



Programação Orientada a Objetos Utilizando Java

Professor MSc. Odair Jacinto da Silva
odair.silva@unimetrocamp.edu.br

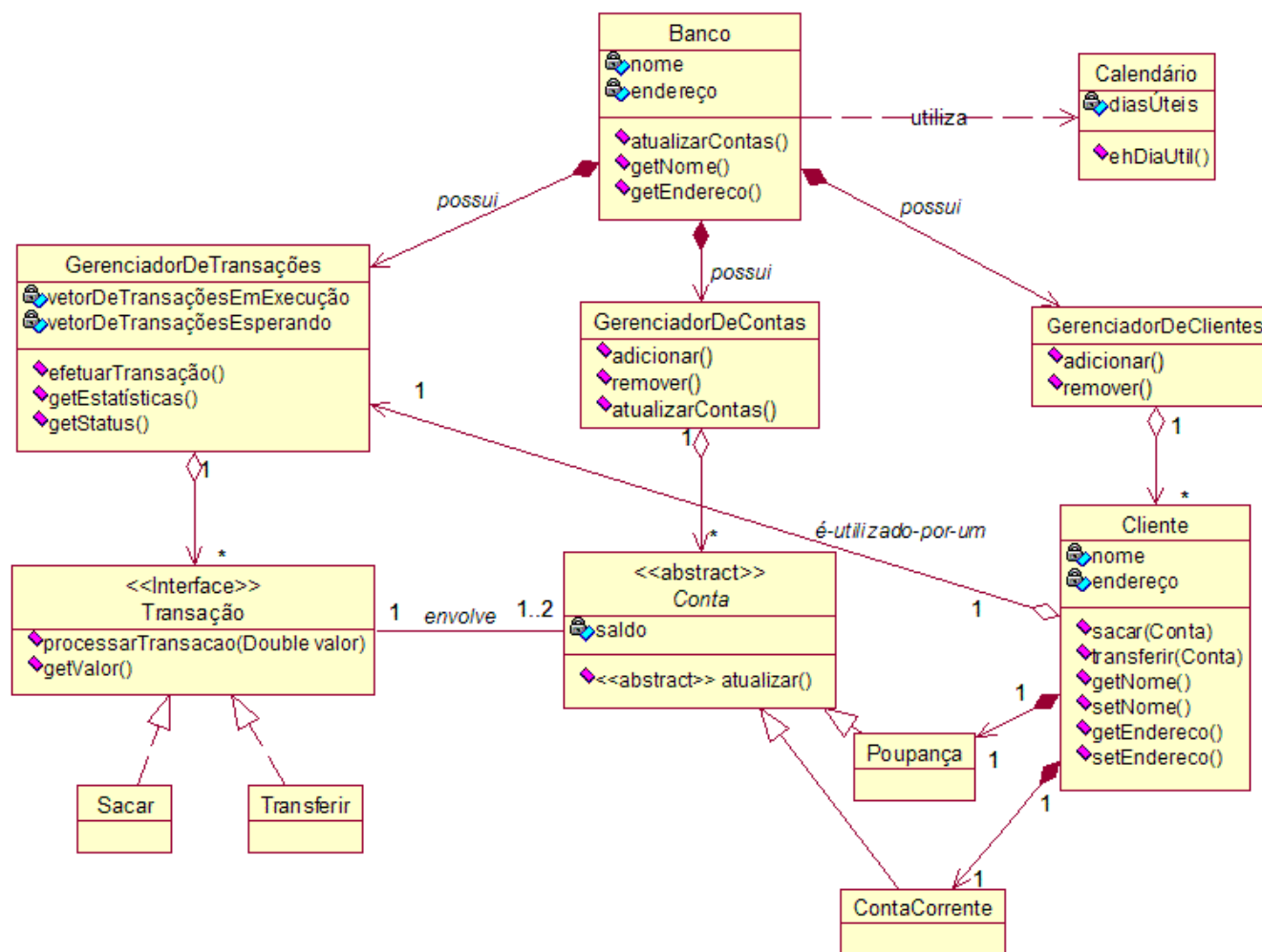
Motivação

- Programas eram lineares e com poucos módulos (programação estruturada)
- Aumento da complexidade dos sistemas e difícil reusabilidade dos mesmos
- Criação de um **novo Paradigma** de Análise e Desenvolvimento de Sistemas: Programação Orientada a Objetos

Motivação

- No mundo real, pensamos em conceitos e em entidades concretas e abstratas
- Tudo é objeto:
 - Ex.: carro, computador, música, camisa, cliente, conta bancária, etc

