

Linguagem de Programação Orientada a Objetos II

Revisão de OO1

Prof. Tales Viegas

<https://facebook.com/ProfessorTalesViegas>



Revisão de Conceitos OO

▶ Classes

- *Classes* são estruturas das linguagens de programação OO para conter, para determinado modelo, os dados que devem ser representados e as operações que devem ser efetuadas com estes dados

▶ Objetos

- Um *objeto* ou *instância* é a materialização de uma classe (usada para representar dados e executar ações)
- Para a representação de dados específicos usando classes será necessária a criação de *objetos* ou *instâncias* desta classe



Revisão de Conceitos OO

▶ Métodos

- As operações contidas em uma classe são chamadas de métodos dessa classe
- Métodos são geralmente chamados ou executados explicitamente a partir de outros trechos de código na classe que o contém ou a partir de outras classes

▶ Atributos

- Os dados contidos em uma classe são conhecidos como *campos* ou *atributos* daquela classe
- Este campo deve ter um nome e tipo, que será ou um tipo de dado nativo da linguagem ou uma classe existente na linguagem ou definida pelo programador



Estruturas Fundamentais de Java

► Sintaxe Geral

- Distinção entre maiúsculas e minúsculas
- Nomes de Classe: iniciam em maiúsculas
 - `class Button`, `class NumberFormat`
- Nomes de variáveis: iniciam em minúsculas
 - `int idade`, `float impostoDevido`
- Nomes de métodos: são verbos que iniciam em minúsculas e após usam maiúsculas
 - `imprimirDados()`, `calcularImpostos()`



Classes

```
public class Fruta {  
    int gramas;  
    int caloriasPorGrama;  
  
    public int totalCalorias() {  
        return (gramas*caloriasPorGrama);  
    }  
}
```

← Definição da Classe

← Atributos

← Métodos

Atributos Públicos x Privados

- ▶ Atributos Públicos

- Atributos podem ser acessados e modificados a partir de qualquer classe

- ▶ Atributos Privados

- Atributos só podem ser acessados e modificados a partir de métodos da própria classe que a pertencem



Atributos (públicos e privados)

```
public class Fruta {  
    public int gramas;  
    public int caloriasPorGrama;  
  
    public int totalCalorias() {  
        return (gramas* caloriasPorGrama);  
    }  
}
```

← Atributos Públicos

← Método Público

Atributos (públicos e privados)

```
public class Fruta {  
    private int gramas;  
    private int caloriasPorGrama;  
  
    public int totalCalorias() {  
        return (gramas* caloriasPorGrama);  
    }  
}
```

← Atributos Privados

← Método Público

Métodos

- ▶ Declaração: cabeçalho (interface)
 - valor de retorno
 - nome
 - lista de parâmetros
- ▶ Definição: corpo (código do método)

```
public void setValorCaloriasPorGrama(int valor) {  
    caloriasPorGrama = valor;  
}  
public int getValorCaloriasPorGrama() {  
    return (caloriasPorGrama);  
}
```

Métodos

- ▶ Declaração de um método
 - <acesso><tipo><nome>(<parametros>)
- ▶ Os métodos podem ser:
 - Públicos: podem ser acessados a partir de qualquer classe
 - Privados: só podem ser acessados a partir de métodos da própria classe que a pertencem (proteger métodos que não interessam a outras classes – métodos de implementação)
- ▶ Assinatura de um método
 - Nome + tipos e números de parâmetros (indenpendente do nome das variáveis)
- ▶ Exemplos
 - `void metodo1()` // não têm parâmetro e não retorna nada
 - `public int metodo2()` // retona um dado inteiro
 - `public metodo2(int valor)` // passa um inteiro como parâmetro



Objetos

- ▶ Declaração de classe
 - Define a estrutura de todos os objetos
 - Exemplo: `class Fruta`
- ▶ Declaração de objeto
 - Associa um nome (de objeto) a uma classe
 - Exemplo: `Fruta pera;`
- ▶ Instanciação
 - Criação/inicialização de um objeto
 - Exemplo: `pera = new Fruta();`
- ▶ Declaração + Instanciação
 - Exemplo: `Fruta pera = new Fruta();`



