14-Encapsulation

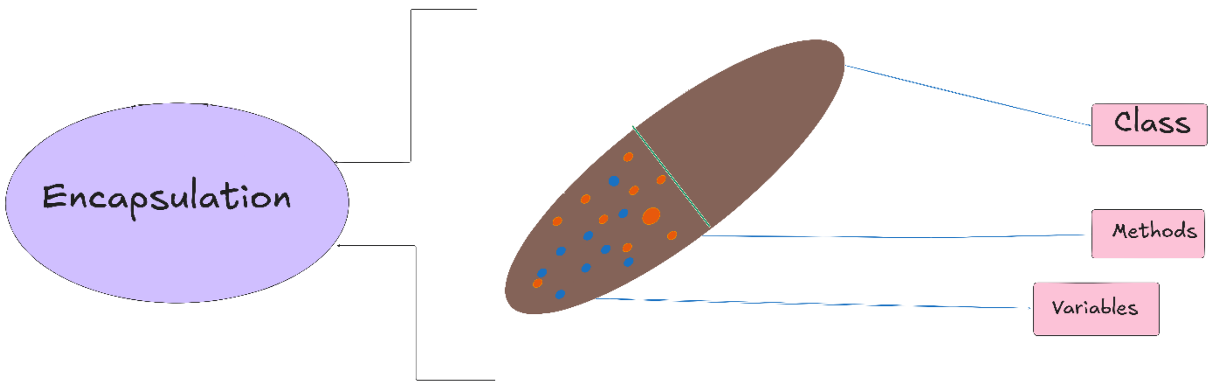
What is Encapsulation?

Encapsulation is a fundamental concept in object-oriented programming (OOP) in Java. It involves hiding the internal details of an object and exposing only the necessary parts to the outside world. This is achieved by:

- Declaring the class variables as private, which makes them accessible only within the class.
- Providing public methods (getters and setters) to access and modify these private variables.

Analogy:

Imagine a medicine capsule 💊. The internal ingredients are hidden from view. Similarly, encapsulation hides the internal state of an object and only exposes certain methods to interact with that state.



Advantages of Encapsulation

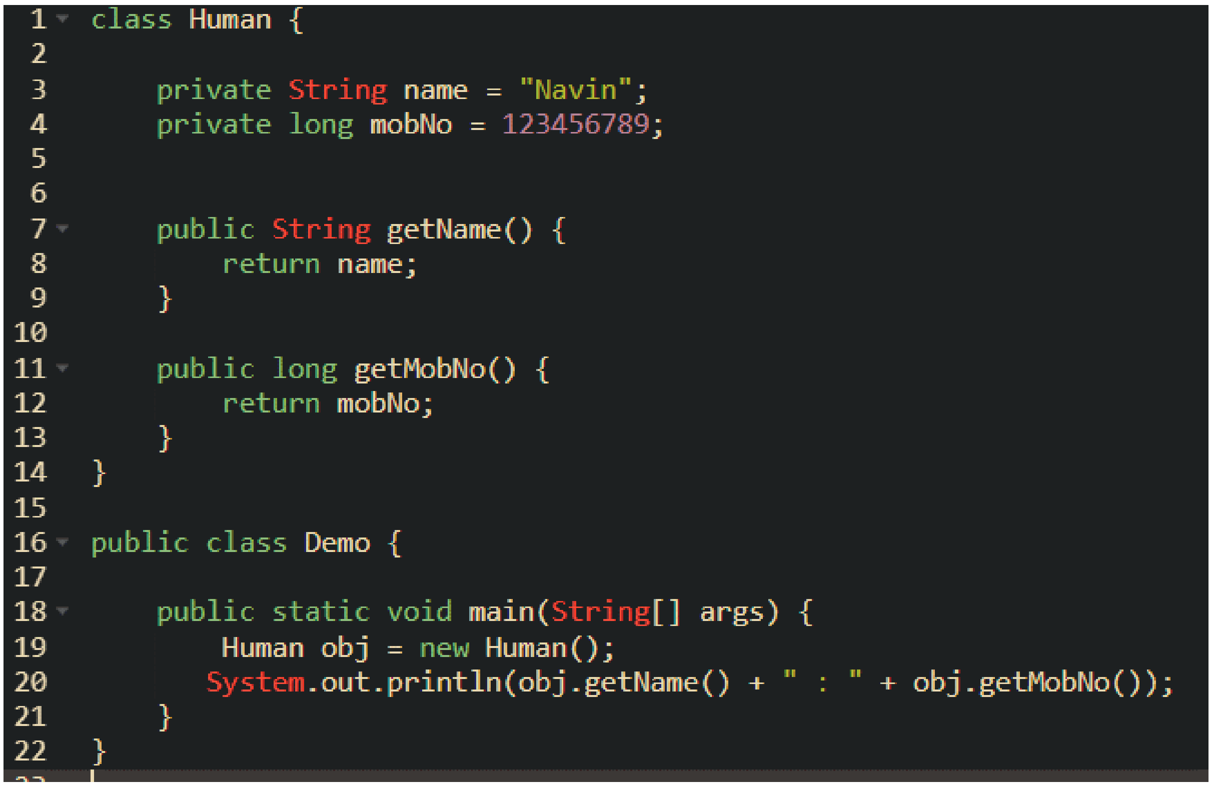
1. **Data Hiding:**
   - Hides the internal implementation details from the user.
   - Users can interact with the object through public methods without knowing the internal workings.
2. **Increased Flexibility:**
   - Variables can be made read-only or write-only depending on needs.
   - For example, omitting setter methods makes variables read-only.
3. **Reusability:**
   - Encourages code reusability and makes it easier to update and maintain.

