

Linguagem de Programação II

Artur Henrique Kronbauer
artur.kronbauer@fieb.org.br



Introdução

- Vamos abordar os seguintes tópicos:
 - Modelos
 - Modelos e Dados
 - Modelos e Operações
 - Modelos e Orientação a Objetos
 - Programação Orientada a Objetos
 - Classes e Objetos
 - Classes em Java – Sintaxe Básica

Modelos

- Modelos são representações simplificadas de objetos, pessoas, itens, tarefas, processos, conceitos, ideias, etc. São usados comumente por pessoas no seu dia-a-dia, independente do uso de computadores.

Restaurante Caseiro Hipotético					
Mesa 1		Mesa 2		Mesa 3	
Kg refeição		Kg refeição		Kg refeição	
Sobremesa		Sobremesa		Sobremesa	
Refrigerante		Refrigerante		Refrigerante	
Cerveja		Cerveja		Cerveja	

Modelo e Dados

- O quadro branco é um modelo do restaurante, representando de forma simplificada as informações do restaurante necessárias para contabilizar os pedidos.
- O modelo representa certos dados ou informações. Os dados contidos no modelo são somente relevantes à abstração do mundo real feita.

Modelos e Operações

- Um modelo normalmente contém operações ou procedimentos associados a ele, por exemplo:
 - Inclusão de um pedido para uma mesa.
 - Modificação do status de um pedido.
 - Encerramento dos pedidos de uma mesa.
- Também é possível a criação de modelos que possuem somente dados ou somente operações.
- Modelos podem conter submodelos e ser parte de outros modelos.

Modelos e Orientação a Objetos

- A simplificação inerente aos modelos é, em muitos casos, necessária: dependendo do contexto, algumas informações devem ser ocultas ou ignoradas.
 - Pessoa como empregado de empresa
 - Pessoa como paciente de uma clínica médica
 - Pessoa como contato comercial
- A criação e uso de modelos é uma tarefa natural e a extensão desta abordagem à programação deu origem ao paradigma **Programação Orientada a Objetos**.

Programação Orientada a Objetos

- Os modelos baseados em objetos são úteis para compreensão de problemas, para comunicação com os usuários das aplicações, para modelar empresa, projetar programas, etc.
- POO é um paradigma de programação de computadores onde se usam **classes** e **objetos**, criados a partir dos **modelos** descritos anteriormente, para representar e processar dados usando programas de computadores.

Programação Orientada a Objetos

- Os modelos baseados em objetos são úteis para compreensão de problemas, para comunicação com os usuários das aplicações, para modelar empresa, projetar programas, etc.
- POO é um paradigma de programação de computadores onde se usam **classes** e **objetos**, criados a partir dos **modelos**, para representar e processar dados usando programas de computadores.

Classes e Objetos

- Programadores que utilizam POO criam e usam **objetos** a partir de **classes**, que são relacionadas diretamente com os modelos descritos.
- Classes são estruturas utilizadas para descrever os modelos, ou seja, contém a descrição dos dados (atributos) e das operações (métodos) que podem ser realizadas sobre eles.
- Um **objeto** ou **instância** é a materialização da classe.

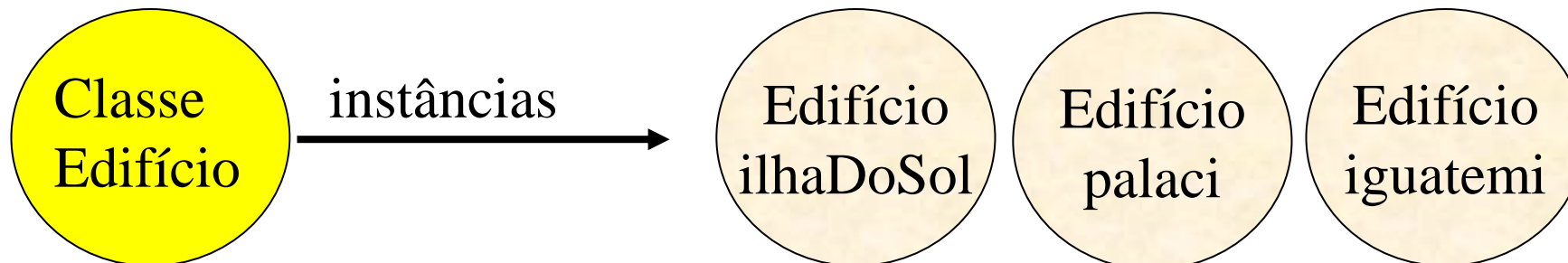


Classes e Objetos

- Os **dados** contidos em uma classe são conhecidos como **campos** ou **atributos** daquela classe.
- Cada campo deve ter um nome e ser de um tipo de dado nativo ou uma classe existente.
- Valores dentro de classes podem ser **variáveis**.
- As operações contidas são os **métodos**.
- **Métodos** podem receber argumentos (parâmetros).
- O nome mais os parâmetros do método são chamados de **assinatura**.

Classes e Objetos

- Para que **objetos** ou **instâncias** possam ser manipulados, é necessária a criação de **referências** a estes objetos, que são variáveis do “tipo” da classe.
- Classe -> Planta do edifício, que descreve o edifício, mas não corresponde fisicamente a ele.
- Instância -> Edifício construído.
- Referência -> Nome do Objeto (ilhaDoSol)



Classes em Java – Sintaxe Básica

- Uma classe em Java será declarada com a palavra-chave **class** seguida do nome da classe.
 - A convenção de mercado é que comece com uma letra maiúscula seguida de letras minúsculas.
 - Se for uma palavra composta, a segunda palavra deve começar com letra maiúscula sem espaço entre as palavras.
- Exemplo:

```
class Empregado {  
    String Nome;  
    public String getNome( ) {  
        return Nome;  
    }  
}
```

Nome da classe

Atributo

Método

Introdução a Linguagem Java

- Vamos abordar os seguintes tópicos:
 - Edições da tecnologia Java
 - Plataforma Java SE
 - Instalação do SDK
 - Instalação do Eclipse
 - Estrutura de uma aplicação Java

Edições da tecnologia Java

- Java ME - Java Micro Edition - dispositivos embarcados e móveis – IoT

<http://www.oracle.com/technetwork/java/javame>

- Java SE - Java Standard Edition - core - desktop e servidores

<http://www.oracle.com/technetwork/java/javase>

- Java EE - Java Enterprise Edition - aplicações corporativas

<http://www.oracle.com/technetwork/java/javaee>

Plataforma Java SE

- Entendendo as versões do Java

<https://www.oracle.com/java/technologies/downloads/>

- Versão atual = 19
 - Versão LTS atual = 17
- As empresas optam por utilizar a última versão LTS
 - LTS - Long Term Support
- Documentação

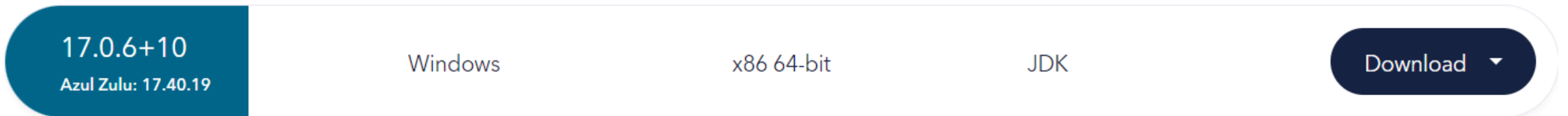
<https://docs.oracle.com/en/java/javase/17/index.html>

Instalação do Java SDK

- Baixar o JDK – Versão Zulu

<https://www.azul.com/downloads-new/?package=jdk#zulu>

- Escolha a opção indicada abaixo para Windows 64 bits ou outra opção SDK de acordo com o seu sistema operacional:



- Obs: Escolha a última versão LTS do SDK.

Instalação do Java SDK

- Vídeo de instalação da versão Zulu

<https://www.youtube.com/watch?v=QekeJBShCy4>


- Configurar variáveis de ambiente do sistema
 - Painel de Controle -> Variáveis de Ambiente (pesquisar -> var)
 - JAVA_HOME:
 - C:\Program Files\Java\zulu17.38.21-ca-jdk17.0.5-win_x64
 - Path: incluir
 - C:\Program Files\Java\zulu17.38.21-ca-jdk17.0.5-win_x64\bin
- Testar no terminal de comando (cmd):
 - java --version

Instalação do Eclipse

- Baixar o Eclipse no link a seguir:

<https://www.eclipse.org/downloads/packages/>


- Escolha a opção abaixo de acordo com o seu SO:



Eclipse IDE for Java Developers

302 MB 337,664 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration



Windows **x86_64**
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Instalação do Eclipse

- Descompacte o zip no drive C da sua máquina:

 eclipse-java-2022-12-R-win32-x86_64.zip

- Abra a pasta aonde descompactou a IDE e execute o arquivo:

 eclipse.exe	01/12/2022 20:19	Aplicativo	521 KB
---	------------------	------------	--------

- A IDE irá perguntar qual será a sua workspace (local que guardará seus projetos)
 - Escolha ou crie uma pasta no local e nome de sua preferência.
 - Ao abrir novamente o Eclipse, ele retorna a perguntar, qual será a workspace, indicando como padrão a última que utilizou.

