Programação Orientada à Objetos (POO)

CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. João Paulo Aramuni

UNIVERSIDADE FUMEC

Sumário

* **Características Avançadas das Classes**
  * Atributos e métodos estáticos
  * Modificador final
  * Classes internas
  * Classes abstratas
  * Interfaces

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * Quando se cria objetos de uma determinada classe, cada objeto tem sua cópia separada com os valores dos atributos definidos para a classe.
  * Em situações em que é necessário que todos os objetos de uma classe compartilhem um mesmo atributo, semelhante ao que ocorre com variáveis globais em linguagens de programação tradicionais, deve-se definir o atributo como estático. Neste caso, o atributo estático funciona como uma variável global da classe.
  * Um **atributo estático** é declarado usando-se a palavra-chave **static** antes do nome do atributo.

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * **Exemplo:**

```
public class QualStatic {
    private static int contEstatico = 0;
    private int contNaoEstatico = 0;
    public QualStatic() {
            contEstatico++;
            contNaoEstatico++; }
    public void incContEstatico() { contEstatico++;}
    public void incContNaoEstatico() {
            contNaoEstatico++; }
    public int leContEstatico() {
            return contEstatico; }
    public int leContNaoEstatico() {
            return contNaoEstatico;}
```

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * **Exemplo:**

```
public class Principal {
   public static void main (String[ ] args) {
    QualStatic obj1 = new QualStatic();
    System.out.println("Cont Estatico obj1 = " + obj1.leContEstatico()); // 1
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 1
    QualStatic obj2 = new QualStatic();
    System.out.println("Cont Estatico obj1 = " + obj1.leContEstatico()); // 2
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 1
    System.out.println("Cont Estatico obj2 = " + obj2.leContEstatico()); // 2
    System.out.println("Cont Nao Estatico obj2 = " + obj2.leContNaoEstatico());  // 1
    obj1.incContEstatico();
    obj1.incContNaoEstatico();
    System.out.println("Cont Estatico obj1 = " + obj1.leContEstatico()); // 3
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 2
    System.out.println("Cont Estatico obj2 = " + obj2.leContEstatico()); // 3
    System.out.println("Cont Nao Estatico obj2 = " + obj2.leContNaoEstatico());  // 1
  }}}
```

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * Assim como atributos estáticos, **métodos estáticos** não precisam de um objeto para serem ativados. Pode-se invocar diretamente um método estático sem necessidade de se criar um objeto.
  * Você usa métodos estáticos em duas situações:
    * Quando um método não precisa acessar o estado do objeto porque todos os parâmetros necessários foram fornecidos como parâmetros explícitos (exemplo **Math.pow**).
    * Quando um método precisa acessar somente campos estáticos da classe.
  * Declaração:
    * Precedido da palavra-chave **static**.
    * Exemplo:
      * **public static double** potencia(**double** x)

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * Ativação:
    * NomeDaClasse.método()
  * Exemplos:
    * Todos os métodos da classe java.lang.Math.
      * **double** cosseno = Math.cos(60);
      * **double** valor = Math.sqrt(144);
    * Método main:
      * **public static void main(String[] args)**
        * Pode ser chamado antes de qualquer objeto existir.
        * Geralmente instancia um objeto aplicação

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * **Exemplo:**

```
public class QualStatic {
    private static int contEstatico = 0;
    private int contNaoEstatico = 0;
    public QualStatic() {
            contEstatico++;
            contNaoEstatico++; }
    public void incContEstatico() { contEstatico++;}
    public void incContNaoEstatico() {
            contNaoEstatico++; }
    public static int leContEstatico() {
            return contEstatico; }
    public int leContNaoEstatico() {
            return contNaoEstatico;}
```

