ELE2 – LPG3

Eletiva 2 – Linguagem de Programação 3 (Java)
Prof. Davi Reis

Aula 02 – Orientação a Objetos – Revisão Básica

Prof. Davi dos Reis

https://linktr.ee/prof.davi.reis

Conceitos

- Classe
  - Objeto
  - Instância
- Construtor e Destrutor
- Encapsulamento (membros)
  - Atributos
  - Métodos
- Herança
- Polimorfismo

Classe

- Abstrai um conjunto de objetos/entidades com características e comportamentos similares entre si.
- Define (ou descreve) a estrutura e o comportamento que os objetos compartilharão.
- Para isso, dispõe de:
  - Atributos: características ou estados possíveis destes objetos.
  - Métodos: ações que o objeto pode realizar.

Estrutura de um Classe

- Atributos
- Métodos
- Construtores
- Destrutor

- Sintaxe:

```
class NomeDaClasse{
    modificador tipo atributo
    modificador tipo método (tipo parâmetro1, tipo parâmetro2..., tipo parâmetroN){
        //Código...
    }
}
```

Atributos

- São declarações das classes que representam características de instância que possam assumir estados.
- No exemplo, **dia**, **mes** e **ano** são atributos do tipo inteiro da classe **Data**, enquanto **hora**, **minuto** e **segundo** são atributos do tipo inteiro da classe **Horario**. Na classe Principal, nenhum atributo foi declarado.

```
1  package aularevisao;
2
3  class Data{
4      int dia, mes, ano;
5  }
6  class Horario{
7      int hora, minuto, segundo;
8  }
9  public class Principal {
10     public static void main(String[] args) {
11         Data data = new Data();
12         data.ano = 2025; data.mes = 12; data.dia = 31;
13
14         Horario horario = new Horario();
15         horario.hora = 23; horario.minuto = 59; horario.segundo = 59;
16
17         System.out.println(String.format("Data: %d/%d/%d",
18                 data.dia, data.mes, data.ano));
19
20         System.out.println(String.format("Hora: %d:%d:%d",
21                 horario.hora, horario.minuto, horario.segundo));
22     }
23 }
```

```
Saída - AulaRevisao (run)
run:
Data: 31/12/2025
Hora: 23:59:59
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


Métodos

- São declarações das classes que representam ações que a mesma pode executar em função dos valores de seus atributos.
- No exemplo, um método chamado **retornarInfo()** em cada classe retorna uma String representando a data e a hora em função dos valores dos atributos de suas respectivas instâncias.

```
1  package aularevisao;
2
3  class Data{
4      int dia, mes, ano;
5      String retornarInfo(){
6          return String.format("Data: %d/%d/%d", dia, mes, ano);
7      }
8  }
9  class Horario{
10     int hora, minuto, segundo;
11     String retornarInfo(){
12         return String.format("Hora: %d:%d:%d", hora, minuto, segundo);
13     }
14 }
15 public class Principal {
16     public static void main(String[] args) {
17         Data data = new Data();
18         data.ano = 2025; data.mes = 12; data.dia = 31;
19
20         Horario horario = new Horario();
21         horario.hora = 23; horario.minuto = 59; horario.segundo = 59;
22
23         System.out.println(data.retornarInfo());
24         System.out.println(horario.retornarInfo());
25     }
26 }
```

```
Saída - AulaRevisao (run)
run:
Data: 31/12/2025
Hora: 23:59:59
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Objetos e Instâncias

- Objeto é uma entidade que pode ser física, conceitual ou de software. Ou seja, uma representação genérica.

- Instância é usada com o sentido de exemplo. É a concretização da classe. Ou seja, são os objetos de fato criados e ocupando espaço na memória.

Construtor

- Método especial. É disparado automaticamente na instanciação da classe.
- Utilizado basicamente para inicializar os estados dos atributos.
- Deve possuir sempre o mesmo nome da classe.
- Pode ser do tipo **default** ou **não-default.**
- Retorna sempre a instância da classe.
- Normalmente **public**, mas ter outros modificadores de acesso.
- No exemplo a seguir, foram criados dois construtores. Cada um deles é chamado de acordo com a criação da instância, pela lista de argumentos.

