VEM SER

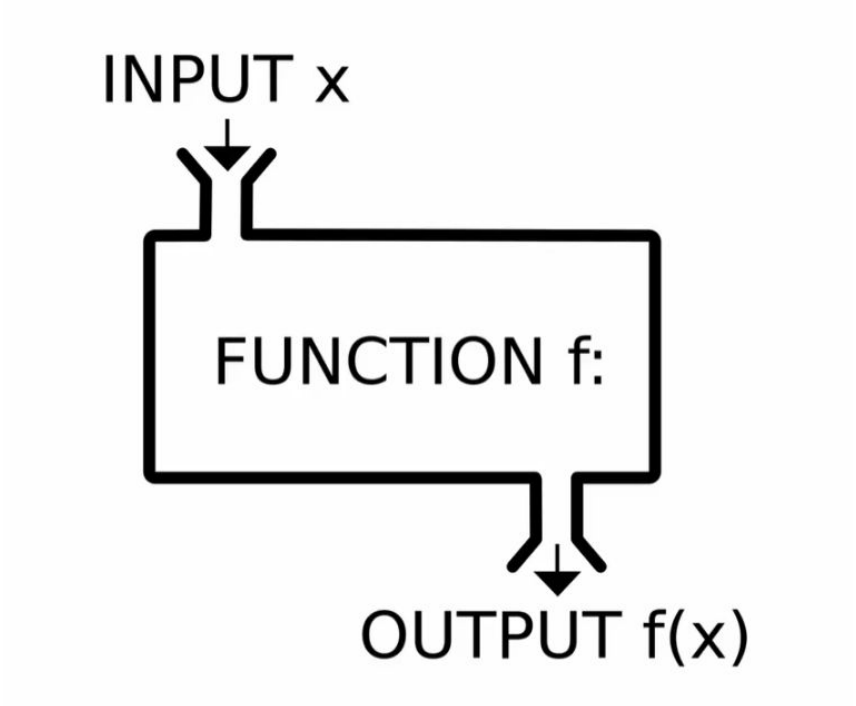
Módulo 01 - Java + OO

Aula 04 - Orientação a Objetos

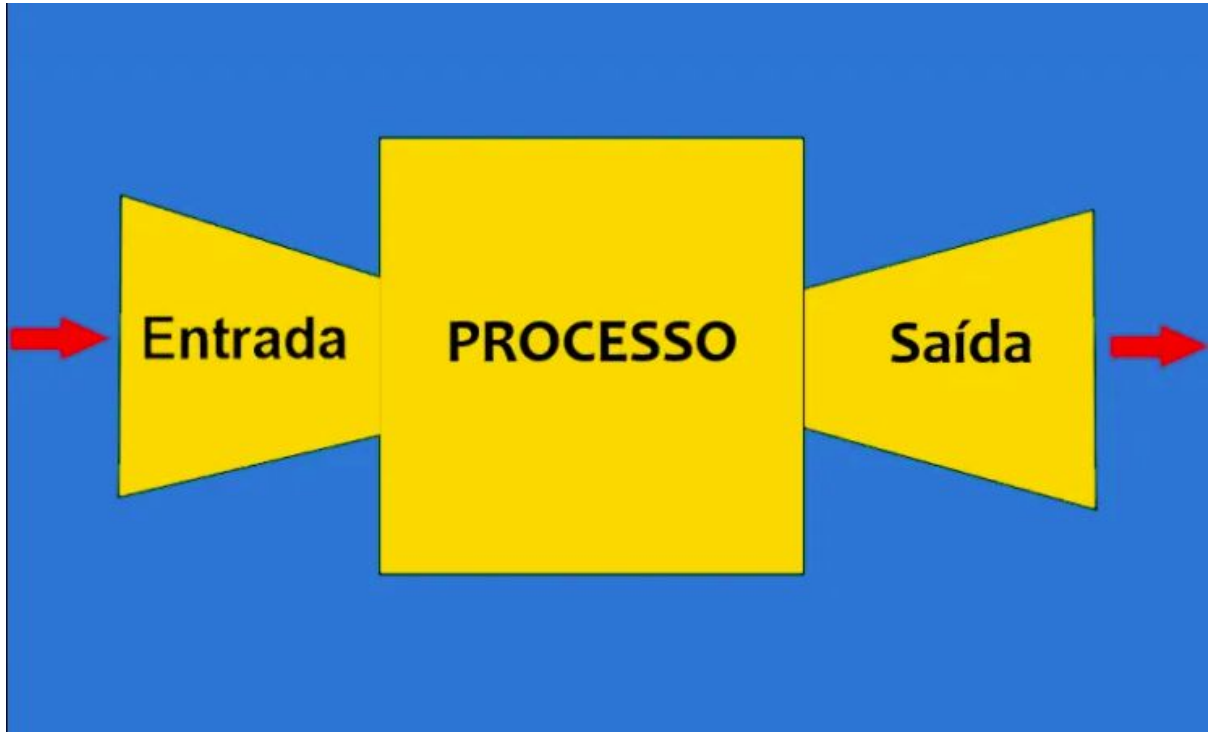
Conteúdo da aula

- Encapsulamento
- Modificadores de acesso
- Construtores
- Conceito de estático (*static*) no Java
- Herança
- Interfaces

Relembrando funções...



Relembrando funções...

