TEPOO

AULA 03 – ORIENTAÇÃO A OBJETOS

UNIP
UNIVERSIDADE PAULISTA

Quais as vantagens

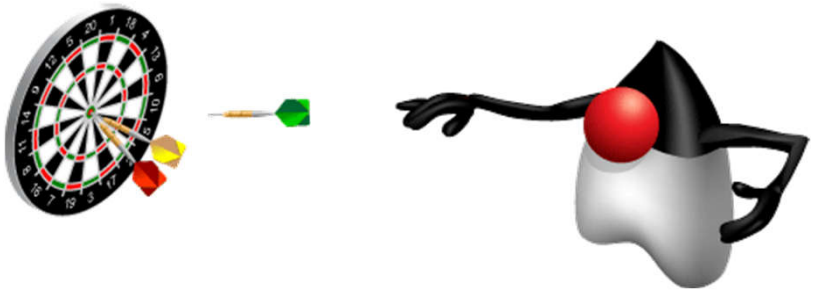
Orientação a objetos vai te ajudar em muito em se organizar e escrever menos, além de as responsabilidades nos pontos certos, flexibilizando sua aplicação, **encapsulando a lógica de** negócios. Outra enorme vantagem, onde você realmente vai economizar montanhas de código, é o **polimorfismo** das referências.

Quais as vantagens

- Pacote
- Classe
- Atributos
- Método main()
- Instâncias
- Métodos get e set
- Método toString()
- Construtor
- Herança
- Polimorfismo
- Classe abstrata
- Interface
- Membros estáticos

Objetivos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar uma classe
- Modelar uma classe



Relevância

O que é taxonomia?

Muito antes do alvorecer a ciência os seres humanos já nomeavam as espécies;

Isso os permitia obter sucesso nas suas atividade de caça e coleta;

A Taxonomia, palavra de origem grega cujo significado é “estudo das classificações”, surgiu no século XVII;

Ela ganhou força no século seguinte, graças as trabalho do naturalista sueco Carl Linnaeus, que inventou um sistema para organizar os seres vivos em grupos cada vez memores;

Neste sistema, os membros de um grupo particular compartilham determinadas características.

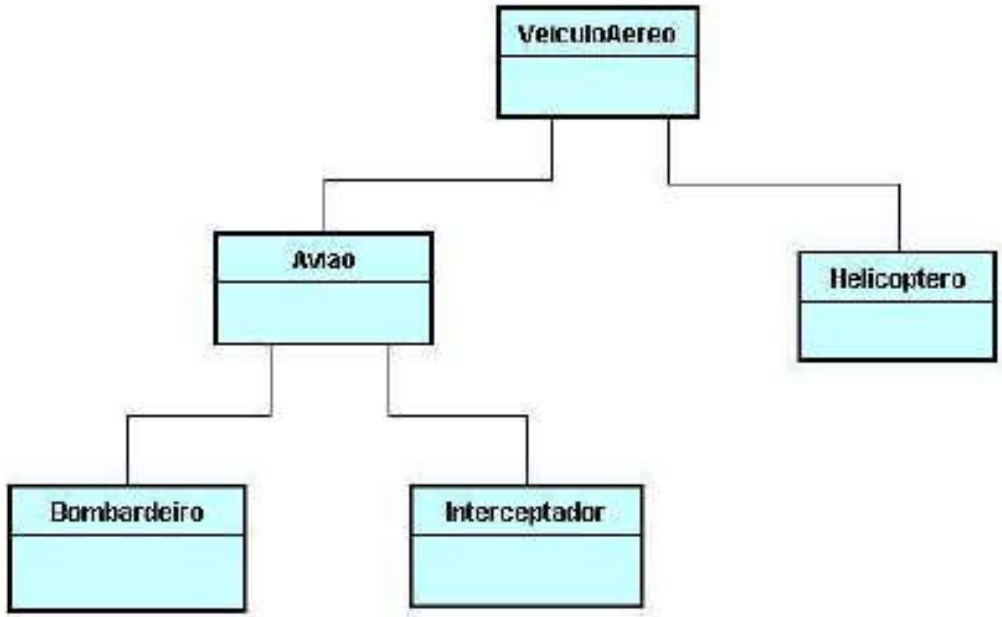
Taxonomia


[TAXONOMIA PADRÃO]
O Universo de Linnaeus
Carl Linnaeus desenvolveu as bases para a moderna taxonomia no século 18, ordenando todos os seres biológicos em grupos hierárquicos, partindo do nível dos reinos (como animais, plantas, fungos) e descendo até o nível das espécies individuais, cada um com um conjunto exclusivo de características observáveis.
Reino
Filo
Classe
Ordem
Família
Género
Espécie
Menos Específico
Mais Específico


Hierarquia Taxonómica
REINO
Animalia
FILO
Chordata
CLASSE
Aves
ORDEM
Strigiformes
FAMÍLIA
Strigidae
GÉNERO
Athene
ESPÉCIE
Athene noctua
STRI - Rapinas Nocturnas de Portugal