Primeiros passos para utilizar o Eclipse

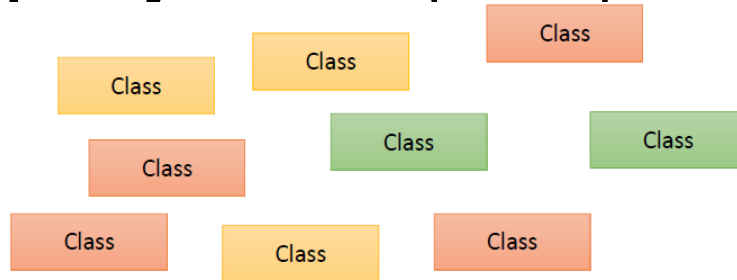
- Mudar o layout para o nosso curso:
 - Window -> Perspective -> Open Perspective -> Java
- Mostrar a aba Console:
 - Window -> Show View -> Console
- Para voltar ao layout padrão do nosso curso caso tenha aberto ou fechado alguma janela (Zerar o layout):
 - Window -> Perspective -> Reset Perspective
- Criar um projeto:
 - File -> New -> Java Project
 - Criar

Primeiros passos para utilizar o Eclipse

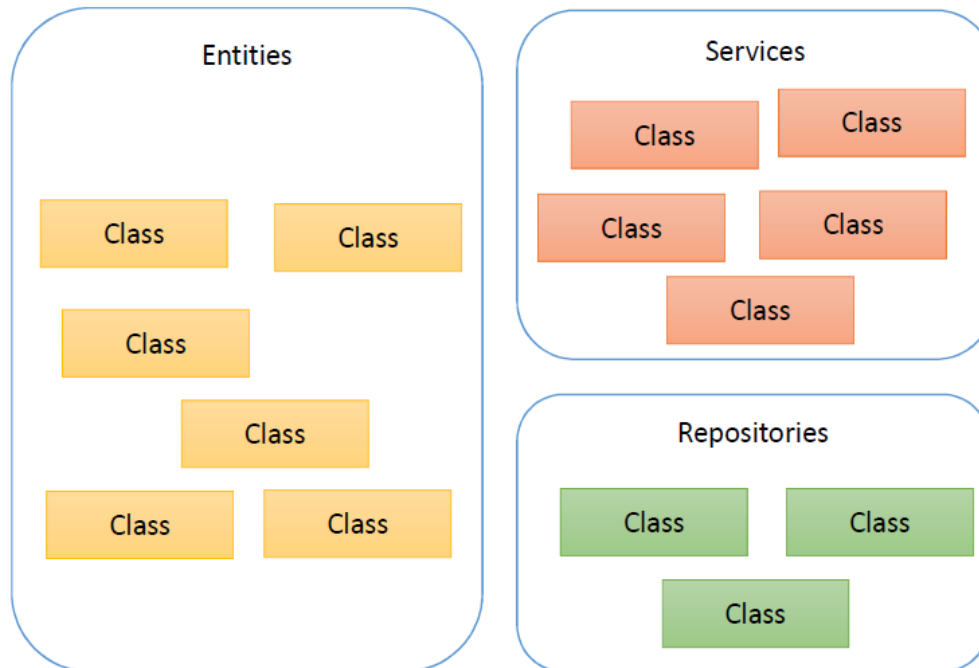
- Criar a primeira classe:
 - Botão direito na pasta "src" -> New -> Class
 - Package: coloque nomes indicando a que pacote a classe pertence. Por exemplo:
 - Se for um programa executável (com o método main) -> **aplicacao**
 - Se for uma classe de persistência de dados -> **servicos**
 - Se for uma classe indicando entidades do projeto -> **entidades**
 - Exemplos de classes que representam entidades para um modelo acadêmico:
 - Professor, Aluno, SalaDeAula, Disciplina
 - Para nomes de classes utilize a primeira letra em maiúsculo
 - Se for uma classe executável, marque a opção: public static void main(String[] args)
- Mudar o tamanho da fonte:
 - CTRL +
 - CTRL -

Estrutura de uma aplicação Java

- Uma **aplicação** é composta por classes

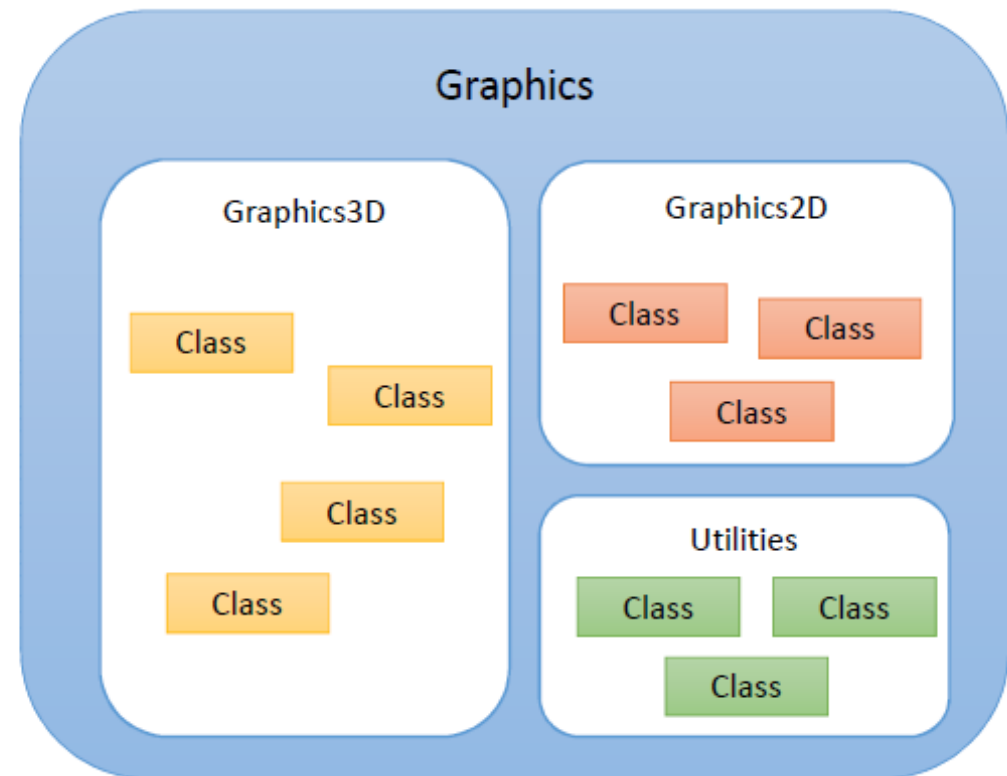
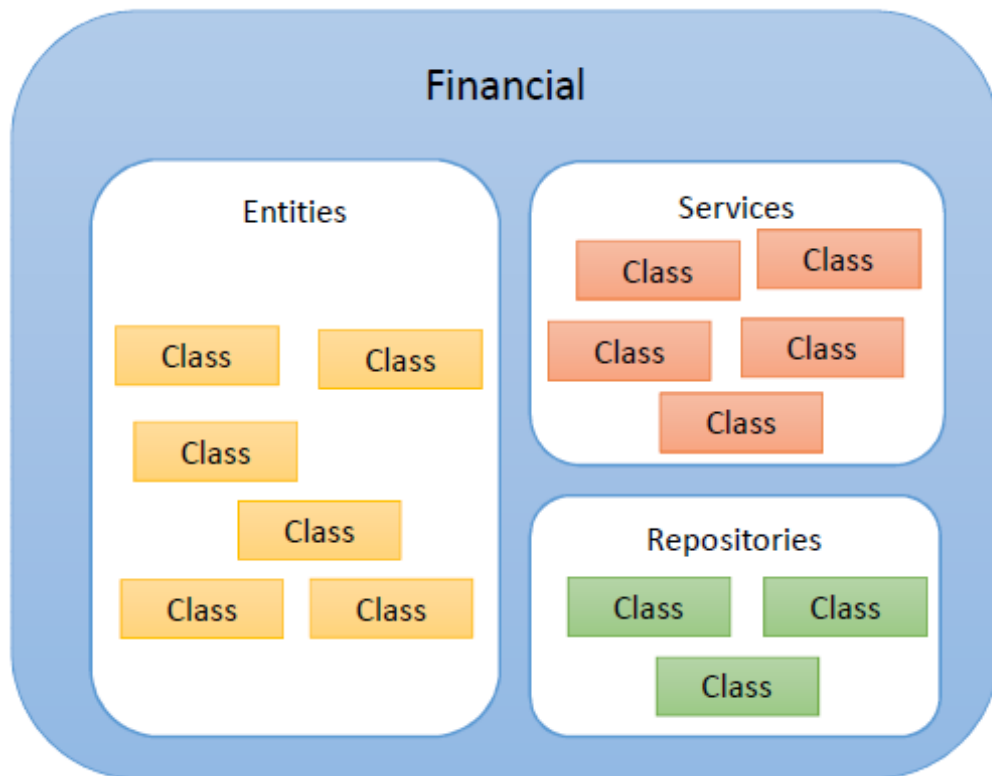


