Orientação a objetos

Herança

Roni Schanuel
04-05-2023

Firjan SENAI

RECAPITULANDO

O que vimos até agora

- O que é Java
- Eclipse IDE
- Nosso primeiro código em Java : “Olá Mundo!”
- Variáveis Primitivas e Controle de Fluxo
- Orientação a objetos básica
- Modificadores de Acesso e Atributos de Classe
- Escopo de Variáveis
- O atributo “static”

EXERCICIO – AULA ANTERIOR

1) Criar uma classe pedido com os seguintes atributos:

- **numero (int)**
- **dataPedido (LocalDate)**
- **quantidade (double)**
- **valor (double)**
- **total (double)**

Inserir o construtor com os atributos **numero, dataPedido, quantidade e valor**
Inserir os getters.

Criar um método com o nome **finalizarPedido** na classe **Pedido,** caso o dia do pedido for um domingo o cliente terá um desconto de 10% no valor do pedido. O total do pedido será a quantidade * valor com o desconto aplicado.

Criar 3 instâncias em uma nova classe com o **main**

Finalizar o pedido

Mostrar o total dos pedidos

HERANÇA - ILUSTRANDO COM EXEMPLO

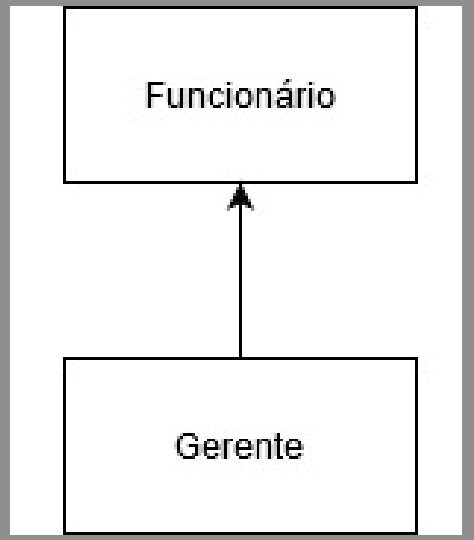
- Um banco possui contas, clientes e funcionários
- Como seria a classe funcionário?

```
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;
    // métodos e construtores
}
```

HERANÇA - ILUSTRANDO COM EXEMPLO

- O Gerente é um funcionário especial
  - Além do funcionário comum, temos outros cargos, como os gerentes. Naturalmente, eles têm informações em comum com os demais funcionários e outras informações exclusivas

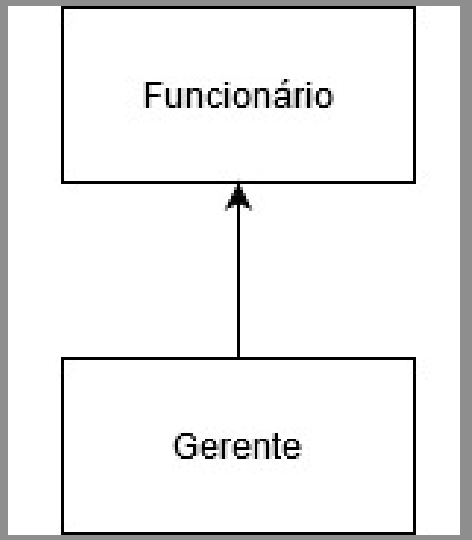
```
public class Gerente {
    // declaração de nome, cpf e salário omitidas
    private int senha;
    private int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        // implementação do método
    }
}
```



HERANÇA - ILUSTRANDO COM EXEMPLO

- Estendendo a classe Funcionario
  - Em Java, existe um jeito de relacionarmos uma classe de tal maneira que uma delas herda tudo que a outra possui. Em nosso caso, queremos que Gerente possua todos os métodos e atributos de Funcionário.
  - Para isso utilizamos a cláusula **extends** na definição da classe.

