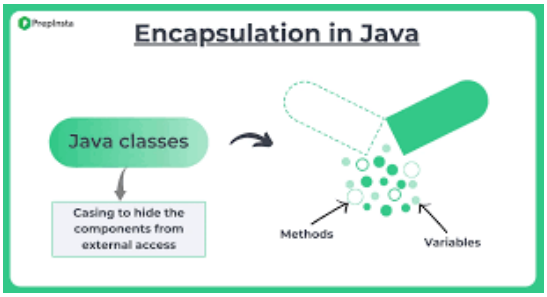
Encapsulation Concepts in JAVA [Assignment Questions/Answers] 09 Feb'23

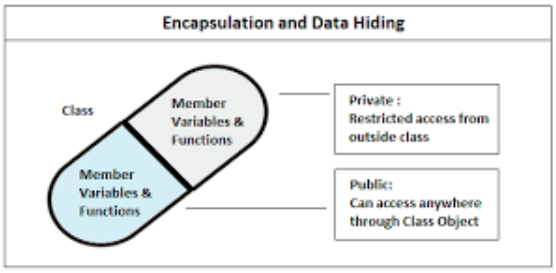
ANS 01:

Encapsulation:



Binding of data and corresponding methods into a single unit is called "Encapsulation".
If any JAVA class follows data hiding and abstraction then such class is referred as "Encapsulation".

Encapsulation = Data Hiding + Data Abstraction



Data hiding means hiding the internal data within the class to prevent its direct access from outside the class. If we talk about data encapsulation so, Data encapsulation hides the private methods and class data parts, whereas Data hiding only hides class data components.

ANS 02:

Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.
Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of fields and methods inside a single class. It prevents outer classes from accessing and changing fields and methods of a class. This also helps to achieve data hiding.

ANS 03:

Getter and Setter are methods used to protect your data and make your code more secure. Getter returns the value (accessors), it returns the value of data type int, String, double, float, etc. For the program's convenience, getter starts with the word "get" followed by the variable name.

While Setter sets or updates the value (mutators). It sets the value for any variable used in a class's programs. and starts with the word "set" followed by the variable name. Getter and Setter make the programmer convenient in setting and getting the value for a particular data type. In both getter and setter, the first letter of the variable should be capital.

EX:

```
// Java Program to Illustrate Getter and Setter

// Importing input output classes
import java.io.*;

// Class 1
// Helper class
class GetSet {

        // Member variable of this class
        private String name;

        // Method 1 - Getter
        public String getName() { return name; }

        // Method 2 - Setter
        public void setName(String N)
        {

                // This keyword refers to current instance itself
                this.name = N;
        }
}

// Class 2
// Main class
class Launch {

        // Main driver method
        public static void main(String[] args)
        {
                // Creating an object of class 1 in main() method
                GetSet obj = new GetSet();

                // Setting the name by calling setter method
                obj.setName("PWians");
                // Getting the name by calling getter method
                System.out.println(obj.getName());
        }
}
```

ANS 04:

THIS Keyword:

The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

this keyword would always point to current object, and this variable would hold the address the active object present in the heap memory.

EX:

```
class Student

{
            private String name;
             private Integer id;
             private String address;


             Student(String name,Integer id, String address){

                          this.name = name;
                          this.id = id;
                          this.address = address;
             }

             public void display()
            {

                          System.out.println("Name is    :: "+name);
                          System.out.println("Id is      :: "+id);
                          System.out.println("Address is :: "+address);
             }

}
class Demo

{
             public static void main(String[] args)

             {

                          Student std = new Student("sachin",10,"MI”);

                          std.display();
             }
}
```

O/P:

Name is :: sachin

Id is :: 10

Address is :: MI

ANS 05:

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.

ANS 06:

Encapsulation in Java can be achieved by:

* Declaring the variables of a class as private.
* Providing public setter and getter methods to modify and view the variables values.

Now, let’s look at the code to get a better understanding of encapsulation:

```
public class Student {
private String name;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
}
class Test{
public static void main(String[] args) {
```

```
Student s=new Student();
s.setName("Harry Potter");
System.out.println(s.getName());
}
}
```

As you can see in the above code, I have created a class Student which has a private variable name. Next, I have created a getter and setter to get and set the name of a student. With the help of these methods, any class which wishes to access the name variable has to do it using these getter and setter methods.

Now let's see one more example and understand Encapsulation in depth. In this example, the Car class has two fields –name and topSpeed. Here, both are declared as private, meaning they can not be accessed directly outside the class. We have some getter and setter methods like getName, setName, setTopSpeed etc., and they are declared as public. These methods are exposed to "outsiders" and can be used to change and retrieve data from the Car object. We have one method to set the top speed of the vehicle and two getter methods to retrieve the max speed value either in MPH or KMHt. So basically, this is what encapsulation does – it hides the implementation and gives us the values we want. Now, let's look at the code below.

```
package PW;
public class Car {
private String name;
private double topSpeed;
public Car() {}
public String getName(){
return name;
}
public void setName(String name){
this.name= name;
}
public void setTopSpeed(double speedMPH){
 topSpeed = speedMPH;
}
public double getTopSpeedMPH(){
return topSpeed;
}
public double getTopSpeedKMH(){
return topSpeed*1.609344;
}
}
```

Here, the main program creates a Car object with a given name and uses the setter method to store the top speed for this instance. By doing this, we can easily get the speed in MPH or KMH without caring about how speed is converted in the Car class.

```
package PW;
public class Example{
public static void main(String args[])
Car car =new Car();
car.setName("Mustang GT 4.8-litre V8");
car.setTopSpeed(201);
System.out.println(car.getName()+ " top speed in MPH is " + car.getTopSpeedMPH());
System.out.println(car.getName() + " top speed in KMH is " + car.getTopSpeedKMH());
```

So, this is how Encapsulation can be achieved in Java. Now, let's move further and see why do we need Encapsulation.