PW SKILLS
JAVA With DSA & System Design
Assignment – Encapsulation in Java
Day - 20

1. **What is Encapsulation in Java? Why is it called Data hiding?**
   Ans: Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.
   Data Hiding: It is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. The user will only know that we are passing the values to a setter method and variables are getting initialized with that value.

2. **What are the important features of Encapsulation?**
   Ans: Features of Encapsulation:
   - Data Hiding:
   - Increased Flexibility:
   - Reusability:
   - Testing code is easy

3. **What are getter and setter methods in Java? Explain with an example.**
   Ans: Getter and Setter methods:
   - **Getter** and **Setter** are those methods in Java that are used to retrieve the value of a data member and update or set the value of a data member respectively.
   - Getters are also called **accessors** and Setters are also called **mutators**.
   - Getter and setter in java help in providing data hiding to prevent direct unauthorized access to the variables.
   - By using getter and setter in java we are indirectly accessing the private variables. Once defined, getters and setters can be accessed inside the main method.

   Example:
```
class Student{
   private int age;
   private String name;
   public void show()
   {
      System.out.println(name +" "+ age);
   }


   public int getAge() {
      return age;
   }


   public void setAge(int age) {
      this.age = age;
   }
```

```
    public String getName() {
        return name;
    }


    public void setName(String name) {
        this.name = name;
    }
}

class SetterAndGetter{
    public static void main(String[] args) {
        Student obj1 = new Student();
        Student obj2 = new Student();

        obj1.setAge(18);
        obj2.setAge(20);
        obj1.setName("Shadab");
        obj2.setName("Siddiqui");

        int Student1 = obj1.getAge();
        String Student1n = obj1.getName();
        System.out.println(Student1n);
        System.out.println(Student1);

    }
}
```

Output:

```
Shadab
18
```

4. **What is the use of 'this' keyword? Explain with an example.**

Ans: In Java, **this** keyword is a reference variable that refers to the current object.

Usage of 'this' keyword"

Here are the 6 uses of java this keyword.

- this can be used to refer to the current class instance variable.
- this can be used to invoke current class method (implicitly)
- this () can be used to invoke the current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as an argument in the constructor call.
- this can be used to return the current class instance from the method.

Example:

```
        class Test
    {
      int a;
      int b;

     public  void Test(int a, int b)
      {
          this.a = a;
          this.b = b;
      }

      void display()
      {
          //Displaying value of variables a and b
          System.out.println("a = " + a + "  b = " + b);
      }

      public static void main(String[] args)
      {
          Test object = new Test(10, 20);
          object.display();
      }
```

Output:

```
a = 10 b = 20
```

5. **What is the advantage of Encapsulation?**
   Ans: advantage of Encapsulation:
   - **Better Control -** Encapsulation provides ultimate control over the data members and data methods inside the class.
   - **Getter and Setter -** The standard IDEs provide in-built support for 'Getter and Setter' methods, which increases the programming pace.
   - **Security -** Encapsulation prevents access to data members and data methods by any external classes. The encapsulation process improves the security of the encapsulated data.
   - **Flexibility -** Changes made to one part of the code can be successfully implemented without affecting any other part of the code.
   - **Data Hiding -** Data hiding is a procedure done to avoid access to the data members and data methods and their logical implementation. Data hiding can be done by using the access specifiers.
   - **Easy to Reuse**
     Encapsulation enables you to easily change the methods, reuse the code, and execute new requirements in your program.
6. **How to achieve encapsulation in Java? Give an example.**
   Ans: There are a few ways by which we can achieve encapsulation in java. Some of them are:
   a. Declaring the class variables as **private** so that they are inaccessible from outside the scope of the class.
   Example:

```
class Employee {
  // private field of class
  private String name;

  // getter method
  public String getName() {
    return name;
  }

  //setter method
  public void setName(String n) {
    this.name = n;
  }
}
```

b. Designing **getter** and **setter** methods for the class and using them accordingly. The presence of getter and setter methods in a class define what type of class it is.
   Example:

```
class Employee {

  // private variables
  private String name;
  private int age;
  private int salary;

  //public getter and setter methods for each variables
  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public int getSalary() {
    return salary;
  }
  public void setSalary(int salary) {
    this.salary = salary;
  }
}

public class Main {

  public static void main(String[] args) {
    //object of class Employee
    Employee newObj = new Employee();
    newObj.setName("James");
    newObj.setAge(25);
    System.out.println(
      "Name of employee: " +
      newObj.getName() +
      "\n" +
      "Age of employee: " +
      newObj.getAge()
    );
  }
}
```