Encapsulamento - Craitson Luiz Mayer

Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, o mais isolado possível. A idéia é tornar o software mais flexível, fácil de modificar e de criar novas implementações. O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe. É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada. Usamos o nível de acesso mais restritivo, private, que faça sentido para um membro particular. Sempre usamos private, a menos que tenhamos um bom motivo para deixá-lo com outro nível de acesso. Não devemos permitir o acesso público aos membros, exceto em caso de ser constantes. Isso porque membros públicos tendem a nos ligar a uma implementação em particular e limita a nossa flexibilidade em mudar o código. O encapsulamento que é dividido em dois níveis:

- Nível de classe: Quando determinamos o acesso de uma classe inteira que pode ser public ou Package-Private (padrão);
- Nível de membro: Quando determinamos o acesso de atributos ou métodos de uma classe que podem ser public, private, protected ou Package-Private (padrão).

Referencias

https://www.devmedia.com.br/encapsulamento-polimorfismo-heranca-em-java/12991

Exemplos

Exemplo 1:

```
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 - 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/jav a/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/spri ngframework/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