FULL STACK DEVELOPMENT – WORKSHEET 4

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

Ans:-

Object-Oriented Programming (OOP) is a programming paradigm that organizes data and functions into reusable structures called objects. Java is an object-oriented programming language, and it encapsulates real-world entities and their interactions using several fundamental concepts of OOP. Here are some important OOP concepts in Java with examples:

1. Classes and Objects: A class is a blueprint or template for creating objects. An object is an instance of a class. For example, consider a class named `Car` that defines properties like `color`, `brand`, and `model`. An object of the `Car` class can be created as follows:

java

```
Car car1 = new Car();
car1.color = "Red";
car1.brand = "Toyota";
car1.model = "Camry";
```

2. Inheritance: Inheritance allows classes to inherit properties and methods from another class. It promotes code reusability and improves the organization of code. For example, a `SUV` class can inherit from the `Car` class, acquiring its properties and methods:

java

```
class SUV extends Car {
    // additional properties and methods specific to SUV
}
```

3. Encapsulation: Encapsulation is the bundling of data and methods within a class, hiding the internal details of how an object is implemented. It provides data abstraction and data protection. For example, consider a `Person` class with private data members `name` and `age`, and public getter and setter methods:

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

    public String getName() {
        return name;
    }

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

    // other methods
}
```

4. Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common parent class. It enables the same method to be used for different objects, resulting in code flexibility and extensibility. For example, a `calculateArea()` method can be defined in multiple classes inheriting from a common `Shape` class:

```
java
abstract class Shape {
    abstract void calculateArea();
}

class Circle extends Shape {
    void calculateArea() {
        // calculate area of circle
    }
}

class Rectangle extends Shape {
    void calculateArea() {
        // calculate area of rectangle
    }
}
```

5. Abstraction: Abstraction focuses on the essential features of an object, hiding unnecessary details. It is achieved by using abstract classes and interfaces. For example, a `Bank` interface can define common methods like `deposit()` and

`withdraw()`, which can then be implemented by various bank classes:

```
java
interface Bank {
    void deposit();
    void withdraw();
}

class HDFCBank implements Bank {
    public void deposit() {
        // deposit implementation for HDFC Bank
    }

    public void withdraw() {
        // withdraw implementation for HDFC Bank
    }
}

class ICICIBank implements Bank {
    public void deposit() {
        // deposit implementation for ICICI Bank
    }

    public void withdraw() {
        // withdraw implementation for ICICI Bank
    }
}
```

These are just a few examples of the OOP concepts in Java. Understanding and applying these concepts helps in writing clean, modular, and maintainable code.Object-Oriented Programming (OOP) is a programming paradigm that organizes data and functions into reusable structures called objects. Java is an object-oriented programming language, and it encapsulates real-world entities and their interactions using several fundamental concepts of OOP. Here are some important OOP concepts in Java with examples:

1. Classes and Objects: A class is a blueprint or template for creating objects. An object is an instance of a class. For example, consider a class named `Car` that defines properties like `color`, `brand`, and `model`. An object of the `Car` class can be created as follows:

java

```
Car car1 = new Car();
car1.color = "Red";
car1.brand = "Toyota";
car1.model = "Camry";
```

2. Inheritance: Inheritance allows classes to inherit properties and methods from another class. It promotes code reusability and improves the organization of code. For example, a `SUV` class can inherit from the `Car` class, acquiring its properties and methods:

java

```
class SUV extends Car {
    // additional properties and methods specific to SUV
}
```

3. Encapsulation: Encapsulation is the bundling of data and methods within a class, hiding the internal details of how an object is implemented. It provides data abstraction and data protection. For example, consider a `Person` class with private data members `name` and `age`, and public getter and setter methods:

java

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

    public String getName() {
        return name;
    }

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

    // other methods
}
```

4. Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common parent class. It enables the same method to be used for different objects, resulting in code flexibility and extensibility. For example, a `calculateArea()` method can be defined in multiple classes inheriting from a common `Shape` class:

```
java
abstract class Shape {
    abstract void calculateArea();
}

class Circle extends Shape {
    void calculateArea() {
        // calculate area of circle
    }
}

class Rectangle extends Shape {
    void calculateArea() {
        // calculate area of rectangle
    }
}
```

5. Abstraction: Abstraction focuses on the essential features of an object, hiding unnecessary details. It is achieved by using abstract classes and interfaces. For

example, a `Bank` interface can define common methods like `deposit()` and `withdraw()`, which can then be implemented by various bank classes:

java

```
interface Bank {
    void deposit();
    void withdraw();
}

class HDFCBank implements Bank {
    public void deposit() {
        // deposit implementation for HDFC Bank
    }

    public void withdraw() {
        // withdraw implementation for HDFC Bank
    }
}

class ICICIBank implements Bank {
    public void deposit() {
        // deposit implementation for ICICI Bank
    }

    public void withdraw() {
        // withdraw implementation for ICICI Bank
    }
```

```
}
```

These are just a few examples of the OOP concepts in Java. Understanding and applying these concepts helps in writing clean, modular, and maintainable code.

