FULL STACK DEVELOPMENT – WORKSHEET 4

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

Ans: Object-oriented programming is a core of Java Programming, which is used for designing a program using classes and objects. OOPs, can also be characterized as data controlling for accessing the code. In this approach, programmers define the data type of a data structure and the operations that are applied to the data structure.

In the world of Java programming, it's essential to grasp the fundamental object-oriented concepts in Java. Object-Oriented Programming, or OOPs in Java, is not just a set of rules; it is a powerful paradigm that drives Java's design and development principles.

Objects: In Java, an object is more than just a runtime entity; it's the embodiment of a class. These objects mirror real-world entities, complete with both state and behaviour. Think of a 'Car' object based on a 'Car' class. It possesses attributes like colour, brand, and speed, along with behaviours like 'accelerate' and 'brake'. This is where Java OOP concepts come to life.

Classes: A class in Java serves as a blueprint for creating objects. It bundles data (attributes) and behaviour (methods) that define the object. Consider a 'Car' class; it defines attributes like colour and methods like 'accelerate()'. Essentially, a class is a template that characterizes what can be instantiated as an object, exemplifying OOPs concepts.

Abstraction: Abstraction in Java involves concealing complexity while exposing only the essential aspects. It's realized through abstract classes and interfaces. For instance, a 'Vehicle' interface declares a 'move()' method, but the actual implementation is left to classes like 'Car' or 'Bike'. This focus on "what an object does" rather than "how it does it" is a core OOPs concept in Java.

Inheritance: Inheritance is a mechanism where a new class (subclass) derives attributes and methods from an existing class (superclass). This fosters code reusability and establishes a hierarchical relationship between classes. For example, an 'ElectricCar' class can inherit traits from the 'Car' class, showcasing the practicality of Java OOPs concepts.

Polymorphism: Polymorphism allows Java methods and objects to assume multiple forms. It finds common use in method overloading (same method name, different parameters) and method overriding (same method name and parameters in subclass as in parent class). Consider a 'draw()' method implemented differently in 'Circle', 'Square', and 'Triangle' classes as a prime example of polymorphism in Java.

Encapsulation: Encapsulation is a pivotal OOPs principle in Java. It entails bundling data (attributes) and code (methods) into a cohesive unit. Moreover, it limits direct access to an object's components, preventing inadvertent interference. Techniques like using private variables and offering public getter and setter methods exemplify Java OOPs concepts put into practice.

OOPs Concepts in Java with Examples:

- **Class Example:** class Car { String color; void accelerate() { ... } }
- **Object Example:** Car myCar = new Car();
- **Inheritance Example:** class ElectricCar extends Car { ... }
- **Polymorphism Example:** Method overloading and overriding.
- **Abstraction Example:** Abstract classes and interfaces.
- **Encapsulation Example:** Private fields with public getters and setters.

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

Ans: Encapsulation:

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

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
   // ... other methods
}
```

Inheritance:

```
class Student extends Person {
    private String school;
    // ... other methods
}
```

Polymorphism:

```
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}
```

class Circle extends Shape {

```
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}
```

Abstraction:

```
abstract class Animal {
    public abstract void makeSound();
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}
```

Multiple Choice Questions:

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

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.

Ans: (D) 1,2,3 and 4

Q3. When does method overloading is determined?

Ans: (B) At compile time

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

Ans: (A) 0

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

Ans: (A) Dot operator

Q6. Objects are the variables of the type ____?

Ans. (C) Class

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

Ans: (A) Private Data

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);
}
}
```

Ans: (C) Compile error

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.

Ans: (A) Only 1,2 and 3

For Q10 to Q25 find output with explanation.

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: Explanation:

- In Java, when a method is called on an object, the actual method that gets executed is determined at runtime based on the actual object type, not the reference type.
- Here, b is a reference of type Base, but it points to an object of type Derived.
- When b.show() is called, Java uses dynamic method dispatch to determine that the show() method of the Derived class should be executed, because the actual object is of type Derived.

Output:

```
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: Explanation

1. Class Base: It has a method show() which is marked as final. This means that the method show() in the Base class cannot be overridden by any subclass.

2. Class Derived: This class attempts to override the show() method of the Base class. However, since the show() method in the Base class is final, this is not allowed in Java.

3. Class Main: This class contains the main method which creates an instance of Derived but assigns it to a reference of type Base. It then calls the show() method on this reference.

Compilation Error: Since the show() method in Base is declared as final, any attempt to override this method in Derived results in a compilation error. The error message would typically be something like:

```
Error: Derived.java:4: show() in Derived cannot override show() in Base
overridden method is final
```

```
public void show() {
          ^
```

Output of the Corrected Code:

```
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: Explanation

1. Class Base: This class contains a static method named show() which prints "Base::show() called".
2. Class Derived: This class contains a static method named show() which prints "Derived::show() called". This method hides the show() method in the Base class but does not override it because static methods are not polymorphic in Java.