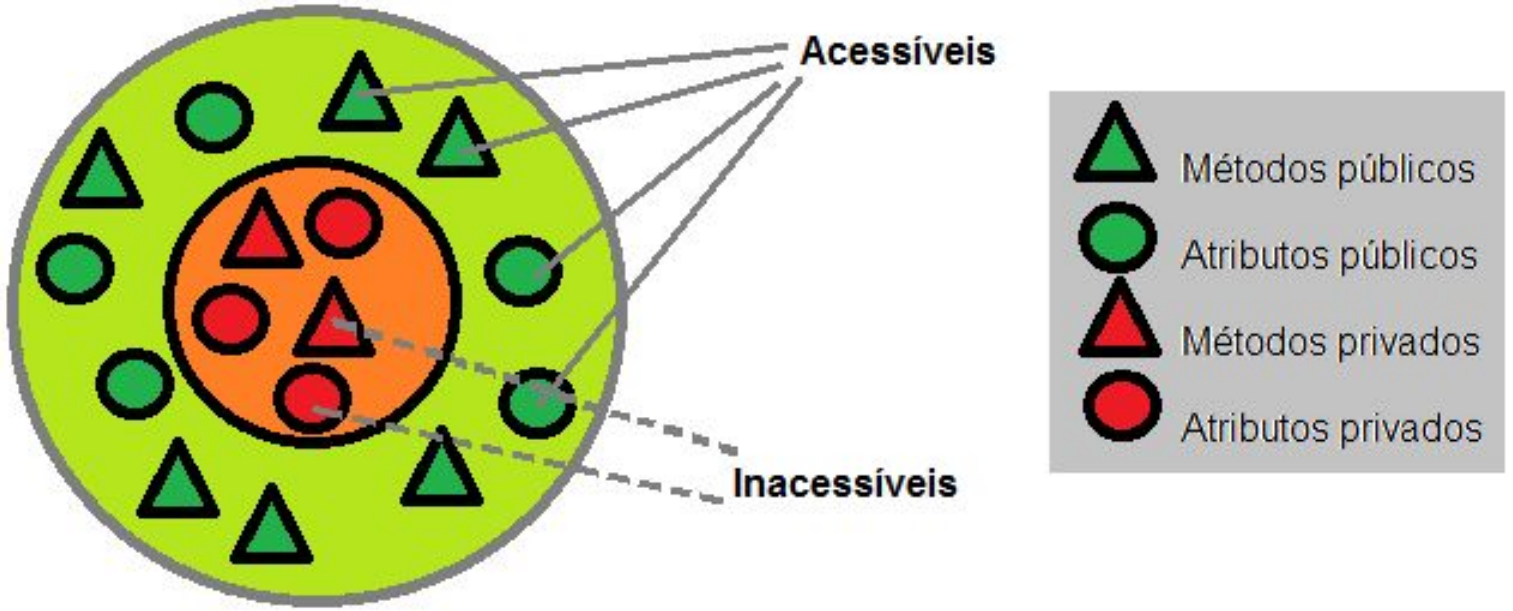


Encapsulamento

Encapsulamento

- Deixar visível/público apenas o essencial;
- Uma boa prática é “esconder” todos os atributos da classe e utilizar métodos assessores (*getters/setters*);
- Isso é muito importante para garantir a segurança do código e evitar problemas indesejados.

Encapsulamento



Crédito da imagem: http://tecnopode.blogspot.com/2018/01/poo-f-08.html

Encapsulamento

```
class Carro {

  private String modelo;

  public String getModelo() { // getter
    return modelo;
  }

  public void setModelo(String modelo) { // setter
    this.modelo = modelo;
  }
}
```

Encapsulamento

```
Carro fiesta = new Carro();

fiesta.setModelo("Celta"); // "setando" o modelo