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

Ans:-

Example 1:

```
java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Example 2:

```
java
class Addition {
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        int sum = x + y;
        System.out.println("Sum: " + sum);
    }
}
```

Example 3:

java

```
class Circle {
    public static void main(String[] args) {
        double radius = 5.0;
        double pi = 3.14159;
        double area = pi * radius * radius;
        System.out.println("Area of the circle: " + area);
    }
}
```

Example 4:

java

```
class Concatenate {
    public static void main(String[] args) {
        String firstName = "John";
        String lastName = "Doe";
        String fullName = firstName + " " + lastName;
        System.out.println("Full Name: " + fullName);
    }
}
```

Example 5:

java

```
class GetInput {
```

```
    public static void main(String[] args) {
        System.out.print("Enter your name: ");
        Scanner scanner = new Scanner(System.in);
        String name = scanner.nextLine();
        System.out.println("Hello, " + name + "!");
        scanner.close();
    }
}
```

These programs demonstrate basic concepts such as printing "Hello, World!", performing addition, calculating the area of a circle, concatenating strings, and getting input from the user.

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:- 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:- 1, 3 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:- At compile time

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

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

Ans:- 0

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 and B both as required

Q6.Objects are the variables of the type_____?

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

Ans:- Class

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:- 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);
          }
         }
```

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

Ans:- 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:- Only 1, 3 and 4

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:-

Output

The output of the given Java program will be:

"Derived::show() called"

Explanation:
- The class "Main" is the main class of the program with the main method.
- In the main method, an object "b" of type "Base" is created and assigned a reference to an object of type "Derived".
- When the method "b.show()" is called, it will invoke the "show" method of the "Derived" class due to dynamic method dispatch. This is because the object being referred to by "b" is of type "Derived".
- The "show" method in the "Derived" class will print "Derived::show() called" to the console.

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:-

Output

The output of the given Java program will be:

"Derived::show() called"

Explanation:

- The class "Main" is the main class of the program with the main method.
- In the main method, an object "b" of type "Base" is created and assigned a reference to an object of type "Derived".
- Both the "show" methods in the "Base" and "Derived" classes are defined, but the "show" method in the "Base" class is declared with the "final" keyword.
- Even though the object being referred to by "b" is of type "Derived", the "show" method of the "Base" class will be called because it is declared as final and cannot be overridden by subclasses.
- The "show" method in the "Base" class will print "Base::show() called" to the console.

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:-

Output

The output of the program will be:

Derived::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:-

Output

The output of the given Java program will be:

"Test class Derived class"

Explanation:
- The class "Derived" has a method "getDetails" that prints "Derived class" to the console using the printf method.
- The class "Test" extends the "Derived" class and overrides the "getDetails" method.
- In the overridden method of the "Test" class, it first prints "Test class" to the console using the printf method.
- Then, it calls the "getDetails" method of the superclass (i.e., the "Derived" class) using the super keyword.
- In the main method, an object "obj" of type "Derived" is created and assigned a reference to an object of type "Test".
- When the method "obj.getDetails()" is called, it will invoke the overridden "getDetails" method in the "Test" class.
- As a result, "Test class" is printed first, followed by "Derived class" when the superclass method is called using the super keyword.

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:-

Output

The output of the following program will be:

Test class Name

Explanation:
- The program defines a class `Derived` with a `getDetails` method that takes a `String` parameter and prints "Derived class" followed by the value of the parameter.
- The program also defines a class `Test` that extends the `Derived` class and overrides the `getDetails` method. The overridden method also takes a `String` parameter and prints "Test class" followed by the value of the parameter. It then returns an `int` value of 0.
- In the `main` method of the `Test` class, an object of the `Test` class is created. This object calls the `getDetails` method with the argument "Name".
- Since the `getDetails` method is overridden in the `Test` class, it is the version of the method that gets called. Hence, "Test class Name" is printed to the console.