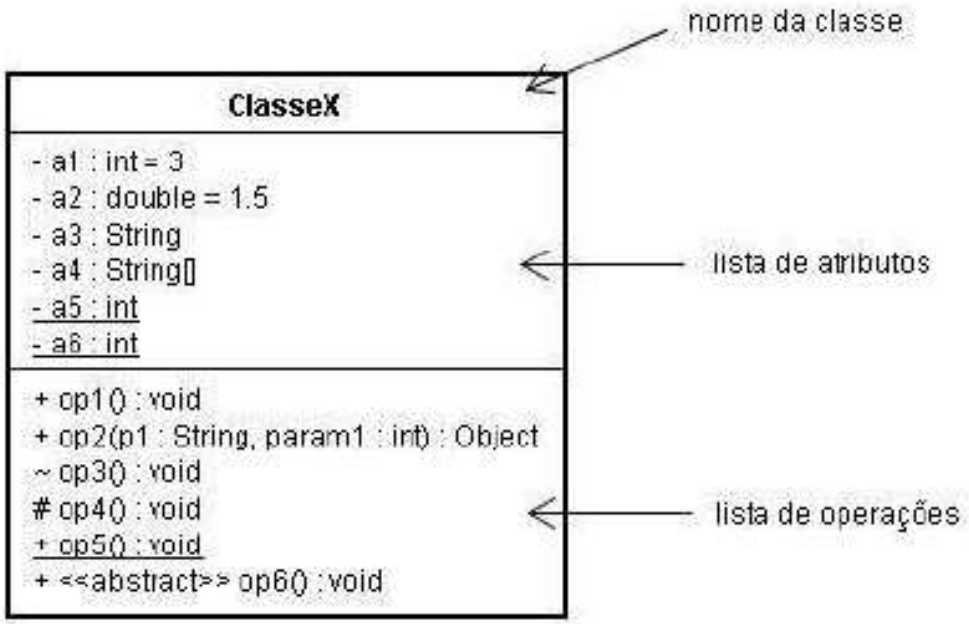
Taxonomia

- Posteriormente, a palavra **taxonomia** começou a ser usada em um sentido mais abrangente, podendo ser aplicada na classificação de quase tudo – objetos animados, inanimados, lugares e eventos.



Taxonomia

- Na UML, uma classe é graficamente representada por um retângulo dividido em três seções.



Generalização

- A **generalização** é atividade de identificar aspectos comuns e não comuns entre os conceitos pertencentes a um domínio de aplicação;
- Ela nos permite definir relações entre superclasses – conceitos gerais – e subclasses – conceitos específicos;
- Tais relações formam uma **taxonomia** de conceitos de um certo domínio, que é ilustrada através de uma hierarquia de classes.

Relevância

- Como você decide quais componentes são necessários para algo que irá construir, como uma casa ou uma mobília?
- O que é uma taxonomia?
- Como os organismos de uma taxonomia estão relacionados?
- Qual é a diferença entre atributos e valores?

Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Analisando um Problema Usando a Análise Orientada a Objetos (OOA)

- C&B vende roupas de seu catálogo. Como os negócios estão crescendo 30% ao ano, eles precisam de um novo sistema de entrada de pedidos.

Processo de Pedido da Lojas C&B

Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Identificando o Domínio de um Problema

- O domínio de um problema é o escopo do problema que será solucionado.
- Exemplo: “Criar um sistema para permitir que o método de entrada de pedidos on-line aceite e verifique o pagamento de um pedido”.

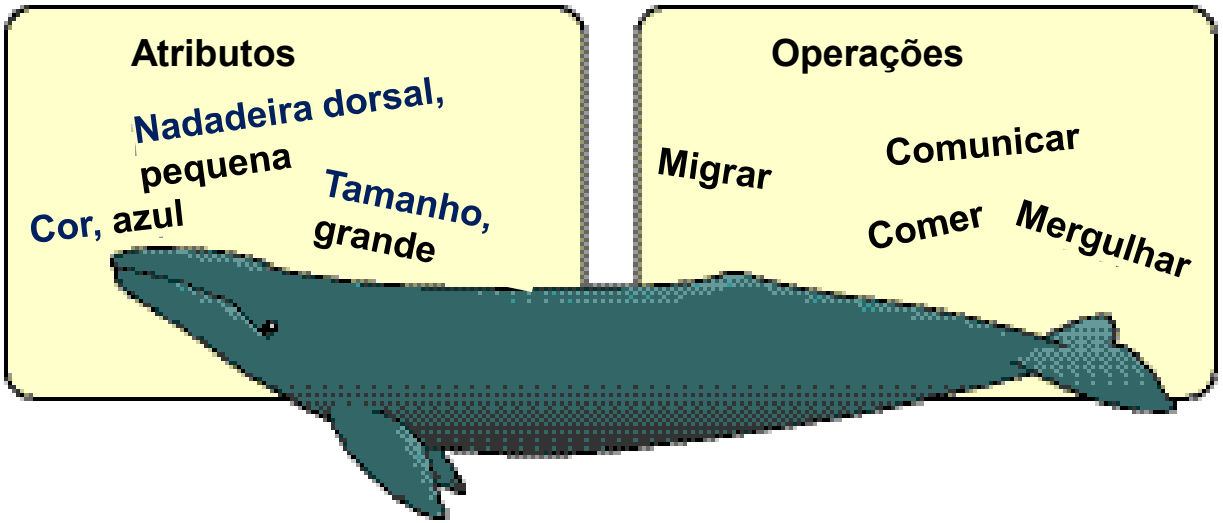
Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Identificando Objetos

- Os objetos podem ser físicos ou conceituais.
- Os objetos têm *atributos* (características) como tamanho, nome, forma etc.
- Os objetos têm *operações* (o que eles podem fazer), como definir um valor, exibir uma tela ou aumentar a velocidade.

