

Orientação a Objetos

Lucas Alberto Schlestein



Revisão Prova Nivelamento



Programação Orientada a Objetos





Apresentação do Conteúdo

Introdução à Programação Java

Orientação a Objetos:

- Classe
- Objeto
- Construtor
- Modificadores de Acesso
- Encapsulamento
- Interface
- Herança
- Polimorfismo
- Classes abstratas

Visão Geral

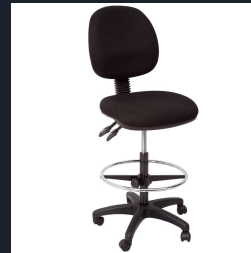
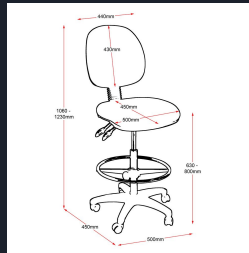
As entidades que colaboram nas funcionalidades de um sistema são suas Classes.

Classe é uma abstração do mundo real.

Cada classe tem suas características. Por exemplo, se pensarmos em uma classe Cadeira.

Existem vários tipos de cadeiras, cada uma com suas características, mas nem por isso deixando de ser uma cadeira;

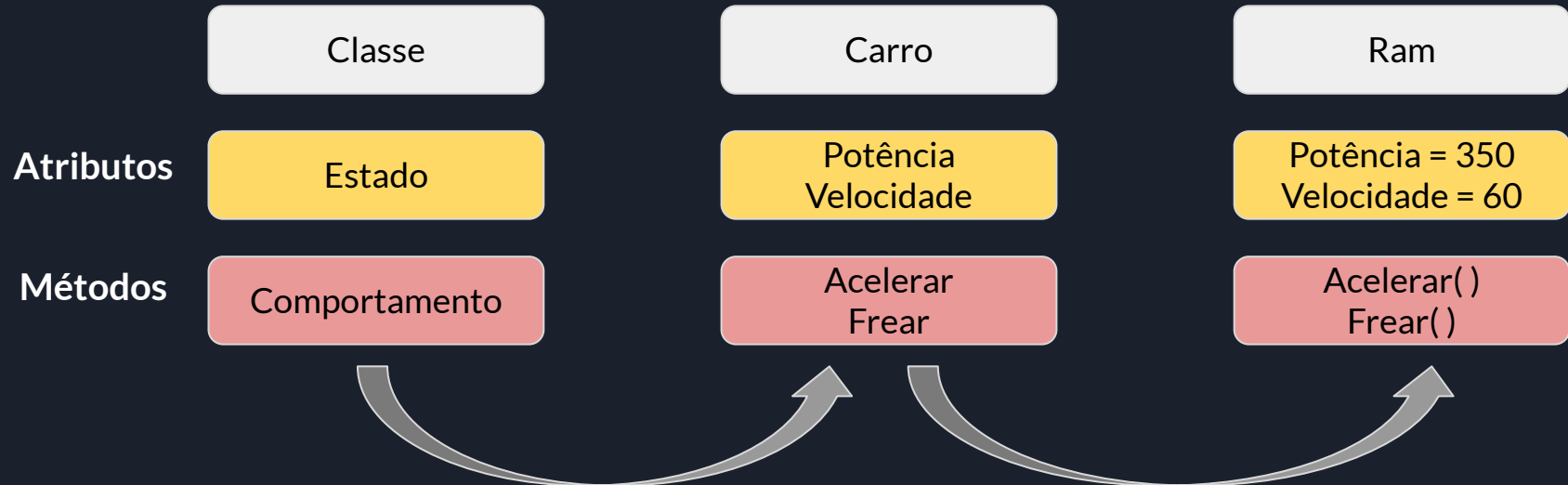
Instância concreta de uma classe. Objeto = instância da Classe.



Visão Geral

Características do objeto = Atributos

Comportamento do objeto = métodos





Visão Geral

Características do objeto = Atributos

Comportamento do objeto = métodos

Classe

Abstrato

Tem atributos

Tem ações

Objeto

Concreto

Tem valores

Executa ações

Relacionamento entre Classes

Sistema para uma pizzeria:

Sabemos que normal, o valor de uma **pizza** varia de acordo com seus ingredientes.

Também, ao fazer o pedido, a pessoa pode querer outros itens, que podem ser adicionado ao **carrinho**.

Também sabemos, que o valor da **entrega**, é definido pela distância até o endereço do cliente.



Relacionamento entre Classes

Pizza

Calcula o preço da pizza baseado na quantidade de ingredientes.

Carrinho

Calcula o valor total da compra com a entrega.

Entrega

Calcula o valor da entrega, conforme a distância e o dia da semana.



Programação Orientada a Objetos

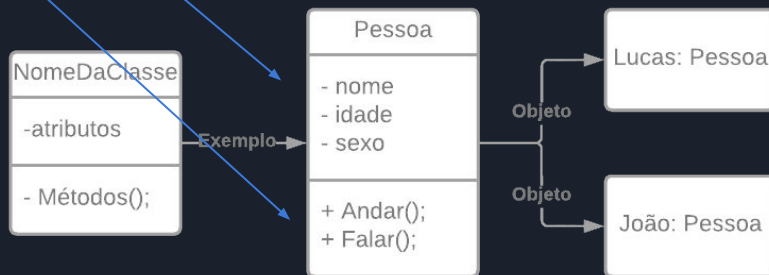
Conceitos da POO:

- Em orientação a objetos, uma classe é uma estrutura de código utilizada para representar um conjunto de objetos do mundo real
- A classe define o comportamento dos objetos através de **atributos** e **métodos**:

- **Atributos** = variáveis na linguagem estruturada

- **Métodos** = funções na linguagem estruturada

A classe descreve as características e funcionalidades dos objetos:

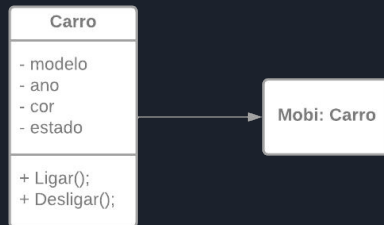


Programação Orientada a Objetos

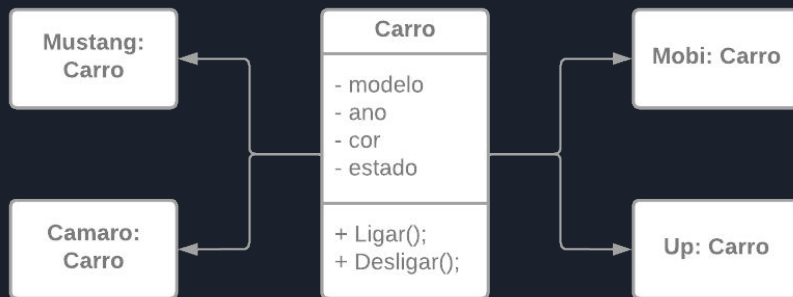
Conceitos da POO:

O objeto é a instância de uma classe, ou uma classe instanciada

Ex: *Classe* carro *objeto* Mobi



Uma classe pode dar origem a vários objetos (instâncias)





Programação Orientada a Objetos

Estado e Comportamento:

- O *estado* de um objeto é representado por seus *atributos*
- O *comportamento* é representado pelos seus *métodos*

Exemplo:

Classe Carro
Objeto Mobi

- Atributos: modelo, ano, cor, estado
- Métodos: ligar, desligar



Programação Orientada a Objetos

Encapsulamento:

- É o empacotamento dos atributos e métodos numa classe

- *Proteção* dos dados:

Público (+)

Privado (-)

Protegido (#)

Carro
+ modelo - ano - cor # estado
+ Ligar(); + Desligar();

Programação Orientada a Objetos

Encapsulamento:

- É o empacotamento dos atributos e métodos numa classe

- *Proteção* dos dados:

Público (+)

Acesso público, qualquer classe acessa seus atributos

Privado (-)

Acesso privado a própria classe, somente ela mesma pode acessar seus atributos.

Protegido (#)

Dá acesso à própria classe, e as subclasses que a herdarem, no mesmo pacote..

Default()

Quando não definido o acesso é default. Dessa forma somente as classes no mesmo pacote tem acesso ao métodos ou atributos.

Carro
+ modelo - ano - cor # estado fabricante
+ ligar() + desligar()

Programação Orientada a Objetos

Abstração:

- Definir o que é (e o que não é) importante
- Classificar o que é útil naquele momento
- Extrair características essenciais
- Representa um modelo da realidade
 - "Cuidar para não ser abstrato demais"
- Classes abstratas não podem ser instanciadas
 - Apenas possui atributos e métodos que encapsulam uma classe, análoga a um rascunho
 - Elas devem ser herdadas e implementadas

EnviarEmail

- destinatario
- assunto
mensagem

+ enviar()



Programação Orientada a Objetos

Polimorfismo:

A palavra polimorfismo vem do grego:

poli = muitas, *morphos* = formas

Em outras palavras, **muitas formas**

Polimorfismo permite “programar no geral” ao invés de “programar no específico”.

Exemplos de polimorfismo em Java são:

- Sobrescrita de métodos (**Override**)

Métodos, parâmetros e retornos iguais, implementação de código diferentes

- Sobrecarga de métodos (**Overload**)

Métodos de mesmo nome, porém com parâmetros e implementações diferenciadas



Programação Orientada a Objetos

Resumo:

Classe: representa um conjunto de objetos

Subclasse: classe filha que herda os atributos e os métodos da classe mãe (Herança)

Objeto: instância de uma classe

Atributo: características do objeto

Método: funcionalidades do objeto

Mensagem: troca de informação entre os objetos

Associação: utilização de recursos entre objetos

Abstração: classe não instanciável (rascunho)

Polimorfismo: métodos com muitas formas: sobrecarga, sobrescrita

Herança: capacidade de herdar características e ações (métodos e atributos) de outra classe, ou classe mãe



Programação Orientada a Objetos

Perguntas e Respostas:

- O que é uma Classe?
- O que é um Objeto?
- O que é Herança?
- O que é Polimorfismo?
- Como Proteger Classes?
- O que é um Método?
- O que é um Atributo?

