

# Programação Orientada a Objeto

## 12. Introdução à UML e Relacionamento entre Classes (Prática)

Prof. Dr. Thiago L. T. da Silveira

`tltsilveira@inf.ufrgs.br`

2º Semestre de 2020

# Objetivos

- Exercitar os conceitos de POO vistos na aula anterior!
  - Escrevendo, compilando e interpretando (executando) aplicações em Java;
  - Modelando (e/ou interpretando) sistemas baseados em OO;
  - Relações entre classes;

# Exemplo #1 Parte #0

- Todo aluno matriculado na disciplina “Trabalho de Conclusão de Curso” é orientado por um único professor. Alguns professores orientam nenhum ou vários alunos.

# Exemplo #1 Parte #1

- Todo aluno matriculado na disciplina “Trabalho de Conclusão de Curso” é orientado por um único professor. Alguns professores orientam nenhum ou vários alunos.



## Exemplo #2 Parte #0

- Um item para venda em um supermercado pode ser colocado na cesta de compras de um cliente ou não ser vendido. Um cliente pode optar por incluir na sua cesta de compras diversas unidades de um mesmo item.

## Exemplo #2 Parte #1

- Um item para venda em um supermercado pode ser colocado na cesta de compras de um cliente ou não ser vendido. Um cliente pode optar por incluir na sua cesta de compras diversas unidades de um mesmo item.

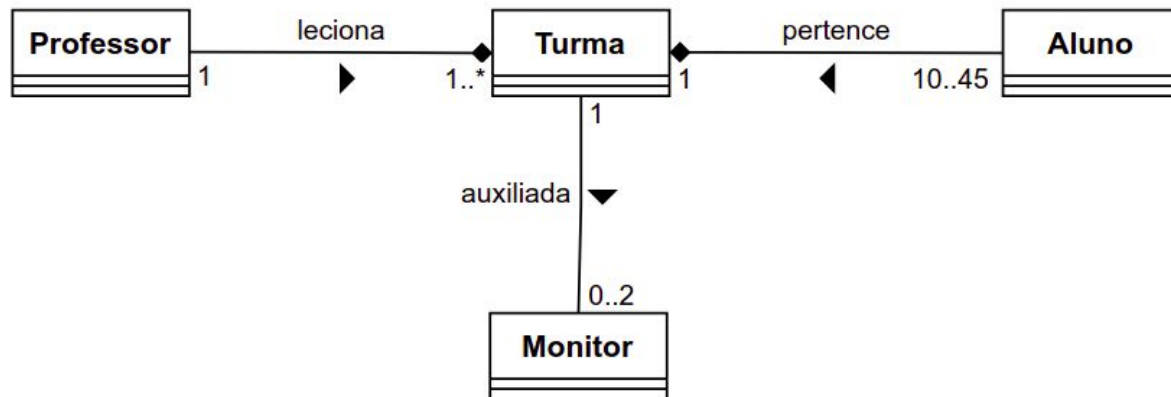


## Exemplo #3 Parte #0

- Uma turma de Programação Orientada a Objeto tem um único professor responsável e pode ter até dois monitores que o auxiliam. Um professor pode estar responsável por mais de uma turma, enquanto que os monitores assistem uma turma por semestre. As turmas têm, pelo menos, 10 alunos matriculados e não mais que 45.

## Exemplo #3 Parte #1

- Uma turma de Programação Orientada a Objeto tem um único professor responsável e pode ter até dois monitores que o auxiliam. Um professor pode estar responsável por mais de uma turma, enquanto que os monitores assistem uma turma por semestre. As turmas têm, pelo menos, 10 alunos matriculados e não mais que 45.



