Object-Oriented Programming (OOP) is a programming paradigm that focuses on organizing code into objects, which are instances of classes. Java is an object-oriented programming language that incorporates key OOP concepts to enable developers to build modular, reusable, and maintainable code. Here are some essential OOP concepts in Java:

1. Classes and Objects: A class is a blueprint or template that defines the properties and behaviors of objects. Objects are instances of classes that represent specific entities or concepts. For example, a "Car" class can define the attributes (color, model, etc.) and behaviors (start, accelerate, etc.) of a car object.

2. Encapsulation: Encapsulation is the principle of hiding the internal details of an object and providing access to its functionalities through well-defined interfaces. It helps in achieving data abstraction and code security. For instance, in a "Person" class, the private variables (e.g., name, age) can be accessed and modified using public methods (e.g., getName(), setName()).

3. Inheritance: Inheritance allows the creation of new classes (derived or child classes) based on existing classes (base or parent classes). The derived classes inherit the properties and behaviors of the parent class, facilitating code reuse and promoting a hierarchical organization. For example, a "Student" class can inherit from a "Person" class to gain its attributes and methods.

4. Polymorphism: Polymorphism enables objects of different classes to be treated as objects of a common superclass. It allows methods to be overridden in derived classes, providing different implementations while maintaining a consistent interface. For instance, a "Shape" superclass can have a method called "calculate Area()", which can be overridden in specific shape subclasses like "Circle" or "Rectangle".

5. Abstraction: Abstraction focuses on defining essential features and hiding unnecessary details. It simplifies complex systems by breaking them down into manageable modules. Abstract classes and interfaces provide a way to achieve abstraction in Java. Abstract classes cannot be instantiated and serve as templates for subclasses. Interfaces define a contract of methods that classes must implement. They allow multiple inheritances of behaviours.

3Ans: A. Making at least one member function as pure virtual function

1Ans: Making at least one member function as pure virtual function

2Ans: 1, 3 and 4

3Ans: At compile time

4Ans: 0

5Ans: Dot Operator

6Ans: Class

7Ans: Private data

8Ans: 0

9Ans: Only 1, 3 and 4

10Ans: In the program, there are two classes: "Base" and "Derived". The "Derived" class is a subclass of the "Base" class and it overrides the "show()" method defined in the "Base" class.

In the "main" method of the "Main" class, an object of type "Derived" is created and assigned to a reference variable of type "Base". This is known as polymorphism and allows a subclass object to be referred to by a superclass reference.

When the "show()" method is called on the "Base" object "b", the JVM determines the actual type of the object at runtime, which is "Derived". Therefore, the overridden version of the "show()" method in the "Derived" class is invoked.

Hence, the output will be "Derived::show() called".

11Ans: In the program, there are three classes: "Base", "Derived", and "Main". The "Derived" class is a subclass of the "Base" class and it attempts to override the "show()" method defined in the "Base" class.

In the "Base" class, the "show()" method is declared with the "final" keyword. The "final" keyword when used with a method means that the method cannot be overridden by any subclass. Therefore, the "show()" method in the "Base" class is not meant to be overridden.

In the "main" method of the "Main" class, an object of type "Derived" is created and assigned to a reference variable of type "Base". This is allowed because a subclass object can be referred to by a superclass reference.

When the "show()" method is called on the "Base" object "b", the JVM still determines the actual type of the object at runtime, which is "Derived". However, since the "show()" method is declared as final in the "Base" class, it cannot be overridden by the "Derived" class. Therefore, the original implementation of the "show()" method in the "Base" class is invoked.

Hence, the output will be "Derived::show() called".

12Ans: In the program, there are three classes: "Base", "Derived", and "Main". Both the "Base" and "Derived" classes have a static method named "show()" which is not overridden.

In the "main" method of the "Main" class, an object of type "Derived" is created and assigned to a reference variable of type "Base". This is allowed because a subclass object can be referred to by a superclass reference.

When a static method is called, it is resolved at compile-time based on the declared type of the reference variable, not the actual type of the object. In this case, the reference variable "b" is of type "Base", so the static method "show()" in the "Base" class is invoked.

Therefore, the output will be "Base::show() called".

