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

Ans:-Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that involves bundling data (instance variables) and methods (functions) that operate on that data into a single unit called a class. It provides a way to hide the internal implementation details of an object and expose only the necessary functionalities through well-defined methods. Encapsulation promotes information hiding, data protection, and code organization, making it a crucial principle for building maintainable and robust software systems.

Ques 2. What are the important features of Encapsulation?

Ans:- Encapsulation is one of the four fundamental principles of object-oriented programming (OOP), along with inheritance, polymorphism, and abstraction. It's a concept that plays a critical role in designing well-structured, maintainable, and secure software. The key features of encapsulation in Java include:

**Data Hiding:** Encapsulation hides the internal details of an object's state (data) from external entities. The instance variables (fields) of a class are declared as private, preventing direct access and modification from outside the class. This promotes data security and integrity.

**Access Control:** Access modifiers (private, protected, and public) are used to control the visibility and access level of class members (fields and methods). Encapsulation allows you to expose only the necessary methods (interfaces) to interact with the object's data, keeping the rest of the implementation hidden.

**Security**: By hiding the implementation details of an object, encapsulation helps prevent unauthorized or unintended access to sensitive data. This is particularly important for maintaining the security of an application.

**Code Organization:** By encapsulating related data and methods within a class, encapsulation contributes to organized and modular code. Each class represents a cohesive unit of functionality, making it easier to manage and extend the system.

**Data Validation and Consistency:** Encapsulation allows you to enforce validation checks and ensure data consistency within the class's methods. For example, setter methods can include validation logic to prevent the assignment of invalid values to fields.

**Reduced** **Dependency**: Encapsulation reduces direct dependencies between different parts of a program. External code only interacts with the public methods, reducing the impact of changes made to the class's internal implementation.

Ques 3. What are the getter and setter methods in java Example with an Example?

Ans:- Getter and setter methods are frequently used in Java programming. Getter and setter methods in Java are widely used to access and manipulate the values of class fields. Usually, class fields are decorated with a private access specifier. Thus, to access them, public access specifiers are used with the getter and setter methods.

Example:-

import java.lang.String;

class Student{

private String name;

private int age;

public String getName(){

return name;

}

public static setName(String name){

this.name=name;

public int getAge(){

return age;

}

public void setAge(int age){

if(age>5 && age<30){

this.age=age;

}else{

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

}

}

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

Ans:- In Java, the this keyword is used to refer to the current instance of the class in which it is used. It is often used to distinguish between instance variables and parameters or local variables with the same name. The this keyword can also be used to invoke other constructors within the same class.

Example:

class Person {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public Person(String name) {

this(name, 0); // Sets age to 0 as a default value

}

public void displayInfo() {

System.out.println("Name: " + this.name);

System.out.println("Age: " + this.age);

}

}

public class Main {

public static void main(String[] args) {

Person person1 = new Person("Alice", 25);

person1.displayInfo();

Person person2 = new Person("Bob");

person2.displayInfo();

}

}

Ques 5. What is the advantage of Encapsulation?

Ans:- Here are some specific advantages of encapsulation in Java:

**Data Hiding**: Encapsulation allows you to hide the internal details and implementation of a class from outside access. By using access modifiers like private, you can restrict direct access to certain class members.

**Controlled Access**: Encapsulation allows you to define specific methods (getters and setters) to access and modify the internal state of an object.

**Flexibility and Maintenance**: By encapsulating data and providing well-defined interfaces for interaction, you can change the internal implementation of a class without affecting the code that uses it.

**Code Reusability**: Encapsulation promotes modularity, which makes it easier to reuse code. You can create well-encapsulated classes and use them in different parts of your application or even in different projects.

Improved Security: By restricting direct access to sensitive data, you can prevent unauthorized modifications or access.

Ques 6 How to achieve encapsulation in java? Give an example.

Ans:- Encapsulation in Java is achieved by using access modifiers to control the visibility of class members (fields and methods) and providing public methods (getters and setters) to interact with the private fields. This ensures that the internal state of an object can only be accessed and modified through controlled methods, maintaining data integrity and control over how the data is used.

Example:

class Student {

private String name;

private int age;

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age >= 0) {

this.age = age;

} else {

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

}

}

}

public class Main {

public static void main(String[] args) {

// Create a Student object

Student student = new Student("Alice", 20);

// Accessing fields through getters

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

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

// Modifying fields through setters

student.setName("Bob");

student.setAge(25);

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

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

}

}