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.

•Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.

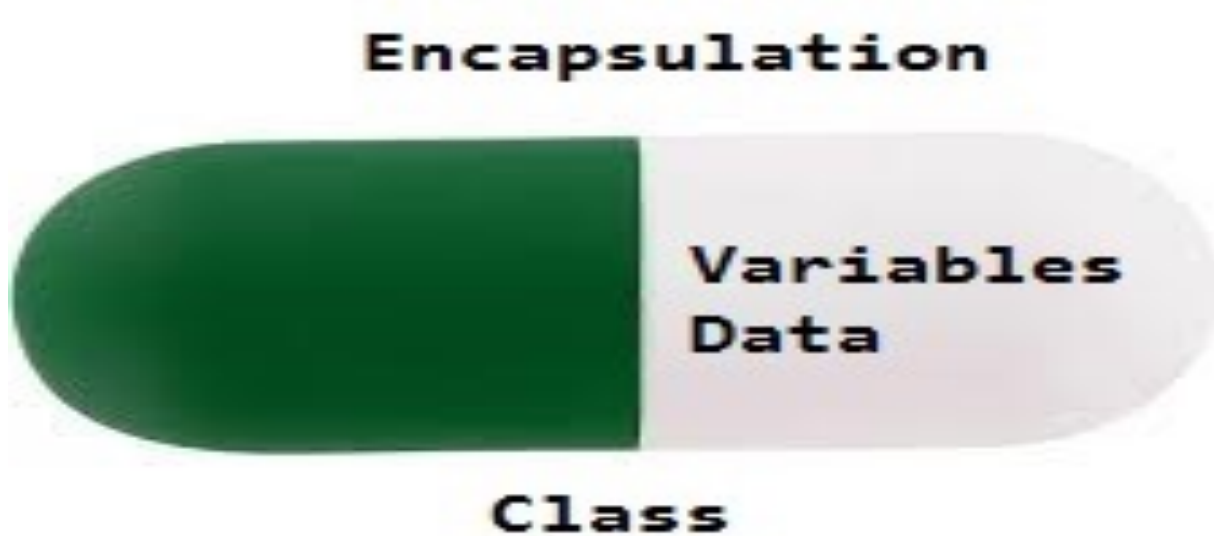
•As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a **combination of data-hiding and abstraction**.

•Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

•It is more defined with the setter and getter method.

- **Advantages of Encapsulation**:

- **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.
- **Increased Flexibility:** We can make the variables of the class read-only or write-only depending on our requirement. If we wish to make the variables read-only then we have to omit the setter methods like setName(), setAge(), etc. from the above program or if we wish to make the variables write-only then we have to omit the get methods like getName(), getAge(), etc. from the above program
- **Reusability:** Encapsulation also improves the re-usability and is easy to change with new requirements.
- **Testing code is easy:** Encapsulated code is easy to test for unit testing.


Encapsulation
Variables
Data
Class

```
// fields to calculate area
class 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) {

        Area rectangle = new Area(2, 16);
        rectangle.getArea();
}
}
```

The program to access variables of the class EncapsulateDemo is shown below:

```
// Java program to demonstrate encapsulation
class Encapsulate {
        // private variables declared
        // these can only be accessed by
        // public methods of class
        private String gName;
        private int gRoll;
        private int gAge;

        // get method for age to access
        // private variable gAge
        public int getAge() { return gAge; }

        // get method for name to access
```

```
// private variable gName
public String getName() { return gName; }

// get method for roll to access
// private variable gRoll
public int getRoll() { return gRoll; }

// set method for age to access
// private variable gage
public void setAge(int newAge) { gAge = newAge; }

// set method for name to access
// private variable gName
public void setName(String newName)
{
```

```
                gName = newName;
        }

        // set method for roll to access
        // private variable gRoll
        public void setRoll(int newRoll) { gRoll = newRoll; }
}

public class TestEncapsulation {
        public static void main(String[] args)
        {
                Encapsulate obj = new Encapsulate();

                // setting values of the variables
                obj.setName("Harsh");
```

```
                obj.setAge(19);
                obj.setRoll(51);

                // Displaying values of the variables
                System.out.println(" name: " + obj.getName());
                System.out.println(" age: " + obj.getAge());
                System.out.println("roll: " + obj.getRoll());

                // Direct access of gRoll is not possible
                // due to encapsulation
                // System.out.println(" roll: " +
                // obj.gName);
        }
}
```

- **Output**

  name: Harsh
- age: 19
- roll: 51
- In the above program, the class Encapsulate is encapsulated as the variables are declared as private. The get methods like getAge() , getName() , getRoll() are set as public, these methods are used to access these variables. The setter methods like setName(), setAge(), setRoll() are also declared as public and are used to set the values of the variables.

- Getter setter method example-

```
class Name {

    private int age; // Private is using to hide the data

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

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

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

		Name n1 = new Name();

		n1.setAge(19);

		System.out.println("The age of the person is: "
							+ n1.getAge());
	}
}
```

- **Advantages of Encapsulation in Java:**

1. Improves security of an object's internal state by hiding it from the outside world.
2. Increases modularity and maintainability by making it easier to change the implementation without affecting other parts of the code.
3. Enables data abstraction, allowing objects to be treated as a single unit.
4. Allows for easy addition of new methods and fields without affecting the existing code.
5. Supports the object-oriented principle of information hiding, making it easier to change the implementation without affecting the rest of the code.

- **Disadvantages of Encapsulation in Java:**

1. Can lead to increased complexity, especially if not used properly.
2. Can make it more difficult to understand how the system works.
3. May limit the flexibility of the implementation.

- Get and Set
- You learned from the previous chapter that private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.
- The get method returns the variable value, and the set method sets the value.
- Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in upper case:
-

- Example
- public class Person {
- private String name; // private = restricted access //
- Getter
- public String getName()
- { return name;
- } //
- Setter
- public void setName(String newName) {
- this.name = newName; } }

- Example explained
- The get method returns the value of the variable name.
- The set method takes a parameter (newName) and assigns it to the name variable. The this keyword is used to refer to the current object.
- However, as the name variable is declared as private, we **cannot** access it from outside this class:

- we use the getName() and setName() methods to access and update the variable:
- Example
- public class Main {
- public static void main(String[] args) {
- Person myObj = new Person();
- myObj.setName("John"); // Set the value of the name variable to "John"
- System.out.println(myObj.getName());
- } }
-