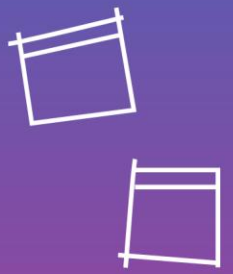
DBC
DIGITAL BUSINESS COMPANY®

VEM SER
DBC



Orientação a Objetos II

Sumário

- Encapsulamento
- Modificadores de Acesso
- Construtor
- Conceito Estático No Java
- Herança
- Interfaces

Encapsulamento

Encapsulamento

- Deixar visível somente o que é importante.

Encapsulamento

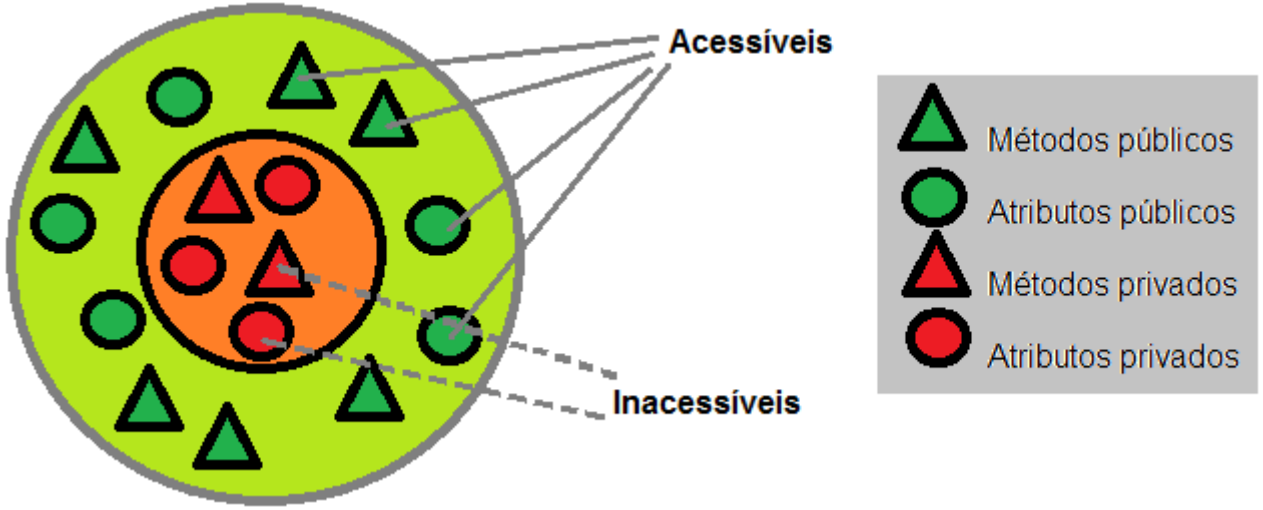
- Deixar visível somente o que é importante.
- Uma boa prática é “esconder” todos os atributos da classe e utilizar com métodos assessores (famosos getters/setters).

Encapsulamento

- Deixar visível somente o que é importante.
- Uma boa prática é “esconder” todos os atributos da classe e utilizar com métodos assessores (famosos getters/setters).
- Deixar como público somente os métodos relevantes.

Encapsulamento

- Deixar visível somente o que é importante.
- Uma boa prática é “esconder” todos os atributos da classe e utilizar com métodos assessores (famosos getters/setters).
- Deixar como público somente os métodos relevantes.



Encapsulamento

```
class Carro {
    private String modelo;

    public String getModelo() {
        return modelo;
    }

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

Encapsulamento

```
class Carro {
    private String modelo;

    public String getModelo() {
        return modelo;
    }

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

```
Carro fiesta = new Carro();
fiesta.setModelo("Ford Fiesta");
System.out.println(fiesta.getModelo());
```

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

https://www.javatpoint.com/access-modifiers

Let’s practice;

Exercício #1

- 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

Método Construtor

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

Método Construtor

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

```
class Carro {
    private String modelo;

    public Carro() {
    }

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

Método Construtor

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

```
class Carro {
   private String modelo;

   public Carro() {
   }

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

```
Carro fiesta = new Carro("Ford Fiesta");
System.out.println(fiesta.getModelo());
```