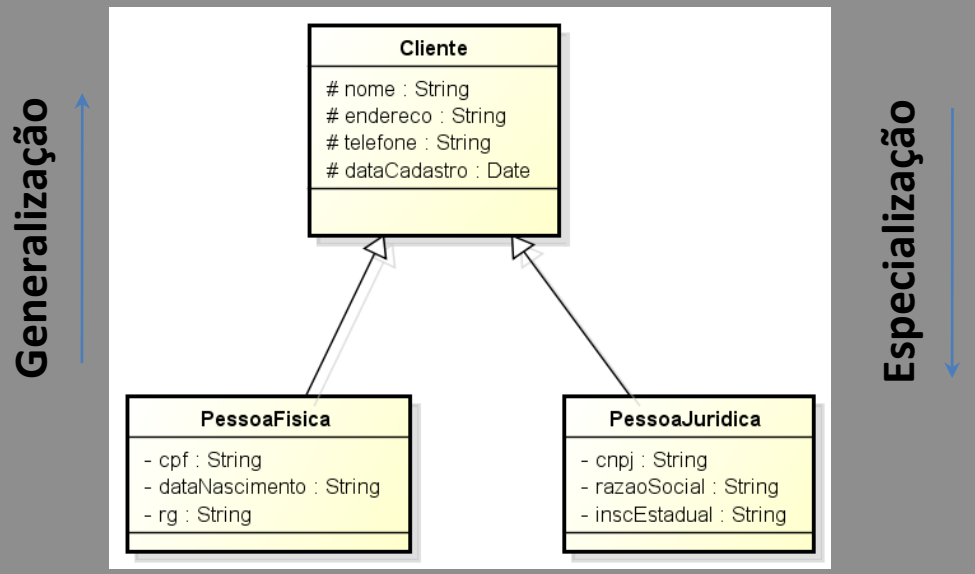
```
public class Gerente extends Funcionario{
    // declaração de nome, cpf e salário omitidas
    private int senha;
    private int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        // implementação do método
    }
}
```



GENERALIZAÇÃO/ESPECIALIZAÇÃO

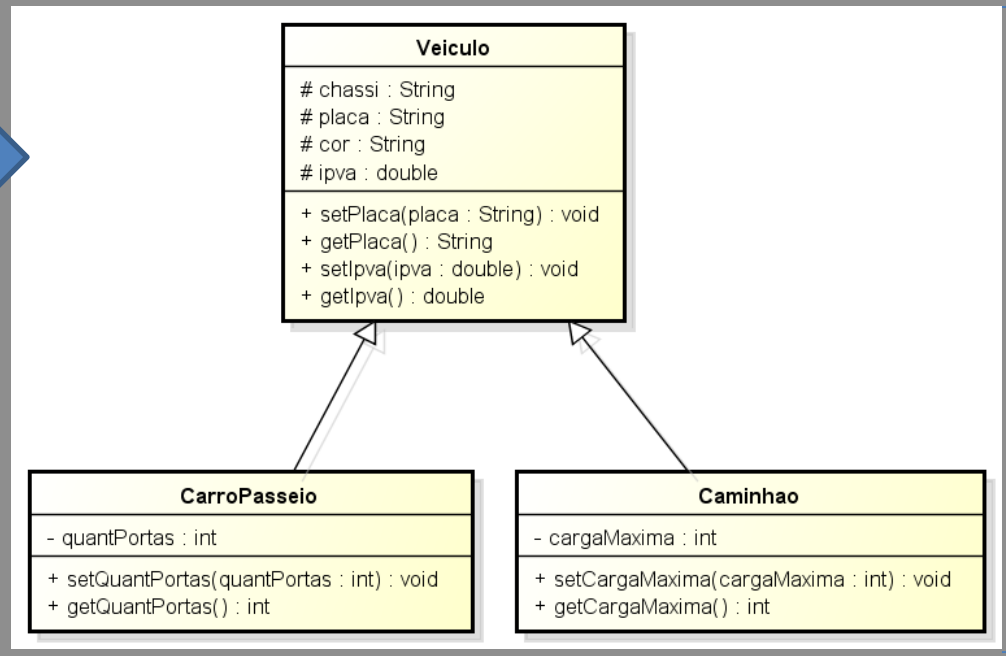
- A **generalização** indica que uma classe mais geral, a superclasse, tem atributos, operações e associações comuns que são compartilhados por classes mais especializadas, as subclasses. O objetivo dessa operação é a criação de uma classe genérica que representará os atributos e métodos existentes em duas ou mais classes específicas.

- A **especialização** se caracteriza pela criação de uma ou mais classes específicas a partir de uma classe genérica para representar atributos e métodos que são distintos entre elas.



