LCP - Aula 02

Prof. Lucas Guerreiro

Aula anterior

- ➔ Fundamentos de Linguagens Comerciais
- ➔ Conceitos de Orientação a Objetos
- ➔ Características de Java
- ➔ Práticas em Java

Aula 02

Objetivo: Mais conceitos de OO na Prática (Herança, Polimorfismo, Encapsulamento, Abstração). Práticas em Java com modificadores de acesso, classes abstratas e interface.

Herança

- ➔ Conceito de uma classe mais genérica ser herdada para classes mais específicas que compartilham parte de seu comportamento (atributos e métodos), porém com algumas implementações específicas na subclasse.
- ➔ Herança simples: apenas uma super classe
- ➔ Herança múltipla: subclasse origina de mais de uma super classe

Exemplo: Animal -> Mamíferos, Aves

Herança

Exemplo: Funcionários têm nome, salário e cargo. Gerentes têm os mesmos atributos que funcionários, porém estes têm um Bônus de 15% de seu salário.

```
public class Funcionario{
        String nome;
        String cargo;
        float salario;
        void setNome(String nome){...};
        void setCargo(String nome){...};
        void setSalario(String nome){...};
        String getNome(){...};
        String getCargo(){...};
        float getSalario(){...};
}
```

Funcionario.java

```
public class Gerente extends Funcionario{
        double bonus;
        void setBonus(double porcentagem){this.bonus = p_bonus * this.salario};
        float getBonus(){...};
        public static void main(String[] args){
                Gerente gerente = new Gerente();
                gerente.setNome("José");
                gerente.setCargo("Gerente de Vendas");
                gerente.setSalario(5000);
                gerente.setBonus(0.15);
                double valor_bonus = gerente.getBonus();
                System.out.println(valor_bonus);
        }
}
```

Gerente.java

Polimorfismo

- "Muitas formas" -> Métodos de mesmo nome com comportamentos diferentes. Polimorfismo pode ser de sobrecarga ou sobreposição.
    - **Sobrecarga**: reescrita do método dentro da mesma classe. Utilizado para tratar diferentes entradas (tipo ou número de parâmetros diferente).
    - **Sobreposição**: reescrita de um método da super classe na sub classe. Utilizado para implementar um comportamento diferente na classe herdada.

Polimorfismo

Implementar o exemplo de Funcionário/Gerente com Sobrecarga e Sobreposição.

```
public class Funcionario{
        String nome;
        String cargo;
        double salario;
        void setNome(String nome){...};
        void setCargo(String nome){...};
        void setSalario(double salario){...};
        void setSalario(double salario, double p_bonus){...};
        String getNome(){...};
        String getCargo(){...};
        double getSalario(){...};
}
```

Sobrecarga

FuncionarioSobrecarga.java

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

		FuncionarioSobrecarga engenheiro = new FuncionarioSobrecarga();

		engenheiro.setNome("José");

		engenheiro.setCargo("Engenheiro");

		engenheiro.setSalario(5000);

		FuncionarioSobrecarga gerente = new FuncionarioSobrecarga();

		gerente.setNome("João");

		gerente.setCargo("Gerente de Vendas");

		gerente.setSalario(5000.00, 0.15);

		System.out.printf("Salário do %s %s é %.2f", engenheiro.getCargo(), engenheiro.getNome(),
engenheiro.getSalario());

		System.out.println();

		System.out.printf("Salário do %s %s é %.2f", gerente.getCargo(), gerente.getNome(),
gerente.getSalario());

}
```

Sobrecarga

FuncionarioSobrecarga.java

```
public class Funcionario{
        String nome;
        String cargo;
        double salario;
        void setNome(String nome){...};
        void setCargo(String nome){...};
        void setSalario(String nome){...};
        String getNome(){...};
        String getCargo(){...};
        double getSalario(){...};
}
```

Sobreposição

Funcionario.java

```
public class GerenteSobreposicao extends Funcionario{
    void setSalario(float salario, float porcentagem_bonus){...};
    public static void main(String[] args){
        GerenteSobreposicao gerente = new GerenteSobreposicao();
        gerente.setNome("José");
        gerente.setCargo("Gerente de Vendas");
        gerente.setSalario(5000, 0.15);
        double salario_gerente = gerente.getSalario();
    }
}
```

Sobreposição

GerenteSobreposicao.java