13Ans:

In the program, there are two classes: "Derived" and "Test". The class "Test" extends the class "Derived" and overrides the method "getDetails()".

In the "getDetails()" method of the "Test" class, the first statement prints "Test class". Then, the "super.getDetails()" statement is used to invoke the overridden method "getDetails()" of the superclass, which is the "Derived" class.

In the "main" method, an object of type "Test" is created and assigned to a reference variable of type "Derived". This is allowed because a subclass object can be referred to by a superclass reference.

When the "getDetails()" method is called on the "Derived" object "obj", the JVM determines the actual type of the object at runtime, which is "Test". Therefore, the overridden version of the "getDetails()" method in the "Test" class is invoked.

The output will be "Test class Derived class", as the "getDetails()" method in the "Test" class is invoked first, followed by the "super.getDetails()" statement, which calls the "getDetails()" method of the superclass.

Hence, the output will be "Test class Derived class".

14 Ans: In the program, there are two classes: "Derived" and "Test". The class "Test" extends the class "Derived" and overrides the method "getDetails(String temp)".

In the "getDetails(String temp)" method of the "Test" class, the first statement prints "Test class" followed by the value of the "temp" parameter. The method also returns an integer value of 0.

In the "main" method, an object of type "Test" is created. When the "getDetails(String temp)" method is called on the "Test" object "obj" with the argument "Name", the overridden version of the method in the "Test" class is invoked.

Since method overriding is based on the signature (name and parameters) of the method, and the return type is not considered, the overridden method in the "Test" class correctly overrides the method in the "Derived" class.

Therefore, the output will be "Test class Name", as the "getDetails(String temp)" method of the "Test" class is invoked.

Note: If the return type of the overridden method in the "Test" class were different from the return type of the method in the "Derived" class, it would result in a compile-time error. However, in this case, the return types of both methods match (both are integers).

Hence, the output will be "Test class Name".

15Ans: In the program, there are two classes: "test" and "HasStatic". The "HasStatic" class has a private static variable "x" and the "test" class has a public static variable "y".

In the "main" method of the "HasStatic" class, several instances of the class are created and manipulated.

Therefore, the output will be: Adding to 100, x = 103 Adding to 0, y = 3 2 3

16Ans: The given code has a syntax error in the method declaration of "m1(float f, int i);". There is an extra semicolon (;) at the end of the method declaration, which is not allowed in Java.

17Ans: In the program, there is a variable declaration int temp = null; and an object declaration Integer data = null;. Both of these statements will cause a compilation error.

In Java, primitive types such as int cannot hold the value null because they are not reference types. The int data type represents a primitive integer value and cannot be assigned a null value. Therefore, the statement int temp = null; will result in a compilation error.

On the other hand, the Integer class is a wrapper class that encapsulates an int value and allows it to be treated as an object. The Integer class can hold a null value because it is a reference type. However, when attempting to concatenate a null value with a string using the + operator in the System.out.println() statement, a NullPointerException will occur.

Therefore, the given program will not compile successfully, and an error will be encountered during compilation.

18Ans: The class "Test" has two protected integer variables, "x" and "y". By default, when no value is assigned to an integer variable, it is initialized to 0.

In the "main" method of the "Main" class, an object of type "Test" is created using the default constructor. Since the "x" and "y" variables are not explicitly initialized, they are assigned the default value of 0.

When printing the values of "t.x" and "t.y" using the System.out.println() statement, the values of "x" and "y" will be concatenated with a space in between.

Therefore, the output will be "0 0".

19Ans: The "Test1" class has a constructor that takes an integer parameter. When an object of the "Test1" class is created using the statement Test1 t1 = new Test1(10);, the constructor of "Test1" is called with the value 10. This prints "Constructor called 10".

The "Test2" class has a member variable "t1" of type "Test1". When an object of the "Test2" class is created using the statement Test2 t2 = new Test2(5);, the constructor of "Test2" is called with the value 5. Inside the constructor of "Test2", a new object of "Test1" is created using the statement t1 = new Test1(i);, where "i" is the parameter value passed to the constructor. This prints "Constructor called 5".

Therefore, the output will be: "Constructor called 10 Constructor called 5"

20Ans: In the program, a 2D array x is declared and initialized with three rows of varying lengths: {{1,2}, {3,4,5}, {6,7,8,9}}.