4. **Easier Testing:**
    - Encapsulated code is easier to test and debug, especially in unit testing.
5. **Freedom for Programmers:**
    - Programmers can change the internal implementation without affecting external code, as long as the public interface remains the same.

Disadvantages of Encapsulation

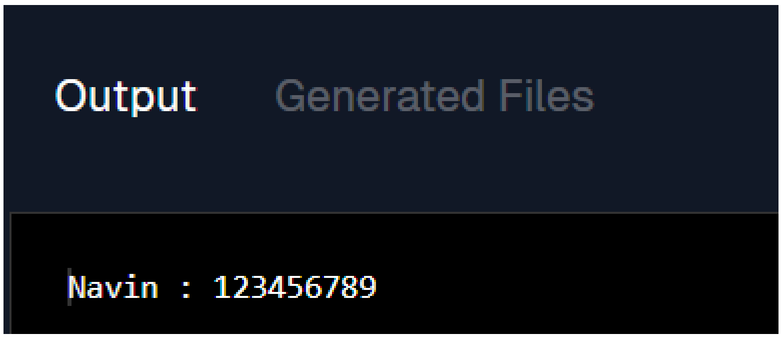
1. **Increased Complexity:**
    - May add complexity, especially if not implemented properly.
2. **Understanding Difficulty:**
    - Can make the system harder to understand for others who are reading or maintaining the code.
3. **Limited Flexibility:**
    - Can limit flexibility in certain scenarios where direct access to internal data might be beneficial.

Example 1: Basic Encapsulation



```java
class Human {

    private String name = "Navin";
    private long mobNo = 123456789;


    public String getName() {
        return name;
    }

    public long getMobNo() {
        return mobNo;
    }
}

public class Demo {

    public static void main(String[] args) {
        Human obj = new Human();
        System.out.println(obj.getName() + " : " + obj.getMobNo());
    }
}
```

Output:

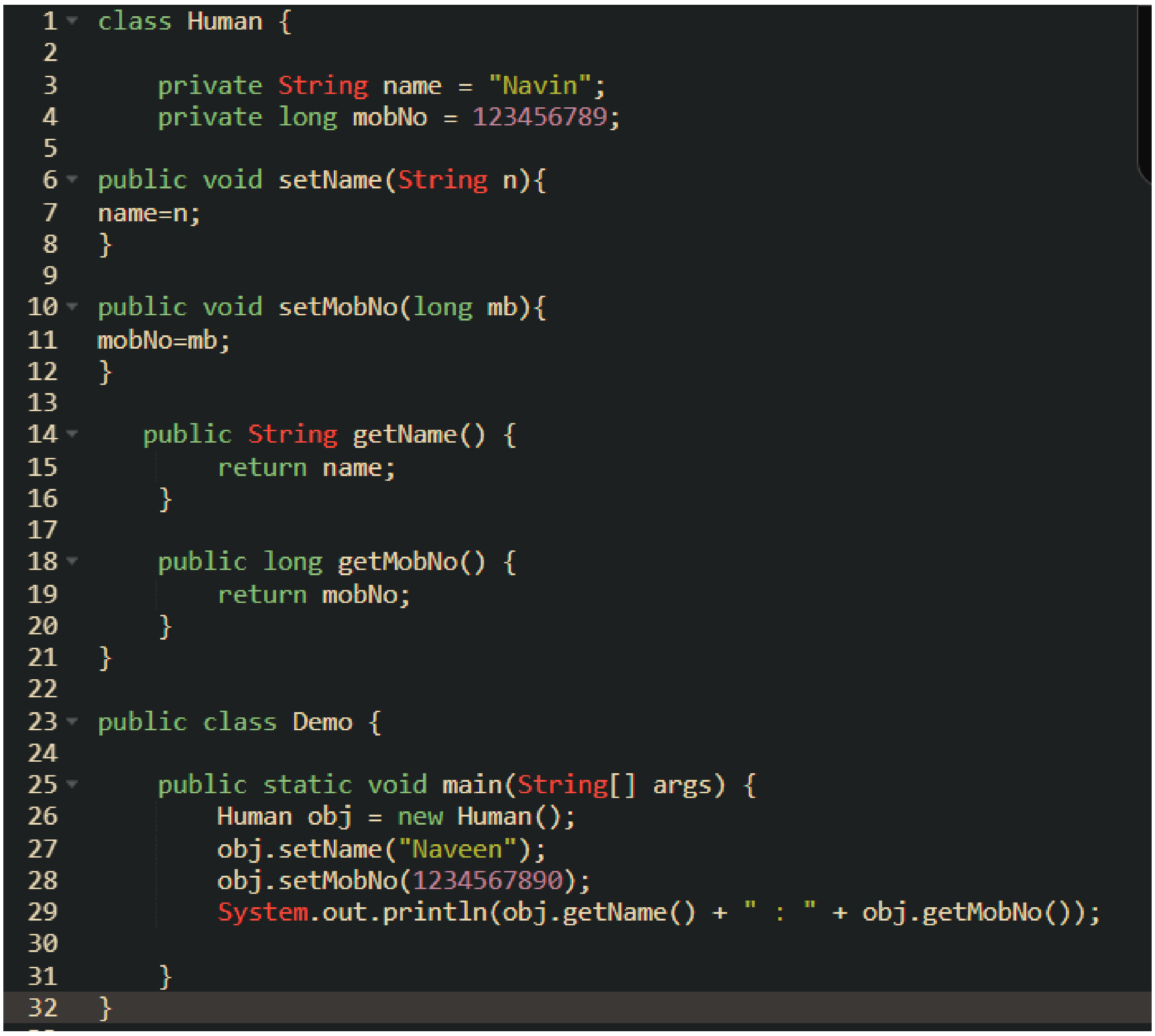


```
Navin : 123456789
```

Explanation:

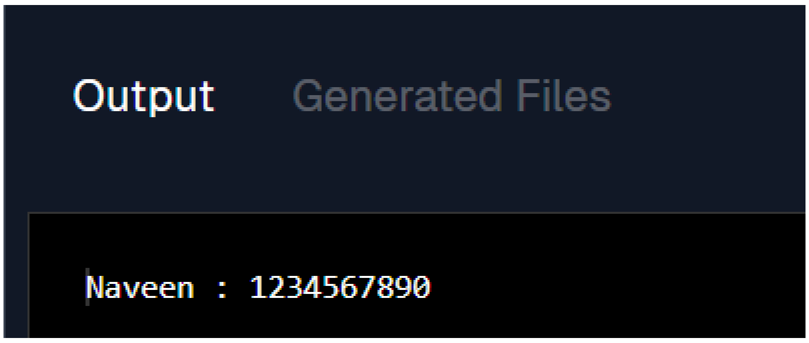
- **Private Variables:** name and mobNo are private, so they cannot be accessed directly from outside the Human class.
- **Getter Methods:** getName() and getMobNo() are used to retrieve these values.
- **Show Method:** Uses the getter methods to display the private data.

Example 2: Using Setters



```java
class Human {

    private String name = "Navin";
    private long mobNo = 123456789;

public void setName(String n){
name=n;
}

public void setMobNo(long mb){
mobNo=mb;
}

   public String getName() {
        return name;
    }

    public long getMobNo() {
        return mobNo;
    }
}

public class Demo {

    public static void main(String[] args) {
        Human obj = new Human();
        obj.setName("Naveen");
        obj.setMobNo(1234567890);
        System.out.println(obj.getName() + " : " + obj.getMobNo());

    }
}
```

Output:



```
Naveen : 1234567890
```

Explanation:

- **Setter Methods:** setName() and setMobNo() allow changing the private variables.
- **Updated Values:** The main method demonstrates setting new values for name and mobNo and then retrieving and displaying these values.

Summary

Encapsulation is crucial for protecting the internal state of an object and providing a controlled way of accessing and modifying that state. By using private variables and public getter and setter methods, you can control how data is accessed and updated, which enhances the flexibility, reusability, and maintainability of your code.