Construtor
Exemplo:

```
1   package aularevisao;
2
3   class Data{
4       int dia, mes, ano;
5       public Data() {
6           dia = mes = ano = 0;
7       }
8       public Data(int dia, int mes, int ano) {
9           this.dia = dia; this.mes = mes; this.ano = ano;
10      }
11      String retornarInfo(){
12          return String.format("Data: %d/%d/%d", dia, mes, ano);
13      }
14  }
15
16  public class Principal {
17      public static void main(String[] args) {
18          Data data = new Data();
19          data.ano = 2025; data.mes = 12; data.dia = 31;
20          System.out.println(data.retornarInfo());
21
22          Data data2 = new Data(31,12,2029);
23          System.out.println(data2.retornarInfo());
24      }
25  }
26
```

Saída - AulaRevisao (run)

```
Data: 31/12/2025
Data: 31/12/2029
```


Destrutor

- O método Destrutor não pode ser sobrecarregado porque não aceita parâmetros e é executado quando o objeto é excluído.

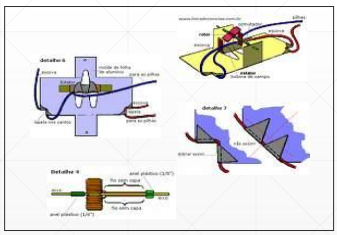
```
obj.finalize();
```

- Não é recomendado para a liberação de recursos, pois o momento em que ele é chamado pelo *garbage collector* pode ser imprevisível.
- Para liberação de recursos, é aconselhável utilizar como “try-finally” ou implementar interfaces como “AutoCloseable”, se necessário.

Encapsulamento

- Mecanismo que permite separar detalhes de funcionamento – características (atributos) e funções (métodos) – de sua interface.
- Para isso, limita o acesso aos membros da classe
  - Exemplo: Para utilizarmos um liquidificador, não precisamos saber detalhes de seu funcionamento. A única interface que conhecemos são seus botões.

Classe **Liquidificador** →



Instância de um objeto de liquidificador: muito mais fácil de usar!

Encapsulamento

- Utiliza-se de modificadores de acesso, tais como:
  - **Public**: Acessível a partir de qualquer outra classe, independente do package.
  - **Protected**: Acessível a partir de qualquer classe derivada, no mesmo package.
  - **Private**: Acessível apenas dentro da própria classe.
  - **Static**: Membro compartilhado com todas as instâncias da classe.
  - **Final**: Constante da classe.
  - **Friendly**: Acessível a partir de qualquer outra classe do mesmo package (também chamado de “default” ou “package-private”).

Encapsulamento - Atributo

- Os atributos são características (variáveis) definidas em uma classe e podem ser classificados em dois tipos:
  - Atributos de Instância: são propriedades que pertencem a cada objeto individualmente e armazenam o estado específico de cada instância.
  - Atributos de Classe: são propriedades associadas à classe em si, cujo valor/estado é compartilhado por todas as instâncias dessa classe.

```
class CartaDeBaralho{
        public char Símbolo;
        public image Naipe;
        public static ListaDeSímbolos[] = {0,1,2,3,4,5,6,7,8,9,J,Q,K,A};
        public static ListaDeNaipes[] = { ♥ , ♠ , ♦ , ♣ };
}
```

Atributos de Instância

Atributos de Classe

Encapsulamento - Método

- Os métodos são rotinas executadas por um objeto ao receber uma mensagem, determinando seu comportamento. Eles são análogos às funções ou procedimentos da programação estruturada, mas estão associados a um contexto específico de um objeto ou classe.
  - É uma rotina que é executada por um objeto ao receber uma mensagem. De certo modo, é a descrição da forma como o objeto realiza uma tarefa.
  - Os métodos determinam o comportamento dos objetos de uma classe e são análogos às funções ou procedimentos da programação estruturada.
  - Assim como os atributos, os métodos podem ser encapsulados por meio de modificadores de acesso, controlando como eles podem ser chamados dentro e fora da classe.

Encapsulamento de atributos
Exemplo:

```
1   package aularevisao;
2   class Data{
3       private int dia, mes, ano;
4
5       public int getDia() { return dia; }
6       public void setDia(int dia) {
7           if (dia > 0 && dia < 32) this.dia = dia;
8       }
9       public int getMes() { return mes; }
10      public void setMes(int mes) {
11          if (mes > 0 && mes < 13) this.mes = mes;
12      }
13      public int getAno() { return ano; }
14      public void setAno(int ano) { this.ano = ano; }
15
16      String retornarInfo(){
17          return String.format("Data: %d/%d/%d", dia, mes, ano);
18      }
19  }
20
21  public class Principal {
22      public static void main(String[] args) {
23          Data data = new Data();
24          data.setDia(31); data.setMes(12);  data.setAno(2025);
25          System.out.println(data.retornarInfo());
26      }
27  }
```