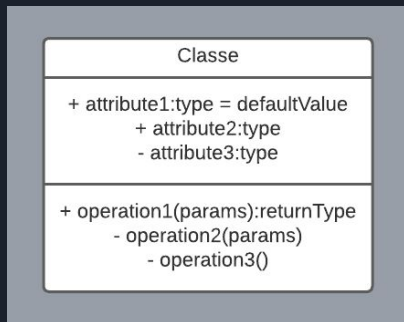
Programação Orientada a Objetos

UML

Compartimento superior: Nome da classe

Compartimento intermediário: Atributos da classe

Compartimento inferior: Métodos da classe



Fonte: lucidchart.com



Programação Orientada a Objetos

Modelagem Orientada a Objetos:

Modelagem Orientada a Objetos Linguagem

UML - Padrão para modelagem de sistemas

- Amplamente difundida e consolidada
- Largamente utilizada na indústria de software

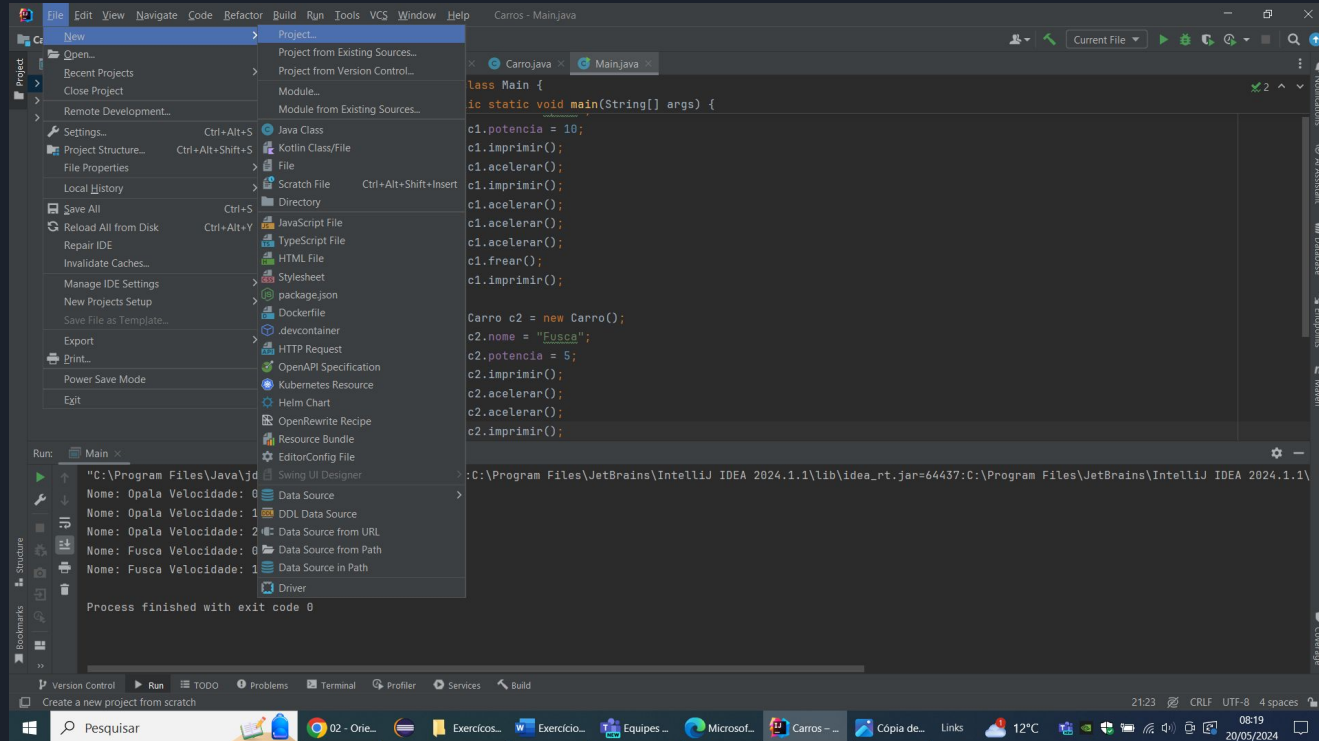
Mais informações em: <http://www.uml.org>