HERANÇA

- Quando trabalhamos com várias classes e algumas classes tem características em comum, essas características podem ser colocadas em uma classe base ou super classe. A partir de uma classe base podemos criar subclasses e acrescentar a cada uma suas particularidades.



Super Classe

Sub Classes

Diagrama de classes da UML

HERANÇA - EXEMPLO

*Veiculo.java

```java
package aula;

public class Veiculo {
    private String chassi;
    private String placa;
    private String cor;
    private double ipva;
}
```

Caminhao.java

```java
package aula;

public class Caminhao extends Veiculo {
    private int cargaMaxima;
}
```

*VeiculoPasseio.java

```java
package aula;

public class VeiculoPasseio extends Veiculo {
    private int quantPortas;

}
```

extends indica que a subclasse está herdando de **Veiculo**

HERANÇA - EXEMPLO

*Veiculo.java

```java
package aula;
public class Veiculo {
    private String chassi;
    private String placa;
    private String cor;
    private double ipva;

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }

}
```

*VeiculoPasseio.java

```java
package aula;
public class VeiculoPasseio extends Veiculo {
    private int quantPortas;

    public int getQuantPortas() {
        return quantPortas;
    }

    public void setQuantPortas(int quantPortas) {
        this.quantPortas = quantPortas;
    }

}
```

*Caminhao.java

```java
package aula;
public class Caminhao extends Veiculo {
    private int cargaMaxima;

    public int getCargaMaxima() {
        return cargaMaxima;
    }

    public void setCargaMaxima(int cargaMaxima) {
        this.cargaMaxima = cargaMaxima;
    }

}
```

*TestaHeranca.java

```java
package aula;
public class TestaHeranca {
    public static void main(String[] args) {
        Caminhao c = new Caminhao();
        VeiculoPasseio vp = new VeiculoPasseio();
        vp.setPlaca("lvc-9889");
        vp.setQuantPortas(4);
        c.setCargaMaxima(1000);
        c.setPlaca("ABC-3454");
        System.out.println(vp.getPlaca()
                +" Portas:"+ vp.getQuantPortas());
        System.out.println(c.getPlaca()
                +" Carga:"+ c.getCargaMaxima());
    }
}
```

SOBRESCRITA DE MÉTODOS (OVERRIDING)

Uma subclasse pode redefinir um método. Caso o método da superclasse não atenda a subclasse, existe a possibilidade de alterá-lo. Para que isso ocorra, o método da subclasse deve possuir o mesmo nome, a mesma lista de parâmetros e o mesmo tipo de retorno da sua superclasse.

```
public void adicionaipva(double valor){
    this.ipva += valor;
}
```

adicione o método acima na superclasse Veiculo

adicione o método abaixo na classe Caminhao

```
public void adicionaipva(double valor){
    this.ipva += valor * 2;
}
```

adicione o método acima na superclasse Veiculo

O reajuste do ipva para caminhões é o dobro do valor na classe Caminhao. A sobrescrita de método foi utilizada.

Como o atributo valor do ipva está privado na classe Veiculo será retornado um erro de visibilidade do atributo

USO DO MODIFICADOR PROTECTED

- Para acessarmos os atributos da superclasse precisamos trocar o modificador de acesso da classe **Veiculo** para **protected**. O modificador **protected** deixará o atributo visível para todas as outras classes e subclasses que pertencem ao mesmo pacote.

*Veiculo.java

```
package aula;
public class Veiculo {
    protected String chassi;
    protected String placa;
    protected String cor;
    protected double ipva;
```

altere o modificador dos atributos na classe Veiculo para protected

USO APÓS ALTERAÇÃO DO MODIFICADOR

*TestaHeranca.java

```
package aula;
public class TestaHeranca {
    public static void main(String[] args) {
        Caminhao c = new Caminhao();
        VeiculoPasseio vp = new VeiculoPasseio();

        vp.setPlaca("lvc-9889");
        vp.setQuantPortas(4);
        c.setCargaMaxima(1000);
        c.setPlaca("ABC-3454");

        vp.adicionaipva(400);
        c.adicionaipva(400);

        System.out.println(vp.getPlaca()
                +" Portas:"+ vp.getQuantPortas()+" Ipva:"+ vp.getIpva());
        System.out.println(c.getPlaca()
                +" Carga:"+ c.getCargaMaxima()  +" Ipva:"+ c.getIpva());

    }
}
```

adicione o que está em destaque na classe TesteHeranca

EXERCÍCIO

1) Criar uma classe com o nome **ImpostoDeRenda** com os seguintes atributos com visibilidade **protected.**

String (nome, telefone e email)
double (rendimentos)

Insira o **construtor** com os atributos nome e rendimentos.

