**Session: Understanding Encapsulation in Java**

**Objective:**

* To understand the concept of Encapsulation in Object-Oriented Programming (OOP).
* To learn how to implement encapsulation using access modifiers in Java.

**1. What is Encapsulation?**

**Definition**: Encapsulation is one of the fundamental principles of Object-Oriented Programming. It is the mechanism of wrapping the data (variables) and the code (methods) together into a single unit called a **class**. The main goal of encapsulation is to **protect the data** by restricting access to certain components, ensuring that data can only be accessed in specific ways.

**Key Concepts**:

* **Data Hiding**: Keeping the internal state of an object hidden from the outside world.
* **Access Control**: Allowing access to data through well-defined methods.

**2. Why is Encapsulation Important?**

* **Data Security**: By restricting direct access to the data and using getters and setters, you can ensure that the data is only changed in specific ways.
* **Flexibility**: Encapsulation allows you to modify the internal implementation without affecting the external code that uses the object.
* **Improved Maintainability**: It ensures that the implementation details are hidden, so changes are easier to manage.

**3. How Does Encapsulation Work in Java?**

**Access Modifiers**: Java provides four access modifiers to control access to data:

* **private**: The member is accessible only within the class.
* **default (no modifier)**: The member is accessible only within the same package.
* **protected**: The member is accessible within the same package and by subclasses.
* **public**: The member is accessible from any other class.

**4. Example of Encapsulation in Java**

**Step 1: Create a Class with Private Variables**

java

public class Person {

// Private variables (data members)

private String name;

private int age;

// Getter method for name

public String getName() {

return name;

}

// Setter method for name

public void setName(String name) {

this.name = name;

}

// Getter method for age

public int getAge() {

return age;

}

// Setter method for age

public void setAge(int age) {

if (age > 0) {

this.age = age;

} else {

System.out.println("Age cannot be negative or zero.");

}

}

}

**Step 2: Using the Class with Encapsulation**

java

public class Main {

public static void main(String[] args) {

// Create an instance of Person

Person person = new Person();

// Set values using setter methods

person.setName("John Doe");

person.setAge(25);

// Get values using getter methods

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

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

}

}

**5. Benefits of Using Getters and Setters**

* **Control over Data**: You can validate data before setting it or take actions whenever data is retrieved.
* **Abstraction**: The internal representation of data is hidden from the user, making it easier to modify later without breaking code that uses the class.

**6. Real-World Analogy**

Imagine a **bank account**:

* **Private data**: The balance of the account.
* **Public methods**: Deposit, Withdraw, Check Balance — these methods interact with the private balance data. You can only change the balance using these methods, not directly.

**7. Exercise: Implementing Encapsulation**

**Task**: Create a class called **Car** with private fields:

* private String brand
* private int speed
* Create getter and setter methods for both fields.

Make sure that the speed can only be set to a positive value.

**8. Conclusion**

* Encapsulation helps protect data integrity by controlling how data is accessed and modified.
* It ensures that objects interact with their data only through well-defined interfaces (methods).
* By using encapsulation, you can design robust and flexible applications.

**Q&A Session**

* Encourage the students to ask questions regarding the use of encapsulation in real-world scenarios.
* Discuss any doubts they have on accessing and modifying private data in Java.