- **Package** é um agrupamento lógico de classes relacionadas



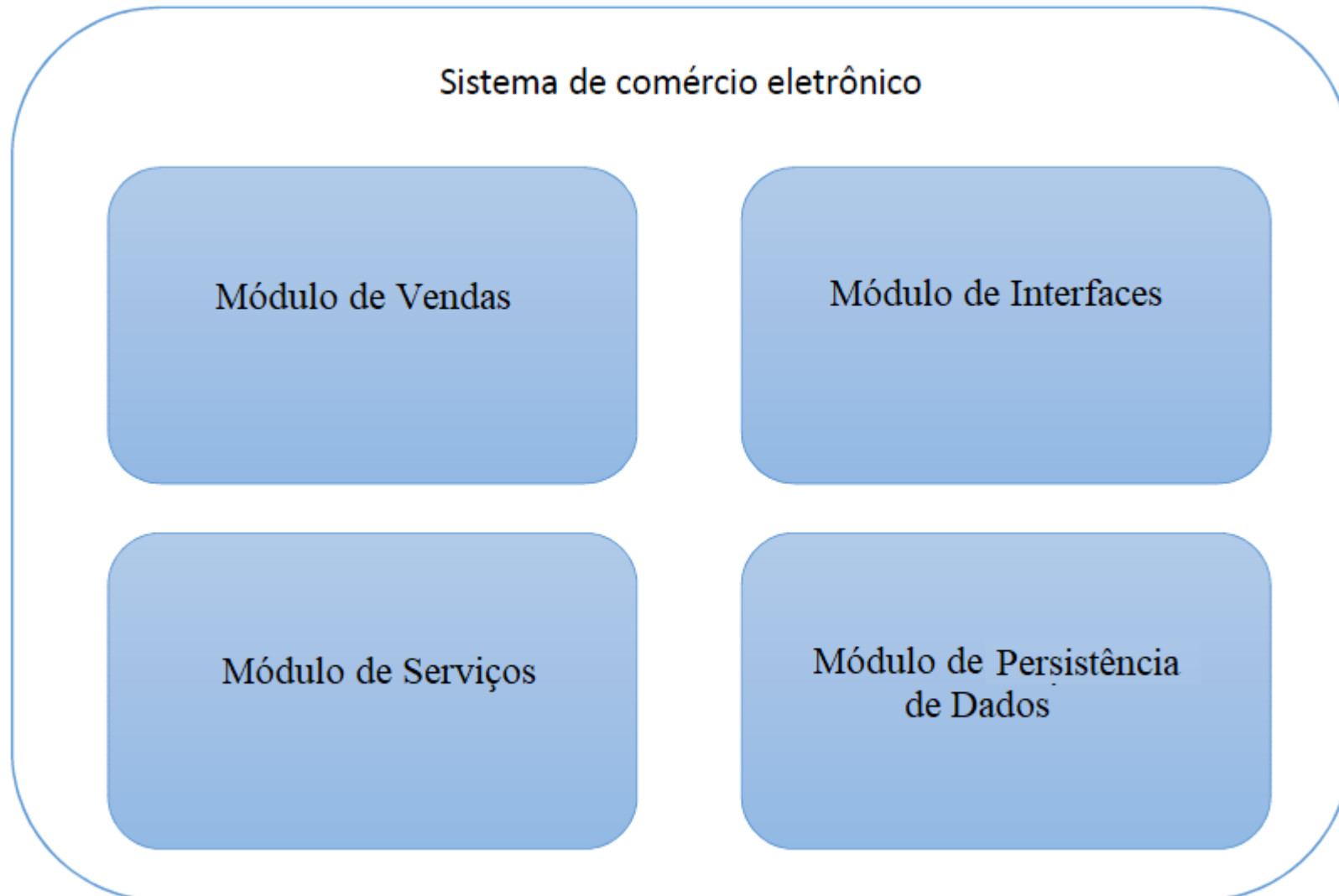
Estrutura de uma aplicação Java

- No Java (9 ou superior) um **módulo** é um agrupamento lógico de pacotes relacionados
- Runtime é um agrupamento físico (arquivos físicos compilados)



Estrutura de uma aplicação Java

- Uma aplicação é um agrupamento de módulos relacionados



Estrutura de uma aplicação Java

- Aplicações orientadas a objetos, são compostos por:
 - Conjuntos de objetos que representam os objetos existentes no negócio modelado.
 - Conjuntos de objetos que representam estruturas adequadas para a manipulação de dados.
 - Os objetos que compõem o programa comunicam-se por trocas de mensagens.
 - O programa precisa ter um ponto de entrada que indique à máquina virtual onde iniciar a execução.

```
public class Aplicacao {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

Estrutura de uma aplicação Java

- `public static void main(String [] args)`
 - **public**: é visível para qualquer outra classe.
 - **static**: dispensa a criação de objetos.
 - **void**: nenhum retorno esperado.
 - **main**: convenção para indicar para a máquina virtual qual o ponto de entrada da aplicação.
 - **String[] args**: parâmetros passados pela aplicação via linha de comando. args é o identificador.

Estrutura de uma aplicação Java

- Saída de Dados Padrão
 - Sem quebra de linha
 - **System.out.print("String"+variável)**
 - Com quebra de linha
 - **System.out.println("String"+variável)**
- Saída formatada.
 - **System.out.printf("String = %.2f%n",variável)**
 - %.2f -> duas casas após a vírgula
 - %n -> para pular a linha
 - %d -> para variáveis inteiras
 - %s -> para Strings

Estrutura de uma aplicação Java

- Entrada de dados com **Scanner**

```
import java.util.Scanner;  
public class MeuPrograma {
```

Precisamos importar a classe **Scanner** da biblioteca **java.util**

```
    public static void main(String[] args) {  
        int i;  
        Scanner sc=new Scanner(System.in);  
        i=sc.nextInt();  
        System.out.println("Valor digitado =" + i);  
        sc.close();  
    }
```

Precisamos criar um objeto instância da classe **Scanner** e informar a ele de onde virão os dados. Neste caso, do objeto **System.in** (teclado)

Utilizamos o objeto **sc** para fazer a leitura do teclado. Neste caso, **sc** está lendo um número inteiro.

Outras entradas do Scanner

nextByte()
nextInt()
nextFloat()
nextBoolean()
next().charAt(0)

nextShort()
nextLong()
nextDouble()
nextLine()

Introdução a Orientação a Objetos

- Vamos abordar os seguintes tópicos:
 - Classes
 - Atributos
 - Métodos
 - Membros estáticos

Resolvendo um problema sem OO

- Problema exemplo:
 - Fazer um programa para ler as medidas dos lados de dois triângulos X e Y (suponha medidas válidas). Em seguida, mostrar o valor das áreas dos dois triângulos e dizer qual dos dois triângulos possui a maior área.
 - A fórmula para calcular a área de um triângulo a partir das medidas de seus lados a, b e c é a seguinte (fórmula de Heron):

$$area = \sqrt{p(p-a)(p-b)(p-c)} \quad \text{onde} \quad p = \frac{a+b+c}{2}$$

- Exemplo:

Informe a dimensão dos três lados do triângulo X:

3,00

4,00

5,00

Informe a dimensão dos três lados do triângulo Y:

7,50

4,50

4,02

Área do triângulo X = 6,00

Área do triângulo Y = 7,56

Maior área = Triângulo Y

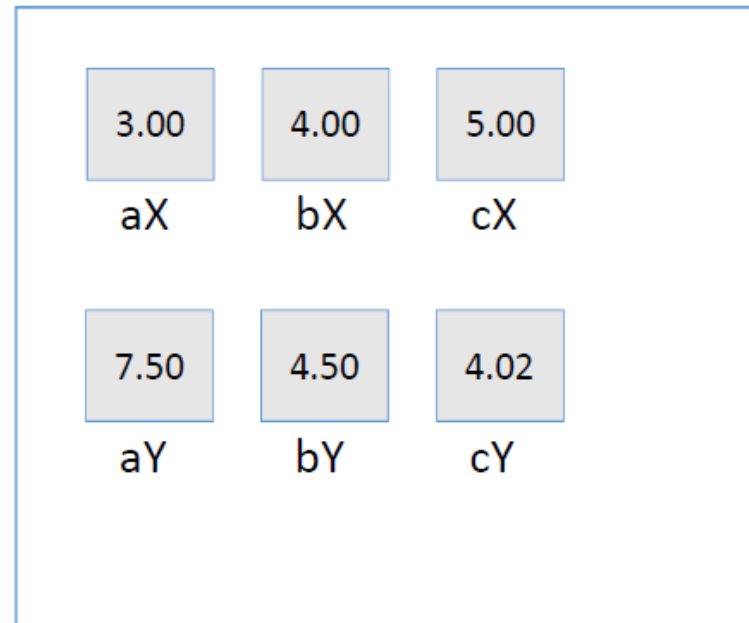
Resolvendo um problema sem OO

- Código disponível no GitHub:
 - Projeto Completo
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1>
 - Commit 3 “Resolvendo um problema sem OO”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1/commit/70edc0c787f1406dec032deefa0022abdc1018b4>
 - Programa.java
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1/commit/70edc0c787f1406dec032deefa0022abdc1018b4#diff-aaa4cf0c3807e4fb3b625ca43564649a57d71ee03e7407ae7e3866f79a7ef5cd>

Resolvendo um problema sem OO

- Discussão:
 - Triângulo é uma entidade com três atributos: a, b, c.
 - Estamos usando três variáveis distintas para representar cada triângulo:
 - `double aX, bX, cX, aY, bY, cY;`

Memória:



- Para melhorar isso, vamos usar uma CLASSE para representar um triângulo.

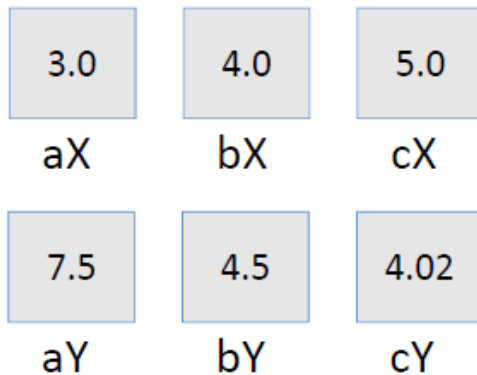
Classe

- É um tipo estruturado que pode conter (membros):
 - Atributos (dados / campos)
 - Métodos (funções / operações)
- A classe também pode prover muitos outros recursos, tais como:
 - Construtores
 - Sobrecarga
 - Encapsulamento
 - Herança
 - Polimorfismo
- Exemplos:
 - Entidades: Produto, Cliente, Triangulo
 - Serviços: ProdutoService, ClienteService, EmailService, StorageService
 - Controladores: ProdutoController, ClienteController
 - Utilitários: Calculadora, Compactador
 - Outros (views, repositórios, gerenciadores, etc.)

Criando a Classe Triângulo

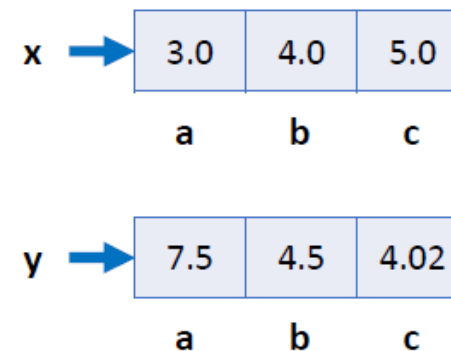
```
package entidades;  
  