Identificando Objetos



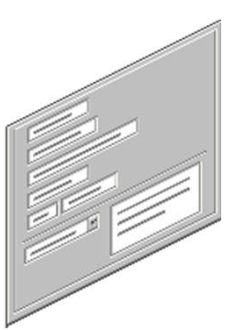
Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Critérios Adicionais para o Reconhecimento de Objetos

- Relevância para o domínio do problema:
  - O objeto existe dentro dos limites do domínio do problema?
  - O objeto é necessário para que a solução seja completa?
  - O objeto é necessário como parte de uma interação entre um usuário e o sistema?
- Existência independente

Objetos Possíveis
no Estudo de Caso da Lojas C&B



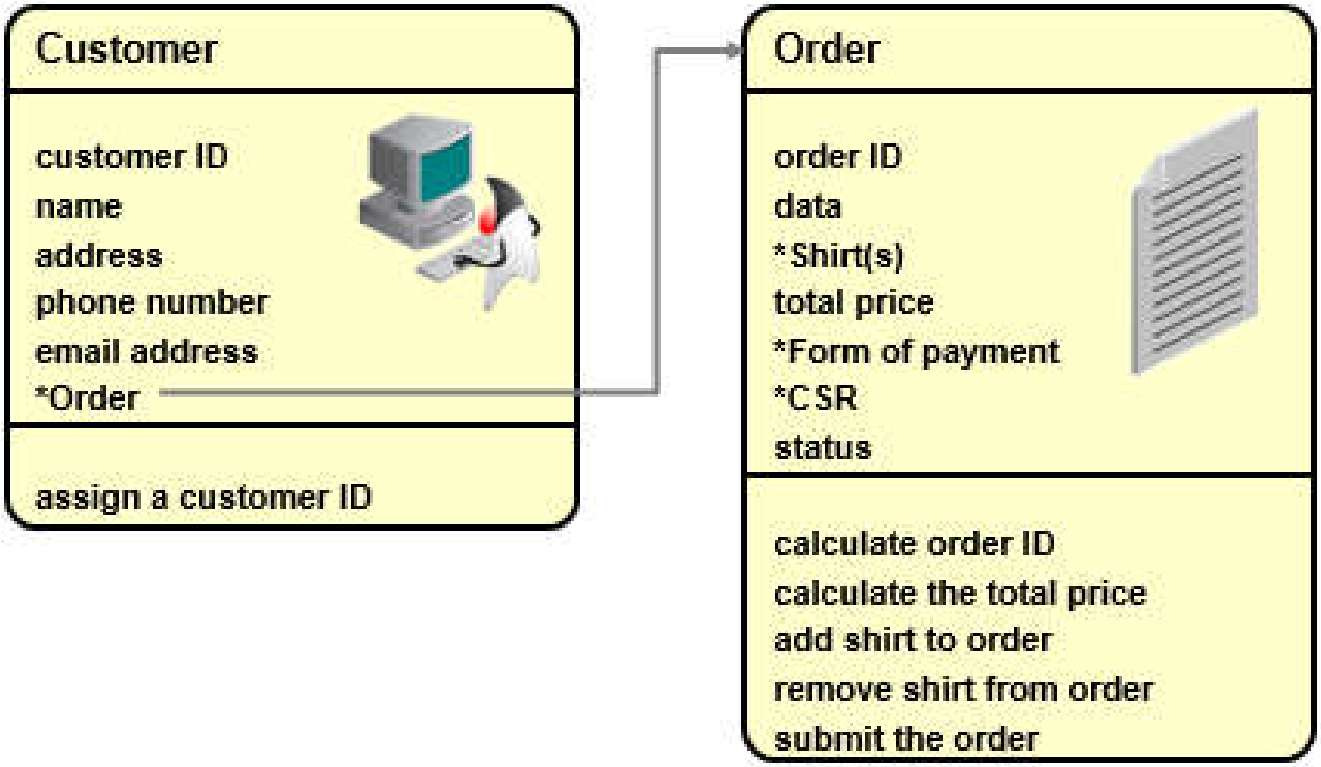
Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Identificando Atributos e Operações de Objetos

- Atributos são dados, como:
  - ID
  - Objeto Order
- Operações são ações, como:
  - Delete item
  - Change ID

