Java Advanced

Prof. Dr. Marcel Stefan Wagner

Aula 04 – Revisão de POO

FIAP

Tópicos Abordados

1. Introdução à Programação Orientada a Objetos
2. Classes
3. Encapsulamento
4. Herança
5. Abstração
6. Polimorfismo
7. Temas para a Próxima Aula
8. Referências Bibliográficas

Introdução


POLIMORFISMO
HERANÇA
POO
ABSTRAÇÃO
ENCAPSULAMENTO
facebook.com/ateomomento
www.ateomomento.com.br

Encapsulamento

• Permite definir quais partes (classes e/ou atributos) que podem ser acessadas ou não pelos objetos.

• E assim completamos nosso conceito de encapsulamento, em que temos classes dos objetos de negócio com atributos privados (acessíveis somente pela própria classe) e métodos públicos (acessíveis por quaisquer classes), *get* para recuperar os valores dos atributos e métodos públicos *set* para defini-los.

Encapsulamento

Vamos agora começar a declarar os atributos. Neste momento, precisamos pensar quais as informações de aluno que são importantes para o meu sistema. Vamos hipoteticamente criar um sistema no qual queremos armazenar as informações do aluno relativas a nome, endereço, idade, nota de matemática, nota de português e nota de geografia. Teremos então a seguinte definição para nossa classe:

```
1  public class RegistraAluno {
2      private String nome;
3      private String endereco;
4      private int idade;
5      private double notaMatematica;
6      private double notaPortugues;
7      private double notaGeografia;
8  }
```

Vale ressaltar que, quando queremos armazenar alguma informação textual sobre um objeto, o tipo mais ideal do atributo é a classe **String**, como fizemos com o atributo nome e endereço. Além disso, escolhemos o tipo primitivo **int** para armazenar a idade, já que é um valor inteiro, e o tipo primitivo **double** para armazenar as notas, já que estas são todas com precisão de duas casas à direita da vírgula.

Algumas observações de boas práticas:

- As declarações dos atributos são feitas sempre na primeira parte do código da classe.
- Declare um atributo por linha, mesmo que ele seja do mesmo tipo.
- Declare atributo como **private**; assim, somente a própria classe pode manipulá-lo; esse é o conceito de **encapsulamento**, muito importante na Orientação a Objeto.

Programação Orientada a Objetos (POO)

Encapsulamento

```java
public class RegistraAluno {
    private String nome;
    private String endereco;
    private int idade;
    private double notaMatematica;
    private double notaPortugues;
    private double notaGeografia;

    private static int contadorEstudante;

    // retorna o nome do estudante
    public String getNome(){
        return nome;
    }

    // define ou altera o nome do estudate
    public void setNome(String temp){
        nome = temp;
    }

    // retorna o endereço do estudante
    public String getEndereco(){
            return endereco;
    }

    // define ou altera o endereco do estudate
     public void setEndereco(String temp){
         endereco = temp;
    }

    // retorna a idade do estudante
    public int getIdade(){
            return idade;
    }

    // define ou altera idade do estudate
     public void setEndereco(int temp){
         idade = temp;
    }

    // define ou altera as notas
     public void setNotaMatematica(double temp){
        notaMatematica = temp;
    }

    public void setNotaPortugues(double temp){
            notaPortugues = temp;
    }

    public void setNotaGeografia(double temp){
            notaGeografia = temp;
    }

    // retorna a média do estudante
    public double getMedia(){
        double resultado = 0;
        resultado = (notaMatematica + notaPortugues + notaGeografia) / 3;
        return resultado;
    }

    // retorna a quantidade de estudantes cadastrados
    public static int getQuantidadeAlunos(){
        return contadorEstudante;
    }

}
```

Encapsulamento

Por fim, vamos implementar uma classe (classe **AppRegistraAluno**), que representa uma aplicação que utiliza nosso objeto aluno do mundo real, representado computacionalmente pela classe **RegistraAluno:**

```
1  public class AppRegistraAluno {
2      public static void main(String args[]){
3
4          // cria 3 objetos RegistraAluno
5          RegistraAluno ana = new RegistraAluno();
6          RegistraAluno beto = new RegistraAluno();
7          RegistraAluno carlos = new RegistraAluno();
8
9          ana.setNome("Ana Machado");
10         beto.setNome("Roberto da Silva");
11         carlos.setNome("Carlos Alberto");
12
13         System.out.println(ana.getNome());
14
15         System.out.println("Contador: "+RegistraAluno.getQuantidadeAlunos());
16     }
17
18 }
```