Objetos

```
public class TestaFruta {  
    public static void main(String[] args) {
```

```
        Fruta pera; ← Declarando
```

```
        pera = new Fruta(); ← Instanciando
```

```
        Fruta uva = new Fruta(); ← Declarando  
                                     e Instanciando
```

```
    }
```



Objetos


```
public class Main{  
    public static void main(String[] args) {  
  
        Fruta pera = new Fruta();  
        Fruta uva = new Fruta();  
  
        int calPera, calUva;  
        calPera = pera.totalCalorias();  
        calUva = uva.totalCalorias();  
    }  
}
```



Chamando
métodos



Objetos

```
public class Main{  
    public static void main(String[] args) {  
  
        Fruta pera = new Fruta();  
        Fruta uva = new Fruta();  
  
        pera.gramas = 150;   
  
    }  
}
```

Acesso
Ilegal
(sendo o atributo privado)

Pergunta: e se o atributo for público ?



Operador instanceof

- ▶ Serve para verificar se um objeto é instância de uma determinada classe.
- ▶ Sintaxe :
 - <referência à instância> **instanceof** <nome da classe> : boolean
- ▶ Exemplo :

```
if (pera instanceof Fruta) {  
    System.out.println("pera eh uma instancia de  
    Fruta");  
}
```




Objetos

▶ Objetos podem ser:

- Copiados: fazer uma cópia de cada campo de um objeto em outro objeto
 - Exemplo: `uva.gramas=pera.gramas; //` deverão ser públicos
- Atribuídos: fazer com que um objeto seja substituído por outro objeto
 - Exemplo: `uva = pera;`
 - Diferente de tipos primitivos que copiam valores
- Usados como parâmetros/passados como argumentos/devolvidos como resultados
 - Exemplo: `Fruta somaFrutas(Fruta f) { ... }`

Atribuição de Objetos

```
public class TestaFruta {  
    public static void main(String[] args) {  
  
        Fruta pera = new Fruta();  
        Fruta uva = new Fruta();  
        uva = pera;   
        // pera e uva apontarão para o mesmo objeto  
        // uva e pera são handles ou referências  
  
    }  
}
```

Atribuição de Objetos

```
public class TestaFruta {  
    public static void main(String[] args) {  
        Fruta pera = new Fruta();  
        System.out.println("Id obj pera = " + pera);  
        Fruta uva = new Fruta();  
        System.out.println("Id obj uva = " + uva);  
        uva = pera;  
        // pera e uva apontarão para o mesmo objeto  
        // uva e pera são handles ou referências  
        System.out.println("Id obj uva = " + uva);  
    }  
}
```




Atribuição de Campos

```
public class TestaFruta {  
    public static void main(String[] args) {  
        Fruta pera = new Fruta();  
        Fruta uva = new Fruta();  
        pera.gramas = uva.gramas;  
        pera.caloriasPorGrama = uva.caloriasPorGrama;  
    }  
}
```

Os atributos deverão ser públicos !



Construtores

- ▶ Método executado uma única vez quando um objeto é criado (new)
 - ▶ Nome do construtor = nome da classe
 - ▶ Construtores são sempre *public* e não possuem tipo de retorno (nem mesmo *void*)
 - ▶ Quando nenhum construtor é definido, Java define um construtor padrão
 - ▶ Construtor geralmente fornece valores iniciais (inicialização) para o(s) campo(s)
 - ▶ Pode existir mais de um construtor para cada classe (diferentes assinaturas)
 - Padrão: construtor não parametrizado
 - Demais construtores: diferentes parâmetros
- 

Construtores

```
public class Fruta {  
    private int gramas;  
    private int caloriasPorGrama;  
    // Construtor de Fruta  
    public Fruta(){  
        gramas = 0;  
        caloriasPorGrama = 0;  
    }  
    // demais metodos  
}
```



Método
Construtor sem
parâmetros



Construtores

```
public class Fruta {  
    private int gramas;  
    private int caloriasPorGrama;  
    // Construtor de Fruta  
    public Fruta(int g, int c){  
        gramas = g;  
        caloriasPorGrama = c;  
    }  
    // demais metodos  
}
```