Sugestão de SW para Modelagem

<https://www.lucidchart.com/>

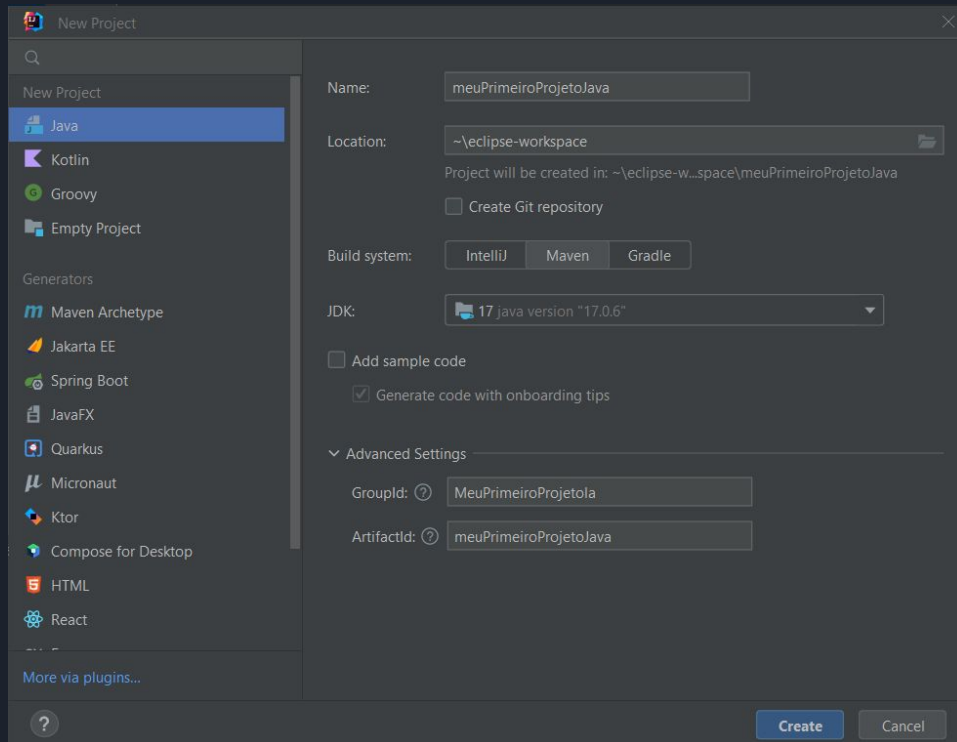


Criando a Primeira Classe IDE IntelliJ IDEA



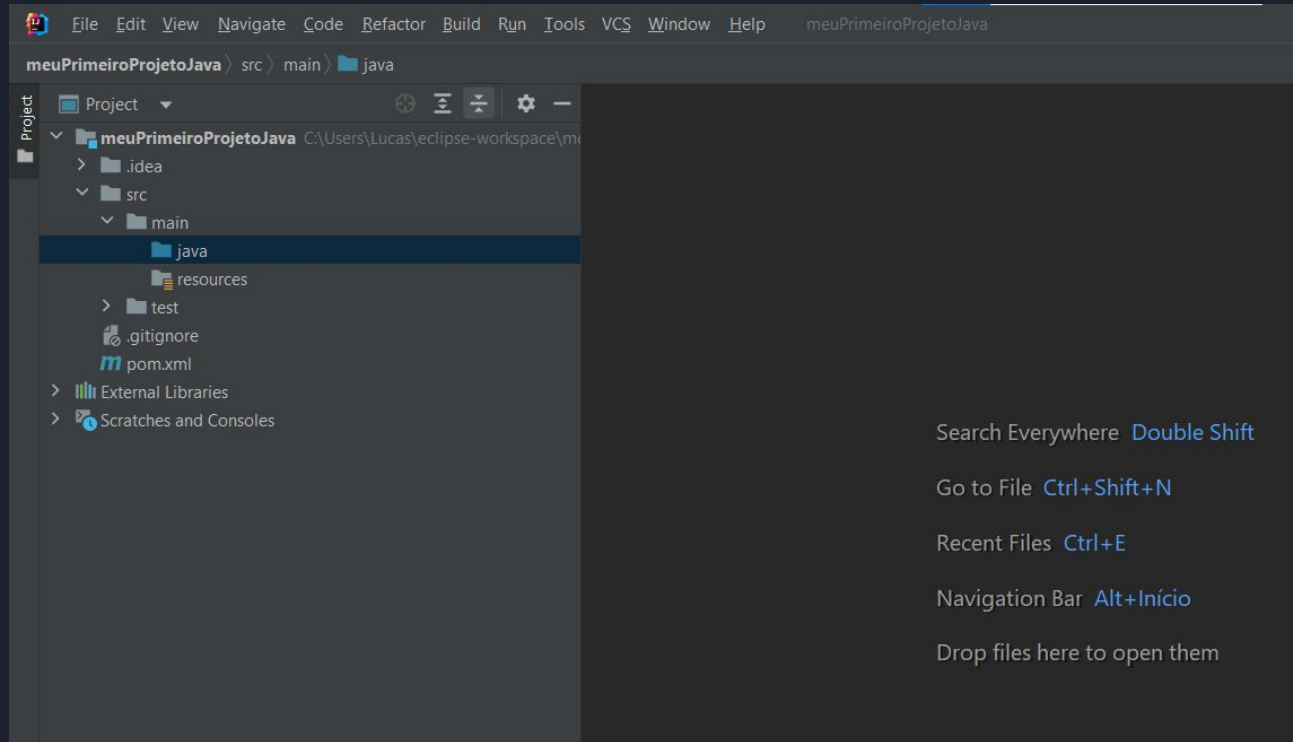
Criando a Primeira Classe

IDE IntelliJ IDEA



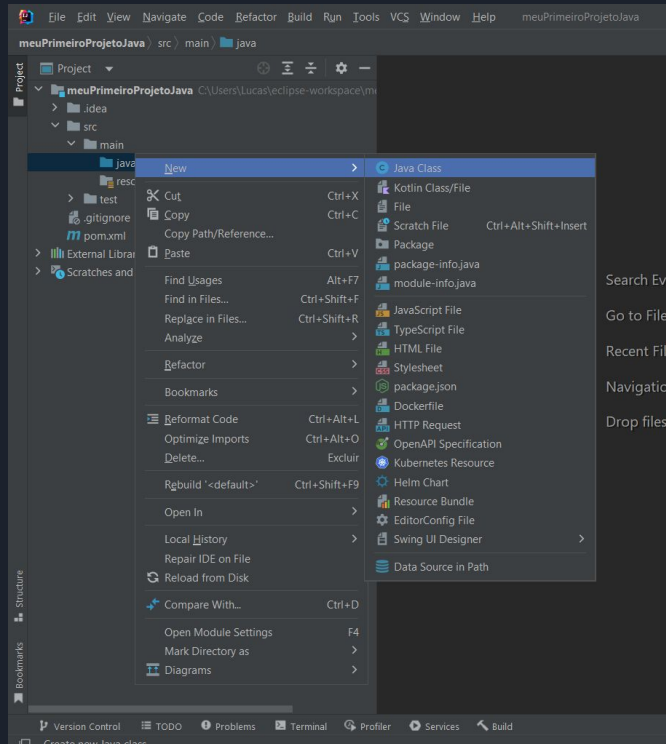
Criando a Primeira Classe

IDE IntelliJ IDEA



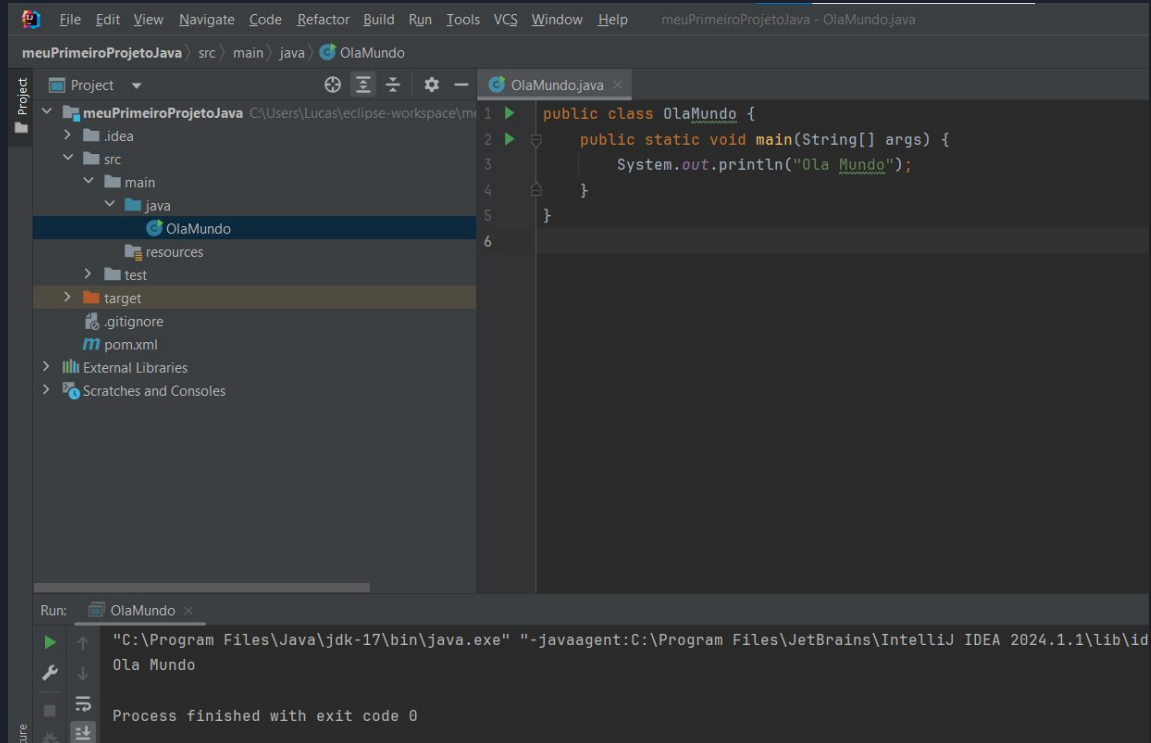
Criando a Primeira Classe

IDE IntelliJ IDEA



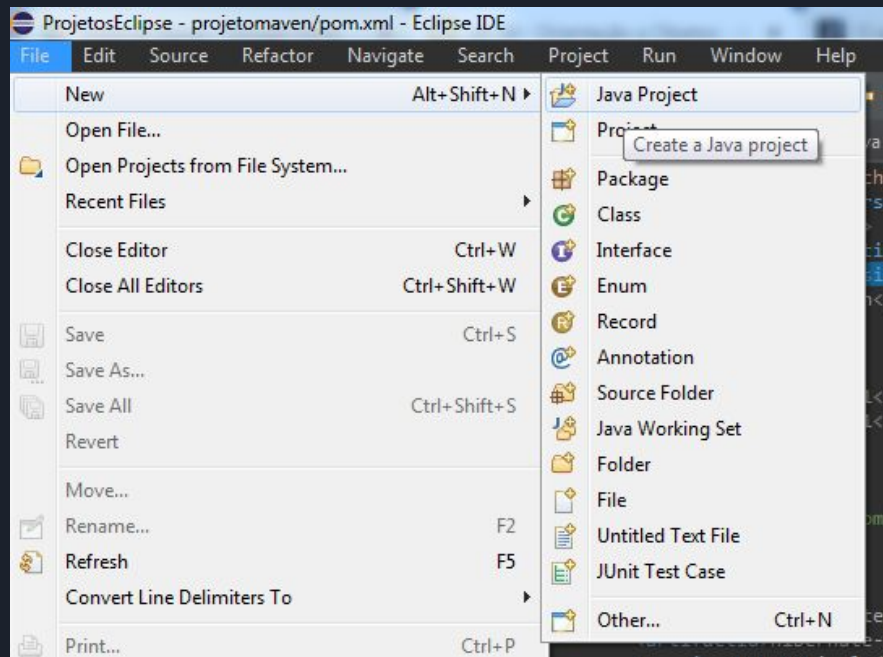
Criando a Primeira Classe

IDE IntelliJ IDEA



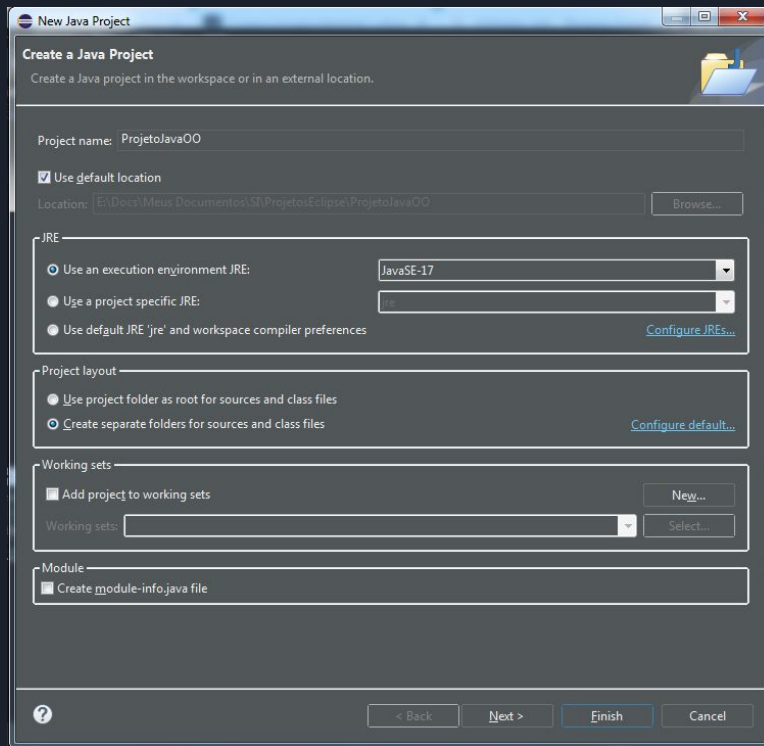
Criando a Primeira Classe

IDE Eclipse



Criando a Primeira Classe

IDE Eclipse



The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The dialog is titled 'New Java Project' and has a subtitle 'Create a Java project in the workspace or in an external location.' The 'Project name' field is set to 'ProjetoJavaOO'. The 'Use default location' checkbox is checked, and the 'Location' field shows the default path 'E:\Docs\Meus Documentos\SE\ProjetoEclipse\ProjetoJavaOO'. The 'JRE' section has three radio buttons: 'Use an execution environment JRE:' (selected), 'Use a project specific JRE:', and 'Use default JRE 'jre' and workspace compiler preferences'. The 'Project layout' section has two radio buttons: 'Use project folder as root for sources and class files' and 'Create separate folders for sources and class files'. The 'Working sets' section has a checkbox 'Add project to working sets' and a 'Working sets' dropdown menu. The 'Module' section has a checkbox 'Create module-info.java file'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

Create a Java Project
Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location
Location: [Browse...](#)

JRE

☒ Use an execution environment JRE: [Configure JREs...](#)

☐ Use a project specific JRE:

☐ Use default JRE 'jre' and workspace compiler preferences

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

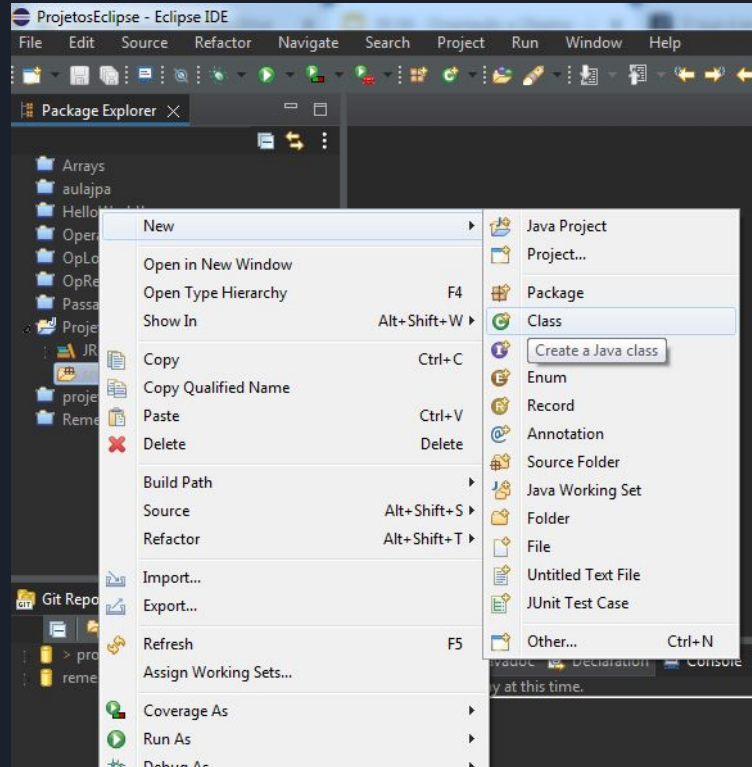
Module

☐ Create module-info.java file

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

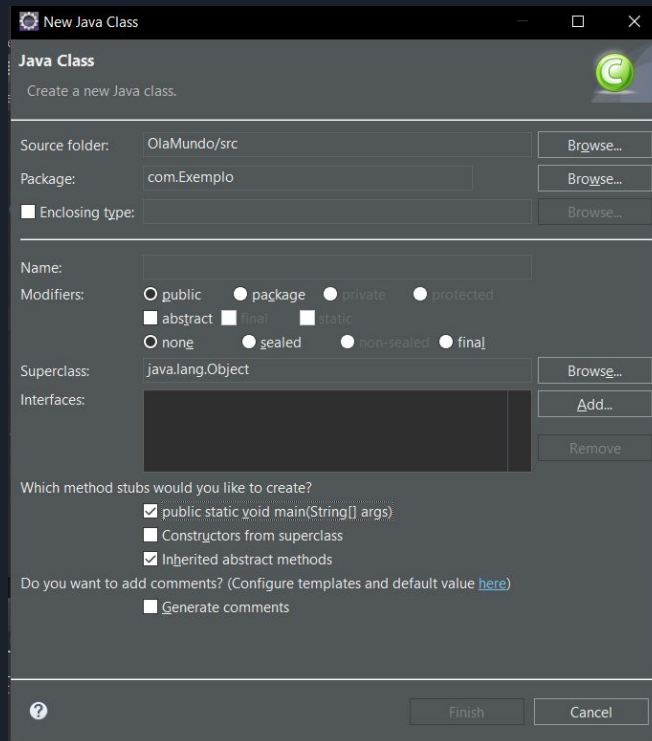
Criando a Primeira Classe

IDE Eclipse



Criando a Primeira Classe

IDE Eclipse



The screenshot shows the 'New Java Class' dialog box in the Eclipse IDE. The dialog is titled 'New Java Class' and has a 'Java Class' subtitle. It contains several fields and options for creating a new Java class.

Source folder: OlaMundo/src **Browse...**

Package: com.Exemplo **Browse...**

Enclosing type: **Browse...**

Name:

Modifiers:

- ☐ public ☒ package ☐ private ☐ protected
- ☐ abstract ☐ final ☐ static
- ☐ none ☒ sealed ☐ non-sealed ☐ final

Superclass: java.lang.Object **Browse...**

Interfaces: **Add...** **Remove**

Which method stubs would you like to create?

- ☒ public static void main(String[] args)
- ☐ Constructors from superclass
- ☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments

Finish **Cancel**

Introdução a Linguagem Java

Fundamentos

Primeira Classe em Java

Nome da Classe

Static: palavra-chave JVM

Void: Retorno do método, quando não retorna deve ser void, vazio, e precisa ser especificado.

