

LINGUAGEM DE PROGRAMAÇÃO ORIENTADA À OBJETOS – LPOO

Aula 06 – Introdução a Programação Orientada a Objetos (POO) -

Profa Thais Rocha = thais.rocha@docente.unip.br

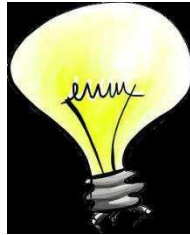


TÓPICOS DE HOJE

- Métodos, Construtores e Objetos
- Abstração, Associação, Composição e Encapsulamento
- Introdução a Herança e Polimorfismo

OBJETOS - PROPIEDADES

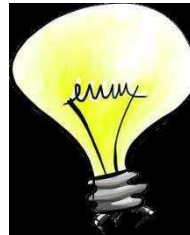
- Estado



Acesa

Apagada

- Comportamiento



Acender

Apagar

- Identidade



MÉTODOS

Método corresponde a uma ação (operação ou comportamento) que uma classe pode realizar.

Cada método funciona de forma independente, sendo utilizado apenas quando a ação é solicitada.

Todo o método é criado seguindo regras:

- As tarefas que um objeto pode realizar são definidas pelos seus métodos.
- Os métodos são os elementos básicos para a construção dos programas OO.
- Métodos não podem ser criados dentro de outros métodos ou fora de uma classe.

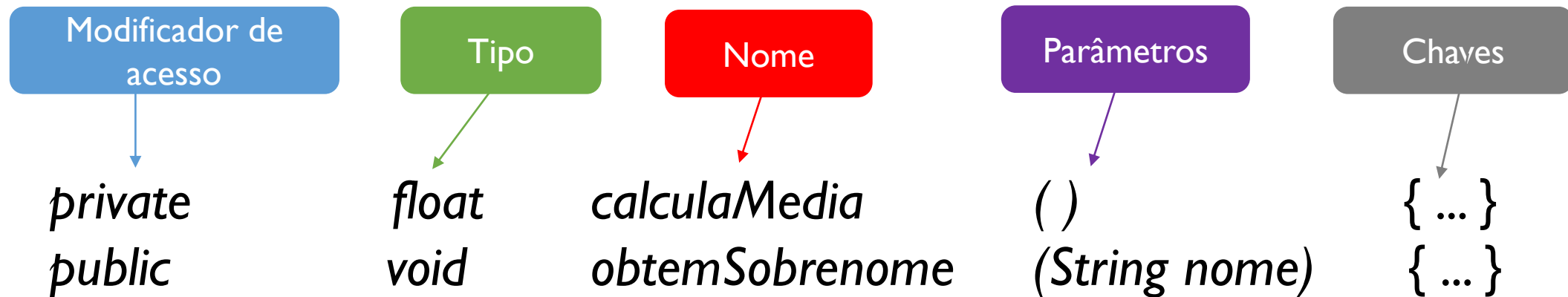
CRIANDO MÉTODOS

- Cada método deve incluir as **2 partes (cabeçalho e o corpo)** apresentadas
- **Cabeçalho (header)** - o cabeçalho de um método fornece informações sobre como outros métodos pode interagir com ele. *Um cabeçalho de método também é chamado de **declaração**.*
- **Corpo (body)** - O corpo do método contém o declarações que realizam o trabalho do método. *O **corpo de um método** é chamado de **implementação**.*

ASSINATURA DO MÉTODO

O cabeçalho (assinatura) do método é a primeira linha de um método. Ele contém o seguinte:

- **Modificador de acesso:** public ou private;
- **Tipo do retorno:** identifica o tipo de dados que a variável irá armazenar;
- **Nome:** começando com letra minúscula;
- **Parâmetros (opcional)**
- **Chave**



MÉTODOS – TIPOS DE RETORNO

- Um tipo de retorno descreve o **tipo de dados que o método envia de volta para sua chamada método**
- Nem todos os métodos retornam um valor para seus métodos de chamada;
- Os métodos sem retorno tem um tipo de retorno de específico chamada de **void**.

MÉTODOS – PARÂMETROS

- Alguns métodos exigem que os dados sejam enviados a eles quando forem chamados.
- Os dados que você usa em uma chamada a um método são chamados de **argumentos ou parâmetros**.
- Métodos que recebem parâmetros são flexíveis porque podem produzir diferentes resultados dependendo de quais dados eles recebem.

MÉTODOS – PARÂMETROS

- Como um exemplo da vida real, quando você faz uma reserva em um restaurante, não precisamos criar um método diferente para cada data do ano em todas as horas possíveis do dia.
- Em vez disso, você pode fornecer a data e hora **como parâmetros**, para a pessoa que faz a chamada do método.
- O método que faz o registro da reserva, é então realizado da mesma maneira, não importa o que data e hora são fornecidas.


CRIANDO MÉTODO COM APENAS UM PARÂMETRO

- Quando um método pode receber um parâmetro, sua declaração contém os mesmos elementos que um que não aceita um parâmetro
 - modificadores de acesso (opcional)
 - o tipo de retorno para o método,
 - nome do método
 - conjunto de parênteses que inclui dois itens:
 - tipo do parâmetro e o nome

public static void predictRaise(double salary)

No cabeçalho do método para `predictRaise()`, o parâmetro `salary` indica que o método receberá um valor do tipo `double`, e que dentro método, o valor passado será conhecido como `salary`.

CRIANDO MÉTODO COM APENAS UM PARÂMETRO



The diagram illustrates the components of a method parameter. Two boxes at the top are connected by lines to the parameter 'salary' in the code below. The box on the left, labeled 'Parameter data type', points to the word 'double'. The box on the right, labeled 'Parameter identifier', points to the word 'salary'.

```
public static void predictRaise(double salary)
{
    double newSalary;
    final double RAISE_RATE = 1.10;
    newSalary = salary * RAISE_RATE;
    System.out.println("Current salary: " +
        salary + "    After raise: " +
        newSalary);
}
```

MÉTODOS – CRIANDO MÉTODOS COM RETORNO

- Vimos ate agora, como podemos criar métodos sem retorno (void) e também receber um ou mais parâmetros ou até mesmo não receber nenhum parâmetro.
- Entretanto, muitas vezes precisamos fazer métodos que tem como a função retorna um informação no final da execução
- O tipo de retorno de um método pode ser qualquer tipo usado em Java, o que inclui os tipos primitivos e bem como tipos de classe (incluindo tipos de classe que você cria).

