Programação de Sistemas Computacionais

Outros assuntos

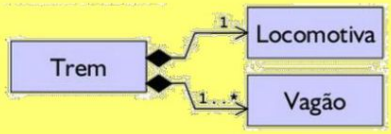
emerson@paduan.pro.br

Associação Composição Agregação

xxxx

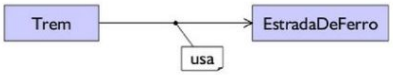
Composição, Agregação e Associação

- **Composição**
  - Um trem **é formado por** 1 locomotiva e n vagões;
  - Relacionamento **tem-um**;



- **Agregação**
  - Uma locomotiva (todo) **tem** um farol (parte), mas não deixa de existir se não o tiver;

- **Associação**
  - Um trem **usa** uma estrada de ferro (a estrada não faz parte do trem, mas ele depende dela).



emerson@paduan.pro.br

Como implementar

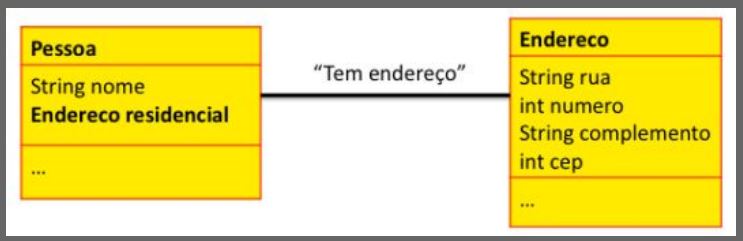
Um sistema é composto por várias Classes:

- Quando um ou mais *atributos* de uma Classe é uma *referência* para outra classe.

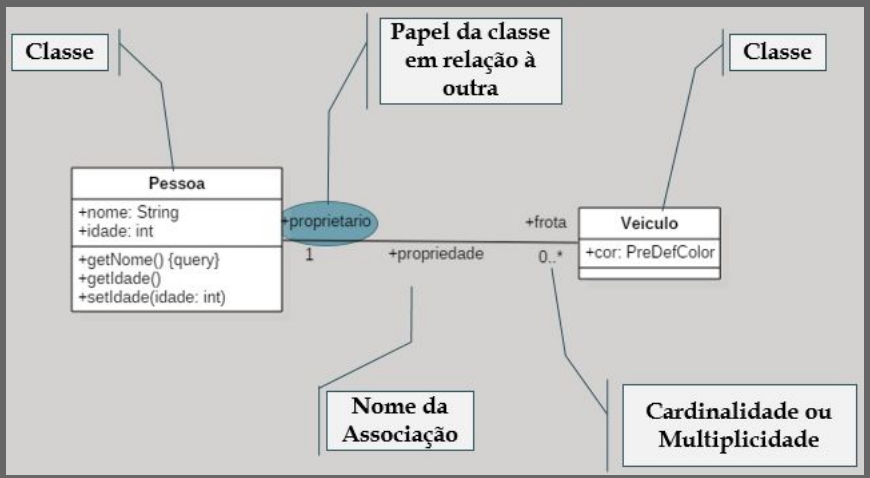
- As classes se conectam para poderem se comunicar por troca de mensagens (chamadas de métodos).

emerson@paduan.pro.br

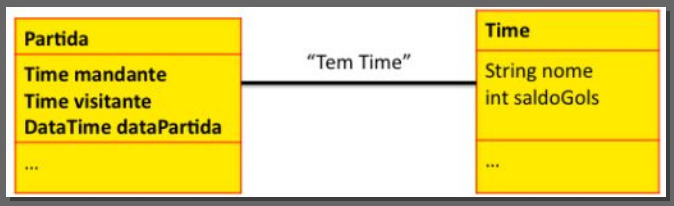
Exemplo


Pessoa
String nome
Endereco residencial
...
"Tem endereço"
Endereco
String rua
int numero
String complemento
int cep
...

UML


Classe
Papel da classe em relação à outra
Classe
Pessoa
+nome: String
+idade: int
+getNome() {query}
+getIdade()
+setIdade(idade: int)
+proprietario
1
+propriedade
+frota
0..*
Veiculo
+cor: PreDefColor
Nome da Associação
Cardinalidade ou Multiplicidade

