Programação Orientada a Objetos

Aula 02 – Herança e polimorfismo
INF31098/INF31030

Prof. Dr. Jonathan Ramos
jonathan@unir.br

Departamento Acadêmico de Ciências de Computação – DACC

Núcleo de Tecnologia – NT

26/09/2022

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Sumário

1 Resolução exercício aula anterior

2 Herança

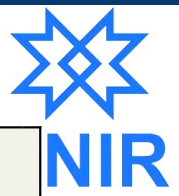
3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

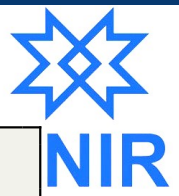
6 Github – Salvando os códigos (Opcional)

Resolução exercício aula anterior



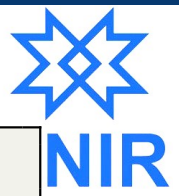

```java
private double salarioMensal;
private double salarioAnual;

public void setSalarioMensal(double sMensal) {
  this.salarioMensal = sMensal;
  this.calcularSalarioAnual();
}

public double getSalarioMensal() {
  return this.salarioMensal;
}

public double getSalarioAnual() {
  return this.salarioMensal;
}

// Somente a classe pessoa pode alterar o salario anual.
private void setSalarioAnual(double sAnual) {
  this.salarioMensal = sAnual;
}

private void calcularSalarioAnual() {
  double valorCalculado = salarioMensal*12;
  this.setSalarioAnual(valorCalculado);
}
```

Resolução exercício aula anterior




```java
public class Main {
  public static void main(String []args) {
    Pessoa p = new Pessoa();
    p.setNome("Jonathan");
    p.setIdade(18);
    p.setSalarioMensal(500.00);

    System.out.println(
        "Nome: "             + p.getNome() +
        " Idade: "           + p.getIdade() +
        " Salario Mensal: " + p.getSalarioMensal() +
        " Salario Anual: "   + p.getSalarioAnual());
  }
}
```

Resolução exercício aula anterior




```java
public class Main {
  public static void main(String []args) {
    Pessoa p = new Pessoa();
    p.setNome("Jonathan");
    p.setIdade(18);
    p.setSalarioMensal(500.00);

    System.out.println(
        "Nome: "             + p.getNome() +
        " Idade: "           + p.getIdade() +
        " Salario Mensal: " + p.getSalarioMensal() +
        " Salario Anual: "   + p.getSalarioAnual());
  }
}
```

Definindo uma função padrão pra retomar uma string com os dados escolhidos da classe (Pessoa.java):

```java
@Override
public String toString() {
  return "Nome: " + this.getNome() +
      " Idade: " + this.getIdade() +
      " Salario Mensal: " + this.getSalarioMensal() +
      " Salario Anual: " + this.getSalarioAnual();
}
```

JavaDoc

UNIR

Para dar informações sobre as funções, os parâmetros e muito mais.

- Muito útil para **relembrar depois de um tempo sem mexer no código**...
- Muito útil para que **outras pessoas entenda seu código**...
- Os códigos em HTML[1] é opcional, **provavelmente irão aprender em outro momento**...

```
1  /**
2   * <p><b>Irá definir o salário anual baseado no salário mensal</b></
       p>
3   * @param sMensal é o salário da pessoa por mês
4   * @return outro param que não temos....
5   * @see internet link here or link to other function
6   */
7  public void setSalarioMensal(double sMensal) {
8    this.salarioMensal = sMensal;
9    this.calcularSalarioAnual();
10 }
```

JavaDoc

UNIR

Para dar informações sobre as funções, os parâmetros e muito mais.