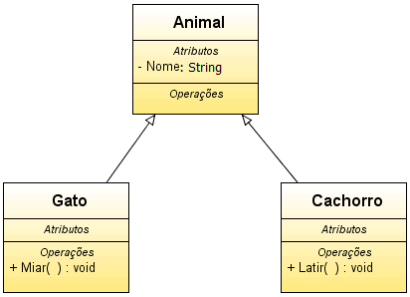
```
Saída - AulaRevisao (run)
run:
Data: 31/12/2025
```


Herança - Definição

- Conhecida também como Generalização.
- Definição de uma classe a partir de outra, herdando os seus membros (não-privados).
- A classe herdeira é chamada subclasse.
- A classe herdada é chamada superclasse.
- Pode ser utilizado em cascata, criando hierarquias com várias gerações de classes.
- Permite que as classes compartilhem atributos e métodos baseados em um **relacionamento**.

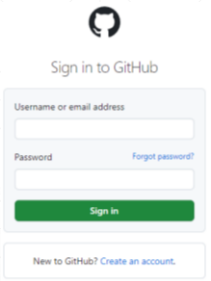
Subclasse e superclasse

- Superclasse é a classe herdada. Ou seja, a classe Pai.
- Subclasse é a classe herdeira. Ou seja, a classe Filha.
- No exemplo, Animal é a superclasse.
- No exemplo, Cachorro e Gato são subclasses.



Antes de prosseguir com código...

- Recomenda-se fortemente o uso e a familiarização com ferramentas de controle de versão, colaboração, integração etc., tais como o Git (e GitHub), pois é algo praticamente obrigatório no mercado de trabalho.
- Por isso, mas não somente, é importante aprender a utilizá-la e criar um portifólio com todos os seus códigos e projetos que estiver estudando ou desenvolvendo.
- Para auxiliar, segue um curso simples e gratuito: https://www.udemy.com/course/git-e-github-para-iniciantes/



Herança
Exemplo:

```
abstract class Pessoa {

    protected String nome;
    private String email;

    public String getNome() { return nome; }

    public String getEmail() { return email; }

    public Pessoa(String nome, String email) {
        this.nome = nome;
        this.email = email;
    }
}
```

```
public class Aluno extends Pessoa {

    private int ra;

    public Aluno(String nome, String email, int RA) {
        super(nome, email);
        this.ra = RA;
    }
}
```

```
public class Professor extends Pessoa {

    public Professor(String nome, String email) {
        super(nome, email);
    }
}
```


Polimorfismo

- Grego: “muitas formas”
  - *poli* = muitas, *morphos* = formas.
- Permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam.
- Um mesmo método pode apresentar várias formas, de acordo com seu contexto.

Polimorfismo

- Permite que a semântica de uma interface seja efetivamente separada da implementação que a representa.
- Basicamente, *override* e *overload.*
- Benefícios:
  - Clareza e manutenção do código;
  - Divisão da complexidade;
  - Aplicações flexíveis.

Sobreposição (Override)

- Utilizamos quando queremos reescrever um método na classe Filha que já existe e está implementado na classe Pai.
- Podemos chamar o método pai (se for desejado) através da palavra reservada **super.**
- Também chamado de sobrescrita ou polimorfismo vertical.

Override
Exemplo:

```java
package aularevisao;

import java.awt.Graphics;

abstract class Figura{
    abstract void desenhar(Graphics g, int x, int y);
    abstract float calcularArea();
}
```

```java
package aularevisao;

import java.awt.Graphics;

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    void desenhar(Graphics g, int x, int y) {
        g.drawRect(x, y, x + lado, y + lado);
    }
    @Override
    float calcularArea() {
        return lado * lado;
    }
}
```


Sobrecarga (Overload)

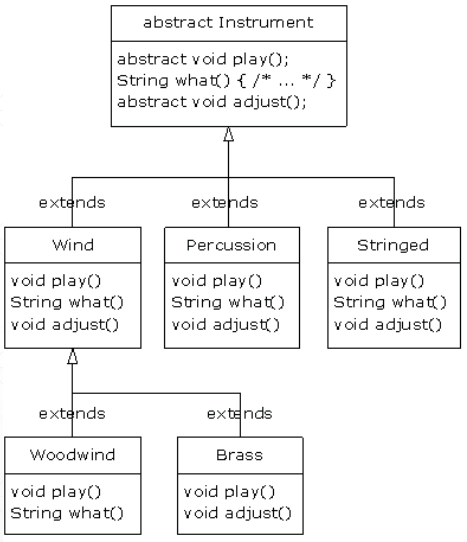
- Quando há 2 ou mais métodos com o mesmo nome, mas com assinatura (e comportamento) diferente, declarados na mesma classe.
- Também conhecido como overload ou polimorfismo horizontal.

```java
// Exemplo 1: Métodos sobrecarregados para calcular a área de uma figura
public double calcularArea(double lado);
public double calcularArea(double base, double altura);

// Exemplo 2: Métodos sobrecarregados para exibir uma mensagem
public void exibirMensagem(String mensagem);
public void exibirMensagem(String mensagem, int vezes);
```

Classes abstratas

- São classes que abstraem um conjunto de características que podem ou não serem implementadas.
- Não podem ser instanciadas.
- “A implementamos quando sabemos o que fazer mas não como fazer”.