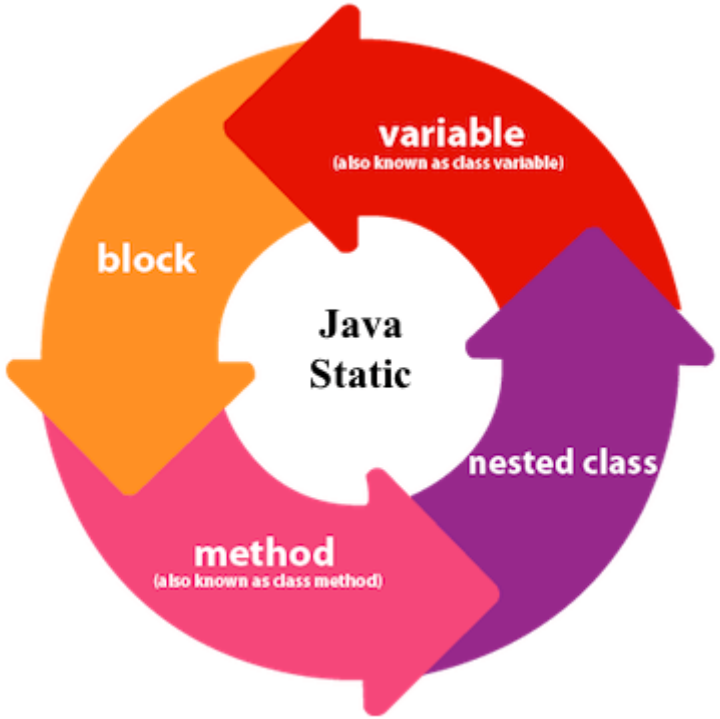
Let’s practice;

Exercício #2

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

Conceito Estático No Java

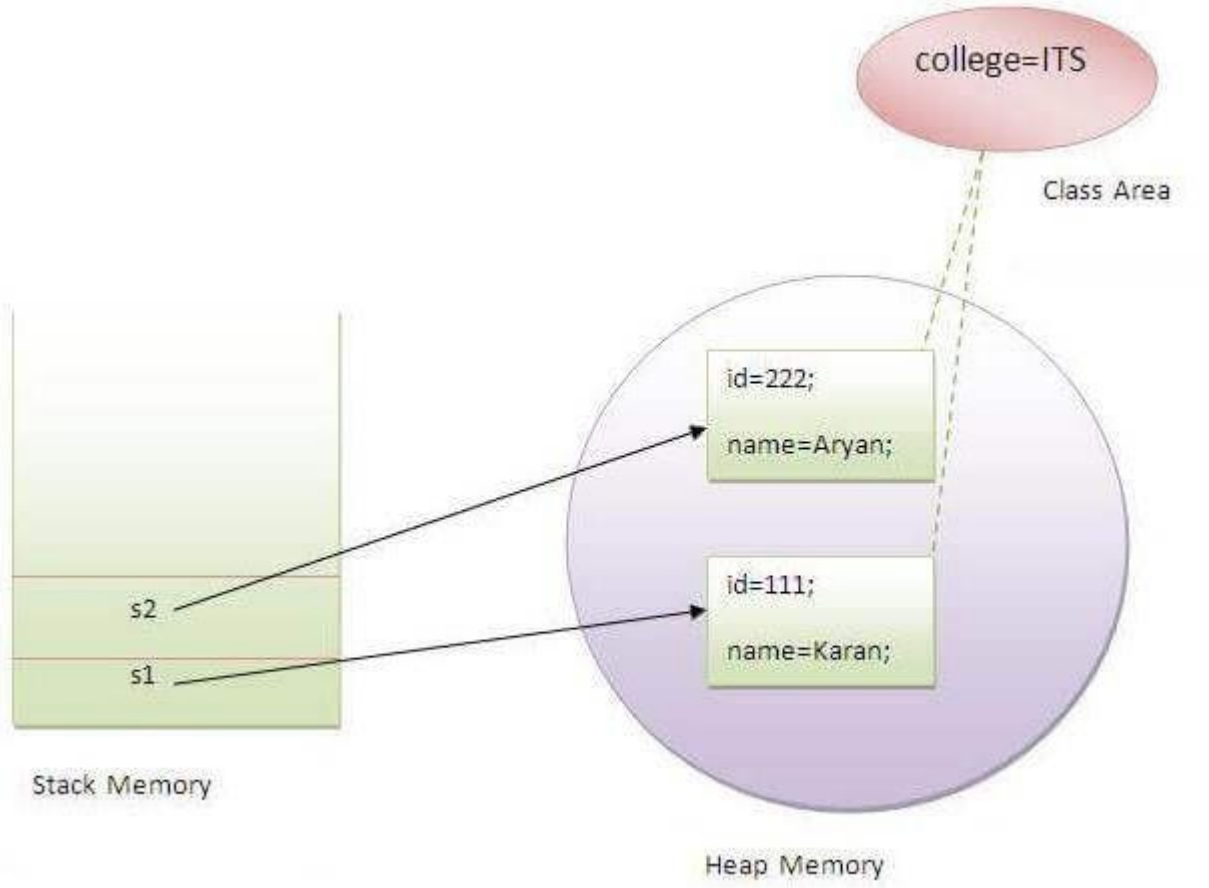
- Tudo que é estático **PERTENCE À CLASSE** e **NÃO AO OBJETO**.



