



FULL STACK DEVELOPMENT WORKSHEET – 4

Q1. Write in brief about OOPS Concept in java with Examples. (In your own words)

Ans. The idea of objects is central to the programming paradigm known as object-oriented programming (OOP). OOP is a core component of the Java language. It enables you to organise data and behaviour into objects, which you may then use to develop modular, reusable, and scaleable code. Encapsulation, Inheritance, Polymorphism, and Abstraction are the four major tenets of OOP.

i)Encapsulation: The act of combining data (variables) and methods (functions) that manipulate the data into a single entity known as an object is called encapsulation. It restricts access to the object's internal state to its public methods, often known as getter and setter methods, and conceals the object's internal state from the outside world. Data integrity and security are therefore ensured.

Example: public class Student {

```
    private String name;

    private int rollNumber;

    // Getter and setter methods

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public int getRollNumber() {

        return rollNumber;

    }

    public void setRollNumber(int rollNumber) {

        this.rollNumber = rollNumber;

    }

}
```

```
1  ✓ public class Student {  
2      private String name;  
3      private int rollNumber;  
4  
5      // Getter and setter methods  
6  ✓  public String getName() {  
7      |   return name;  
8      |  
9      |  
10     |  
10  ✓  public void setName(String name) {  
11     |   this.name = name;  
12     |  
13     |  
14  ✓  public int getRollNumber() {  
15     |   return rollNumber;  
16     |  
17     |  
18  ✓  public void setRollNumber(int rollNumber) {  
19     |   this.rollNumber = rollNumber;  
20     |  
21     |  
21     }  
    }
```

ii) Inheritance: One class (subclass/derived class) can inherit traits and behaviours from another class (superclass/base class) through the technique of inheritance. It encourages code reuse and creates a connection between classes. The functionality of the superclass may be replaced or expanded by the subclass.

Example: // Superclass

```
class Animal {  
  
    void sound() {  
  
        System.out.println("Some sound");  
  
    }  
  
}
```

// Subclass inheriting from Animal

```
class Dog extends Animal {  
  
    void sound() {  
  
        System.out.println("Bark");  
  
    }  
  
}
```

```
1 // Superclass
2 class Animal {
3     void sound() {
4         System.out.println(x:"Some sound");
5     }
6 }
7
8 // Subclass inheriting from Animal
9 class Dog extends Animal {
10    void sound() {
11        System.out.println(x:"Bark");
12    }
13 }
```

iii) Polymorphism: The word "polymorphism" signifies "many forms." It enables objects of several classes to be considered as belonging to a single superclass. Because of polymorphism, you may call runtime methods from derived classes using a base class reference. Runtime polymorphism (method overriding) and compile-time polymorphism (method overloading) are the two kinds.

Example: // Method Overriding (Runtime Polymorphism)

```
class Animal {
    void sound() {
        System.out.println("Some sound");
    }
}
```

```
class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}
```

// Method Overloading (Compile-time Polymorphism)

```
class Calculator {  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    double add(double a, double b) {  
        return a + b;  
    }  
}
```

```
1 // Method Overriding (Runtime Polymorphism)  
2 class Animal {  
3     void sound() {  
4         System.out.println(x:"Some sound");  
5     }  
6 }  
7  
8 class Dog extends Animal {  
9     void sound() {  
10        System.out.println(x:"Bark");  
11    }  
12 }  
13  
14 // Method Overloading (Compile-time Polymorphism)  
15 class Calculator {  
16     int add(int a, int b) {  
17         return a + b;  
18     }  
19  
20     double add(double a, double b) {  
21         return a + b;  
22     }  
23 }
```

iv) Abstraction: Abstraction is the idea that just the essential elements of an object should be shown and complicated implementation details should be hidden. Java uses abstract classes and interfaces to implement abstraction. Methods without a body that are available in abstract classes must be implemented by derived classes. Any class that implements an interface may have abstract methods that they must declare.