Classes abstratas
Exemplo:

```java
package aulavisao;

import java.awt.Graphics;

abstract class Figura{
    abstract void desenhar(Graphics g, int x, int y);
    abstract float calcularArea();
}
```

```java
package aulavisao;

import java.awt.Graphics;

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    void desenhar(Graphics g, int x, int y) {
        g.drawRect(x, y, x + lado, y + lado);
    }
    @Override
    float calcularArea() {
        return lado * lado;
    }
}
```

```java
class Retangulo extends Figura{
    private int base, altura;
    public void setBase(int base) { this.base = base; }
    public void setAltura(int altura) { this.altura = altura; }

    @Override
    void desenhar(Graphics g, int x, int y) {
        g.drawRect(x, y, x + base, y + altura);
    }
    @Override
    float calcularArea() {
        return base * altura;
    }
}
```

Interface

- Contrato que define métodos (modelando comportamentos) que uma classe deve implementar.
- Permite que classes diferentes implementem a mesma interface de maneira específica.
- Estabelece ponto de comunicação entre objetos (por exemplo, em pacotes diferentes).
- Reduz acoplamento, aumenta flexibilidade e promove reutilização de código.
- Uma classe pode implementar várias interfaces. Com isso, supre (em partes) a falta de herança múltipla (visto que o Java trabalha apenas com herança única).
- Na prática, uma interface assemelha-se a uma classe abstrata que contém apenas métodos abstratos (e, eventualmente, constantes).

Interface
Exemplo:

```
package aularevisao;

interface ICalculo{
    abstract float calcularArea();
    abstract float calcularPerimetro();
}
```

```
package aularevisao;

import java.awt.Graphics;

interface IGrafico{
    abstract void desenhar(Graphics g, int x, int y);
}
```

```
abstract class Figura implements ICalculo, IGrafico{
}

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    public void desenhar(Graphics g, int x, int y) {
        g.drawRect(x, y, x + lado, y + lado);
    }
    @Override
    public float calcularArea() {
        return lado * lado;
    }
    @Override
    public float calcularPerimetro() {
        return (lado + lado) * 2;
    }
}
```

Upcasting e Downcasting

- Upcasting é quando transformamos (através de uma atribuição) um tipo de baixo para cima, na hierarquia;
- Downcasting é quando transformamos (também através de uma atribuição) um tipo de cima para baixo na hierarquia.
- Partindo das classes abaixo como exemplo...

```
package aularevisao;

abstract class Figura{
    abstract float calcularArea();
}
```

```
package aularevisao;

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    float calcularArea() {
        return lado * lado;
    }
}
```


Upcasting
Exemplo

```
package aularevisao;

abstract class Figura{
    abstract float calcularArea();
}
```

```
package aularevisao;

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    float calcularArea() {
        return lado * lado;
    }
}
```

```
public class Principal {
    public static void main(String[] args) {
        Quadrado objQuadradoUp = new Quadrado();
        objQuadradoUp.setLado(5);
        // ...(outras linhas de código por aqui)...
        Figura objFiguraUp = objQuadradoUp;
        System.out.println(
            String.format("[Upcasting] Área da figura: %.2f cm².",
                    objFiguraUp.calcularArea()));
    }
}
```

```
- AulaRevisao (run)
run:
[Upcasting] Área da figura: 25,00 cm².
```

Downcasting
Exemplo

```java
package aularevisao;

abstract class Figura{
    abstract float calcularArea();
}
```

```java
package aularevisao;

class Quadrado extends Figura{
    private int lado;
    public void setLado(int lado) { this.lado = lado; }

    @Override
    float calcularArea() {
        return lado * lado;
    }
}
```

```java
public class Principal {
    public static void main(String[] args) {
        Figura objFiguraDown = new Quadrado();
        // ...(outras linhas de código por aqui)...
        Quadrado objQuadradoDown = (Quadrado)objFiguraDown;
        objQuadradoDown.setLado(7);
        System.out.println(
            String.format("[Downcasting] Área do quadrado: %.2f cm².",
                    objQuadradoDown.calcularArea()));
    }
}
```

```
- AulaRevisao (run)
run:
[Downcasting] Área do quadrado: 49,00 cm².
```


Associação

- Vínculo que permite que objetos de uma ou mais classes se relacionem;
- É possível que um objeto convoque comportamentos e estados de outros objetos;
- As associações podem ser:
  - Unárias: Entre objetos de uma mesma classe;
  - Múltiplas: Entre mais de dois objetos.