public class Triangulo {  
    public double a, b, c;  
}
```

`double aX, bX, cX, aY, bY, cY;`



`Triangulo x, y;`

`x = new Triangulo();`
`y = new Triangulo();`



Criando a Classe Triângulo

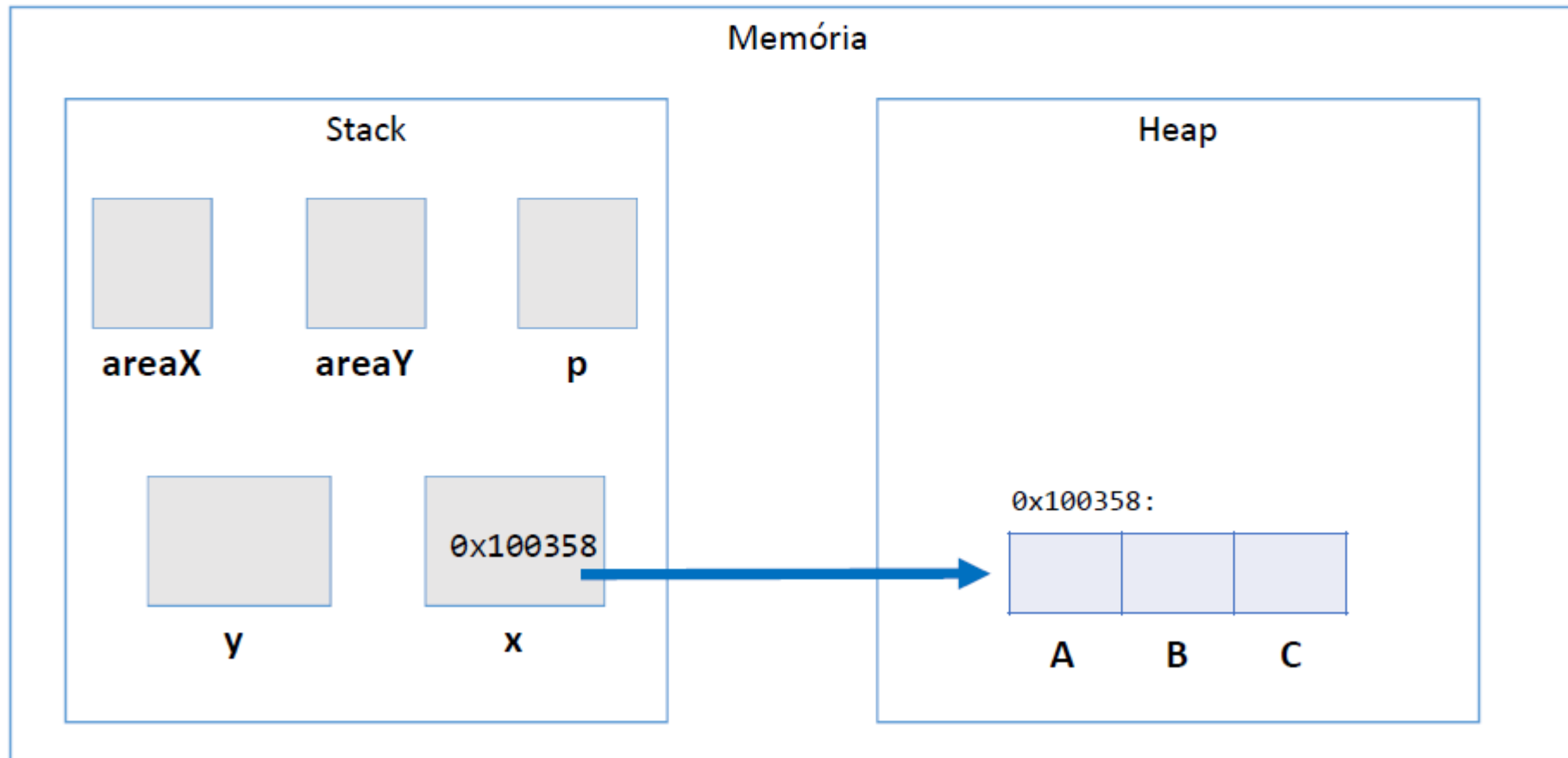
- Código disponível no GitHub:
 - Projeto Completo
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1>
 - Commit 4 “Criando uma classe Triângulo”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1/commit/061df0d35f02ea172fb4a728216a6ddfeb010f23>

Instanciação

(alocação dinâmica de memória)

```
double areaX, areaY, p;  
Triangulo x, y;
```

```
x = new Triangulo();
```

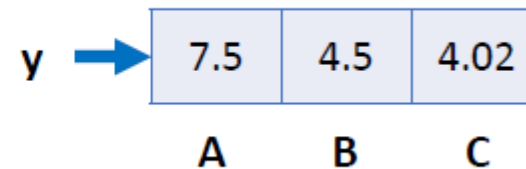
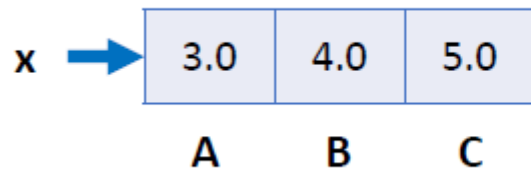


Classe, Objetos e Atributos

- Classe: é a definição do tipo

```
package entidades;  
  
public class Triangulo {  
    public double a, b, c;  
}
```

- Objetos: são instâncias da classe

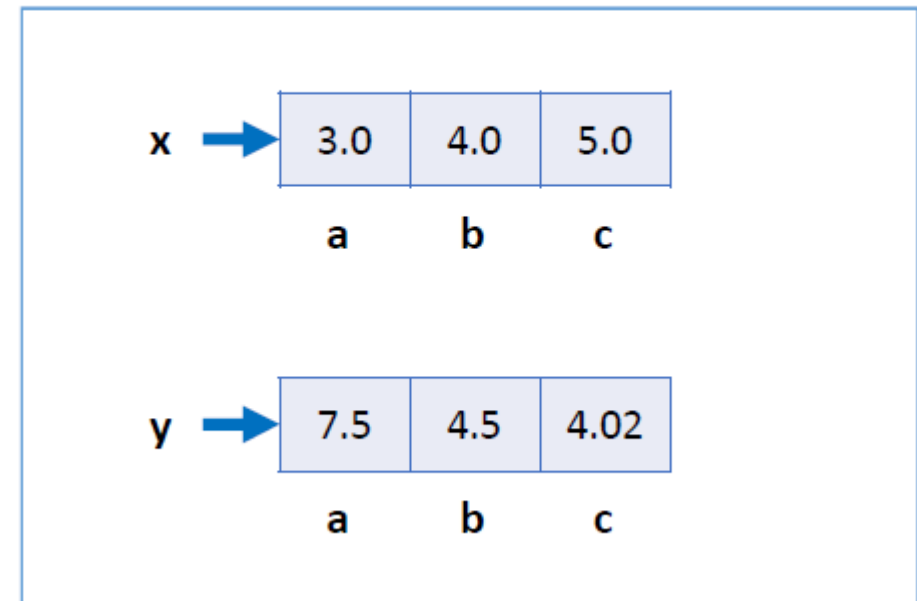


Classe, Objetos e Atributos

- Discussão:
 - Com o uso de CLASSE, agora nós temos uma variável composta do tipo "Triangulo" para representar cada triângulo:

```
Triangulo x, y;  
x = new Triangulo();  
y = new Triangulo();
```

Memória:

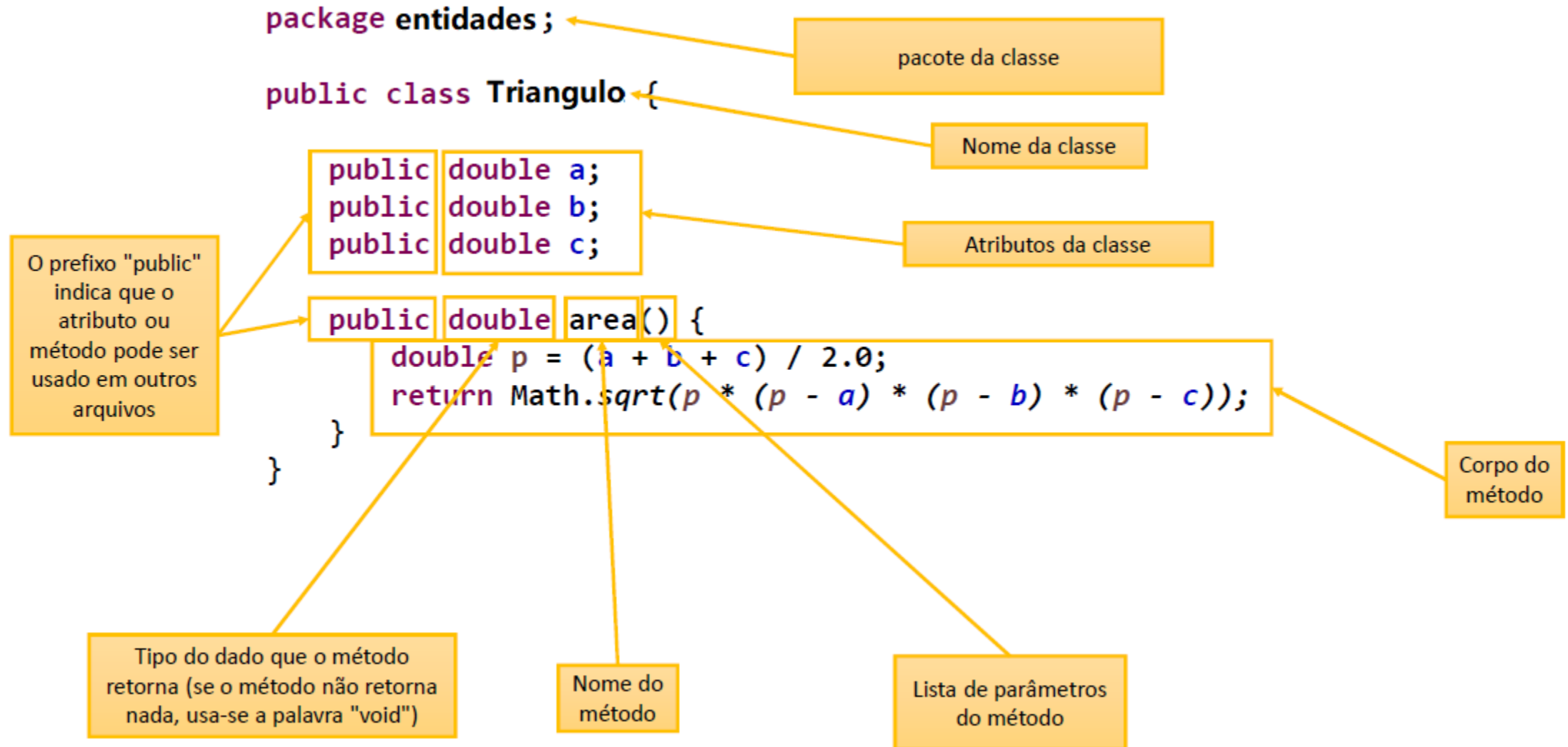


- Agora vamos melhorar nossa CLASSE, acrescentando nela um MÉTODO para calcular a área.

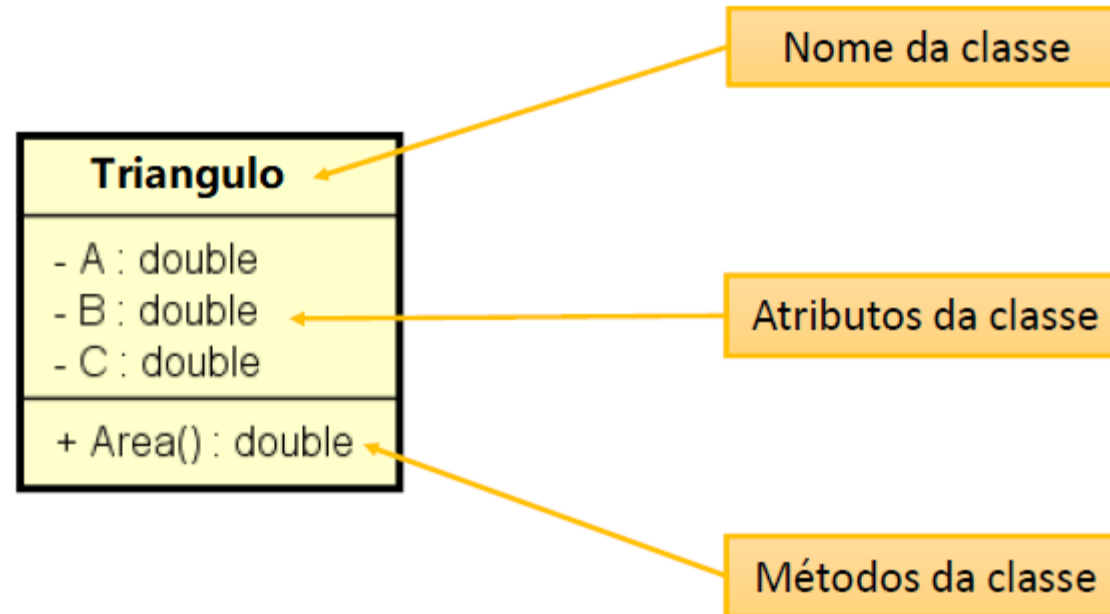
Criando um método

- Criando um método para obtermos os benefícios de **reaproveitamento e delegação**
- Código disponível no GitHub:
 - Projeto Completo
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1>
 - Commit 5 “Criando o método area”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo1/commit/324d9eaab11879f03cde99a0db628ac5555c18e8>

Visão geral da classe Triangulo




Projeto da classe Triângulo - UML



Benefícios da criação do método

- Discussão:
 - Quais são os benefícios de se calcular a área de um triângulo por meio de um MÉTODO dentro da CLASSE Triângulo?
 - 1) **Reaproveitamento de código**: nós eliminamos o código repetido (cálculo das áreas dos triângulos x e y) no programa principal.
 - 2) **Delegação de responsabilidades**: quem deve ser responsável por saber como calcular a área de um triângulo é o próprio triângulo. A lógica do cálculo da área não deve estar em outro lugar.

Membros Estáticos

- Membros estáticos são também chamados membros de classe em oposição a membros e instância.
- São membros que fazem sentido independentemente de objetos. Não precisam de objeto para serem chamados. São chamados a partir do próprio nome da classe.
- Aplicações comuns:
 - Classes utilitárias.  `Math.sqrt(double)`
 - Declaração de constantes.
 - Uma classe que possui somente membros estáticos, pode ser uma classe estática também. Esta classe não poderá ser instanciada.

Membros Estáticos

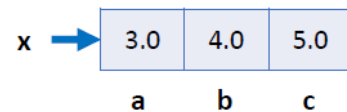
- Vamos resolver o exercício abaixo para demonstração da utilização de membros estáticos:
 - Fazer um programa para ler um valor numérico qualquer, mostrar quanto seria o valor de uma circunferência e do volume de uma esfera para um raio daquele valor. Informar também o valor do PI com duas casas decimais.
- Código disponível no GitHub:
 - Projeto Completo
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo2>

Membros Estáticos

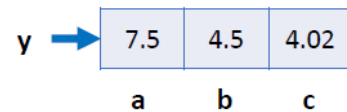
- Vamos evoluir o projeto do problema proposto em três versões:
 - Códigos disponíveis no gitHub
 - Versão 1:
 - Commit3: “Métodos na própria classe do programa”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo2/commit/1c8f5aef943ba3cfce766b45e6c368ce273936d6>
 - Versão 2:
 - Commit4: “Classe Calculadora com membros de instância”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo2/commit/c4c45b5eb3f456e2462f210d9dc0161646e4cc54>
 - Versão 3:
 - Commit5: “Classe Calculadora com membros estáticos”
 - Link: <https://github.com/arturKronbauer/IntroducaoOO-Exemplo2/commit/4496c0b325544f0ccad2eb366549349d273850d8>

Membros Estáticos

- Discussão:
 - No problema dos triângulos, cada triângulo possui sua área.
 - Area() é uma operação concernente ao objeto: cada triângulo possui sua área.



`x.area()` → 6.0



`y.area()` → 7.5638

- Já no caso da calculadora, os valores dos cálculos não mudam para calculadoras diferentes, ou seja, são cálculos estáticos. O valor de Pi também é estático.

```
Calculator calc1 = new Calculator();  
Calculator calc2 = new Calculator();
```

`calc1` → 3.14
Pi

`calc1.PI`
3.14
`calc1.circumference(3.0)`
18.85

`calc2` → 3.14
Pi

`calc2.PI`
3.14
`calc2.circumference(3.0)`
18.85

Construtores, palavra this, sobrecarga, encapsulamento

- Vamos abordar os seguintes tópicos:
 - Construtores
 - This
 - Sobrecarga
 - Encapsulamento

Construtores

- É uma operação especial da classe, que executa no momento da instanciação do objeto
- Usos comuns:
 - Iniciar valores dos atributos
 - Permitir ou obrigar que o objeto receba dados / dependências no momento de sua instanciação (injeção de dependência)
 - Se um construtor customizado não for especificado, a classe disponibiliza o construtor padrão:
 - `Produto p = new Produto();`
 - É possível especificar mais de um construtor na mesma classe (sobrecarga)

Construtores

- Para exemplificação da utilização de construtores vamos utilizar o seguinte exemplo:
 - Fazer um programa para ler os dados de um produto em estoque (nome, preço e quantidade no estoque). Em seguida:
 - Mostrar os dados do produto (nome, preço, quantidade no estoque, valor total no estoque)
 - Realizar uma entrada no estoque e mostrar novamente os dados do produto
 - Realizar uma saída no estoque e mostrar novamente os dados do produto
- Obs: a primeira versão não utiliza construtor e a segunda utiliza. Faremos uma discussão para ressaltar os benefícios de utilizar construtores.

Construtores

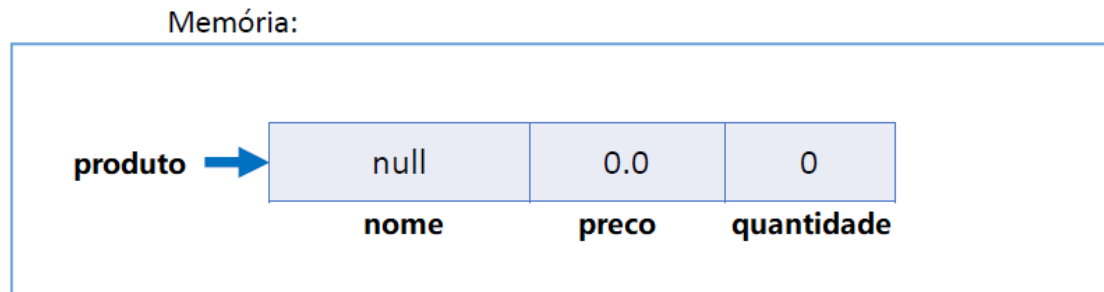
- Vamos evoluir o projeto do problema proposto para mostrar os conceitos de construtores, this, sobrecarga e encapsulamento.
- Disponíveis no gitHub
 - Versão 1:
 - Commit3: “Projeto sem construtores”
 - Link: <https://github.com/arturKronbauer/exemplo-Construtores-This-Sobrecarga-Encapsulamento/commit/72db9b3ad38523f278b3924a5c8f9e73e3189c90>

Construtores

- Proposta de melhoria

- Quando executamos o comando abaixo, instanciamos um produto com seus atributos “vazios”:

```
produto = new Produto();
```



- Entretanto, faz sentido um produto que não tem nome? Faz sentido um produto que não tem preço?
 - Com o intuito de evitar a existência de produtos sem nome e sem preço, é possível fazer com que seja “obrigatória” a iniciação desses valores?
- Versão 2:
 - Commit4: “Projeto com construtor”
 - Link: <https://github.com/arturKronbauer/exemplo-Construtores-This-Sobrecarga-Encapsulamento/commit/880c464b7cf79c690643dfd4fb20810b1e347a3c>

Palavra this

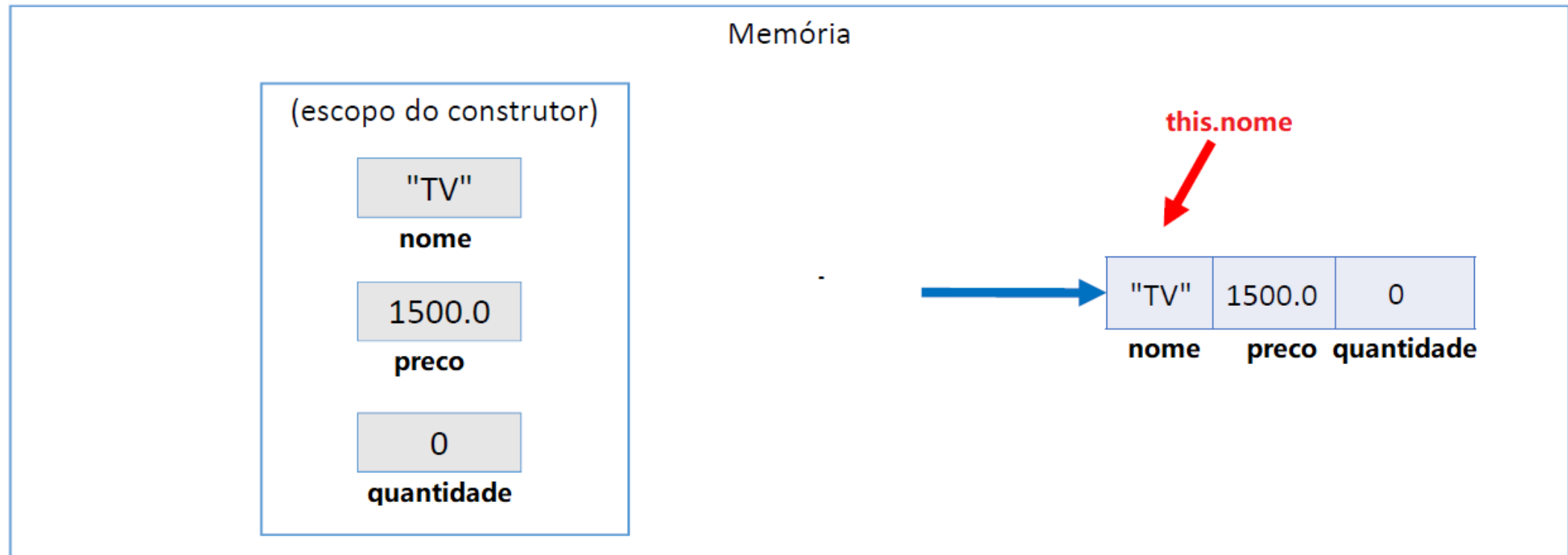
- É uma referência para o próprio objeto
- Usos comuns:
 - Diferenciar atributos de variáveis locais
 - Passar o próprio objeto como argumento na chamada de um método ou construtor

Palavra this

- Diferenciar atributos de variáveis locais

```
Produto produto = new Produto("TV", 1500.00, 0);
```

```
public Produto(String nome, double preco, int quantidade) {  
    this.nome = nome;  
    this.preco = preco;  
    this.quantidade = quantidade;  
}
```



Palavra this

- Passar o próprio objeto como argumento na chamada de um método ou construtor

```
public class Funcionario {  
    String nome;  
    float salario;  
    public Funcionario(String nome, float salario) {  
        this.nome = nome;  
        this.salario = salario;  
    }  
  
    public float ajustaSalario(float percentual) {  
        float novoSalario;  
        novoSalario = this.salario + calculo(percentual,this);  
        return novoSalario;  
    }  
  
    private float calculo(float percentual, Funcionario obj) {  
        float valorAcrescido;  
        valorAcrescido = obj.salario * percentual / 100;  
        return valorAcrescido;  
    }  
}
```

Uso do **this** como parâmetro para referenciar o objeto corrente

Uso do **this** para acessar um atributo do objeto.

Sobrecarga

- Um método pode ter o mesmo nome que outro método na mesma classe;
- Isto é usual quando existem duas ou mais formas diferentes de se realizar a mesma tarefa;
- Neste caso diz-se que o método está sobrecarregado;
- Java permite a criação de métodos com nome iguais, contanto que as assinaturas sejam diferentes;
- A assinatura é composta do nome mais os tipos de argumento. O tipo de retorno não faz parte da assinatura;
- Diferenças do tipos dos parâmetros é que definem assinaturas diferentes.

Sobrecarga

- Java admite também, que se sobrecarregue os construtores de uma classe.
- As restrições são similares às aquelas aplicadas aos métodos sobrecarregados
- Deve-se se referir de dentro de um construtor sobrecarregado para outro, através do uso da palavra reservada `this`.
- Nota: é possível também incluir um construtor padrão

Sobrecarga

- Proposta de melhoria para o projeto
 - Vamos criar um construtor opcional, o qual recebe apenas nome e preço do produto. A quantidade em estoque deste novo produto, por padrão, deverá então ser iniciada com o valor zero.
- Versão 3:
 - Commit5: “Incluindo sobrecarga no projeto”
 - Link: <https://github.com/arturKronbauer/exemplo-Construtores-This-Sobrecarga-Encapsulamento/commit/d5313ee0ff44892c7807a66786076188f739a431>

Encapsulamento

- É um princípio que consiste em esconder detalhes de implementação de uma classe, expondo apenas operações seguras e que mantenham os objetos em um estado consistente.
- Regra de ouro: o objeto deve sempre estar em um estado consistente, e a própria classe deve garantir isso.
- Regra geral básica: um objeto NÃO deve expor nenhum atributo (modificador de acesso private)
- Os atributos devem ser acessados por meio de métodos get e set



Encapsulamento

- Gerando automaticamente getters e setters no Eclipse.
 - Botão direito -> Source -> Generate Getters and Setters
- Gerando automaticamente construtores no Eclipse.
 - Botão direito -> Source -> Generate Constructor using Fields

Modificadores de acesso

- Documentação:
 - <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>
- **private**: o membro só pode ser acessado na própria classe
- **(nada)**: o membro só pode ser acessado nas classes do mesmo pacote
- **protected**: o membro só pode ser acessado no mesmo pacote, bem como em subclasses de pacotes diferentes
- **public**: o membro é acessado por todas as classes (ao menos que ele resida em um módulo diferente que não exporte o pacote onde ele está)

Encapsulamento e Modificadores

- Proposta de melhoria para o projeto
 - Vamos acrescentar os getters e setters na classe produto e proteger todos os seus atributos para que não possam ser acessados diretamente. Reescreva a classe Programa para que se ajuste ao encapsulamento provido na classe produto.
- Versão 4:
 - Commit6: “Incluindo no projeto os conceitos de encapsulamento”
 - Link: <https://github.com/arturKronbauer/exemplo-Construtores-This-Sobrecarga-Encapsulamento/commit/6a21b4d42f9b81a8100b3c4d313ae64e5ea2d8bc>

Comportamento de memoria e listas

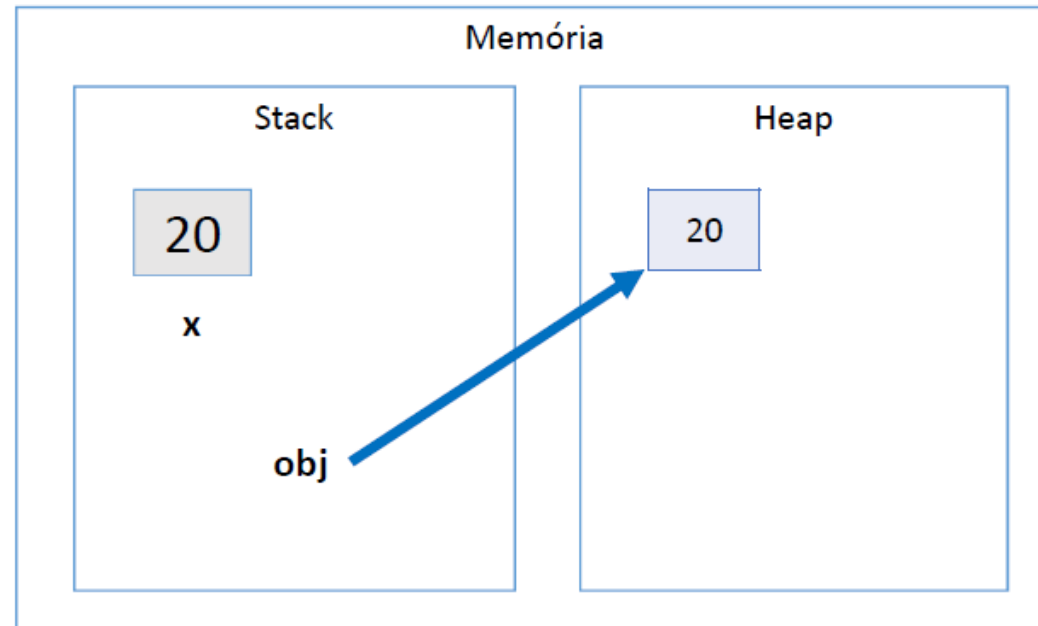
- Vamos abordar os seguintes tópicos:
 - Boxing
 - Unboxing
 - Wrapper classes
 - Laço for each
 - listas

Boxing

- É o processo de conversão de um objeto tipo valor (tipo primitivo) para um objeto tipo referência compatível

```
int x = 20;
```

```
Object obj = x;
```



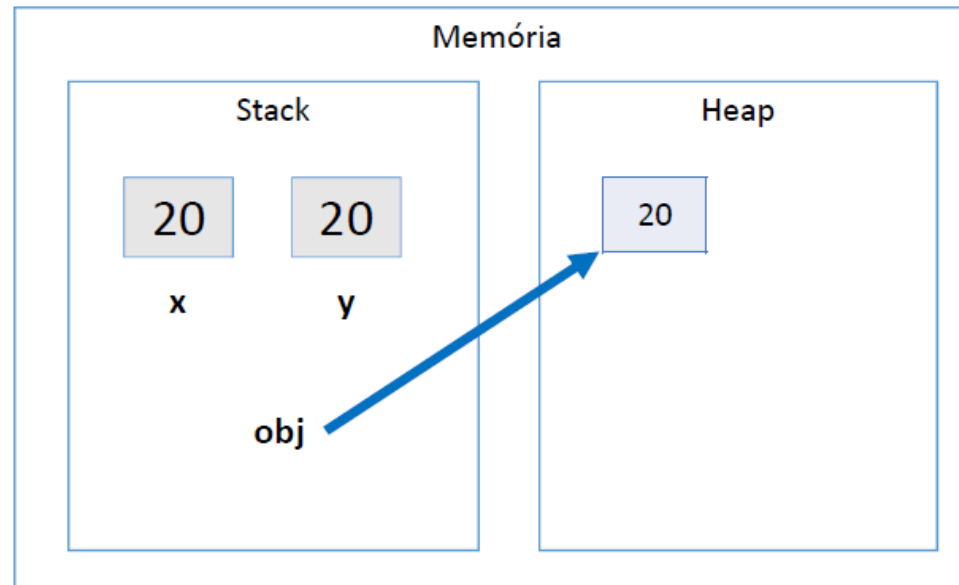
Unboxing

- É o processo de conversão de um objeto tipo referência para um objeto tipo valor (tipo primitivo) compatível