System.out.println(fiesta.getModelo());
```

Recomendação de leitura

https://www.tutorialspoint.com/java/java_encapsulation.htm

Modificadores de acesso

Modificadores de acesso

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Crédito da imagem: https://www.javatpoint.com/access-modifiers

Recomendação de leitura

https://www.tutorialspoint.com/java/java_access_modifiers.htm

Vamos praticar!

Transforme todos os atributos do Exercício #1 da aula passada em *private*

- Crie métodos getters and setters;
- Modifique o que for necessário para que o programa main passe a rodar.

DBC

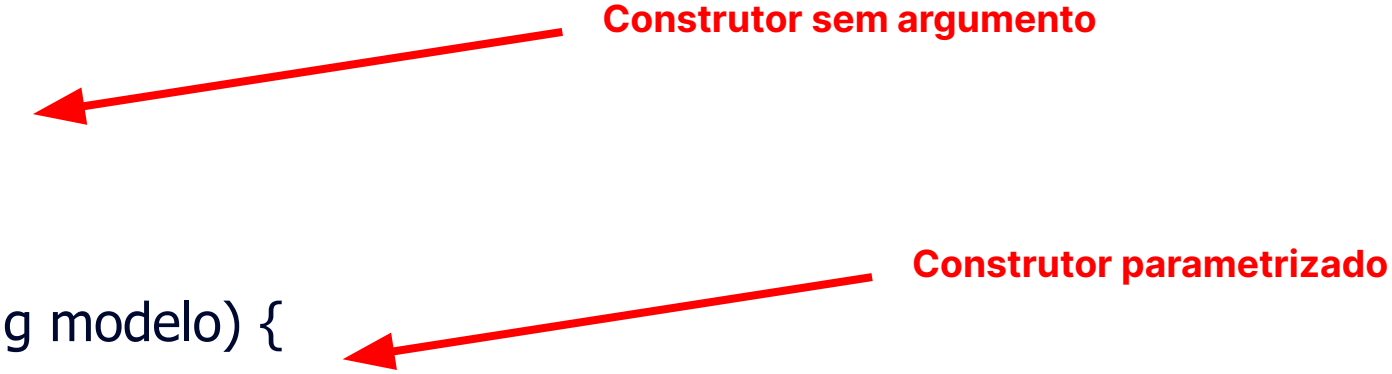
Constructor

Método construtor

Serve para construir a classe ao instanciar um objeto com o “new”.

