# Encapsulation Assignment Questions

## Assignment Questions:

1. What is Encapsulation in Java? Why is it called data hiding?

Ans → Encapsulation is a fundamental principle of object-oriented programming (OOP) in Java that refers to the mechanism of bundling data (variables) and methods (functions) that manipulate the data into a single unit, called a class. The purpose of encapsulation is to hide the implementation details of a class from the outside world and to provide a well-defined interface for interacting with the class.

Encapsulation is called data hiding because it allows the internal state of an object to be hidden from the outside world. In other words, the data members of a class are declared as private, which means they can only be accessed within the class. This prevents the data from being modified or accessed directly by other classes or methods, which can help prevent errors and ensure data integrity.

2. What are the important feature of Encapsulation?

Ans →

a. Data Hiding: Encapsulation hides the implementation details of a class from the outside world by making the data members of a class private. This prevents direct access to the data and ensures that the data is accessed and modified through public methods.
b. Access Modifiers: Access modifiers (public, private, protected) are used to control the access to the data members and methods of a class. Private access modifier is used to hide the data members from outside access, while public and protected access modifiers are used to provide controlled access to the data members and methods of a class.
c. Class Design: Encapsulation allows for better class design by promoting loose coupling between classes. It allows classes to be designed in a modular way, where each class is responsible for its own data and behavior.
d. Encapsulation and Inheritance: Encapsulation is a key feature in inheritance as it allows subclasses to inherit the data and behavior of the superclass without exposing the implementation details of the superclass.
e. Encapsulation and Polymorphism: Encapsulation is also important for polymorphism as it allows for a consistent interface to be used across different implementations of a class. By encapsulating the data and behavior of a class, subclasses can override or extend the behavior of the superclass without affecting other parts of the program.

3. What are getter and setter features in Java Explain with an

example?

Ans → Getter and setter methods are a type of public method used to access and modify the private data members of a class. These methods are used to implement encapsulation, which hides the internal state of an object from the outside world.

```
public class Person {
  private String name;
  private int age;

  // Getter methods
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
  // Setter methods
  public void setName(String name) {
    this.name = name;
  }
  public void setAge(int age) {
    this.age = age;
  }
}
```

In the above example, we have defined a class called Person with private data members name and age. To access and modify these private data members, we have defined public getter and setter methods.
The getter methods getName() and getAge() are used to retrieve the values of the private data members name and age, respectively. These methods return the values of the corresponding data members.
The setter methods setName(String name) and setAge(int age) are used to set the values of the private data members name and age, respectively. These methods take a parameter and set the value of the corresponding data member to the value of the parameter.

For example, to set the name of a person object p to "John", we can use the setName() method as follows:

```
p.setName("John");
```

And to retrieve the age of the person object p, we can use the getAge() method as follows:

```
int age = p.getAge();
```

## 4. What is the true use of this keyword explain with an example?

Ans → The ‘this’ keyword in Java is used to refer to the current object within a method or constructor of a class. It is often used to distinguish between the instance variables of a class and the local variables or parameters of a method with the same name.

Here's an example to demonstrate the use of this keyword:

```
public class Person {
  private String name;
```

```
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to print the name and age of the person object
    public void printInfo() {
        System.out.println("Name: " + this.name);
        System.out.println("Age: " + this.age);
    }
}
```

In the above example, we have defined a class called Person with private data members name and age. We have also defined a constructor to initialize these data members and a method printInfo() to print the values of these data members.

Within the constructor, we are using this keyword to refer to the instance variables name and age. This is necessary because the constructor parameters name and age have the same names as the instance variables. By using this, we are specifying that we want to assign the parameter values to the instance variables.

Within the printInfo() method, we are again using this keyword to refer to the instance variables name and age. This is not strictly necessary in this case, but it is a common convention to use this to refer to instance variables, especially when they are accessed within a method or constructor.

## 5. What is the advantage of Encapsulation?

Ans → The advantage of encapsulation in Java is that it provides better control over the data and behavior of a class, leading to more secure and robust code. Here are some of the main advantages of encapsulation:

a. Data Hiding: Encapsulation allows the implementation details of a class to be hidden from the outside world. By making the data members of a class private, direct access to the data is prevented and the data can only be accessed and modified through public methods. This improves security and reduces the risk of accidental modification or corruption of the data.
b. Controlled Access: Encapsulation allows controlled access to the data and behavior of a class. By using access modifiers (public, private, protected), the visibility of data members and methods can be controlled. This helps to prevent unwanted modifications to the data and ensures that the data is accessed and modified through well-defined public methods.
c. Better Class Design: Encapsulation allows for better class design by promoting loose coupling between classes. By encapsulating the data and behavior of a class, it can be designed in a modular way, with each class responsible for its own data and behavior. This promotes code reuse, reduces dependencies between classes, and makes the code easier to maintain and modify.
d. Inheritance: Encapsulation is a key feature in inheritance as it allows subclasses to inherit the data and behavior of the superclass without exposing the implementation

details of the superclass. This makes it easier to create subclasses that extend the functionality of the superclass without affecting other parts of the program.

## 6. How to achieve encapsulation in Java? Give an example.

Ans → In Java, encapsulation can be achieved by declaring the data members of a class as private and providing public methods (getters and setters) to access and modify the data. This ensures that the data is accessed and modified only through the public methods, which can be used to enforce any necessary constraints or validation.
Here's an example to illustrate how encapsulation can be achieved in Java:

```
public class Person {
  private String name;
  private int age;

  // Constructor
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  // Getter for name
  public String getName() {
    return this.name;
  }

  // Setter for name
  public void setName(String name) {
    this.name = name;
  }

  // Getter for age
  public int getAge() {
    return this.age;
  }

  // Setter for age
  public void setAge(int age) {
    this.age = age;
  }
}
```

In the above example, we have defined a class called Person with private data members name and age. We have also defined public getters and setters for these data members.

The getters and setters allow external code to access and modify the data members of the class, but only through the public methods. This provides better control over the data and behavior of the class and promotes encapsulation.