Modificadores de Acesso em Java

Modifier	Class	Package	Subclass	World
`public`	✔	✔	✔	✔
`protected`	✔	✔	✔	✘
`no modifier*`	✔	✔	✘	✘
`private`	✔	✘	✘	✘

Herança

• Permite que características comuns a diversas classes sejam “herdadas” de uma classe base, ou superclasse.

• É uma forma de reutilização de *software*.

– Novas classes são criadas a partir das classes existentes, absorvendo seus atributos e comportamentos e adicionando novos recursos.

Herança

Herança

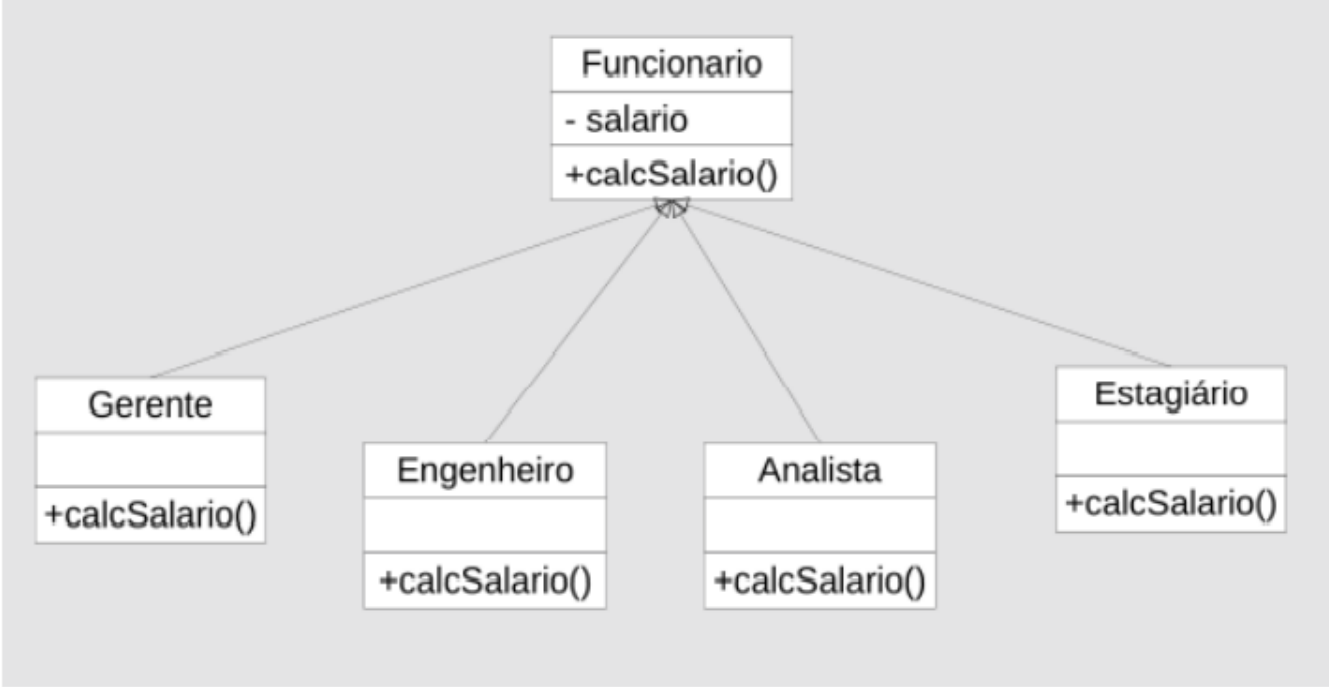


Diagrama de Classes em UML (*Unified Modeling Language*)

Herança em Java

Herança em Java

Encapsulamento

```java
public class Funcionario {

    protected String nome;
    protected double salario;

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salario;
    }
    public void printNome() {
        System.out.println("Nome: " + this.nome);
    }
    public void printSalario() {
        System.out.println("Salário: " + this.salario);
    }
    public void calcSalario() {
        this.salario = 1000;
    }
}
```