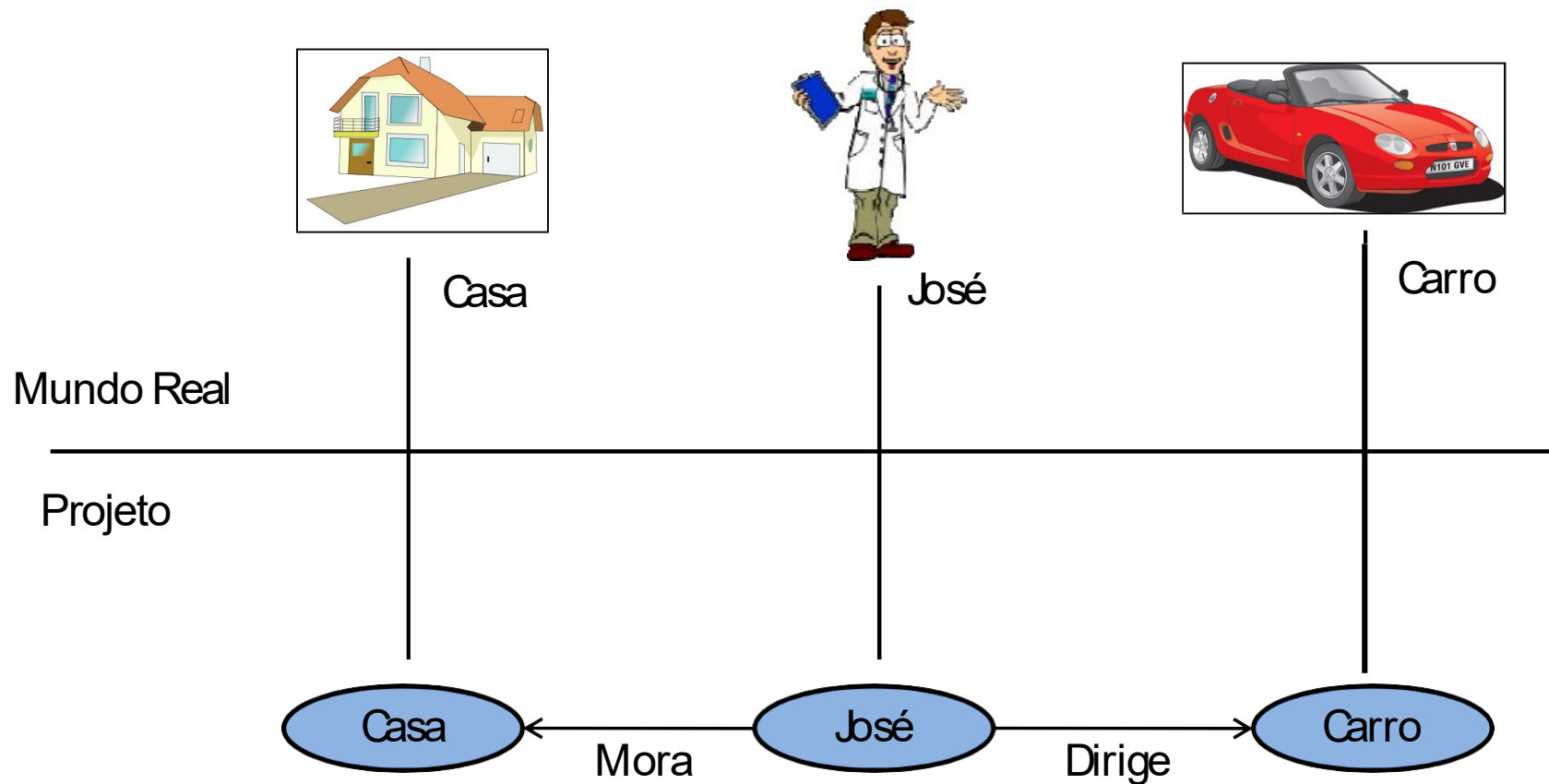
Diagrama de Classes (UML)



Programação Orientada a Objetos

- Paradigma de Programação
 - Dominante nos dias atuais
- Substituiu as técnicas de programação procedimental (estruturada)
- “Fornece um mapeamento direto entre o mundo real e as unidades de organização utilizadas no projeto”
- Diversas unidades de software, chamadas de objetos, que interagem entre si

Programação Orientada a Objetos



Programação Orientada a Objetos

- Vantagens:
 - Flexibilidade
 - Reusabilidade
 - Robustez
 - Modularidade

Programação Orientada a Objetos

- Elementos básicos:
 - Classes
 - Objetos (e Instâncias dos Objetos)

Objetos

- Entidades concretas ou abstratas
- Tem características e podem executar ações
- “um objeto representa um item identificável, uma unidade ou entidade, individual, seja real ou abstrato, com uma regra bem definida”

OBJETO = DADOS + OPERAÇÕES

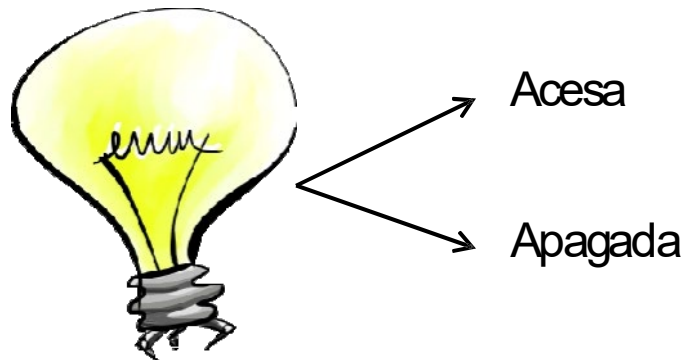
- Possuem:
 - Estado
 - Comportamento
 - Identidade

Objetos

- Estado:
 - Define os estados possíveis que um objeto pode assumir
 - São os valores dos atributos (propriedades)

- Ex.:

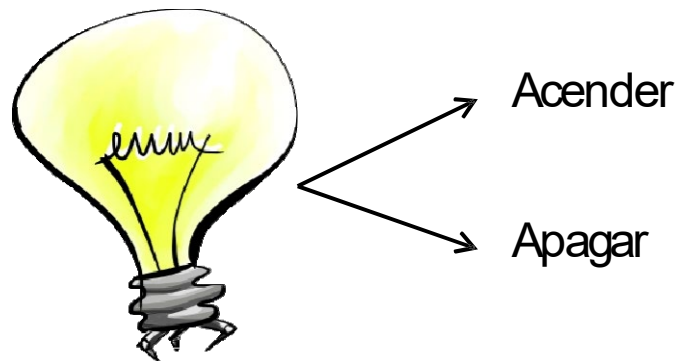
- Lâmpada



Objetos

- Comportamento:
 - São as funções que podem ser executadas por um determinado objeto
 - Corresponde aos métodos
 - O que você pode fazer com um determinado objeto

- Ex.:
 - Lâmpada



Objetos

- Identidade:
 - Um objeto é único, mesmo que o seu estado seja idêntico ao de outro

- Ex.:
 - Lâmpada



→ Cód. De Fabricação 001; Incandescente;

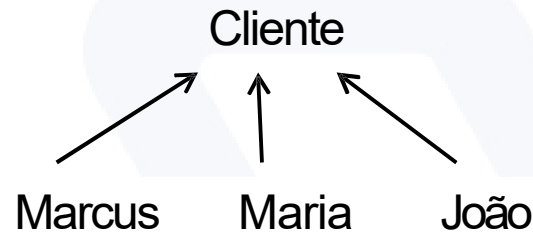
→ Cód. De Fabricação 002; Incandescente;

Classes

- Modelo ou esquema a partir do qual os objetos são criados (instanciados)
- Modelam os objetos definindo:
 - Tipo de dados que o objeto armazena, ou seja, os estados possíveis que ele pode assumir (atributos)
 - Tipos de operações que podem ser executadas pelo objeto, ou seja, o seu comportamento (métodos)
- Abstração de objetos de características semelhantes (molde)
- Essência do objeto