3. Class Main: This class contains the main method, which creates an instance of Derived but assigns it to a reference of type Base. It then calls the show() method on this reference.

Output:

```
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: Explanation

1. Class Derived: This class contains a method getDetails() which prints "Derived class ".
2. Class Test: This class extends Derived and overrides the getDetails() method.The overridden method prints "Test class " and then calls super.getDetails() to invoke the getDetails() method of the superclass Derived.

3. Class Test (continued): The main method creates an instance of Test but assigns it to a reference of type Derived then calls the getDetails() method on this reference.

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: Explanation

1. Class Derived: This class contains a method getDetails(String temp) which prints "Derived class " + temp.

2. Class Test: This class attempts to define a method getDetails(String temp) which has the same parameter list as the method in Derived, but it has a different return type (int instead of void).

Compilation Issue: In Java, method overriding requires the method in the subclass to have the same return type (or a covariant return type) as the method in the superclass. The method in Test has a different return type (int) from the method in Derived (void), which is not allowed for overriding. Therefore, this results in a compilation error.

The specific error message would typically be something like:

```
Error: method does not override or implement a method from a supertype
public int getDetails(String temp) {
                  ^
```

Output of the Corrected Code:

```
Test class Name
```

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: Explanation:

HasStatic Class:

- HasStatic hs1 = new HasStatic();
  - hs1.x refers to the static variable x, which is 100.
  - hs1.x++ increments x to 101.
- HasStatic hs2 = new HasStatic();
  - hs2.x refers to the static variable x, which is now 101.
  - hs2.x++ increments x to 102.
- hs1 = new HasStatic();
  - This creates a new instance hs1, but it does not affect the static variable x.
  - hs1.x++ increments x to 103.
- HasStatic.x++
  - This directly increments the static variable x to 104.
- System.out.println("Adding to 100, x = " + x);

   - This prints the value of x, which is now 104.

test Class:

- test t1 = new test();
   - t1.y refers to the static variable y, which is 0.
   - t1.y++ increments y to 1.
- test t2 = new test();
   - t2.y refers to the static variable y, which is now 1.
   - t2.y++ increments y to 2.
- t1 = new test();
   - This creates a new instance t1, but it does not affect the static variable y.
   - t1.y++ increments y to 3.
- System.out.print("Adding to 0, ");
   - This prints the string "Adding to 0, ".
- System.out.println("y = " + t1.y + " " + t2.y + " " + test.y);
   - t1.y, t2.y, and test.y all refer to the same static variable y, which is now 3.
   - This prints "y = 3 3 3

```
Output: Adding to 100, x = 104
        Adding to 0, y = 3 3 3
```

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: Explanation:

Method Declarations:

- public void m1(int i, float f) prints "int float method".
- public void m1(float f, int i) prints "float int method".

Main Method:

- San s = new San(); creates a new instance of the San class.
- s.m1(20, 20); attempts to call a method with two integer arguments.

Compilation Error: The call s.m1(20, 20); will cause a compilation error because neither of the m1 methods accepts two integers. Java does not have a method m1(int, int) in the San class, and therefore, it cannot determine which method to call based on the provided arguments.

Corrected Execution

- s.m1(20, 20.0f); matches the method m1(int i, float f) and will print "int float method".

Output:

```
int float 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: Explanation:

Compilation Errors

1. int temp = null;:
   - This line causes a compilation error because you cannot assign null to a primitive int.
2. Integer data = null;:
   - This line is valid because Integer is a reference type and can hold null.

Output:

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: Explanation:

Class Test:

- This class has two protected integer variables x and y.
- These variables are not initialized, so they will have their default values.

Class Main:

- The main method is the entry point of the program.
- An instance of the Test class is created.
- The values of x and y are printed.

Output:

```
0 0
```

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: Explanation:

Code Analysis

Class Test1:

- This class has a constructor that takes an integer x and prints a message with the value of x.

Class Test2:

- This class has an instance variable t1 of type Test1, which is initialized with new Test1(10) at the point of declaration.
- The constructor of Test2 takes an integer i and reassigns t1 with new Test1(i).

Main Method:

- An instance of Test2 is created with the value 5 passed to its constructor.

Step-by-Step Execution

Static Initialization of Test2:

- When Test2 is loaded, its instance variable t1 is initialized to new Test1(10).
- This will call the constructor of Test1 with x = 10 and print "Constructor called 10".