Objeto com Outro Objeto como Atributo



Possíveis Atributos e Operações dos Objetos do Estudo de Caso da Lojas C&B

Order	Shirt	Customer
order ID data *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code	customer ID name address phone number email address *Order
calculate order ID calculate the total price add shirt to order remove shirt from order submit the order	calculate shirt ID display shirt information	assign a customer ID

Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Solução do Estudo de Caso: *Classes*

Classe	Order	Shirt	Customer	Form of Payment	Catalog	CSR

Solução do Estudo de Caso: *Atributos*

Classe	`Order`	`Shirt`	`Customer`
Atributos	order ID date *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code	customer ID name address phone number email address *Order

Solução do Estudo de Caso: *Atributos*

Classe	`Form of Payment`	`Catalog`	`CSR`
Atributos	customer ID name address phone number email address *Order	*Shirt(s)	name extension

Solução do Estudo de Caso: *Comportamentos*

Classe	`Order`	`Shirt`	`Customer`
Atributos	order ID date *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code	customer ID name address phone number email address *Order
Comportamentos	calculate order ID calculate the total price add shirt to order remove shirt from order submit the order	calculate shirt ID display shirt information	assign a customer ID

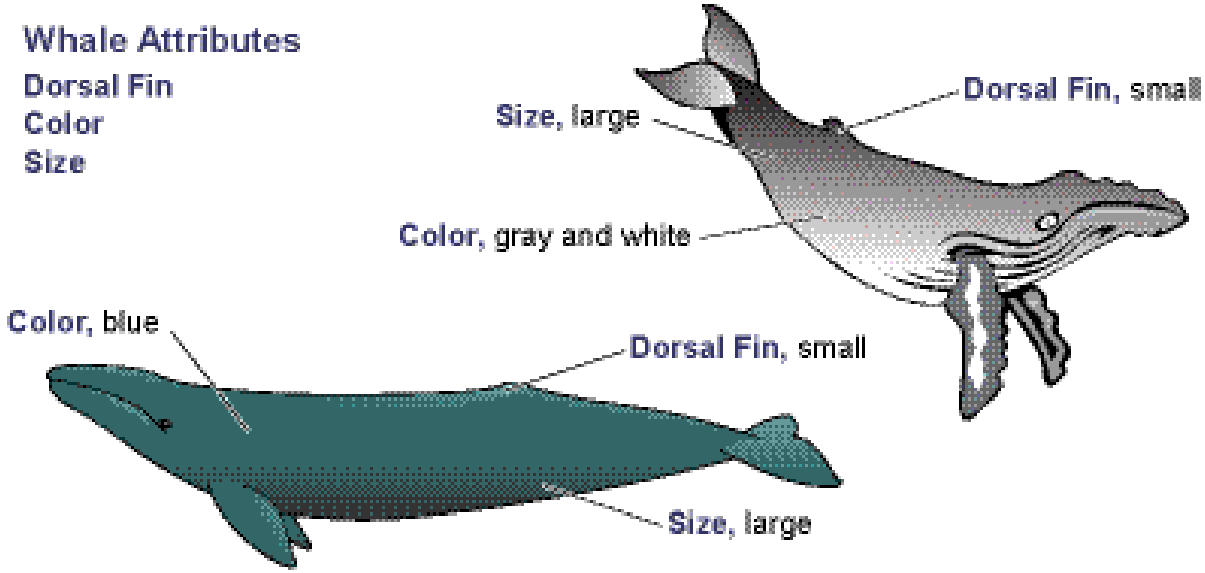
Solução do Estudo de Caso: *Comportamentos*

Classe	`Form of Payment`	`Catalog`	`CSR`
Atributos	customer ID name address phone number email address *Order	*Shirt(s)	name extension
Comportamentos	verify credit card number verify check payment	add a shirt remove a shirt	process order