Classes

- Objetos são instâncias de classes



- Lembrando: Todo código Java está dentro de uma Classe
- A biblioteca Java padrão fornece milhares de classes para vários propósitos

Classes

- Declaração de uma classe em Java:

```
[<modificadores da classe>] class <nome_classe> [extends  
    <nome_superclasse>]  
    [implements <interface_1>, <interface_2>, ...] {  
  
    // Variáveis e métodos da classe  
  
}
```

[] = Opcionais

<> = Identificadores e palavras reservadas

Classes

- Exemplo de declarações de classe:

```
class Lampada {
```

```
// Variáveis  
// Métodos
```

```
}
```

```
public class Lampada {
```

```
// Variáveis  
// Métodos
```

```
}
```


Atributos

- Definem as características do objeto

```
[<modificadores_atributo>] <tipo_atributo> <nome_atributo> [= valor_inicial];
```

[] = Opcionais

<> = Identificadores e palavras reservadas

Atributos

- Exemplo:

```
public boolean estadoLampada =false;  
double valor;  
String tipo =“fluorescente”;
```

Métodos

- Definem as ações que um objeto pode executar
- Sua definição corresponde a duas partes:
 - Assinatura
 - Corpo

```
[<modificadores_método>] <tipo_retorno> <nome_método> ([<parametros>]){  
  
    • // Corpo do Método  
  
}
```

- [] = Opcionais
- <> = Identificadores e palavras reservadas

Métodos

```
public void acenderLampada(){  
    estadoLampada = true;  
}
```

```
public int somar(int a, int b){  
    int resultado = a + b;  
    return resultado;  
}
```

Métodos

- Exemplo Completo:

```
public class Lampada{  
  
    public boolean estadoLampada =false;  
  
    public void acenderLampada(){  
        estadoLampada =true;  
    }  
  
    public void apagarLampada(){  
        estadoLampada =false;  
    }  
  
    public boolean verEstadoLampada(){  
        return estadoLampada;  
    }  
  
}
```

Exemplo 1

```
Calculadora.java  testeCalculadora.java
1
2 public class Calculadora {
3     double n1;
4     double n2;
5
6     double soma() {
7         return n1+n2;
8     }
9
10    double produto() {
11        return n1*n2;
12    }
13
14    double potencia() {
15        return Math.pow(n1,n2);
16    }
17 }
18
```

Problems @ Javadoc Declaration Console Coverage

<terminated> testeCalculadora [Java Application] C:\Program Files\Java\jre1.8.0_...

Soma de 8.0 + 2.0 = 10.0
Produto de 8.0 * 2.0 = 16.0
Potencia de 8.0 ^ 2.0 = 64.0

```
Calculadora.java  testeCalculadora.java
1
2 public class testeCalculadora {
3
4     public static void main(String[] args) {
5         Calculadora c = new Calculadora();
6         c.n1=8;
7         c.n2=2;
8
9         System.out.println("Soma de "+c.n1+" + "+c.n2+" = "+c.soma());
10        System.out.println("Produto de "+c.n1+" * "+c.n2+" = "+c.produto());
11        System.out.println("Potencia de "+c.n1+" ^ "+c.n2+" = "+c.potencia());
12
13    }
14
15 }
16
```

Exemplo 2

```
Conta.java  testeConta.java
1
2 public class Conta {
3     String banco;
4     String numero;
5     double saldo;
6     double limite;
7
8     public void extrato() {
9         System.out.println("#####");
10        System.out.println("Extrato da Conta");
11        System.out.println("Banco:"+banco);
12        System.out.println("Numero:"+numero);
13        System.out.println("Saldo R$ "+saldo);
14        System.out.println("Limite R$"+limite);
15        System.out.println("#####");
16    }
17
18    public void depositar(double valor) {
19        saldo+=valor;
20    }
21
22    public void sacar(double valor){
23        if(saldo<valor)
24            System.out.println("Saldo insuficiente.");
25        else {
26            saldo-=valor;
27            System.out.println("Saque realizado com sucesso.");
28        }
29    }
30 }
```

Exemplo 2

```
Conta.java  testeConta.java ✕
1
2 public class testeConta {
3
4 public static void main(String[] args) {
5     Conta c1 = new Conta();
6     Conta c2 = new Conta();
7
8     c1.banco="Itau";
9     c1.numero="1234-5";
10    c1.saldo=1000.00;
11    c1.limite=0.00;
12
13    c2.banco="NuBank";
14    c2.numero="0001-1";
15    c2.saldo=500.00;
16    c2.limite=100.00;
17
18    c1.extrato();
19    c2.extrato();
20
21 }
22
23 }
24
```

```
Problems  @ Javadoc  Declaration  Console  Coverage
<terminated> testeConta [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Feb 25, 2019, 9:41:42 AM)
#####
Extrato da Conta
Banco:Itau
Numero:1234-5
Saldo R$ 1000.0
Limite R$0.0
#####
#####
Extrato da Conta
Banco:NuBank
Numero:0001-1
Saldo R$ 500.0
Limite R$100.0
#####
```


Exemplo 3

```
Calculadora.java  testeCalculadora.java  Conta.java  testeConta.java  *testeTodas.java ✕
1 public class testeTodas {
2
3     public static void main(String[] args) {
4         Calculadora c = new Calculadora();
5         Conta cc1 = new Conta();
6         Conta cc2 = new Conta();
7
8         cc1.banco="Itau";
9         cc1.numero="1234-5";
10        cc1.saldo=1000.00;
11        cc1.limite=0.00;
12
13        cc2.banco="NuBank";
14        cc2.numero="0001-1";
15        cc2.saldo=500.00;
16        cc2.limite=100.00;
17
18        c.n1=cc1.saldo;
19        c.n2=cc2.saldo;
20        System.out.println("Saldo das duas contas R$ "+c.soma());
21
22    }
23
24 }
25
```

Exercícios

1. Qual a motivação para a utilização do paradigma orientado a objetos na programação de software?
2. Definir classe.
3. Definir objetos.
4. Definir atributos e métodos.

Revendo os conceitos

- **Classes:** As classes de programação são projetos de um objeto, aonde têm características e comportamentos, ou seja, permite armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja “abstrato”.