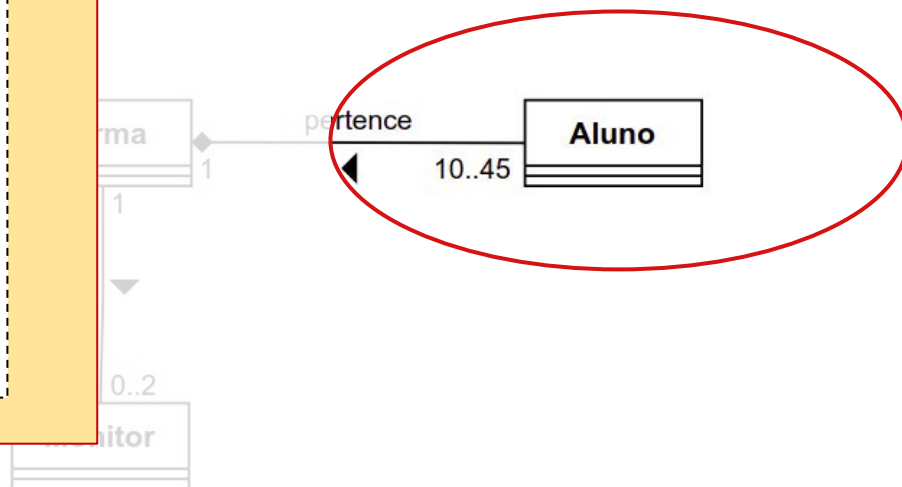


# Arrays em Java

## Arrays em Java ...

```
class Aluno {  
    private String nome;  
  
    public Aluno(String nome){  
        this.nome = nome;  
    }  
    /* ... */  
}  
/* ... */  
public class Aplicacao {  
  
    public static void main (String[] args){  
        Aluno[] alunos = new Aluno[45];  
        alunos[0] = new Aluno("Fulano");  
        alunos[1] = new Aluno("Cicrano");  
        /* ... */  
    }  
}
```

toda a Objeto tem um único professor  
tores que o auxiliam. Um professor pode  
na turma, enquanto que os monitores  
As turmas têm, pelo menos, 10 alunos



# Arrays em Java

## Arrays em Java ...

```
class Aluno {  
  
    private String nome;  
  
    public Aluno(String nome){  
        this.nome = nome;  
    }  
    public String getNome(){  
        return nome;  
    }  
    /* ... */  
}  
/* ... */  
public class Aplicacao {  
  
    public static void main (String[] args){  
        Aluno[] alunos;  
        alunos = new Aluno[45];  
        /* ... */  
    }  
}
```

```
/* ... */  
alunos[0] = new Aluno("Fulano");  
alunos[1] = new Aluno("Cicrano");  
/* ... */  
for (int i = 2; i < 45; i++){  
    alunos[i] = new Aluno("Padrão");  
}  
for (int i = 0; i < alunos.length; i++){  
    System.out.println(alunos[i].getNome());  
}  
for (Aluno aluno : alunos){  
    System.out.println(aluno.getNome());  
}  
}
```

# Arrays em Java

## Arrays em Java ...

```
class Aluno {  
    private String nome;  
  
    public Aluno(String nome){  
        this.nome = nome;  
    }  
    public String getNome(){  
        return nome;  
    }  
    /* ... */  
}  
/* ... */  
public class Aplicacao {  
  
    public static void main (String[] args){  
        Aluno[] alunos;  
        alunos = new Aluno[45];  
        /* ... */  
    }  
}
```

```
/* ... */  
alunos[0] = new Aluno("Fulano");  
alunos[1] = new Aluno("Cicrano");  
/* ... */  
for (int i = 2; i < 45; i++){  
    alunos[i] = new Aluno("Padrão");  
}  
for (int i = 0; i < alunos.length; i++){  
    System.out.println(alunos[i].getNome());  
}  
for (Aluno aluno : alunos){  
    System.out.println(aluno.getNome());  
}  
}
```

**Atributos "array de alguma coisa" tem esse formato!**

# Arrays em Java

## Arrays em Java ...

```
class Aluno {  
    private String nome;  
  
    public Aluno(String nome){  
        this.nome = nome;  
    }  
    public String getNome(){  
        return nome;  
    }  
    /* ... */  
}  
/* ... */  
public class Aplicacao {  
  
    public static void main (String[] args){  
        Aluno[] alunos;  
        alunos = new Aluno[45];  
        /* ... */  
    }  
}
```

```
/* ... */  
alunos[0] = new Aluno("Fulano");  
alunos[1] = new Aluno("Cicrano");  
/* ... */  
for (int i = 2; i < 45; i++){  
    alunos[i] = new Aluno("Padrão");  
}  
for (int i = 0; i < alunos.length; i++){  
    System.out.println(alunos[i].getNome());  
}  
for (Aluno aluno : alunos){  
    System.out.println(aluno.getNome());  
}  
}
```

