Program 5.1

Aim: Write a java program using abstract class and abstract method

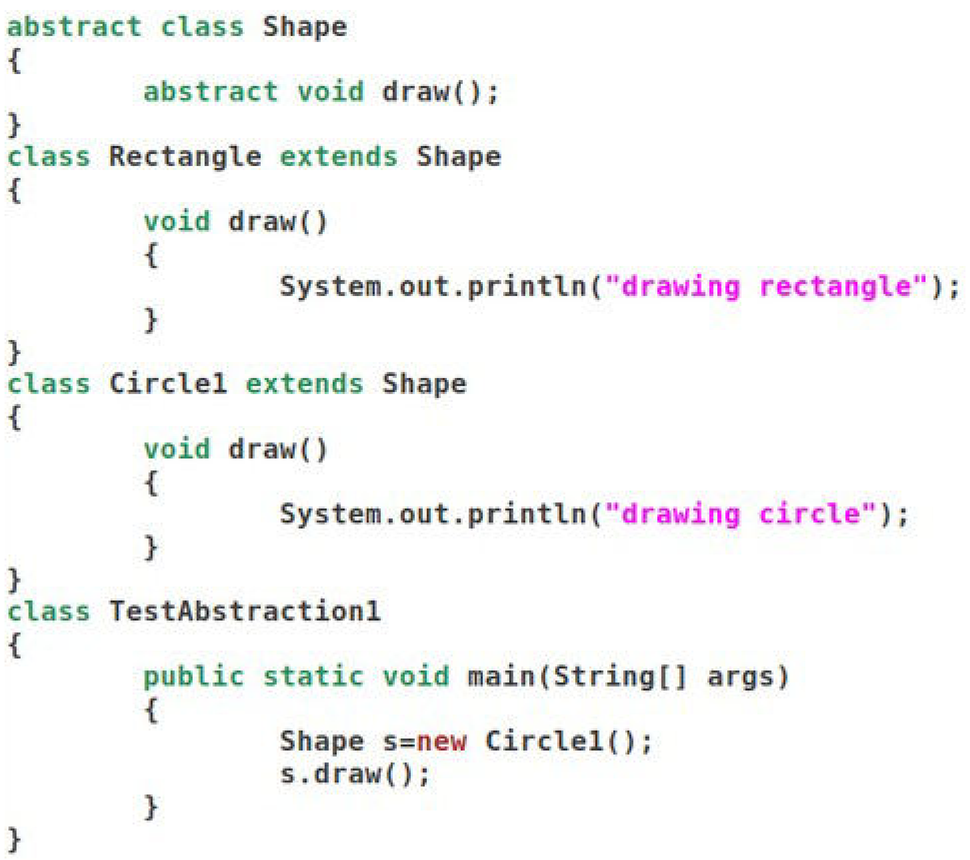
Description:

The `abstract` keyword is a non-access modifier, used for classes and methods:

- **abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

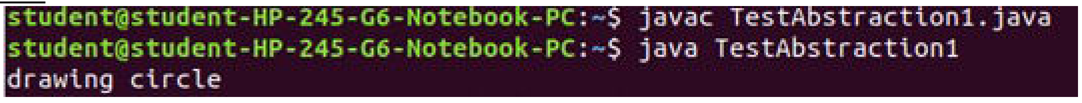
An abstract class can have both abstract and regular methods

Program:



```java
abstract class Shape
{
        abstract void draw();
}
class Rectangle extends Shape
{
        void draw()
        {
                System.out.println("drawing rectangle");
        }
}
class Circle1 extends Shape
{
        void draw()
        {
                System.out.println("drawing circle");
        }
}
class TestAbstraction1
{
        public static void main(String[] args)
        {
                Shape s=new Circle1();
                s.draw();
        }
}
```

Output:



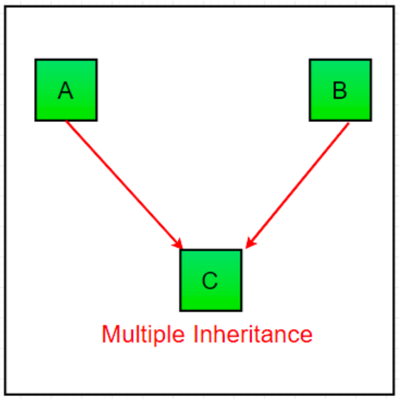
```
student@student-HP-245-G6-Notebook-PC:~$ javac TestAbstraction1.java
student@student-HP-245-G6-Notebook-PC:~$ java TestAbstraction1
drawing circle
```

Program 5.2.(i)

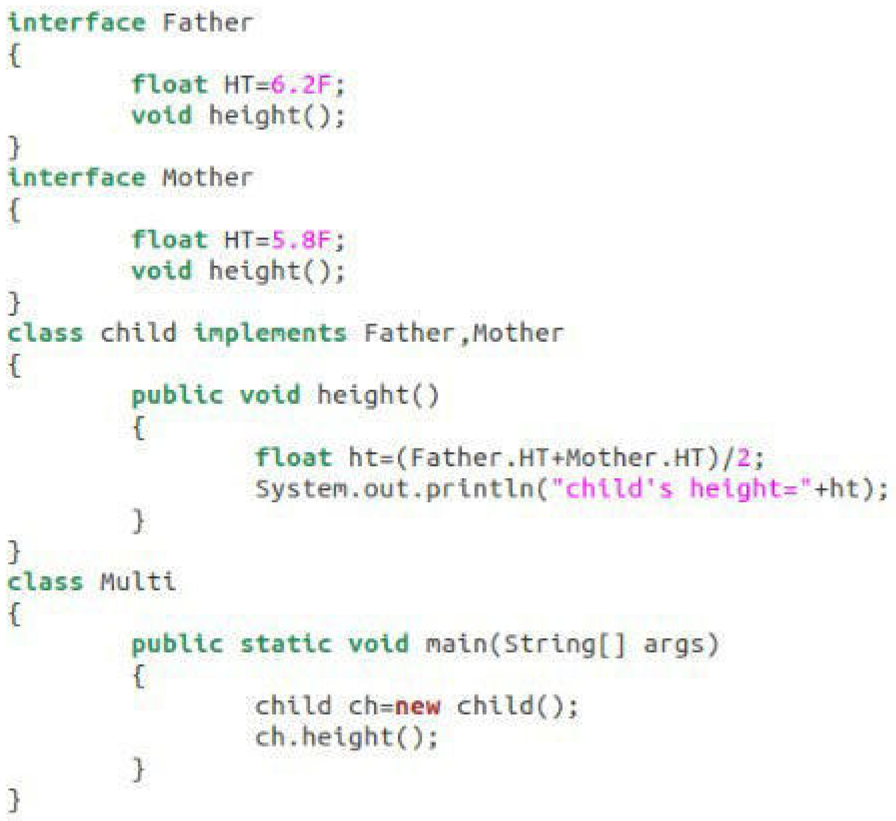
Aim: Write a java program to implement the concept of multiple inheritance using intefaces.

Description:

- In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes.
- Java does **not** support multiple inheritance with classes.
- In java, we can achieve multiple inheritances only through interfaces
- In the image below, Class C is derived from interface A and B.