Associação

- Cada associação possui características de:
  - Cardinalidade ou multiplicidade - determina quantos objetos no sistema são possíveis em cada vértice da associação;
  - Navegação - se é possível para cada objeto acessar outro objeto da mesma associação.

Associação - Exemplos

- Basicamente podemos ter:
  - **Agregação:** quando duas ou mais classes formam outra classe. Neste caso, a ausência de uma delas não compromete a existência da classe resultante da associação.

```
package RegraNegocio;

public class Aula {
    private Aluno _aluno;
    private Materia _materia;
}
```

  - **Composição:** também ocorre quando duas ou mais classes compõem outra classe. Neste caso, porém, a ausência de uma das classes formadoras compromete a existência da classe resultante da associação, impedindo sua utilização.

```
package RegraNegocio;

public class Carro {
    private Chassi _chassi;
    private Motor _motor;
}
```

Associação
Exemplo prático 1

```
package business;

public class Data {
    private int dia, mes, ano;

    public Data(int dia, int mes, int ano) {
        this.dia = dia;
        this.mes = mes;
        this.ano = ano;
    }

    public String getData() {
        return String.format("%02d/%02d/%02d", dia, mes, ano);
    }
}
```

```
package business;

public class Pessoa {

    private String nome;
    private Data nascimento;

    public String getNome() { return nome; }
    public Data getNascimento() { return nascimento; }

    public Pessoa(String nome, int dia, int mes, int ano) {
        this.nome = nome;
        this.nascimento = new Data(dia, mes, ano);
    }
}
```

```
package view;

import business.Pessoa;

public class Principal {

    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa("Fatec", 31, 10, 1986);

        System.out.println("Nome: " + pessoa1.getNome());
        System.out.println("Nasc.: " + pessoa1.getNascimento().getData());
    }
}
```

```
Nome: Fatec
Nasc.: 31/10/1986
```


Associação
Exemplo prático 2

```
public class Temperatura {
    private float grausCelsius;

    public float getGrausCelsius() { return grausCelsius; }
    public float getGrausFahrenheit() {
        return grausCelsius * 1.8f + 32;
    }
    public void setGrausCelsius(float grausCelsius) {
        this.grausCelsius = grausCelsius;
    }
}
```

```
public class CamaraTermica {
    private Temperatura temperaturaMinima, temperaturaMaxima;

    public Temperatura getTemperaturaMinima() { return temperaturaMinima; }

    public Temperatura getTemperaturaMaxima() { return temperaturaMaxima; }

    public CamaraTermica(float tempMinima, float tempMaxima) {
        // instancia atributos privados e atribui-lhes
        // valores iniciais (recebidos via construtor)
        temperaturaMinima = new Temperatura();
        temperaturaMinima.setGrausCelsius(tempMinima);

        temperaturaMaxima = new Temperatura();
        temperaturaMaxima.setGrausCelsius(tempMaxima);
    }
}
```

```
import negocio.CamaraTermica;

/**...4 linhas */
public class Principal {

    public static void main(String[] args) {
        // instancia objeto CamaraTermica, que contém, em seus
        // atributos, a classe Temperatura
        CamaraTermica objCamaraFria = new CamaraTermica(-5, -1);
        // exibe os valores mínimos e máximos de temperatura, em Celsius e em Fahrenheit
        System.out.println(String.format(
                "[Câmara fria] temp. mínima (°C): %.2f; temp. máxima (°C): %.2f. ",
                objCamaraFria.getTemperaturaMinima().getGrausCelsius(),
                objCamaraFria.getTemperaturaMaxima().getGrausCelsius()));
        System.out.println(String.format(
                "[Câmara fria] temp. mínima (°F): %.2f; temp. máxima (°F): %.2f. ",
                objCamaraFria.getTemperaturaMinima().getGrausFahrenheit(),
                objCamaraFria.getTemperaturaMaxima().getGrausFahrenheit()));

    }
}
```

```
Saída
run:
[Câmara fria] temp. mínima (°C): -5,00; temp. máxima (°C): -1,00.
[Câmara fria] temp. mínima (°F): 23,00; temp. máxima (°F): 30,20.
```

Obrigado!