Encapsulation in Java:-

25 October 2025 16:16

Every Object in this world has a most important component which has been given security so no one directly access.

Encapsulation is the process of providing security to the most important component of object by:-

- Preventing direct access to internal data.
- Providing controlled access through method.

Advantage of Encapsulation

Data Protection(Data Hiding):- The internal data of an object is hidden from the outside world, preventing direct access.

By making variables private, we prevent unauthorized or accidental change.

Data Integrity:- with the help of setter and getter methods we can ensure that the correct data is assigned to the variable.

Reusability:- with the help of encapsulation the reusability of the code increases also changing of the code becomes easier when we need to add new feature in the code.

Flexibility and maintainability:- we can changes the internal implementation of a class without affecting other parts of the program.

Better code organization:- Encapsulation groups related variables and method into one unit. This make the code clear and easier to understand.

Improved security:- sensitive data is hidden, reducing the risk of misuse or unintended interference.

Implementation of Encapsulation in Java

In java, encapsulation is implemented by declaring instance variables as private, restricting direct access. Public getter method retrieve variable values, while setter method modify them, enabling controlled access. This approach the class to enforce data validation and maintain a consistent internal state, enhancing security and flexibility.

- In encapsulation, the variable or data of a class are hidden from any other class and can be accessed only through any member of its function of its own class.

- A private class can hide its members or method from the end user using abstraction to hide implementation details, by combining data hiding and abstraction.

Setters and Getters in Java

In Java, getter and setter are two conventional methods that are used for retrieving and updating value of a variable.
As we cannot access the private members of the class directly outside the class. But we can give protected access by providing setters and getters.
So, a setter is a method that updates value of a variable. And a getter is a method that reads

value of a variable.
Getter and setter are also known as accessor and mutator in Java.

Setter method

- Used to modify the value of a private variable.
- It has a public access modifier and take the parameter.
- Return type should be void.
- It should have some arguments.
- It is used to write value

Getter method

- Used to retrieve the value of a private variable.
- It has a public access modifier and return type should not be void.
- It has used to read the value.
- It should not have any arguments.

Why setter and getter?

So far, setter and getter methods protect a variable's value from unexpected changes by outside world - the caller code. When a variable is hidden by private modifier and can be accessed only through getter and setter, it is encapsulated. Encapsulation is one of the fundamental principles in object oriented programming (OOP), thus implementing getter and setter is one of ways to enforce encapsulation in program's code.

```
class Student {
    // Private variables - data hiding
    private String name;
    private int age;

    // Setter method (write)
    public void setName(String name) {
        this.name = name;
    }

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

    // Setter for age with validation
    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        } else {
            System.out.println("Invalid age!");
        }
    }
    // Getter for age
    public int getAge() {
        return age;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.setName("Vishal");
        s1.setAge(21);

        System.out.println("Name: " + s1.getName());
        System.out.println("Age: " + s1.getAge());
    }
}
```

Shadowing problems and this keyword

Shadowing happen when a local variable(like method or constructor parameter) has the same name as an instance variable. In this case, the local variable "shadows" (hides) the instance variable inside the current scope.

```
class Student {
    String name;

    Student(String name) {
        // Here, 'name' (local parameter) shadows the instance variable 'name'
        name = name;  // This assigns parameter to itself, not instance variable!
    }

    void show() {
        System.out.println("Name: " + name);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Vishal");
        s1.show();  // Output: Name: null
    }
}
```

This keyword:-

This keyword in java is a reference variable that refers to the current object of class. It is mainly used inside constructor and method to distinguish b/w instance variable and parameters or to call other constructor or method of the same class.

```
class Student {
    String name;

    Student(String name) {
        this.name = name;  // 'this.name' → instance variable; 'name' → parameter
    }

    void show() {
        System.out.println("Name: " + this.name);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Vishal");
        s1.show();  // Output: Name: Vishal
    }
}
```