Program:



```
interface Father
{
        float HT=6.2F;
        void height();
}
interface Mother
{
        float HT=5.8F;
        void height();
}
class child implements Father,Mother
{
        public void height()
        {
                float ht=(Father.HT+Mother.HT)/2;
                System.out.println("child's height="+ht);
        }
}
class Multi
{
        public static void main(String[] args)
        {
                child ch=new child();
                ch.height();
        }
}
```

Output:
child’s height=6.0

Program 5.2.(ii)

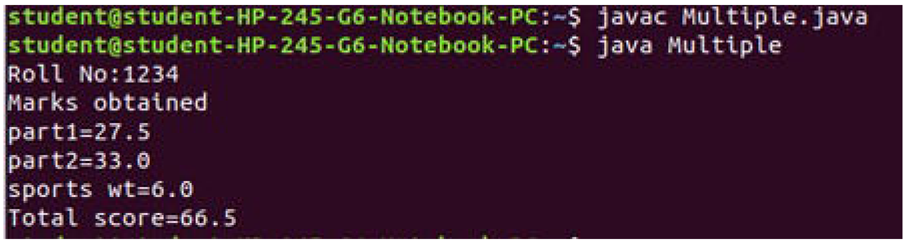
Aim: Write a program which illustrates the implementation of multiple inheritance using interfaces in java.

Program:

```java
class Student
{
        int rollNumber;
        void getNumber(int n)
        {
                rollNumber=n;
        }
        void putNumber()
        {
                System.out.println("Roll No:"+rollNumber);
        }
}
class Test extends Student
{
        float part1,part2;
        void getMarks(float m1,float m2)
        {
                part1=m1;
                part2=m2;
        }
        void putMarks()
        {
                System.out.println("Marks obtained");
                System.out.println("part1="+part1);
                System.out.println("part2="+part2);
        }
}

interface Sports
{
        float sportWt=6.0F;
        void putWt();
}
class Results extends Test implements Sports
{
        float total;
        public void putWt()
        {
                System.out.println("sports wt="+sportWt);
        }
        void display()
        {
                total=part1+part2+sportWt;
                putNumber();
                putMarks();
                putWt();
                System.out.println("Total score="+total);
        }
}
class Multiple
{
        public static void main(String[] args)
        {
                Results student1 = new Results();
                student1.getNumber(1234);
                student1.getMarks(27.5F,33.0F);
                student1.display();
        }
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ javac Multiple.java
student@student-HP-245-G6-Notebook-PC:~$ java Multiple
Roll No:1234
Marks obtained
part1=27.5
part2=33.0
sports wt=6.0
Total score=66.5
```

Program 5.3

AIM: Write a java program on implementing an interface.

Description:

- Interfaces used as “superclasses” whose properties are inherited by classes.
- It is therefore necessary to create a class that inherits the given interface.
- This is done as follows:

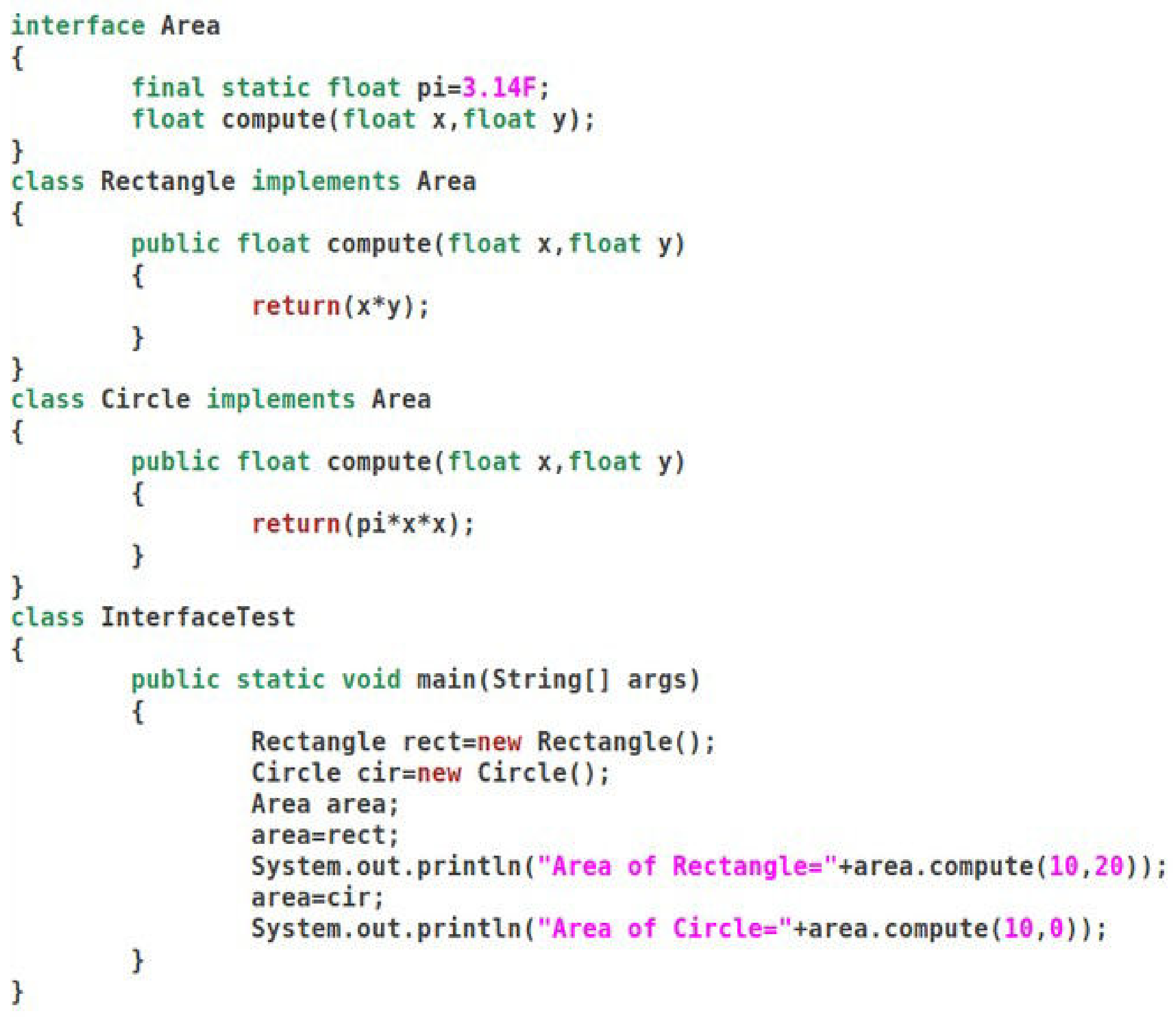
```
class classname implements interfacename
{
        body of classname
}
```

- Here the class classname “implements” the interface interfacename
- A more general form of implementation may look like this:

```
class classname extends superclass implements interface1,interface2
{
        body of classname
}
```

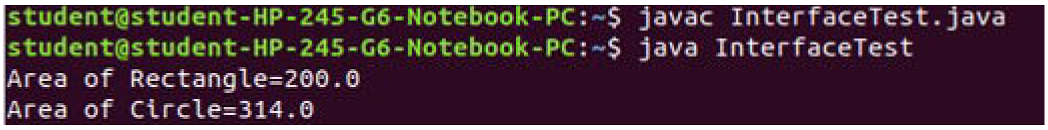
- This shows that a classname can extend another class while implementing interfaces.
- When a class implements more than one interface, they are seperated by a comma.

Program:



```java
interface Area
{
        final static float pi=3.14F;
        float compute(float x,float y);
}
class Rectangle implements Area
{
        public float compute(float x,float y)
        {
                return(x*y);
        }
}
class Circle implements Area
{
        public float compute(float x,float y)
        {
                return(pi*x*x);
        }
}
class InterfaceTest
{
        public static void main(String[] args)
        {
                Rectangle rect=new Rectangle();
                Circle cir=new Circle();
                Area area;
                area=rect;
                System.out.println("Area of Rectangle="+area.compute(10,20));
                area=cir;
                System.out.println("Area of Circle="+area.compute(10,0));
        }
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ javac InterfaceTest.java
student@student-HP-245-G6-Notebook-PC:~$ java InterfaceTest
Area of Rectangle=200.0
Area of Circle=314.0
```