Main: Método principal da classe. Para que o interpretador entenda que a mesma deve ser executada.

String[] args: utilizado para passar parâmetros para a classe, no momento da execução.

```
1 public class Principal {
2
3
4     public static void main(String[] args) {
5         System.out.println("Minha Primeira Classe em Java");
6     }
7
8 }
9
```

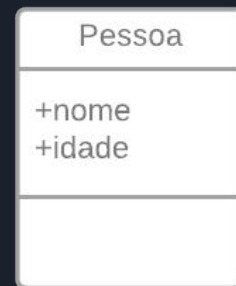
Código a ser executado dentro do método principal 'main'.

Classes e Atributos

Primeira Classe em Java

- Qual é o nome da classe?
- Qual é o nome do arquivo .java?
- A classe pode ter um nome diferente do arquivo?
- A classe possui atributos? Métodos? Quais?
- O que faz essa classe?
- Ela é executável?

```
Pessoa.java x
1
2 public class Pessoa {
3     public String nome;
4     public int idade;
5
6 }
7
```



Classes e Atributos

- Instanciar é o mesmo que criar um novo objeto
- Qual é o nome do objeto criado?
- Qual o resultado da execução da classe Principal?

```
Pessoa.java X
1
2 public class Pessoa {
3     public String nome;
4     public int idade;
5
6 }
7
```

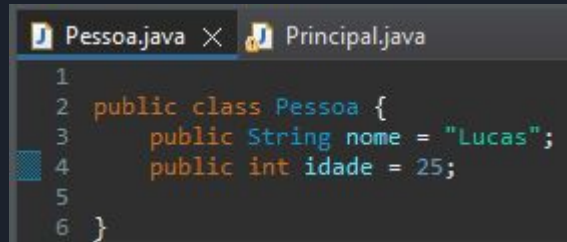
```
Pessoa.java Principal.java X
1
2 public class Principal {
3
4     public static void main(String[] args) {
5         Pessoa pessoa = new Pessoa();
6         System.out.println("A classe Pessoa foi instanciada");
7
8     }
9
10 }
```




Classes e Atributos

Classe `Pessoa` com valores nos atributos

Atribuindo valores padrão (default) a classe `Pessoa`.



```
Pessoa.java x Principal.java
1
2 public class Pessoa {
3     public String nome = "Lucas";
4     public int idade = 25;
5
6 }
```



Classes e Atributos

Instanciando a classe Pessoa e obtendo o valor de seus atributos

```
public class Principal {  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa();  
        System.out.println("O nome da Pessoa é: " + pessoa.nome);  
        System.out.println("E sua idade é: " + pessoa.idade);  
    }  
}
```

- Qual o resultado da execução da classe **Principal**?
- Acesso direto aos atributos e aos valores dos atributos. Mas como é possível?

Pelo fato dos atributos estarem definidos como **public**.

Se os atributos estivessem definidos com acesso **private**, somente a própria classe poderia ter acesso a eles.

Classes e Atributos

Atribuindo valores após a criação de objetos.

```
Pessoa.java X
1
2 public class Pessoa {
3     public String nome;
4     public int idade;
5
6 }
7
```

```
Pessoa.java Principal.java X
1
2 public class Principal {
3
4     public static void main(String[] args) {
5         Pessoa pessoa1 = new Pessoa();
6         pessoa1.nome = "Pedro";
7
8         Pessoa pessoa2 = new Pessoa();
9         pessoa2.nome = "Carlos";
10
11         System.out.println("O nome da Pessoa 1 é: " + pessoa1.nome);
12         System.out.println("O nome da Pessoa 2 é: " + pessoa2.nome);
13
14     }
15
16 }
17
```

Qual é o resultado da execução da classe Principal?



Classes e Atributos

Pergunta: mas se a Classe deve ser um “molde”, todos os objetos desse molde sairão com os valores default quando instanciados? Isso é correto? É o que espera-se?

Normalmente, cria-se um objeto e, após, define-se os valores dos atributos.

A ideia é: cada objeto tenha seus atributos ou comportamentos configurados de acordo com a necessidade, após a instanciação (criação do objeto), e não que a classe já passe esses valores no momento da instanciação.

Podem ocorrer variações, cada caso é um caso.

Classes e Atributos

Exercício 1

Criar três objetos do tipo Pessoa, e exibir apenas a idade dessas pessoas:

```
Pessoa.java  Principal.java X
1
2 public class Principal {
3
4     public static void main(String[] args) {
5         Pessoa pessoa1 = new Pessoa();
6         pessoa1.idade = 18;
7
8         Pessoa pessoa2 = new Pessoa();
9         pessoa2.idade = 25;
10
11        Pessoa pessoa3 = new Pessoa();
12        pessoa3.idade = 23;
13
14        System.out.println("A idade da Pessoa 1 é: " + pessoa1.idade);
15        System.out.println("A idade da Pessoa 2 é: " + pessoa2.idade);
16        System.out.println("A idade da Pessoa 3 é: " + pessoa3.idade);
17
18    }
19
20 }
```

```
A idade da Pessoa 1 é: 18
A idade da Pessoa 2 é: 25
A idade da Pessoa 3 é: 23
```



Classes e Atributos

Exercício 2

Crie uma classe chamada **Professor** que contenha um atributo público chamado nome, do tipo String. Crie também uma classe chamada **Laboratorio**, que contenha um atributo público, chamado local, do tipo String.

Além disso, crie uma classe executável chamada **Disciplina**, responsável por instanciar as classes Professor e Laboratorio, definindo valor aos atributos e exibindo na tela o resultado da criação desses objetos.

Por exemplo: o resultado da execução da classe Disciplina deve ser algo similar com:

O nome do professor é: Lucas Schlestein

O local da aula é: Sala 2

Classes e Atributos

Exercício 2

```
1
2 public class Professor {
3     public String nome;
4 }
```

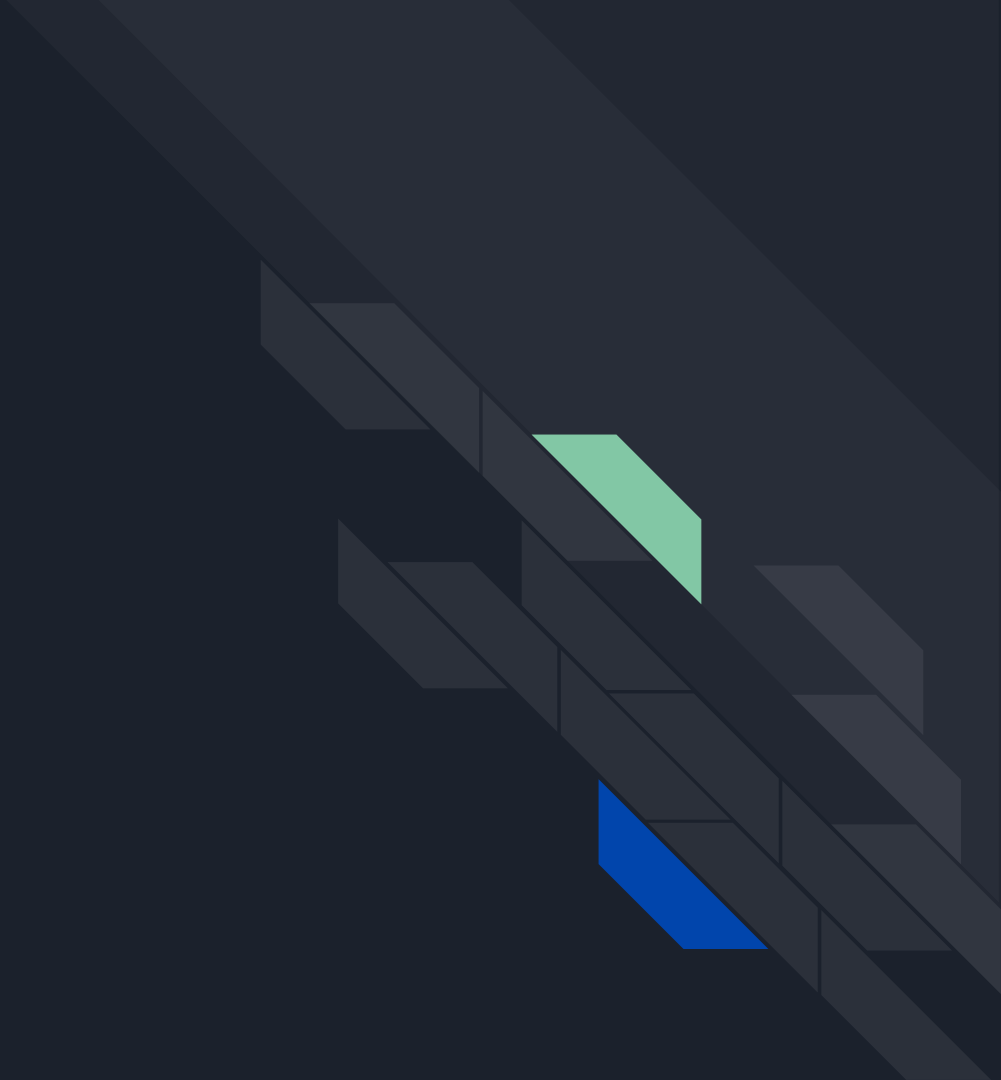
```
1
2 public class Laboratorio {
3     public String local;
4 }
5
```

```
1
2 public class Disciplina {
3
4     public static void main(String[] args) {
5         Professor professor = new Professor();
6         professor.nome = "Lucas Schlestein";
7         Laboratorio laboratorio = new Laboratorio();
8         laboratorio.local = "Sala 2";
9
10        System.out.println("O nome do professor é: " + professor.nome);
11        System.out.println("O local de aula é: " + laboratorio.local);
12    }
13 }
```

Resultado da execução:

```
O nome do professor é: Lucas Schlestein
O local de aula é: Sala 2
```

Métodos



Classes, Atributos e Métodos

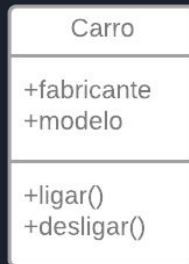
Classe em Java Com atributos e métodos

- Qual é o nome da classe?
- A classe possui atributos? Métodos? Quais?
- O que faz essa classe?
- Ela é executável?

```
Carro.java X
public class Carro {
    public String fabricante;
    public String modelo;

    public void ligar() {
        System.out.println("Carro ligado");
    }

    public void desligar() {
        System.err.println("Carro desligado");
    }
}
```



Classes, Atributos e Métodos

Instanciando a classe carro

Carro.java X

```
public class Carro {  
    public String fabricante;  
    public String modelo;  
  
    public void ligar() {  
        System.out.println("Carro ligado");  
    }  
    public void desligar() {  
        System.err.println("Carro desligado");  
    }  
}
```

- Qual o objeto criado?
- Qual o resultado da execução da classe Principal (LocadoraVeiculos)?

