



ANHANGUERA  
EDUCACIONAL

# Programação Orientada a Objetos II

## Introdução

Prof. Rodrigo Rocha  
[prof.rodrigorocha@yahoo.com](mailto:prof.rodrigorocha@yahoo.com)

[www.bolinhabolinha.com](http://www.bolinhabolinha.com)



ANHANGUERA  
EDUCACIONAL

## Apresentação

- Prof. Rodrigo Rocha – [prof.rodrigorocha@yahoo.com](mailto:prof.rodrigorocha@yahoo.com)
- Ementa
  - Introdução ao Desenvolvimento de Interfaces Gráficas
  - Herança Múltipla
  - Polimorfismo
  - Classe Abstrata
  - Interfaces
  - Tratamento de Exceções
  - Arquivos



## Bibliografia

- **Livro texto**

- 1) SANTOS, Rafael. **Introdução à Programação Orientada a Objetos usando : Java.** 1<sup>a</sup> ed. Rio de Janeiro: Elsevier, 2008.

- **Complementar**

- DEITEL, H. M. & DEITEL, P. J., LISBOA, C. A . L. **Java, como programar.** 6<sup>a</sup> Ed. São Paulo: Pearson, 2006.



## Metodologia e Avaliação

- **Metodologia**

- Aula expositiva
- Dinâmicas de grupo
- Debates
- Exercício em classe
- Laboratório

- **Avaliação**

- 40%
  - Atividades (3,0) - **ATPS**
  - Avaliação (7,0)
- 60%
  - Atividades (3,0) - **ATPS**
  - Avaliação Oficial (7,0)



## Cronograma

Semana nº.	Tema
1	Apresentação da Disciplina e Metodologia de Trabalho. Revisão de Orientação à Objetos.
2	Introdução ao Desenvolvimento de Interfaces Gráficas (Bibliotecas AWT e Swing, Containers Swing, Botões, Caixas de Texto e Rótulos).
3	Introdução ao Desenvolvimento de Interfaces Gráficas (Gerenciadores de Layout, Tratamento de Eventos).
4	Introdução ao Desenvolvimento de Interfaces Gráficas (Tratamento de Eventos).
5	Revisão (Herança e Polimorfismo). Conceito de Herança Múltipla.
6	Polimorfismo (Chamadas de Métodos Polimórficas, Passagem de Parâmetros Polimórficos).
7	Classe Abstrata (Definição de Métodos Abstratos, Implementação de Classes Abstratas).
8	Classe Abstrata (Definição de Métodos Abstratos, Implementação de Classes Abstratas).
9	Atividades de Avaliação.
10	Interfaces (definição de Contratos de Métodos, Implementação de Interfaces).
11	Interfaces (Implementação de Interfaces, Herança Múltipla Através de Interfaces).
12	Interfaces (Implementação de Interfaces, Herança Múltipla Através de Interfaces).
13	Tratamento de Exceções (Definição dos Mecanismos de Exceções, Exceções Verificadas e Não Verificadas).
14	Tratamento de Exceções (Captura e Tratamento de Exceções, Definição de Novos Tipos de Exceções).
15	Arquivos.
16	Arquivos.
17	Coleções.
18	Prova Escrita Oficial.
19	Exercícios de Revisão.
20	Prova Substitutiva.



## Revisão

- Diferencia maiúsculas de minúsculas
- Nome do arquivo deve ser igual da classe
- Compilar
  - javac NomeDoPrograma.java
    - gerou o .class
  - java NomeDoPrograma
- Comentários
  - //
  - /\* bloco \*/
  - /\*\* java doc \*/
- Início e final de bloco
  - {
  - }



## Revisão

- Escrever na tela
  - System.out.print
  - System.out.println
  - System.out.printf //semelhante ao C
- Entrada de dados
  - `import java.util.Scanner; //utiliza classe Scanner`
  - `Scanner meuScanner = new Scanner(System.in);`
  - `System.out.println(meuScanner.nextLine());`
- Os tipos de entrada podem ser
  - nextInt – inteiros
  - nextDouble – números reais
  - nextLine - texto
  - next - uma palavra
  - findInLine(".").charAt(0) - um caracter



## Declaração de variáveis

- tipo nome\_da\_variável;
  - exemplo: double juros;
- O tipo pode ser:
  - **inteiro**
    - byte -128 até 127
    - short 32768 até 32767
    - int -2147483648 to 2147483647
    - long -9223372036854775808 to 9223372036854775807
  - **real**
    - float -3.4x10<sup>38</sup> to 3.4x10<sup>38</sup>
    - double -1.8x10<sup>308</sup> to 1.8x10<sup>308</sup>
  - **texto**
    - char um caracter
    - String conjunto de caracteres
  - **lógico**
    - boolean verdadeiro ou falso
- Declarando
  - int numero; int numero2 = 10;



## Operadores

### ■ atribuição

- = atribui um valor
- ++ incrementa de 1 em 1, pode ser antes:
  - > (++variável) incrementa antes de usar
  - > (a++variável) usa seu valor, depois incrementa
- -- decrementa de 1 em 1

- += atribui somando
- -= atribui subtraindo
- \*= atribui multiplicando
- /= atribui dividindo

Assignment operator	Sample expression	Explanation	Assigns
Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g



