Program: 15

Aim – Create a class Box that uses a parameterized constructor to initialize the dimensions of a box. The dimensions of the Box are width, height, depth. The class should have a method that can return the volume of the box. Create an object of the Box class and test the functionalities.

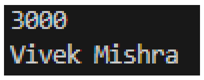
Theory – Parameterized constructors help to create objects with states defined by the programmer. Objects in java can be initialized with the default constructor or by a parameterized constructor. Initializing objects with parameterized constructors requires the same number and order of arguments to be passed by the user concerning the parameterized constructor being used.

Source Code:

```
public class pr15 {
    public static class Box {
        int width;
        int hight;
        int length;

        public Box(int wi, int hi, int len) {
            this.width = wi;
            this.hight = hi;
            this.length = len;
        }
        public int s_volume() {
            return width * length * hight;
        }
    }
    public static void main(String[] args) {
        Box B1 = new Box(10, 20, 15);
        System.out.print(B1.s_volume());
    }
}
```

Output:



```
3000
Vivek Mishra
```

Program: 16

Aim – WAP to display the use of this keyword.

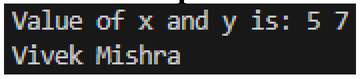
Theory – Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Source Code:

```
public class pr16 {
    public static class rp{
        int x;
        int y;
        public rp(int x, int y){
            this.x=x;
            this.y=y;
        }
        void show(){
            System.out.println("Value of x and y is: "+x+" "+y);
        }
    }
    public static void main(String[] args) {
        rp x=new rp(5, 7);
        x.show();
        System.out.println("Vivek Mishra");
    }
}
```

Output:



```
Value of x and y is: 5 7
Vivek Mishra
```

Program: 17

Aim – Write a program that can count the number of instances created for the class.

Theory – The idea is to use static member in the class to count objects. A static member is shared by all objects of the class, all static data is initialized to zero when the first object is created if no other initialization is present, and constructor and static member function can only access static data member, other static member functions and any other functions from outside the class.

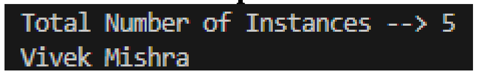
Source Code:

```
public class pr17 {
   private static int count;
   public static class instance{
      instance(){

         count++;
      }
   }
   public static void main(String args[]){
      instance i1 = new instance();
      instance i2 = new instance();
      instance i3 = new instance();
      instance i4 = new instance();
      instance i5 = new instance();
      System.out.print("Total Number of Instances --> " + count);
         System.out.println("Vivek Mishra");
   }

}
```

Output:



```
Total Number of Instances --> 5
Vivek Mishra
```

Program: 18

Aim – Java Program to get the cube of a given number using the static method.

Theory – If you apply static keyword with any method, it is known as static method.

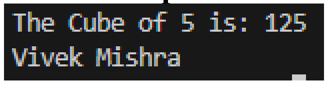
1. A static method belongs to the class rather than the object of a class.
2. A static method can be invoked without the need for creating an instance of a class.
3. A static method can access static data member and can change the value of it.

Source Code :

```
public class pr19 {
    public static int getCube(int num) {
        return num * num * num;
    }
    public static void main(String[] args) {
        int number = 5;
        int cube = getCube(number);
        System.out.println("The Cube of " + number + " is: " + cube);
        System.out.println("Vivek Mishra");
    }

}
```

Output:



```
The Cube of 5 is: 125
Vivek Mishra
```

PROGRAM 19

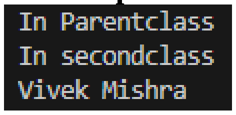
Aim: Write a program that implements method overriding.

Theory: If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Source Code:

```
public class pr19{
   public static class first{
      int x;
      void meth(){
         System.out.println("In Parentclass "+ x);
      }
   }
   public static class sec extends first{
      void meth(){
         System.out.println("In secondclass");
      }
   }
   public static void main(String[] args) {
      first A = new first();
      A.x=5;
      A.meth();
      sec B = new sec();
      B.meth();
      System.out.println("Vivek Mishra ");
   }
}
```

Output:



```
In Parentclass
In secondclass
Vivek Mishra
```

PROGRAM: 20

Aim: Write a program to illustrate simple inheritance.