LocadoraVeiculos.java X

```
1 |  
2 public class LocadoraVeiculos {  
3     public static void main(String[] args) {  
4         Carro carro = new Carro();  
5         carro.ligar();  
6         carro.desligar();  
7     }  
8 }  
9
```

Classes, Atributos e Métodos

Definindo **valores aos atributos** e criando um **método** para exibir esses valores


```
Carro.java X
1 public class Carro {
2     public String fabricante = "VW";
3     public String modelo = "Jetta";
4
5     public void ligar() {
6         System.out.println("Carro ligado");
7     }
8
9     public void desligar() {
10        System.out.println("Carro desligado");
11    }
12
13    public void exibirDados() {
14        System.out.println("Dados do carro: " + fabricante + " | " + modelo);
15    }
16 }
```

Classes, Atributos e Métodos

Instanciando a classe carro e exibindo seu dados

- Qual o objeto criado?
- Qual o resultado da execução da classe Principal (LocadoraVeiculos)?
- Como exibir os dados do objeto?

```
LocadoraVeiculos.java X
1
2 public class LocadoraVeiculos {
3     public static void main(String[] args) {
4         Carro carro = new Carro();
5         carro.exibirDados();
6     }
7 }
```



Classes, Atributos e Métodos

Exercício 3

Crie um método para atribuir valores para *fabricante* e *modelo* do carro, sendo que esses valores são recebidos pelo método através de parâmetros. Após, implemente a classe **LocadoraVeiculos** para instanciar a classe **Carro**, chamar o método correspondente e passar os parâmetros necessários. Além de um método para atribuir os valores, crie um método que retorna os valores atribuídos. O resultado da execução da classe LocadoraVeiculos deve ser:

Dados do carro: VW Gol

Classes, Atributos e Métodos

Exercício 3

Carro.java

```
1 public class Carro {
2     public String fabricante = "VW";
3     public String modelo = "Jetta";
4     public void configuraDados(String fabricanteCarro, String modeloCarro) {
5         fabricante = fabricanteCarro;
6         modelo = modeloCarro;
7     }
8     public void exibirDados() {
9         System.out.println("Dados do carro: " + fabricante + " " + modelo);
10    }
11 }
```

Carro.java

LocadoraVeiculos.java

```
1
2 public class LocadoraVeiculos {
3     public static void main(String[] args) {
4         Carro carro = new Carro();
5         carro.configuraDados("Vw", "Gol");
6         carro.exibirDados();
7     }
8 }
```



Classes, Atributos e Métodos

Exercício 4

Crie uma classe chamada *Moto* com três atributos (*marca*, *modelo* e *cilindradas*) e dois métodos (atribuir valores e retornar valores).

Na classe **LocadoraVeiculos**, crie um objeto do tipo *Carro* e dois objetos do tipo *Moto*, sendo que os objetos serão criados de acordo com a solicitação desses dados ao usuário, via linha de execução, conforme exemplo abaixo.

Após a criação dos objetos, utilize o método para retornar valores e exiba na tela o conteúdo dos objetos criados

```
1 import java.util.Scanner;
2
3 public class LocadoraVeiculos {
4     public static void main(String[] args) {
5         Scanner leitura = new Scanner(System.in);
6         System.out.println("Insira o Fabricante do Veículo:");
7         String fabricante = leitura.nextLine();
```

Construtores





Construtores

Um método **construtor** pode ser utilizado para inicializar um objeto de uma classe.

O Java requer uma chamada de **construtor** para todo o objeto que é criado. Por padrão, o compilador fornece um **construtor-padrão** sem parâmetros em qualquer classe que não inclua explicitamente um **construtor**.

Ao declarar uma classe, é possível fornecer seu próprio **construtor** a fim de especificar a inicialização personalizada para os objetos da classe



Construtores

A palavra-chave **new** chama o **construtor** da classe para realizar a inicialização.

Ou seja, quando uma classe é instanciada, o primeiro **método** a ser executado é o **construtor** daquela classe.

A chamada de um **construtor** é indicada pelo nome da classe seguido por parênteses.

O nome do construtor deve ser igual ao nome da classe.



Construtores

Construtores não podem retornar valor, e portanto, não se declara o tipo de retorno em um método desse tipo;

Quando não declaramos pelo menos um construtor em nossa classe, a linguagem define um construtor padrão vazio, que é declarado automaticamente em todas as classes.



Construtores

- Qual o nome da classe?
- Ela possui construtor?
- O que faz a classe?

```
*Carro.java X
1 public class Carro {
2     private String modelo;
3
4     public void exibirModelo() {
5         System.out.println("Dados do carro: " + modelo);
6     }
7 }
```



Construtores

- Qual o nome da classe?
- Ela possui construtor?
- O que faz a classe?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro();
4         carro.exibirModelo();
5     }
6 }
7
```

Construtores

- Qual o nome da classe?
- Ela possui construtor?
- O que faz a classe?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro();
4         carro.exibirModelo();
5     }
6 }
7
```

Dados do carro: null



Construtores

- Qual o nome da classe?
- Ela possui construtor?
- O que faz a classe?

```
Carro.java X
1 public class Carro {
2     private String modelo;
3
4     public Carro() {
5         modelo = "Jetta";
6     }
7
8     public void exibirModelo() {
9         System.out.println("Dados do carro: " + modelo);
10    }
11 }
```



Construtores

- Qual o nome da classe?
- Ela possui construtor?
- O que faz a classe?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro();
4         carro.exibirModelo();
5     }
6 }
7
```




Construtores

- Ela possui construtor?
- O que faz a classe?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro();
4         carro.exibirModelo();
5     }
6 }
7
```

Dados do carro: Jetta

Construtores e Parâmetros

```
Carro.java X
1 public class Carro {
2     private String modeloCarro;
3
4     public Carro(String modelo) {
5         modeloCarro = modelo;
6     }
7
8     public void exibirModelo() {
9         System.out.println("Dados do carro: " + modeloCarro);
10    }
11 }
```

- A classe possui construtor?
- O construtor possui parâmetro?

Qual?



Construtores

```
Principal.java ×  
1 public class Principal {  
2     public static void main(String[] args) {  
3         Carro carro = new Carro("Opala");  
4         carro.exibirModelo();  
5     }  
6 }
```

- Utilizando a classe Principal.
- Qual parâmetro foi passado ao construtor?
- Qual o resultado esperado?



Construtores

```
Principal.java ×  
1 public class Principal {  
2     public static void main(String[] args) {  
3         Carro carro = new Carro("Opala");  
4         carro.exibirModelo();  
5     }  
6 }
```

- Utilizando a classe Principal.
- Qual parâmetro foi passado ao construtor?
- Qual o resultado esperado?

```
Dados do carro: Opala
```

Construtores e Parâmetros

- A classe possui construtor?
- Quais os atributos dessa classe?

```
Carro.java X
1 public class Carro {
2     private String modeloCarro;
3     public String corCarro = "Vermelho";
4
5     public Carro(String modelo, String cor) {
6         modeloCarro = modelo;
7         corCarro = cor;
8     }
9
10    public void exibirModelo() {
11        System.out.println("O modelo do carro é: " + modeloCarro);
12        System.out.println("A cor do carro: " + corCarro);
13    }
14 }
```

Construtores

- Utilizando a classe Principal.
- Quais valores foram passados ao construtor?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro("Fusca", "Preto");
4         System.out.println(carro.corCarro);
5         carro.exibirModelo();
6     }
7 }
8
```

Construtores

- Utilizando a classe Principal.
- Quais valores foram passados ao construtor?
- Qual o resultado esperado?

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro = new Carro("Fusca", "Preto");
4         System.out.println(carro.corCarro);
5         carro.exibirModelo();
6     }
7 }
8
```

```
Preto
O modelo do carro é: Fusca
A cor do carro: Preto
```

Construtores: Vários construtores

```
Carro.java X
1 public class Carro {
2     private String modeloCarro;
3     public String corCarro = "Vermelho";
4
5     public Carro(String modelo, String cor) {
6         modeloCarro = modelo;
7         corCarro = cor;
8     }
9
10    public Carro(String modelo) {
11        modeloCarro = modelo;
12    }
13
14    public void exibirModelo() {
15        System.out.println("O modelo do carro é: " + modeloCarro);
16        System.out.println("A cor do carro: " + corCarro);
17    }
18 }
```

É possível uma classe possuir mais de um construtor? Mas como, com o mesmo nome?

A diferenciação é feita pelo número e tipo de parâmetros em cada construtor, no momento da criação de cada objeto.

Construtores: Vários construtores

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro1 = new Carro("Fusca");
4         carro1.exibirModelo();
5         Carro carro2 = new Carro("Onix", "Branco");
6         carro2.exibirModelo();
7     }
8 }
```

Utilizando a classe **Principal** novamente:

Qual o construtor que será chamado?

Várias instâncias da mesma classe? Qual o resultado?

Construtores: Vários construtores

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro1 = new Carro("Fusca");
4         carro1.exibirModelo();
5         Carro carro2 = new Carro("Onix", "Branco");
6         carro2.exibirModelo();
7     }
8 }
```

Utilizando a classe **Principal** novamente:

Qual o construtor que será chamado?

Várias instâncias da mesma classe? Qual o resultado?

```
O modelo do carro é: Fusca
A cor do carro: Vermelho
O modelo do carro é: Onix
A cor do carro: Branco
```

Construtores: Vários construtores

Além do número de parâmetros, o **construtor** também considera o tipo do parâmetro.

```
Carro.java X
1 public class Carro {
2     private String modeloCarro;
3     public String corCarro = "Vermelho";
4     private int anoCarro;
5
6     public Carro(String modelo, String cor) {
7         modeloCarro = modelo;
8         corCarro = cor;
9     }
10
11     public Carro(String modelo) {
12         modeloCarro = modelo;
13     }
14
15     public String Carro(int ano) {
16         anoCarro = ano;
17         return "O ano do carro é: " + anoCarro;
18     }
19
20     public void exibirModelo() {
21         System.out.println("O modelo do carro é: " + modeloCarro);
22         System.out.println("A cor do carro: " + corCarro);
23     }
24 }
```

Construtores: Vários construtores

```
Principal.java ×  
1 public class Principal {  
2     public static void main(String[] args) {  
3         Carro carro1 = new Carro("Fusca");  
4         carro1.exibirModelo();  
5         System.out.println(carro1.Carro(1994));  
6     }  
7 }
```

Utilizando a classe **Principal** novamente:

Qual o resultado esperado?

```
O modelo do carro é: Fusca  
A cor do carro: Vermelho  
O ano do carro é: 1994
```

Copiar Objetos

```
Principal.java ×
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro1 = new Carro("Fusca");
4         Carro carro2 = carro1;
5         carro2.exibirModelo();
6     }
7 }
```

Utilizando a classe **Principal** novamente:

Qual o resultado esperado?

Copiar Objetos

```
Principal.java X
1 public class Principal {
2     public static void main(String[] args) {
3         Carro carro1 = new Carro("Fusca");
4         Carro carro2 = carro1;
5         carro2.exibirModelo();
6     }
7 }
```

Utilizando a classe **Principal** novamente:

Qual o resultado esperado?