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:-

Output

[0:13 am, 01/07/2023] Rani Amma: The output of the code will be:

Adding to 100, x = 103
Adding to 0, y = 2 1 3

Explanation:
1. The variable "x" in the class "HasStatic" is declared as private and static, which means it can be accessed and modified by all instances of the class. It is initially set to 100.
2. Two objects of the class "HasStatic" are created - "hs1" and "hs2". Both of them increment the value of "x" by 1.
3. Another object "hs1" is created, and its "x" value is incremented by 1 (now x = 102). The static member "x" is incremented by 1 (now x = 101).
4. The value of "x" is printed as "Adding to 100, x = 103".
5. The class "test" has a static variable "y" which is initially set to 0.
6. Two objects of the class "test" are created - "t1" and "t2". Both of them increment the value of "y" by 1.
7. Another object "t1" is created, and its "y" value is incremented by 1 (now y = 2).
8. The value of "y" for "t1", "t2", and "test" is printed as "Adding to 0, y = 2 1 3".
[0:13 am, 01/07/2023] Rani Amma: 15

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:-

Output

The code provided will not compile because there is a syntax error in the code.

The error is in the method declaration of `m1(float f, int i);` in the `San` class. There is a semicolon (`;`) immediately after the method declaration, which is incorrect. The semicolon should not be present and it should instead have an opening curly brace (`{`) to start the method's body.

Here's the corrected code:

```
java
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);
    }
}
```

The corrected code will produce the following output:

float int method

Explanation:
- The `San` class has two overloaded methods named `m1`, one that takes an `int` and a `float` parameter, and another one that takes a `float` and an `int` parameter.
- In the `main` method, an object of the `San` class is created and the `m1` method is called with the arguments `20` and `20`.
- Since the arguments match the parameter types of the second `m1` method (`float` and `int`), that method will be invoked.
- Consequently, the output will be "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:-

Output

The given program has a compilation error and will not produce any output.

Explanation:
- In the main method, the variable "temp" is declared as an int but is assigned the value of null. However, null cannot be assigned to a variable of primitive type int. This will result in a compilation error.
- The variable "data" is declared as an Integer wrapper class object and is also assigned the value of null. This is allowed because null can be assigned to reference types.
- The System.out.println statement tries to print the values of "temp" and "data". However, since "temp" has a compilation error, the program cannot be successfully compiled.
- As a result, the program will not produce any output.

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:-

Output

The output will be "0 0".

Explanation:
In the given code, the class Test defines two protected instance variables, x and y. Protected variables can be accessed within the same package and by any subclass, even if they are in a different package.

In the main method of the Main class, an object of the Test class is created using the default constructor. Since the default constructor does not assign any values to the instance variables, the default values for int variables (0) are assigned to x and y.

The println statement prints the values of x and y, which are both 0, separated by a space. Hence, the output is "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:-

Output

The given code will produce the following output:

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

Explanation:

- The class `Test1` has a constructor that takes an integer argument `x` and prints the message "Constructor called " followed by the value of `x`.
- The class `Test2` has a member variable `t1` of type `Test1` which is initialized with a new instance of `Test1` with the value `10`.
- `Test2` also has a constructor that takes an integer argument `i`. This constructor initializes `t1` to a new instance of `Test1` with the value `i`.
- In the `main` method, a new instance of `Test2` is created with the value `5`. This will invoke the constructor of `Test2`, which in turn initializes `t1` to a new instance of `Test1` with the value `5`.
- Therefore, the output is "Constructor called 10" followed by "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:-

Output

The given code will produce the following output:

7

Explanation:
- The variable `x` is a two-dimensional array where each element is itself an array of integers.
- The variable `y` is also a two-dimensional array, but it is assigned the value of `x`.
- When `y` is assigned the value of `x`, it creates a new reference `y` that points to the same array as `x`.
- Therefore, `x` and `y` refer to the same two-dimensional array in memory.
- The statement `y[2][1]` accesses the element at index `2` of the outer array (which corresponds to the array `{6, 7, 8, 9}`), and then accesses the element at index `1` of that inner array (which corresponds to the value `7`).
- Hence, the output is `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:-

Output

The given code will produce the following output:

2

Explanation:

- The program defines two classes, `A` and `B`. `B` is a subclass of `A`.
- Both classes have a method called `display()`.
- In the `Dynamic_dispatch` class, an object `obj2` of class `B` is created and its variables `i` and `j` are assigned values `1` and `2` respectively.
- The variable `r` of type `A` is then declared and assigned the value of `obj2`. This is allowed because `B` is a subclass of `A` and inherits all its members.
- When the method `display()` is called using `r.display()`, the method implementation in class `B` will be called because `r` is referencing an object of class `B`. This is known as dynamic dispatch or runtime polymorphism.
- Inside the overridden `display()` method of class `B`, the value of variable `j` (which is `2`) is printed.
- Therefore, the output 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:-