Método construtor

```
class Carro {
  private String modelo;

  public Carro() {
  }

  public Carro(String modelo) {
    this.modelo = modelo;
  }
}
```



Recomendações

https://www.tutorialspoint.com/java/java_constructors.htm

https://youtu.be/rW11EAkxFnc

Instanciando com argumentos

```
Carro fiesta = new Carro("Ford Fiesta");

System.out.println(fiesta.getModelo());
```

Vamos praticar!

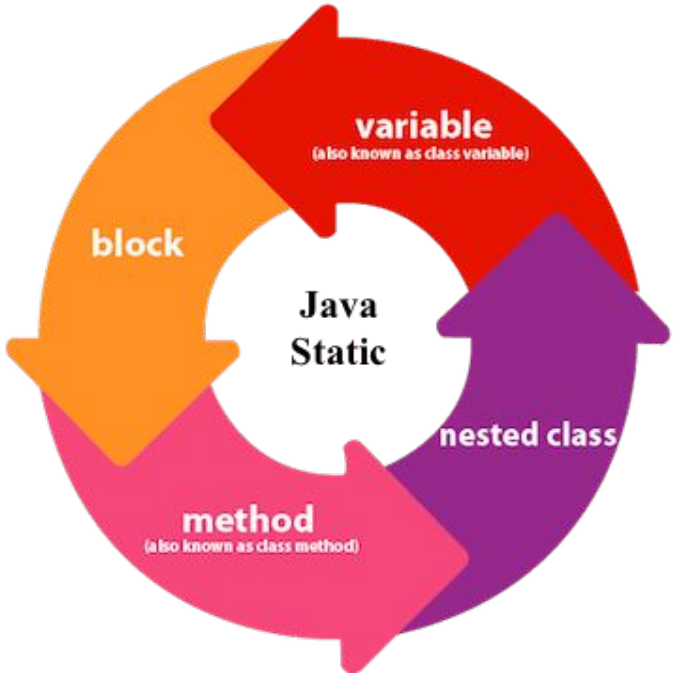
- **Crie um construtor public vazio (padrão) para a classe Pessoa;**
- **Escolha pelo menos 2 campos da classe Pessoa e crie um construtor parametrizado com eles**
  - Teste esse construtor no seu método *main*.