- Muito útil para **relembrar depois de um tempo sem mexer no código**...
- Muito útil para que **outras pessoas entenda seu código**...
- Os códigos em HTML[1] é opcional, **provavelmente irão aprender em outro momento**...

```
1  /**
2   * <p><b>Irá definir o salário anual baseado no salário mensal</b></
       p>
3   * @param sMensal é o salário da pessoa por mês
4   * @return outro param que não temos....
5   * @see internet link here or link to other function
6   */
7  public void setSalarioMensal(double sMensal) {
8    this.salarioMensal = sMensal;
9    this.calcularSalarioAnual();
10 }
```

Ir na classe principal e apertar Alt + Tab na chamada da função **setSalarioMensal(valor)** pra ver o javadoc

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Herança

UNIR

Uma das abstrações mais importantes da orientação a objetos

- Permite a **criação de novas classes** a partir de outras previamente criadas;
- Essas novas classes são chamadas de **subclasses**, ou classes derivadas;
- As classes que deram origem às subclasses, são chamadas de **superclasses**, ou **classes base**;
- **Reusabilidade de código!**

Herança

UNIR

Uma das abstrações mais importantes da orientação a objetos

- Permite a **criação de novas classes** a partir de outras previamente criadas;
- Essas novas classes são chamadas de **subclasses**, ou classes derivadas;
- As classes que deram origem às subclasses, são chamadas de **superclasses**, ou **classes base**;
- **Reusabilidade de código!**

Exemplo de uma escola

Uma escola tem:

- Professor, Aluno e Funcionário.

Herança

UNIR

Uma das abstrações mais importantes da orientação a objetos

- Permite a **criação de novas classes** a partir de outras previamente criadas;
- Essas novas classes são chamadas de **subclasses**, ou classes derivadas;
- As classes que deram origem às subclasses, são chamadas de **superclasses**, ou **classes base**;
- **Reusabilidade de código!**

Exemplo de uma escola

Uma escola tem:

- Professor, Aluno e Funcionário.
- **Todos são uma pessoa!**

Herança: SuperClasse Pessoa.java

UNIR

Definindo a classe **Pessoa.java**:

```
1  import java.util.Date;
2  public class Pessoa {
3    private String nome, CPF;
4    private Date dataNascimento;
5
6    public Pessoa(String nome, String CPF, Date data) {
7      this.setNome(nome);
8      this.setCPF(CPF);
9      this.setDataNascimento(data);
10   }
11
12   // Getters e Setters
13
14   @Override
15   public String toString() {
16     return " Nome: " + this.nome +
17           " CPF: " + this.CPF +
18           " Nascimento: " + this.dataNascimento;
19   }
20 }
```

Herança: SubClasse Aluno

Definindo a classe **Aluno.java**:

```
1  import java.util.Date;
2  public class Aluno extends Pessoa {
3    private String matricula;
4
5    // Construtor
6    public Aluno(String matricula, String nome, String CPF, Date data)
         {
7      super(nome, CPF, data);
8      this.matricula = matricula;
9    }
10   // Getters e Setters...
11
12   @Override
13   public String toString() {
14     return "Matricula: " + this.matricula + super.toString();
15   }
16 }
```

Herança: SubClasse Aluno

Definindo a classe **Aluno.java**:

```
1  import java.util.Date;
2  public class Aluno extends Pessoa {
3    private String matricula;
4
5    // Construtor
6    public Aluno(String matricula, String nome, String CPF, Date data)
         {
7      super(nome, CPF, data);
8      this.matricula = matricula;
9    }
10   // Getters e Setters...
11
12   @Override
13   public String toString() {
14     return "Matricula: " + this.matricula + super.toString();
15   }
16 }
```

Note que agora temos:

- **extends**: invoca a classe pessoa, e adiciona ela como se fosse parte da classe Aluno (estende-a, por isso **extends**);
- **super()**: seta os valores da superclasse Pessoa, por isso **super**.

Herança: SubClasses Professor e Funcionario

UNIR

Definindo a classe **Prefessor.java** e **Funcionario.java**:

```java
public class Professor extends Pessoa { // Professor.java
  private double salarioMensal;
  private String disciplina;

  public Professor([...], String nome, String CPF, Date data) {
    super(nome, CPF, data)
  }
  // Getters, Setters,  toString()...
}
public class Funcionario extends Pessoa { // Funcionario.java
    private double salarioMensal;
  private Date dataAdmissao;
  private String cargo;

  public Funcionario([...], String nome, String CPF, Date data) {
    super(nome, CPF, data);
  }
  // Getters, Setters, toString()...
}
```