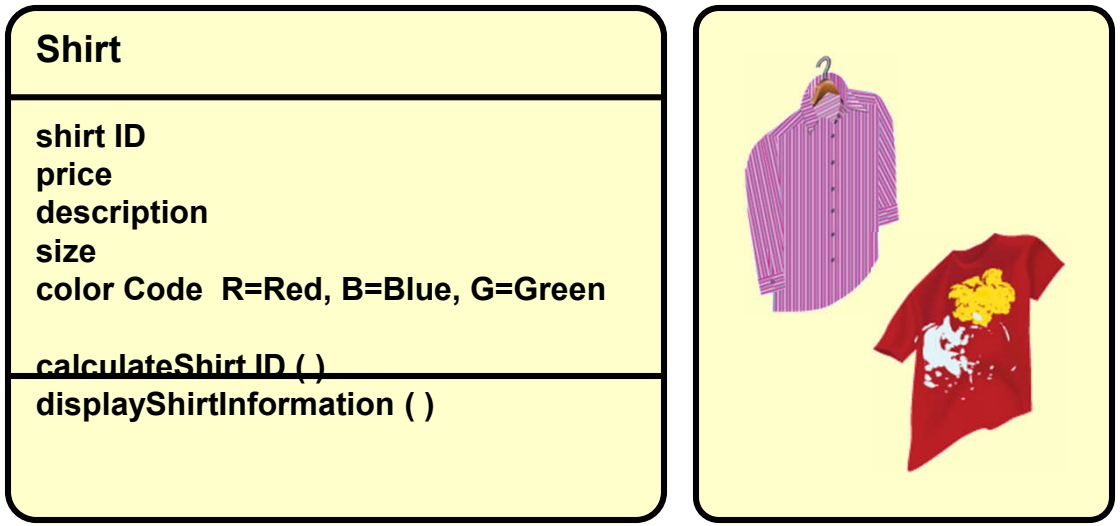
Tópicos

- Analisar um problema usando a análise orientada a objetos (OOA)
- Identificar o domínio de um problema
- Identificar os objetos
- Definir critérios adicionais para o reconhecimento de objetos
- Definir atributos e operações
- Discutir a solução de um estudo de caso
- Projetar e modelar uma classe

Projetando Classes



Classe e Objetos Resultantes

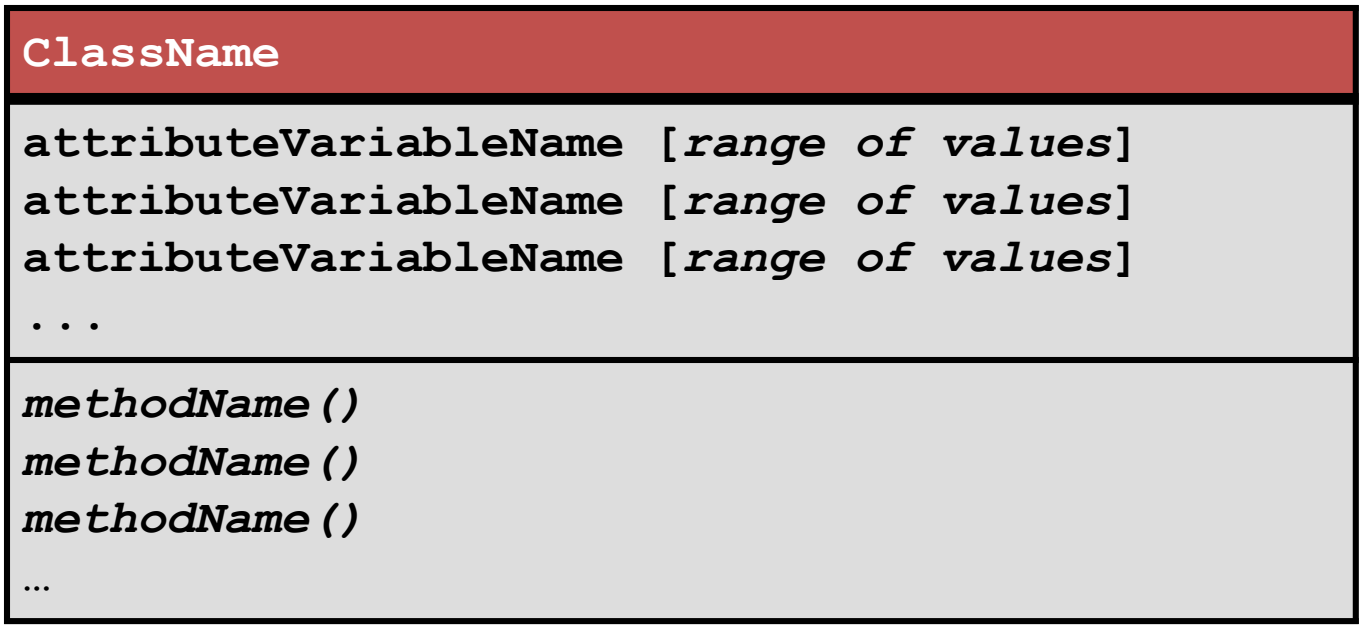


Classe Shirt

Objetos Shirt

Modelando Classes

- Sintaxe:



Modelando Classes

- Exemplo:

Shirt
shirtID price description size colorCode R=Red, B=Blue, G=Green
calculateShirtID() displayInformation()

Quiz

- Quais dos seguintes termos representam duas propriedades diferentes de um objeto?

a. Métodos e operações
b. O domínio de um problema
c. Atributos e operações
d. Variáveis e dados

Quiz

- Qual das afirmações abaixo é verdadeira?

a. Um objeto é um plano gráfico (blueprint) de uma classe.
b. Um objeto e uma classe são exatamente a mesma coisa.
c. Um objeto é uma instância de uma classe.
d. Um atributo não pode ser uma referência a outro objeto.

Visão Geral do Exercício 3.1: Analisando um Problema Usando a Análise Orientada a Objetos

1. Crie o projeto Conta bancária e codifique as classes Conta, TesteConta e TesteConta , preste atenção na codificação que será criada. Atenção para os nomes das classes e as palavras reservadas que foram usadas. Caso modifique os nomes das classes toda a codificação que depende desta classe deve possuir a referência da classe criada.