Revendo os conceitos

- Características das classes:
 - Toda classe possui um nome;
 - Possuem visibilidade, exemplo: public, private, protected;
 - Possuem membros como: Características e Ações;
 - Para criar uma classe basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves { }.

Revendo os conceitos

```
public class Teste{  
    //ATRIBUTOS OU PROPRIEDADES  
    //MÉTODOS  
}
```

Revendo os conceitos

```
public class Caes {  
  
    public String nome;  
    public int peso;  
    public String corOlhos;  
    public int quantPatas;  
  
    public void falar(){  
        //MÉTODO FALAR  
    }  
  
    public void andar(){  
        //MÉTODO ANDAR  
    }  
  
    public void comer(){  
        //MÉTODO COMER  
    }  
  
    public void dormir(){  
        //MÉTODO DORMIR  
    }  
}
```

Revendo os conceitos

- **Objetos:** Os objetos são características definidas pelas classes.
- **Atributos:** Os atributos são as propriedades de um objeto, também são conhecidos como variáveis ou campos. Essas propriedades definem o estado de um objeto, fazendo com que esses valores possam sofrer alterações.

Revendo os conceitos

```
public class TestaCaes {  
  
    public static void main(String[] args) {  
        Cachorro cachorro1 = new Cachorro();  
        cachorro1.nome = "Pluto";  
        cachorro1.corOlhos = "azuis";  
        cachorro1.peso = 53;  
        cachorro1.quantPatas = 4;  
  
        Cachorro cachorro2 = new Cachorro();  
        cachorro2.nome = "Rex";  
        cachorro2.corOlhos = "amarelo";  
        cachorro2.peso = 22;  
        cachorro2.quantPatas = 3;  
  
        Cachorro cachorro3 = new Cachorro();  
        cachorro3.nome = "Bob";  
        cachorro3.corOlhos = "marrom";  
        cachorro3.peso = 13;  
        cachorro3.quantPatas = 4;  
  
    }  
}
```


Revendo os conceitos

- **Métodos:** Os métodos são ações ou procedimentos, onde podem interagir e se comunicarem com outros objetos. A execução dessas ações se dá através de mensagens, tendo como função o envio de uma solicitação ao objeto para que seja efetuada a rotina desejada.

Revendo os conceitos

```
class Cachorro{  
    int tamanho;  
    String nome;  
  
    void latir(){  
        if(tamanho > 60)  
            System.out.println("Woof, Woof!");  
        else if(tamanho > 14)  
            System.out.println("Ruff!, Ruff!");  
        else  
            System.out.println("Yip!, Yip!");  
    }  
}
```

Revendo os conceitos

```
public class Testa_Cachorro {  
  
    public static void main(String[] args) {  
  
        Cachorro bob = new Cachorro();  
        bob.tamanho = 70;  
        Cachorro rex = new Cachorro();  
        rex.tamanho = 8;  
        Cachorro scooby = new Cachorro();  
        scooby.tamanho = 35;  
  
        bob.latir();  
        rex.latir();  
        scooby.latir();  
  
    }  
}
```

Exercícios

- Defina, com seus atributos e métodos uma classe para representar um(a):
 1. Aluno.
 2. Retângulo.
 3. Ponto.
 4. Bomba combustível.
 5. Disciplina na faculdade.
 6. Elevador.

Modificadores de Acesso

- Os **modificadores de acesso** são padrões de visibilidade de acessos às **classes, atributos e métodos**.
- Esses modificadores são palavras-chaves reservadas pelo **Java**, ou seja, palavras reservadas não podem ser usadas como nome de métodos, classes ou atributos.
- Como boas práticas do **Java**, na maioria das declarações de variáveis de instância são definidos os seus atributos com a palavra-chave **private**, para garantir a segurança de alterações acidentais, sendo somente acessíveis através dos métodos.
- Essa ação tem como efeito ajudar no encapsulamento dos dados, preservando ainda mais a segurança e a aplicação de programação orientada a objetos do Java.

Modificadores de Acesso

- **public**
 - Uma declaração com o modificador public pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.
- **private**
 - Os membros da classe definidos como não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.
- **protected**
 - O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.
- **default (padrão):**
 - A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador.

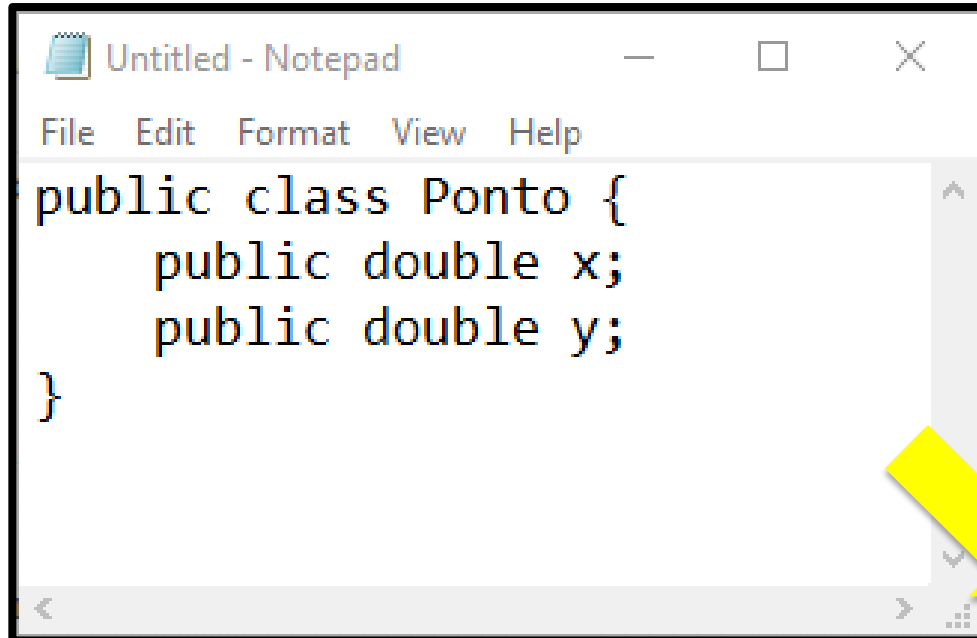
Tabela dos Modificadores de Acesso

	private	default	protected	public
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasses)	não	não	sim	sim
pacotes diferentes (sem subclasses)	não	não	não	sim

Métodos Get/Set

- Quando temos uma classe pública com seus métodos sendo diretamente acessados, dizemos que ela não oferece os benefícios do **encapsulamento**.
- O **encapsulamento** nos oferece a ideia de tornar o software mais flexível, fácil de modificar e de criar novas implementações.
- O **encapsulamento** oferece um controle de acesso aos atributos e métodos de uma classe.
- É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.

