DUPLA - DAVI PRADA HOFFMANN , MARCELO CEOLIN

Encapsulamento

Definições

Encapsulamento é uma prática que consiste na separação dos aspectos internos e externos de um objeto em uma Classe, com modificadores de acesso necessários (public, private, protected) para que o código possa ser salvo do acesso não autorizado pelo mundo exterior.

O encapsulamento possui uma série de vantagens que aumentam a reutilização, flexibilidade e capacidade de manutenção do código.

Flexibilidade: é mais flexível e fácil mudar o código encapsulado com novos requisitos. Por exemplo, se o requisito para ajustar a idade de uma pessoa muda, podemos atualizar facilmente a lógica no método setter ().

Reutilização: o código encapsulado pode ser reutilizado em todo o aplicativo ou em vários aplicativos. Por exemplo, a classe Pessoa pode ser reutilizada sempre que esse tipo de objeto for necessário.

Manutenção: A aplicação é encapsulada em unidades separadas (classes, interfaces, métodos, setters, getters, etc.), por isso é fácil mudar ou atualizar uma parte da aplicação sem afetar outras partes, o que reduz o tempo de manutenção.

Referencias:

https://beginnersbook.com/2013/05/encapsulation-in-java/
http://www.beingjavaguys.com/2013/10/encapsulation-in-java.html

Exemplos

```
Exemplo 1: (https://www.javatpoint.com/encapsulation)
\\Classe Student.java
package com.javatpoint;
```

```
public class Student{
   private String name;

   public String getName(){
        return name;
   }
   public void setName(String name){
        this.name=name
   }
}
\\ Classe Test.java
package com.javatpoint;
class Test{
   public static void main(String[] args){
        Student s=new Student();
        s.setName("vijay");
        System.out.println(s.getName());
   }
}
Resultado do código: vijay
```

Ferramenta

1. Verificar se todos os atributos e métodos estão com seus respectivos modificadores de acesso;
2. Verificar se todos os atributos, caso necessário estão com seus respectivos métodos de atribuição e leitura;
3. Verificar se os valores dos atributos estão sendo validados de acordo com a regra de negócio
4. Verificar o tamanho dos métodos, caso o método estiver muito grande, fragmentá-lo em métodos menores e colocar como private;
5. Verificar se os métodos de regra interno da classe estão legíveis somente para esta classe.

Inspeção

```
> 5 classes devem ser inspecionadas com a ferramenta
> referenciar origem
```

Exemplo 1 - Java.util.Random (https://docs.oracle.com/javase/7/docs/api/java/util/Random.html)

1 - Verdadeiro
2 - Verdadeiro
3 - Verdadeiro
4 - Falso
5 - Verdadeiro

Exemplo 2 - Java.util.Date (https://docs.oracle.com/javase/6/docs/api/java/util/Date.html)

1 - Verdadeiro
2 - Verdadeiro
3 - Verdadeiro
4 - Falso
5 - Verdadeiro

Exemplo 3 (https://github.com/Luiz-Otavio-Dorigon/PontoInteligenteApi/blob/master/src/main/java/br/com/dorigon/pontoeletronico/api/controllers/PessoaJuridicaController.java)

1 - Verdadeiro
2 - Verdadeiro
3 - Verdadeiro
4 - Falso
5 - Verdadeiro

Exemplo 4 - (https://github.com/spring-projects/spring-data-jpa/blob/master/src/main/java/org/springframework/data/jpa/domain/Specifications.java)

1 - Verdadeiro
2 - Verdadeiro
3 - Verdadeiro
4 - Falso
5 - Verdadeiro

Exemplo 5 - (https://github.com/marcondesmacaneiro/aulas-api/blob/master/src/main/java/br/com/marcondesmacaneiro/view/CensoRestController.java)

1 - Verdadeiro
2 - Verdadeiro
3 - Verdadeiro

4 - Falso
5 - Verdadeiro