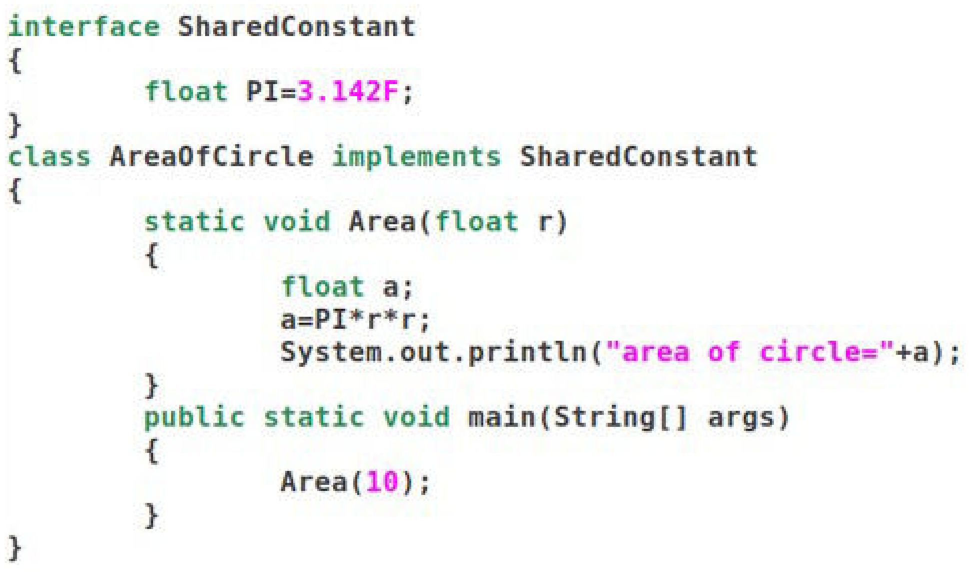
Program 5. 4

Aim: Write a java program using interface variable

Description:

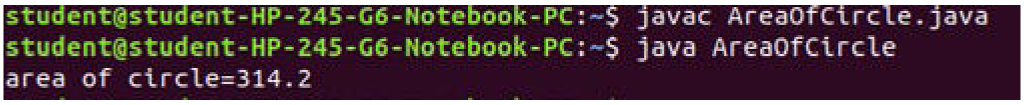
- We can use interfaces to import shared constants into multiple classes by simply declaring an interface that contains variables that are initialized to the desired values.
- The constant values will be available to any class that implements the interface.
- The values can be used in any method,as part of any variable declaration or anywhere where we can use a final value.

Program:



```
interface SharedConstant
{
        float PI=3.142F;
}
class AreaOfCircle implements SharedConstant
{
        static void Area(float r)
        {
                float a;
                a=PI*r*r;
                System.out.println("area of circle="+a);
        }
        public static void main(String[] args)
        {
                Area(10);
        }
}
```

Output:



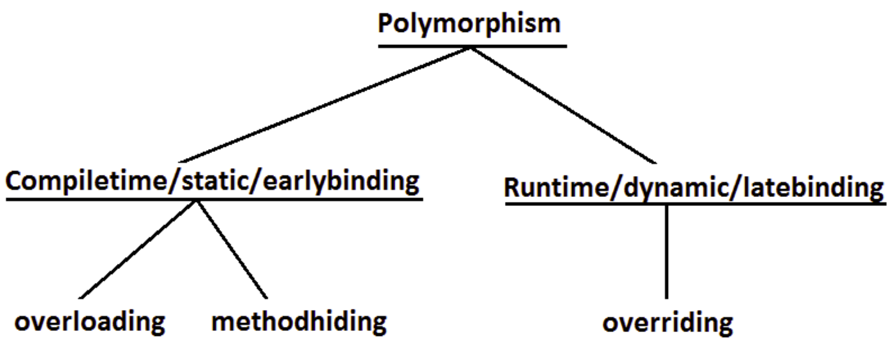
```
student@student-HP-245-G6-Notebook-PC:~$ javac AreaOfCircle.java
student@student-HP-245-G6-Notebook-PC:~$ java AreaOfCircle
area of circle=314.2
```