**A instanciação de atributos array  
geralmente acontece no construtor!**


# Arrays em Java

## Arrays em Java ...

```
class Aluno {  
    private String nome;
```

***E a manipulação deles em outros métodos!***

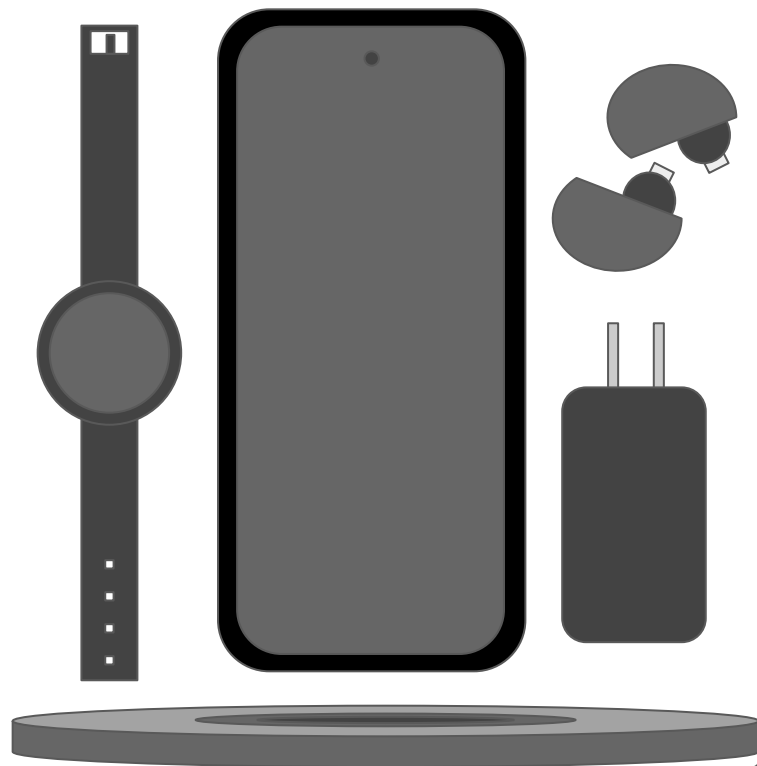
```
}  
public String getNome(){  
    return nome;  
}  
/* ... */  
}  
/* ... */  
public class Aplicacao {  
  
    public static void main (String[] args){  
        Aluno[] alunos;  
        alunos = new Aluno[45];  
        /* ... */  
    }  
}
```



```
/* ... */  
alunos[0] = new Aluno("Fulano");  
alunos[1] = new Aluno("Cicrano");  
/* ... */  
for (int i = 2; i < 45; i++){  
    alunos[i] = new Aluno("Padrão");  
}  
for (int i = 0; i < alunos.length; i++){  
    System.out.println(alunos[i].getNome());  
}  
for (Aluno aluno : alunos){  
    System.out.println(aluno.getNome());  
}  
}
```

# Exercício #1 Parte #0

- Todo mundo quer o novo *jPhone*! Todo o novo usuário que quer estar “bem conectado”, “precisa” adquirir todo o ecossistema *Pear* (*smartphone*, fones sem fio, carregadores com e sem fio, *smartwatch*). Vamos modelar parte de um sistema de vendas *online*, que registra estoque de produtos; vendas (carrinho com produtos com venda concluída); e pagamentos (com parcelamento) [...]



# Exercício #1 Parte #1

- Elaborar o diagrama de classes (na perspectiva conceitual) contendo as classes **Produto**, **Pagamento** e **Venda** e seus relacionamentos (associação e/ou agregação e/ou composição);
  - **Associação**: objetos podem usar outros objetos, principalmente como atributos
  - **Agregação**: objetos podem conter outros objetos como um elemento “parte”. Os elementos “parte” podem existir mesmo sem a existência do elemento “todo”.
  - **Composição**: um objeto “todo” podem ser formados por outros objetos “partes”. Os elementos “parte” só existem enquanto o elemento “todo” existir.

# Exercício #1 Parte #2

- Um **Produto** tem uma **descricao** textual, um **valor** unitário (ambos fixos) e estoque **quantidadeEstoque** (variável conforme objetos da classe **Venda**);
- O construtor da classe **Produto** recebe (e atribui) valores para todos seus atributos;
- Pense quais *getters* e *setters* precisam ser implementados e implemente-os;
- **Produto** não tem outros métodos (exceto os *getters* e/ou *setters* **necessários**)  
[...]