```
int x = 20;
```

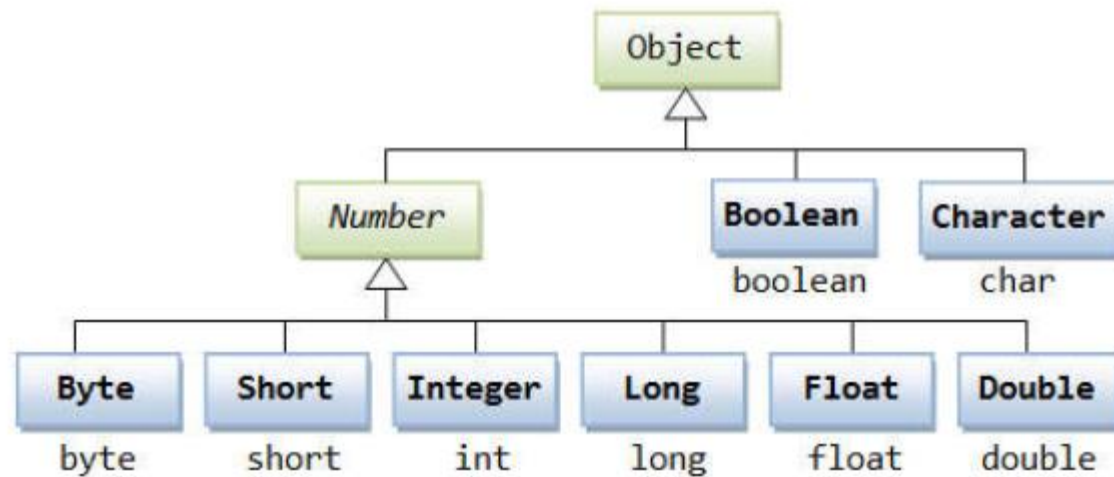
```
Object obj = x;
```

```
int y = (int) obj;
```



Wrapper classes

- São classes equivalentes aos tipos primitivos
- Boxing e unboxing é natural na linguagem
- Uso comum: campos de entidades em sistemas de informação, por exemplo, banco de dados
 - Pois tipos referência (classes) aceitam valor **null** e usufruem dos recursos OO



Wrapper classes

- Exemplo de utilizações:

```
Integer x = 10;
```

```
int y = x * 2;
```

```
public class Product {  
  
    public String name;  
    public Double price;  
    public Integer quantity;  
  
    (...)
```

Laço for each

- Sintaxe opcional e simplificada para percorrer coleções

- Sintaxe:

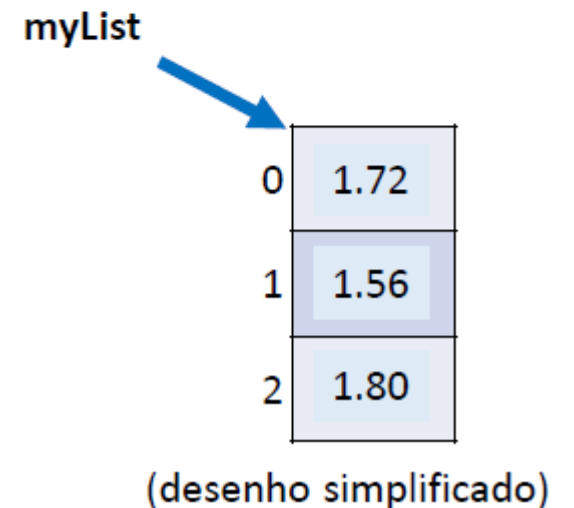
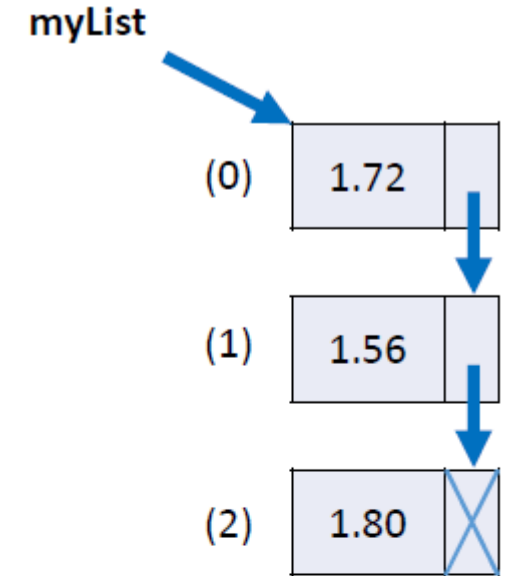
```
for (Tipo apelido : coleção) {  
    <comando 1>  
    <comando 2>  
}
```

- Leitura: "para cada objeto 'obj' contido em vect, faça:"

```
String[] vect = new String[] {"Maria", "Bob", "Alex"};  
  
for (int i=0; i< vect.length; i++) {  
    System.out.println(vect[i]);  
}  
  
for (String obj : vect) {  
    System.out.println(obj);  
}
```

Listas

- Lista é uma estrutura de dados:
 - Homogênea (dados do mesmo tipo)
 - Ordenada (elementos acessados por meio de posições)
 - Inicia vazia e seus elementos são alocados sob demanda
 - Cada elemento ocupa um "nó" (ou nodo) da lista
- Tipo (interface): List
- Classes que implementam: ArrayList, LinkedList, etc.
- Vantagens:
 - Tamanho variável
 - Facilidade para se realizar inserções e deleções
- Desvantagens:
 - Acesso sequencial aos elementos



Listas

- Métodos para manipulação de listas:
 - Tamanho da lista: `size()`
 - Obter o elemento de uma posição: `get(position)`
 - Inserir elemento na lista: `add(obj)`, `add(int, obj)`
 - Remover elementos da lista: `remove(obj)`, `remove(int)`, `removeIf(Predicate)`
 - Encontrar posição de elemento: `indexOf(obj)`, `lastIndexOf(obj)`
 - Filtrar lista com base em predicado:
 - `List<Integer> result = list.stream().filter(x -> x > 4).collect(Collectors.toList());`
 - Encontrar primeira ocorrência com base em predicado:
 - `Integer result = list.stream().filter(x -> x > 4).findFirst().orElse(null);`

Listas

- Lista é uma estrutura de dados:
 - Homogênea (dados do mesmo tipo)
 - Ordenada (elementos acessados por meio de posições)
 - Inicia vazia e seus elementos são alocados sob demanda
 - Cada elemento ocupa um "nó" (ou nodo) da lista
- Tipo (interface): List
- Classes que implementam: ArrayList, LinkedList, etc.
- Vantagens:
 - Tamanho variável
 - Facilidade para se realizar inserções e deleções
- Desvantagens:
 - Acesso sequencial aos elementos

Listas

- Vamos analisar um exemplo de listas que possa manipular vários nomes de pessoas e realizar as seguintes operações:
 - Como instância uma lista.
 - Como adicionar dados no final da lista.
 - Como adicionar um dado em um índice específico.
 - Como pegar o tamanho de uma lista.
 - Como escrever uma lista.
 - Como remover elementos que satisfazem a uma expressão lambda (predicado).
 - Como identificar o índice de um determinado elemento da lista.
 - Como criar uma lista com os elementos de uma lista original que satisfaçam a uma expressão lambda.
 - Como filtrar um elemento a partir de uma expressão lambda.
- O código está disponível no Github:
 - Link: <https://github.com/arturKronbauer/exemplo-Manipulacao-Listas>

Enumeração e Composição

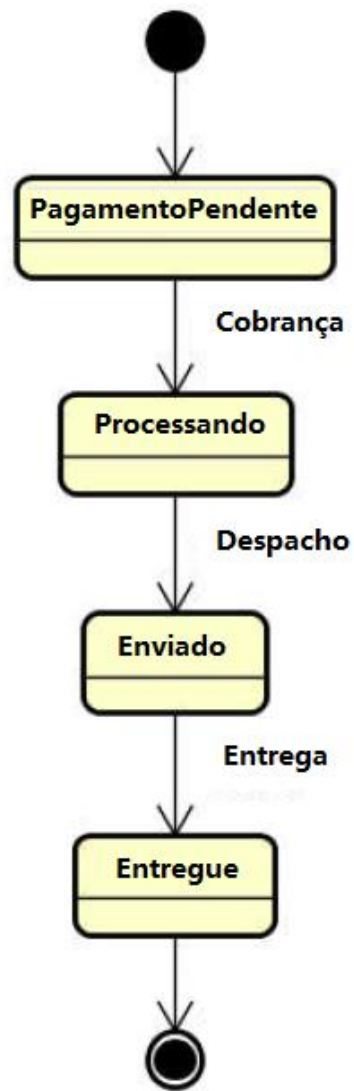
- Vamos abordar os seguintes tópicos:
 - Enumeração
 - Definição / discussão
 - Exemplo: estados de um pedido
 - Conversão de string para enum
 - Representação UML
 - Categorias de Classes
 - Composição

Enumeração

- Enumeração é um tipo especial que serve para especificar de forma literal um conjunto de constantes relacionadas.
- Palavra chave em Java: **enum**
- Vantagem: melhor semântica, código mais legível e auxiliado pelo compilador
- Referência:
<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

Enumeração

- Exemplo: Ciclo de vida de um pedido.



Enumeração

- Representação da classe do tipo **enum (StatusPedido)** e da sua utilização no **Pedido**.