Output

The given code will produce the following output:

2

Explanation:

- The program defines two classes, `A` and `B`. `B` is a subclass of `A`.
- Both classes have a method called `display()`.
- In the `method_overriding` class, an object `obj` of class `B` is created and its variables `i` and `j` are assigned values `1` and `2` respectively.
- When the method `display()` is called using `obj.display()`, the method implementation in class `B` will be called because `obj` is referencing an object of class `B`. This is known as method overriding.
- Inside the overridden `display()` method of class `B`, the value of variable `j` (which is `2`) is printed.
- Therefore, the output is `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:-

Output

The given code will produce the following output:

1 2

Explanation:

- The program defines two classes, `A` and `B`. `B` is a subclass of `A`.
- Class `A` has two instance variables, `i` and `j`. `i` is declared as public and `j` is declared as protected.
- Class `B` extends class `A` and defines an additional instance variable `j` which hides the inherited `j` variable from class `A`.
- Class `B` also defines a method called `display()`.
- In the `Output` class, an object `obj` of class `B` is created and its variables `i` and `j` are assigned values `1` and `2` respectively.
- When the method `display()` is called using `obj.display()`, the overridden `display()` method in class `B` is executed.
- Inside the `display()` method, `super.j = 3;` is used to assign the value `3` to the inherited `j` variable from class `A`. This allows access to the inherited `j` variable since it is declared as protected in class `A`.
- The statement `System.out.println(i + " " + j);` prints the values of `i` and `j`, which are `1` and `2` respectively.
- Therefore, the output is `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:-

Output

The given code will produce the following output:

1 2

Explanation:
- The program defines three classes, `A`, `B`, and `super_use`.
- Class `A` has two public instance variables, `i` and `j`. It also has a constructor `A()` which initializes `i` to `1` and `j` to `2`.
- Class `B` extends class `A` and defines an additional instance variable `a`. It also has a constructor `B()` which calls the `super()` constructor of class `A`.
- The `super()` constructor is used to initialize the inherited instance variables of class `A` in the constructor of class `B`.
- In the `super_use` class, an object `obj` of class `B` is created and its instance variables `i` and `j` are printed using `System.out.println(obj.i + " " + obj.j)`.
- Since `B` is a subclass of `A`, it inherits the instance variables `i` and `j` from class `A`.
- The constructor `B()` initializes these inherited variables using the `super()` constructor of class `A` which sets `i` to `1` and `j` to `2`.
- Therefore, the output is `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:-

Output

The given code will produce the following output:

obj1.a = 2 obj1.b = 3
obj2.a = 3 obj1.b = 4

Explanation:

- The program defines a class `Test` with two instance variables `a` and `b`, both initially set to `1` and `2` respectively.
- The `func()` method of class `Test` takes an object of type `Test` as a parameter and returns a new `Test` object.
- Inside the `func()` method, a new `Test` object `obj3` is created and assigned to `obj`.
- Then, the `a` instance variable of `obj3` is assigned the value of the sum of `obj.a++` and `++obj.b`.
  - `obj.a++` means that the current value of `obj.a` is used first and then it is incremented by 1. So, `obj.a++` evaluates to `1` and `obj.a` becomes `2`.
  - `++obj.b` increments `obj.b` by 1 before its value is used. So, `++obj.b` evaluates to `3` and `obj.b` becomes `3`.
  - Therefore, `obj3.a` becomes `1 + 3 = 4`.
- Next, the `obj.b` is assigned the value of `obj.b`, which is `3`. So, `obj.b` remains `3`.
- Finally, `obj3` is returned from the `func()` method.
- In the `main()` method, a new `Test` object `obj1` is created.
- Then, `obj1.func(obj1)` is called and the returned `obj3` is assigned to `obj2`.

- The instance variables `a` and `b` of `obj1` and `obj2` are printed using `System.out.println()`.
- At this point, the values of `obj1.a` and `obj1.b` are updated by the `func()` method, so `obj1.a` becomes `2` and `obj1.b` becomes `3`.
- The values of `obj2.a` and `obj2.b` are the same as `obj1.a` and `obj1.b` because `obj3` is just a reference to `obj1`.
- Therefore, the output is:

```
obj1.a = 2 obj1.b = 3
obj2.a = 2 obj1.b = 3
```