Programação Orientada a Objetos

Herança e Polimorfismo

Odair Jacinto da Silva

Pilares da POO – Abstração (1)

- A abstração é um dos três pilares da Programação Orientada a Objetos (POO).
- Significa literalmente perceber uma entidade em um sistema ou contexto de uma perspectiva específica.
- Descartamos detalhes desnecessários (ou, pelo menos, não nos concentramos neles) e focamos apenas nos aspectos necessários para o contexto ou sistema em consideração (escopo do problema/solução).

Pilares da POO – Abstração (2)

- Abstraction is the concept of **taking some object** from the real world, and **converting it to programming terms**. Such as creating a Human **class** and giving it *int health, int age, String name*, etc. **properties**, and *eat()* etc. **methods**.

Pilares da POO – Encapsulamento (1)

- Descreve a ideia de agrupar dados e métodos que operam/utilizam estes dados em uma classe em Java, por exemplo.
- Utilizado para ocultar a representação interna, ou estado, de um objeto do exterior.
- A ideia geral desse mecanismo é simples: se você possui um atributo que não deve ser visível de fora do objeto, então você “empacota” ele com métodos que fornecem acesso de leitura ou gravação.
- É possível ocultar informações específicas e controlar o acesso ao estado interno do objeto.

Pilares da POO – Encapsulamento (2)

- Para encapsular os atributos devemos modificar seus acessos e implementar métodos get/set que serão responsáveis por permitir acesso aos atributos, de fora do objeto.
- Os modificadores de acesso, de atributos e métodos, em Java, são: sem modificador, private, public e protected.
  - **Private**: Este é o modificador de acesso mais restritivo e mais usado. Se você usar o modificador private com um atributo ou método, **ele poderá ser acessado apenas dentro da mesma classe**. Subclasses ou quaisquer outras classes dentro do mesmo ou de um pacote diferente não podem acessar esse atributo ou método.

Pilares da POO – Encapsulamento (3)

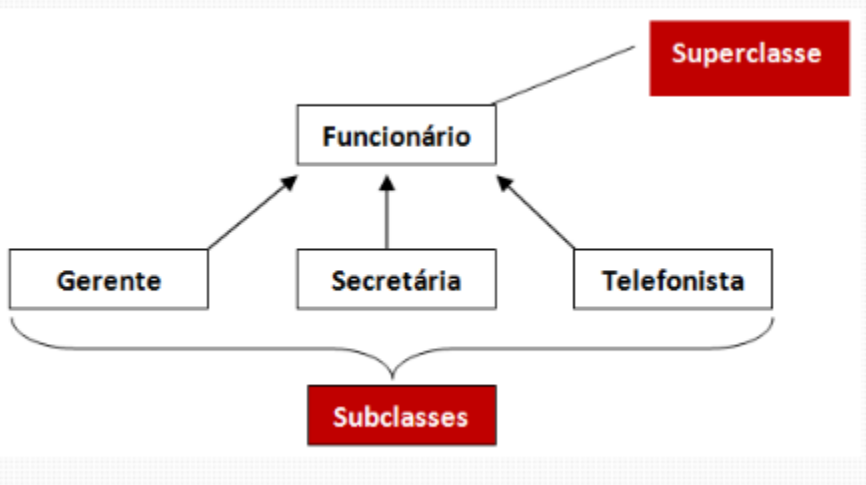
- **Sem modificador**: A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador..
- **Public**: Uma declaração com o modificador public pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.
- **Protected**: Torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	sim	sim	sim	sim
Protected	sim	sim	sim	não
Sem Modificador (Padrão)	sim	sim	não	não
Private	sim	não	não	não

Fonte: Controlling Access to Members of a Class

Pilares da POO – Herança (1)

- Na POO o significado de herança é o mesmo do mundo real.
- Assim como um filho pode herdar alguma característica do pai, na POO é permitido que uma classe herde atributos e métodos da outra.
- Os modificadores de acessos das classes, métodos e atributos devem estar com visibilidade **public** ou **protected** para permitir a herança.