Métodos Get/Set



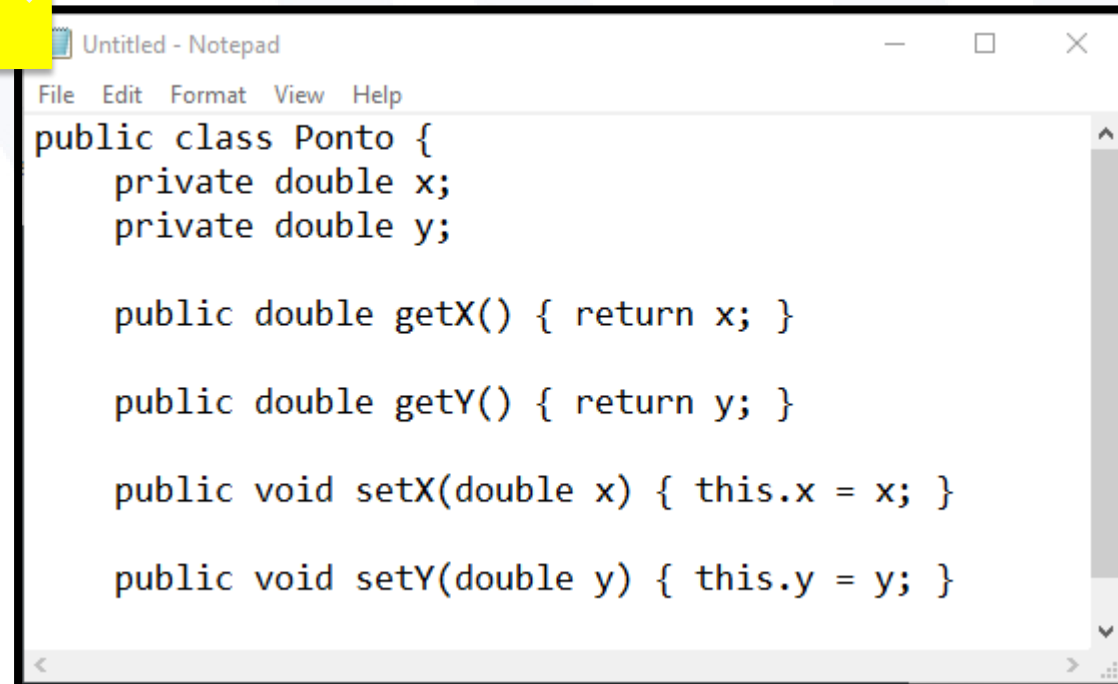
Untitled - Notepad

File Edit Format View Help

```
public class Ponto {  
    public double x;  
    public double y;  
}
```

A yellow arrow points from the bottom right corner of this window to the top left corner of the window below it.

**IDE Eclipse utilizar:
Menu Source->
Generate Getters and Setters**



Untitled - Notepad

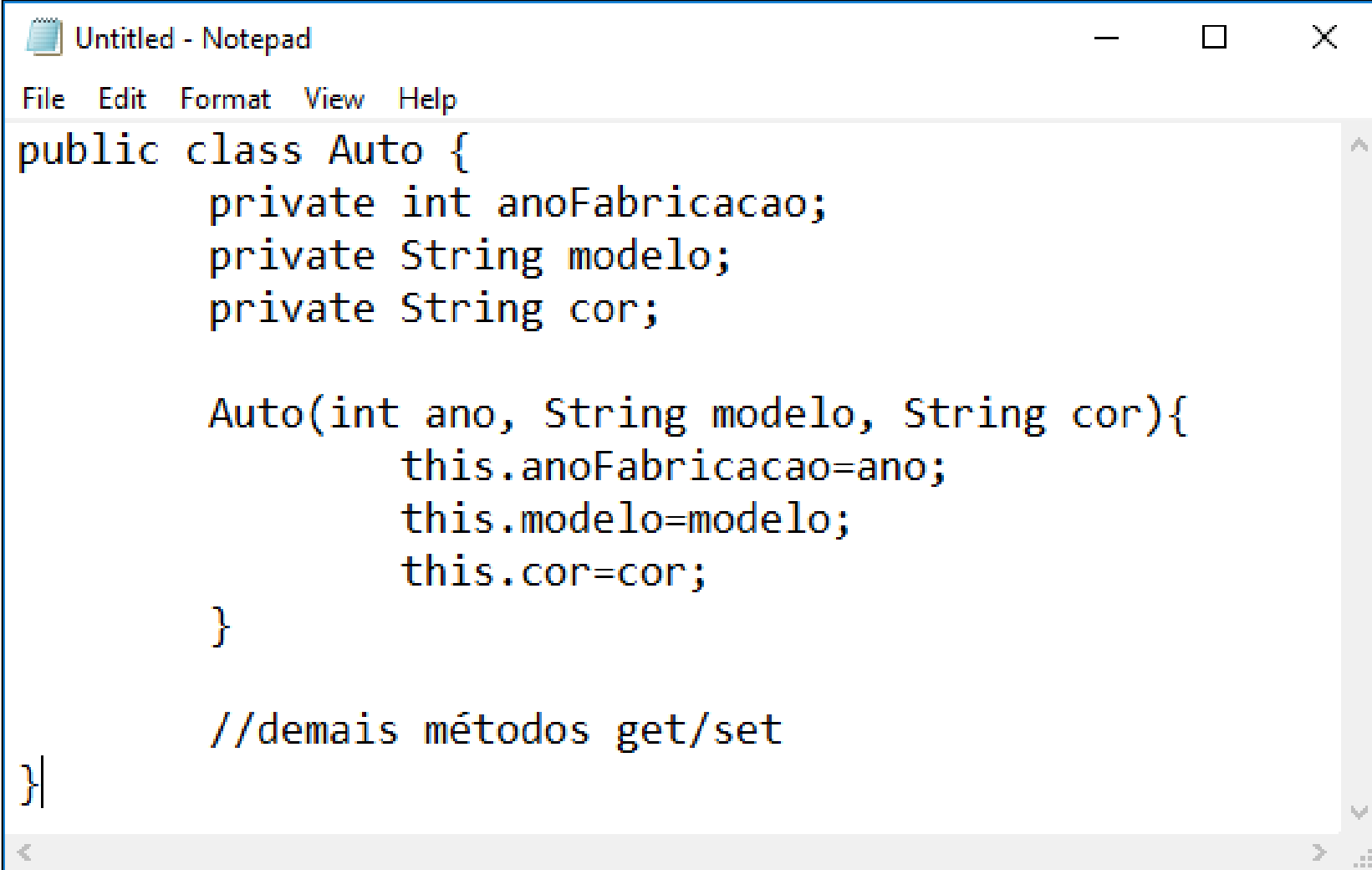
File Edit Format View Help

```
public class Ponto {  
    private double x;  
    private double y;  
  
    public double getX() { return x; }  
  
    public double getY() { return y; }  
  
    public void setX(double x) { this.x = x; }  
  
    public void setY(double y) { this.y = y; }  
}
```