MÉTODOS – MULTIPLOS PARAMÊTROS

- Normalmente, vamos precisar criar métodos que devera receber diferentes parâmetros.
- Um método pode conter, nenhum, um ou mais parâmetros de diferentes tipos de dados .Veja os exemplos

```
public void exibelnformacoesUsuario ( ) { ... }
```

```
public void formataNome (String nome) { ... }
```

```
public void cadastroUsuario(String nome, int cpf, int rg, byte idade) { ... }
```

MÉTODOS – CRIANDO MÉTODOS COM RETORNO

- Vejamos alguns exemplo de métodos com retorno:

```
public String buscaEnderecoCompleto(String cep) { ... }
```

```
public int calculaQtdeCaracteres (String frase) { ... }
```

```
public float calculaMediaPonderada (double nota1, double nota2, double peso) { ... }
```

```
public double calculaDesconto (double preco, double porcentagem) { ... }
```

```
public boolean validaCPF (String cpf) { ... }
```

MÉTODOS - RESUMO

- Como vimos anteriormente podemos criar métodos de diferentes maneiras:
- **Sem retorno (void)** – *public void calculaSalario()*
- **Com retorno** – *public double calculaDesconto()*
- **Sem parâmetros** – *public void imprimeRelatorio()*
- **Com parâmetros**
 - **Único:** *public int calculaIdade(Data dataNascimento)*
 - **Múltiplos:** *public boolean verificaValidade (int idProduto, Data dataValidade)*

CONSTRUTOR

- Construtor é um tipo especial de método que **cria e inicializa as classes**.
- Um **construtor padrão** é aquele que não requer argumentos. Ele é **criado automaticamente pelo compilador Java para qualquer classe**. Ex:

```
Funcionario f = new Funcionario( );
```

- O construtor padrão automaticamente fornece os seguintes **valores iniciais específicos** para os campos de dados de um objeto:
 - *numéricos são definidos como 0 (zero).*
 - *Caracteres (char) são definidos como Unicode '\u0000'.*
 - *booleanos são definidos como false.*
 - *Os campos que são referências de objeto (por exemplo, String) são definidos como nulos (ou vazios).*

CONSTRUTORES (cont.)

- Podemos também iniciar a classe e atribuir valores aos atributos usando um construtor com parâmetros. Veja o exemplo a seguir
- **Classe:** Cliente
- **Atributos:** id, nome, idade, endereco, cpf e rg
- **Construtores**

```
public Cliente() {  
    super();  
}
```

Default (padrão)

```
public Cliente(int id, String nome, byte idade,  
                String endereco, String cpf, String rg) {  
    super();  
    this.id = id;  
    this.nome = nome;  
    this.idade = idade;  
    this.endereco = endereco;  
    this.cpf = cpf;  
    this.rg = rg;  
}
```