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

Let’s practice;

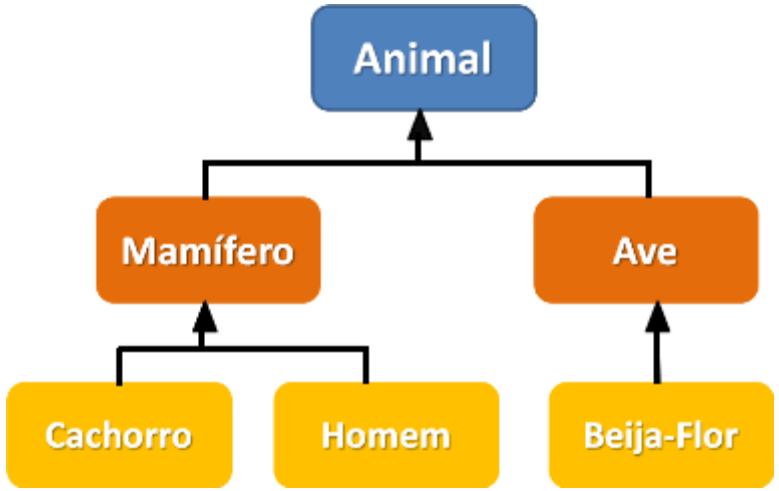
Conceito Estático No Java



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

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

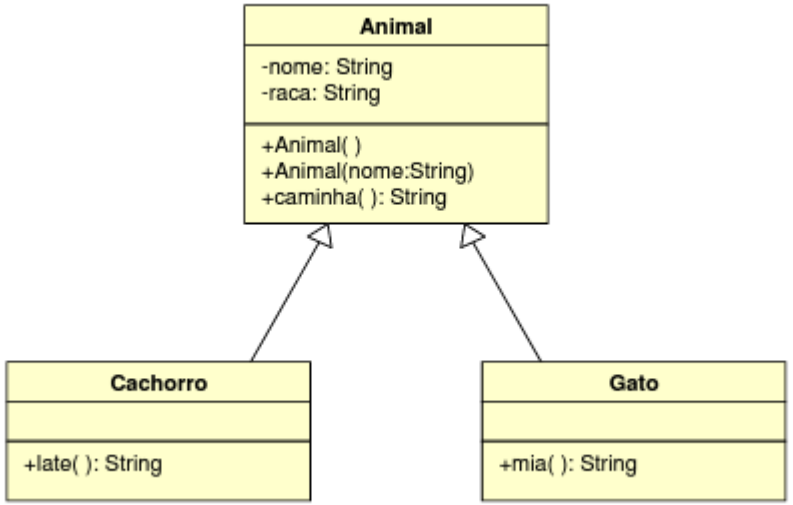
```
abstract class Animal {
    private String nome;

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

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

```
class Mamifero extends Animal {
    public Mamifero(String nome){
        super(nome);
    }
}
```

Let’s practice;

Exercício #3

- Implemente o diagrama de classes abaixo com seus respectivos métodos (A classe animal é abstrata)



- Crie um método main e teste as classes criadas.

Interfaces

- Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

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

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.

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

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

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

Interface de Contrato

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

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

Interface de Contrato

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

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

```
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 interface Funcionario {

}

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;
}
```

Interface de Marcação

```
public interface Funcionario {

}

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;
}
```

```
public class MeuApp {
    public void calculaSalarioDeFuncionario(Funcionario funcionario) {
        if (funcionario instanceof Gerente) {
            //calculo para gerente
        } else if (funcionario instanceof Coordenador) {
            //calculo para coordenador
        } else if (funcionario instanceof Operador) {
            //calculo para operador
        }
    }
}
```

Lets practice;

#Task

- Corrija o homework anterior, na sequência copie e cole a pasta e renomeie para “conta-corrente2”
- “Tune” o programa da aula anterior conforme o novo diagrama: https://lucid.app/lucidchart/7d7ba9e0-5cb4-4595-ba64-05316c1ffd35/view

#Task 2 em grupo

- Crie 2 interfaces relevantes para o seu projeto