53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR

TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

JAVA I

Herança, Reescrita, Polimorfismo

31/07/2020

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) Crie uma classe chamada Empregado que inclui as três informações a seguir como atributos:

- nome
- sobrenome
- salario

Sua classe deve ter um construtor que inicializa os três atributos.

Forneça um método set e get para cada atributo.

Escreva um aplicativo de teste que demonstra as capacidades da classe.

Crie um método calculaImpostoRenda para descontar no salário de acordo com a tabela:

Salário	Desconto
menores que 1903,98	0%
entre que 1903,98 e 2826,65	7,5%
entre que 2826,66 e 3751,05	15%
entre que 3751,06 e 4664,68	22,5%
maior que 4664,68	27,5%

Crie duas instâncias da classe Empregado

Calcule o imposto de renda

Exibe os dados do usuário com o salário já descontado o imposto de renda

EXERCICIO – AULA ANTERIOR

2) 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 para finalizar o pedido que caso o dia do pedido for um domingo o cliente terá um desconto de 10% no valor do pedido.

Criar 3 instâncias em uma nova classe com o main

Finalizar o pedido

Mostrar o total dos pedidos

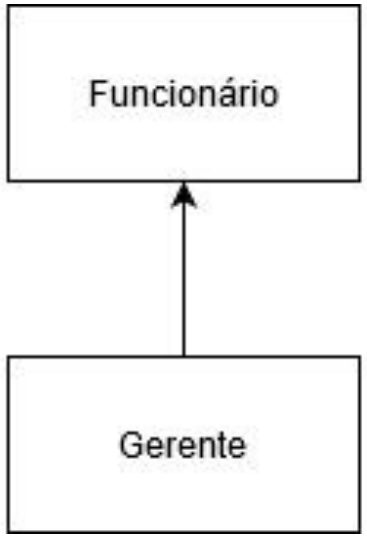
ILUSTRANDO COM EXEMPLO

- 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
}
```

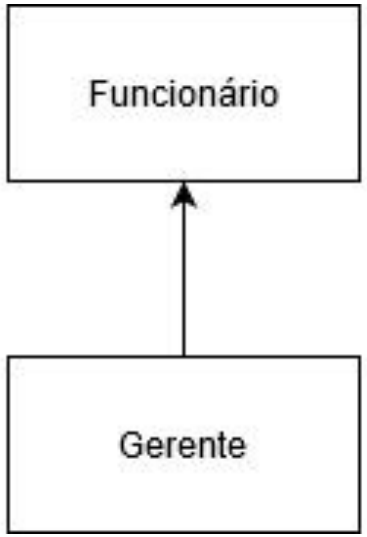
- O Gerente é um funcionário especial
  - Além do funcionário comum, temos outros cargos, como os gerentes. Naturalmente, eles têm informações em comum com os demais funcionários e outras informações exclusivas

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

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



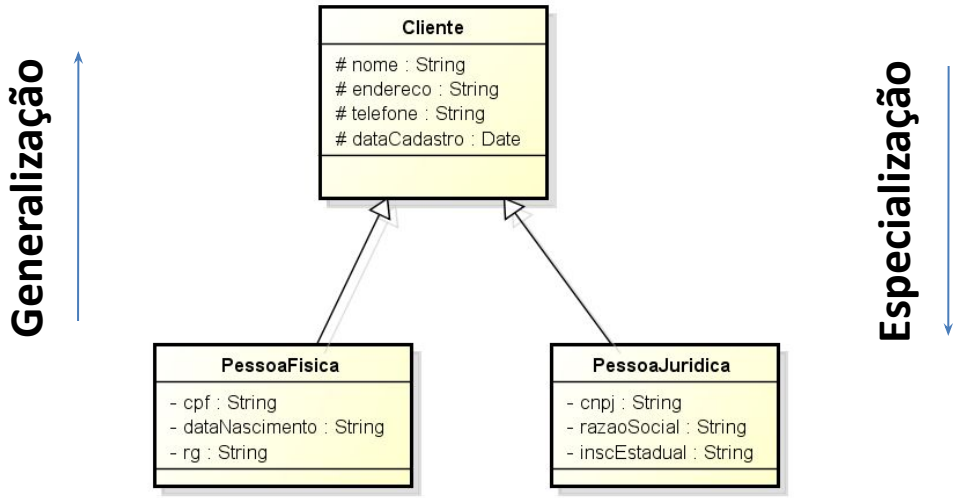
- Estendendo a classe funcionário
  - Em Java, existe um jeito de relacionarmos uma classe de tal maneira que uma delas herda tudo que a outra tem. Em nosso caso, queremos que Gerente possua todos os métodos e atributos de Funcionário.
  - Para isso utilizamos a cláusula **extends** na definição da classe.