Construtor

- Toda classe tem um construtor: operação declarada com o mesmo nome da classe, que não retorna valor e só pode ser usada na inicialização;
- Se um construtor não é explicitamente declarado em uma classe, o sistema cria um construtor default para a classe;

Construtor



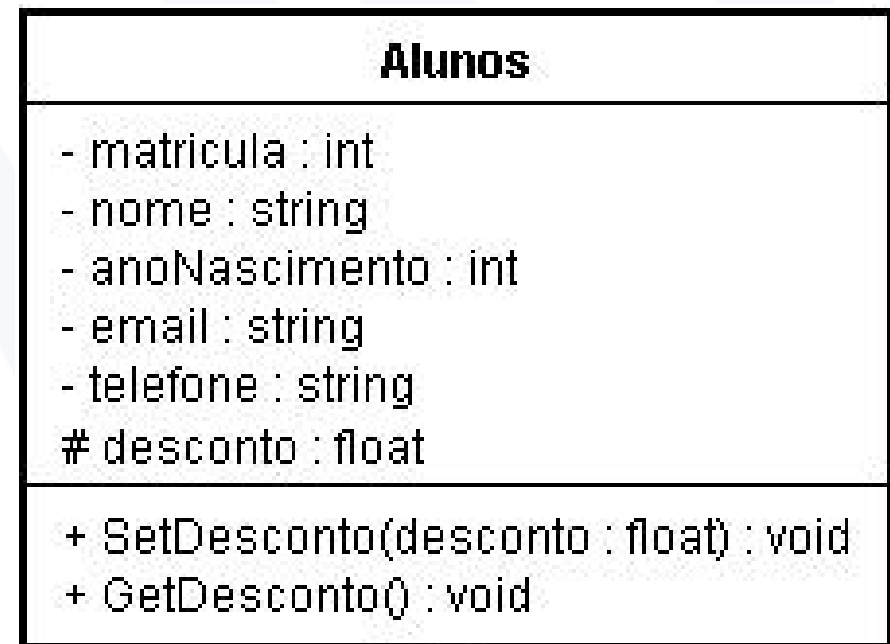
```
Untitled - Notepad
File Edit Format View Help
public class Auto {
    private int anoFabricacao;
    private String modelo;
    private String cor;

    Auto(int ano, String modelo, String cor){
        this.anoFabricacao=ano;
        this.modelo=modelo;
        this.cor=cor;
    }

    //demais métodos get/set
}
```

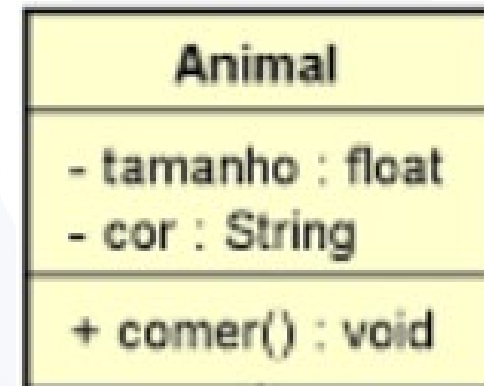
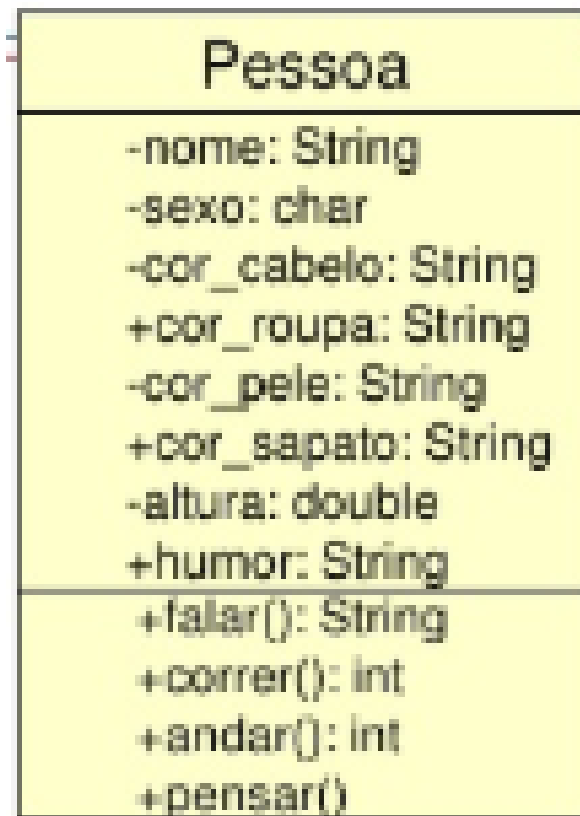
Notação UML

- - : private
- + : public
- # : protected



Exercício

- Implementar as seguintes classes:



Exercício

- Implementar a classe abaixo, incluindo um construtor para todos os seus atributos e os métodos get/set.



