### **Q1. What is Encapsulation in Java? Why is it called Data Hiding?**

**Answer**: Encapsulation is a concept in object-oriented programming that bundles the data (variables) and methods (functions) that operate on the data into a single unit, called a class. In Java, encapsulation is achieved by declaring the class variables as private and providing public getter and setter methods to access and update the values of private variables.

It's called **data hiding** because it restricts direct access to some of the object's components, which means the internal representation of an object is hidden from the outside. Only authorized methods (getters and setters) are allowed to interact with the data, providing controlled access.

### **Q2. What are the Important Features of Encapsulation?**

**Answer**: The important features of encapsulation include:

1. **Data Hiding**: Protects object integrity by preventing unauthorized access and modification of data.
2. **Controlled Access**: Provides controlled access to the data through getter and setter methods.
3. **Improved Maintainability**: Encapsulation makes code more modular and maintainable, as it keeps the data and methods in a single unit.
4. **Enhanced Security**: Protects data integrity and restricts misuse by hiding implementation details.
5. **Increased Flexibility**: Allows changes to implementation without impacting external code relying on it.

### **Q3. What are Getter and Setter Methods in Java? Explain with an Example.**

**Answer**: **Getter** and **Setter** methods are used to access and modify private variables of a class. The getter method retrieves the value of a private variable, and the setter method sets or updates the value.

**Example**:

java

class Student {

private String name; // private variable

// Getter method

public String getName() {

return name;

}

// Setter method

public void setName(String name) {

this.name = name;

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student();

student.setName("Alice"); // Setting name using setter

System.out.println(student.getName()); // Accessing name using getter

}

}

In this example, the Student class has a private variable name. The getName() method allows retrieving the value of name, while setName() allows setting it.

### **Q4. What is the Use of the this Keyword? Explain with an Example.**

**Answer**: The this keyword in Java is a reference to the current instance of the class. It's used to differentiate between class attributes and parameters with the same name, or to pass the current instance to another method.

**Example**:

java

class Employee {

private String name;

// Constructor with a parameter having the same name as the class variable

public Employee(String name) {

this.name = name; // `this.name` refers to the class variable, `name` refers to the parameter

}

public String getName() {

return this.name;

}

}

public class Main {

public static void main(String[] args) {

Employee emp = new Employee("John");

System.out.println(emp.getName()); // Outputs: John

}

}

In this example, this.name refers to the class variable name, while name in Employee(String name) refers to the constructor parameter.

### **Q5. What is the Advantage of Encapsulation?**

**Answer**: The advantages of encapsulation include:

1. **Improves Security**: Protects sensitive data by restricting access.
2. **Increases Flexibility**: Allows changing the internal implementation without affecting external code.
3. **Simplifies Maintenance**: Modularity of code makes it easier to modify and maintain.
4. **Reduces Complexity**: Hides complex implementation details from other classes.
5. **Enhances Code Reusability**: Encapsulated classes are easy to reuse across different parts of an application.

### **Q6. How to Achieve Encapsulation in Java? Give an Example.**

**Answer**: To achieve encapsulation in Java:

1. Declare class variables as private.
2. Provide public getter and setter methods to access and modify the private variables.

**Example**:

java

class Car {

private String model; // private variable

private int year;

// Getter and Setter for model

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

// Getter and Setter for year

public int getYear() {

return year;

}

public void setYear(int year) {

if (year > 1885) { // Basic validation

this.year = year;

} else {

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

}

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car();

car.setModel("Tesla Model S"); // Setting model

car.setYear(2023); // Setting year

System.out.println("Car Model: " + car.getModel());

System.out.println("Manufacturing Year: " + car.getYear());

}

}

In this example, the Car class encapsulates the variables model and year. The private variables cannot be accessed directly but can be accessed or modified using getModel, setModel, getYear, and setYear methods.