Constructor of Test2:

- When the constructor Test2(int i) is called with i = 5, it reassigns t1 to new Test1(i).
- This will call the constructor of Test1 with x = 5 and print "Constructor called 5".

Output:

```
Constructor called 10

Constructor called 5
```

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: Explanation:

Array Initialization:

- int []x[] = {{1,2}, {3,4,5}, {6,7,8,9}};
  - This declares and initializes a 2D array x.

Array Assignment:

- int [][]y = x;
  - This assigns the reference of array x to y, so both y and x refer to the same 2D array.

Array Access:

- System.out.println(y[2][1]);
  - This accesses the element at row 2, column 1 of the array y.
  - y[2] refers to the array {6, 7, 8, 9}.
  - y[2][1] refers to the element at index 1 of this array, which is 7.

Output:

```
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: Explanation:** Class A:

- It has an integer field i.
- It has a method display() that prints the value of i.

Class B:

- It extends class A.
- It has an additional integer field j.
- It overrides the display() method to print the value of j.

Class Dynamic_dispatch:

- In the main method:
  - An instance of B named obj2 is created.
  - obj2.i is set to 1.
  - obj2.j is set to 2.
  - A reference variable r of type A is assigned to reference obj2.
  - The display() method is called on r.

Because r is of type A but refers to an object of type B, the display() method of class B is called due to dynamic method dispatch (runtime polymorphism). This means the overridden display() method in B will execute, which prints the value of j.

Output:

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: Explanation :

Class A:

- It has an integer field i.
- It has a method display() that prints the value of i.

Class B:

- It extends class A.
- It has an additional integer field j.

    - It overrides the display() method to print the value of j.

Class method_overriding:

    - In the main method:
        - An instance of B named obj is created.
        - obj.i is set to 1.
        - obj.j is set to 2.
        - The display() method is called on obj.

Since obj is an instance of class B and the display() method is overridden in B, the display() method of class B will be executed. This method prints the value of j.

Output:

```
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: Explanation:

Class A:

- It has a public integer field i.
- It has a protected integer field j.

Class B:

- It extends class A.
- It has an integer field j that hides the j field from class A.
- It has a method display() which:
    - Sets super.j (the j from class A) to 3.
    - Prints the values of i and the local j field of class B.

Class Output:

- In the main method:
    - An instance of B named obj is created.
    - obj.i is set to 1.
    - obj.j is set to 2 (this sets the j field in class B, not the one in class A).
    - The display() method is called on obj.

When display() is called:

- super.j = 3; sets the j field in class A to 3.

- System.out.println(i + " " + j); prints the value of i from class A and the value of j from class B.

Output:

```
1 2
```

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: Explanation:

Class A:

- It has public integer fields i and j.
- The constructor initializes i to 1 and j to 2.

Class B:

- It extends class A.
- It has an integer field a.
- Its constructor calls super(), which is implicit even if not written because it calls the superclass constructor. In this case, it explicitly calls the superclass constructor, ensuring the fields i and j in class A are initialized.

Class super_use:

- In the main method:
  - An instance of B named obj is created.
  - The constructor of class B calls the constructor of class A, which initializes i to 1 and j to 2.
  - The System.out.println(obj.i + " " + obj.j); statement prints the values of i and j from the object obj.

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: Explanation:

Initialization:

- Test obj1 = new Test();
  - This creates a new Test object obj1 with a = 1 and b = 2.

Function Call:

- Test obj2 = obj1.func(obj1);
  - The method func is called with obj1 as the argument.
  - Inside func:
    - Test obj3 = new Test(); creates a new Test object obj3.
    - obj3 = obj; assigns obj1 to obj3, so now obj3 and obj (which is obj1) refer to the same object.

        - obj3.a = obj.a++ + ++obj.b; is evaluated as follows:
            - obj.a++ uses the current value of a (which is 1) and then increments a by 1. So, obj.a++ is 1, and obj.a becomes 2.
            - ++obj.b increments b by 1 before using it. So, ++obj.b is 3, and obj.b becomes 3.
            - Now, obj3.a = 1 + 3 = 4.
        - obj.b = obj.b; leaves obj.b unchanged, so obj.b remains 3.

Return:

- The method returns obj3, which is obj1. So, obj2 and obj1 refer to the same object.

Printing:

- System.out.println("obj1.a = " + obj1.a + " obj1.b = " + obj1.b);
    - This prints the values of a and b of obj1.
    - Since obj1 was modified by func, obj1.a is 4 and obj1.b is 3.
- System.out.println("obj2.a = " + obj2.a + " obj1.b = " + obj2.b);
    - Since obj2 refers to the same object as obj1, obj2.a is also 4 and obj2.b is 3.

Output:

```
obj1.a = 4 obj1.b = 3
obj2.a = 4 obj1.b = 3
```