```
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
    }
}
```



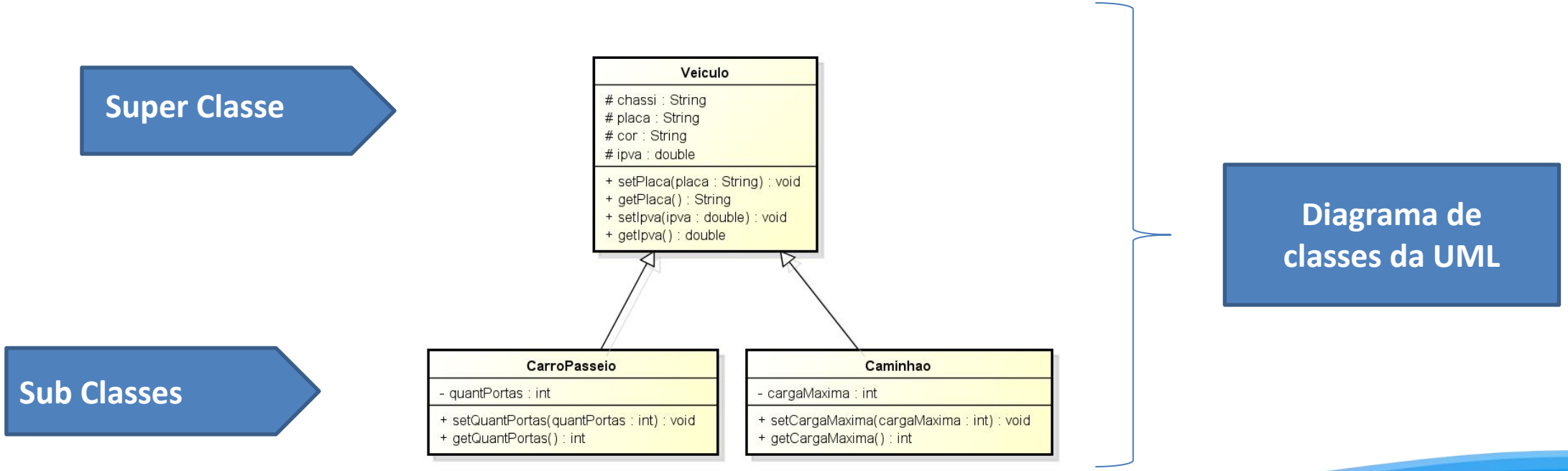
GENERALIZAÇÃO/ESPECIALIZAÇÃO

- 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 duas 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.



HERANÇA - EXEMPLO

*Veiculo.java

```
package aula;

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

Caminhao.java

```
package aula;

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

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

*VeiculoPasseio.java

```
package aula;

public class VeiculoPasseio extends Veiculo {
    private int quantPortas;

}
```

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

HERANÇA - EXEMPLO

*VeiculoPasseio.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

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

    public int getCargaMaxima() {
        return cargaMaxima;
    }

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

}
```

*Veiculo.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;
    }

}
```

*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");
        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

```
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.

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**

- Atributos **protected:**
  String(nome,telefone,email)
  double(rendimentos)

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

- Criar uma nova classe com o nome **PessoaFisica** herdando de **ImpostoDeRenda**

- Atributos **private:**
  String(cpf e rg).

- Criar uma nova classe com o nome **PessoaJuridica** herdando de **ImpostoDeRenda**

- Atributos **private** do **PessoaJuridica : String**(cnpj e inscEstadual).

- Insira o **construtor** cheio 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 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;

}
```

PessoaFisica.java

```java
package exercicios;

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

PessoaJuridica.java

```java
package exercicios;

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

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());
    }
}
```

Inserir o getter para o nome

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 hash code.

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

Adicione as duas linhas na classe TestaIr e execute.

- Como herdamos da classe **Object** podemos sobrescrever o método **toString** pois não estamos interessados no valor que está sendo exibido.

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

Adicione o toString na classe PessoaFisica

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

Adicione o toString na classe PessoaJuridica

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

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

EXERCÍCIO

2) Criar uma classe com o nome Funcionario

- Atributos protected do Funcionario :
  String(nome,cpf)
  double(salario)
  String(turno)

  - Criar uma classe com o nome Gerente
  - Atributos private do Gerente : String(nivel)
  - Criar uma classe com o nome Assistente
  - Atributos private do Assistente : double(adicional)

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

- Insira o método toString na classes Funcionario para exibir o nome e o salário
  - Métodos :
    - Todos os funcionários tem 1% na participação sendo que os gerentes tem mais 200 reais na participação.
    - No salário do Assistente será acrescentado o valor do adicional.