Criar uma nova classe com o nome **PessoaFisica** herdando de **ImpostoDeRenda** com os seguintes atributos privados:

String (cpf e rg)

Criar uma nova classe com o nome **PessoaJuridica** herdando de **ImpostoDeRenda** com os seguintes atributos privados:

String (cnpj e inscEstadual).

Insira o **construtor** com todos atributos para ambas as classe.

PessoaFisica (nome, rendimentos, cpf e rg)
PessoaJuridica (nome, rendimentos , cnpj e inscEstadual).

Métodos das classes em comum **PessoaFisica e PessoaJuridica**:

Crie o método **calculoIR.** Para pessoa física deverá ser calculado o desconto 12% do rendimento e para e pessoa jurídica 15% do valor do rendimento.

Construa dois objetos em outra classe com o nome **TestaIR**

Exiba os dados e o valor a pagar de cada tipo de pessoa.

EXERCÍCIO - RESOLUÇÃO

*ImpostoDeRenda.java

```java
package exercicios;

public class ImpostoDeRenda {
    protected String nome;
    protected String telefone;
    protected String email;
    protected double rendimentos;

}
```

PessoaJuridica.java

```java
package exercicios;

public class PessoaJuridica extends ImpostoDeRenda {
    private String cnpj;
    private String inscEstadual;
```

PessoaFisica.java

```java
package exercicios;

public class PessoaFisica extends ImpostoDeRenda {
    private String cpf;
    private String rg;
```

EXERCÍCIO - RESOLUÇÃO

ImpostoDeRenda.java

```java
package exercicios;

public class ImpostoDeRenda {
    protected String nome;
    protected String telefone;
    protected String email;
    protected double rendimentos;

    public ImpostoDeRenda(String nome, double rendimentos) {
        this.nome = nome;
        this.rendimentos = rendimentos;
    }
}
```

*PessoaJuridica.java

```java
package exercicios;

public class PessoaJuridica extends ImpostoDeRenda {
    private String cnpj;
    private String inscEstadual;

    public PessoaJuridica(String nome, double rendimentos,
            String cnpj,String inscEstadual) {
        super(nome, rendimentos);
        this.cnpj = cnpj;
        this.inscEstadual = inscEstadual;
    }
}
```

*PessoaFisica.java

```java
package exercicios;

public class PessoaFisica extends ImpostoDeRenda {
    private String cpf;
    private String rg;

    public PessoaFisica(String nome, double rendimentos,
            String cpf, String rg) {
        super(nome, rendimentos);
        this.cpf = cpf;
        this.rg = rg;
    }
}
```

Faz referência ao construtor as super classe

Uma subclasse herda todos atributos, métodos de sua superclasse. Construtores não são herdados por subclasses, mas o construtor da superclasse pode ser chamado a partir da subclasse utilizando o comando **super**.

EXERCÍCIO - RESOLUÇÃO

Métodos calculaIR
classe PessoaFisica

```
public double calculaIr() {
    return this.rendimentos * 0.12;
}
```

Métodos calculaIR
classe PessoaJuridica

```
public double calculaIr() {
    return this.rendimentos * 0.15;
}
```

EXERCÍCIO - RESOLUÇÃO

Classe TestaIr

*TestaIr.java

```
package exercicios;

public class TestaIr {
    public static void main(String[] args) {
        PessoaFisica pf = new PessoaFisica("Mariazinha", 2000., "129450908-19", "0983445");
        PessoaJuridica pj = new PessoaJuridica("Xpto comercio LTDA", 65000., "909490900001-98", "1234");

        System.out.println(pf.getNome());
        System.out.println("Imposto a Pagar: " + pf.calculaIr());


        System.out.println(pj.getNome());
        System.out.println("Imposto a Pagar: " + pj.calculaIr());
    }
}
```

CLASSE OBJECT

Toda classe em Java herda implicitamente a classe **Object**. A classe **Object**, possui alguns métodos, dentre eles o **toString**. O método **toString** descreve qual instância de objeto está sendo utilizada . Ela retorna um texto com o nome da classe mais um código hexadecimal chamado de hashcode. Como herdamos da classe **Object** podemos sobrescrever o método **toString** pois não estamos interessados no valor que está sendo exibido.

