Encapsulation in Java

- Encapsulation is a key concept in object-oriented programming.
- It means keeping the data (fields) and the code (methods) that works on the data bundled together inside one class.
- Data Hiding restricts access to certain parts of the data, exposing only what is necessary.
- Encapsulation helps in making your code more organized, secure, and easier to maintain!

Why Use Encapsulation?

1. **Organized Code**: It keeps related fields and methods together, making the code easier to understand and manage.
2. **Control Over Data**: You can control how the data is accessed and modified by using methods to get and set the values.
3. **Independent Components**: It helps in breaking down a system into smaller, manageable pieces that can be developed and tested separately.
4. **Data Hiding**: It hides the details of how the data is stored and managed, exposing only what is necessary.

Example 1: Basic Encapsulation

```
class Area {
  int length;
  int breadth;

  Area(int length, int breadth) {
    this.length = length;
    this.breadth = breadth;
  }

  public void getArea() {
    int area = length * breadth;
    System.out.println("Area: " + area);
  }
}

class Main {
  public static void main(String[] args) {
    Area rectangle = new Area(5, 6);
    rectangle.getArea();
  }       }
```

Example 2: Encapsulation with Data Hiding

```
class Person {
  private int age;

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    if (age >= 0) {
      this.age = age;
    }
  }
}

class Main {
  public static void main(String[] args) {
    Person p1 = new Person();
    p1.setAge(24);
    System.out.println("My age is " + p1.getAge());
  }
}
```

Here, age is a private field, which means it cannot be accessed directly from outside the Person class. Instead, we use the getAge() and setAge() methods to access and modify it. This is known as data hiding because we are controlling how the data is accessed and updated, preventing unauthorized changes.

How Access Modifiers Work in Java?

In Java, access modifiers are keywords that set the accessibility of classes, methods, and other members. These keywords determine whether a field or method in a class can be used or invoked by another method in another class or sub-class.

Access modifiers may also be used to restrict access.

In Java, we have four types of access modifiers, which are:

- Default
- Public
- Private
- Protected

Default Access Modifier

The default access modifier is also called package-private. You use it to make all members within the same package visible, but they can be accessed only within the same package.

Note that when no access modifier is specified or declared for a class, method, or data member, it automatically takes the default access modifier.

Example:

```
class SampleClass
{
    void output()
        {
            System.out.println("Hello World! This is an Introduction to OOP.");
        }
}
class Main
{
    public static void main(String args[])
        {
            SampleClass obj = new SampleClass();
            obj.output();
        }
}
```

Public Access Modifier

The public access modifier allows a class, a method, or a data field to be accessible from any class or package in a Java program. The public access modifier is accessible within the package as well as outside the package.

In general, a public access modifier does not restrict the entity at all.

Example:

```
public class Car {
    // public variable
    public int tireCount;

    // public method
    public void display() {
        System.out.println("I am a Car.");
        System.out.println("I have " + tireCount + " tires.");
    }
}

public class Main {
    public static void main( String[] args ) {
        // accessing the car class
        Car car = new Car();

        // accessing the public variable
        car.tireCount = 4;
        // accessing the public method
        car.display();
    }
}
```

Private Access Modifier

The **private access modifier** is an access modifier that has the lowest accessibility level. This means that the methods and fields declared as private are not accessible outside the class. They are accessible only within the class which has these private entities as its members.

You may also note that the private entities are not visible even to the subclasses of the class.

Example:

```
class SampleClass
{

   private String activity;
}

public class Main
{

   public static void main(String[] main)
   {

      SampleClass task = new SampleClass();

      task.activity = "We are learning the core concepts of OOP.";
   }
}
```

- When we run the above program, we will get an error message. This is because we are trying to access the private variable and field from another class.
- So, the best way to access these private variables is to use the getter and setter methods.
- Getters and setters are used to protect your data, particularly when creating classes. When we create a getter method for each instance variable, the method returns its value while a setter method sets its value.

Example:

```
class SampleClass
{

   private String task;

   // This is the getter method.
   public String getTask()
   {

      return this.task;
   }

   // This is the setter method.
   public void setTask(String task)
   {

      this.task= task;
   }
}

public class Main
{

   public static void main(String[] main)
   {

      SampleClass task = new SampleClass();

      // We want to access the private variable using the getter and setter.

      task.setTask("We are learning the core concepts of OOP.");

      System.out.println(task.getTask());
   }
}
```

Protected Access Modifier

When methods and data members are declared protected, we can access them within the same package as well as from subclasses.

We can also say that the protected access modifier is somehow similar to the default access modifier. It is just that it has one exception, which is its visibility in subclasses.

Note: Classes cannot be declared protected. This access modifier is generally used in a parent-child relationship.

Example:

```
package learners;

public class Multiplication
{
  protected int multiplyTwoNumbers(int a, int b)
  {
        return a*b;
  }
}
```

```
package javalearners;

import learners.*;

class Test extends Multiplication
{
  public static void main(String args[])
  {
        Test obj = new Test();

        System.out.println(obj.multiplyTwoNumbers(2, 4));
  }
}
```