Example: // Abstract Class

```
abstract class Shape {  
    abstract void draw();  
}
```

```
// Interface

interface Printable {

    void print();

}

// Concrete class implementing Shape and Printable

class Circle extends Shape implements Printable {

    void draw() {

        System.out.println("Drawing a circle");

    }

    public void print() {

        System.out.println("Printing a circle");

    }

}
```

```
1 // Abstract Class
2 abstract class Shape {
3     abstract void draw();
4 }
5
6 // Interface
7 interface Printable {
8     void print();
9 }
10
11 // Concrete class implementing Shape and Printable
12 class Circle extends Shape implements Printable {
13     void draw() {
14         System.out.println(x:"Drawing a circle");
15     }
16
17     public void print() {
18         System.out.println(x:"Printing a circle");
19     }
20 }
```

With the help of Java's OOP ideas, software systems can be designed and managed in a structured and effective manner, which improves the organisation, flexibility, readability, and maintainability of the code.

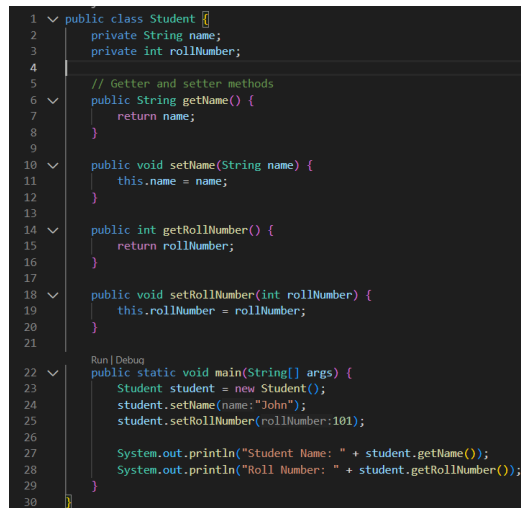
Q2. Write simple programs (wherever applicable) for every example given in Answer 2.

Ans. Here are simple Java programs for each of the examples mentioned in Answer 2:

Encapsulation Example:

```
public class Student {  
  
    private String name;  
  
    private int rollNumber;  
  
    // Getter and setter methods  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getRollNumber() {  
        return rollNumber;  
    }  
  
    public void setRollNumber(int rollNumber) {  
        this.rollNumber = rollNumber;  
    }  
}
```

```
public static void main(String[] args) {  
  
    Student student = new Student();  
  
    student.setName("John");  
  
    student.setRollNumber(101);  
  
  
    System.out.println("Student Name: " + student.getName());  
  
    System.out.println("Roll Number: " + student.getRollNumber());  
  
}  
}
```



```
1  public class Student {  
2      private String name;  
3      private int rollNumber;  
4  
5      // Getter and setter methods  
6      public String getName() {  
7          return name;  
8      }  
9  
10     public void setName(String name) {  
11         this.name = name;  
12     }  
13  
14     public int getRollNumber() {  
15         return rollNumber;  
16     }  
17  
18     public void setRollNumber(int rollNumber) {  
19         this.rollNumber = rollNumber;  
20     }  
21  
22     Run | Debug  
23     public static void main(String[] args) {  
24         Student student = new Student();  
25         student.setName(name:"John");  
26         student.setRollNumber(rollNumber:101);  
27  
28         System.out.println("Student Name: " + student.getName());  
29         System.out.println("Roll Number: " + student.getRollNumber());  
30     }  
}
```

Inheritance Example:

// Superclass

```
class Animal {  
  
    void sound() {  
  
        System.out.println("Some sound");  
  
    }  
  
}
```



```
// Subclass inheriting from Animal

class Dog extends Animal {

    void sound() {

        System.out.println("Bark");

    }

    public static void main(String[] args) {

        Dog dog = new Dog();

        dog.sound(); // Output: Bark

    }

}
```

```
1 // Superclass
2 class Animal {
3     void sound() {
4         System.out.println(x:"Some sound");
5     }
6 }
7
8 // Subclass inheriting from Animal
9 class Dog extends Animal {
10    void sound() {
11        System.out.println(x:"Bark");
12    }
13
14    Run | Debug
15    public static void main(String[] args) {
16        Dog dog = new Dog();
17        dog.sound(); // Output: Bark
18    }
19 }
```

