**Final keyword**

|  |
| --- |
| In java final keyword is used to declare final fields, final methods and final classes. |

**Final fields**

|  |
| --- |
| The variables which are declared as final are called as constant variables whose values can’t be changeable. |

**Final methods**

|  |
| --- |
| We can’t override final methods |

**Final class**

|  |
| --- |
| We can’t inherit a final class  Ex: System class is final so we can’t inherit it |

**Interfaces**

**What is an interface?**

|  |
| --- |
| 1. Interface is a pure abstract class which contains only abstract methods and final fields 2. All the methods of an interface are by default declared as public and abstract. 3. All the fields of an interface are by default declared as public, static and final. 4. Every field must be initialized at the time of declaration only. 5. A class can inherit an interface by using a keyword called implements 6. One interface can extend another interface 7. We can’t create objects for interfaces 8. We can declare a reference variable by using interface name   When we can use extends keyword?   1. If a class inherits another class we have to use extends 2. if an interface inherits another interface, then we have to use extends   When we can use implements keyword?   1. If a class inherits an interface, we have to use implements keyword |

**CalcInf.java**

public interface CalcInf

{

    int a=100;     //public static final int a=100;

    int add(int a,int b);  //public abstract int add(int a,int b);

    int sub(int a,int b);

    int multi(int a,int b);

    int div(int a,int b);

    int mod(int a,int b);

}

**Calc.java**

public class Calc implements CalcInf

{

    @Override

    public int add(int a,int b)

    {return a+b;

    }

    @Override

    public int sub(int a,int b)

    {return a-b;}

    @Override

    public int multi(int a,int b)

    {return a\*b;}

    @Override

    public int div(int a,int b)

    {return a/b;}

    @Override

    public int mod(int a,int b)

    {return a%b;}

}

**Main2.java**

import java.util.Scanner;

public class Main2

{

    public static void main(String[] args)

    {

        Calc calc=new Calc();

        Scanner scan=new Scanner(System.in);

        System.out.println("Enter two int values:\t");

        int a=scan.nextInt();

        int b=scan.nextInt();

        System.out.println(calc.add(a,b));

        System.out.println(calc.sub(a,b));

        System.out.println(calc.multi(a,b));

        System.out.println(calc.div(a,b));

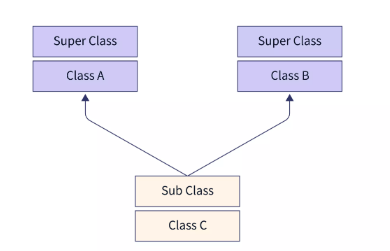
        System.out.println(calc.mod(a,b));

    }

}

**Multiple Inheritance**

|  |
| --- |
| 1. If a class inherits multiple bases classes at a time, it is known as multiple inheritance. 2. Java doesn’t support multiple inheritance directly but we can achieve it through interfaces 3. In Java it is possible to inherit multiple interfaces by a single class at a time. |

****

**Example on implementing multiple inheritance by using interfaces**

interface Inf1

{

    int add(int a,int b);

}

interface Inf2 extends Inf1{

    int sub(int a,int b);

}

interface Inf3

{

    int multi(int a,int b);

}

abstract class ABParent

{   abstract int div(int a,int b);

    int mod(int a,int b)

    {   return a%b;

    }

}

class MyCalc extends ABParent implements Inf2,Inf3

{

    int div(int a,int b)

    {

        return a/b;

    }

    public int add(int a,int b)

    {return a+b;

    }

    public int sub(int a,int b)

    {return a-b;

    }

    public int multi(int a,int b)

    {

        return a\*b;

    }

}

public class InterfaceDemo2

{

    public static void main(String[] args)

    {

        MyCalc calc=new MyCalc();

        System.out.println(calc.add(10, 2));

        System.out.println(calc.sub(10, 2));

        System.out.println(calc.multi(10, 2));

        System.out.println(calc.div(10, 2));

        System.out.println(calc.mod(10, 2));

    }

}

Output:

12

8

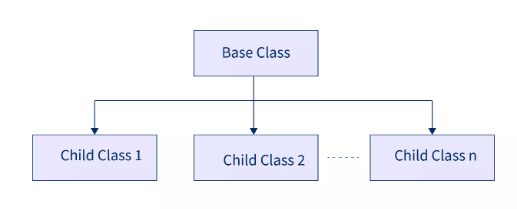
20

5

0

**Hierarchical Inheritance**

|  |
| --- |
| If a single base class is inherited by multiple child classes, then it is called as hierarchical inheritance. |

****

**Abstraction, loose coupling and hierarchical implements example**

interface Shape

{

    void draw();

}

class Rectangle implements Shape

{   public void draw()

    {System.out.println("Rectangle is drawn...");

    }

}

class Triangle implements Shape{

    public void draw()

    {System.out.println("Triangle is drawn...");

    }

}

class Circle implements Shape

{

    public void draw()

    {System.out.println("Circle is drawn...");

    }

}

public class Drawing

{

    public static Shape getShape(int opt)

    {

        Shape s=null;

        if(opt==1)

        {   s=new Rectangle();

        }else if(opt==2)

        {   s=new Triangle();

        }else if(opt==3)

        {   s=new Circle();

        }

        return s;

    }

}

import java.util.Scanner;

public class Usage

{

    public static void main(String[] args)

    {

        //i am given a class called Drawing and it is having a static method called getShape

        //if you pass 1 to the getShape()  method it returns a Shape object which contains draw() method which is used to draw a Rectangle

        //if you pass 2 to the getShape()  method it returns a Shape object which contains draw() method which is used to draw a Triangle

        //if you pass 3 to the getShape() method it returns a Shape object which contains draw() method which is used to draw a Rectangle

        Scanner scan=new Scanner(System.in);

        System.out.println("1.Draw Rectangle");

        System.out.println("2.Draw Triangle");

        System.out.println("3.Draw Circle");

        System.out.print("option(1/2/3):\t");

        int opt=scan.nextInt();

        Shape s=Drawing.getShape(opt);

        if(s!=null)

            s.draw();

        else

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

    }

}

**What is the advantages of interfaces**

1. Abstraction
2. Loose coupling
3. Multiple inheritance
4. Here we mention the feature (tasks or methods) which we are going to implement in future.

**What is an inner class?**

|  |
| --- |
| We can write a class within another class; these classes are called as inner classes.   1. Static inner class 2. Non-static inner class 3. Anonymous inner class 4. Local inner class |

**Anonymous Inner classes**

|  |
| --- |
| 1. It is an inner class which can be created, by using already existed classes or interfaces. 2. Anonymous inner class behaves like child class of already existed class. |

**Example on anonymous inner classes**

class Don

{

    String hero="Amitab";

    String heroIn="ZenathAman";

    String story="Don runs mafia, commissioner kills don and replaces him with duplicate person, and destroys mafia";

    String movieName="DON";

    void songs()

    {   System.out.println("Running, walking, excercises");

    }

    void fights()

    {

        System.out.println("dorlude, musti yuddam");

    }

    void show()

    {

        System.out.println("Starring "+hero+" IN & As "+movieName);

        System.out.println("Beatiful Herein "+heroIn);

        System.out.println(story);

        songs();

        fights();

    }

}

class Outer

{

    //Here i am writing child class of Don class and also creating object for child class.

    //here the child class is having no name so it is called as anonymous class

    //i have written this child(anonymous) class within another class so it is inner class

    static Don billa=new Don(){

        //instance block

        {

            hero="Prabhas";

            heroIn="Anushka";

            movieName="Billa";

        }

        void songs()

        {   System.out.println("Western Dance");

        }

        void fights()

        {

            System.out.println("Martial Arts");

        }

    };

    static class Inner{

        int a,b;

        static int s;

        static void fun1()

        {   System.out.println("static method of Inner class");

        }

        void display()

        {}

    }

    class Inner2{

        int a,b;

        static int s;

        static void fun1()

        {   System.out.println("static method of Inner class");

        }

        void display()

        {}

    }

}

public class InnerClassDemo {

    public static void main(String[] args) {

        Outer.billa.show();

    }

}

Output:

Starring Prabhas IN & As Billa

Beatiful Herein Anushka

Don runs mafia, commissioner kills don and replaces him with duplicate person, and destroys mafia

Western Dance

Martial Arts

**Another example on anonymous inner class (creating it by implementing an interface)**

interface Greet

{

    String hi(String name); //public abstract String hi(String name);

}

public class AnonymousInner2

{

    public static void main(String[] args) {

        Greet gt=new Greet(){

            public String hi(String name)

            {

                return "Hi "+name;

            }

        };

        System.out.println(gt.hi("Madhu"));

    }

}

Output:

Hi Madhu

**Functional interface**

**What is functional interface?**

|  |
| --- |
| 1. It is an interface which contains only one abstract method 2. It can contain fields 3. It can contains default methods and static methods also (1.8 version) |

**How to write a lambda expression?**

|  |
| --- |
| 1. Writing a lambda expression means implementing functional interface 2. Lambda expression makes the code simpler |

**Example on functional interfaces and lambda expressions**

//Greet is a functional interface

interface Greet

{

    String hi(String name); //public abstract String hi(String name);

}

public class AnonymousInner2

{

    public static void main(String[] args)

    {

        // Greet gt=new Greet(){

        //     public String hi(String name)

        //     {

        //         return "Hi "+name;

        //     }

        // };

        Greet gt=(String name)->"Hi "+name;

        System.out.println(gt.hi("Madhu"));

    }

}

Output:

Hi Madhu

**Lambda expression with default methods, static methods and final fields existed in an interfaces**

interface One

{

    int a=100;

    //default methods will be inherited into child classes

    //it is also possible to override them if needed in child classes.

    default String hello(String name)

    {

        return "Hello "+name;

    }

    //static method of interface will not be inherted into child classes

    static void fun1()

    {

        System.out.println("static method of interface....");

    }

    String bye(String name);

}

// class Simple implements One

// {

//     public String bye(String name)

//     {

//         return "Bye "+name+" Sir";

//     }

// }

public class LambdaDemo1

{

    public static void main(String[] args)

    {

        //One o1=new Simple();//upcasting

        One o1=(String name)->"Bye "+name+" Sir";

        System.out.println(o1.bye("Madhu"));

        System.out.println(o1.hello("madhu"));

        One.fun1();

    }

}

Output:

Bye Madhu Sir

Hello madhu

static method of interface....

**Another example on lambda expressions and functional interfaces**

import java.util.Scanner;

interface One

{

    String bye(String name);

}

interface Two

{

    void add(int a,int b);

}

interface Three{

    int add();

}

public class LambdaDemo1

{

    public static void main(String[] args)

    {

        One o1=(String name)->"Bye "+name+" Sir";

        Two t=(int a,int b)->{

            int c=a+b;

            System.out.printf("%d + %d = %d%n",a,b,c);

        };

        Three thr=()->{

            Scanner scan=new Scanner(System.in);

            System.out.print("Enter a,b:\t");

            int a=scan.nextInt();

            int b=scan.nextInt();

            return a+b;

        };

        System.out.println(o1.bye("Madhu"));

        t.add(10, 2);

    }

}

Output:

Bye Madhu Sir

10 + 2 = 12

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while..**
    26. **Functions(methods)**
    27. **Class**
    28. **Object**
    29. **Instance variables**
    30. **Static variables**
    31. **Instance methods**
    32. **Static methods**
    33. **Instance block**
    34. **Static block**
    35. **Constructors**
    36. **This keyword**
    37. **String constant pool**
    38. **Accessor and mutator methods**
    39. **Inheritance**
    40. **Super()**
    41. **Method overloading**
    42. **Method overriding**
    43. **Upcasting & Down casting**
    44. **Static and dynamic binding**
    45. **Compile time and runtime Polymorphism**
    46. **Abstract classes**
    47. **Interfaces**
    48. **Inner classes**
    49. **Anonymous inner classes**
    50. **Functional interfaces**
    51. **Lambda expressions**
    52. **Singleton**
    53. **Immutable object**
    54. **Creating record using contextual keyword called ‘record’**