Visão Geral do Exercício 3.2: Projetando uma Solução

class Conta

```
1   public class Conta {
2
3   private int numero;
4   private double saldo=100;
5   private double juros=200;
6   private char vencimento;
7
8   public double getJuros (){
9       return juros;
10
11  }
12  public int getnumero (){
13      return numero;
14
15  }
16  public double getsaldo (){
17      return saldo;
18
19  }
20  public char getvencimento (){
21      return vencimento;
22          }
23
24  public void setnumero (int numero){
25      this.numero = numero;
26  }
27
28  public void setvencimento (char vencimento){
29      this.vencimento = vencimento;
30  }
31  public void debito (double valor){
32      this.saldo -=valor;
33  }
34  public void credito (double valor){
35      this.saldo +=valor;
36              } }
```

Visão Geral do Exercício 3.2: Projetando uma Solução

```
1
2  public class TesteConta {
3
4  public static void main (String args[]){
5      Conta conta1=new Conta ();
6      System.out.println ("Ref. conta1:"+ conta1);
7
8      Conta conta2 = new Conta ();
9      System.out.println ("Ref. conta2:"+conta2);
10
11
12 }
13 }
14
```

Visão Geral do Exercício 3.2: Projetando uma Solução

class TesteConta2

```
1
2  public class TesteConta2 {
3
4
5      public static void main (String args [])
6      {
7          Conta conta1 = new Conta();
8          System.out.println ("Ref. conta1:"+conta1);
9          conta1.setnumero(1);
10         conta1.credito (100);
11         conta1.debito (20);
12         System.out.println ("conta1:Numero:"+conta1.getnumero());
13         System.out.println ("Conta1:Saldo"+conta1.getsaldo());
14         System.out.println ();
15
16         Conta conta2 = new Conta ();
17         System.out.println ("Ref. Conta2:"+conta2);
18         conta2.setnumero(2);
19         conta2.credito (200);
20         conta2.debito (40);
21
22         System.out.println ("Conta2: Numero:"+conta2.getnumero());
23         System.out.println ("Conta2: Saldo:"+conta2.getsaldo());
24
25
26     }
27 }
```

Polimorfismo

- Existem 2 tipos:
  - Polimorfismo Estático ou Sobrecarga
  - Polimorfismo Dinâmico ou Sobreposição

Sobrecarga de método

- Os métodos com o *mesmo* nome podem ser declarados na mesma classe, contanto que tenham *diferentes* conjuntos de parâmetros (determinados pelo número, tipos e ordem dos parâmetros).

- A sobrecarga de métodos é comumente utilizada para criar vários métodos com o *mesmo* nome que realizam as *mesmas* tarefas, ou tarefas *semelhantes*, mas sobre tipos *diferentes* ou números *diferentes* de argumentos.

Exemplo Sobrecarga de método

```java
/**
 * MetodoSobrecarregado
 */
public class MetodoSobrecarregado {

    // teste dos metodos sobrecarregados
    Run | Debug
    public static void main(String[] args) {
        System.out.printf("O quadrado do número inteiro 7 é %d%n", quadrado(7));
        System.out.printf("O quadrado do número flutuante 7.5 é %.2f%n", quadrado(7.5));
    }

    //método quadrado com o argumento inteiro
    public static int quadrado(int intValor){
        System.out.printf("%n O quadrado do argumento do tipo inteiro %d%n", intValor);
        return intValor * intValor;
    }

    //método quadrado com o argumento double
    public static double quadrado(double doubleValor){
        System.out.printf("%n O quadrado do argumento do tipo double %.2f%n", doubleValor);
        return doubleValor * doubleValor;
    }
}
```

Exemplo Sobrecarga de método (cont.)

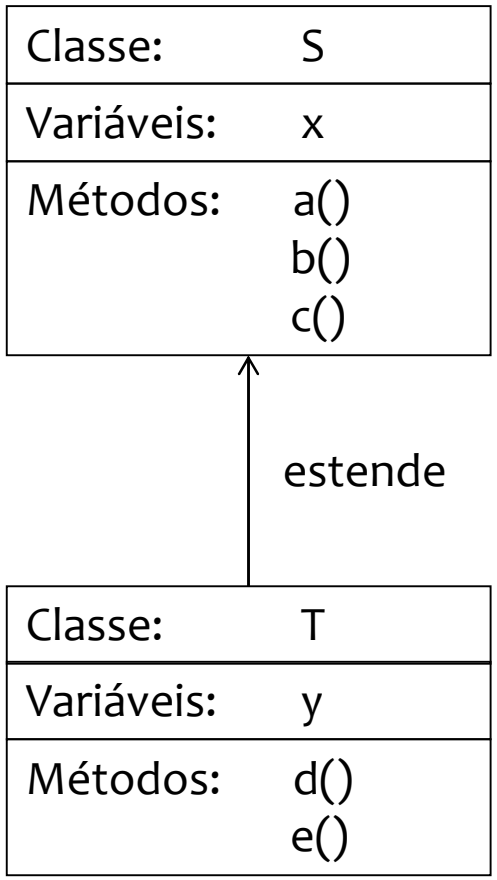
```
 O quadrado do argumento do tipo inteiro 7
O quadrado do número inteiro 7 é 49

 O quadrado do argumento do tipo double 7,50
O quadrado do número flutuante 7.5 é 56,25
```

Revisão de Herança (cont.)

