1. What is Encapsulation in Java? Why is it called Data Hiding?

Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP) in Java. It refers to the bundling of data (variables) and methods (functions) that operate on the data into a single unit, called a class. Encapsulation restricts direct access to some of an object's components, which is a way of data hiding.

Encapsulation is called **Data Hiding** because it hides the internal state of the object from the outside world. By using access modifiers (private, protected, and public), you can control what parts of a class's data and methods are accessible or modifiable from outside the class.

2. What are the Important Features of Encapsulation?

- **Control Access to Data**: Encapsulation allows you to control the level of access to the data members (variables) of a class.
- **Modularity**: It separates the internal implementation of a class from its external interface.
- **Maintainability**: It helps in maintaining and modifying the code independently without affecting other parts of the code.
- **Flexibility and Extensibility**: It provides the flexibility to change the implementation details without affecting the external code.
- **Improved Security**: It safeguards the internal state of an object by preventing unauthorized or unintended modifications.

3. What are Getter and Setter Methods in Java? Explain with an Example

Getter and Setter methods are used to access and update the private variables of a class from outside the class. A getter method retrieves the value of an attribute, while a setter method updates the value of an attribute.

```
public class Person {
    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) { // validation to ensure age is positive
            this.age = age;
        }
    }
}
```

4. What is the Use of `this` Keyword? Explain with an Example

The `this` keyword in Java is a reference to the current object whose method or constructor is being invoked. It can be used to differentiate between instance variables and parameters or to invoke other constructors in the same class.

```
public class Employee {
    private String name;
    private double salary;

    // Constructor
    public Employee(String name, double salary) {
        this.name = name; // 'this' refers to the current object's instance variable
        this.salary = salary;
    }

    // Method to display employee details
    public void display() {
        System.out.println("Name: " + this.name); // 'this' is optional here
        System.out.println("Salary: " + this.salary);
    }

    public static void main(String[] args) {
        Employee emp = new Employee("John", 50000);
        emp.display();
    }
}
```

5. What is the Advantage of Encapsulation?

- **Data Protection**: Encapsulation protects an object's state by preventing external code from accessing or modifying it directly.
- **Simplified Maintenance**: It helps in maintaining the code as internal changes to encapsulated code do not affect other parts of the program.
- **Improved Code Reusability**: Encapsulated classes can be reused across different programs without modifications.
- **Enhanced Flexibility**: It allows changing the internal implementation of a class without affecting external code that relies on the class.
- **Increased Modularity**: It separates the internal workings of a class from its external interface, promoting better modularity in the program.

6. How to Achieve Encapsulation in Java? Give an Example

Encapsulation in Java can be achieved by:

1. Declaring the class variables as private.
2. Providing public getter and setter methods to access and update the value of private variables.

```
public class Student {
    private String studentName;
    private int studentId;

    // Getter method for studentName
    public String getStudentName() {
        return studentName;
    }

    // Setter method for studentName
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }

    // Getter method for studentId
    public int getStudentId() {
        return studentId;
    }

    // Setter method for studentId
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }

    public static void main(String[] args) {
        Student student = new Student();
        student.setStudentName("Alice");
        student.setStudentId(101);

        System.out.println("Student Name: " + student.getStudentName());
        System.out.println("Student ID: " + student.getStudentId());
    }
}
```

In this example, the `Student` class encapsulates the `studentName` and `studentId` variables. These variables are private and can only be accessed or modified using the public getter and setter methods.