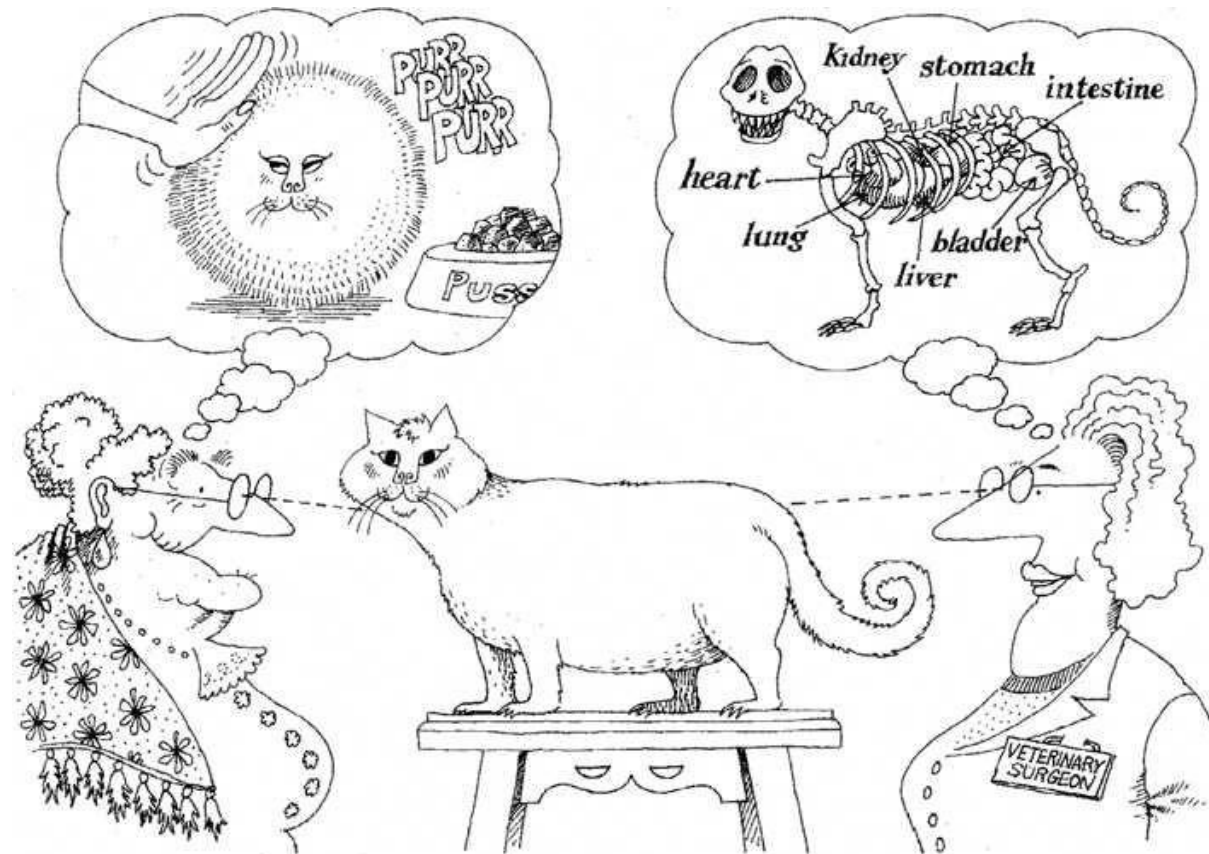
Com parâmetros

ABSTRAÇÃO

- **Abstração** é uma das formas fundamentais que nós lidamos com a complexidade.;
- Não se analisa o “todo”, em POO é importante analisar as partes para entender o todo.
- Quando queremos diminuir a complexidade de alguma coisa, ignoramos detalhes sobre as partes para concentrar a atenção no nível mais alto de um problema;

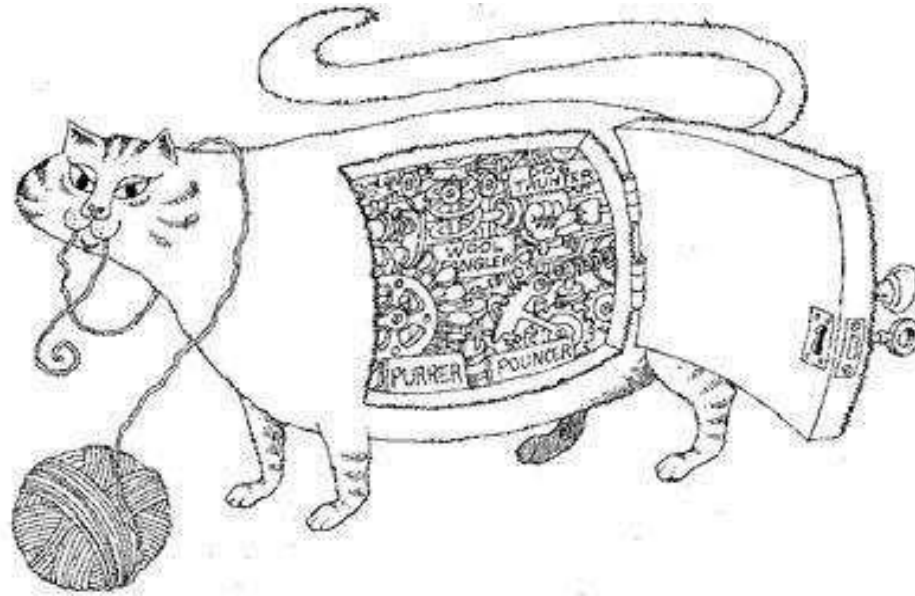
ABSTRAÇÃO

- Foca a característica essencial de alguns objetos relativo a **perspectiva do visualizador**



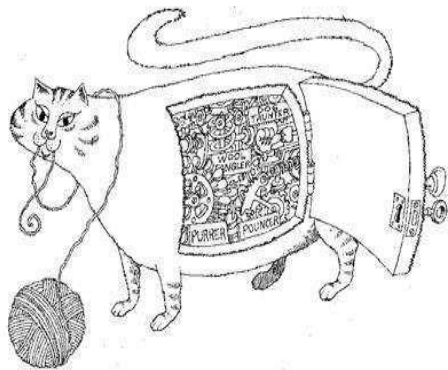
ENCAPSULAMENTO

- **Encapsulamento** é o processo de esconder todos os detalhes de um objeto que não contribuem para as suas características essenciais;



ENCAPSULAMENTO

- O encapsulamento é o modo de dar ao objeto seu comportamento “caixa-preta”, que é o segredo da reutilização e confiabilidade.



Se o estado de um objeto foi modificado sem uma chamada de método desse objeto, então o encapsulamento foi quebrado

MODULARIDADE

- **Modularização** é o processo de dividir um todo em partes bem definidas, que podem ser construídas e examinadas separadamente.
- Essas partes se interagem entre si, fazendo com que o sistema funcione de forma adequada
- **Particionar um programa em componentes individuais, pode reduzir a complexidade.**



RESUMO

- **Objeto**
 - Qualquer entidade que possui características e comportamento
- **Classe**
 - Descreve um tipo de objeto
 - Define atributos e métodos
- **Atributo**
 - Define características do objeto
- **Método**
 - Operações que o objeto pode realizar

Person
-name : String -birthDate : Date
+getName() : String +setName(name) : void +isBirthday() : boolean

Book
-title : String -authors : String[]
+getTitle() : String +getAuthors() : String[] +addAuthor(name)

CLASSES COMO ATRIBUTOS PARA OUTRAS CLASSES

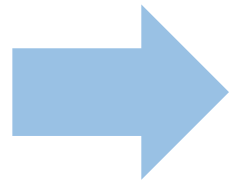
- Os atributos podem ter o tipo primitivos (int, long, double...) e wrappers (Integer, Double, Boolean, String) e **TAMBÉM OUTRAS CLASSES.**
- Exemplo: Na classe Cliente criada anteriormente temos um atributo chamado endereco do tipo String. Para armazenar o endereço completo do cliente poderíamos fazer na seguinte forma.

```
String endereco = "Avenida Fabrício Vampré, 1000, CEP: 13484-323, Jd.  
São Paulo, Limeira, SP";  
Cliente c = new Cliente()  
c.setEndereco(endereco);
```


CLASSES COMO ATRIBUTOS PARA OUTRAS CLASSES

- Entretanto, podemos modificar para que o endereço seja transformado em uma classe, da seguinte maneira
- Classe: Endereco
- Atributos: id, nome, logradouro, numero, bairro, CEP, cidade, estado

```
public class Cliente {  
    private Integer id;  
    private String nome;  
    private String endereco;  
    ....  
}
```



```
public class Cliente {  
    private Integer id;  
    private String nome;  
    private Endereco endereco;  
    ....  
}
```