# Exercício #1 Parte #2

- Um **Produto** tem uma **descricao** textual, um **valor** unitário (ambos fixos) e estoque **quantidadeEstoque** (variável conforme objetos da classe **Venda**);
- O construtor da classe **Produto** recebe (e atribui) valores para todos seus atributos;
- Pense quais *getters* e *setters* precisam ser implementados e implemente-os;
- **Produto** não tem outros métodos (exceto os *getters* e/ou *setters* **necessários**)  
[...]



***Mais claro depois de ver o funcionamento do sistema como um todo!***

# Exercício #1 Parte #2

- Um **Pagamento** tem uma **data** textual e um **valor** (ambos fixos);
- O construtor da classe **Pagamento** recebe (e atribui) valores para todos seus atributos;
- Pense quais *getters* e *setters* precisam ser implementados e implemente-os;
- **Pagamento** não tem outros métodos (exceto os *getters* e/ou *setters* **necessários**) [...]

# Exercício #1 Parte #2

- Um **Pagamento** tem uma **data** textual e um **valor** (ambos fixos);
- O construtor da classe **Pagamento** recebe (e atribui) valores para todos seus atributos;
- Pense quais *getters* e *setters* precisam ser implementados e implemente-os;
- **Pagamento** não tem outros métodos (exceto os *getters* e/ou *setters* **necessários**) [...]



***Mais claro depois de ver o funcionamento do sistema como um todo!***

# Exercício #1 Parte #2

- **Produtos** são pagos com **Pagamentos** e vendidos em operações de **Venda**;
- Uma **Venda** tem um **numeroNFe** (número de nota fiscal - inteiro, sequencial começando em 1, atribuído automaticamente);
- Uma **Venda** também tem um **cpf** (**String**), uma **data** (**String**), e as **quantidades** “de” e referências “para” os **produtos** (**Produto**) **vendidos** (no mínimo 1 e máximo **maxVendas** = 15);
- Deve-se ter um **contador** para gerenciar o número de **produtos** da **Venda**;
- Uma **Venda** também tem referências às **parcelas** (**Pagamento**) associadas [...]

# Exercício #1 Parte #2

- Produtos são pagos com Pagamentos e vendidos em operações de Venda;
- Uma Venda tem um numeroNFe (número de nota fiscal - inteiro, sequencial começando em 1, atribuído automaticamente);
- *Os atributos quantidades, produtos e parcelas devem ser arrays!*
- Deve-se ter um contador para gerenciar o número de produtos da Venda;
- Uma Venda também tem referências às parcelas (Pagamento) associadas [...]

# Exercício #1 Parte #2

- O construtor de **Venda** recebe (e manipula) **cpf**, **data** e **numeroParcelas** (número de parcelas - de 1 a 10) do **Pagamento** (e atribui o **numeroNFe** de nota fiscal);
  - Lembre-se também de instanciar os arrays **quantidades**, **produtos** e **parcelas**;
- Uma **Venda** tem quatro métodos: **registraProduto**, **registraPagamento**, **calculaTotal** e **imprimeRecibo** [...]

# Exercício #1 Parte #2

- O método **registraProduto** recebe um **produto** (**Produto**) e uma **quantidade** associada à **Venda** corrente e não retorna nada;
- O método **registraProduto** deve verificar se é possível fazer a operação (dado o limite de itens de uma **Venda** (**maxVendas**) e a compatibilidade de **quantidade** com o estoque (**quantidadeEstoque**) daquele **Produto**);
- Se for possível, atualizar **produtos**, **quantidades**, estoque do produto (**quantidadeEstoque**) e **contador**;
- Se não for, indicar ao usuário porque não foi [...]

## Exercício #1 Parte #2

- O método **registraPagamento** recebe uma **data** (**String**) e um **valor** associados a um **Pagamento** e o **identificador** da parcela (1 para primeira parcela, 2 para segunda parcela, ...) e não retorna nada;
- O método **registraPagamento** deve verificar se o **identificador** corresponde a uma parcela prevista;
- Se sim, instanciar um **Pagamento** e associar esse pagamento a **parcelas**;
- Se não, indicar erro [...]



