02

# Plataforma Java : independência de plataforma, OO e encapsulamento.

## Compare e contraste as funcionalidades e componentes da plataforma Java como: independência de plataforma, orientação a objeto, encapsulamento etc.

### linguagem X plataforma

Quando pensamos na palavra **Java** geralmente pensamos na linguagem, mas na verdade o Java é muito mais que isso. É toda uma plataforma de desenvolvimento, com características bem distintas que permitiram sua adoção em massa por empresas e desenvolvedores.

Quando escrevemos um programa na linguagem Java, para conseguirmos executá-lo, é necessário compilar o código. Uma das diferenças do Java para outras linguagens mais tradicionais é que o resultado da compilação não é exatamente código de máquina, que será interpretado pelo sistema operacional do computador.

O código Java compilado é chamado de **bytecode**. É um arquivo binário que possui a extensão `.class` .

Esse arquivo será lido e interpretado pela **máquina virtual java**, ou simplesmente **JVM**. A JVM é como um computador genérico. Seu papel é interpretar as instruções do bytecode e converter estas instruções para código de máquina, para ser executado pelo sistema operacional nativo do computador.

Existem implementações de JVM para os principais sistemas operacionais, além de diversos dispositivos, como máquinas industriais e até mesmo geladeiras e televisões. Estas implementações são escritas e testadas de maneira que elas sempre interpretem o bytecode da mesma forma, ou seja, não existem diferenças entre as instruções, independente de qual a plataforma nativa. Isso torna a linguagem Java muito poderosa, pois permite que um programa escrito e compilado uma única vez possa ser executado em diversos sistemas operacionais e dispositivos, sem precisar de grandes adaptações. Esse era até mesmo o lema comercial do Java, ::Write Once and Run Anywhere::, uma vez escrito e compilado o código, ele pode ser executado em qualquer ambiente que possua uma virtual machine do Java.

Portanto, note que o termo Java é usado em diversos contextos: a linguagem em si, um pacote de bibliotecas - Java Standard Edition ::JSE::, Java Enterprise Edition ::JEE:: etc. - o compilador e a Java Virtual Machine. Tudo isso é o que chamamos de plataforma Java.

## Orientação a objetos

Java é uma linguagem de alto nível, orientada a objetos. Mas o que significa ser orientada a objetos?

Orientação a objetos é um paradigma de programação, segundo o qual estruturamos nosso código em entidades conhecidas como **objetos**, que possuem dentro de si **dados** na forma de atributos, e **comportamento**, na forma de métodos.

Objetos são como pequenos componentes especializados em executar uma funcionalidade em um sistema. Inclusive, uma das boas práticas da área é que cada objeto tenha uma única responsabilidade. Assim como no mundo real, para executarmos um determinado comportamento, precisamos da interação de

vários objetos/componentes. Por serem componentes independentes e especializados, objetos favorecem o reúso de código e reduzem o custo de manutenção, já que a mudança no comportamento de um objeto será refletida em todos os lugares onde esse objeto é usado.

## Encapsulamento

Um dos conceitos mais básicos e importantes da orientação a objetos é o de encapsulamento, a técnica de esconder atributos e detalhes de implementação de um objeto, para que quando eles sejam alterados tal alteração não tenha que ser replicada em vários lugares do sistema.

Em Java, a forma mais simples de se obter um código encapsulado é declarando os atributos de uma classe como `private`, evitando com isso que outros objetos possam acessar e manipular estes atributos. Caso desejemos que outros objetos tenham acesso a estes dados, liberamos por meio de métodos, controlando como será feita a leitura e escrita das informações.

Vamos ver um exemplo simples, veja a classe `Person`:

```
public class Person{
    public String firstName;
    public String lastName;

    public Person(String firstName, String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

COPIAR CÓDIGO

Desse jeito, qualquer classe no sistema consegue ver estes atributos, e ainda manipulá-los, mudar seus valores. Vamos começar a alterar este código, restringindo o acesso aos atributos usando o modificador de acesso `private` :

```
public class Person{
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

COPIAR CÓDIGO

Ótimo, agora nossos atributos estão encapsulados. Imagine agora que uma outra classe precise imprimir o nome completo de uma pessoa. Vamos implementar esta funcionalidade:

```
//Person.java
public class Person{
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // Expose the full name of the person,
```

```
    // but not how it's stored internally
    public String getFullName(){
        return this.firstName + " "
            + this.lastName;
    }
}

//Test.java

class Test{
    public static void main(String[] args){
        Person p = new Person("Mario", "Amaral");
        System.out.print(p.getFullName());
    }
}
```

COPIAR CÓDIGO

Se, por algum motivo, você resolver não armazenar mais o nome e o sobrenome em campos separados, a seguinte alteração será necessária:

```
//Person.java
public class Person{
    private String fullName;

    public Person(String firstName, String lastName){
        //storing in just one field.
        this.fullName = firstName + " " + this.lastName;
    }

    public String getFullName(){
        return this.fullName;
    }
```

```
}

//Test.java
class Test{
    public static void main(String[] args){
        Person p = new Person("Mario", "Amaral");
        System.out.print(p.getFullName());
    }
}
```

COPIAR CÓDIGO

Repare que mudamos a maneira como armazenamos o nome em nossa classe `Person`, mas não precisamos fazer nenhuma alteração na classe `Test`. Este é um exemplo de um código bem encapsulado. Alterações são realizadas apenas nas classes cuja implementação vai mudar, e não no sistema inteiro. O encapsulamento vai além de variáveis membro privadas. Veremos detalhes de sua implementação mais à frente, em uma seção dedicada ao tema. Nessa seção da prova será cobrada a vantagem e características dessa técnica.