MÉTODOS - OVERLOADING

- Com a **sobrecarga (overloading) de método**, vários métodos podem ter **o mesmo nome com parâmetros diferentes**:

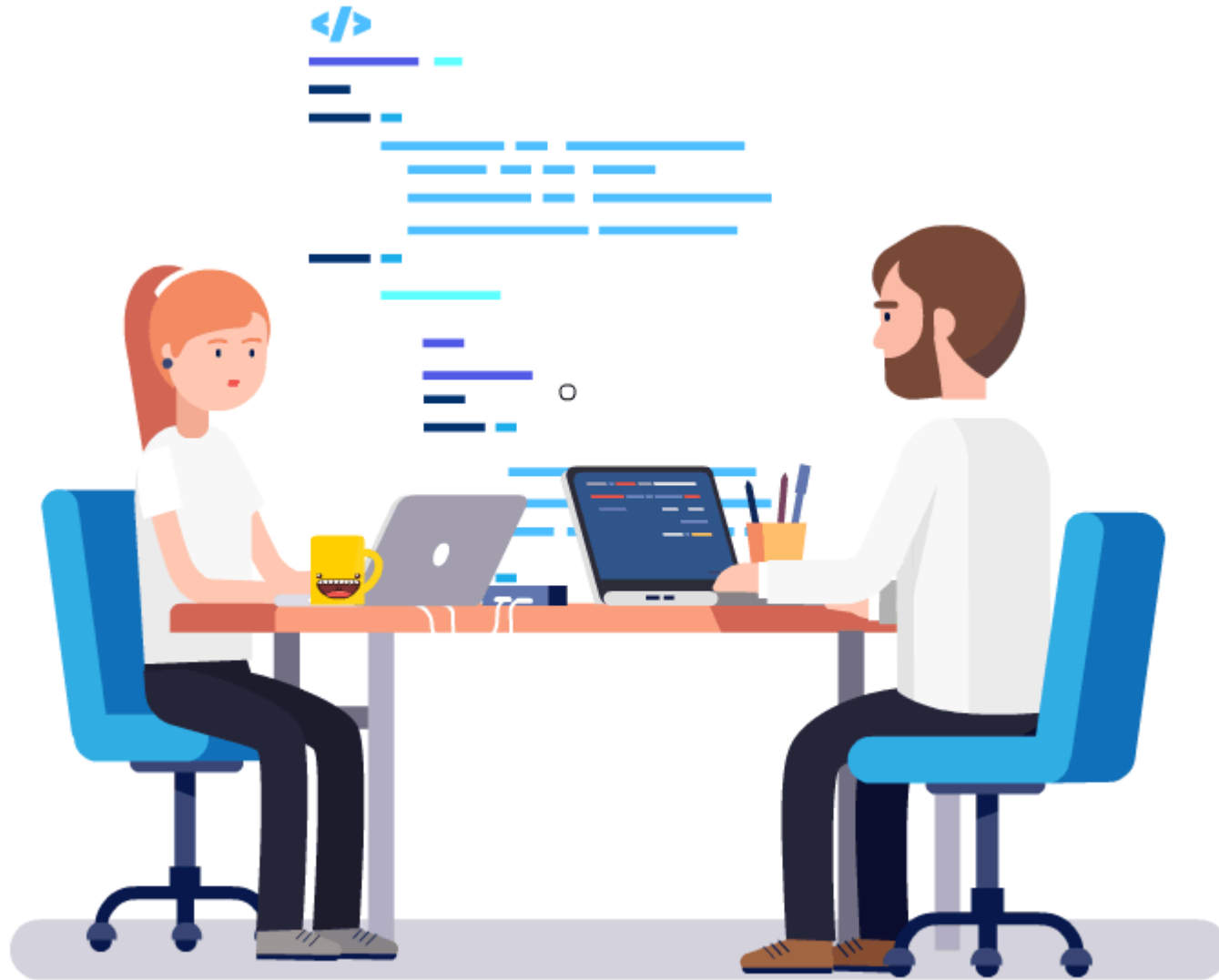
Exemplo:

```
int myMethod(int x)  
float myMethod(float x)  
double myMethod(double x, double y)
```

Vamos analisar alguns exemplos:

https://www.w3schools.com/java/java_methods_overloading.asp

PRATICANDO ...



EXERCÍCIOS

1) Criar um método **sem retorno** chamado **formatarNome**, que recebe com parâmetro:

- nomeCompleto – String

O método será responsável em exibir na tela o nome do usuario em letra maiuscula.

2) Criar um método **sem retorno** chamado **calculaQtdeCaracteres**, que recebe com parâmetro

- uma frase

O método será responsável em exibir na tela (Scanner ou System.out.println) a quantidade de caracteres da frase fornecida.

3) Sobre os atributos e métodos de uma classe, podemos afirmar que:

- a) atributos não apresentam parêntesis;
- b) métodos só apresentam parêntesis se possuírem parâmetros;
- c) atributos devem possuir uma implementação declarada na classe que será executada no objeto;
- d) métodos sem parâmetros comportam-se como atributos;
- e) métodos com retorno void devem possuir o comando return em sua implementação.

4) Sobre os atributos e métodos de uma classe, podemos afirmar que:

- a) atributos não apresentam parêntesis;
- b) métodos só apresentam parêntesis se possuírem parâmetros;
- c) atributos devem possuir uma implementação declarada na classe que será executada no objeto;
- d) métodos sem parâmetros comportam-se como atributos;
- e) métodos com retorno void devem possuir o comando return em sua implementação.

5) Qual das alternativas abaixo pode ser afirmada corretamente com relação aos métodos construtores?

- a) Métodos construtores não possuem valor de retorno, por isso são sempre void.
- b) Métodos construtores não podem receber parâmetros.
- c) Métodos construtores podem ser executados a qualquer momento.
- d) Métodos construtores podem ter qualquer nome.
- e) Uma classe pode ter mais de um método construtor.