Método
Construtor com
parâmetros



Construtores

```
public class TestaFruta {  
    public static void main(String[] args) {  
        int valorCalorico;  
        Fruta laranja=new Fruta(200,3);  
        valorCalorico=laranja.totalCalorias();  
        System.out.println("Valor calorico:"+ valorCalorico);  
    }  
}
```

Utilizando um Construtor



Palavra Reservada *static*


▶ Variáveis Estáticas

- Variáveis declaradas como *static* são chamadas atributos de classe
- A variável estática será a mesma para todas as instâncias (independente do número de instâncias da classe)
- Exemplos:
 - Se uma instância alterar valor todas as outras instâncias irão detectar esta mudança



Palavra Reservada *static*

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    private static int nroPessoas;  
    Pessoa(String nome, int idade) {  
        nome = nome;  
        idade = idade;  
        nroPessoas ++;  
    }  
    public int getNroInstancias() {  
        return nroPessoas;  
    }  
}
```



Palavra Reservada *static*

- ▶ Métodos definidos como estáticos podem ser utilizados sem que seja necessário criar uma instância da classe à qual pertencem
- ▶ Exemplos :
 - `System.out.println` : `println` é um método estático da classe `System`
 - `public static void main (String[] args)` : é possível executar o método `main`, sem que seja necessário criar uma instância da classe `Principal`



Herança: o que herda ?

- ▶ Subclasse herda:
 - Atributos e métodos públicos e protegidos (protected)
- ▶ Subclasse NÃO herda:
 - Atributos e métodos privados
 - Construtores
 - Métodos de mesma assinatura (redefine)
 - Atributos de mesmo nome (esconde)



Palavras-chave *this* e *super*

- ▶ Palavra-chave *this*
 - É uma referência a própria instância

```
public class Pessoa {  
    protected String nome;  
    protected char sexo;  
  
    public Pessoa(String nome, char sexo) {  
        this.nome = nome;  
        this.sexo = sexo;  
    }  
}
```



Palavras-chave *this* e *super*

- ▶ Palavra-chave *super*
 - Realiza acesso aos métodos ancestrais
 - Permitem aumentar a reutilização de código
 - Na classe filha (Aluno):

```
public void imprimir() {  
    super.imprimir();  
    System.out.println("Matricula: " + matricula);  
}
```

Acesso ao método
da classe
Ancestral (Pessoa)

```
public void imprimir() {  
    System.out.println("Nome: " + nome);  
    System.out.println("Sexo: " + sexo);  
}
```

Palavra reservada *final*

- ▶ Palavra reservada *final*
 - Atributos definidos como final não podem ter seu valor alterado
 - Métodos final não podem sofrer polimorfismo

```
public class Aluno extends Pessoa {  
    private int matricula;  
    private final int numMaximoDisciplinasSemestre = 5;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
    ...  
    public final void teste() {  
        ...  
    }  
}
```

Coleções

- Também chamada container
- É um objeto que representa um grupo de objetos, em outras palavras, é simplesmente um objeto que agrupa múltiplos elementos em um simples unidade
- Coleções são utilizadas para armazenar, recuperar, manipular, e comunicar dados agregados
- Exemplos: baralho (uma coleção de cartas), caixa de correio (uma coleção de cartas), lista telefônica (uma coleção de nomes e telefones)



Coleções

- ▶ Vantagens do uso de coleções:
 - Reduzir o esforço de programação:
 - fornecendo estruturas de dados úteis e algoritmos que não necessitam ser escritos pelo programador
 - Aumentar o desempenho:
 - fornecendo implementações de alto desempenho de estrutura de dados e algoritmos
 - Reduzir o esforço para aprender APIs:
 - eliminar a necessidade de aprender API de coleções *ah hoc*
 - Reduzir o esforço de projetar e implementar APIs:
 - eliminar a necessidade de produzir coleções *ad hoc*



Lista de Objetos (List)

- ▶ Lista de Objetos (List)
 - Arrays com algumas capacidades adicionais, uma das quais é a capacidade de ter seu tamanho modificado de acordo com a necessidade
 - Existe uma interface **List** que declara que métodos podem ser utilizados para manipulação de listas e duas classes que implementam esta interface



Lista de Objetos (List)

▶ Lista de Objetos (List)

- Cada classe possui um mecanismo diferente para a representação interna dos objetos nas listas
- Classes que implementam List
 - ArrayList
 - LinkedList
- Pacote *java.util*
 - `import java.util.ArrayList;`
 - `import java.util.LinkedList;`
 - Consulte
 - <http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html>
 - <http://java.sun.com/j2se/1.5.0/docs/api/java/util/LinkedList.html>



Lista de Objetos (List)

▶ Lista de Objetos (List)

◦ ArrayList

- Implementa a lista internamente como um array e tem desempenho melhor, exceto por operações como inserção e remoção de elementos da lista

◦ LinkedList


- Têm melhor desempenho para as operações de inserção e remoção mas é, em geral, mais lenta para acesso seqüencial aos elementos da lista
- É mais conveniente para implementar pilhas e filas (métodos addFirst, addLast, getFisrt, getLast, removeFisrt, removeLast)



Lista de Objetos (ArrayList)

▶ Exemplo ArrayList (1)