- Por que usar herança? Para evitar código redundante
- A sub classe **estende** ou **herda** uma super classe
- Não necessita fornecer uma nova implementação para os métodos genéricos
- Deve apenas definir os métodos que são especializados para esta sub classe em particular

Revisão de Herança (cont.)

- T estende S
- T possui 2 variáveis: x e y
- E 5 métodos: a(), b(), c(), d() e e()

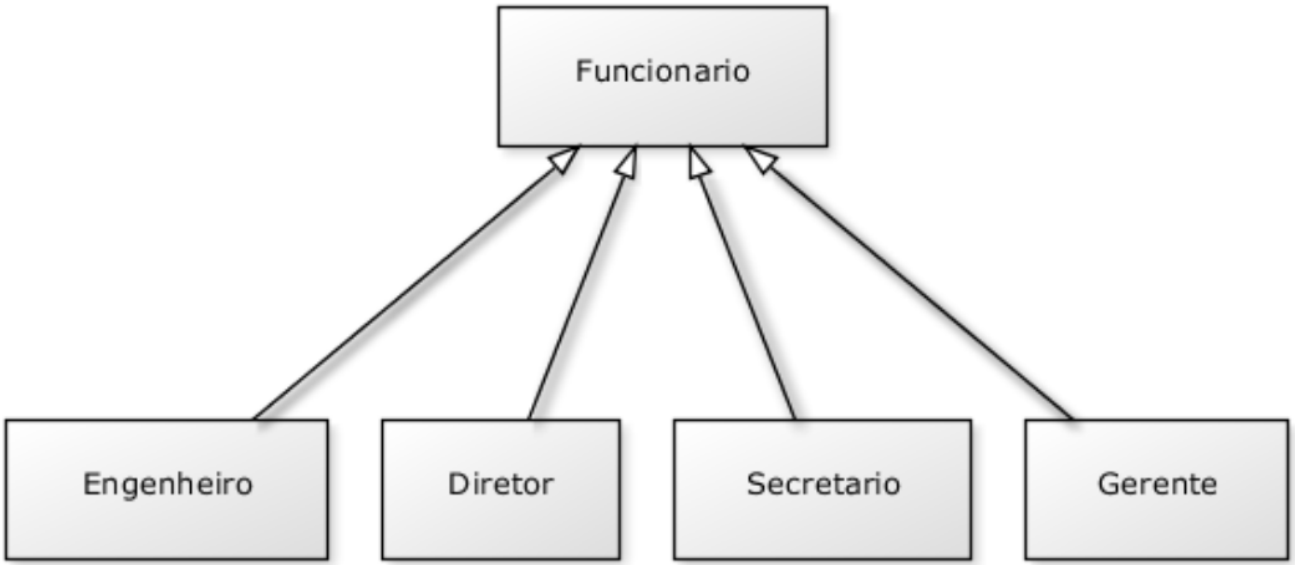


Superclasses e Subclasses

Superclasse	Subclasse
Formato	Circulo, Triangulo, Quadrado
Estudante	1º Grau, 2º Grau, Universitário
Veículo	Carro, Moto, Ônibus, Caminhão

Herança (cont.)

- Relacionamentos de herança formam estruturas *hierárquicas* na forma de árvores.
- Uma classe pode ter várias filhas, mas pode ter apenas uma mãe.