Program 5.5

Aim: Write a java program using method overriding

Description:

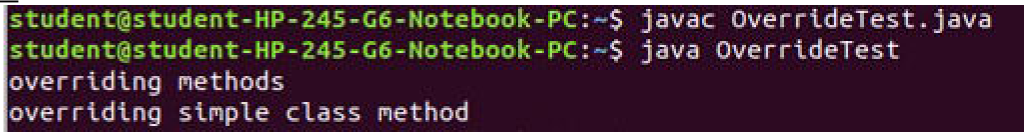
- Method overriding is one of the way by which java achieve run time polymorphism.
- The version of a method that is executed will be determined by the object that is used to invoke it.
- If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.
- In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed.



Program:

```java
class SimpleClass
{
        public void display()
        {
                System.out.println("overriding methods");
        }
}
class OverrideClass extends SimpleClass
{
        public void display()
        {
                super.display();
                System.out.println("overriding simple class method");
        }
}
class OverrideTest
{
        public static void main(String[] args)
        {
                OverrideClass obj=new OverrideClass();
                obj.display();
        }
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ javac OverrideTest.java
student@student-HP-245-G6-Notebook-PC:~$ java OverrideTest
overriding methods
overriding simple class method
```

PROGRAM 5.6

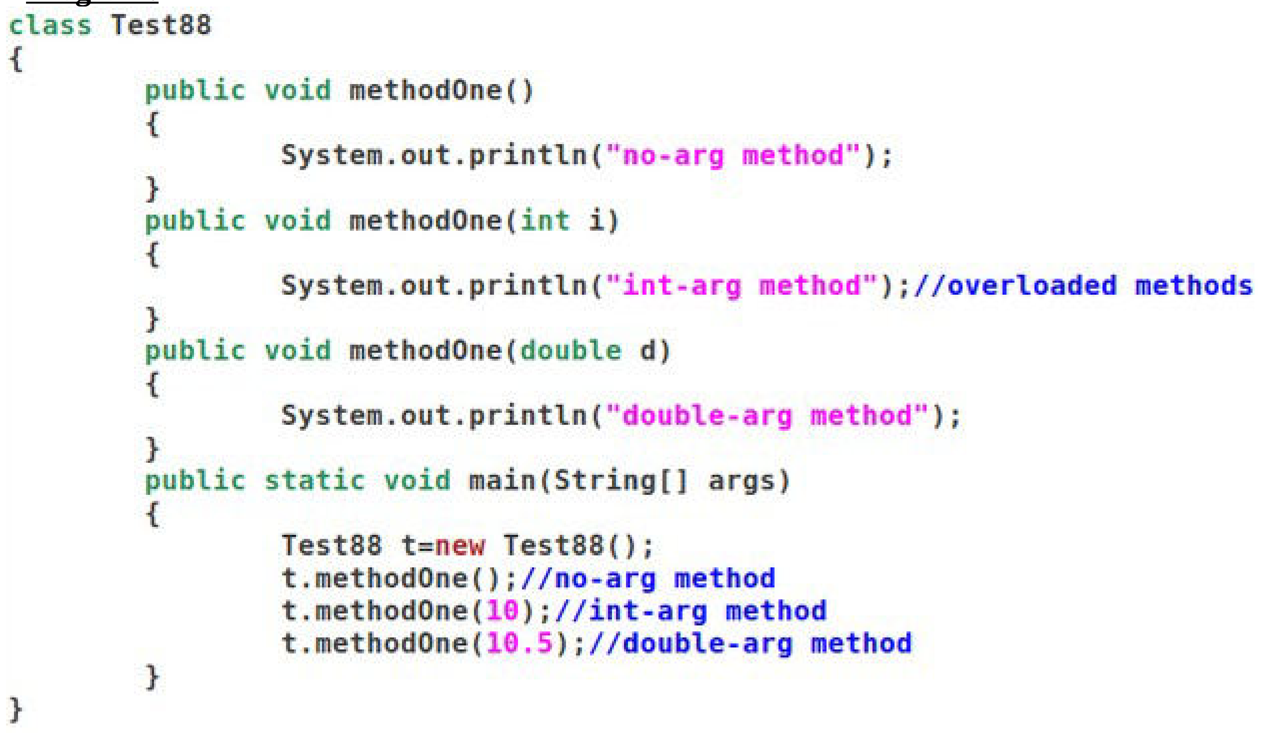
Aim: Write a java program using method overloading