```
import java.util.ArrayList;
import java.util.ListIterator;
public class ArrayListTest {
    public static void main(String[] args) {
        ArrayList pessoas = new ArrayList();
        pessoas.add("Ronaldo");
        pessoas.add("Robinho");
        pessoas.add("Ronaldinho");
        System.out.println("Impressao da ArrayList (usando
                                                                    iterador:");
        ListIterator iterator = pessoas.listIterator();
        while(iterator.hasNext()){
            System.out.println(iterator.next());
        }
    }
}
```




Lista de Objetos (ArrayList)

▶ Exemplo ArrayList (2)

```
import java.util.ArrayList;
import java.util.ListIterator;
public class ArrayListTest {
    public static void main(String[] args) {
        ArrayList pessoas = new ArrayList();
        pessoas.add("Ronaldo");
        pessoas.add("Robinho");
        pessoas.add("Ronaldinho");
        System.out.println("Impressao da ArrayList (usando
                                                                    indices):");

        for(int i=0;i<pessoas.size();i++){
            System.out.println(pessoas.get(i));
        }
    }
}
```



Exceções

- ▶ Capturando um exceção
 - Um exceção deve ser detectada e tratada
 - O código que potencialmente poderá gerar uma exceção é delimitada pelo bloco *try*
 - Cada bloco *try* contém uma ou mais chamadas de método que podem causar uma exceção e cláusulas *catch* para todos os tipos de exceção que o bloco *try* pode tratar



Exemplo de Exceções 1

```
import java.util.*;
```

```
public class TestaExcecoes1 {  
    public static void main(String[] args) {
```



```
try {
```

```
    Scanner teclado = new Scanner(System.in);  
    System.out.println("Quantos anos voce tem ?");  
    String inputLine = teclado.nextLine();  
    int idade = Integer.parseInt(inputLine);  
    idade++;  
    System.out.println("No proximo ano voce tera: " + idade);
```

Capturando
uma exceção



```
}
```

```
catch (NumberFormatException exception){
```

```
    System.out.println("Erro de conversao: " + exception);
```

```
}
```

```
}
```

```
}
```



Exemplo de Exceções 2

```
import java.util.*;
```

```
public class TestaExcecoes2 {  
    public static void main(String[] args) {
```

```
        try {
```

Capturando
múltiplas exceções

```
            Scanner teclado = new Scanner(System.in);  
            System.out.println("Quantos anos voce tem ?");  
            String inputLine = teclado.nextLine();  
            int idade = Integer.parseInt(inputLine);  
            idade++;  
            System.out.println("No proximo ano voce tera: " + idade);
```

```
        } catch (NumberFormatException exception) {
```

```
            System.out.println("Erro de conversao: " + exception);
```

```
        } catch (NoSuchElementException exception) {
```

```
            System.out.println("Erro de I/O: " + exception);
```

```
        }
```

```
    }
```

```
}
```