Exercício

- Implementar uma **classe Calculadora**, com dois atributos privados inteiros, um construtor para seus atributos e os respectivos métodos get/set. Implementar também os métodos `adicao()`, que retorna a soma dos valores dos atributos, além dos métodos `subtracao()`, `produto()` e `divisao()`, que realizam as operações indicadas pelos seus respectivos nomes.
- Faça um programa que teste, de forma completa, o uso desta classe.

Calculadora.java x tstCalculadora.java

```
1 public class Calculadora {
2     private float valor1;
3     private float valor2;
4
5     Calculadora(float valor1, float valor2){
6         this.valor1=valor1;
7         this.valor2=valor2; }
8
9     public float getValor1() {
10         return valor1; }
11
12     public void setValor1(float valor1) {
13         this.valor1 = valor1; }
14
15     public float getValor2() {
16         return valor2; }
17
18     public void setValor2(float valor2) {
19         this.valor2 = valor2; }
20
21     public float somar() {
22         return this.valor1+this.valor2; }
23
24     public float multiplicar() {
25         return this.valor1*this.valor2; }
26
27     public float subtrair() {
28         return this.valor1-this.valor2; }
29
30     public float dividir() {
31         return this.valor1/this.valor2; }
32 }
```

Calculadora.java

tstCalculadora.java x

```
1 public class tstCalculadora {
2
3     public static void main(String[] args) {
4         Calculadora c1 = new Calculadora(5,5);
5
6         System.out.println("Soma: "+c1.somar());
7         System.out.println("Subtracao: "+c1.subtrair());
8         System.out.println("Produto: "+c1.multiplicar());
9         System.out.println("Divisao: "+c1.dividir());
10
11         c1.setValor1(10);
12         c1.setValor2(-2);
13
14         System.out.println("Soma: "+c1.somar());
15         System.out.println("Subtracao: "+c1.subtrair());
16         System.out.println("Produto: "+c1.multiplicar());
17         System.out.println("Divisao: "+c1.dividir());
18     }
19 }
```


Exercício

- Implemente a seguinte classe e um programa para testá-la. O método `analisaIMC()` deve retornar uma expressão que indica o resultado da análise do IMC, por exemplo “Abaixo do peso”. Consulte na Internet ou nos slides desta aula como fazer esta análise. O método `calcIMC()` retorna o índice do IMC, com base na altura e no peso da pessoa. Ele é utilizado por `analisaIMC()` para obter o índice e então retornar a análise do IMC.

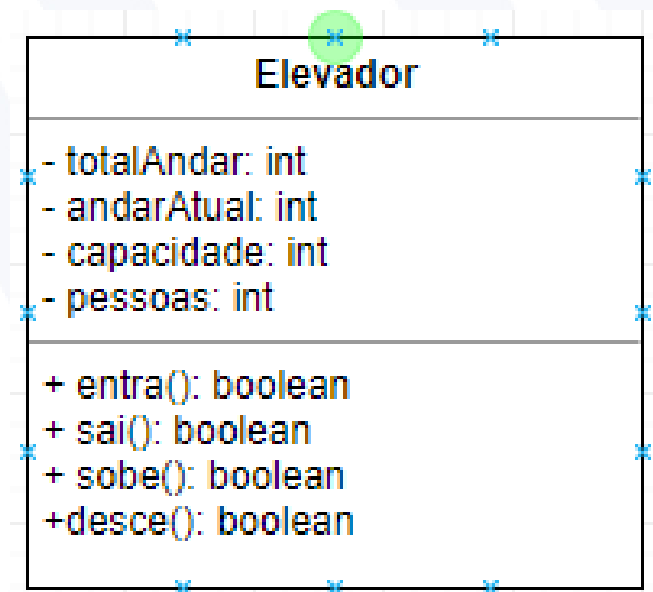
Humano
-nome:String -idade:int -altura:double -peso:double -sexo:String
+Humano(String, int, double, double, String) +setAltura(double) +setPeso(double) +analisaIMC():String -calcIMC():double

Exercício

- Crie uma classe (e o programa para testá-la) denominada **Elevador** para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual (térreo = 1), total de andares no prédio, capacidade do elevador, e quantas pessoas estão presentes nele. O objeto deve ser inicializado, por um construtor, com a capacidade do elevador e o total de andares no prédio (os elevadores sempre começam no térreo e vazio);

A classe deve também disponibilizar os seguintes métodos:

- **Entra:** para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço);
- **Sai:** para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);
- **Sobe:** para subir um andar (não deve subir se já estiver no último andar); informar que chegou no andar indicado.
- **Desce:** para descer um andar (não deve descer se já estiver no térreo); informar que chegou no andar indicado.



Exercício

- Implente a classe a seguir e um programa para testá-la.

Carro	
- capacidadeTanque: float	
- volumeTanque: float	
- rendimento: float	
+ Carro(float capacidade, float volume, float rendimento)	
+ encheTanque(float litros)	
+ passeio(float km)	

- Observações no próximo slide.

Exercício

- Sobre a classe Carro:
 - Um carro deve ser criado pelo construtor da classe.
 - Não são necessários gets/sets.
 - Ao encher um tanque é necessário, antes, verificar se existe espaço para a quantidade de litros que se deseja abastecer.
 - Se for possível abastecer, então atualizar o volume no tanque.
 - Ao realizar um passeio é necessário, antes, verificar se o volume de combustível existente permite o deslocamento informado, considerando o rendimento do carro.
 - Informar o volume no tanque após a realização do passeio, bem como atualizar este atributo.
 - Exemplos de saída no próximo slide.