Description:

Two methods are said to be overload, if and only if both having the same name but different argument types.

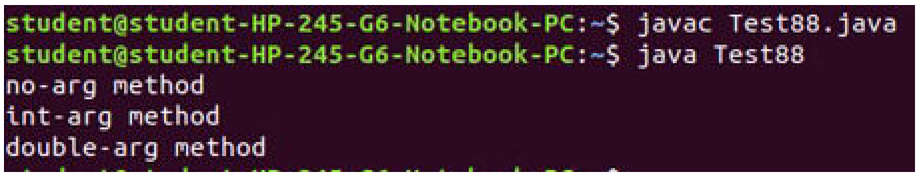
- Overloaded methods MUST change the argument list
- Overloaded methods CAN change the return type.
- Overloaded methods CAN change the access modifier.
- Overloaded methods CAN declare new or boarder checked exceptions.

Program:



```
class Test88
{
        public void methodOne()
        {
                System.out.println("no-arg method");
        }
        public void methodOne(int i)
        {
                System.out.println("int-arg method");//overloaded methods
        }
        public void methodOne(double d)
        {
                System.out.println("double-arg method");
        }
        public static void main(String[] args)
        {
                Test88 t=new Test88();
                t.methodOne();//no-arg method
                t.methodOne(10);//int-arg method
                t.methodOne(10.5);//double-arg method
        }
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ javac Test88.java
student@student-HP-245-G6-Notebook-PC:~$ java Test88
no-arg method
int-arg method
double-arg method
```