# Exercício #1 Parte #2

- O método **calculaTotal** não recebe nada e retorna o total associado à **Venda** corrente, isto é, a soma dos **valores** de todos os **produtos** vendidos;
  - Lembrar de considerar as **quantidades** de unidades vendidas!

# Exercício #1 Parte #2

- O método **imprimeRecibo** não recebe nenhum atributo e não retorna nada;
- Para  $M$  produtos e  $N$  pagamentos, o método **imprimeRecibo** deve imprimir:

*"Dados da Venda:*

*CPF: <cpf-da-venda>*

*Data: <data-da-venda>*

*Total de produtos: <quantidade-de-**produtos**-comprados>*

*<1> <descrição-produto-1> <quantidade-produto-1> <valor-**total**-produto-1>*

*...*

*<M> <descrição-produto-M> <quantidade-produto-M> <valor-**total**-produto-M>*

*Valor total: R\$: <valor-total-da-venda>*

*Parcelas:*

*<1> <data-pagamento-1> <valor-pagamento-1>*

*...*

*<N> <data-pagamento-N> <valor-pagamento-N>"*

*Confira o exemplo de  
execução!*

# Exercício #1 Parte #3

- Criar uma outra classe em Java (**Aplicacao**) que representa o programa principal e contém o método *main*;
- Instancie cinco **Produtos** de acordo com a tabela abaixo:

Objeto	Atributo <b>descricao</b>	Atributo <b>quantidadeEstoque</b>	Atributo <b>valor</b>
produto1	"iPhone"	500	6999.00
produto2	"airBuds"	1500	500.00
produto3	"Pear watch"	800	2999.00
produto4	"Carregador sem fio"	250	250.00
produto5	"Carregador com fio"	10	69.90

# Exercício #1 Parte #3

- Instancie uma **Venda** com **cpf**, **data** e **numeroParcelas** valendo "123.456.789-10", "04/03/2021" e 4, respectivamente;
- Registre os cinco **Produtos** criados nessa **Venda** com **quantidade** valendo 2, 2, 1, 1 e 2, respectivamente;
- Obtenha o **valorTotal** da **Venda** com **calculaTotal** e divida esse **valorTotal** em **numeroParcelas**;
- Registre os **numeroParcelas** "**Pagamentos**" com **datas** valendo "04/03/2021", "04/04/2021", "04/05/2021" e "04/06/2021", respectivamente e **valor** igual a **valorTotal/numeroParcelas**;
- Chame **imprimeRecibo**.

# Exercício #1 - Exemplo de Execução

Dados da Venda:

CPF: 123.456.789-10

Data: 04/03/2021

Total de produtos na compra: 5

1. airBuds 2 R\$ 1000.0
2. iPhone 2 R\$ 13998.0
3. Pear watch 1 R\$ 2999.0
4. Carregador sem fio 1 R\$ 250.0
5. Carregador com fio 2 R\$ 139.8

Valor total: R\$ 18386.8

Parcelas:

1. 04/03/2021 R\$ 4596.7
2. 04/04/2021 R\$ 4596.7
3. 04/05/2021 R\$ 4596.7
4. 04/06/2021 R\$ 4596.7

# Exercício #1 Parte #4

- Teste o caso em que **quantidade** em **registraProduto** é maior que **quantidadeEstoque** do **Produto** associado;
- Teste o caso em que **contador** é maior que **maxVendas** em **registraProduto**;
- Teste o caso em que **identificador** de **registraPagamento** é maior que **numeroParcelas** (informado no construtor de **Venda**);
- Não é necessário apresentar nada com relação a isso!

# Atividades

- Entrega de um arquivo **.zip** (contendo a estrutura de diretórios da aplicação e os quatro arquivos .java e uma imagem/PDF)
  - Entregue o diagrama de classe do Exercício #1 Parte #1
  - Entregue a implementação do Exercício #1 Partes #2 a #3
  - Entrega até às 23:55h de 11/03/2021



# Programação Orientada a Objeto

## 12. Introdução à UML e Relacionamento entre Classes (Prática)

Prof. Dr. Thiago L. T. da Silveira

`tltsilveira@inf.ufrgs.br`

2º Semestre de 2020