Herança: SubClasses Professor e Funcionario

UNIR

Definindo a classe **Prefessor.java** e **Funcionario.java**:

```
1  public class Professor extends Pessoa { // Professor.java
2    private double salarioMensal;
3    private String disciplina;
4
5    public Professor([...], String nome, String CPF, Date data) {
6      super(nome, CPF, data)
7    }
8    // Getters, Setters,  toString()...
9  }
10 public class Funcionario extends Pessoa { // Funcionario.java
11     private double salarioMensal;
12   private Date dataAdmissao;
13   private String cargo;
14
15   public Funcionario([...], String nome, String CPF, Date data) {
16     super(nome, CPF, data);
17   }
18   // Getters, Setters, toString()...
19 }
```

Reusabilidade de código

- Não precisamos **reescrever** as variáveis da pessoa (nome, CPF, dataNascimento) para cada subclasse...

Herança: testando no main

UNIR

Vendo o resultado no main:

```
1  import java.util.Date;
2  public class Main {
3      public static void main(String[] args) {
4          String nome = "Aluno a";
5          String CPF = "123.456.789-00";
6          Date data = new Date();
7          String matricula = "1234356";
8
9          Aluno a = new Aluno(matricula, nome, CPF, data);
10
11         System.out.println("Como os atributos ficaram: ");
12         System.out.println(a.toString());
13     }
14 }
```

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Sobrecarga (*Overload*)

UNIR

Em Java, dois ou mais métodos podem ter o mesmo nome desde que:

- Diferem em **quantidade de parâmetros** (como já vimos nos **construtores**...);
- Possuem mesma quantidade de parâmetros, porém com **parâmetros de tipos diferentes** (String, int, double, List, etc);
- **Os dois acima**.

```
1 // função inicial
2 void func() { ... }
3 // mesma função com parâmetro de entrada diferente
4 void func(int a) { ... }
5 // mesma função com apenas um parâmetro mas com o tipo (double)
      diferente
6 float func(double a) { ... }
7 // mesma função com dois parâmetros
8 float func(int a, float b) { ... }
```

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Modificadores de acesso

Os modificadores de acesso alteram a visibilidade da classe/método/atributo dentro do projeto Java[a]:

[a] <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

- **protected**: declarações visíveis apenas dentro do pacote;
  - Veremos mais pra frente o que é um pacote...
- **default** (nada escrito): visível apenas dentro do pacote;
  - Porém, não visível também às subclasses (diferente do default que é)...
- **private**: declarações visíveis apenas dentro da classe;
- **public**: visível em qualquer lugar;

Modificador	Classe	Pacote	subclasse	Todos
public	✔	✔	✔	✔
protected	✔	✔	✔	✗
default	✔	✔	✗	✗
private	✔	✗	✗	✗

Tabela: Visibilidade dos modificadores.

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Um exemplo mais complexo para trabalharmos com herança, polimorfismo, encapsulamento, etc.

A descrição do cliente ao pedir o programa foi:

Na minha lojinha **LingSell**[a], gostaria de mapear como OO uma venda de um determinado produto:

[a] Venda de produtos da China.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Um exemplo mais complexo para trabalharmos com herança, polimorfismo, encapsulamento, etc.

A descrição do **cliente** ao **pedir** o programa foi:

Na minha lojinha **LingSell**[a], gostaria de mapear como OO uma venda de um determinado produto:

- O Produto tem uma descrição, um valor e quantidade de meses de garantia a partir da data da compra, etc;

[a] Venda de produtos da China.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Um exemplo mais complexo para trabalharmos com herança, polimorfismo, encapsulamento, etc.

A descrição do **cliente** ao **pedir** o programa foi:

Na minha lojinha **LingSell**[a], gostaria de mapear como OO uma venda de um determinado produto:

- O Produto tem uma descrição, um valor e quantidade de meses de garantia a partir da data da compra, etc;
- O Cliente possui nome, data de nascimento, algum documento de identificação, um endereço, telefone para contato, email, etc;

[a] Venda de produtos da China.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Um exemplo mais complexo para trabalharmos com herança, polimorfismo, encapsulamento, etc.