Exercício

- Exemplos saída (teste) da classe Carro:

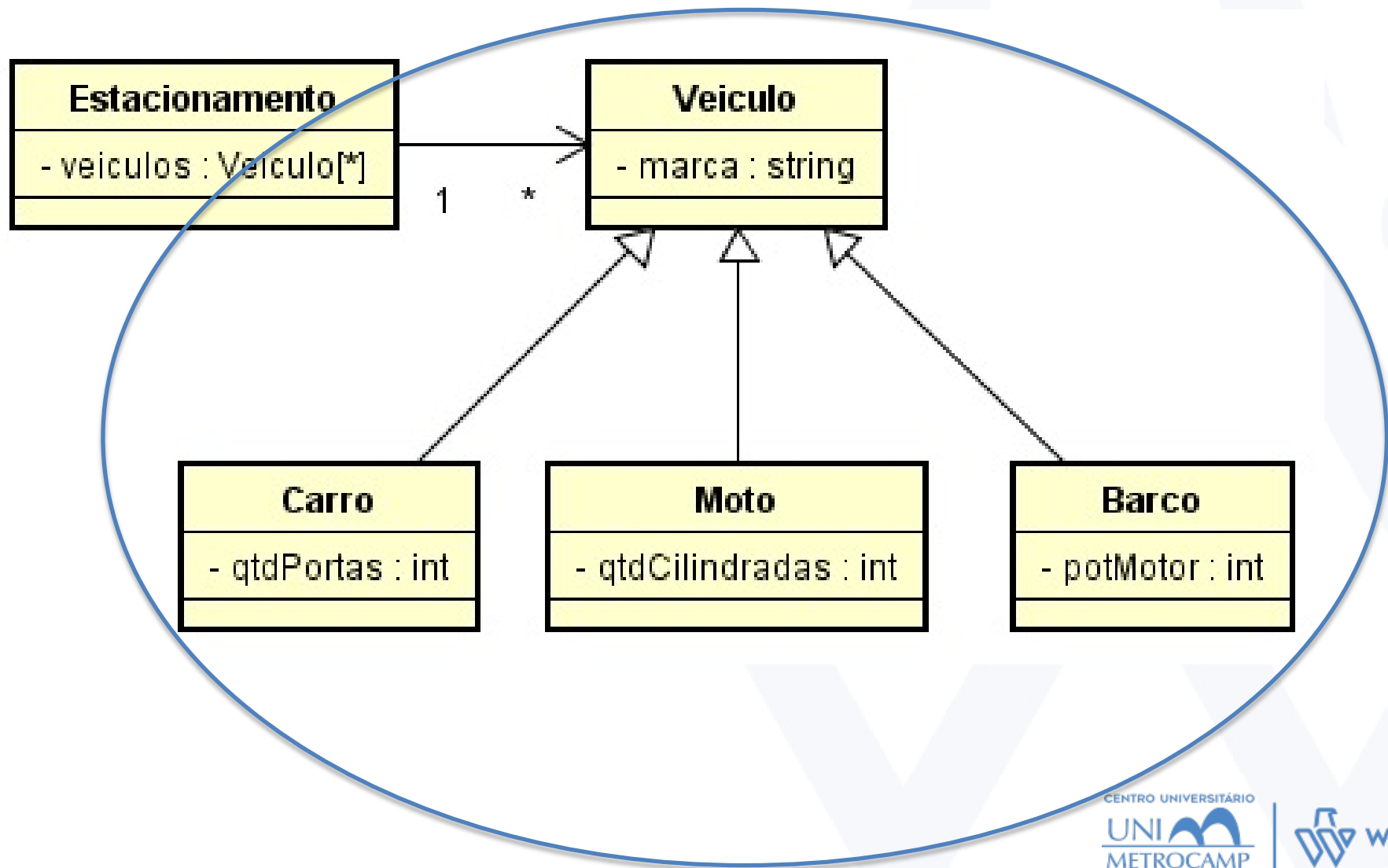
```
Não há combustivel suficiente para realizar o passeio!  
Volume no tanque (l): 0.0
```

```
Abastecimento realizado.  
Volume no tanque (l):20.0
```

```
Passeio será realizado com segurança!  
Volume no tanque apos passeio (l): 19.0
```

```
Impossível abaster: 50.0 litros.  
Volume no tanque (l): 19.0  
Capacidade do tanque (l):50.0
```

Herança



Herança

- A herança é um princípio da POO que permite a criação de novas classes a partir de outras previamente criadas.
- Essas novas classes são chamadas de subclasses, ou classes derivadas; e as classes já existentes, que deram origem às subclasses, são chamadas de superclasses, ou classes base.
- Deste modo é possível criar uma hierarquia dessas classes, tornando, assim, classes mais amplas e classes mais específicas.
- Uma subclasse herda métodos e atributos de sua superclasse; apesar disso, pode escrevê-los novamente para uma forma mais específica de representar o comportamento do método herdado.

Herança


- Para que o valor do atributo de uma super classe seja acessado em uma subclasse, utiliza-se o modificador (#) **protected**.

Exemplo

- Calculadora Básica e Científica.
 - A **básica** foi desenvolvida para realizar as 4 operações matemáticas básicas.
 - A **científica** “se aproveita” (ou reutiliza) o código da básica, para implementar uma nova calculadora, com operações adicionais.

Exemplo

- A classe Calculadora Básica:



```
public class CalcBasica {  
    protected float n1;  
    protected float n2;  
  
    public CalcBasica(float n1, float n2){  
        this.n1=n1;  
        this.n2=n2;  
    }  
  
    public float soma() {return this.n1+this.n2;}  
  
    public float subtracao() {return this.n1-this.n2;}  
  
    public float produto() {return this.n1*this.n2;}  
  
    public float divisao() {return this.n1/this.n2;}  
  
}
```

Exemplo

- O teste da Calculadora Básica:

```
public class tstCalcBasica {  
    public static void main(String[] args) {  
        CalcBasica c = new CalcBasica(9,9);  
  
        System.out.println("Soma: "+c.soma());  
        System.out.println("Subtração: "+c.subtracao());  
        System.out.println("Divisão: "+c.divisao());  
        System.out.println("Produto: "+c.produto());  
    }  
}
```

Exemplo

- A classe da Calculadora Científica, **herdando** da Calculadora Básica:

```
public class CalcCientifica extends CalcBasica {  
    public CalcCientifica(float n1, float n2) {  
        super(n1, n2);  
    }  
    public float potencia() {  
        return (float) Math.pow(this.n1, this.n2);  
    }  
}
```

Exemplo

- O teste da Calculadora Científica:

```
public class tstCalcCientifica {  
    public static void main(String[] args) {  
        CalcCientifica c = new CalcCientifica(3,3);  
  
        System.out.println("Soma: "+c.soma());  
        System.out.println("Subtração: "+c.subtracao());  
        System.out.println("Divisão: "+c.divisao());  
        System.out.println("Produto: "+c.produto());  
        System.out.println("Potencia: "+c.potencia());  
    }  
}
```

Exercício

- Implementar:

