Encapsulation and Acess Modifiers

W.A E.M Weerasinghe

Encapsulation

- Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.
- We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
- The Java Bean class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

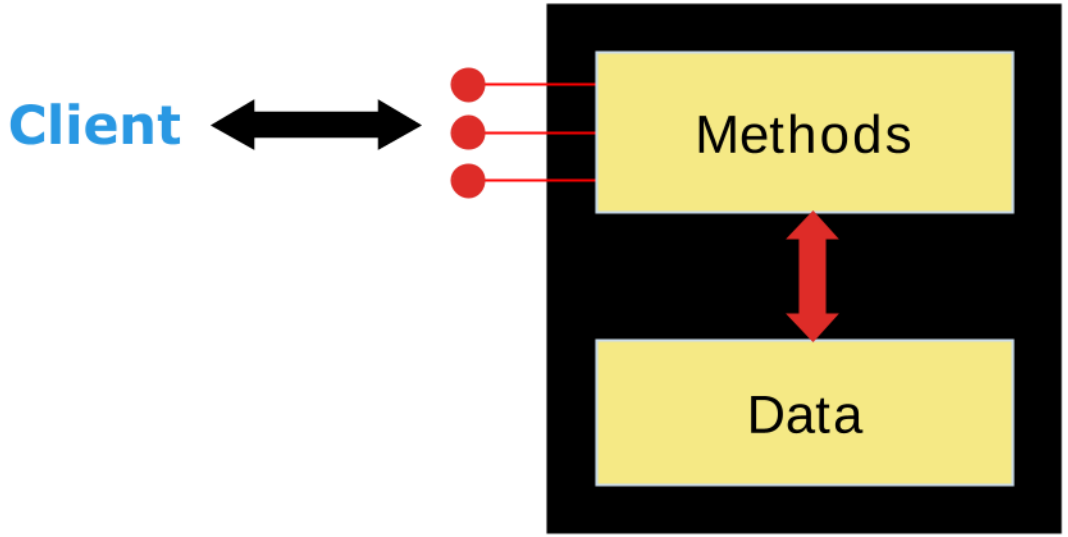
- By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.
- It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is **easy to test**. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Encapsulation

- One object (called the *client*) may use another object for the *services* it provides (server)
- The client of an object may request its services (call its methods), but it should not have to be aware of how those services are accomplished
- Only object's method can change of the object 'state(It's Variable)
- That is, an object should be *self-governing*

Encapsulation

- An encapsulated object can be assumed of as a *black box* -- its inner workings are hidden from the client
- The client invokes the interface methods of the object, which manages the instance data



Access Modifiers in Java

- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.
- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Access Modifiers in Java

There are four types of Java access modifiers:

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Private

The private access modifier is accessible only within the class.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Default

- The default modifier is accessible only within package.
- It cannot be accessed from outside the package. ***It provides more accessibility than private. But it is more restrictive than protected, and public***

Default

```java
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

```java
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

- In this example, created two packages pack and mypack.
- accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.
- the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifier.

Protected

```java
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```java
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

- In this example, created the two packages pack and mypack.
- The A class of pack package is public, so can be accessed from outside the package.
- But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Public

```java
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```java
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Understanding Java Access Modifiers

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