Pilares da POO Herança (2)

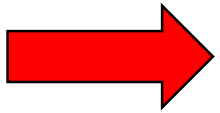
- Superclasse Funcionário

```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 double calculaBonificacao(){
        return this.salario * 0.1;
    }
}
```

Pilares da POO
Herança (3)

- Subclasse Gerente

Método Sobrescrito (overrides)



```java
public class Gerente extends Funcionario {
    protected String usuario;
    protected String senha;

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public  double calculaBonificacao(){
        return this.getSalario() * 0.6 + 100;
    }
}
```

```java
public class testaFuncionario {
    public static void main(String[] args) {
        Gerente gerente = new Gerente();
        gerente.setNome("Carlos Vieira");
        gerente.setSalario(3000.58);
        gerente.setUsuario("carlos.vieira");
        gerente.setSenha("5523");

        Funcionario funcionario = new Funcionario();
        funcionario.setNome("Pedro Castelo");
        funcionario.setSalario(1500);

        System.out.println("##### Gerente #####");
        System.out.println("Nome.: "+gerente.getNome());
        System.out.println("Salário.: "+gerente.getSalario());
        System.out.println("Usuário.: "+gerente.getUsuario());
        System.out.println("Senha.: "+gerente.getSenha());
        System.out.println("Bonificação.: "+gerente.calculaBonificacao());
        System.out.println();

        System.out.println("##### Funcionário #####");
        System.out.println("Nome.: "+funcionario.getNome());
        System.out.println("Salário.: "+funcionario.getSalario());
        System.out.println("Bonificação.: "+funcionario.calculaBonificacao());
        System.out.println();
    }
}
```

Pilares da POO Herança (4)

- Utilizando construtor

no exemplo anterior

- Observar o **construtor**
- Observar que os métodos set

foram removidos

```
public class Funcionario {
    protected String nome;
    protected double salario;

    Funcionario(String nome, double salario){
        this.nome=nome;
        this.salario=salario;
    }

    public String getNome() {
        return nome;
    }

    public double getSalario() {
        return salario;
    }

    public double calculaBonificacao(){
        return this.salario * 0.1;
    }
}
```

Pilares da POO Herança (5)

- Utilizando construtor

no exemplo anterior

- Observar o **construtor**
- Observar que os métodos set

foram removidos

```
public class Gerente extends Funcionario {
    protected String usuario;
    protected String senha;

    Gerente(String nome, double salario,
            String usuario, String senha){
        super(nome, salario);
        this.usuario=usuario;
        this.senha=senha;
    }

    public String getUsuario() {
        return usuario;
    }

    public String getSenha() {
        return senha;
    }

    public  double calculaBonificacao(){
        return this.getSalario() * 0.6 + 100;
    }
}
```

```java
public class testaFuncionario {

    public static void main(String[] args) {
        Gerente gerente = new Gerente("Carlos Vieira",3000.58,"carlos.vieira","5523");
        Funcionario funcionario = new Funcionario("Pedro Castelo",1500);

        System.out.println("##### Gerente #####");
        System.out.println("Nome.: "+gerente.getNome());
        System.out.println("Salário.: "+gerente.getSalario());
        System.out.println("Usuário.: "+gerente.getUsuario());
        System.out.println("Senha.: "+gerente.getSenha());
        System.out.println("Bonificação.: "+gerente.calculaBonificacao());
        System.out.println();

        System.out.println("##### Funcionário #####");
        System.out.println("Nome.: "+funcionario.getNome());
        System.out.println("Salário.: "+funcionario.getSalario());
        System.out.println("Bonificação.: "+funcionario.calculaBonificacao());
        System.out.println();
    }
}
```

Exercício 1

"O desenvolvimento de software é extremamente amplo. Nesse mercado, existem diversas linguagens de programação, que seguem diferentes paradigmas. Um desses paradigmas é a Orientação a Objetos, que atualmente é o mais difundido entre todos. Isso acontece porque se trata de um padrão que tem evoluído muito, principalmente em questões voltadas para segurança e reaproveitamento de código, o que é muito importante no desenvolvimento de qualquer aplicação moderna."

