**1. Introduction to Encapsulation:**

* **Definition:** Encapsulation is one of the four fundamental OOP concepts (along with inheritance, polymorphism, and abstraction). It is the mechanism of wrapping the **data (variables)** and **code (methods)** together as a single unit. It restricts direct access to certain components of an object, making the object’s state secure and allowing only controlled access through public methods.
* **Real-Life Example:** Think of a **capsule** (medicine) where the drug is enclosed inside a protective shell. The capsule controls the release of the drug, protecting the user from the raw ingredients. Similarly, in Java, encapsulation hides the internal state of an object and provides controlled access through methods.

**2. Encapsulation in Java:**

* **Encapsulation in Java is achieved by:**
  1. **Declaring variables as private**: This prevents direct access to the variables.
  2. **Providing public getter and setter methods**: These methods allow controlled access to the variables, so you can control what values are set or retrieved.

**Example:**

java

Copy code

class Person {

// Private variables - encapsulated data

private String name;

private int age;

// Public getter method for name

public String getName() {

return name;

}

// Public setter method for name

public void setName(String name) {

this.name = name;

}

// Public getter method for age

public int getAge() {

return age;

}

// Public setter method for age

public void setAge(int age) {

// Only allow valid ages

if (age > 0) {

this.age = age;

} else {

System.out.println("Invalid age");

}

}

}

public class Main {

public static void main(String[] args) {

Person person = new Person();

// Set values using setter methods

person.setName("John");

person.setAge(30);

// Get values using getter methods

System.out.println("Name: " + person.getName());

System.out.println("Age: " + person.getAge());

}

}

**Explanation:**

* **Private Variables:** The name and age variables are declared as private. This ensures that they cannot be directly accessed or modified from outside the Person class.
* **Getter and Setter Methods:** The getName(), setName(), getAge(), and setAge() methods are public, providing controlled access to the private variables.

**3. Advantages of Encapsulation:**

1. **Data Hiding:**
   * By making variables private, we hide the implementation details of a class. External classes can only interact with the object's data through public methods, which ensures that the internal state is protected.
2. **Controlled Access:**
   * Getter and setter methods provide control over how the variables are accessed and modified. For example, you can add validation logic to ensure that only valid data is assigned (like the age check in the example).
3. **Improved Maintainability:**
   * Encapsulation allows changes to be made to the internal implementation of a class without affecting other parts of the program that use that class. This improves code maintainability.
4. **Flexibility and Reusability:**
   * Since the internal implementation is hidden, encapsulated code can be reused without requiring changes to the rest of the codebase.

**4. Encapsulation vs. Abstraction:**

* **Encapsulation** hides the **data** by restricting access through private variables and public methods.
* **Abstraction** hides the **implementation details** and only shows what an object can do (methods).

**Example:**

* Abstraction: Knowing how to use a TV remote (interface) without understanding how the TV works internally.
* Encapsulation: The TV itself, which hides all the complex hardware inside and only exposes buttons or methods to interact with it.

**5. Practical Exercise:**

* **Task 1:** Create a BankAccount class with private variables accountNumber, balance, and accountHolderName. Implement getter and setter methods for these variables, with a check to ensure the balance cannot be negative.
* **Task 2:** Create a Student class with private variables name, rollNumber, and marks. Implement getter and setter methods and ensure the marks cannot exceed 100.

**6. Common Encapsulation Scenarios:**

1. **Encapsulation in Real-World Applications:**
   * In any application, encapsulation is used to secure sensitive data. For example, in a banking application, a user’s bank balance and transaction details are encapsulated to prevent unauthorized access.

**7. Summary:**

* **Encapsulation** is the process of binding the data (variables) and the code (methods) that manipulates that data into a single unit, such as a class.
* It is achieved by making the class variables private and providing public getter and setter methods.
* Encapsulation provides **data hiding**, **controlled access**, and **better maintainability** in Java applications.