Herança em Java

Herança em Java

```java
public class Gerente extends Funcionario {

    public void calcSalario() {
        this.salario = 20000;
    }
}


public class Engenheiro extends Funcionario {

    public void calcSalario() {
        this.salario = 10000;
    }
}
```

Herança em Java

Herança em Java

```java
public class Gerente extends Funcionario {

    public void calcSalario() {
        this.salario = 20000;
    }
}
```

Palavra chave para herança

```java
public class Engenheiro extends Funcionario {

    public void calcSalario() {
        this.salario = 10000;
    }
}
```

Herança em Java

Herança em Java

```
public class Analista extends Funcionario {

    public void calcSalario() {
        this.salario = 5000;
    }
}


public class Estagiario extends Funcionario {

}
```

Herança em Java

Herança em Java

```java
public class Analista extends Funcionario {

    public void calcSalario() {
        this.salario = 5000;
    }
}


public class Estagiario extends Funcionario {

}
```

Não é necessário reimplementar a classe pai

Herança em Java

Herança em Java

```
public static void main(String[] args) {
    Gerente g = new Gerente();
    g.setNome("Pedro")
    g.printNome();
    g.calcSalario();
    g.printSalario();

    Engenheiro e = new Engenheiro();
    e.setNome("Patricia")
    e.printNome();
    e.calcSalario();
    e.printSalario();

    Analista a = new Analista();
    a.setNome("José")
    a.printNome();
    a.calcSalario();
    a.printSalario();

    Estagiario estag = new Estagiario();
    estag.setNome("Julia")
    estag.printNome();
    estag.calcSalario();
    estag.printSalario();
}
```

Herança em Java

Herança em Java

- Saída do programa:

```
Nome: Pedro
Salario: 20000.0
Nome: Patricia
Salario: 10000.0
Nome: José
Salario: 5000.0
Nome: Julia
Salario: 1000.0
```

Herança em Java

Exercícios

- **Exercício 2** - Faça um programa em Java para uma concessionária, onde ela vende carros, motos e caminhões. Utilize herança.

Abstração

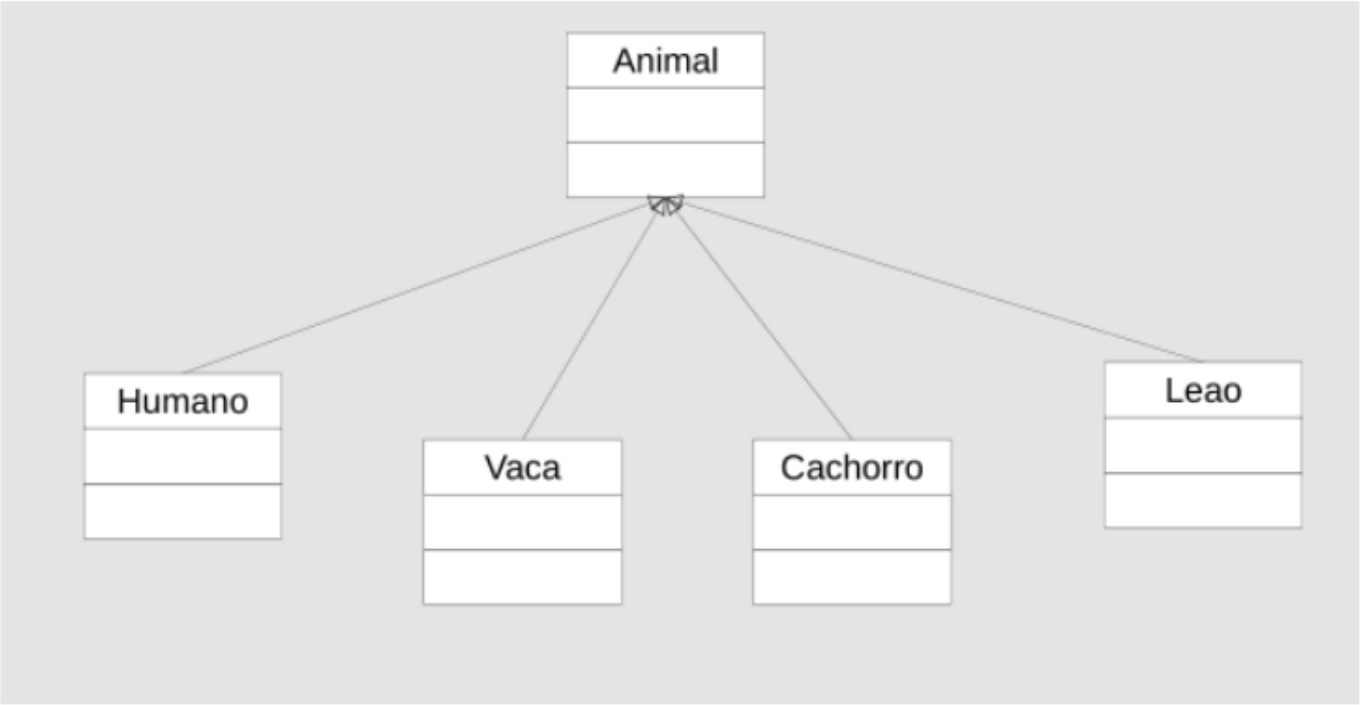
• Essa característica permite que grandes sistemas sejam especificados em um nível muito geral, muito antes de ocorrer a implementação dos métodos individuais.

• Classes que não podem ser instanciadas!

• Permite definir métodos sem implementação - que devem ser redefinidos nas subclasses.

Classes Abstratas

Abstração



Classes Abstratas

Abstração em Java

```
public abstract class Animal {
    public abstract void falar()
}

public class Humano extends Animal {
    public void falar() {
        System.out.println("Eu posso falar - bla bla bla.");
    }
}

public class Vaca extends Animal {
    public void falar() {
        System.out.println("Eu posso mugir - muuuuuu.");
    }
}
```

Classes Abstratas

Abstração em Java

```
public abstract class Animal {
    public abstract void falar()
}

public class Humano extends Animal {
    public void falar() {
        System.out.println("Eu posso falar - bla bla bla.");
    }
}

public class Vaca extends Animal {
    public void falar() {
        System.out.println("Eu posso mugir - muuuuuu.");
    }
}
```

Define a classe abstrata

Define o método abstrato

Classes Abstratas

Abstração em Java

```java
public class Cachorro extends Animal {
    public void falar() {
        System.out.println("Eu posso latir - au au au.");
    }
}

public class Leao extends Animal {
    public void falar() {
        System.out.println("Eu posso rugir - roooooaaaarr.");
    }
}
```

Classes Abstratas

Abstração em Java

```
public class Main {

    public static void main (String[] args) {
        Humano h = new Humano();
        h.falar();

        Vaca v = new Vaca();
        v.falar();

        Cachorro c = new Cachorro();
        c.falar();

        Leao l = new Leao();
        l.falar();
    }
}
```

Classes Abstratas

Abstração em Java

- Será mostrado na tela:

```
Eu posso falar - bla bla bla.
Eu posso mugir - muuuuuu.
Eu posso latir - au au au.
Eu posso rugir - roooooaaaarr.
```

Classes Abstratas

Exercícios

- **Exercício 3** - Faça um programa em Java para um aplicativo de desenho, onde temos uma Forma abstrata (cor e método para calcular a área).
  - Deve-se ter o círculo, quadrado, retângulo e triângulo.

Referências

George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. **Sistemas Distribuídos: Conceitos e Projeto**. Bookman Editora, 5 edition, 2013.

Harvey M Deitel, Paul J Deitel, David R Choffnes, et al. **Sistemas Operacionais**. Pearson/Prentice Hall, 3 edition, 2005.

Maarten Van Steen and A Tanenbaum. **Sistemas Distribuídos: Princípios e Paradigmas**. Pearson/Prentice Hall, 2 edition, 2007.

Harvey M Deitel and Paul J Deitel. **Java, como programar**. Ed. Pearson/Prentice Hall, 8 edition, 2010.

Obrigado!

Agradecimento pela parceria e elaboração de materiais aos professores:
Prof. Me. Gustavo Torres Custódio
Prof. Thiago Yamamoto

Contato: profmarcel.wagner@fiap.com.br

Cursos:
Tecnologia em Análise e Desenvolvimento de Sistemas (TDS)
Tecnologia em Defesa Cibernética (TDC)
Engenharia de Software (ES)