A descrição do **cliente** ao **pedir** o programa foi:

Na minha lojinha **LingSell**[a], gostaria de mapear como OO uma venda de um determinado produto:

- O Produto tem uma descrição, um valor e quantidade de meses de garantia a partir da data da compra, etc;
- O Cliente possui nome, data de nascimento, algum documento de identificação, um endereço, telefone para contato, email, etc;
- No Endereço precisa ter o CEP, pais, logradouro, número da casa e complemento, estado, etc;

[a] Venda de produtos da China.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Um exemplo mais complexo para trabalharmos com herança, polimorfismo, encapsulamento, etc.

A descrição do **cliente** ao **pedir** o programa foi:

Na minha lojinha **LingSell**[a], gostaria de mapear como OO uma venda de um determinado produto:

- O Produto tem uma descrição, um valor e quantidade de meses de garantia a partir da data da compra, etc;
- O Cliente possui nome, data de nascimento, algum documento de identificação, um endereço, telefone para contato, email, etc;
- No Endereço precisa ter o CEP, pais, logradouro, número da casa e complemento, estado, etc;
- A Venda em si deverá conter todos essas informações, pois no futuro outra classe irá herdar esses dados; Além disso, A venda deve conter o tipo de pagamento, data, etc;

[a] Venda de produtos da China.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,
  - Já sabemos da existência de 4 classes: Produto, Cliente, Endereço e Venda;

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,
  - Já sabemos da existência de 4 classes: Produto, Cliente, Endereço e Venda;
  - Começando pelo mais simples, sabemos que o Cliente herdará informações de Endereço, certo?

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,
  - Já sabemos da existência de 4 classes: Produto, Cliente, Endereço e Venda;
  - Começando pelo mais simples, sabemos que o Cliente herdará informações de Endereço, certo?
  - Um Produto sempre vai estar na Venda;

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,
  - Já sabemos da existência de 4 classes: Produto, Cliente, Endereço e Venda;
  - Começando pelo mais simples, sabemos que o Cliente herdará informações de Endereço, certo?
  - Um Produto sempre vai estar na Venda;
  - Toda venda tem um cliente, não é mesmo?

Exercício prático em aula no Eclipse – **Para enviar por email**

UNIR

Por simplicidade, assumiremos que uma venda sempre ocorre para um único produto e um único cliente, e que ninguém compra mais de um produto por vez.

Primeiro passo... Faremos no quadro (resposta no último slide)

- Mapear o que será Classe, subclasse, Superclasse etc,
  - Já sabemos da existência de 4 classes: Produto, Cliente, Endereço e Venda;
  - Começando pelo mais simples, sabemos que o Cliente herdará informações de Endereço, certo?
  - Um Produto sempre vai estar na Venda;
  - Toda venda tem um cliente, não é mesmo?
  - Portanto, Venda herda informações do Produto e do Cliente;
- Mapear as variáveis e onde eles poderão ser vistas;
- Mapear os possíveis métodos:

Resposta do mapeamento..