Adicione as duas linhas na classe TestaIr e execute.

```
System.out.println(pf.toString());
System.out.println(pj.toString());
```

Adicione o toString na classe PessoaFisica

```
@Override
public String toString() {
    return this.nome + " Rendimentos: " + this.rendimentos + " Cpf: " + this.cpf;
}
```

Adicione o toString na classe PessoaJuridica

```
@Override
public String toString() {
    return this.nome + " Rendimentos: " + this.rendimentos + " Cnpj: " + this.cnpj;
}
```

Remova os outros System.out.println e deixe somente essas duas linhas para exibição na tela na classeTestaIr

```
System.out.println(pf.toString());
System.out.println(pj.toString());
```

EXERCÍCIO

2) Criar uma classe com o nome **Funcionario** com os seguintes atributos **protected**:

String(nome,cpf)
double(salario)
String(turno)

Criar uma classe com o nome **Gerente** com os seguintes atributos privados:

String(setor)

Criar uma classe com o nome **Assistente** com os seguintes atributos privados:

double(adicional)

Insira o construtor na classe **Funcionario(nome e salario) ,**
Gerente(nome, salario e setor) e Assistente (nome, salario e adicional).

Insira o método **toString** na classes **Funcionario** para exibir o nome e o salário

- Métodos :
  - Criar o método **aumentarSalario** na classe **Funcionario.** Todos os funcionários terão 2% de aumento de salário que será adicionado ao salário do funcionário, sendo que os gerentes tem mais 200,00 de bônus que será adicionado ao salário.
  - No salário do Assistente será acrescentado o valor do adicional.

EXERCÍCIO - RESOLUÇÃO

```java
public class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;
    protected String turno;

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

    @Override
    public String toString() {
        return "Nome:" + nome + "Salario:" + salario;
    }

    public String getNome() {
        return nome;
    }

    public String getCpf() {
        return cpf;
    }

    public double getSalario() {
        return salario;
    }

    public double aumentarSalario() {
        return salario = salario * 1.02;
    }
}
```

```java
public class Gerente extends Funcionario {
    private String setor;

    public Gerente(String nome, double salario, String setor) {
        super(nome, salario);
        this.setor = setor;
    }

    public String getSetor() {
        return setor;
    }

    public void setSetor(String setor) {
        this.setor = setor;
    }

    @Override
    public double aumentarSalario() {
        return super.aumentarSalario() + 200;
    }
}
```