Public, Private e Protected

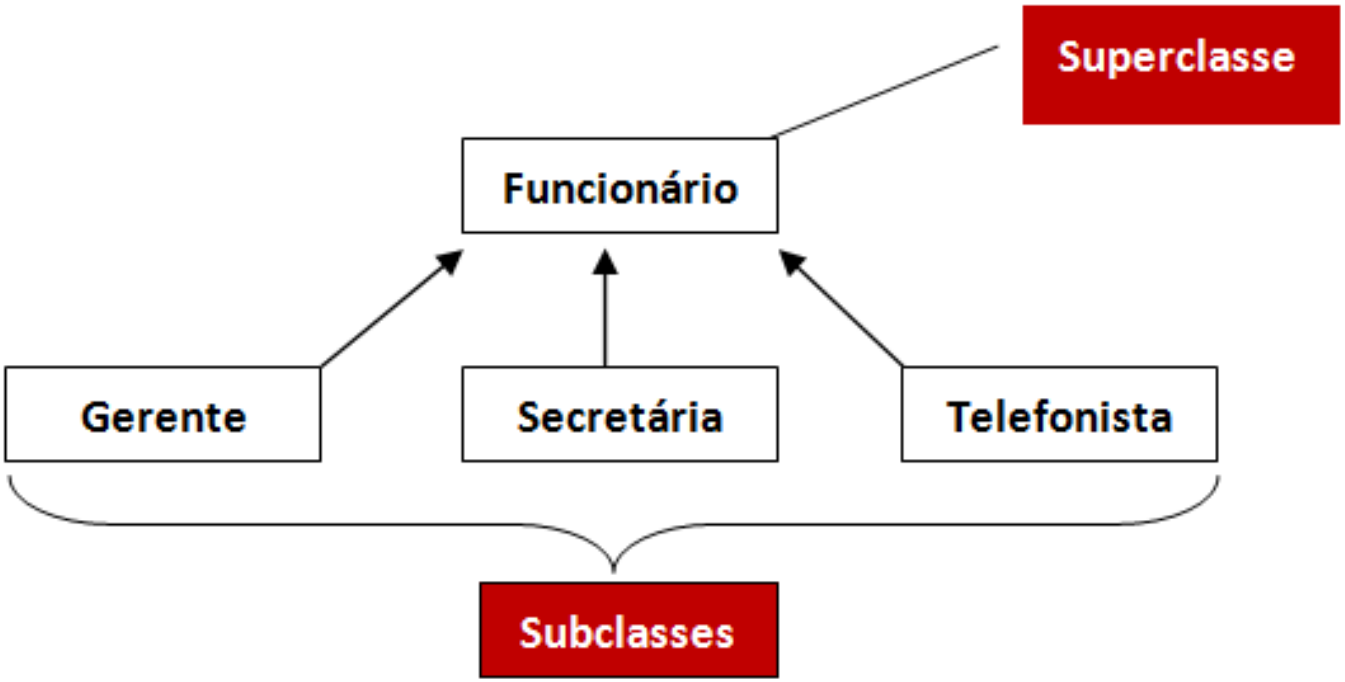
- Os membros **public** de uma classe são acessíveis onde quer que o programa tenha uma *referência* a um *objeto* dessa classe ou a uma de suas *subclasses*.
- Os membros **private** de uma classe só são acessíveis dentro da própria classe.
- Os membros **protected** só pode ser acessado (visível) pela própria classe, por suas subclasses, e pelas classes que se encontram no mesmo pacote.

Métodos setters e getters

Métodos getters	Métodos setters

public String getNome() {
return nome;
}
 	
public double getSalario() {
return salario;
}
	public void setNome(String nome) {
	this.nome = nome;
	}
	
	public void setSalario(double salario) {
	this.salario = salario;
	}

Exemplo


Superclasse
Funcionário
Gerente
Secretária
Telefonista
Subclasses

Exemplo (cont.)

```java
public class Funcionario {
    private String nome;
    private double salario;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public double calculaBonificacao(){
        return this.salario * 0.1;
    }

}
```

Exemplo (cont.)

```
1
2  public class Gerente extends Funcionario {
3      private String usuario;
4      private String senha;
5
6      public String getUsuario() {
7          return usuario;
8      }
9
10     public void setUsuario(String usuario) {
11         this.usuario = usuario;
12     }
13
14     public String getSenha() {
15         return senha;
16     }
17
18     public void setSenha(String senha) {
19         this.senha = senha;
20     }
21
22     public  double calculaBonificacao(){
23         return this.getSalario() * 0.6 + 100;
24     }
25 }
```

Sobrescrita

Exemplo (cont.)

```
1  public class Telefonista extends Funcionario {
2      private int estacaoDeTrabalho;
3
4      public void setEstacaoDeTrabalho(int estacaoDeTrabalho) {
5          this.estacaoDeTrabalho = estacaoDeTrabalho;
6      }
7
8      public int getEstacaoDeTrabalho() {
9          return estacaoDeTrabalho;
10     }
11 }
```

Exemplo (cont.)

```
1  public class Secretaria extends Funcionario {
2      private int ramal;
3
4      public void setRamal(int ramal) {
5          this.ramal = ramal;
6      }
7
8      public int getRamal() {
9          return ramal;
10     }
11 }
```

Exemplo (cont.)

```
1
2  public class TestaFuncionario {
3
4      public static void main(String[] args) {
5
6          Gerente gerente = new Gerente();
7          gerente.setNome("Carlos Vieira");
8          gerente.setSalario(3000.58);
9          gerente.setUsuario("carlos.vieira");
10         gerente.setSenha("5523");
11
12         Funcionario funcionario = new Funcionario();
13         funcionario.setNome("Pedro Castelo");
14         funcionario.setSalario(1500);
15
16         Telefonista telefonista = new Telefonista();
17         telefonista.setNome("Luana Brana");
18         telefonista.setSalario(1300.00);
19         telefonista.setEstacaoDeTrabalho(20);
20
21         Secretaria secretaria = new Secretaria();
22         secretaria.setNome("Maria Ribeiro");
23         secretaria.setSalario(1125.25);
24         secretaria.setRamal(5);
```

Exercício 1

- Criar uma classe Funcionário com os seguintes atributos: nome, CPF e salário.
- Criar uma classe Gerente com os seguintes atributos: nome, CPF, salário, senha e número de funcionários gerenciados e com o método: validar senha.
- Criar uma classe TestaGerente com os métodos: setNome e setSenha.

Exercício 2

- Incluir um novo método bonificação. Esse método representa uma bonificação que todos os funcionários recebem no fim do ano e é referente a 10% do valor do salário. Porém, o gerente recebe uma bonificação de 15%.