Características Avançadas das Classes

* **Atributos e métodos estáticos**
  * **Exemplo:**

```
public class Principal {
   public static void main (String[ ] args) {
    QualStatic obj1 = new QualStatic();
    System.out.println("Cont Estatico obj1 = " + QualStatic.leContEstatico()); // 1
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 1
    QualStatic obj2 = new QualStatic();
    System.out.println("Cont Estatico obj1 = " + QualStatic.leContEstatico()); // 2
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 1
    System.out.println("Cont Estatico obj2 = " + QualStatic.leContEstatico()); // 2
    System.out.println("Cont Nao Estatico obj2 = " + obj2.leContNaoEstatico());  // 1
    obj1.incContEstatico();
    obj1.incContNaoEstatico();
    System.out.println("Cont Estatico obj1 = " + QualStatic.leContEstatico()); // 3
    System.out.println("Cont Nao Estatico obj1 = " + obj1.leContNaoEstatico());  // 2
    System.out.println("Cont Estatico obj2 = " + QualStatic.leContEstatico()); // 3
    System.out.println("Cont Nao Estatico obj2 = " + obj2.leContNaoEstatico());  // 1
  }}}
```

Características Avançadas das Classes

* **Modificador final**
  * **final** é uma palavra chave em Java que indica que um atributo, um método ou uma classe não podem ser modificados ao longo do restante da hierarquia de descendentes.

  * **final** são usados para definir constantes.
    * Apenas valores dos tipos primitivos podem ser usados para definir constantes.
    * O valor é definido na declaração ou nos **construtores** da classe, e não pode ser alterado.
    * Exemplo: **final** double PI = 3.14;
    * Para objetos e arranjos, apenas a referência é constante (o conteúdo do objeto ou arranjo pode ser modificado).

Características Avançadas das Classes

- **Modificador final**
  - Na lista de parâmetros de um método, final indica que o parâmetro não pode ser modificado.
  - public void exemplo (final int par1, double par2) { ... }
    - par1 não pode ser modificado dentro do método.
  - Para método, final indica que o método não pode ser redefinido (overwide) em classes derivadas.
    - final public void exemplo () { ... }
      - método exemplo não pode ser redefinido em classes herdeiras.
  - Para classe, final indica que a classe não pode ser derivada.
    - final public class Exemplo { ... }
    - public class subExemplo extends Exemplo { ... } //ERRO
      - classe Exemplo não pode ser derivada.

Características Avançadas das Classes

* **Classes internas**
  * Como o próprio nome diz, são classes que são definidas dentro de outra classe.
  * As classes internas tem um relacionamento especial com sua classe externa (classe onde ela está definida), em relação as outras classes.
  * Isto se deve pelo fato de que as classes internas podem acessar os membros privados da classe externa.

Características Avançadas das Classes

* **Classes internas**
  * No mercado de trabalho não é comum encontrarmos classes internas.
  * O padrão adotado, geralmente, é criar um arquivo físico .java para cada classe.

Características Avançadas das Classes

* **Classes internas**
  * Exemplo:

```
class PrimeiroNivel{
  //uma classe de primeiro nivel
}
class PrimeiroNivel2{
  //outra classe de primeiro nivel
  class Interna{
    //esta e uma classe interna, dentro da classe PrimeiroNivel2
  }
}
```

Características Avançadas das Classes

* **Classes abstratas**
  * Classe Abstrata é uma classe que não pode ser instanciada, ou seja, não se pode criar objetos diretamente de uma classe abstrata.

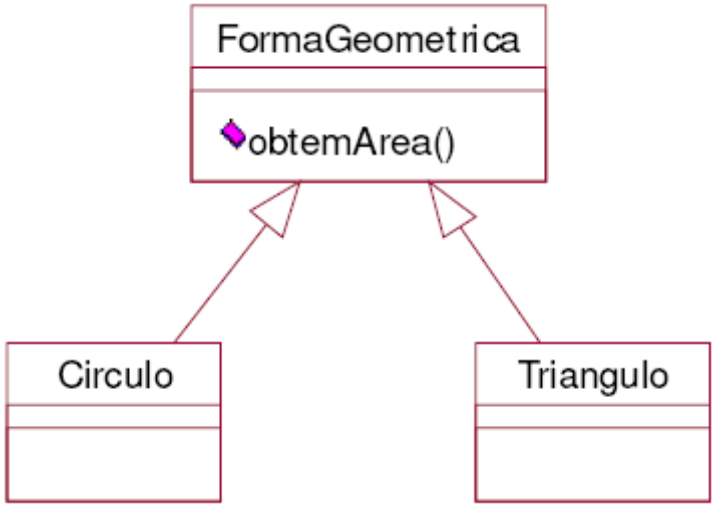
```
abstract public class Exemplo {
    public static void main (String[] args) {
            Exemplo obj = new Exemplo(); //  ERRO
    }
}
```

Características Avançadas das Classes

* **Classes abstratas**
  * **abstract** é a palavra-chave em Java que indica que uma classe ou um método não tem definição concreta.
  * **Classes abstratas** correspondem às especificações genéricas, que deverão ser concretizadas em classes derivadas.