Disponível em: https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264/. Acesso em: 17.11.2018

Considere o programa abaixo escrito na linguagem Java:

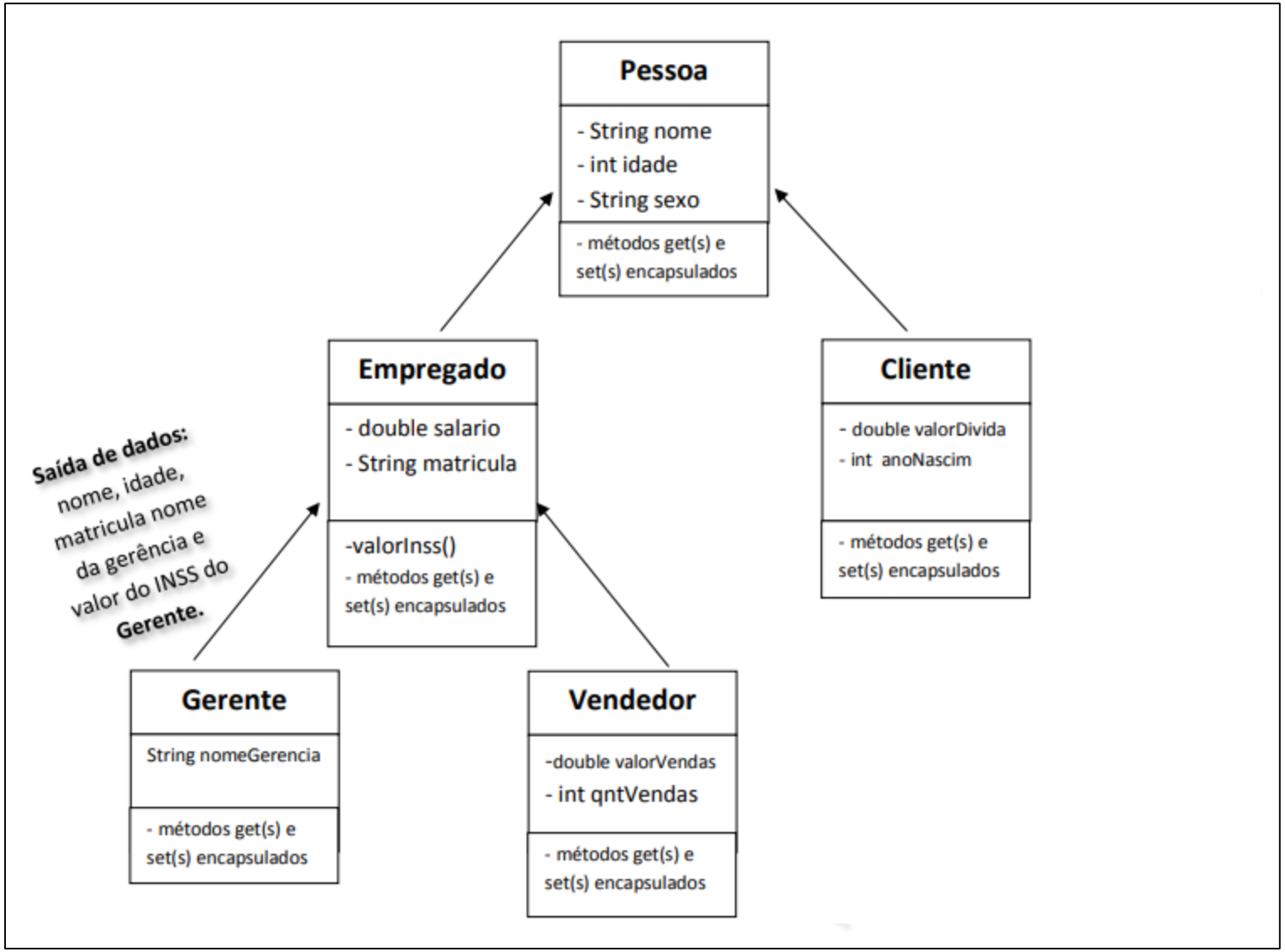
```
Public class veículo {}
Public class carro extends veículo {}
Public class avião extends veículo {}
```

Qual a afirmativa correta?

- ○ **a)** A classe veículo é subclasse da classe avião.
- ○ **b)** A classe avião é subclasse da classe carro.
- ○ **c)** A classe veículo é superclasse das classes carro e avião.
- ○ **d)** As classes carro e avião são superclasses da classe veículo.
- ○ **e)** As classes veículo e carro são subclasses da classe maquinas.

Exercício 2

- Quantos atributos possui um **Gerente**?
  E um **Vendedor**?
- Como seria o **construtor** para Pessoa, Empregado e Vendedor?