Polymorphism Example (Method Overriding):

// Method Overriding (Runtime Polymorphism)

```
class Animal {

    void sound() {

        System.out.println("Some sound");

    }

}
```

```
}  
}
```

```
class Dog extends Animal {  
    void sound() {  
        System.out.println("Bark");  
    }  
}
```

```
public static void main(String[] args) {  
    Animal myDog = new Dog();  
    myDog.sound(); // Output: Bark  
}  
}
```

```
1 // Method Overriding (Runtime Polymorphism)  
2 class Animal {  
3     void sound() {  
4         System.out.println(x:"Some sound");  
5     }  
6 }  
7  
8 class Dog extends Animal {  
9     void sound() {  
10        System.out.println(x:"Bark");  
11    }  
12  
13 Run | Debug  
14 public static void main(String[] args) {  
15     Animal myDog = new Dog();  
16     myDog.sound(); // Output: Bark  
17 }
```

Polymorphism Example (Method Overloading):

// Method Overloading (Compile-time Polymorphism)

```
class Calculator {  
    int add(int a, int b) {
```

```
    return a + b;
}
```

```
double add(double a, double b) {
    return a + b;
}
```

```
public static void main(String[] args) {
    Calculator calculator = new Calculator();
    System.out.println(calculator.add(5, 10)); // Output: 15
    System.out.println(calculator.add(2.5, 3.5)); // Output: 6.0
}
}
```

```
1 // Method Overloading (Compile-time Polymorphism)
2 class Calculator {
3     int add(int a, int b) {
4         return a + b;
5     }
6
7     double add(double a, double b) {
8         return a + b;
9     }
10
11     Run | Debug
12     public static void main(String[] args) {
13         Calculator calculator = new Calculator();
14         System.out.println(calculator.add(5, 10)); // Output: 15
15         System.out.println(calculator.add(2.5, 3.5)); // Output: 6.0
16     }
17 }
```

Abstraction Example:

// Abstract Class

```
abstract class Shape {
    abstract void draw();
}
```

```
// Interface

interface Printable {

    void print();

}

// Concrete class implementing Shape and Printable
class Circle extends Shape implements Printable {

    void draw() {

        System.out.println("Drawing a circle");

    }

    public void print() {

        System.out.println("Printing a circle");

    }

    public static void main(String[] args) {

        Circle circle = new Circle();

        circle.draw(); // Output: Drawing a circle

        circle.print(); // Output: Printing a circle

    }

}
```

```
1 // Abstract Class
2 abstract class Shape {
3     abstract void draw();
4 }
5
6 // Interface
7 interface Printable {
8     void print();
9 }
10
11 // Concrete class implementing Shape and Printable
12 class Circle extends Shape implements Printable {
13     void draw() {
14         System.out.println(x:"Drawing a circle");
15     }
16
17     public void print() {
18         System.out.println(x:"Printing a circle");
19     }
20
21     Run | Debug
22     public static void main(String[] args) {
23         Circle circle = new Circle();
24         circle.draw(); // Output: Drawing a circle
25         circle.print(); // Output: Printing a circle
26     }
27 }
```

These Java programmes include examples of encapsulation, inheritance, polymorphism (including method overloading and method overriding), and abstraction.

Multiple Choice Questions:

Q1. Which of the following is used to make an Abstract class?

- A. Making at least one member function as pure virtual function
- B. Making at least one member function as virtual function
- C. Declaring as Abstract class using virtual keyword
- D. Declaring as Abstract class using static keyword

Ans. A. Making at least one member function as pure virtual function

Q2. Which of the following is true about interfaces in java.