```
public class Funcionario{
	String nome;
	String cargo;
	double salario;
	public Funcionario(String nome){
		this.nome = nome;
	}
	public Funcionario(String nome, String cargo){
		this.nome = nome;
		this.cargo = cargo;
	}
	public Funcionario(String nome, String cargo, double salario){
		this.nome = nome;
		this.cargo = cargo;
		this.salario = salario;
	}
	public static void main(String[] args){
		Funcionario func01 = new Funcionario("João");
		Funcionario func02 = new Funcionario("José", "Engenheiro");
		Funcionario func03 = new Funcionario("Maria", "Gerente", 5000);
	}
}
```

Sobrecarga + Construtores

FuncionarioConstrutores.java

Exemplo

```
public class Relogio{
	int hora, minuto, segundo;
	public Relogio(int h, int m, int s){
		this.hora = h;
		this.minuto = m;
		this.segundo = s;
	}
	public Relogio(int h, int m){
		this.hora = h;
		this.minuto = m;
		this.segundo = 0;
	}
	public Relogio(int h){
		this.hora = h;
		this.minuto = 0;
		this.segundo = 0;
	}
	public String getHorario(){
		return this.hora + ":" + this.minuto + ":" + this.segundo;
	}
	public static void main(String[] args){
		Relogio r1 = new Relogio(20, 15, 10);
		Relogio r2 = new Relogio(20, 10);
		Relogio r3 = new Relogio(19);
		System.out.println(r1.getHorario());
	}
}
```

Método 1

Relogio1.java

Exemplo

```
public class Relogio{
	int hora, minuto, segundo;
	public Relogio(int h, int m, int s){
		this.hora = h;
		this.minuto = m;
		this.segundo = s;
	}
	public Relogio(int h, int m){
		this(h, m, 0);
	}
	public Relogio(int h){
		this(h, 0, 0);
	}
	public String toString(){
		return this.hora + ":" + this.minuto + ":" + this.segundo;
	}
	public static void main(String[] args){
		Relogio r1 = new Relogio(20, 15, 10);
		Relogio r2 = new Relogio(20, 10);
		Relogio r3 = new Relogio(19);
		System.out.println("Hora = " + r1);
	}
}
```

Método 2

Relogio2.java

Modificadores - Classe

- Classes são definidas no formato:

[modificadores] ***class NomeClasse*** *[extends SuperClasse] [implements Interface]*

modificadores podem ser:

➔ **public**: implementação comum da classe, com definições feitas na super classe, sub classe ou na própria classe
➔ **abstract**: indica que a classe (e seus respectivos métodos) deverão ser escritos nas implementações das classes herdadas, auxiliando na implementação de abstrações
➔ **final**: indica que a implementação da classe é final, não podendo ser herdada

Modificadores de Acesso

- ➔ **public**: todos têm acesso ao componente
- ➔ **protected**: acesso na própria classe, subclasse ou outras classes dentro do mesmo pacote
- ➔ **private**: acesso restrito dentro da própria classe

Modificador	Classe	Pacote	Subclasse	Mundo
`public`	✅	✅	✅	✅
`protected`	✅	✅	✅	
sem modificador	✅	✅		
`private`	✅			

Modificadores - Método

[modificador_acesso] [static] [abstract] [final] [native] [synchronyzed] ***retorno nomeMetodo*** *([parametros]) [throws excecoes]*

static: método definido da classe

abstract: método deve ser sobrescrito na subclasse

final: método não pode ser sobrescrito pela subclasse

native: permite o uso de um método definido em outra linguagem

synchronyzed: não permite a execução de threads no método

throws excecoes: lança exceções para o método

Método 1

```
public class FuncionarioStatic1 {

	private String nome;
	private String cargo;
	private double salario;
	private static int total_funcionarios = 0;
	public FuncionarioStatic1(String nome){
		this.nome = nome;
		total_funcionarios++;
	}
	public FuncionarioStatic1(String nome, String cargo){
		this.nome = nome;
		this.cargo = cargo;
		total_funcionarios++;
	}
	public FuncionarioStatic1(String nome, String cargo, double salario){
		this.nome = nome;
		this.cargo = cargo;
		this.salario = salario;
		total_funcionarios++;
	}
	public static int getTotalFuncionarios() {
		return total_funcionarios;
	}
```

FuncionarioStatic1.java

Método 1