Características Avançadas das Classes

* **Classes abstratas**



* Classes abstratas devem possuir pelo menos uma subclasse para serem utilizáveis.
* No exemplo acima, FormaGeometrica é uma classe abstrata. Todos os objetos ou são círculos ou são triângulos, não existem instâncias diretas de FormaGeometrica.

Características Avançadas das Classes

* **Classes abstratas**
  * Um método abstrato cria apenas uma declaração de um método que deverá ser implementado em uma classe derivada. É um método que só faz sentido para as subclasses.

    * Exemplo, o método **obtemArea**() da classe FormaGeometrica é um **método abstrato**. Só faz sentido obter área de uma forma geométrica concreta como um círculo ou um triângulo.

Características Avançadas das Classes

* **Classes abstratas**
  * Classe com pelo menos um método abstrato é uma classe abstrata.
  * Uma subclasse de uma classe abstrata permanece abstrata, mesmo não sendo declarada explicitamente, até que redefina e implemente todos os métodos abstratos.

Características Avançadas das Classes

* **Classes abstratas**

```
abstract public class FormaGeometrica {
        abstract public double obtemArea();
}
public class Circulo extends FormaGeometrica {
        private double raio;
        . . .
        public double obtemArea() {
          return Math.PI * Math.pow(raio,2);
}}
public class Triangulo extends FormaGeometrica {
        private double lado1, lado2, lado3;
        . . .
        public double obtemArea() {
          double sp = (lado1 + lado2 + lado3) / 2;
          return Math.sqrt(sp*(sp-lado1)*(sp-lado2)*(sp-lado3));
}}
```

Características Avançadas das Classes

* **Classes abstratas e polimorfismo**
  * Um método abstrato é uma promessa de implementação que será cumprida pelas subclasses. Outras classes podem confiar nessa promessa e implementar códigos genéricos usando o conceito de polimorfismo.

```
public class GeoPoli {
 private FormaGeometrica[] fgs;
 . . .
 public void adicionaFormaGeo (FormaGeometrica fg) { . . . }
 public void imprimeAreas() {
                for (int i=0; i<numFigs; i++)
                System.out.println (fgs[i].obtemArea()); // polimorfismo
 }
}
```

Características Avançadas das Classes

* **Regras para projetos para Herança**
  * Operações e Atributos comuns pertencem à superclasse.
  * Use herança para modelar relacionamento “É-UM” somente.
  * Não use herança a menos que todos os métodos herdados façam sentido.
  * Use polimorfismo ao invés de informação de tipo.

Características Avançadas das Classes

* **Classe abstrata - Exemplo**

```
public abstract class Pessoa {
  private String nome;
public Pessoa(String nome) {
    this.nome = nome;
  }
  public String getNome() {
    return nome;
  }
 // Método getDescricao - método abstrato
  public abstract String getDescricao();
}
```

Características Avançadas das Classes

* **Classe abstrata - Exemplo**

```
public class Empregado extends Pessoa {
private double salario;
private Date dataAniver;
public Empregado(String nome, double salario, int ano, int mes, int dia)  {
    // passar nome para o construtor da superclass
    super(nome);
    this.salario = salario;
    GregorianCalendar calendar = new GregorianCalendar(ano, mes - 1, dia);
    dataAniver = calendar.getTime();   }
  public double getSalario() {
    return salario;    }
  public Date getDataAniver() {
    return dataAniver;    }
  // implementação do Método getDescricao
  public String getDescricao()   {
    return String.format("O Empregado tem o salário de $%.2f", salario);    }
  public void reajustaSalario(double porcentagem) {
    double ajuste = salario * porcentagem / 100;
    salario += ajuste;    }
}
```

