ficam vazios
  - Booleans com false
  - Outros objetos com nulo

15

Chamando construtores dentro de outro

- Pode-se usar a palavra reservada this para chamar um outro construtor
- Isso deve ser feito na primeira linha do construtor
- Ex:

```
//em uma classe conta...
public Conta() {
    this("-1000", 1.00);
}

public Conta(String numero, double saldoInicial) {
    this.numero = numero;
    saldo = saldoInicial
}
//...em outra classe
public static void main(String[] args) {
    Conta c1 = new Conta();
}
```

16

Array de Objetos

- Todo array de objetos é na verdade um array de referências.

```
Conta[ ] minhasContas;
minhasContas = new Conta[10];
```

- Quantas contas foram criadas?
  - Nenhuma.Foram criadas 10 referências para Contas.
  - Atualmente elas não referenciam nada (null).
  - Caso acessemos, um dos itens, teremos NullPointerException
- Armazenando elementos no array:

```
minhasContas[0] = new Conta(1000);
minhasContas[1] = new Conta(3200);
```

- Os objetos devem ser inicialmente instanciados e só depois atribuídos às referências do array.

17

Trabalhando com Repositórios

- Tipicamente,
  - há classes básicas de um sistema que representam entidades do mundo real
  - classes que gerenciam as classes básicas
    - Gerenciam coleções de classes básicas e são conhecidas como repositórios
- Repositórios tem como funções meios de salvar/ persistir de objetos:
  - Podem ser arrays, listas, árvores, arquivos ou bancos de dados
- Operações comuns são: consultar, adicionar, alterar, excluir

18

"Esqueleto" de um Repositório

```
class Banco {
   void inserir(Conta c) {...}
   void alterar(Conta c) {...}
   Conta consultar(String numero) {...}
   void excluir(numero) {...}
   void creditar(String numero, double valor) {...}
   void debitar(String numero, double valor) {...}
   double consultarSaldo(String numero) {}
   void transferir(String numCred,
                   String numDeb,
                   double valor) {...}
   Banco () {...}
   //...
}
```

19

"Esqueleto" de um Repositório

- Inserir:
  - Insere um objeto no repositório
  - Testa se já **não** existe um repetido
  - Testa se o objeto a ser inserido possui os requisitos mínimos: atributos preenchidos e valores aceitáveis e consistentes
  - Sinônimos: cadastrar, persistir, salvar...
- Alterar:
  - Altera ou substitui um elemento já cadastrado
  - Testa se o objeto **já** existe
  - Testa se o objeto a ser alterado possui os requisitos mínimos: atributos preenchidos e valores aceitáveis e consistentes
  - Sinônimos: persistir, salvar, atualizar...

20

"Esqueleto" de um Repositório

- Consultar:
  - Consulta um objeto por um objeto e o retorna se foi encontrado, do contrário, retorna nulo
  - É utilizado em métodos como inserir, alterar, excluir ...
  - Sinônimos: pesquisar, localizar...
- Excluir:
  - Remove um objeto de uma coleção
  - Testa se o objeto existe
  - Sinônimos: deletar, remover, apagar...

21

"Esqueleto" de um Repositório

- Regras de negócio:
  - Métodos especiais que realizam operações desde que determinadas condições sejam rigorosamente cumpridas:
    - Ex: creditar, debitar, transferir...
    - Emprestar apenas se:
      - Um livro estiver disponível
      - Se o usuário não estiver com um mesmo exemplar alugado
      - Não houver reservas
      - O usuário não tiver mais que N livros já emprestados
      - Etc
- Regras de validação:
  - Métodos que formatos e preenchimento:
    - Campos obrigatórios preenchidos
    - Formatos válidos como em uma placa de carro (LLL-NNNN)
    - CPF preenchidos com dígitos verificadores...

22

Repositório com Arrays

```
public class Banco {
   Conta[ ] contas;
   int indice = 0;

  Banco( ) {
      contas = new Conta[50];
   }

   void inserir(Conta c) {
      contas[indice] = c;
      indice = indice + 1;
   }
}
```

23

Consultar

```
Conta consultar(String numero) {
    Conta c = null;
    for (int i=0; i < indice; i++)) {
       if (contas[i].numero().equals(numero)) {
         c = contas[i];
         break;
       }
    }
    return c;
}
```

24

Debitar

```
void debitar(String numero,
             double valor) {
    Conta c;
    c = consultar(numero);
    if (c != null)
       c.debitar(valor);
    else
       System.out.println("Conta
             Inexistente!");
  }
```

25

Utilizando a classe Banco

```
public class ExecutaBanco {
   public static void main(String args[]) {
      ...
      Banco b = new Banco();
      Conta c1 = new Conta("111", 50.00);
      b.inserir(c1);
      b.debitar(c1.numero, 10.00);
      ....
      Conta c2 = b.consultar("111");
      ...
   }
}
```

26