Object-Oriented Analysis and Design using JAVA

B.Tech (CSE/IT) 5th SEM
2020-2021

Lecture-35 Coupling and cohesion in OOAD

Introduction

Cohesion and **Coupling** deal with the quality of an OO design. Generally, good OO design should be loosely coupled and highly cohesive. Lot of the design principles, design patterns which have been created are based on the idea of “Loose coupling and high cohesion”.

The aim of the design should be to make the application:

- easier to develop
- easier to maintain
- easier to add new features
- less Fragile.

Coupling

- Coupling is the degree to which one class knows about another class. Let us consider two classes class **A** and class **B**. If class **A** knows class **B** through its interface only i.e it interacts with class **B** through its API then class **A** and class **B** are said to be loosely coupled.

- If on the other hand class **A** apart from interacting class **B** by means of its interface also interacts through the non-interface stuff of class **B** then they are said to be tightly coupled. Suppose the developer changes the class **B**‘s non-interface part i.e non API stuff then in case of loose coupling class **A** does not breakdown but tight coupling causes the class **A** to break.

- So its always a good OO design principle to use loose coupling between the classes i.e all interactions between the objects in OO system should use the APIs. An aspect of good class and API design is that classes should be well encapsulated.

Tight coupling

- If a class A has some ***public*** **data members** and another class B is accessing these data members ***directly using the dot operator***(which is possible because data members were declared **public**), the two classes are said to be **tightly coupled**.

- Such *tight coupling* between two classes leads to the ***bad designing***. You would ask - Why? Well, here's the explanation.

- Let's say, the class A has a String data member, *name*, which is declared **public** and the class also has *getter and setter* methods that have implemented some checks to make sure - Valid access of the data member, *name* i.e. *it is only accessed when its value is* ***not null***, and Valid setting of the data member, *name* i.e. *it cannot be set to a* ***null*** *value*.

- But these checks implemented in the methods of class A to ensure *valid access* and *valid setting* of its data member, *name*, are *bypassed by its* ***direct access*** by class B i.e. ***tight coupling*** between two classes.

Reference: https://www.decodejava.com/coupling-cohesion-java.htm

```java
//Tight coupling

class A
{
public String name; //public data member of A class


public String getName()
{

//Checking a valid access to "name"
if(name!=null)
        return name;
else
        return"not initiaized";
}


public void setName(String s)
{

//Checking a valid setting to "name"
if (s==null)
{
        System.out.println("You can't initialized name to a null");
}
}

}



class B
{
public static void main(String... ar)
{
A ob= new A();

//Directly setting the value of data member "name" of class A, due to tight coupling betw
ob.name=null;

//Direct access of data member "name" of class A, due to tight coupling between two class
System.out.println("Name is " + ob.name);
}
}
```

Program Analysis

Class **A** has an instance variable, *name*, which is declared **public**.

- Class **A** has two **public** *getter and setter* methods which check for valid access and valid setting of data member - *name*.
- Class **B** creates an object of **A** class and sets the value of its data member, *name*, to **null** and accesses it value **directly by the dot operator** because it was declared **public**.
- Hence, the checks implemented in **getName()** and **setName()** methods of class Names, to *access and set* the data member's value of class A are never called and are rather, *bypassed*. It shows class **A** is *tightly coupled* to class **B**, which is a *bad design*, compromising the data security checks.

Reference: https://www.decodejava.com/coupling-cohesion-java.htm

Loose coupling

A good application designing is creating an application with loosely coupled classes by following proper **encapsulation**, i.e. by declaring data members of a class with the **private** access modifier, which disallows other classes to directly access these data members, forcing them to call ***public* getter, setter methods** to access these *private* data members. Let's understand this by an example -

```java
//Loose coupling example

class A
{

//data member "name" is declared private to implement loose coupling.
private String name;


public String getName()
{

//Checking a valid access to name
if(name!=null)
        return name;
else
        return"not initiaized";
}

public void setName(String s)
{

//Checking a valid setting to name
if (s==null)
{
        System.out.println("You can't initialize name to a null");
}
}

}

class B
{
public static void main(String... ar)
{
A ob= new A();

//Calling setter method, as the direct access of "name" is not possible i.e. loose coupli
ob.setName(null);

//Calling getter method, as the direct access of "name" is not possible i.e. loose coupli
System.out.println("Name is " + ob.getName());
}
}
```

Program Analysis

•

Class **A** has an instance variable, *name*, which is declared **private**.

• Class **A** has two **public** *getter and setter methods* which check for valid access and valid setting of the data member, *name*.

• Class **B** creates an object of class A, calls the **getName()** and **setName()** methods and their ***implemented checks are properly executed*** before the value of instance member, *name*, is accessed or set. It shows class **A** is *loosely coupled* to class **B**, which is a *good programming design*.

High and Low Cohesion

Pictorial view of high cohesion and low cohesion:

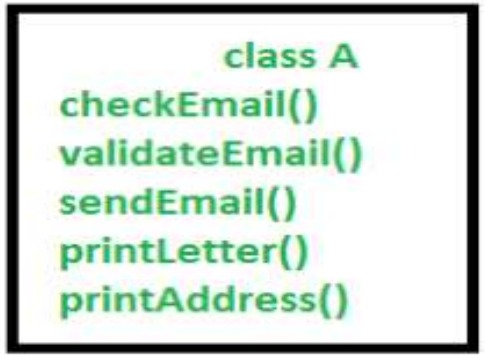

Fig: Low cohesion

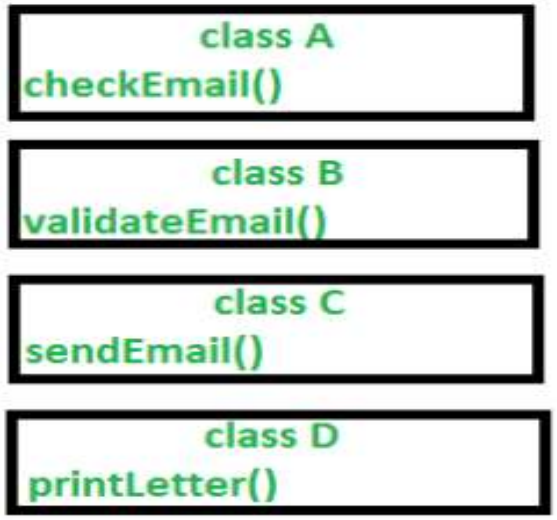

Fig: High cohesion

Explanation : In the above image, we can see that in low cohesion only one class is responsible to execute lots of job which are not in common which reduces the chance of re-usability and maintenance. But in high cohesion there is a separate class for all the jobs to execute a specific job, which result better usability and maintenance.

https://www.geeksforgeeks.org/cohesion-in-java/

High and Low Cohesion

Difference between high cohesion and low cohesion:

- High cohesion is when you have a class that does a well defined job. Low cohesion is when a class does a lot of jobs that don’t have much in common.

- High cohesion gives us better maintaining facility and Low cohesion results in monolithic classes that are difficult to maintain, understand and reduces re-usability

https://www.geeksforgeeks.org/cohesion-in-java/

Thank You