- 1) An interface can contain the following type of members.public, static, final fields (i.e., constants)default and static methods with bodies**
- 2) An instance of the interface can be created.**
- 3) A class can implement multiple interfaces.**
- 4) Many classes can implement the same interface.**

- A. 1, 3 and 4
- B. 1, 2 and 4
- C. 2, 3 and 4
- D. 1,2,3 and 4

Ans. B. 1, 2 and 4

Q3. When does method overloading is determined?

- A. At run time
- B. At compile time
- C. At coding time
- D. At execution time

Ans. B. At compile time

According to the number and type of arguments supplied to the method at compile time, Java determines if a method is overloadable.

Q4. What is the number of parameters that a default constructor requires?

- A. 0
- B. 1
- C. 2
- D. 3

Ans. A. 0

In Java, a constructor without arguments is referred to as a default constructor. If no constructor is defined in the class, it is automatically built by the compiler.

Q5. To access data members of a class, which of the following is used?

- A. Dot Operator
- B. Arrow Operator
- C. A and B both as required
- D. Direct call

Ans. A. Dot Operator

To access the data members and methods of a class in Java, use the dot operator (.). In Java, the arrow operator (->) is not used.

Q6. Objects are the variables of the type ____?

- A. String
- B. Boolean
- C. Class
- D. All data types can be included

Ans. C. Class

Object-oriented programming defines objects as variables of a class type. They are able to store and work with data of that class's kind.

Q7. A non-member function cannot access which data of the class?

- A. Private data
- B. Public data
- C. Protected data
- D. All of the above

Ans. A. Private data

Non-member functions are unable to directly access a class's secret data members. Only the class itself has access to private members. On the other hand, depending on their access level, public and protected members can occasionally be accessible from outside the class.

Q8. Predict the output of following Java program

```
class Test {  
  
    int i;  
  
}  
  
class Main {  
  
    public static void main(String args[]) {  
  
        Test t = new Test();  
  
        System.out.println(t.i);  
  
    }  
  
}
```

- A. garbage value
- B. 0
- C. compiler error
- D. runtime Error

Ans. B. 0

Instance variables in the Test class, such as int i, are automatically initialised in Java to their default values. The default value for integers is 0. The output of the provided programme will thus be 0.

Q9.Which of the following is/are true about packages in Java?

- 1) Every class is part of some package.**
- 2) All classes in a file are part of the same package.**
- 3) If no package is specified, the classes in the file go into a special unnamed package**
- 4) If no package is specified, a new package is created with folder name of class and the class is put in this package.**

A. Only 1, 2 and 3

B. Only 1, 2 and 4

C. Only 4

D. Only 1, 3 and 4

Ans. A. Only 1, 2 and 3

Explanation:

1.Every class belongs to a package. (True)

2.All classes in a file belong to the same package. (True)

3.If no package is specified, the classes in the file are placed in a special unnamed package. (True)

Option 4 is incorrect because Java does not create a new package based on the folder name of the class. The package structure is explicitly defined by the programmer.

Q10. Predict the Output of following Java Program.

```
class Base {  
    public void show() {  
        System.out.println("Base::show() called");  
    }  
}  
  
class Derived extends Base {  
    public void show() {  
        System.out.println("Derived::show() called");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Base b = new Derived();  
        b.show();  
    }  
}
```

Ans. The output of the Java programme is as follows:

```
sql Copy code  
Derived::show() called
```

Explanation:

- i) Derived and Base are the two specified classes. Base has a subclass called Derived.
- ii) The main method of the Main class produces a Derived class object and assigns it to a Base reference variable.

java


 Copy code

```
Base b = new Derived();
```

Polymorphism is applied here. Although the reference variable `b` is of type `Base`, it refers to a `Derived` class object. This implies that while you can use methods from the `Base` class on this object, if those methods have been overridden, the `Derived` class's overridden methods will be used instead.