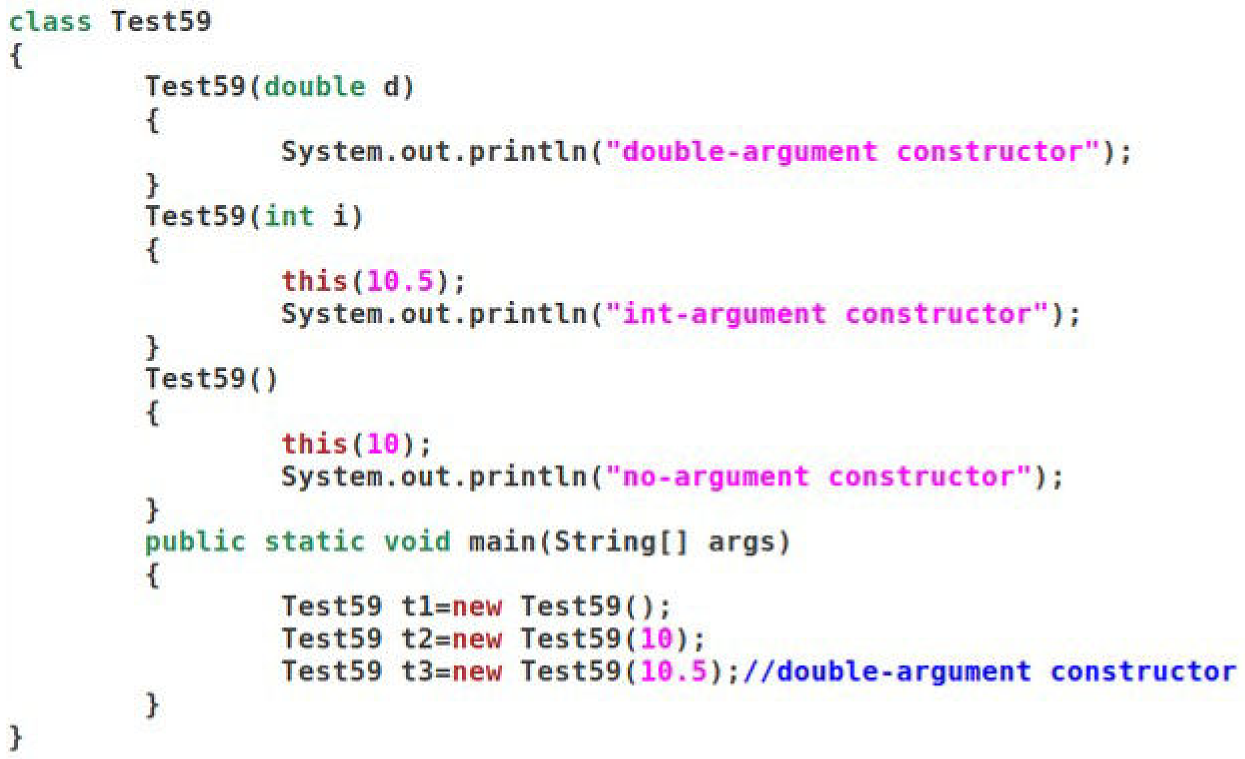
PROGRAM 5.7

Aim: Write a java program using constructor overloading.

Description:

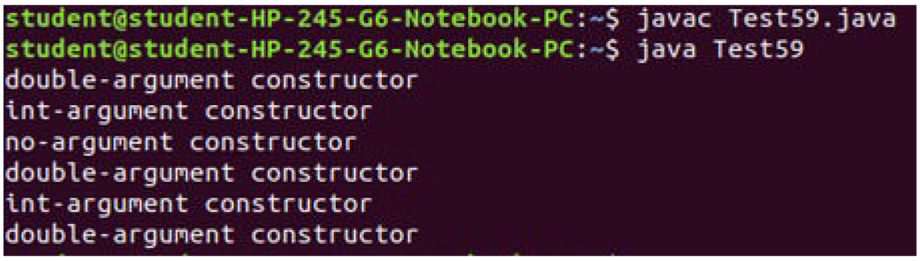
- A class can contain more than one constructor and all these constructors having the same name but different arguments and hence these constructors are considered as overloaded constructors.
- Parent class constructor by default won't available to the Child. Hence Inheritance concept is not applicable for constructors and hence overriding concept also not applicable to the constructors. But constructors can be overloaded.
- We can take constructor in any java class including abstract class also but we can't take constructor inside interface.

Program:



```java
class Test59
{
        Test59(double d)
        {
                System.out.println("double-argument constructor");
        }
        Test59(int i)
        {
                this(10.5);
                System.out.println("int-argument constructor");
        }
        Test59()
        {
                this(10);
                System.out.println("no-argument constructor");
        }
        public static void main(String[] args)
        {
                Test59 t1=new Test59();
                Test59 t2=new Test59(10);
                Test59 t3=new Test59(10.5);//double-argument constructor
        }
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ javac Test59.java
student@student-HP-245-G6-Notebook-PC:~$ java Test59
double-argument constructor
int-argument constructor
no-argument constructor
double-argument constructor
int-argument constructor
double-argument constructor
```