ENCAPSULAMENTO

- Uma das principais vantagens do paradigma da orientação a objetos é a possibilidade de **encapsular os atributos**, bem como os **métodos** capazes de manipular esses atributos em uma classe.
- **É desejável que os atributos das classes fiquem ocultos ou escondidos dos programadores usuários dessas classes para evitar que os dados sejam manipulados diretamente, mas que sejam manipulados apenas por intermédio dos métodos da classe.**

ENCAPSULAMENTO

A restrição ao acesso a atributos e métodos em classes é feita por meio de **modificadores de acesso** que são declarados dentro das classes, antes dos métodos e dos campos.

A restrição de acesso é estabelecida na definição da classe usando um dos modificadores:

- **private**
- **protected**
- **public**

ENCAPSULAMENTO – MODIFICADORES DE ACESSO

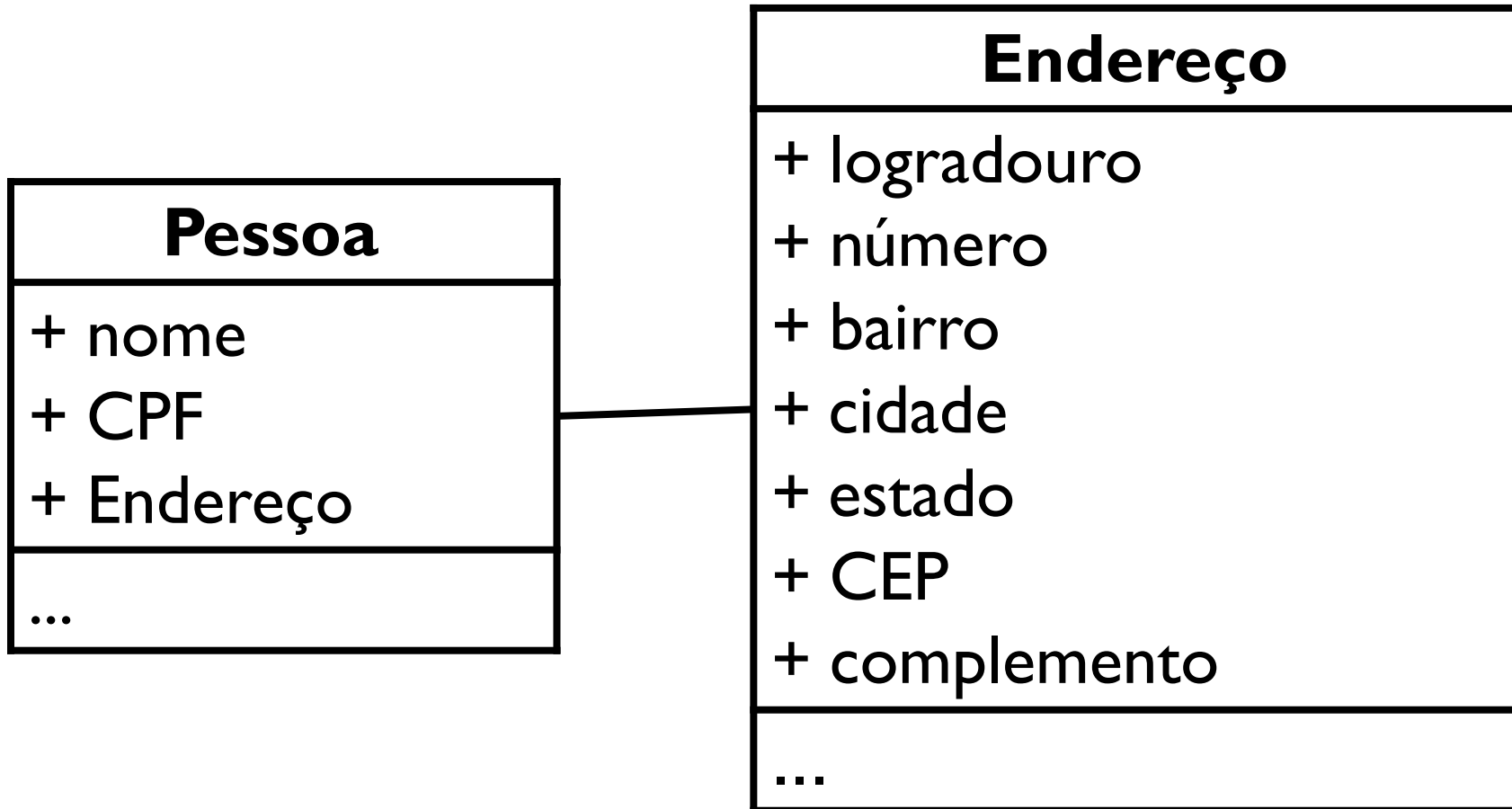
- **Modificador public:** garante que o atributo ou método da classe declarado com esse modificador possa ser acessado ou executado a partir de qualquer outra classe, ou seja, sem restrição.
- **Modificador private:** atributos ou métodos declarados com esse modificador só podem ser acessados, modificados ou executados por métodos da própria classe, sendo completamente ocultos para o programador usuário que usar instâncias desta classe ou criar classes herdeiras ou derivadas.
- **Modificador protected:** funciona como o modificador private, exceto pela diferença de que classes herdeiras ou derivadas também terão acesso ao campo ou método marcado com esse modificador. Assim, é permitido o acesso a todas as classes derivadas.