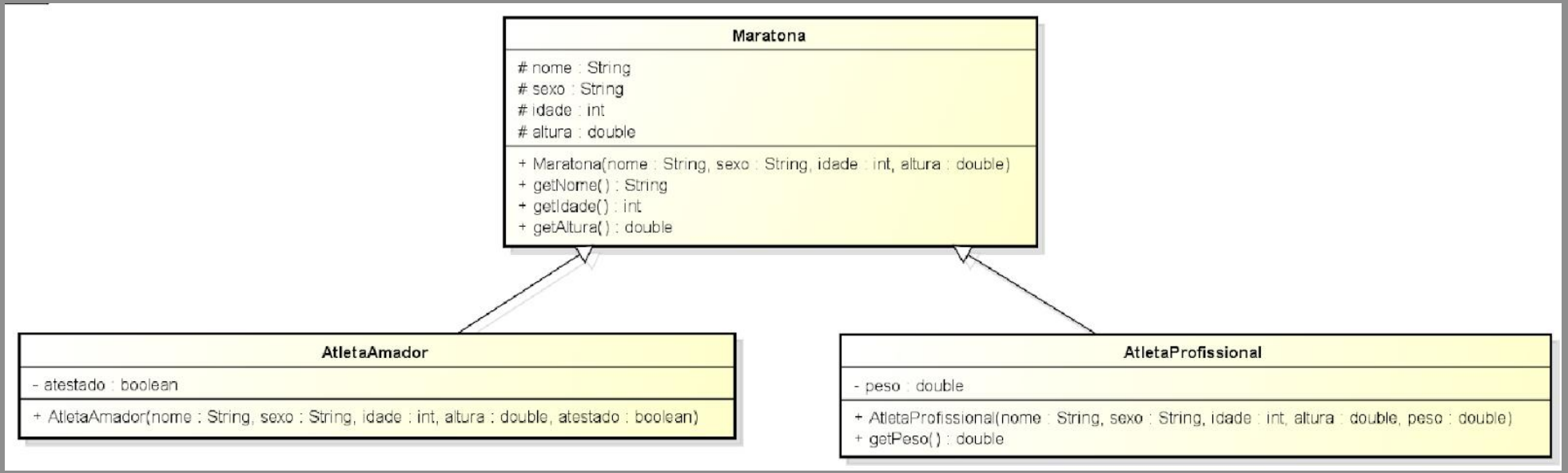
EXERCÍCIO - RESOLUÇÃO

```java
public class Assistente extends Funcionario {
    private double adicional;

    public Assistente(String nome, double salario, double adicional) {
        super(nome, salario);
        this.adicional = adicional;
    }

    public Double getAdicional() {
        return adicional;
    }

    public void setAdicional(double adicional) {
        this.adicional = adicional;
    }

    @Override
    public double aumentarSalario() {
        return super.aumentarSalario() + adicional;
    }

}
```

```java
public class TestaFuncionario {

    public static void main(String[] args) {
        Gerente gerente = new Gerente("Marcos", 3000, "Produção");
        Assistente assistente = new Assistente("Ana", 2000, 100);

        System.out.println(gerente.aumentarSalario());
        System.out.println(assistente.aumentarSalario());

    }

}
```

EXERCÍCIO

1) Crie as classes **Maratona**, **AtletaAmador**, **AtletaProfissional**, seus atributos, construtores, **toString**, **getters** conforme o diagrama de classe abaixo:



Crie um método com o nome **verificaSituacao** em que:

- Atletas com idade **maior que 18 ou altura maior ou igual a 1.80** poderão participar da maratona como maratonista. Deverá ser exibida a mensagem partipará ou não participará da competição

- Instancie 3 atletas em uma classe com o nome **TesteMaratonista**, exiba os dados do **toString** e chame o método **verificaSituacao** para saber se o atleta poderá ou não participar da competição.

EXERCÍCIO – RESOLUÇÃO

```java
public class Maratona {
    protected String nome;
    protected String sexo;
    protected int idade;
    protected double altura;

    public Maratona(String nome, String sexo, int idade, double altura) {
        super();
        this.nome = nome;
        this.sexo = sexo;
        this.idade = idade;
        this.altura = altura;
    }

    @Override
    public String toString() {
        return "Maratona [nome=" + nome + ", sexo=" + sexo + ", idade=" + idade + ", altura=" + altura + "]";
    }

    public String getNome() {
        return nome;
    }

    public int getIdade() {
        return idade;
    }

    public double getAltura() {
        return altura;
    }

    public void verificaSituacao() {
        if (idade > 18 || altura >= 1.8) {
            System.out.println("O Atleta competirá");
        } else {
            System.out.println("O Atleta não competirá");
        }
    }
}
```

```java
public class AtletaAmador extends Maratona {
    private boolean atestado;

    public AtletaAmador(String nome, String sexo, int idade, double altura, boolean atestado) {
        super(nome, sexo, idade, altura);
        this.atestado = atestado;
    }

    public boolean isAtestado() {
        return atestado;
    }

}
```

EXERCÍCIO – RESOLUÇÃO

```java
public class AtletaProfissional extends Maratona {
    private double peso;

    public AtletaProfissional(String nome, String sexo, int idade, double altura, double peso) {
        super(nome, sexo, idade, altura);
        this.peso = peso;
    }

    public double getPeso() {
        return peso;
    }

}
```

```java
public class TesteMaratona {

    public static void main(String[] args) {
        Maratona maratona1 = new AtletaAmador("Maria" , "F", 30, 1.90, true);
        Maratona maratona2 = new AtletaProfissional("Ana" , "F", 15, 1.60, 87);
        Maratona maratona3 = new AtletaProfissional("Marcos" , "M", 17, 1.81, 90);

        System.out.println(maratona1.toString());
        maratona1.verificaSituacao();

        System.out.println(maratona2.toString());
        maratona2.verificaSituacao();

        System.out.println(maratona3.toString());
        maratona3.verificaSituacao();

    }

}
```