DBC

Static

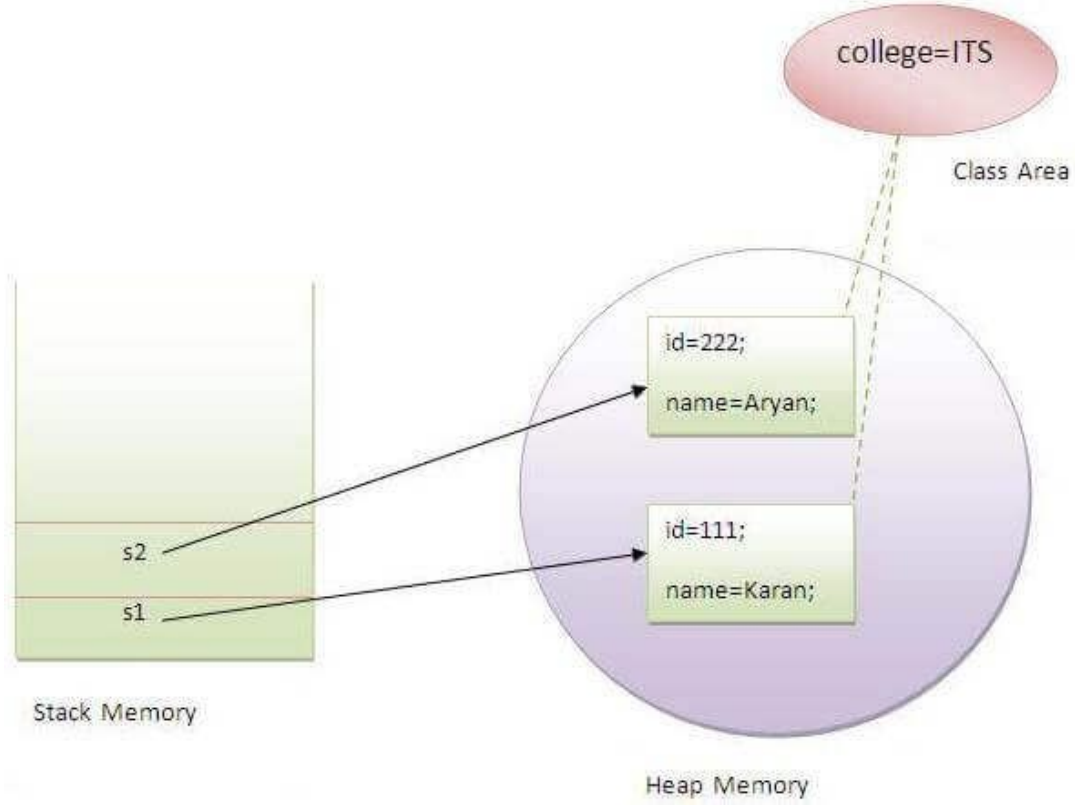
Conceito de estático (*static*) no Java

"A palavra-chave static pertence à classe e não a uma instância da classe."
Ou seja, tudo que é estático **pertence à classe** e não ao objeto.



Crédito da imagem: https://www.javatpoint.com/static-keyword-in-java

Conceito de estático (*static*) no Java



Crédito da imagem: https://www.javatpoint.com/static-keyword-in-java

Dicas de leitura

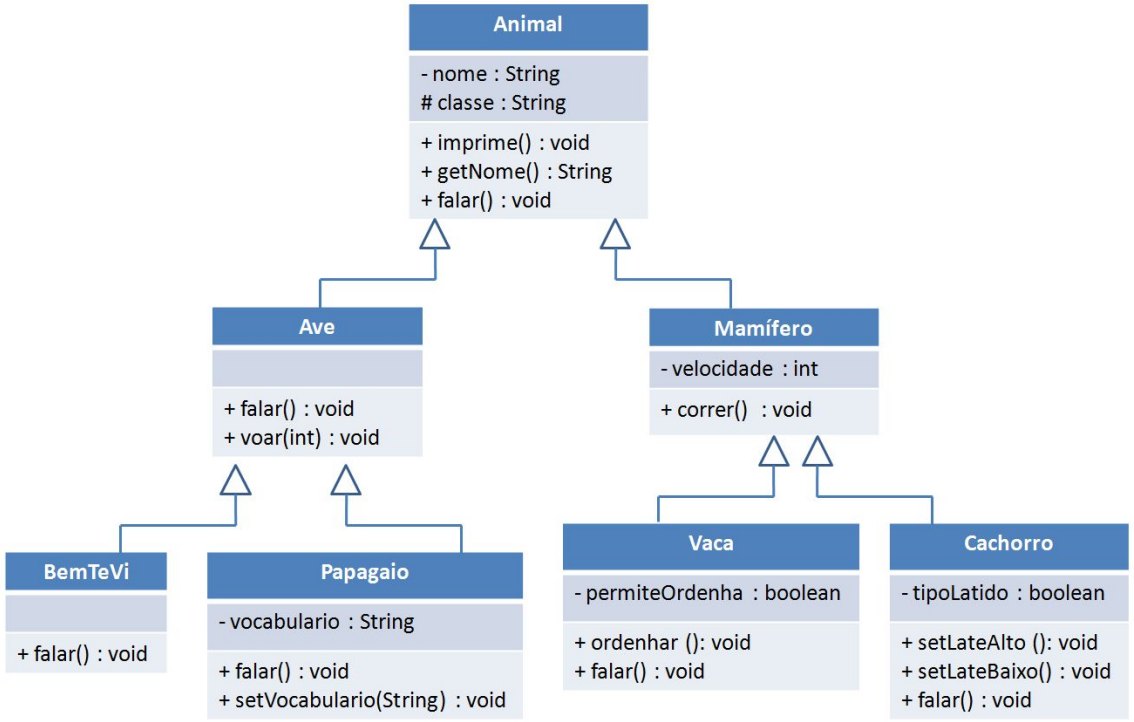
https://www.javatpoint.com/stack-vs-heap-java