ENCAPSULAMENTO – EXEMPLOS

Sem encapsular	Encapsulamento
<pre>public Tipo atributo;</pre>	<pre>private Tipo atributo; public Tipo getAtributo() { return this.atributo; } public void setAtributo(Tipo valor) { this.atributo = valor; }</pre>
<pre>Referencia.atributo=x;</pre>	<pre>Referencia.setAtributo(x);</pre>
<pre>x=Referencia.atributo;</pre>	<pre>x=Referencia.getAtributo();</pre>

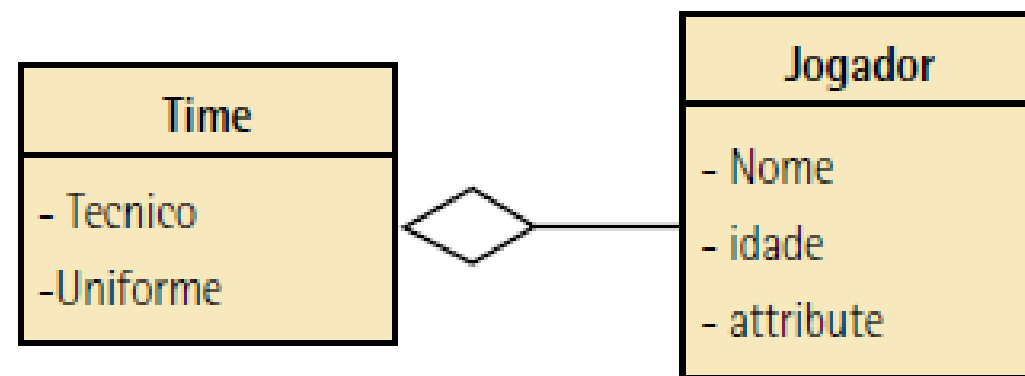
ASSOCIAÇÃO

Relação entre Classes: Associação



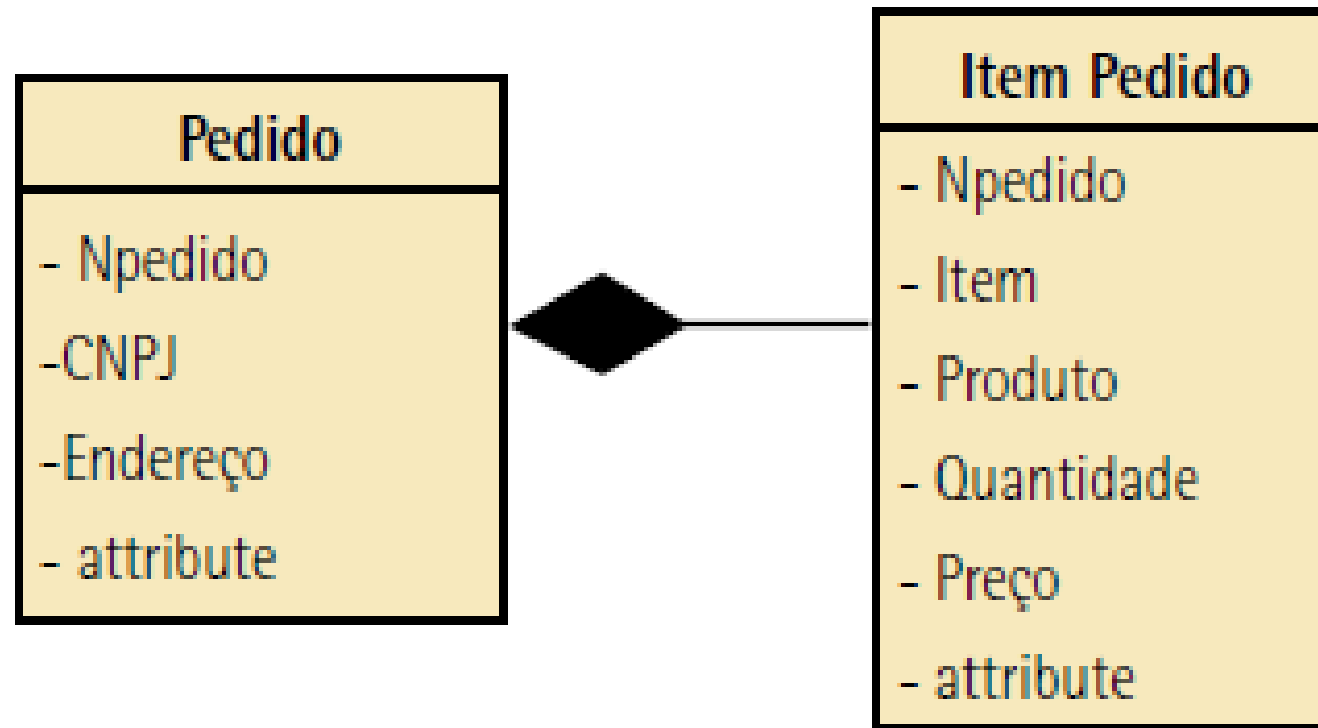
AGREGAÇÃO

- **Agregação** é um tipo especial de associação em que se tenta demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (chamado objeto-parte).
- Ela define uma **dependência fraca** entre as classes, ou seja, **os objetos continuam existindo mesmo que o todo seja removido.**
- **Assim, se temos um time de basquete e pessoas como jogadores, ambos existirão independentemente.**



COMPOSIÇÃO

- O exemplo mais típico é o do pedido de compras. **Ele não existirá se não houver itens, assim como itens não fazem sentido se não estiverem agrupados em um pedido.**



HERANÇA

- **Herança** é um conceito muito importante na Programação Orientada a Objetos (POO).
- Seu objetivo é a derivação de classes, ou seja, a criação de uma classe (classe filha) onde sua base é uma classe já existente (classe pai), sendo adicionados a essa nova classe apenas atributos e métodos que não existam na classe original ou a execução de um método que seja diferente do método da classe original.
- Em outras palavras, a **herança é um mecanismo de reaproveitamento em que as classes originais ficam contidas na nova classe.**

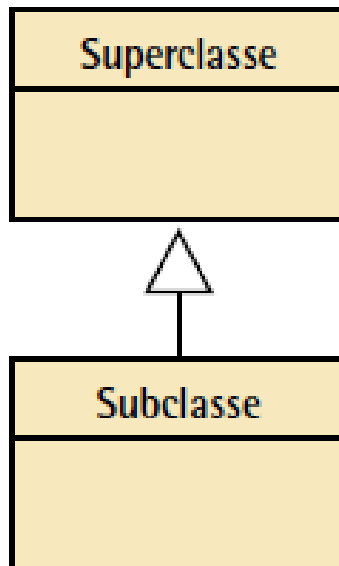
```
class Filho : Pai
```

Nome da superclasse



HERANÇA

- No jargão das linguagens OO, a classe base, aquela que serve para a criação de classes mais especializadas, é chamada de superclasse (ou classe pai, ou classe base) e a que herda as características da superclasse é chamada de subclasse.
- **Em UML, a relação de herança é representada por uma seta:**