POLIMORFISMO

No polimorfismo um objeto pode ser referenciado de várias formas. Na programação orientada a objetos, este termo se refere a uma determinada classe que possui a capacidade de alterar o comportamento de um método para adequá-lo a necessidade solicitada.

Definimos Polimorfismo como um princípio a partir do qual as classes derivadas de uma superclasse são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

Tipos de Polimorfismo

- **Overloading** (sobrecarga de métodos) - Temos dois ou mais métodos com o mesmo nome, mas aceitando parâmetros diferentes com assinaturas diferentes.

- **Overriding** (sobrescrita de métodos) - Um objeto possui um método alterado, a partir de um método herdado de uma super classe.

EXEMPLO POLIMORFISMO

Empregado.java

```java
package aula;
public class Empregado {
    protected String nome,cargo;
    protected double salario;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCargo() {
        return cargo;
    }

    public void setCargo(String cargo) {
        this.cargo = cargo;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public double adicionalSalario() {
        return this.salario *= 1.08;
    }
}
```

Diretor.java

```java
package aula;

public class Diretor extends Empregado{


}
```

Tecnico.java

```java
package aula;

public class Tecnico extends Empregado {
    public double adicionalSalario() {
        return this.salario *= 1.15;
    }
}
```

sobrescrita de método

EXEMPLO POLIMORFISMO

TestaEmpregado.java

```
package aula;

public class TestaEmpregado {
    public static void main(String[] args) {
        Empregado e = new Tecnico();

        e.setNome("Maria Luiza");
        e.setSalario(2000.);

        e.adicionalSalario();
        System.out.println(e.getNome() + "-" + e.getSalario());

    }

}
```

Tecnico é um Empregado. O polimorfismo é utilizado. Instanciamos um Tecnico que é um Empregado.

O método invocado é o de Tecnico e não o de Empregado

O polimorfismo só existe com a herança.

EXEMPLO POLIMORFISMO

*TestaEmpregado.java

```
package aula;

public class TestaEmpregado {
    public static void main(String[] args) {
        Empregado e = new Tecnico();
        Empregado e1 = new Empregado();

        e.setNome("Maria Luiza");
        e.setSalario(2000.);

        e1.setNome("Ana Lucia");
        e1.setSalario(1000.);

        e.adicionalSalario();
        System.out.println(e.getNome() + "-" + e.getSalario());

        e1.adicionalSalario();
        System.out.println(e1.getNome() + "-" + e1.getSalario());

    }

}
```

Adicione as linhas destacadas

Qual método será invocado agora?

EXERCÍCIOS

1) Criar as estrutura de classes conforme diagrama.

Na classe Fixo o método **calcularSalario** é "void" sem retorno e deverá calcular o salário bruto do vendedor da seguinte forma:
Para cada venda efetuada chamar este método e adicionar o cálculo da comissão sobre a venda efetuada no salário bruto do vendedor fixo.

Criar uma classe de teste com o main, instanciar dois vendedores **fixos**, passando os dados dos vendedores fixos no **construtor**
O salário bruto inicial do vendedor fixo será seu salário base.

Fazer a leitura de dados com a classe Scanner
O usuário deverá escolher entre os dois vendedores cadastrados.
O programa deverá ser encerrado quando o usuário escolher uma opção de saída.

No final deverá ser exibido o salário bruto dos vendedores fixos.

```
----------Sistema de Vendas----------
Vendedores:
1-Joana
2-Maria
Escolha o vendedor?
1
Digite o valor da venda
2000

Deseja encerrar o programa? (S/N)
N
Escolha o vendedor?
1
Digite o valor da venda
3000

Deseja encerrar o programa? (S/N)
S
---------Salário dos vendedores:--------
nome:Joana
salário Bruto:2150.0
nome:Maria
salário Bruto:2000.0
------------Total Vendido:--------------
Total:5000.0
```