- iii) On the reference variable `b`, the `show` method is called:

java

 Copy code

```
b.show();
```

The override function in the `Derived` class is called at runtime when the JVM detects the actual object type, which in this instance is `Derived`. Therefore, the result will be:

sql

 Copy code

```
Derived::show() called
```

Q11. What is the output of the below Java program?

```
class Base {  
  
    final public void show() {  
  
        System.out.println("Base::show() called");  
  
    }  
}  
  
class Derived extends Base {  
  
    public void show() {  
  
        System.out.println("Derived::show() called");  
  
    }  
}  
  
class Main {  
  
    public static void main(String[] args) {  
  
        Base b = new Derived();  
  
        b.show();  
  
    }  
}
```

Ans. There are two classes in this Java programme: Base and Derived. Base has a subclass called Derived. Since the Base class's display() function is designated as final, no subclass can override it.

An object of the Derived class is generated in the main method and assigned to a reference of the Base class. Because the reference type is Base, when the display() function is invoked using the

reference b, the Base class's show() method is executed instead. As a result, the following will be the program's output:

```
Base::show() called
```

Q12.Find output of the program.

```
class Base {  
    public static void show() {  
        System.out.println("Base::show() called");  
    }  
}  
  
class Derived extends Base {  
    public static void show() {  
        System.out.println("Derived::show() called");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Base b = new Derived();  
        b.show();  
    }  
}
```

Ans. There are two classes in this Java programme: Base and Derived. Show() is a static function shared by both types. Instead of being resolved at runtime depending on the actual type of the object, static methods are resolved at compile time based on the reference type.

An object of the Derived class is generated in the main method and assigned to a reference of the Base class. When a static method is called, the reference type is used to resolve it at build time. Therefore, regardless of whether the actual object belongs to the Derived class, Base::show() will be used depending on the reference type.

The program's output will be:

```
Base::show() called
```

Q13.What is the output of the following program?

```
class Derived {  
    public void getDetails() {  
        System.out.printf("Derived class ");  
    }  
}  
  
public class Test extends Derived  
{  
    public void getDetails()  
{  
        System.out.printf("Test class ");  
        super.getDetails();  
    }  
  
    public static void main(String[] args)  
{  
        Derived obj = new Test();  
        obj.getDetails();  
    }  
}
```

Ans. Derived and Test are the two classes used in this Java programme. Test overrides the getDetails() function since it is a subclass of Derived. An object of the Test class is generated in the main method and given to a reference of the Derived class.

The overridden function from the Test class will be used when the `getDetails()` method is used with the `obj` reference. The overridden function calls `super.getDetails()` after printing "Test class," which calls the `getDetails()` method from the Derived class.

As a result, the following will be the program's output:

```
Test class Derived class
```

Q14. What is the output of the following program?

```
class Derived
{
    public void getDetails(String temp)
    {
        System.out.println("Derived class " + temp);
    }
}

public class Test extends Derived
{
    public int getDetails(String temp)
    {
        System.out.println("Test class " + temp);
        return 0;
    }

    public static void main(String[] args)
    {
        Test obj = new Test();
        obj.getDetails("Name");
    }
}
```

Ans. You have a base class called Derived and a derived class called Test that extends the Derived class in the sample Java programme. The Test class overrides the getDetails(String temp) function that is present in the Derived class.

A Java subclass can offer a customised implementation of a method that is already defined in its superclass by using a technique called method overriding. A method in a subclass that shares the same signature as a method in the superclass is overridden by the method in the subclass.

In this instance, the Test class's getDetails(String temp) function replaces the Derived class's equivalent method.

The overridden method in the Test class will be called when you create an object of the Test class and use the getDetails("Name") function.

As a result, the program's output will be:

```
Test class Name
```

When the Test class's getDetails function is used, "Test class Name" is printed.

