Java Encapsulation

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**.

Example 1: Java Encapsulation

```
class Area {

  // fields to calculate area
  int length;
  int breadth;

  // constructor to initialize values
  Area(int length, int breadth) {
    this.length = length;
    this.breadth = breadth;
  }

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

class Main {
  public static void main(String[] args) {

    // create object of Area
    // pass value of length and breadth
    Area rectangle = new Area(5, 6);
    rectangle.getArea();
  }
}
```

Output

Area: 30

In the above example, we have created a class named Area. The main purpose of this class is to calculate the area.

To calculate an area, we need two variables: length and breadth and a method: `getArea()`. Hence, we bundled these fields and methods inside a single class.

Here, the fields and methods can be accessed from other classes as well. Hence, this is not **data hiding**.

This is only **encapsulation**. We are just keeping similar codes together.

Note: People often consider encapsulation as data hiding, but that's not entirely true.

Encapsulation refers to the bundling of related fields and methods together. This can be used to achieve data hiding. Encapsulation in itself is not data hiding.

Why Encapsulation?

- In Java, encapsulation helps us to keep related fields and methods together, which makes our code cleaner and easy to read.
- It helps to control the values of our data fields. For example,

```
class Person {
  private int age;

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

  Here, we are making the age variable `private` and applying logic inside the `setAge()` method. Now, age cannot be negative.

- The getter and setter methods provide **read-only** or **write-only** access to our class fields. For example,

```
getName()  // provides read-only access
setName() // provides write-only access
```

- It helps to decouple components of a system. For example, we can encapsulate code into multiple bundles.

  These decoupled components (bundle) can be developed, tested, and debugged independently and concurrently. And, any changes in a particular component do not have any effect on other components.

- We can also achieve data hiding using encapsulation. In the above example, if we change the length and breadth variable into private, then the access to these fields is restricted.

  And, they are kept hidden from outer classes. This is called **data hiding**.

Data Hiding

Data hiding is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding.

We can use access modifiers to achieve data hiding. For example,

Example 2: Data hiding using the private specifier

```
class Person {

  // private field
  private int age;

  // getter method
  public int getAge() {
    return age;
  }

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

class Main {
  public static void main(String[] args) {

    // create an object of Person
    Person p1 = new Person();

    // change age using setter
    p1.setAge(24);

    // access age using getter
    System.out.println("My age is " + p1.getAge());
  }
}
```

Output

My age is 24

In the above example, we have a `private` field age. Since it is `private`, it cannot be accessed from outside the class.

In order to access age, we have used `public` methods: `getAge()` and `setAge()`. These methods are called getter and setter methods.

Making age private allowed us to restrict unauthorized access from outside the class. This is **data hiding**.

If we try to access the age field from the Main class, we will get an error.

```
// error: age has private access in Person
p1.age = 24;
```