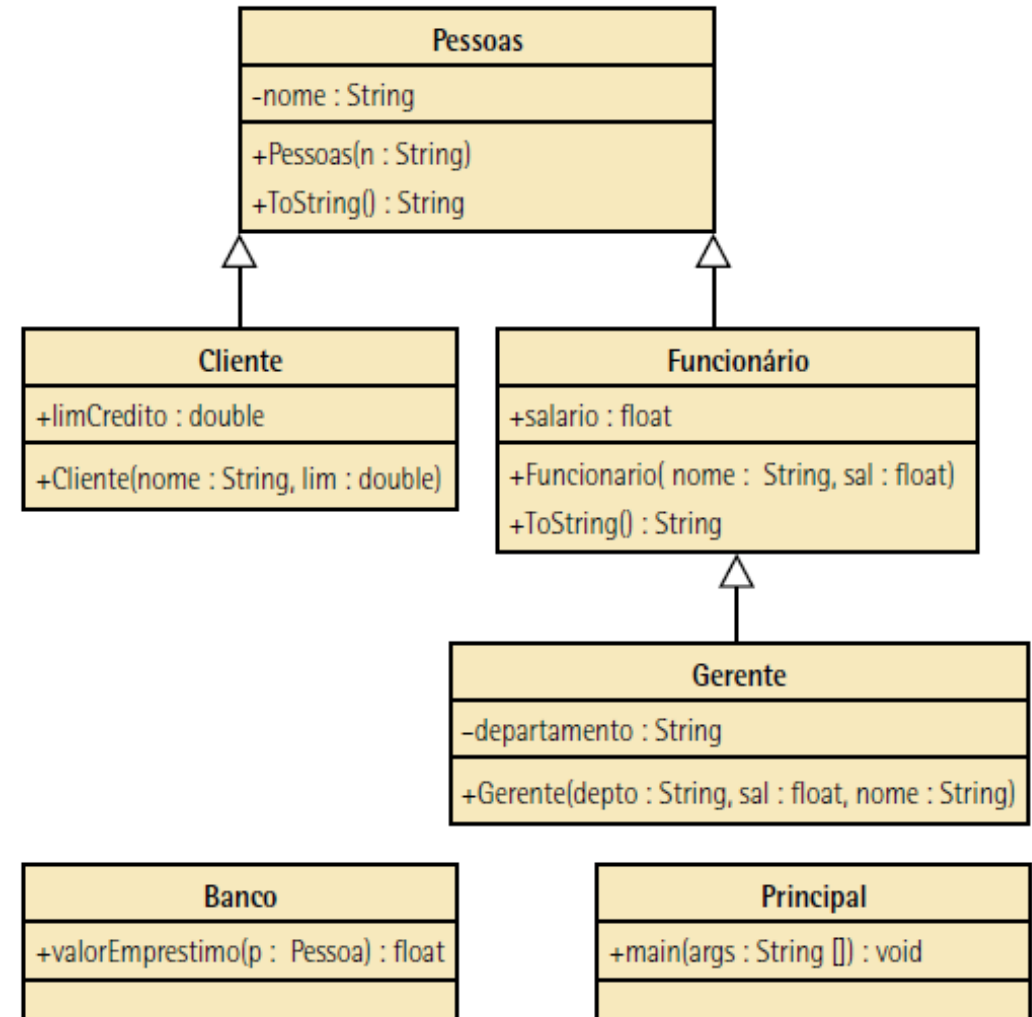
Herança não é cópia, é uma derivação! Alterações realizadas na classe Pai poderão afetar a classe Filha, porém a classe Filha poderá ser definida com comportamentos distintos da classe Pai.