```
package entidades.Enumeradas;

public enum StatusPedido {
    PAGAMENTO_PENDENTE,
    PROCESSANDO,
    ENVIADO,
    ENTREGUE;
}
```

```
package entidades;

import java.util.Date;

import entidades.Enumeradas.StatusPedido;

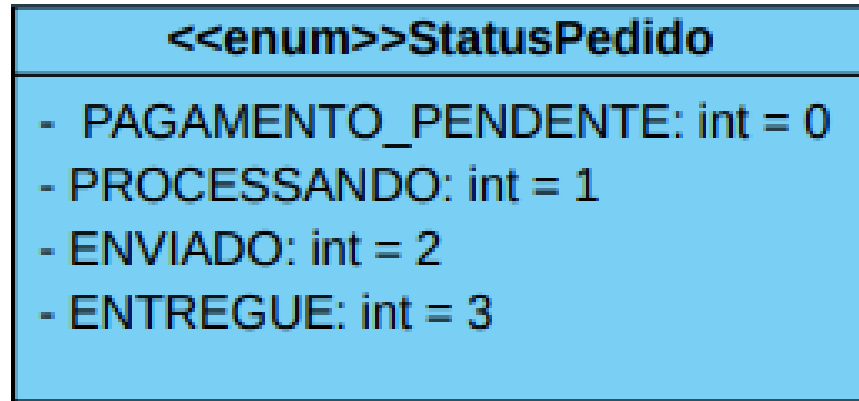
public class Pedido {
    private Integer id;
    private Date momento;
    private StatusPedido status;

    public Pedido(Integer id, Date momento, StatusPedido status) {
        this.id = id;
        this.momento = momento;
        this.status = status;
    }
}
```

- Disponível no Github:
 - Link: <https://github.com/arturKronbauer/exemplo-Enumeracao>

Enumeração

- Notação UML

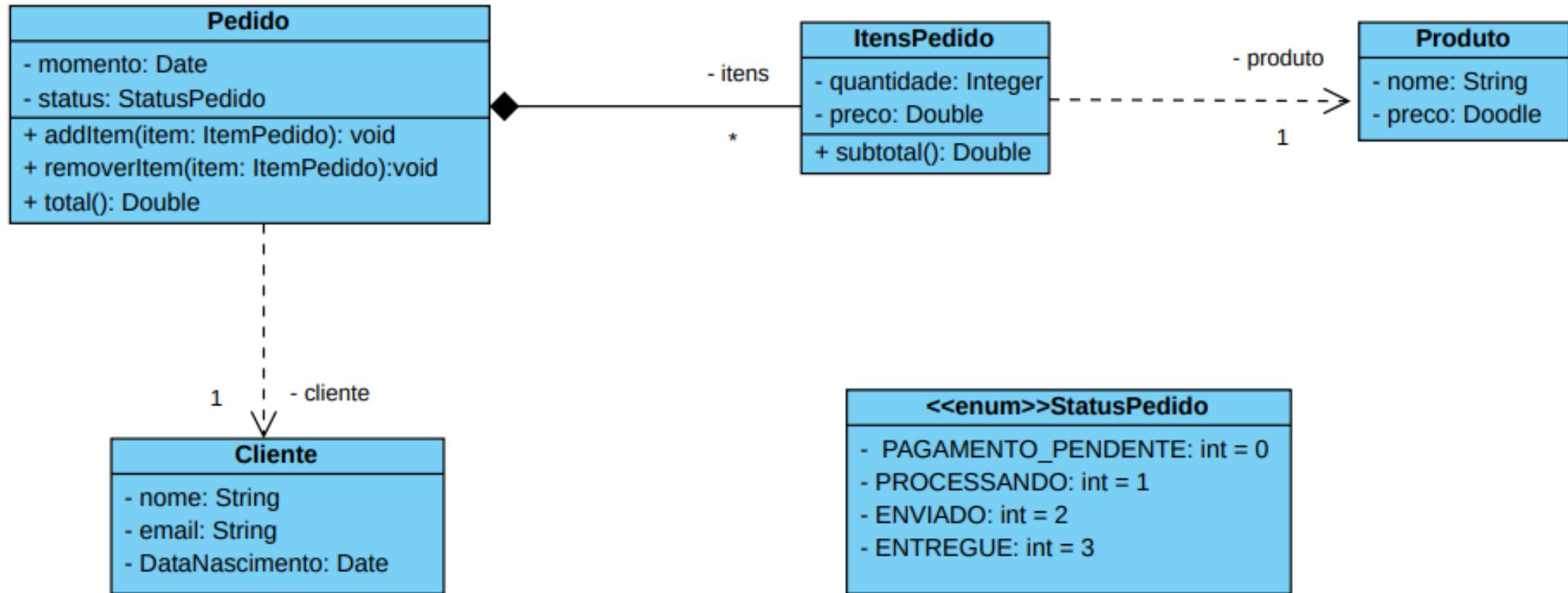


Composição

- É um tipo de associação que permite que um objeto contenha outro
- Relação "tem-um" ou "tem-vários"
- Vantagens
 - Organização: divisão de responsabilidades (cada classe tem a sua responsabilidade)
 - Coesão: cada objeto é responsável por uma única coisa bem definida
 - Flexibilidade: trabalhar com algo que está dividido em partes é mais flexível do que trabalhar com algo muito grande ou mau dividido
 - Reuso: o mesmo objeto pode ser aproveitado em mais de um lugar
- Nota: embora o símbolo UML para composição (todo-parte) seja o diamante preto, neste contexto estamos chamando de composição qualquer associação tipo "tem-um" e "tem-vários".

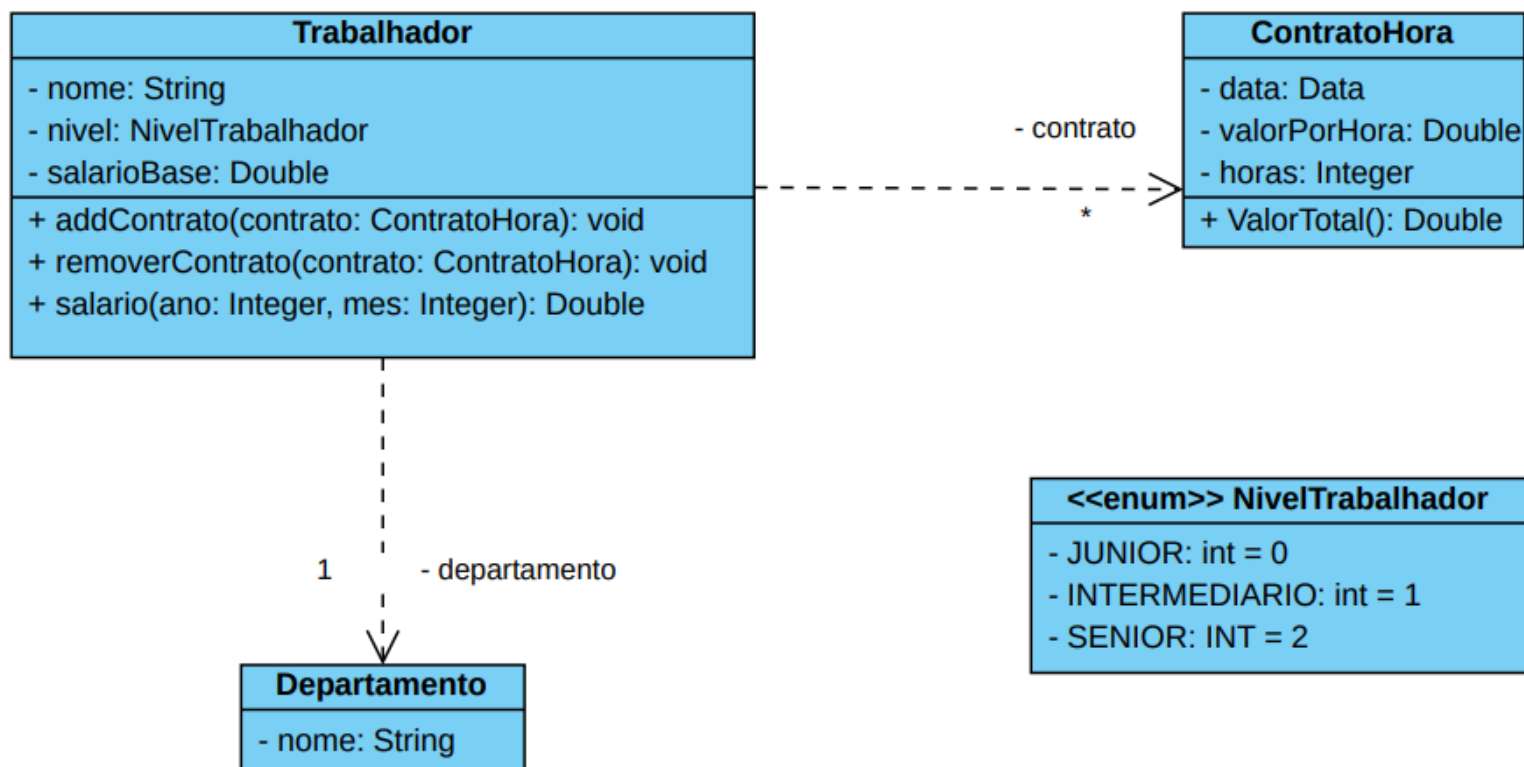
Composição

- Entidades



Composição

- Exemplo de Composição
 - Ler os dados de um trabalhador com N contratos (N fornecido pelo usuário). Depois, solicitar do usuário um mês e mostrar qual foi o salário do funcionário nesse mês.



Composição

- Exemplo de interação

Insira o nome do departamento: **Design**

Insira os dados do trabalhador:

Nome: **Alex**

Nível: **JUNIOR**

Salário base: **1200,00**

Quantos contratos para este trabalhador? **3**

Insira os dados do contrato # 1:

Data (DD/MM/AAAA): **20/02/2023**

Valor por hora: **50,00**

Duração (horas): **20**

Insira os dados do contrato # 2:

Data (DD/MM/AAAA): **13/01/2023**

Valor por hora: **30,00**

Duração (horas): **18**

Insira os dados do contrato # 3:

Data (DD/MM/AAAA): **25/01/2023**

Valor por hora: **80,00**

Duração (horas): **10**

Digite o mês e o ano para calcular o rendimento (MM/AAAA): **01/2023**

Nome: Alex

Departamento: Design

Renda de 02/2023: 2.540,00