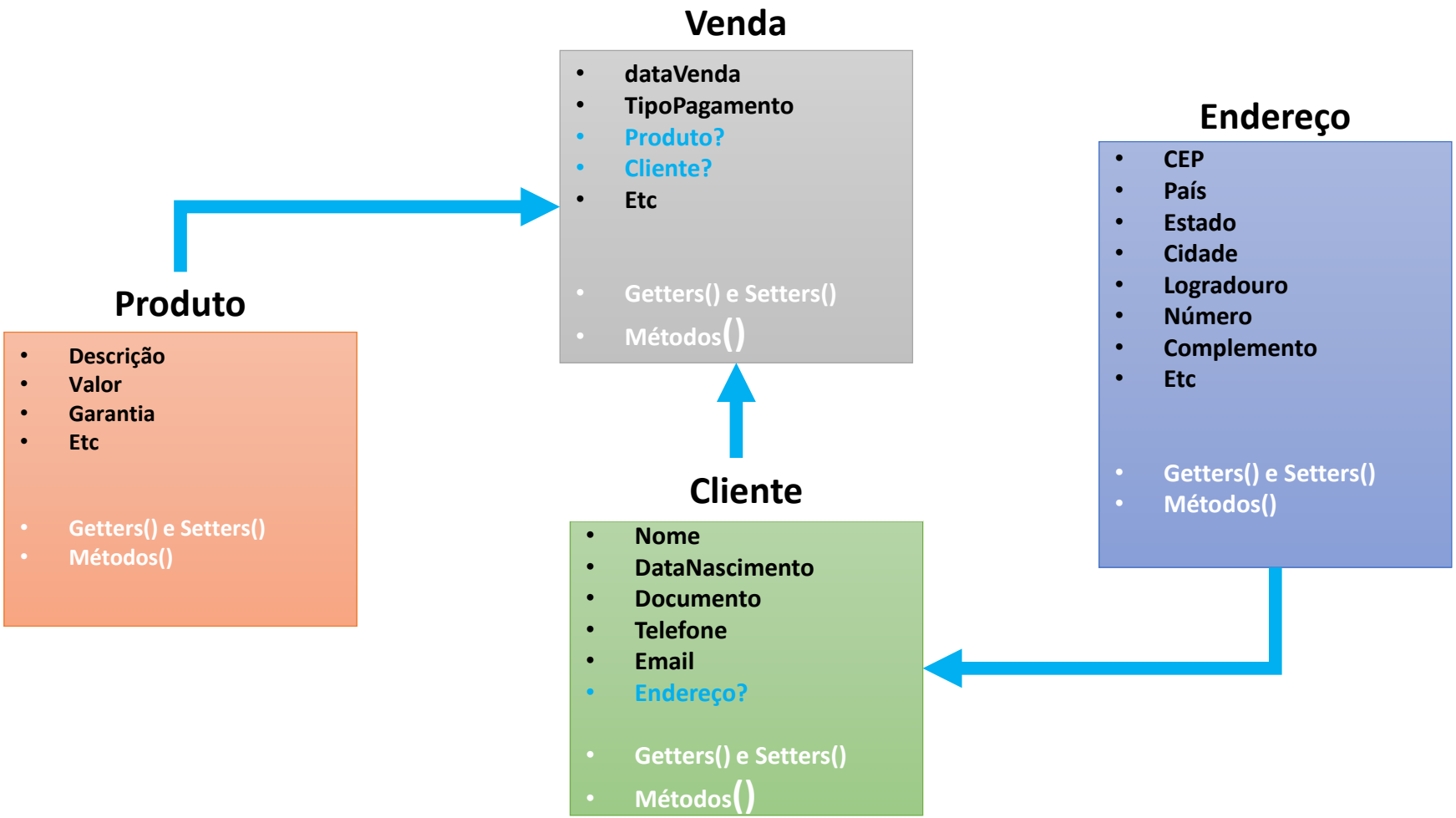


Figura: Mapeamento da OO da Venda da empresa **LingSell**.

Para reflexão...

UNIR

Como faríamos se a empresa **LingSell** pedisse que **uma única venda** pudesse ser feitas para **clientes múltiplos**, assim como **vários produtos**?

Sumário

1 Resolução exercício aula anterior

2 Herança

3 Sobrecarga (*Overload*)

4 Modificadores de acesso

5 Exercício prático em aula no Eclipse

6 Github – Salvando os códigos (Opcional)

Github – Salvando os códigos (Opcional)

UNIR

1. **Instalar** o git <https://git-scm.com/downloads>;
2. **Criar conta** no github se não tiver ainda;
3. Criar repositório no Github com o nome <AulasPOO202-1>
4. Navegue até o diretório do projeto e adicione-o como git;

```
1 echo "# AulasPOO2022-1" >> README.md
2 git init
3 git add README.md
4 git commit -m "first commit"
5 git branch -M main
6 git remote add origin https://github.com/JonathanRamos/AulasPOO2022-1.git
7 git push -u origin main
```

Classes, Objetos, Ambiente, etc

UNIR

FIM!

jonathan@unir.br