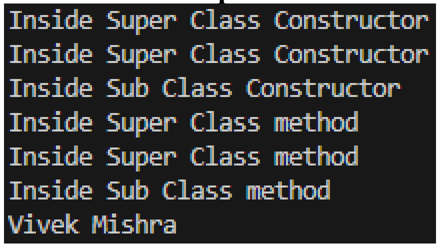
Theory: When a class inherits another class, it is known as a single inheritance or simple inheritance.

Source Code:

```
public class pr20 {
     public static class Super {
        Super() {
            System.out.println("Inside Super Class Constructor");
        }
        void meth1() {
            System.out.println("Inside Super Class method");
        }
    }
    public static class Sub extends Super {
        Sub() {
            System.out.println("Inside Sub Class Constructor");
        }
        void meth2() {
            System.out.println("Inside Sub Class method");
        }
    }
    public static void main(String[] args) {
        Super s1 = new Super();
        Sub s2 =new Sub();
        s1.meth1();
        s2.meth1();
        s2.meth2();
       System.out.println("Vivek Mishra ");

    }
}
```

Output:



```
Inside Super Class Constructor
Inside Super Class Constructor
Inside Sub Class Constructor
Inside Super Class method
Inside Super Class method
Inside Sub Class method
Vivek Mishra
```

PROGRAM: 21

Aim: Write a program to illustrate multilevel inheritance.

Theory: In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.

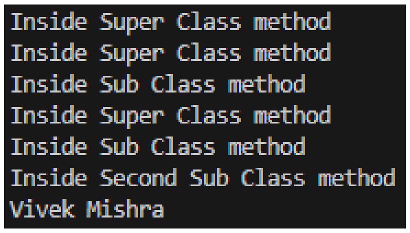
Source Code:

```
public class pr21 {

    public static class Super {
        void meth1() {
            System.out.println("Inside Super Class method");
        }
    }
    public static class Sub extends Super {
        void meth2() {
            System.out.println("Inside Sub Class method");
        }
    }
    public static class SecSub extends Sub{
        void meth3() {
            System.out.println("Inside Second Sub Class method");
        }
    }
    public static void main(String[] args) {
        Super s1 = new Super();
        Sub s2 =new Sub();
        SecSub s3 = new SecSub();
        s1.meth1();
        s2.meth1();
        s2.meth2();
        s3.meth1();
        s3.meth2();
        s3.meth3();
        System.out.println("Vivek Mishra");

    }
}
```

Outout:



```
Inside Super Class method
Inside Super Class method
Inside Sub Class method
Inside Super Class method
Inside Sub Class method
Inside Second Sub Class method
Vivek Mishra
```

PROGRAM: 22

Aim: Write a program illustrating all uses of super keyword.

Theory: The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

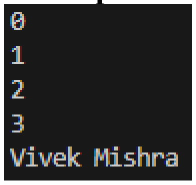
Source Code:

```
public class pr22 {
    public static class C1{
      int x=0;

    }
    public static class C2 extends C1{
        int y;
        C2(int a, int b){
            super.x=a;
            this.y=b;
        }
    }
    public static void main(String[] args) {
        C1 s1=new C1();
        C2 s2=new C2(2 , 3);
        System.out.println(s1.x);
        s1.x=1;
        System.out.println(s1.x);
        System.out.println(s2.x);
        System.out.println(s2.y);
        System.out.println("Vivek Mishra");

    }
}
```

Output:



```
0
1
2
3
Vivek Mishra
```

PROGRAM: 23

Aim: Write a program to show dynamic polymorphism and interfaces.

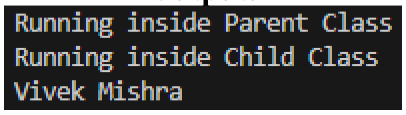
Theory: Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or **dynamic method dispatch**. An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction.