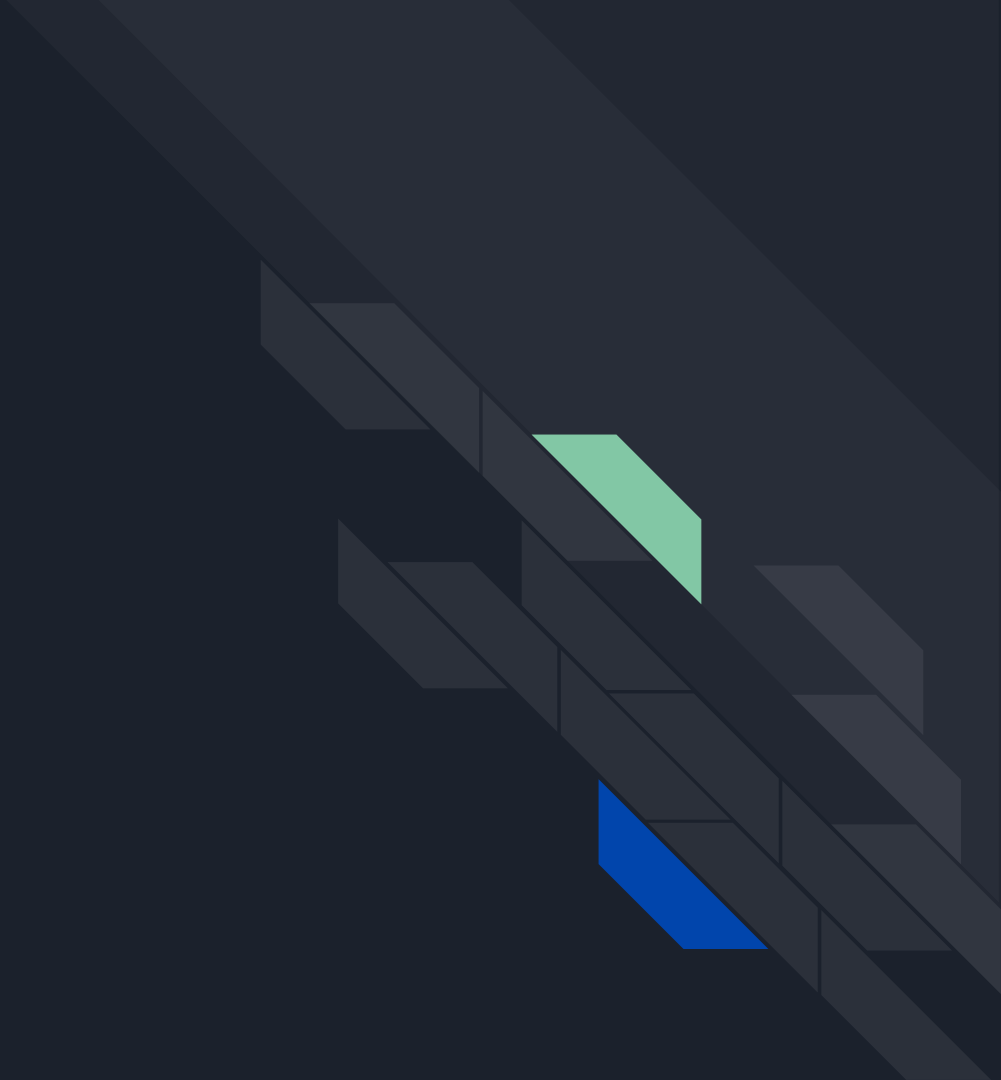
```
O modelo do carro é: Fusca
A cor do carro: Vermelho
```



Exercícios

- 1) Criar uma classe chamada Pessoa com 2 construtores, um que receba o nome e a idade da pessoa e outro recebendo apenas a idade. Solicite ao usuário qual dos construtores ele gostaria de utilizar na instânciação da classe. Além de receber parâmetros, os construtores imprimem na tela o conteúdo dos parâmetros recebidos.
- 2) Criar uma classe chamada Aluno com 3 construtores, sendo que o primeiro recebe o nome e a matrícula do aluno, o segundo recebe apenas a data de nascimento do aluno e o terceiro construtor recebe o nome do aluno, a data de nascimento e o ano em que o aluno ingressou na faculdade. Crie uma classe principal, com 3 objetos, cada um instanciando a classe com um construtor diferente.

Herança





Herança

Herança é uma forma de reutilização de *software*.

Permite além de reutilização, manutenção simples e eficiente: alterações em cascata.

Em Java a herança é observada através da palavra *extends*.

Java **não** permite herança múltipla: um artifício é utilizar uma classe, que herda de outra classe, que herda de outra classe e assim por diante.

Exemplo:

classe b *extends* classe a

classe c *extends* classe b

A classe **Carro** herda os atributos e os métodos da classe **Veículo**

```
Heranca.java x Veiculo.java x Carro.java x Bicicleta.java x
1 public class Veiculo { 2 usages 2 inheritors
2     public int numeroRodas; 4 usages
3     public String tipoPropulsao; 4 usages
4 }
```

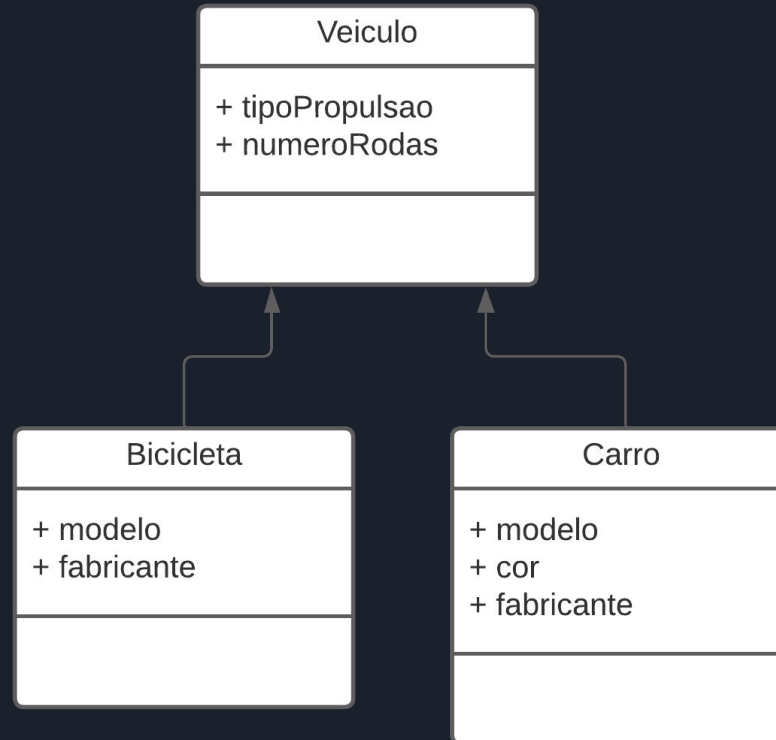
```
Heranca.java x Veiculo.java x Carro.java x Bicicleta.java x
1 public class Carro extends Veiculo{ 2 usages
2     public String modelo; 2 usages
3     public String cor; 1 usage
4     public String fabricante; 2 usages
5 }
```

```
Heranca.java x Veiculo.java x Carro.java x Bicicleta.java x
1 public class Heranca {
2     public static void main(String[] args) {
3
4         Carro c = new Carro();
5         c.modelo = "Fusca";
6         c.fabricante = "VW";
7         c.cor = "Azul";
8         c.numeroRodas = 4;
9         c.tipoPropulsao = "Térmica";
10        System.out.println(c.getClass().getName() + ": Modelo: "+c.modelo+" Fabricante:"+c.fabricante+" N. Rodas:"+c.numeroRodas+" Tipo propulsão:"+c.tipoPropulsao);
11
12        Bicicleta b = new Bicicleta();
13        b.fabricante = "Caloi";
14        b.modelo = "Cross";
15        b.numeroRodas = 2;
16        b.tipoPropulsao = "Humana";
17        System.out.println(b.fabricante+" "+b.modelo+" "+b.numeroRodas+" "+b.tipoPropulsao);
18    }
19 }
```

Qual o resultado da execução da classe **Principal**?

Carro: Modelo: Fusca Fabricante:VW N. Rodas:4 Tipo propulsão:Térmica

Herança em UML



Herança da herança da herança em UML

```
2 public class Desenho {  
3     protected static String nomeAutorDesenho = "Herysson";  
4  
5 }
```

```
2 public class Desenho2D extends Desenho {  
3     protected static int largura;  
4     protected static int altura;  
5  
6     public static void exibeDimensoes(){  
7         System.out.println("Dimensões: " + largura + " x " + altura);  
8     }  
9 }
```

```
1 public class Quadrado extends Desenho2D{  
2     public static void main(String args[]){  
3         largura = 10;  
4         altura = 10;  
5         exibeDimensoes();  
6         System.out.println("O nome do autor é: " + nomeAutorDesenho);  
7     }  
8  
9 }
```

Qual o resultado da execução da classe Quadrado (main)?

Herança da herança da herança em UML

```
2 public class Desenho {  
3     protected static String nomeAutorDesenho = "Herysson";  
4  
5 }
```

```
2 public class Desenho2D extends Desenho {  
3     protected static int largura;  
4     protected static int altura;  
5  
6     public static void exibeDimensoes(){  
7         System.out.println("Dimensões: " + largura + " x " + altura);  
8     }  
9 }
```

```
1 public class Quadrado extends Desenho2D{  
2     public static void main(String args[]){  
3         largura = 10;  
4         altura = 10;  
5         exibeDimensoes();  
6         System.out.println("O nome do autor é: " + nomeAutorDesenho);  
7     }  
8  
9 }
```

Qual o resultado da execução da classe Quadrado (main)?

Dimensões: 10x10
O nome do autor é: Herysson



Herança

Exercício

- 1) Crie uma classe Principal para solicitar e exibir na tela as informações de nome, e-mail e telefone do usuário. Esses atributos são herdados da classe Usuario.
- 2) Crie uma classe chamada Pessoa que herda da classe SerHumano os atributos nome e idade e o método falar. A classe SerHumano também possui herança e herda o atributo tipo e o método andar da classe Animal. Desse modo, crie uma classe Principal para exibir na tela o conteúdo de todos os atributos e realizar a chamada de todos os métodos envolvidos no processo. O método falar retorna a string “Nem todos falam” e o método andar imprime na tela a string “Todos andam, mas o modo é variado”. Solicite ao usuário para informar o nome, a idade e o tipo.



Herança

Exercício

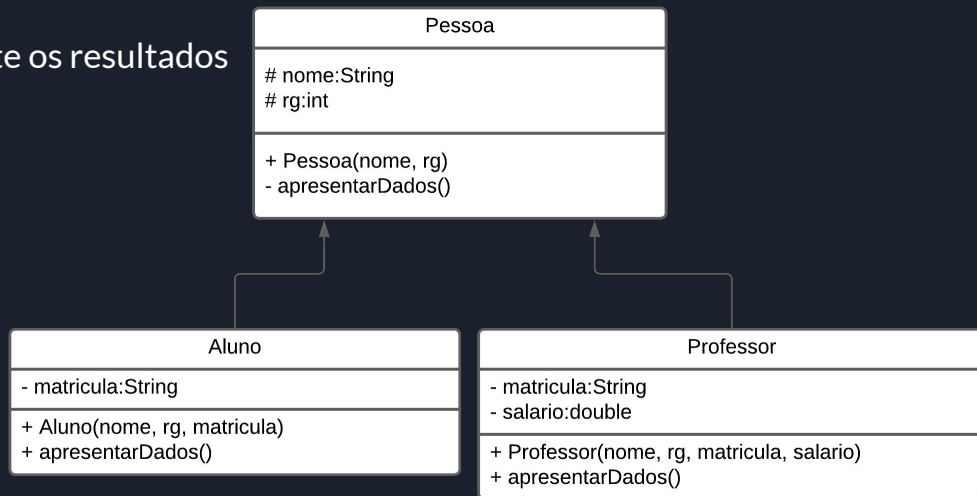
3) Uma empresa está cadastrando seus funcionários em sua base de dados. Sabendo que *Funcionario* é uma *Pessoa*, e que pessoas possuem (nome, idade e telefone), e que seus funcionários são divididos por (setor, cargo e função). Crie uma classe executável, e cadastre 3 funcionários, preferencialmente solicitando as informações via console ao usuário. Após esse cadastro, exiba esses funcionários na tela. A classe *Funcionario* deve herdar as características da classe *Pessoa*.