Q15.What will be the output of the following Java program?

```
class test
{
    public static int y = 0;
}

class HasStatic
{
    private static int x = 100;

    public static void main(String[] args)
    {
        HasStatic hs1 = new HasStatic();
        hs1.x++;
        HasStatic hs2 = new HasStatic();
        hs2.x++;
        hs1 = new HasStatic();
        hs1.x++;
        HasStatic.x++;
        System.out.println("Adding to 100, x = " + x);
        test t1 = new test();
        t1.y++;
        test t2 = new test();
    }
}
```

```
t2.y++;  
  
t1 = new test();  
  
t1.y++;  
  
System.out.print("Adding to 0, ");  
  
System.out.println("y = " + t1.y + " " + t2.y + " " + test.y);  
  
}  
  
}
```

Ans. Let's step-by-step dissect the code:

i) `HasStatic hs1 = new HasStatic();`: The x variable is increased by 101 and a new object of the `HasStatic` class is generated.

ii) `HasStatic hs2 = new HasStatic();`: The x variable is increased to 102 and a new `HasStatic` object of the `HasStatic` class is generated.

iii) `hs1` now corresponds to a new `HasStatic` class object, and the x variable is increased by 103 with the statement `hs1 = new HasStatic()`.

iv) `HasStatic.x++`: The `HasStatic` class's x variable is raised to the value 104.

The result will now be "Adding to 100, x equals 104."

i) `test t1 = new test();`: The y variable is increased by one and a new object of the `test` class is generated.


ii) `test t2 = new test();` creates a second new object of the `test` class and increases the y variable by one.

iii) `t1` now corresponds to a new object of the `test` class, and the y variable is increased by one when you type `t1 = new test()`.

Following the execution of the print command "Adding to 0," the values of `t1.y`, `t2.y`, and `test.y` are shown. Since each of these variables is static, every instance of the class may access them. Thus, all three variables are raised by one.

The program's output will be:

CSS

 Copy code

Adding to 100, x = 104

Adding to 0, y = 1 1 1

Q16.Predict the output**class San****{****public void m1 (int i,float f)****{****System.out.println(" int float method");****}****public void m1(float f,int i);****{****System.out.println("float int method");****}****public static void main(String[] args)****{****San s=new San();****s.m1(20,20);****}****}**

Ans. The Java code that is supplied defines the class San as having two m1 methods, one of which takes parameters of type int i, float f, and another of type float f, int i. The m1 method is called with the inputs (20, 20) in the main function, which also creates an instance of the San class.

The method that is called depends on the order of the parameters in the method call since both methods might take the supplied arguments. In this instance, the method call's parameter order corresponds to the second m1 method's signature: m1(float f, int i).

As a result, the following will be the program's output:

```
float int method
```

Q17.What is the output of the following program?

```
public class Test
{
    public static void main(String[] args)
    {
        int temp = null;

        Integer data = null;

        System.out.println(temp + " " + data);
    }
}
```

Ans. Due to a compilation problem, the provided code will not compile. Java does not allow null values to be assigned to primitive types (like int). Similar to that, you cannot assign null to a variable of a primitive type. A compilation error will occur if the code snippet `int temp = null;` is used.

Utilise the Integer class, which is an object wrapper around the int primitive type, if you wish to utilise a nullable integer. The updated code is provided below:

```
public class Test {
    public static void main(String[] args) {
        Integer temp = null;
        Integer data = null;
        System.out.println(temp + " " + data);
    }
}
```



```
J Test.java
1 public class Test {
2     Run | Debug
3     public static void main(String[] args) {
4         Integer temp = null;
5         Integer data = null;
6         System.out.println(temp + " " + data);
7     }
}
```

Both temp and data are of type Integer in the updated version and are capable of receiving a value of null. The result of running this code is:

```
null null
```

Q18.Find output

```
class Test {

    protected int x, y;

}

class Main {

    public static void main(String args[]) {

        Test t = new Test();

        System.out.println(t.x + " " + t.y);

    }

}
```

Ans. The class Test in this code contains two protected integer variables named x and y. Another class Main makes a Test object and tries to output the x and y values.

In Java, instance variables are given default values when you create an object of a class. The default value for int is 0. X and Y will have values of 0 because they are not expressly initialised.