## Operadores

### ■ matemáticos

- \* multiplicação
- / divisão
- % resto
- + adição
- - subtração

### ■ igualdade

- == igual
- != diferente

### ■ relacional

- > maior que
- >= maior ou igual a
- < menor que
- <= menor ou igual a

### ■ lógico

- ! não

A	~A
V	F
F	V

- && e

A	B	A&B
V	V	V
V	F	F
F	V	F
F	F	F

- || ou

A	B	A&B
V	V	V
V	F	F
F	V	F
F	F	F



## Vetores – Criando e inicializando

### ■ Exemplos

- Definindo e criando

```
int meuVetor[ ];
meuVetor = new int[2];
```

- Definindo e criando em uma só linha

```
int meuVetor2[ ] = new int[2];
```

- Criando atribuindo valores

```
int meuVetor3[ ] = { 10, 20 };
```

### ■ Atribuindo valores

- nome\_vetor[elemento] = valor;

```
meuVetor[0] = 10;
meuVetor[1] = 20;
```



## Método com parâmetro

```
//Classe Lapis.java, utilizando o método escreva com parâmetro
public class Lapis {
    public void escrever(String texto) {
        System.out.println("O lápis escreveu isso:"+texto);
    }
    public String mostrarCor() {
        return "azul";
    }
}

// programa TesteLapis.java
class TesteLapis {
    public static void main(String[] args) {
        //Cria o objeto meuLapis
        Lapis meuLapis = new Lapis();
        //chama o método escrever
        meuLapis.escrever(" eu escrevi isso aqui.");
    }
}
```



## Construtores parametrizados

```
// Classe Lapis.java
public class Lapis {
    private String cor;
    Lapis(String qualCor) { //Construtor com parâmetro
        cor = qualCor;
    }
    public void trocarCor(String novaCor) {
        cor = novaCor;
    }
    public String mostrarCor() {
        return cor;
    }
}

//Programa TesteLapis.java
class TesteLapis
{
    public static void main(String[] args)
    {
        Lapis meuLapis = new Lapis("Amarelo");
        System.out.println(meuLapis.mostrarCor());
    }
}
```



## Destruidores

- Em Java são chamados de finalizadores
- Quando é necessário:
  - executar ações antes do objeto deixar de existir
- Garbage Collection - “Coleta de Lixo”
  - Java automaticamente gerencia a memória
  - quando um objeto não é mais referenciado ele é coletado (destruído) automaticamente

```
protected void finalize() {
    // instruções do finalizador
}
```



## Polimorfismo

- Sobre carga de métodos

```
public class Caneta
{
    public escrevaNaTela(String texto) {
        System.out.println(texto);
    }
    public escrevaNaTela(String linha1, String linha2) {
        System.out.println(linha1);
        System.out.println(linha2);
    }
}
```

- Utilizando

```
static public void main(String args[])
{
    Caneta bic1 = new Caneta()
    bic1.escrevaNaTela("estou escrevendo");
    bic1.escrevaNaTela("primeira linha", "segunda linha");
}
```



## Herança

- public class Pessoas {
- private String nome, sobreNome;
- private double peso,altura;
- ....
- 
- public class Alunos **extends Pessoas** {
- private String nomeCurso;
- private double semestre;
- ....
- 
- public class Professores **extends Pessoas** {
- private String nomeEspecializacao;
- private double valorHoraAula;
- private int numeroAulas;
- ....

herança





## Modificadores de Acesso

### ■ Public

- Membros deste tipo são acessíveis onde o programa tenha uma referência a um objeto desta classe ou subclasse

### ■ Private

- Somente são acessíveis dentro da própria classe
- Não são herdados pelas subclasses

### ■ Protected

- Nível intermediário entre public e private
- Os membros protected de uma classe podem ser acessados pela classe, subclasse e outras classes do mesmo pacote (package)



## Chamada Explícita

### ▪ Criando as classes...

- //AVES
  - private String cor, movimento;
  - private int patas;
  - public Aves() {
  - }
  - public Aves(String novaCor, String novoMovimento, int novaPatas) {
    - cor = novaCor;
    - patas=novaPatas;
    - movimento=novoMovimento;
  - }

- //VOADORAS

- public class AvesVoadoras extends Aves {
  - 
  - 
  - public AvesVoadoras(String corAve, int patasAve) {
    - super(corAve, "voar", patasAve);
  - }

- //PAPAGAIOS

- public class Papagaios extends AvesVoadoras {
  - 
  - 
  - public Papagaios(String corPapagaio) {
    - super(corPapagaio, 2);
  - }

- // Quando instanciarmos um objeto do tipo papagaio, automaticamente ele terá 2 patas e seu movimento será voar

chamada explícita  
ao construtor



## Polimorfismo

```
public class Empregados {  
    // atributo protected, pois faço uso deste na subclasse  
    protected double salario;
```

```
        public void setSalario(double novoSalario){  
            salario = novoSalario;  
        }  
        public double getSalario() {  
            return salario;  
        }  
    }
```

polimorfismo

```
public class EmpregadoComissionado extends Empregados {  
    private double valorVendas, comissao;
```

```
        public double getSalario() {  
            return this.salario + (valorVendas*comissao/100);  
        }  
    }
```

polimorfismo



## Exercícios de revisão

1-) Crie uma superclasse abstrata chamada Avaliação, com as seguintes características:

**Atributos:** - nomeDisciplina

- bimestre
- nota
- pesoDaAvaliação

**Métodos:** - "set" e "get" para todos os atributos.

- getNotaReal, deverá retornar o valor de sua nota final ( $nota * peso / 100$ )

**Construtores:**

- sem parâmetros
- receba como parâmetros o bimestre e o pesoDaAvaliação.

2-) Crie a subclasse AvaliaçãoParcial, que além das características herdadas tenha:

**Atributos:** - pontosDeExercícios

**Métodos:** - "set" e "get" para o atributo pontosDeExercícios

- getNotaReal: deverá retornar o valor da nota final ( $nota * peso / 100 + pontos de bonificação$ ).

**Construtor:** - deverá chamar explicitamente o construtor da classe pai, atribuindo o valor "4" para o peso.

**ATENÇÃO:** Tenha preocupação com as entradas dos usuários