Source Code:

```
public class pr23 {
   interface MyClass {
       void Run();
   }
   static class Parent1 implements MyClass {
       public void Run() {
           System.out.println("Running inside Parent Class");
       }
   }
   static class Child1 extends Parent1 implements MyClass {
       public void Run() {
           System.out.println("Running inside Child Class");
       }
   }
   public static void main(String[] args) {
       Parent1 p1 = new Parent1();
       p1.Run();
       Parent1 p2 = new Child1();
       p2.Run();
       System.out.println("Vivek Mishra");

   }
}
```

Output:



```
Running inside Parent Class
Running inside Child Class
Vivek Mishra
```

PROGRAM: 24

Aim: Create an abstract class Shape. Let Rectangle and Triangle inherit this Shape Class. Add necessary functions.

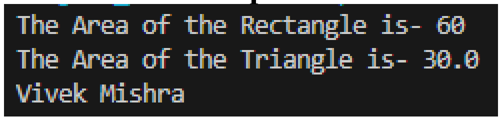
Theory: A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Source Code:

```
public class pr24 {
    static abstract class Shape {
        abstract void Area(int x,int y);
    }
    static class Rectangle extends Shape {
        void Area(int x, int y) {
            System.out.println("The Area of the Rectangle is- "+ x*y);
        }
    }
    static class Triangle extends Shape {
        void Area(int x, int y) {
            System.out.println("The Area of the Triangle is- "+ 0.5*x*y);
        }
    }

    public static void main(String[] args) {
        Rectangle r = new Rectangle();
        r.Area(5,12);
        Triangle t = new Triangle();
        t.Area(5,12);
        System.out.println("Vviek Mishra");
    }
}
```

Output:



```
The Area of the Rectangle is- 60
The Area of the Triangle is- 30.0
Vivek Mishra
```

PROGRAM: 25

Aim: Write a java package to show dynamic polymorphism and interfaces.

Theory: Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or **dynamic method dispatch**. An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction.

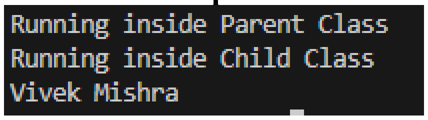
Source Code:

```
package pr;
public class pr25 {
    static interface MyClass {
        void Run();
    }
    static class Parent1 implements MyClass {
        public void Run() {
            System.out.println("Running inside Parent Class");
        }
    }
    static class Child1 extends Parent1 implements MyClass {
        public void Run() {
            System.out.println("Running inside Child Class");
        }
    }
    public static void main(String[] args) {
        Parent1 p1 = new Parent1();
        p1.Run();
        Parent1 p2 = new Child1();
        p2.Run();
       System.out.println("Vviek Mishra");

    }

}
```

Output:



```
Running inside Parent Class
Running inside Child Class
Vivek Mishra
```

PROGRAM: 26

Aim: Write an application that creates an 'interface' and implements it.

Theory: An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction.

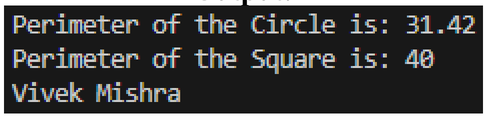
Source Code:

```
public class pr26 {
    static interface MyInterface {
        void Perimeter(int x);
    }
    static class Circle implements MyInterface {
        public void Perimeter(int x) {
            System.out.printf("Perimeter of the Circle is: %.2f\n",2*Math.PI*x);
        }
    }
    static class Square implements MyInterface {
        public void Perimeter(int x) {
            System.out.println("Perimeter of the Square is: "+ 4*x);
        }
    }
    public static void main(String[] args) {
        Circle c = new Circle();
        c.Perimeter(5);
        Square s = new Square();
        s.Perimeter(10);
        System.out.println("Vivek Mishra");

    }

}
```

Output:



```
Perimeter of the Circle is: 31.42
Perimeter of the Square is: 40
Vivek Mishra
```