```
public static void main(String[] args){
        System.out.println("classe: " + FuncionarioStatic1.getTotalFuncionarios());
        FuncionarioStatic1 func01 = new FuncionarioStatic1("João");
        System.out.println("classe: " + FuncionarioStatic1.getTotalFuncionarios());
        System.out.println("func1: " + func01.getTotalFuncionarios());
        FuncionarioStatic1 func02 = new FuncionarioStatic1("José", "Engenheiro");
        System.out.println("classe: " + FuncionarioStatic1.getTotalFuncionarios());
        System.out.println("func1: " + func01.getTotalFuncionarios());
        System.out.println("func2: " + func02.getTotalFuncionarios());
        FuncionarioStatic1 func03 = new FuncionarioStatic1("Maria", "Gerente", 5000);
        System.out.println("classe: " + FuncionarioStatic1.getTotalFuncionarios());
        System.out.println("func1: " + func01.getTotalFuncionarios());
        System.out.println("func2: " + func02.getTotalFuncionarios());
        System.out.println("func3: " + func03.getTotalFuncionarios());
    }

}
```

FuncionarioStatic1.java

Método 2

```
public class Funcionario{
	private String nome;
	private String cargo;
	private float salario;
	private static int total_funcionarios = 0
	public Funcionario(String nome){
		this(nome, "", 0);
	}
	public Funcionario(String nome, String cargo){
		this(nome, cargo, 0)
	}
	public Funcionario(String nome, String cargo, float Salario){
		this.nome = nome;
		this.cargo = cargo;
		this.salario = salario;
		this.total_funcionarios++;
	}
	public static void main(String[] args){
		Funcionario func01 = new Funcionario("João");
		Funcionario func02 = new Funcionario("José", "Engenheiro");
		Funcionario func03 = new Funcionario("Maria", "Gerente", 5000);
	}
}
```

FuncionarioStatic2.java

Modificadores - Variáveis

[modificador_acesso] [static] [final] [volatile] ***tipo nomevariavel***

static: variável da classe

final: definição de uma constante

volatile: permite a alteração da variável de maneira assíncrona

Preferencialmente definir atributos como private e utilizar getters/setters

Exemplo

```
public class Funcionario{
	private final double BONUS_ANUAL = 0.07;
	public Funcionario(){ /* exemplo anterior de Funcionario */}
	public void aumento_anual(){
		this.salario += this.salario * this.BONUS_ANUAL;
	}
	public static void main(String[] args){
		Funcionario func = new Funcionario("José", "Engenheiro", 5000);
		double salario_atual = func.getSalario();
		func.aumento_anual();
		double novo_salario = func.getSalario();
		System.out.printf("Salario anterior = %f, novo salario = %f", salario_atual, novo_salario);
	}
}
```

FuncionarioBonus.java

Abstração

Classes e métodos abstratos indicam que estes deverão ser implementados pelas sub classes.

Não instanciamos objetos de classes abstratas. Elas servem para definir os comportamentos que existirão nas sub classes.

Métodos abstratos não têm corpo na sua definição, somente a assinatura.

Se um método é abstrato a classe deve ser abstrata, mesmo que nem todos os métodos sejam abstratos.

Interface

Interfaces definem quais comportamentos deverão existir nas classes implementadas a partir dela, porém a interface não define o comportamento de nenhum método.

As interfaces definem os métodos, mas sem dizer como eles irão operar.

Os métodos das interfaces são públicos e abstratos.

Pode-se implementar métodos a partir do modificador *default*.

Atributos de interfaces são *final* por padrão.

Pacotes em Java

- ➔ Coleção de classes e interfaces
- ➔ Classes que criamos estão contidas em um pacote
- ➔ Podemos fazer uso de classes externas através de importações de pacotes
- ➔ Importamos apenas classes e interfaces públicas quando usamos outros pacotes (uso de ***import***)
- ➔ **Pacote padrão** e ***java.lang*** são importadas por padrão

Pacotes em Java

java.lang -> já importada por padrão, possui classes básicas utilizadas em Java (String, Math)

java.util -> classes úteis em operações comuns em Java (Dictionary, Hashtable, Vector, etc)

java.io -> operações de entrada e saída (manipulação de arquivos, por exemplo)

java.swing -> interface gráfica

java.awt -> interface gráfica

java.applet -> pacotes para manipulação de applets

Exemplo

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

    double number = 3.1415;

    System.out.println(Math.abs(-number) + " " + Math.round(number) +
Math.sqrt(number));

}
```

Exemplo

Calcular as raízes de uma equação do segundo grau.

- Receber do usuário os valores de a, b e c
- Passar os valores no construtor do objeto
- Criar método (de retorno boolean) que calcula e faz o set dos valores e dois métodos que retornam os valores (x1 e x2)
- No método de cálculo: se for um delta negativo, não realizar o cálculo e retornar isso à função principal e encerrar o programa sem a chamada ao método de retorno/exibição