Orientação a objetos

Herança e Relacionamento

Roni Schanuel
05-05-2023

Firjan SENAI

REVISÃO AULA ANTERIOR - HERANÇA

1) Crie as classes, seus atributos, métodos, construtores, toString, getters e setters conforme o diagrama de classe abaixo:
Método **calcularPagamento** da classe **Plano**

- O plano paga como valor inicial R$80,00 **#valorPago** de consulta para o médico, anestesista ou clínica.
- O desconto inicial do valor de ISS **#valorIss** é de 5%

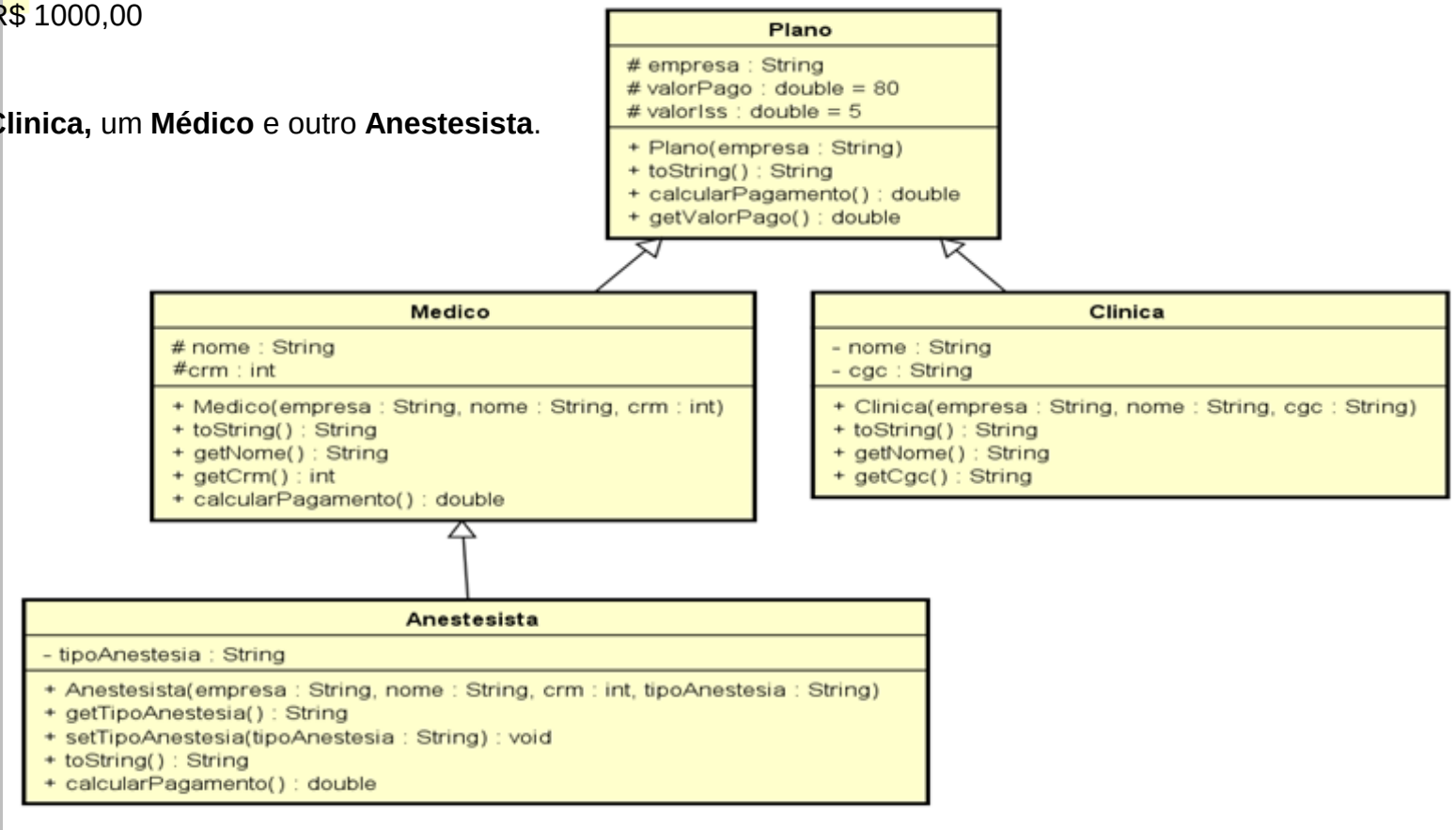
O cálculo do pagamento do plano a um profissional é o valor pago com redução de 5% do valor do ISS que é descontado do valor pago.

Método **calcularPagamento** da classe **Medico e Anestesista**

- Para médicos o valor por consulta pago pelo plano de saúde terá acréscimo de 10% .
- Para os anestesistas além dos 10% terá um acréscimo de mais R$ 1000,00

Classe com o **main**

- Crie a classe de teste com o nome **TestePlano**, Crie um objeto **Clinica,** um **Médico** e outro **Anestesista**.
- Chame o método **calculaPagamento**.
- Exiba os dados do **toString** e o valor a ser pago pelo plano.



RESOLUÇÃO

```java
public class Plano {
    protected String empresa;
    protected double valorPago = 80;
    protected double valorIss = 5;

    public Plano(String empresa) {
        super();
        this.empresa = empresa;
    }

    @Override
    public String toString() {
        return "Plano:" + empresa + " ValorPago:" + String.format("%.2f", valorPago);
    }

    public double calcularPagamento() {
        return valorPago = valorPago - valorPago * valorIss/100;
    }
}
```

RESOLUÇÃO

```java
public class Clinica extends Plano {
    private String nome;
    private String cgc;

    public Clinica(String empresa, String nome, String cgc) {
        super(empresa);
        this.nome = nome;
        this.cgc = cgc;
    }

    @Override
    public String toString() {
        return super.toString() + " Clínica:" + nome;
    }

    public String getNome() {
        return nome;
    }

    public String getCgc() {
        return cgc;
    }

}
```

RESOLUÇÃO

```java
public class Medico extends Plano {
    protected String nome;
    private int crm;

    public Medico(String empresa, String nome, int crm) {
        super(empresa);
        this.nome = nome;
        this.crm = crm;
    }

    @Override
    public String toString() {
        return super.toString() + " Médico:" + nome;
    }

    public String getNome() {
        return nome;
    }

    public int getCrm() {
        return crm;
    }

    @Override
    public double calcularPagamento() {
        return valorPago = super.calcularPagamento() *1.10;
    }

}
```

RESOLUÇÃO

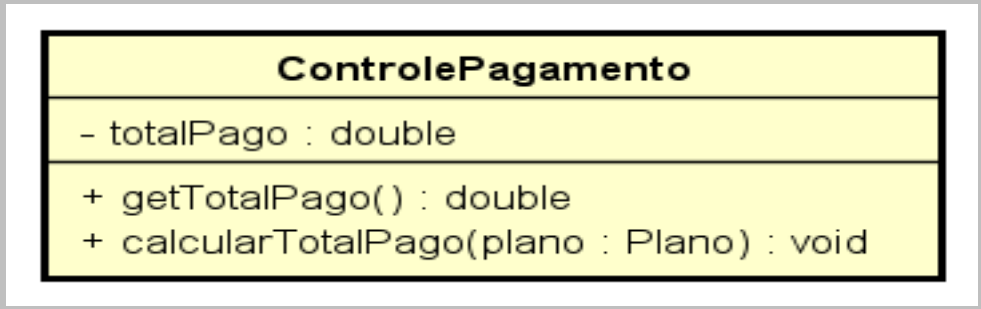
```
public class Anestesista extends Medico {
    private String tipoAnestesia;

    public Anestesista(String empresa, String nome, int crm, String tipoAnestesia) {
        super(empresa, nome, crm);
        this.tipoAnestesia = tipoAnestesia;
    }

    public String getTipoAnestesia() {
        return tipoAnestesia;
    }

    public void setTipoAnestesia(String tipoAnestesia) {
        this.tipoAnestesia = tipoAnestesia;
    }

    @Override
    public String toString() {
        return super.toString() + " Tipo Anestesia:"+ tipoAnestesia;
    }

    @Override
    public double calcularPagamento() {
        return  valorPago = super.calcularPagamento() +1000;
    }

}
```

RESOLUÇÃO

```
public class TestePlano {

    public static void main(String[] args) {
        Clinica clinica = new Clinica("Amil", "Checkup","123.456.789/0001-56");
        Medico medico = new Medico("Golden Cross", "Carlos da Silva", 1345445);
        Anestesista anestesista = new Anestesista("Amil", "Carla dos Santos", 123456, "Peridural");

        clinica.calcularPagamento();
        medico.calcularPagamento();
        anestesista.calcularPagamento();

        System.out.println(clinica.toString());
        System.out.println(medico.toString());
        System.out.println(anestesista.toString());

    }

}
```

EXERCÍCIO – PARTE 2

Crie uma nova classe, com o nome **ControlePagamento.** Esta classe será responsável por totalizar todos os pagamentos realizados pelo plano de saúde. Neste exemplo não utilizaremos um atributo estático para acumular os totais.



Esta classe irá calcular o total pago pelo plano de saude a todos os médicos e clínicas.

```
public class ControlePagamento {
    private double totalPago;

    public double getTotalPago() {
        return totalPago;
    }

    public void calcularTotalPago(Plano plano) {
        totalPago += plano.getValorPago();
    }

}
```

Polimorfismo aplicado .estou passando como argumento o tipo **Plano** que é o tipo mais alto na hierarquia das classes que utilizamos assim podemos passar qualquer elemento abaixo de plano como argumento.

EXERCÍCIO – PARTE 2

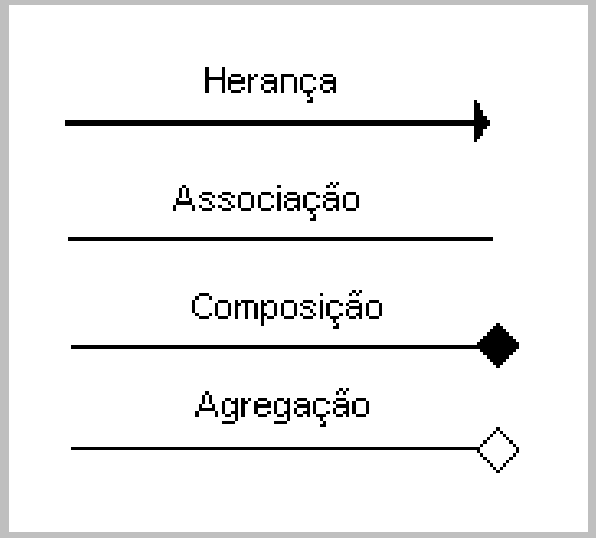
A classe **TestePlano** foi alterada para exibir o total pago pelo plano

```
public class TestePlano {

    public static void main(String[] args) {
        Clinica clinica = new Clinica("Amil", "Checkup", "123.456.789/0001-56");
        Medico medico = new Medico("Golden Cross", "Carlos da Silva", 1345445);
        Anestesista anestesista = new Anestesista("Amil", "Carla dos Santos", 123456, "Peridural");
        ControlePagamento cp = new ControlePagamento();

        clinica.calcularPagamento();
        medico.calcularPagamento();
        anestesista.calcularPagamento();

        cp.calcularTotalPago(clinica);
        cp.calcularTotalPago(medico);
        cp.calcularTotalPago(anestesista);

        System.out.println(clinica.toString());
        System.out.println(medico.toString());
        System.out.println(anestesista.toString());

        System.out.println("O total pago pelo plano:" + String.format("%.2f", cp.getTotalPago()));
    }
}
```

RELACIONAMENTOS

As classes podem possuir relacionamentos entre si, compartilhando informações umas com as outras. Há quatro tipos básicos de relacionamentos:

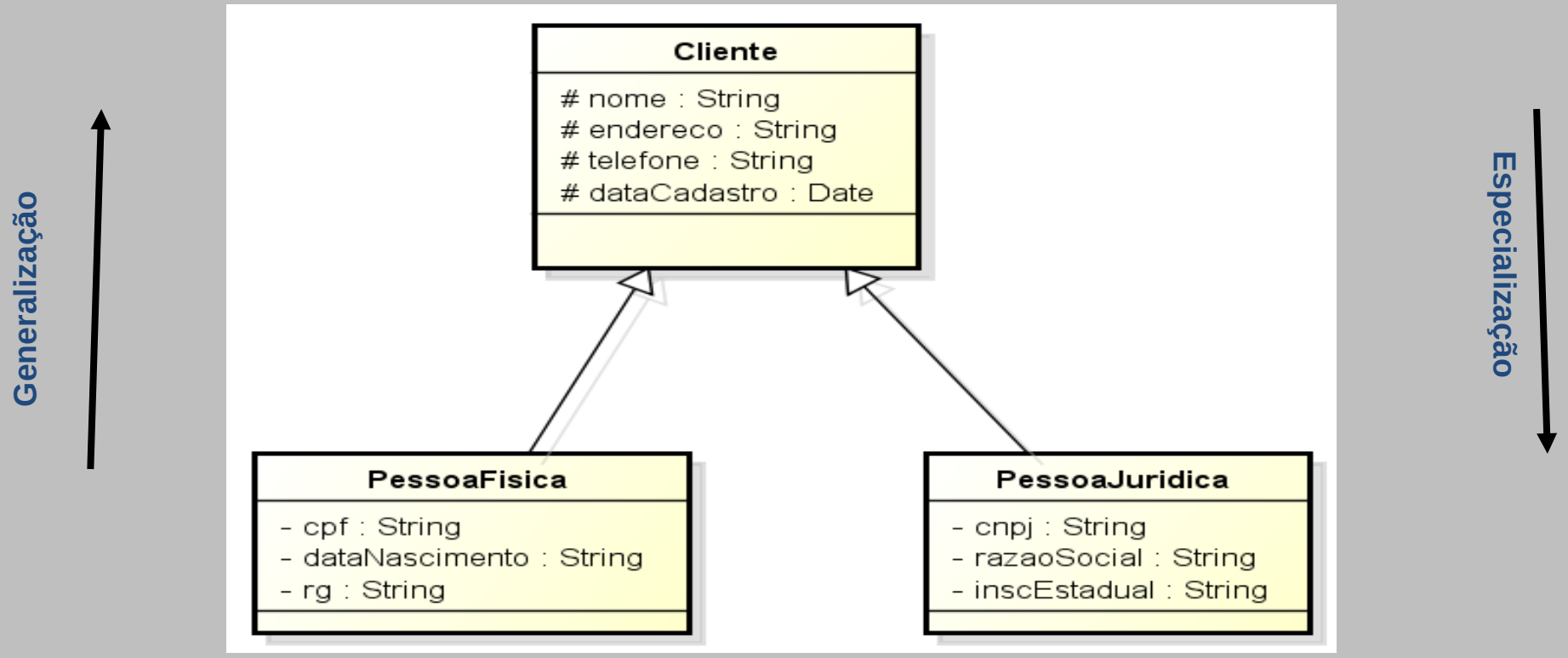
- generalização/especialização
- associação
- composição
- agregação



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.



RELACIONAMENTOS

Associação

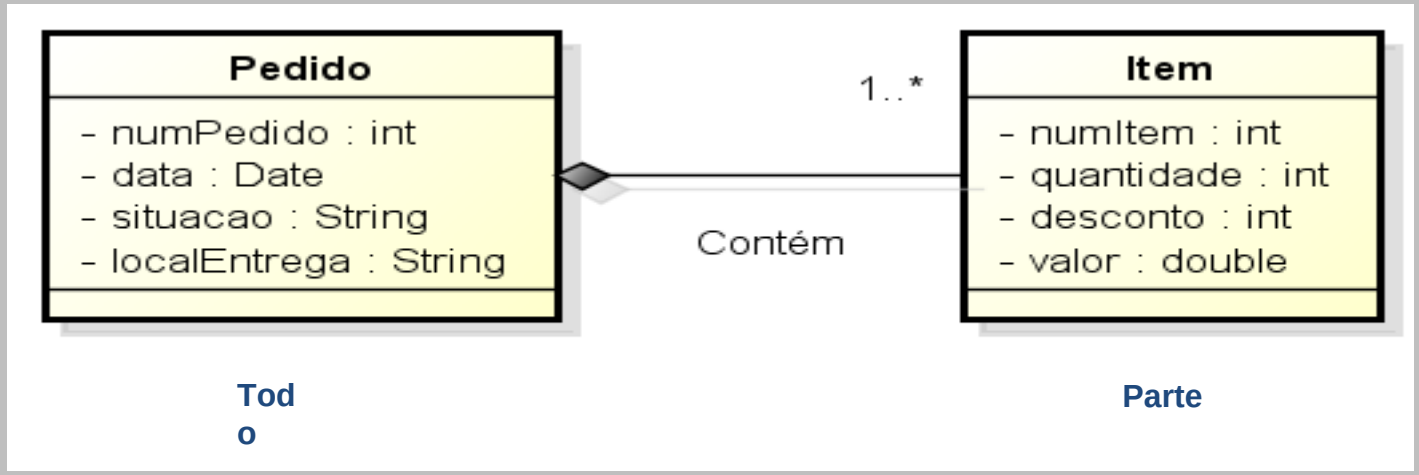
É um relacionamento que descreve o vínculo entre duas classes. É o tipo de relacionamento mais encontrado em diagramas de classe.

Representação UML

Notação UML atributos:
+ publico
- privado
 # protegido

COMPOSIÇÃO

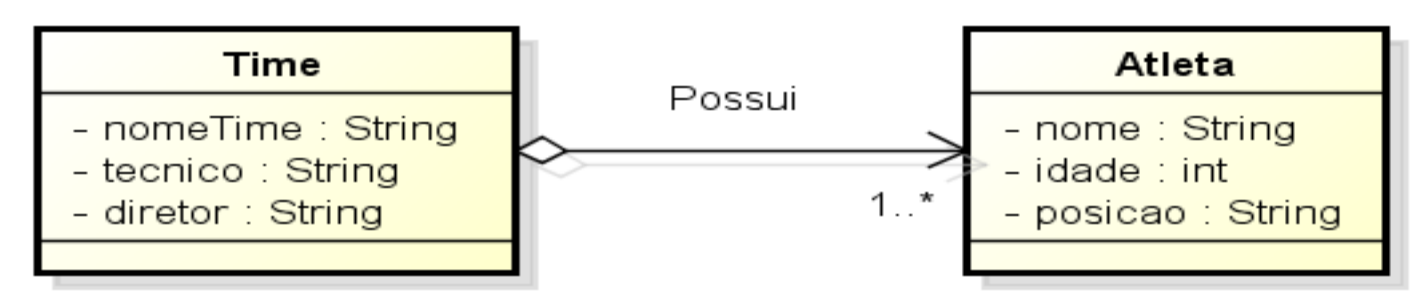
A Composição todo/parte é uma forma especial de associação utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação. Um símbolo de losango preenchido é ilustrado próximo a classe que representa o **todo**.



AGREGAÇÃO

Agregação

Na Agregação a existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo. Um símbolo de losango não preenchido é ilustrado próximo a classe que representa o **todo**.



Um time é formado por atletas, mas os atletas existem independentemente de um time existir.

EXERCÍCIO

Fazer o relacionamento entre Pessoa-Endereco. No exemplo abaixo a Pessoa mora em um Endereço

```java
public class Funcionario {
    private String cpf;
    private String nome;
    private Endereco endereco;

    @Override
    public String toString() {
        return "cpf:" + cpf + " nome:" + nome + " endereco:";
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

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

    public void setEndereco(Endereco endereco) {
        this.endereco = endereco;
    }

    public Endereco getEndereco() {
        return endereco;
    }

}
```

O atributo endereco é do tipo Endereco que irá fazer referência a um objeto deste tipo.

```java
public class Endereco {
    private String rua;
    private String bairro;
    private String cidade;

    public Endereco(String rua, String bairro, String cidade) {
        super();
        this.rua = rua;
        this.bairro = bairro;
        this.cidade = cidade;
    }

    public String getRua() {
        return rua;
    }

    public String getBairro() {
        return bairro;
    }

    public String getCidade() {
        return cidade;
    }

}
```

EXERCÍCIO

```
public class TesteEndereco {

    public static void main(String[] args) {
        Endereco endereco = new Endereco("Rua Alberto Torres 192", "Centro", "Petrópolis");

        Funcionario funcionario = new Funcionario();
        funcionario.setCpf("134123456-25");
        funcionario.setNome("Ana");
        funcionario.setEndereco(endereco);

        System.out.println(funcionario.toString() + funcionario.getEndereco().getRua() + " "
        + funcionario.getEndereco().getBairro());


    }

}
```

EXERCÍCIO

Fazer o relacionamento entre Pessoa-Imóvel. No exemplo abaixo a Pessoa possui um imóvel.

Pessoa.java

```
package aula;

public class Pessoa {
    private String cpf;
    private String nome;
    private String telefone;

    public Pessoa(String cpf, String nome, String telefone) {
        this.cpf = cpf;
        this.nome = nome;
        this.telefone = telefone;
    }

    public String getNome() {
        return nome;
    }


}
```

*Imovel.java

```
package aula;
public class Imovel {
    private String local;
    private String tipo;
    private double valor;
    private Pessoa dono;

    public Imovel(String local, String tipo,
            double valor, Pessoa dono) {
        this.local = local;
        this.tipo = tipo;
        this.valor = valor;
        this.dono = dono;
    }

    public String getTipo() {
        return tipo;
    }

    public double getValor() {
        return valor;
    }

    public Pessoa getDono() {
        return dono;
    }
}
```

o atributo dono é do tipo Pessoa irá fazer referência a um objeto deste tipo.

EXERCÍCIO

Teste.java

```
package aula;
public class Teste {
    public static void main(String[] args) {
      Pessoa p = new Pessoa("384.343.348-90", "Joaquim", "2345-9889");
      Imovel i = new Imovel("Quitandinha", "apto", 98000., p);
      System.out.println(i.getTipo() + "," + i.getDono().getNome() + ","
              + i.getValor());

    }

}
```

O valor de referência da variável p é passado como argumento para o atributo dono na criação do objeto imóvel através do construtor.

EXERCÍCIO

Fazer o relacionamento entre Time-Atleta. No exemplo abaixo o time possui um ou mais atletas.

```
public class Time {
    private String nomeTime;
    private String tecnico;
    private Atleta[] atletas;

    public Time(String nomeTime, String tecnico, Atleta[] atletas) {
        super();
        this.nomeTime = nomeTime;
        this.tecnico = tecnico;
        this.atletas = atletas;
    }

    public String getNomeTime() {
        return nomeTime;
    }

    public String getTecnico() {
        return tecnico;
    }

    public Atleta[] getAtletas() {
        return atletas;
    }

    public void setAtletas(Atleta[] atletas) {
        this.atletas = atletas;
    }

    public void adicionarAtletas(Atleta atleta) {
        for (int i = 0; i < atletas.length; i++) {
            if (atletas[i] == null) {
                atletas[i] = atleta;
                return;
            }
        }
    }

    public void mostrarTime() {
        for (int i = 0; i < atletas.length; i++) {
            System.out.println(atletas[i].getNome());
        }

    }
}
```

vetor de atletas

Construtor

getters e setters

Varre o vetor para verificar se a posição está vazia para adicionar a referência, se estiver vazia adiciona o atleta.

Exibe todos os atletas do time

EXERCÍCIO

```java
package br.com.senai.relacionamentos;

public class Atleta {
    private String nome;
    private int idade;
    private String posicao;

    public Atleta(String nome, int idade, String posicao) {
        super();
        this.nome = nome;
        this.idade = idade;
        this.posicao = posicao;
    }

    public String getNome() {
        return nome;
    }

    public int getIdade() {
        return idade;
    }

    public String getPosicao() {
        return posicao;
    }

}
```

EXERCÍCIO

```
public class TesteAtleta {

    public static void main(String[] args) {

        Atleta atleta1 = new Atleta("Diego Alves", 25, "Goleiro");
        Atleta atleta2 = new Atleta("Rafinha", 32, "Lateral");
        Atleta atleta3 = new Atleta("Arrascaeta", 25, "Meio Campo");

        Time time = new Time("Flamengo", "Jorge Jesus", new Atleta[3]);
        time.adicionarAtletas(atleta1);
        time.adicionarAtletas(atleta2);
        time.adicionarAtletas(atleta3);

        System.out.println("Time:" + time.getNomeTime());
        time.mostrarTime();

    }

}
```

Vetor com três posições

EXERCÍCIO

1) Criar o diagrama abaixo com seus relacionamentos e exibir os dados em uma classe com o main da seguinte forma:

- **Um contato possui um ou vários telefones**
- **Um contato possui um endereço**
- **Um endereço pertence a uma cidade**
- **Uma cidade pertence a um estado**

Exibir os dados no console conforme imagem abaixo:

```
Nome:Roberta
Rua Fonseca Ramos 181 Centro Petrópolis Rio de Janeiro
Telefone:2234-1250
Telefone:2237-1350
```