**Encapsulation in Java with example**

I have already discussed a brief about encapsulation while discussing [**OOPs concepts**](http://beginnersbook.com/2013/04/oops-concepts/). I have also covered [**polymorphism**](http://beginnersbook.com/2013/03/polymorphism-in-java/) and [**inheritance**](http://beginnersbook.com/2013/05/java-inheritance-types/) in separate articles. In this post we will discuss **encapsulation in java with examples**.

**What is encapsulation?**

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. However if we setup public getter and setter methods to update (for e.g. void setSSN(int ssn))and read (for e.g.  int getSSN()) the private data fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That’s why encapsulation is known as **data hiding.**Lets see an example to understand this concept better.

public class EncapsulationDemo{

private int ssn;

private String empName;

private int empAge;

//Getter and Setter methods

public int getEmpSSN(){

return ssn;

}

public String getEmpName(){

return empName;

}

public int getEmpAge(){

return empAge;

}

public void setEmpAge(int newValue){

empAge = newValue;

}

public void setEmpName(String newValue){

empName = newValue;

}

public void setEmpSSN(int newValue){

ssn = newValue;

}

}

public class EncapsTest{

public static void main(String args[]){

EncapsulationDemo obj = new EncapsulationDemo();

obj.setEmpName("Mario");

obj.setEmpAge(32);

obj.setEmpSSN(112233);

System.out.println("Employee Name: " + obj.getEmpName());

System.out.println("Employee SSN: " + obj.getEmpSSN());

System.out.println("Employee Age: " + obj.getEmpAge());

}

}

**Output:**

Employee Name: Mario

Employee SSN: 112233

Employee Age: 32

In above example all the three data members (or data fields) are private which cannot be accessed directly. These fields can be accessed via public methods only. Fields empName, ssn and empAge are made hidden data fields using encapsulation technique of OOPs.

**Advantages of encapsulation:**

1. It improves maintainability and flexibility and re-usability: for e.g. In the above code the implementation code of void setEmpName(String name) and String getEmpName() can be changed at any point of time. Since the implementation is purely hidden for outside classes they would still be accessing the private field empName using the same methods (setEmpName(String name) and getEmpName()). Hence the code can be maintained at any point of time without breaking the classes that uses the code. This improves the re-usability of the underlying class.
2. The fields can be made read-only (If we don’t define setter methods in the class) or write-only (If we don’t define the getter methods in the class). For e.g. If we have a field(or variable) which doesn’t need to change at any cost then we simply define the variable as private and instead of set and get both we just need to define the get method for that variable. Since the set method is not present there is no way an outside class can modify the value of that field.
3. User would not be knowing what is going on behind the scene. They would only be knowing that to update a field call set method and to read a field call get method but what these set and get methods are doing is purely hidden from them.

Encapsulation is also known as “**data Hiding**”.

1. Objects encapsulate data and implementation details. To the outside world, an object is a black box that exhibits a certain behavior.
2. The behavior of this object is what which is useful for the external world or other objects.
3. An object exposes its behavior by means of public methods or functions.
4. The set of functions an object exposes to other objects or external world acts as the interface of the object.