Herança

Herança

- Quando dizemos que uma classe A é ***um tipo de*** classe B
- Dizemos que a classe A herda as características da classe B e que a classe B é ***mãe*** da classe A, estabelecendo então uma relação de herança entre elas.

Herança



Crédito da imagem: https://www.limontec.com/2019/04/aula-4-4-poo-java-ltcode.html

Herança

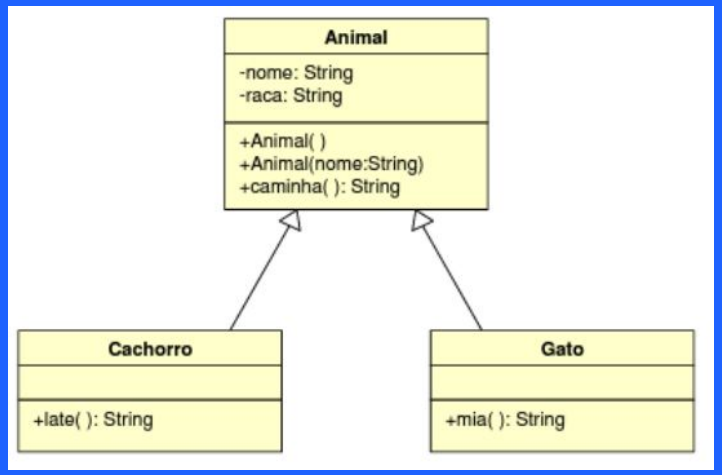
```
abstract class Animal {
  private String nome;

  public Animal(String nome) {  // construtor
    this.nome = nome;
  }

  public String getNome() {
    return nome;
  }
}
```

Herança

```
class Mamifero extends Animal {
}
```

Vamos praticar!

- **Implemente o diagrama de classes abaixo com seus respectivos métodos**



- Crie um método *main* e teste as classes criadas.

DBC

Interfaces

Interfaces

- Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado;
- Dentro das interfaces existem assinaturas de métodos e propriedades, cabendo à classe que a utilizará realizar a implementação das assinaturas, dando comportamentos práticos aos métodos;
- Alguns tipos de interfaces:
    - Interface de contrato;
    - Interface de marcação.

Interface de Contrato

```
public interface Veiculo {
  String getNome();
  String getId();
}

public interface Motor {
  String getModelo();
  String getFabricante();
}
```

Interface de Contrato

```
public class Carro implements Veiculo, Motor {
  @Override
  public String getNome() {
    return null;
  }
  @Override
  public String getId() {
    return null;
  }
  @Override
  public String getModelo() {
    return null;
  }
  @Override
  public String getFabricante() {
    return null;
  }
}
```

Interface de Marcação

```
public interface Funcionario {
}
```

Interface de Marcação

```
public class Gerente implements Funcionario {
  private int id;
  private String nome;
}
public class Coordenador implements Funcionario {
  private int id;
  private String nome;
}
public class Operador implements Funcionario {
  private int id;
  private String nome;
}
```

Recomendação de leitura

https://www.devmedia.com.br/entendendo-interfaces-em-java/25502

Task #1 individual

- Corrija a task anterior, na sequência copie e cole a pasta e renomeie para "**conta-corrente2**"
- Aprimore o programa da aula anterior conforme o novo diagrama:

https://lucid.app/lucidchart/invitations/accept/inv_52c28dc8-d70e-4a48-b4ee-9e73445ef0dc

Task #2 em grupo

- Crie 2 interfaces relevantes para o seu projeto


Let's Tech Up Together