POLIMORFISMO

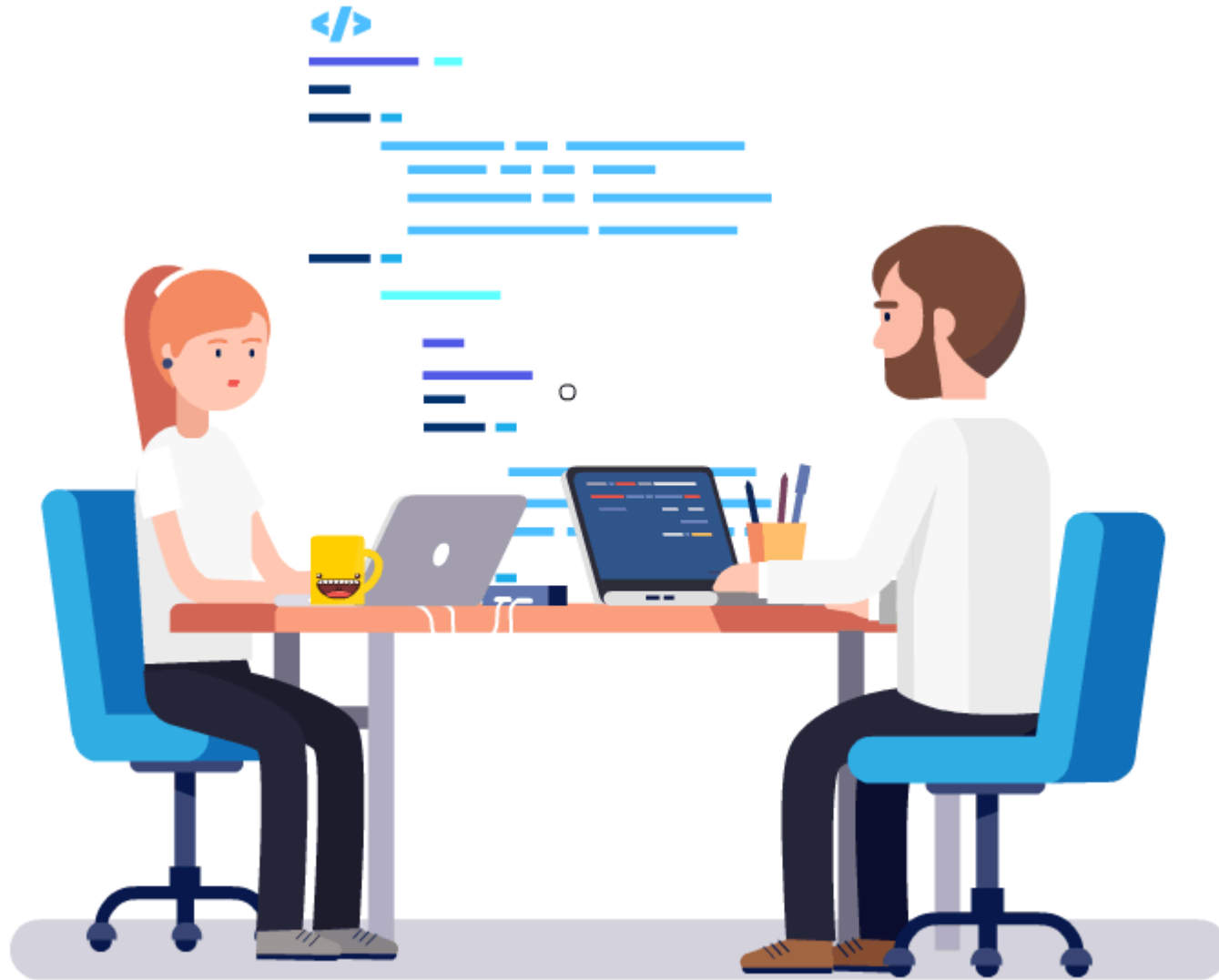
- O **polimorfismo** está relacionado com o conceito de herança, especificamente em relação a métodos. O mecanismo de herança permite a criação de classes a partir de outras já existentes com relações “**é-um-tipo-de**”, de forma que, a partir de uma classe genérica, classes mais especializadas possam ser criadas.
- Quando uma nova classe necessita de todos os atributos e métodos de uma já existente, porém a nova classe possui a execução de um ou mais métodos diferenciados, é **possível herdarmos todos os métodos e atributos da classe original e realizar a alteração do comportamento do método somente na nova.**

POLIMORFISMO

- A classe Pessoa pode ser particularizada em outras categorias.
- No nosso caso, **o cliente é uma pessoa que possui um limite de crédito.**
- **Um funcionário é uma pessoa que tem salário e o gerente é um funcionário que dirige um departamento.**
- Assim, como temos vários níveis de especialização, do mais geral (a pessoa) até o mais específico (o gerente), o relacionamento entre as classes será de herança:



PRATICANDO ...



6) Qual das alternativas abaixo não se aplica ao conceito de herança em orientação a objetos?

- a) ☐ O uso de herança torna o programa mais eficiente no uso de memória e processamento.
- b) ☐ O uso de herança incentiva a prática do reúso.
- c) ☐ O uso de herança torna o código mais fácil de ser entendido.
- d) ☐ O uso de herança reduz o custo de manutenção do código.
- e) ☐ O uso de herança só é possível em linguagens orientadas a objetos.

7) Qual das seguintes afirmações é falsa?

- a) Objeto é uma instância de uma classe.
- b) Objetos reúnem dados e comportamentos relacionados com um único conceito.
- c) Classes não podem se relacionar com outras classes, afinal, cada classe abstrai um único conceito.
- d) O uso de polimorfismo costuma tornar o código mais fácil de ser entendido.
- e) O uso de polimorfismo pode reduzir a quantidade de linhas do código.

EXERCÍCIOS

8) O paradigma orientado a objetos incentiva a prática da reutilização de código. Qual das alternativas abaixo não se aplica a este conceito?

- a) Com o polimorfismo entre classes, economizamos a escrita de código
- b) A delegação permite reutilizar classes em outras classes.
- c) A herança entre classes faz com que definições de atributos e métodos sejam passados de uma classe a outra.
- d) Podemos reutilizar um método construtor de uma superclasse invocando-o do construtor de uma subclasse.
- e) A reutilização de código torna a manutenção do sistema mais rápida e barata.

EXERCÍCIOS COMPLEMENTARES

- I) Faça um resumo sobre a **Histórico da Programação Orientada a Objetos (POO)**, contendo os seguintes tópicos :
- Quando surgir e qual foi a primeira linguagem de programação a utilizar conceitos de POO ?
 - Quem foi o criador da linguagem C# e Java ?
 - Diferença entre programação estrutural e programação orientada a objetos
 - Quais linguagens de programação são exclusivamente baseado no paradigma da programação estrutural ?

EXERCÍCIOS COMPLEMENTARES

2) Sobre a **linguagem de programação Java**, responda as seguintes questões:

- a) Quais são os principais mitos e verdades ?
- b) Quais Framework/API que fazem parte do ecossistema Java
- c) Quais a diferença entre JDK, JRE e JVM ?
- d) Quais os principais etapas de compilação de um código-fonte escrito em Java ?
- e) Explique como é feito a instalação do Java e configuração das variáveis de ambientes JAVA_HOME e PATH.
- f) Quais são as principais IDE para escrever códigos Java. Qual é o seu favorito e porque ?

EXERCÍCIOS COMPLEMENTARES

3) Sobre os conceitos de **Programação Orientada a Objeto (POO)**, responda:

- a) Quais são os principais benefícios da POO em relação a Programação Estrutural ?
- b) Quais são os 3 principais pilares da POO ?
- c) O que são classes ? Dê exemplo justificar a sua resposta
- d) O que são objetos ? Quais as diferença entre eles ?
- e) Explique a diferença entre abstração, encapsulamentos e modularidade ?