Cardinalidade 1:N



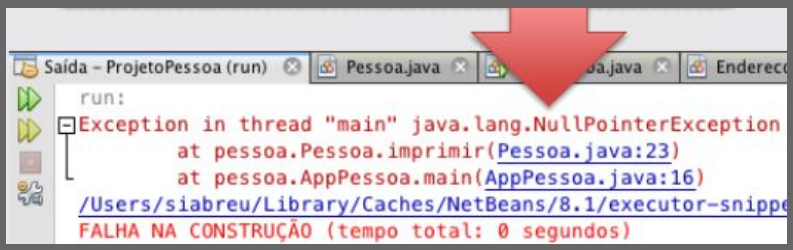
Codificando

```
public class Pessoa{
    //atributos
    private String nome;
    private int idade;
    private char sexo;
    private Endereco end;
    ....

    public String imprimir(){
        return "Nome: " + nome +
               "\nIdade: " + idade +
               "\nSexo: " + sexo +
               "Endereço: " + end.imprimir();
    }
}
```

```
public class Endereco{
    //atributos
    private String logradouro;
    private String complemento;
    private int numero;
    private String cep;
    ....

    public String imprimir(){
        return "Logradouro: " + logradouro +
               "\nComplemento: " + complemento +
               "\nNúmero: " + numero +
               "CEP: " + cep;
    }
}
```

Cuidado!

```
public class AppPessoa {
    public static void main(String[] args) {
        Pessoa objPessoa = new Pessoa();

        System.out.println(objPessoa.imprimir());
    }
}
```



Resolvendo

```
public class Pessoa{
    //atributos
    private String nome;
    private int idade;
    private char sexo;
    private Endereco end;

    //construtor default
    public Pessoa() {
        this.end = new Endereco();
    }

    public String imprimir(){
        return "Nome: " + nome +
               "\nIdade: " + idade +
               "\nSexo: " + sexo +
               "Endereço: " + end.imprimir();
    }
}
```

Como fica:

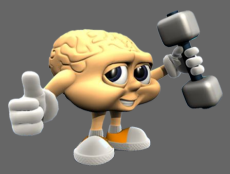
```
public class AppPessoa {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        Scanner entradaString = new Scanner(System.in);
        //objeto da classe Endereco
        Endereco objEnd = new Endereco();
        //Instância da classe Pessoa
        Pessoa objPessoa = new Pessoa();
        char resp;

        // Dados da Pessoa
        System.out.println("Qual o nome? ");
        objPessoa.setNome( entradaString.nextLine() );

        System.out.println("Qual a idade? ");
        objPessoa.setIdade( entrada.nextInt() );

        System.out.println("Qual o sexo? ");
        objPessoa.setSexo( entrada.next().charAt(0) );
        ....
```

```
        // Dados do Endereço da DA PESSOA
        System.out.println("Qual a rua/av?");
        objEnd.setLogradouro(entradaString.nextLine());

        System.out.println("Qual o número?");
        objEnd.setNumero(entrada.nextInt());

        System.out.println("Tem complemento (s/n) ?");
        resp = entrada.next().charAt(0);
        if(resp == 's'){
            System.out.println("Qual o complemento?");
            objEnd.setComplemento(entradaString.nextLine());
        }else{
            objEnd.setComplemento("");
        }

        System.out.println("Qual o CEP: ");
        objEnd.setCep(entradaString.nextLine());

        //ASSOCIA O ENDERECO A PESSOA
        objPessoa.setEnd(objEnd);

        //Imprime dados da Pessoa
        System.out.println("\n===== DADOS DA PESSOA =====");
        System.out.println(objPessoa.imprimir());
    }
}
```


Acesso aos dados End

```
String cep;
....
cep = objPessoa.getEnd().getCep();
```

Exercícios



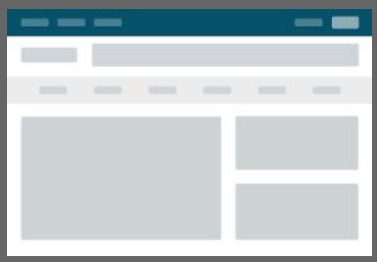
Exercício

Criar um relacionamento de Associação entre as classes Animal e Proprietario:

Animal "Tem UM" Proprietario

- Criar a classe **Animal** com os seguintes atributos:
  nome, raça, ano de nascimento e proprietário
- Criar a Classe **Proprietario** com os seguintes atributos:
  nome e telefone

Classe abstrata



abstract is a model

Definição

Um classe **abstrata** é uma superclasse que define uma forma generalizada para ser compartilhada com todas as subclasses.

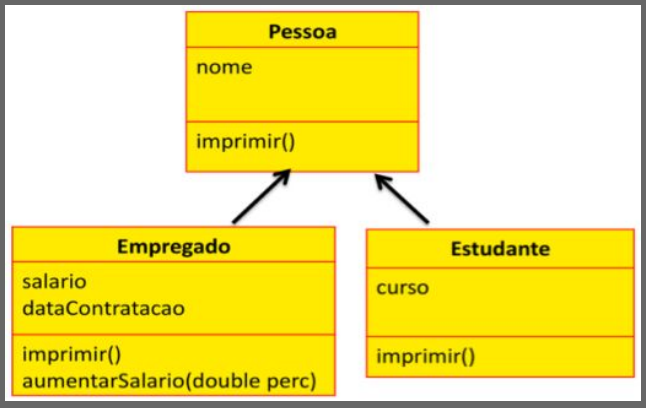
Classes abstratas **não** podem ser instanciadas!

É definida com a palavra-reservada ***abstract*** na definição

Ela permite definir métodos que não possuem implementação e que devem obrigatoriamente ser redefinidos nas subclasses. Estes métodos são também *abstratos*.

Toda classe que possui um método abstrato deve ser abstrata.

Exemplo



Exemplo

```
public abstract class Pessoa {

    private String nome;

    public Pessoa(String valorNome){
        nome = valorNome;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
    //método abstrato que retorna os dados da Pessoa
    public abstract String imprimir();

}
```

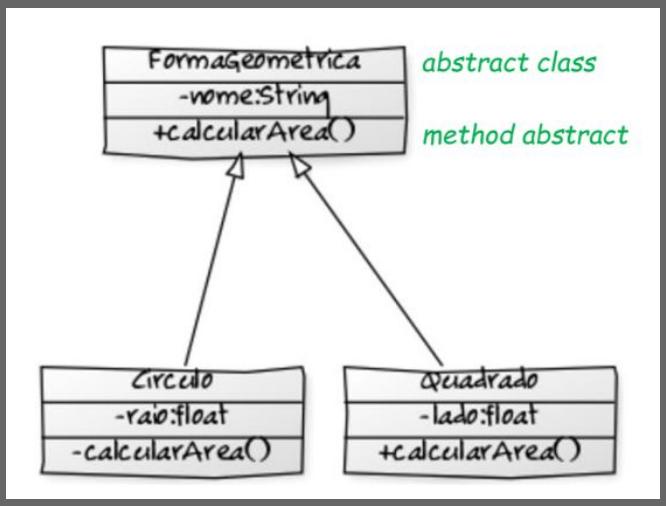
Exemplo

```java
public class Estudante extends Pessoa {

    private String curso;

    public Estudante(String valorNome, String valorCurso){
        super(valorNome);
        curso = valorCurso;
    }

    public String getCurso() {
        return curso;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    @Override
    public String imprimir(){
        return super.getNome()+" está cursando "+curso;
    }
}
```


Exemplo

```java
public class AppPessoa {

    public static void main (String args[]){

        Pessoa [] pessoas = new Pessoa[2];

        //preenche o vetor Pessoa com Estudante e Empregado
        pessoas[0] = new Estudante("Joao Vitor","Ciencia da Computacao");
        pessoas[1] = new Empregado("Manuela",5000,1980,6,1);

        //imprime os dados das pessoas
        for(Pessoa p: pessoas){
            System.out.println(p.imprimir());
        }
    }

}
```

Exemplo 2



Exemplo 2

```
public abstract class FormaGeometrica {
    //atributos
    private String nome;

    //construtores
    public FormaGeometrica() {} //default
    //sobrecarregado
    public FormaGeometrica(String nome) {
        this.nome = nome;
    }
.....
    //métodos da classe
    //método abstrato – deve ser implementado nas subclasses!
    public abstract float calcularArea() ;

    public String imprimir(){
        return "Forma: " + nome;
    }
}
```

Exemplo 2

```
public class Quadrado extends FormaGeometrica{
    //atributos
    private float lado;

    //construtores
    public Quadrado() { //default
        super();
    }
    //sobrecarregado
    public Quadrado(float lado, String nome) {
        super(nome);
        this.lado = lado;
    }

    //implementação do método abstrato
    @Override
    public float calcularArea() {
        return lado * lado;
    }

    @Override
    public String imprimir() {
        return super.imprimir() + "\nLado: " + lado;
    }
}
```


Exemplo 2

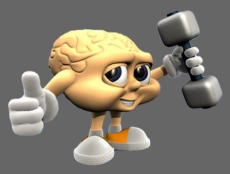
```
public class Circulo extends FormaGeometrica{
    //atributos
    private float raio;

    //construtores
    public Circulo () { //default
        super();
    }

    //sobrecarregado
    public Circulo(float raio, String nome) {
        super(nome);
        this.raio = raio;
    }

    //métodos da classe
    //implementação do método abstrato
    @Override
    public float calcularArea() {
        return (float) Math.PI * raio * raio;
    }

    @Override
    public String imprimir() {
        return super.imprimir() + "Raio: " + raio;
    }
}
```

Exemplo 2

```
public class AppFormas {
   public static void main(String[] args) {
      //uso do polimorfismo - um quadrado é uma forma
      //um círculo é uma forma
      FormaGeometrica quadrado = new Quadrado(8, "Quadradinho");
      FormaGeometrica circulo = new Circulo(5, "bola");

      System.out.println("==== Área do Quadrado ====");
      System.out.println("==> " + quadrado.imprimir());
      System.out.println("==> área: " + quadrado.calcularArea() +" m");

      System.out.println("==== Área do Círculo ====");
      System.out.println("==> " + circulo.imprimir());
      System.out.println("==> área: " + circulo.calcularArea() + " m");
   }
}
```

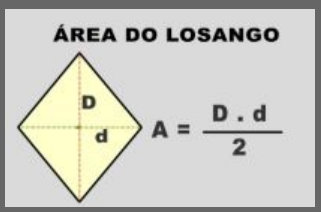

Let's code!

Exercícios

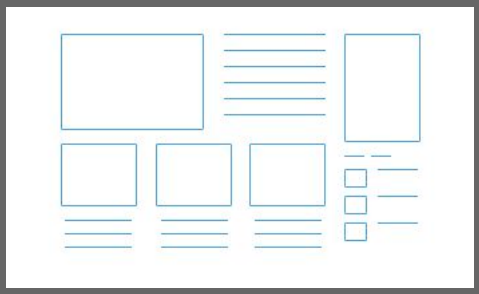


Exercício

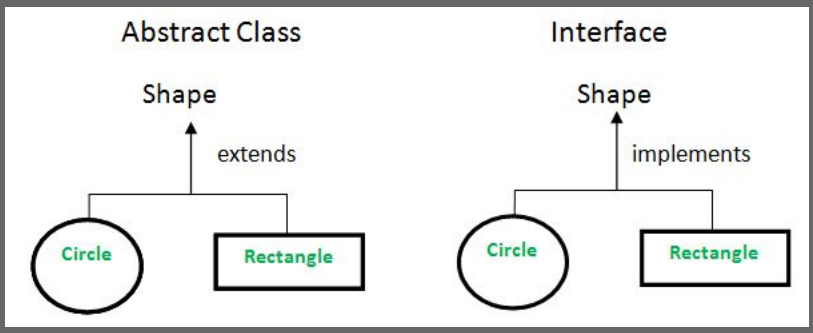
- Usando o exemplo da aula, incluir na hierarquia a classe Losango com os seguintes atributos: diagonal maior e diagonal menor
- Reescrever o método calcularArea()



Interface



interface is abstract



Interface

Um **Interface** é bastante similar à uma classe abstrata em alguns aspectos.

Interfaces **não** podem ser instanciadas, são implicitamente **abstratas** assim como ***todos*** os métodos!

É definida com a palavra-reservada ***interface***

Ela permite definir métodos que não possuem implementação e que devem obrigatoriamente ser redefinidos nas ***classes*** que a **implementam**. (NÃO formam uma hierarquia – herança).

Normalmente adotada dentro de uma equipe de desenvolvimento de software como um CONTRATO.

Interface

```
public interface IMyInterface {
	public void sayHello();
}
```

```
public class MyInterfaceImpl implements IMyInterface {
	String hello = "Hello";

	public void sayHello() {
		System.out.println(hello);
	}
}
```

```
IMyInterface myInterface = new MyInterfaceImpl();

myInterface.sayHello();
```

Interface

```
import com.package1.MyInterface;
import com.package2.MyOtherInterface;

public class MyInterfaceImpl implements MyInterface, MyOtherInterface {
    ...

}
```


Considerações

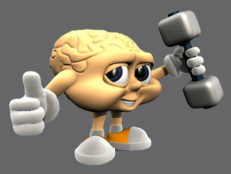
Considere usar **classes abstratas** quando:

- você quer compartilhar código entre classes relacionadas.
- você espera que classes filha tenham muitos comportamentos ou atributos comuns, ou tenham modificadores que não serão públicos.
- você quer declarar atributos que não são static ou final.

Considere usar **Interface** quando:

- você espera que classes não relacionadas usem a interface.
- você quer especificar um comportamento padrão, mas não quer implementar esse comportamento.
- você quer usar múltiplas “heranças”.

Exercícios



Exercício 1

Defina a interface FormaGeometrica que possue o método calculaArea.

Construir a classe Quadrado e a classe Circulo que implementam a interface FormaGeometrica.

Implementar programa de testes que declara duas variáveis do tipo FormaGeometrica e instancia dois objetos, um do tipo Circulo e outro do tipo Quadrado.

Exibir a área dos objetos instanciados.