Exercício 3

- Crie a classe **Imovel**, que possui um endereço e um preço.
- Crie uma classe **Novo**, que **herda Imovel** e possui um adicional no preço. Crie métodos de acesso e impressão deste valor adicional.
- Crie uma classe **Velho**, que **herda Imovel** e possui um desconto no preço. Crie métodos de acesso e impressão para este desconto.
- Implemente a classe de teste.

Pilares da POO – Polimorfismo (1)

- **Poli** significa muitas e **Morphos** significa forma,
  então **Polimorfismo** significa muitas formas.
- Em **POO** Polimorfismo é caracterizado quando duas ou mais classes possuem métodos com o mesmo nome, mas podendo ter implementações diferentes.
- Assim, é possível utilizar qualquer objeto que implemente o mesmo método sem nos preocuparmos com o tipo do objeto.
- Na prática isso nos possibilita remover do nosso código diversos *if statements* ou *switch cases*.

Pilares da POO – Polimorfismo (2)

- **Exemplo**:
  - Temos uma classe “Motor” que tem o método “Ligar”, agora criamos mais duas classes “MotorElétrico” e “MotorExplosão” que Herdam da classe Motor.
  - As duas classes possuem o evento “Ligar”, mas para cada motor o processo para iniciar o funcionamento é diferente, já que um funciona com gasolina e outro com eletricidade, então o método “Ligar” é polimórfico, nas duas classes seu objetivo é ligar, mas seu funcionamento interno é diferente.
  - Quando criarmos estas duas classes filhas (MotorElétrico, MotorExplosão) adaptamos o método “Ligar” herdado da classe pai (Motor).

Pilares da POO – Polimorfismo (3)

- Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C# e C++, que permite ao desenvolvedor usar o mesmo elemento de formas diferentes.
- Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. No Polimorfismo temos dois tipos:
  - Polimorfismo Estático ou Sobrecarga
  - Polimorfismo Dinâmico ou Sobreposição
- O **Polimorfismo Estático** se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.
- O **Polimorfismo Dinâmico** acontece na **herança**, quando a **subclasse sobrepõe o método original**. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

Pilares da POO – Polimorfismo (4)

- Polimorfismo Estático

```
public class Adicao {

    public int soma(int p1, int p2) {
        return p1+p2;
    }

    public int soma(int p1, int p2, int p3) {
        return p1+p2+p3;
    }

    public int soma(int p1, int p2, int p3, int p4) {
        return p1+p2+p3+p4;
    }
}
```

Pilares da POO – Polimorfismo (5)

- Polimorfismo Estático

```
public class testaAdicao {

    public static void main(String[] args) {
        Adicao a = new Adicao();

        System.out.println("Soma com dois parâmetros: "+a.soma(1,3));
        System.out.println("Soma com três parâmetros: "+a.soma(1,3,5));
        System.out.println("Soma com quatro parâmetros: "+a.soma(1,3,5,7));
    }
}
```

Pilares da POO – Polimorfismo (6)

- Polimorfismo Dinâmico
  - Ver exemplo do método **calculaBonificacao**(), nas classes **Funcionario** e **Gerente** (na seção sobre Herança).

Exercício – Qual a saída no programa principal?

Classe A

```
public class ClasseA {

 public int metodoX(){
   return 10
 }
 public int metodoX(int n){
   return metodoX() + n
 }
}
```

Principal

```
public class Principal {
  public static void main(String[] args) {
    ClasseA obj1 = new ClasseA()
    ClasseA obj2 = new ClasseB()
    ClasseB obj3 = new ClasseB()
    int r = obj1.metodoX() + obj2.metodoX() + obj2.metodoX(5) +
    obj3.metodoX(10, 100);
    System.out.println(r)
    }
 }
```

Classe B

```
public class ClasseB extends ClasseA{

  public int metodoX(){
    return 100
  }
  public int metodoX(int n){
    return metodoX() * n
  }
  public int metodoX(int m, int n){
    return metodoX(m) + metodoX()
  }
}
```

Referências

- Stack Exchange. https://bit.ly/2X4t9NI.
- DevMedia. https://bit.ly/2Q5hnkI.
- WeboPedia. https://bit.ly/36SkaUj.