Herança

Exercício

4) Escreva um programa orientado a objetos baseado no diagrama de classes da UML apresentado abaixo:

Crie um objeto de cada classe, e apresente os resultados de cada uma delas na tela.



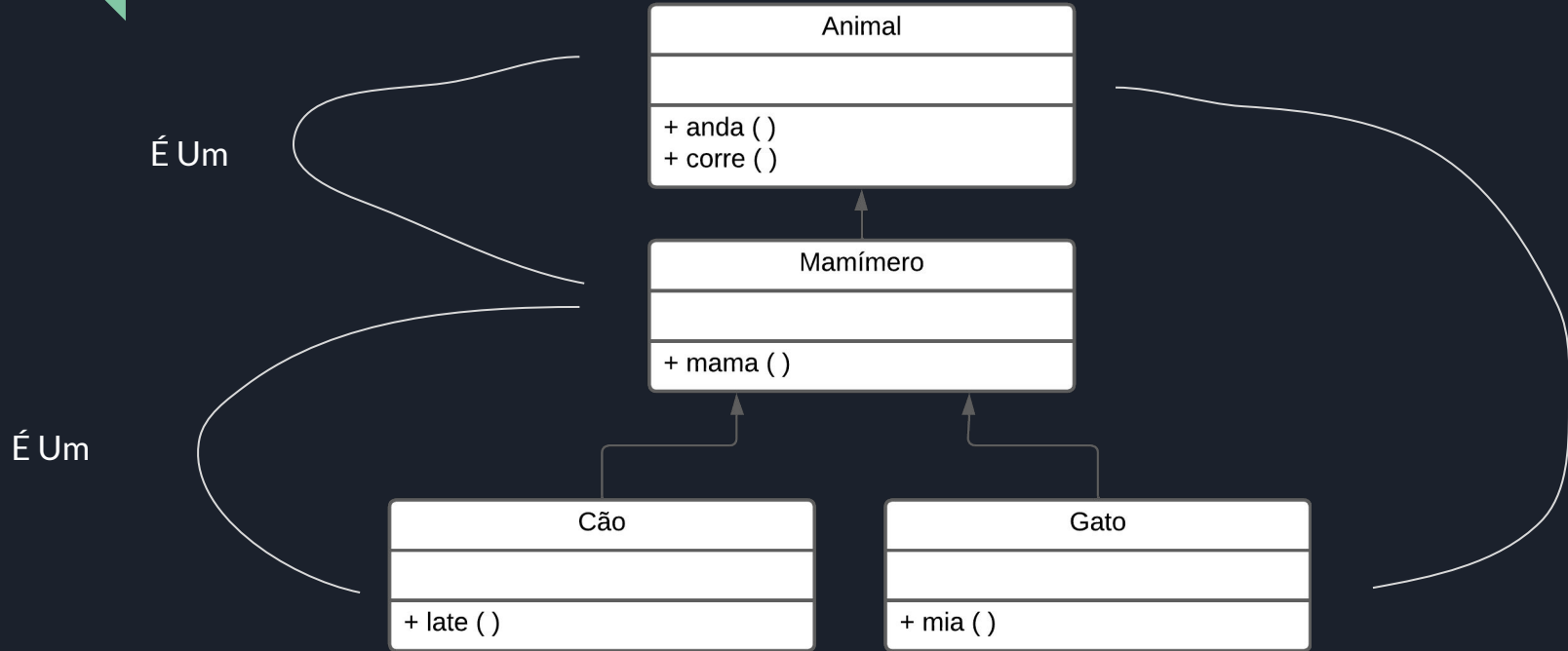
Dica: para passar os parâmetros da sub-classe para a super-classe utilize o método `super(parâmetros)`.

Herança - Especialização

Especialização

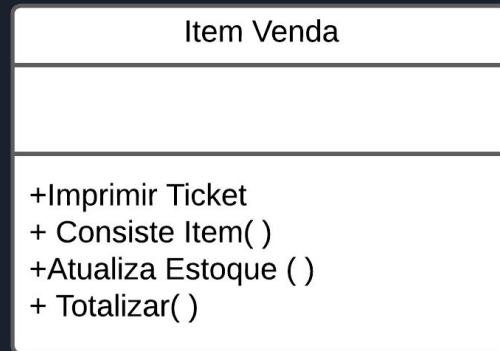
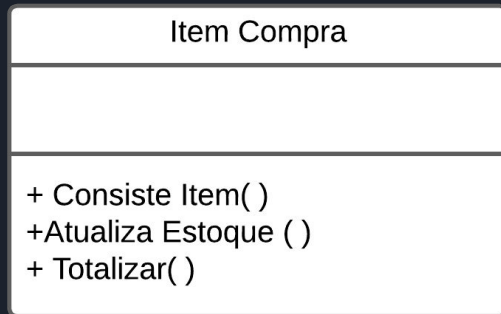


Herança - Aplicação “É um(a)”





Herança - Generalização



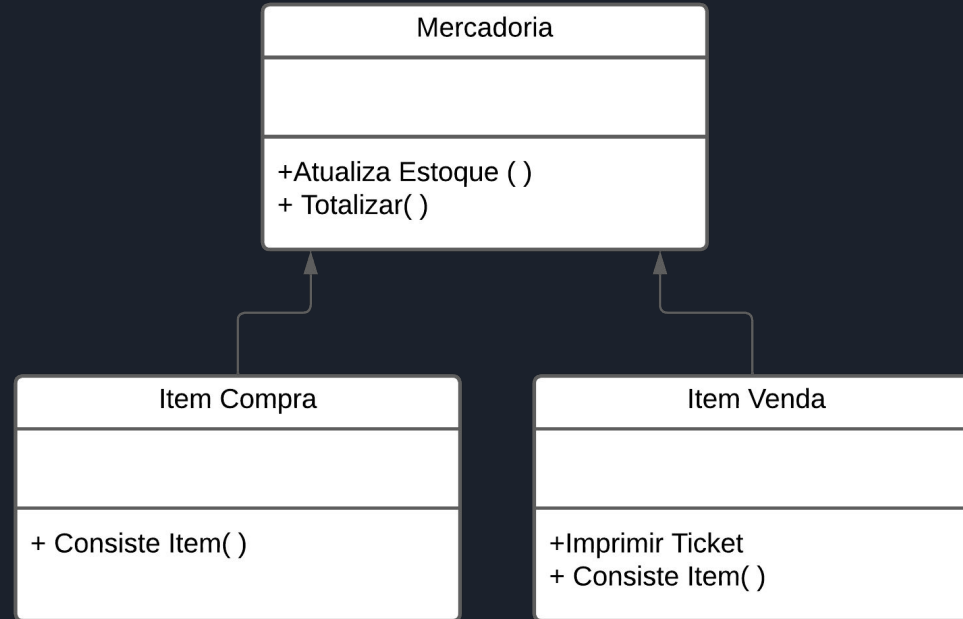
1 - Responsabilidades (métodos) comuns

2 - Descrições semelhantes

Herança - Generalização

3 - Verificar Relacionamento “é um(a)”

Generalizar



Classes Abstratas





Classes Abstratas

Uma classe abstrata é uma classe que pode ter métodos implementados e métodos não implementados.

Uma classe abstrata não pode ser instanciada, ela deve ser implementada. Os métodos serão implementados pelas subclasses que herdaram os métodos da superclasse.

A declaração de uma classe abstrata é iniciada pela palavra reservada `abstract`

Pode ser útil quando é necessária a existência futura de uma classe/método mas não é necessário implementar esse método no mesmo instante (implementação futura).

Classes Abstratas

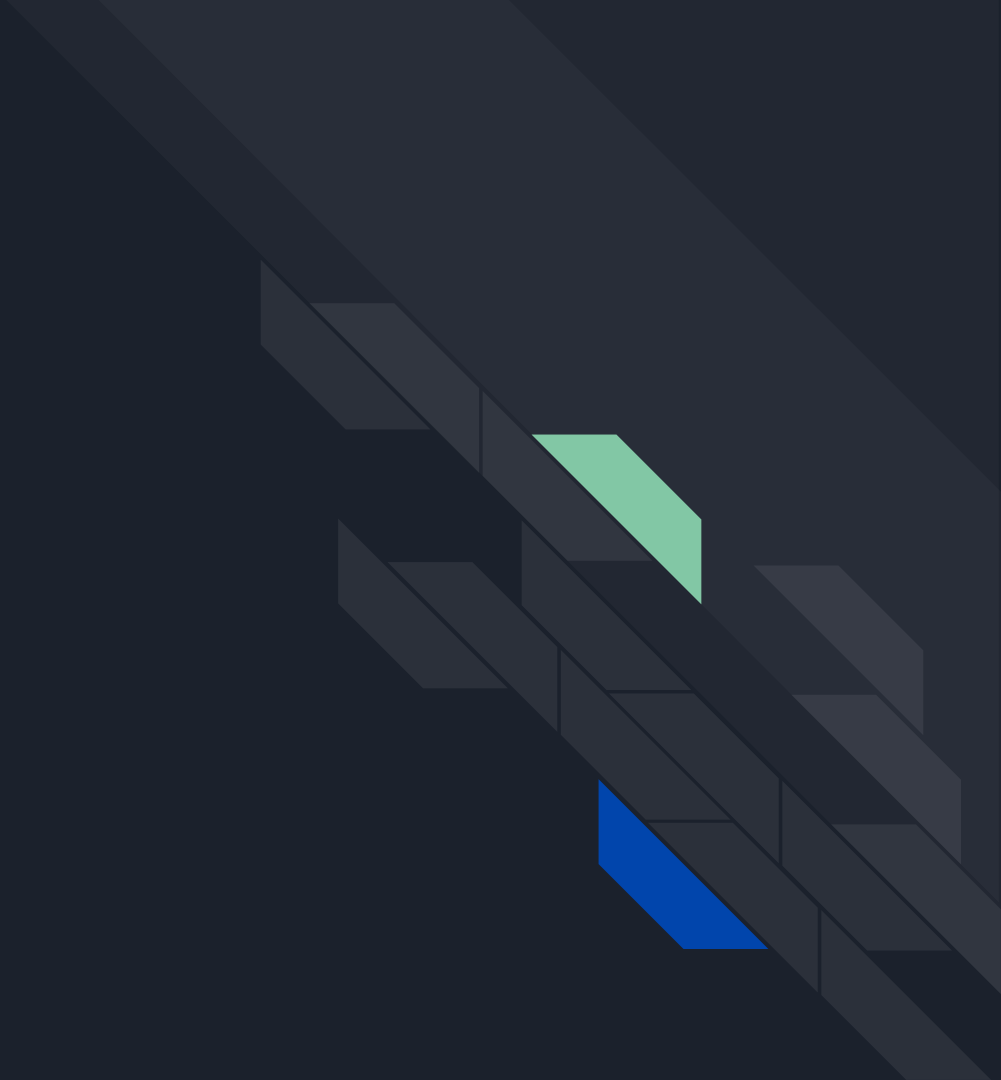
```
11 public abstract class Pessoa {  
12     abstract void atribuiNome();  
13     abstract void exibeNome();  
14 }
```

Herança

```
4 public class Principal extends Pessoa{  
5     String nome;  
6  
7     public static void main (String[] args){  
8         Principal p = new Principal();  
9         p.atribuiNome();  
10        p.exibeNome();  
11    }  
12  
13    @Override  
14    void atribuiNome() {  
15        nome = "Herysson";  
16    }  
17    @Override  
18    void exibeNome() {  
19        System.out.println(nome);  
20    }  
21 }
```

Implementação

Interface





Interface (Contrato)

Como o Java não permite herança múltipla é possível realizar a herança em cascata (a extends b, b extends c, logo a também extends c)

Ou...