EXERCÍCIO - RESOLUÇÃO

Funcionario.java

```java
package exercicios;

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


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

    public String getNome() {
        return nome;
    }

    public double getSalario() {
        return salario + this.partLucros();
    }

    public double partLucros(){
        return this.salario * 0.01;
    }

}
```

Gerente.java

```java
package exercicios;

public class Gerente extends Funcionario {
    private String nivel;

    @Override
    public String toString() {
        return this.nome + " - " + this.salario;
    }

    public Gerente(String nome, double salario) {
        super(nome, salario);
    }


    public double partLucros(){
        return super.partLucros() + 200;
    }

}
```

Assistente.java

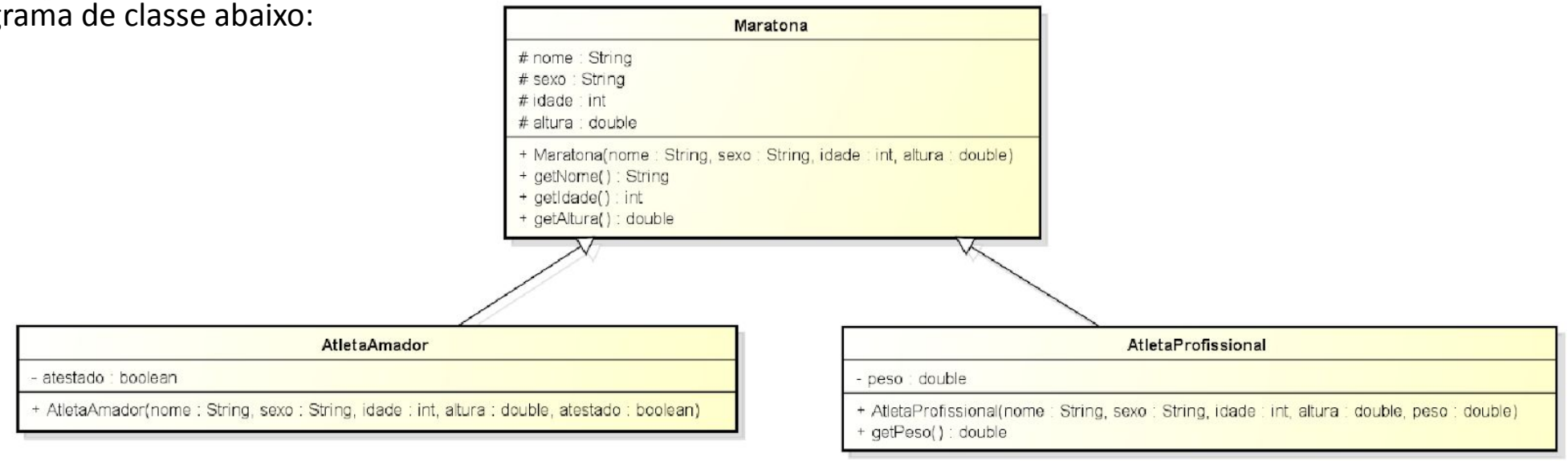
```java
package exercicios;

public class Assistente extends Funcionario {
    private double adicional;

    public String toString() {
        return this.nome + " - " + this.salario;
    }

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

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

TestaFuncionario.java

```java
package exercicios;

public class TestaFuncionario {
    public static void main(String[] args) {
        Gerente g   = new Gerente("Maria Luzia", 4350.);
        Assistente a= new Assistente("Antonio Carlos", 680., 180);

        System.out.println(g.toString());
        System.out.println("Part. Lucros:" + g.partLucros());
        System.out.println("Salario Bruto:" + g.getSalario()+"\n");


        System.out.println(a.toString());
        System.out.println("Part. Lucros:" + a.partLucros());
        System.out.println("Salario Bruto:" + a.getSalario());

    }
}
```

EXERCÍCIO

1) Crie as classes Maratonista, 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 **TestaMaratonista**, exiba os dados do toString e chame o método **verificaSituacao** para saber se o atleta poderá ou não participar da competição.

EXERCÍCIO – RESOLUÇÃO

Maratona.java

```java
package aula;

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) {
        this.nome   = nome;
        this.sexo   = sexo;
        this.idade  = idade;
        this.altura = altura;
    }

    public String getNome() {
        return nome;
    }

    public int getIdade() {
        return idade;
    }

    public double getAltura() {
        return altura;

    }

    public void verificaSituacao() {
        if (idade > 18 || altura >= 1.80)
            System.out.println("O Atleta competirá altura e peso OK" + "\n");
        else
            System.out.println("O Atleta não competirá problema com altura ou peso" + "\n");
    }
}
```

AtletaAmador.java

```java
package aula;

public class AtletaAmador extends Maratona {
    private boolean atestado;

    @Override
    public String toString() {
        return this.nome + " " + this.idade + "anos " + "altura:" +this.altura;
    }

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

AtletaProfissional.java

```java
package aula;

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;
    }


}
```

EXERCÍCIO – RESOLUÇÃO

TestaMaratona.java

```
package aula;

public class TestaMaratona {
    public static void main(String[] args) {
        Maratona m1 = new AtletaAmador("Maria", "F", 30, 1.90, true);
        Maratona m2 = new AtletaProfissional("Ana", "F", 15, 1.60, 87.);
        Maratona m3 = new AtletaProfissional("Marcos", "M", 17, 1.80, 90);

        System.out.println(m1.toString());
        m1.verificaSituacao();

        System.out.println(m2.toString());
        m2.verificaSituacao();

        System.out.println(m3.toString());
        m3.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.

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?