Then, another 2D array y is declared and assigned the value of x using the statement int [][]y = x;. This means that y now refers to the same memory location as x.

Finally, the value at index [2][1] of y is printed using the statement System.out.println(y[2][1]);. Since y is referencing the same array as x, accessing y[2][1] will retrieve the element at the second row and first column of the array x, which is 7.

Therefore, the output will be "7".

21 Ans: In the program, there are three classes: A, B, and Dynamic_dispatch.

The class A has an integer variable i and a method display() which prints the value of i.

The class B extends class A and adds an integer variable j and overrides the display() method to print the value of j.

In the main method of the Dynamic_dispatch class, an object obj2 of class B is created. The variables i and j of obj2 are assigned the values 1 and 2 respectively.

Then, a reference variable r of type A is declared and assigned the object obj2. Since obj2 is an instance of class B, it can be assigned to a reference variable of type A due to polymorphism.

When the display() method is called on the object r, dynamic dispatch or late binding occurs. This means that the method implementation is determined at runtime based on the actual type of the object, which is class B.

Therefore, the output will be "2", as the display() method in class B is called and it prints the value of variable j, which is 2.

22Ans: The class A has an integer variable i and a method display() which prints the value of i.

The class B extends class A and adds an integer variable j and overrides the display() method to print the value of j.

In the main method of the method_overriding class, an object obj of class B is created. The variables i and j of obj are assigned the values 1 and 2 respectively.

When the display() method is called on the object obj, the method implementation is determined at runtime based on the actual type of the object, which is class B. Since the display() method is overridden in class B, it prints the value of variable j, which is 2.

Therefore, the output will be "2".

23 Ans: The class A has two integer variables: i with public access and j with protected access.

The class B extends class A and adds an integer variable j (which hides the j variable from class A) and a display() method.

In the display() method, super.j = 3; is used to assign the value 3 to the j variable inherited from class A. The super keyword is used to refer to the superclass (class A in this case).

In the main method of the Output class, an object obj of class B is created. The variables i and j of obj are assigned the values 1 and 2 respectively.

When the display() method is called on the object obj, it prints the values of i and j. Since i is a public variable, it can be accessed directly. The j variable in class B is accessed without any prefix, referring to the variable in the current class. Therefore, the output will be "1 2".

Therefore, the output of the program will be "1 2".

24 Ans: The class A has two public integer variables i and j. It also has a default constructor that initializes i to 1 and j to 2.

The class B extends class A and adds an integer variable a. It also has a default constructor that invokes the default constructor of class A using the super() keyword.

In the main method of the super_use class, an object obj of class B is created. This triggers the default constructor of class B, which in turn invokes the default constructor of class A using super(). As a result, i and j are initialized to 1 and 2 respectively.

Finally, the values of obj.i and obj.j are printed using System.out.println(obj.i + " " + obj.j). Since i and j are public variables in class A, they can be accessed directly from class B. Therefore, the output will be "1 2".

Therefore, the output of the program will be "1 2".

25 Ans: In the program, there is a class Test with two integer variables a and b. The class also has a method func() that takes a Test object as a parameter.

In the func() method, a new Test object obj3 is created and assigned the value of obj. Then, the value of obj.a++ + ++obj.b is assigned to obj3.a. Here, obj.a++ evaluates to the current value of obj.a (1) and then increments obj.a to 2. Similarly, ++obj.b increments obj.b from 2 to 3. Therefore, obj3.a becomes 1 + 3 = 4. The value of obj.b remains unchanged.

In the main method, an object obj1 of class Test is created. Then, the func() method is called on obj1 and the returned Test object is assigned to obj2.

After the method calls, the values of obj1.a and obj1.b are printed. Since the func() method modified the object obj1, the value of obj1.a becomes 2 (due to the increment in the func() method) and obj1.b becomes 3 (unchanged).

Similarly, the values of obj2.a and obj2.b are printed. Since obj2 is assigned the returned Test object from the func() method, it has the same values as obj1. Therefore, obj2.a is 2 and obj2.b is 3.

Therefore, the output of the program will be "obj1.a = 2 obj1.b = 3" and "obj2.a = 2 obj2.b = 3".