Criar interface: que é um conjunto de declarações de métodos (nome, tipo de retorno, tipos dos argumentos) sem implementação. Fica a critério do programador que deseja implementar a interface em questão providenciar uma implementação desses métodos na classe que ele está desenvolvendo.

Em outras palavras, podemos dizer que uma Interface é utilizada para abstrair um comportamento, comum a várias classes.

A interface também é um tipo de contrato, definindo os métodos que uma classe deve implementar.



Interface

Dessa forma, além de estender alguma superclasse, a classe em desenvolvimento pode implementar várias interfaces.

A palavra reservada *implements* é utilizada para indicar que uma classe implementa uma determinada interface.

A classe herdada:

Fornece uma *interface*

A classe herdeira:

Implementa (*implements*) a(s) superclasse(s)

Interface

Exemplo

`Guest` herda as características de `User`

```
Guest.java User.java X
1 package quest5;
2
3 public interface User {
4     public boolean isAuthorized(String word);
5
6 }
7
```

```
Guest.java X
1 package quest5;
2
3 public class Guest implements User {
4
5     public static void main(String[] args) {
6         Guest guest = new Guest();
7
8         if(guest.isAuthorized("We Are Atos")) {
9             System.out.println("Welcome");
10        }
11        else {
12            System.out.println("Try Again");
13        }
14    }
15
16    @Override
17    public boolean isAuthorized(String word) {
18        boolean status = false;
19        if (word.equals("WeAreAtos")) {
20            status = true;
21        }
22        return status;
23    }
24
25 }
```



Interface

Exercício

1) Crie uma classe abstrata chamada Mensagem com os métodos abstratos `exibeMensagemA()` e `exibeMensagemB()` que deverão ser implementados em uma classe Principal. A mensagem A deverá ser sempre a seguinte: “Cuide bem dos animais”. Já, a mensagem B deverá ser uma mensagem de para preservação do planeta que o usuário, em tempo de execução do sistema, irá digitar. Após, imprima na tela as duas mensagens.



Interface

Exercício

2) Crie uma classe **Compromisso** que possui os atributos nome (do compromisso), data e hora. A classe **Compromisso** é herdada pelas classes **Profissional** e **Pessoal** que possuem uma interface de acesso.

Além disso, a classe **Profissional** possui o método `exibeCompromissoProfissional()` e a classe **Pessoal** possui o método `exibeCompromissoPessoal()`.

Ambos os métodos deverão ser implementados em uma classe **Principal**, que também fará a leitura dos valores dos atributos e as suas respectivas impressões na tela.

Polimorfismo





Polimorfismo

Polimorfismo permite “programar no geral” ao invés de “programar no específico”.

Exemplos de polimorfismo em Java são:

- Sobrescrita de métodos (**Override**)

Métodos, parâmetros e retornos iguais, implementação de código diferentes

- Sobrecarga de métodos (**Overload**)

Métodos de mesmo nome, porém com parâmetros e implementações diferenciada

Essas características são definidas em tempo de execução.

Polimorfismo

Sobrescrita de Métodos

```
Calculadora.java X
1 package Calculadora;
2
3 public class Calculadora {
4     public void calcular(double valor1, double valor2) {
5
6     }
7     public void exibeResultado() {
8
9     }
10 }
```

A classe **Calculadora** possui dois métodos.

calcular()

exibeResultado();

Polimorfismo

Sobrescrita de Métodos

As classes **Soma** e **Multiplicacao** herdam os métodos da classe **Calculadora**, e os implementam de modo diferente, cada um com suas características: sobrescrita de métodos

```
Soma.java X
1 package Calculadora;
2
3 public class Soma extends Calculadora {
4     double resultado;
5
6     public void calcular (double valor1, double valor2) {
7         resultado = valor1+valor2;
8     }
9     public void exibeResultado() {
10        System.out.println("Soma: "+resultado);
11    }
12
13 }
```

```
Multiplicacao.java X
1 package Calculadora;
2
3 public class Multiplicacao extends Calculadora{
4     double resultado;
5
6     public void calcular (double valor1, double valor2) {
7         resultado = valor1*valor2;
8     }
9     public void exibeResultado() {
10        System.out.println("Multiplicação: "+resultado);
11    }
12
13 }
```

Polimorfismo

Sobrescrita de Métodos

A classe **Principal** por sua vez, utiliza as classes para somar e multiplicar os dois valores:

```
Principal.java X
1 package Calculadora;
2
3 public class Principal {
4
5     public static void main(String[] args) {
6         Soma soma = new Soma();
7         soma.calcular(2, 3);
8         soma.exibeResultado();
9         Multiplicacao multiplicacao = new Multiplicacao();
10        multiplicacao.calcular(2, 3);
11        multiplicacao.exibeResultado();
12    }
13
14 }
```

Qual o resultado esperado?

```
Soma: 5.0
Multiplicação: 6.0
```



Polimorfismo

Sobrecarga de Métodos

A classes **Soma** herda os métodos e atributos da classe **Calculadora**, porém o método calcular pode receber parâmetros diferentes e realizar uma operação diferente (em tempo de compilação): *sobrecarga de método*

```
Soma.java X
1 package Calculadora;
2
3 public class Soma extends Calculadora {
4     double resultado;
5
6     public void calcular (double valor1, double valor2) {
7         resultado = valor1+valor2;
8     }
9     public double calcular(double valor1) {
10         resultado=valor1+100;
11         return resultado;
12     }
13     public void exibeResultado() {
14         System.out.println("Soma: "+resultado);
15     }
16
17 }
```



Polimorfismo

Sobrecarga de Métodos

Logo, em tempo de compilação, é escolhido qual o método irá receber a sobrecarga:

```
Principal.java X
1 package Calculadora;
2
3 public class Principal {
4
5     public static void main(String[] args) {
6         Soma soma = new Soma();
7         System.out.println("Retorno da soma, Sobrecarga: "+soma.calcular(2));
8         soma.exibeResultado();
9         soma.calcular(2,3);
10        soma.exibeResultado();
11    }
12
13 }
```

Qual o resultado esperado:

```
Retorno da soma, Sobrecarga: 102.0
Soma: 102.0
Soma: 5.0
```



Polimorfismo

Sobrecarga de Métodos

1) Crie uma classe para cada uma das operações (+, -, /, *). Cada operação deverá herdar a classe Operacao abaixo, e fazer a sobrescrita do método calcular. Uma classe principal deverá solicitar ao usuário os números a serem calculados e exibir o resultado das operações. Utilizar construtores.

```
public class Operacao {  
    protected double a;  
    protected double b;  
    protected double resultado;  
    public double calcular(){  
        return resultado;  
    }  
}
```

2) Crie uma classe para armazenar os atributos nome e email do usuário em um arquivo texto, em que o nome do arquivo é o mesmo nome do usuário que foi digitado. Essa classe deverá permitir a sobrecarga de métodos, em que o usuário poderá gravar o nome e o email no arquivo, ou somente o nome. Crie uma classe Principal para realizar a leitura dos atributos. Se ambos os atributos forem preenchidos a sobrecarga deverá ser uma. Se apenas o nome do usuário for preenchido, então a sobrecarga deverá ser outra. Além disso, ofereça ao usuário a opção de ler o conteúdo do arquivo de acordo com um nome por ele informado.

Exceções

Exceções são classes especiais que representam erros e que podem ser lançadas para o método que fez a invocação delas.

```
public class MinhaException extends Exception{  
    ...  
}
```

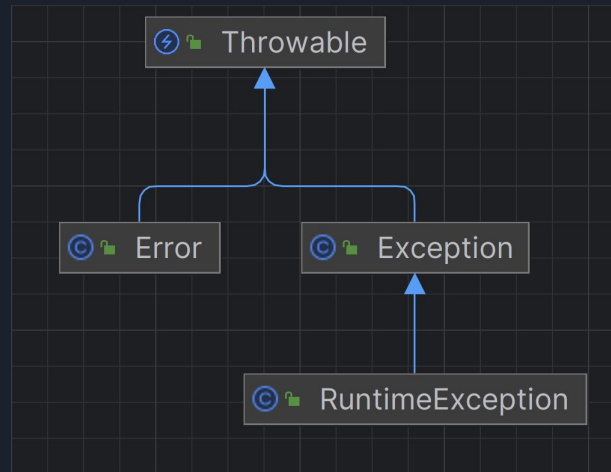
Declaração das exceções que esse método poderá lançar.

```
public void meuMetodo(int param) throws MinhaException{  
    if(error){  
        throw new MinhaException("erro");  
    }  
}
```

Cria uma nova instância da exceção para então chama-lá.

Exceções

Hierarquia de classe de exceções





Exceções

Error:

Erros ou condições anormais que dificilmente conseguiremos tratar:

- `CoderMalfunction`
- `FactoryConfiguration`
- `VirtualMachineError`

RuntimeException:

Não precisam obrigatoriamente ser tratados. Não somos obrigados a tratar. Esses erros são de codificação e não deveriam acontecer.

- `ArithmeticException`
- `ClassCastException`
- `NullPointerException`



Exceções

Exception:

Exceções que podem potencialmente acontecer durante a execução do programa, e devem obrigatoriamente ser tratadas.

- SQLException
- IOException
- PrintException



Exceções

Exercício

1) Crie uma classe para cada uma das operações (+, -, /, *). Cada operação deverá herdar a classe Operacao abaixo, e fazer a sobrescrita do método calcular. Uma classe principal deverá solicitar ao usuário os números a serem calculados e exibir o resultado das operações. Utilizar construtores.

```
public class Operacao {  
    protected double a;  
    protected double b;  
    protected double resultado;  
    public double calcular(){  
        return resultado;  
    }  
}
```

Baseado no exercício resolvido anteriormente, crie uma exceção que seja disparada, caso ambos os operandos sejam 0 (zero). A exceção deverá lançar uma mensagem (“Ambos os valores a serem calculados devem ser diferentes de 0”).

Faça o tratamento dessa exceção, pedindo para que o usuário insira novamente os dados para cálculo.

on