Características Avançadas das Classes

* **Classe abstrata - Exemplo**

```
public class Estudante extends Pessoa {
     private String curso;
  public Estudante(String nome, String curso) {
    // passar nome para o construtor da superclass
    super(nome);
    this.curso = curso;
  }
   // implementação do Método getDescricao
  public String getDescricao() {
    return "O Estudante está cursando " + curso;
  }
}
```

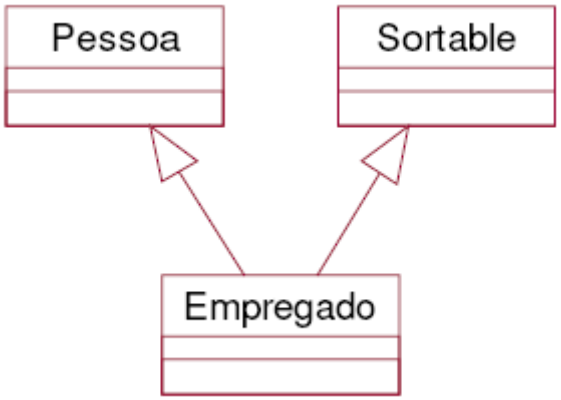
Características Avançadas das Classes

* **Classe abstrata - Exemplo**

```
import java.text.*;
import java.util.*;
public class PessoaClasseAbstrata {
public static void main(String[] args)  {
  Pessoa[ ] pessoal = new Pessoa[4];
     // Preencher o array pessoa com objetos Estudante e Empregado
     pessoal[0] = new Empregado("José da Silva", 5000, 1980, 10, 1);
     pessoal[1] = new Empregado("Eduardo da Silva", 3000, 1982, 12, 1);
     pessoal[2] = new Estudante("Maria das Graças", "Ciência da Computação");
     pessoal[3] = new Estudante("Carlos Alves", "Ciência da Computação");
     // imprimir os nomes e descrições de todos os objetos da Classe Pessoa
     for (int i=0;i<pessoal.length;i++) {
       System.out.println(pessoal[ i ].getNome() + ", " +
                  pessoal[ i ].getDescricao());
        }
  }
}
```

Características Avançadas das Classes

* **Interfaces**
  * Veja o seguinte exemplo:
  * Suponha que a classe Empregado seja também descendente de Sortable. Então tem-se a seguinte hierarquia:



Características Avançadas das Classes

* **Interfaces**
  * Neste caso, Empregado tem duas superclasses, o que caracteriza uma herança múltipla.
  * Herança múltipla ocorre quando uma classe herda de mais de uma superclasse.
  * Java **não** permite herança múltipla de classes, como no exemplo anterior.

Características Avançadas das Classes

* **Interfaces**
  * Java possui apenas um tipo restrito de herança múltipla, onde no máximo uma das superclasses é uma "classe" e as outras são "interfaces".
  * Interface é uma classe "totalmente" abstrata. É uma classe que possui somente métodos abstratos e constantes.
  * Uma interface é declarada usando-se a palavra- chave interface. Não se usa a palavra-chave abstract na declaração dos métodos (todos os métodos já são abstratos).

Características Avançadas das Classes

* **Interfaces**
  * Uma interface é herdada usando-se a palavra- chave implements ao invés de extends. Diz-se que a subclasse implementa a interface, ou seja, implementa os métodos abstratos da interface.

  * Na herança múltipla por exemplo , tem-se:

```
public interface Comparable<T> {
    public int compareTo(T o);
}
public class Empregado extends Pessoa implements
    Comparable<Empregado> { . . . }
```

Características Avançadas das Classes

* **Interfaces**
  * Se empregado também for gravável, tem-se:

    public class Empregado extends Pessoa implements Comparable<Empregado> , Serializable { . . . }

  * Uma interface pode herdar de outra(s) interface(s).

  * A interface geralmente é utilizada como artifício para fazer herança múltipla, já que NÃO é possível fazer extends x, y, z.

  * Além disso, o código se mantem organizado e limpo.

Obrigado.

joaopauloaramuni@gmail.com
joaopauloaramuni@fumec.br

UNIVERSIDADE FUMEC
50 Anos