As a result, the following is what the code will produce:

A screenshot of a terminal window with a dark background. The output '0 0' is displayed in white text. In the top right corner of the terminal, there is a small icon of a document and the text 'Copy code'.

Q19.Find output

// filename: Test2.java

```
class Test1 {  
    Test1(int x)  
    {  
        System.out.println("Constructor called " + x);  
    }  
}  
  
class Test2 {  
    Test1 t1 = new Test1(10);  
    Test2(int i) { t1 = new Test1(i); }  
    public static void main(String[] args)  
    {  
        Test2 t2 = new Test2(5);  
    }  
}
```

Ans. There are two classes, Test1 and Test2, in the provided code.

An instance variable of type Test1 named t1 is defined in the Test2 class, and it is initialised with a Test1 object in the declaration. In addition, the Test2 class has a constructor that accepts an integer as its only variable, builds a new Test1 object with it, and then assigns it to the t1 variable.

Test2 t2 = new Test2(5); calls for the constructor of the Test2 object when you create one in the main function. A new Test1 object is given to t1 inside the constructor with the value 5.

The code's output will be:

```
Constructor called 10
Constructor called 5
```

Explanation:

i)When the t1 variable is initialised in the Test2 class definition, Test1(10) is executed.

ii)When the Test2 object is created in the main method, Test1(5) is called inside the constructor of that object.

Q20.What will be the output of the following Java program?

class Main

{

public static void main(String[] args)

{

int [][]x = {{1,2}, {3,4,5}, {6,7,8,9}};

int [][]y = x;

System.out.println(y[2][1]);

}

}

Ans. The Java programme offered constructs a 2D array `x` with various row lengths before assigning it to another 2D array `y`. A 2D array in Java can contain rows of various lengths, however in this situation, the array `x` has rows of various lengths, and this behaviour may provide unexpected outcomes.

The programme makes an effort to access `y[2][1]` in this unique instance. The array `x` appears as follows:

```
{{1, 2}, {3, 4, 5}, {6, 7, 8, 9}}
```

Now that `y` is referring to the same array as `x` in the assignment `int[][] y = x;`. The third row (index 2) and the second element (index 1) in that row are therefore represented as `y[2][1]`. The second member in the third row of the array `x`, which is composed of the numbers 6, 7, 8, and 9, is 7.

As a result, the program's output will be:

```
7
```

Q21.What will be the output of the following Java program?

```
class A
{
    int i;

    public void display()
    {
        System.out.println(i);
    }
}

class B extends A
{
    int j;

    public void display()
    {
        System.out.println(j);
    }
}

class Dynamic_dispatch
{
    public static void main(String args[])
    {
```

```
B obj2 = new B();  
  
obj2.i = 1;  
  
obj2.j = 2;  
  
A r;  
  
r = obj2;  
  
r.display();  
  
}  
  
}
```

Ans. There are two classes, A and B, in the sample Java programme, with B extending A. Both classes contain a show() function and an int variable.

An object of type B is created in the Dynamic_dispatch class, and its i and j variables are given the values 1 and 2, respectively. The reference to the B object is then copied to a reference r of type A.

Java utilises dynamic method dispatch to decide which version of the show() method to run when r.display() is called based on the actual type of the object, in this instance an object of type B.

As a result, the following will be the program's output:

2

This occurs as a result of the show() function of class B being used to print the value of the variable j, which is 2, which is 2.

Q22. What will be the output of the following Java code?

```
class A
{
    int i;

    void display()
    {
        System.out.println(i);
    }
}

class B extends A
{
    int j;

    void display()
    {
        System.out.println(j);
    }
}

class method_overriding
{
    public static void main(String args[])
    {
```

```
B obj = new B();  
  
obj.i=1;  
  
obj.j=2;  
  
obj.display();  
  
}  
  
}
```

Ans. The classes A and B in this Java code are subclasses of each other, and class B's display() method overrides class A's display() method. The main method creates an object of class B, obj, with the i and j variables set to 1 and 2, respectively. As a result, when the display() method is called using the obj reference, it will call the overridden display() method from class B, and the program's output will be:

2

Q23.What will be the output of the following Java code?

```
class A
{
    public int i;
    protected int j;
}
class B extends A
{
    int j;
    void display()
    {
        super.j = 3;
        System.out.println(i + " " + j);
    }
}
class Output
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.i=1;
```

```
    obj.j=2;  
    obj.display();  
}  
}
```

Ans. Two classes, A and B, are present in this Java code. A subclass of class A is class B. The function show() in class B outputs the values of i and j after setting the j variable in superclass A to 3. An object of type B named obj is created in the main function. The j variable in class B is set to 2, hiding the j variable from class A, while the i variable is set to 1. The superclass A's j variable is set to 3 and the values of i and j are printed when the show() function is invoked.

As a result, the following will be the program's output:



```
1 3
```

Q24.What will be the output of the following Java program?

```
class A
{
    public int i;
    public int j;
    A()
    {
        i = 1;
        j = 2;
    }
}
class B extends A
{
    int a;
    B()
    {
        super();
    }
}
class super_use
{
```

```
public static void main(String args[])  
{  
    B obj = new B();  
    System.out.println(obj.i + " " + obj.j);  
}  
}
```

Ans. Three classes—A, B, and super_use—are defined in the Java programme provided.

The constructor of Class A initialises the two public integer variables i and j to 1 and 2, respectively.

A is a subclass of Class B. Although it lacks any extra variables, super() is used to make an implicit call to A's default constructor.

An object of class B is generated in the super_use class (obj). The variables from class A will be used when you print objects i and j since class B is a subclass of A and inherits its variables.

As a result, the following will be the program's output:



```
1 2
```

Q 25. Find the output of the following program.

```
class Test
{
    int a = 1;
    int b = 2;

    Test func(Test obj)
    {
        Test obj3 = new Test();
        obj3 = obj;
        obj3.a = obj.a++ + ++obj.b;
        obj.b = obj.b;
        return obj3;
    }

    public static void main(String[] args)
    {
        Test obj1 = new Test();
        Test obj2 = obj1.func(obj1);
        System.out.println("obj1.a = " + obj1.a + " obj1.b = " + obj1.b);
        System.out.println("obj2.a = " + obj2.a + " obj1.b = " + obj2.b);
    }
}
```

Ans. Let's dissect the code step by step:

i) When `Test obj1 = new Test()`, a new `Test` object is created with initial values of 1 and 2, respectively.

ii) `Test obj2 = obj1.func(obj1)` calls `obj1`'s `func` method. A new `Test` object, `obj3`, is created in the `func` method, and `obj3` is then given the reference to `obj`. Then, based on certain procedures involving `obj.a` and `obj.b`, the values of `a` and `b` in `obj3` are changed.

Let's examine the actions performed by the `func` method:

- `obj3.a = ++obj.b + ++obj.a;`
- After evaluating to the value of `obj.a`, which is now 1, `obj.a++` increases `obj.a` by 1.
- ++When `obj.b` is evaluated, the modified value of `obj.b` (which is 3 after incrementing by 1) is used instead of the original value.
- `Obj3.a` is given the value $1 + 3 = 4$ as a result.
- `obj.b = obj.b` maintains the value of `obj.b`, which is 3 in this case.

As a result, after invoking `func`, `obj1` and `obj2` will be in the following states:

- Due to the post-increment process in `obj.a++`, object 1.a becomes object 2.
- Due to the pre-increment action in `++obj.b`, `obj1.b` changes to 3.
- (As set inside the `func` function) `Obj2.a` becomes 4.
- `Obj2.b` changes to 3 (unaltered).

Let's print these values right now:

```
Copy code
obj1.a = 2 obj1.b = 3
obj2.a = 4 obj1.b = 3
```

As a result, the following will be the program's output:

```
Copy code
obj1.a = 2 obj1.b = 3
obj2.a = 4 obj1.b = 3
```