ABSTRACT CLASS AND FINAL KEYWORD

Abstract class in Java

- 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).
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Rules of abstract

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Example program

```
//abstract parent class
abstract class Animal{
   //abstract method
   public abstract void sound();
}
//Dog class extends Animal class
public class Dog extends Animal
{
  public void sound()
{
            System.out.println("Woof");
   }
   public static void main(String args[])
{
            Dog obj = new Dog();
            obj.sound();
   }
}
```

```
Output:
Woof
```

The real scenario of Abstract class

```
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");
}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g.,
getShape() method
s.draw();
}
}
Output:drawing circle
```

Abstract class having constructor, data member and methods

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike()
{
System.out.println("bike is created");
}
   abstract void run();
voidchangeGear()
{
System.out.println("gear changed");
}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run()
{
System.out.println("running safely..");
}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[])
{
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
Output:
       bike is created
       running safely..
       gear changed
```

FINAL

- The final keyword is final that is we cannot change.
- We can use final keywords for variables, methods, and class.
- If we use the final keyword for the inheritance that is if we declare any method with the final keyword in the base class so the implementation of the final method will be the same as in derived class.
- We can declare the final method in any subclass for which we want that if any other class extends this subclass

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

CODING:

```
class Bike

{

 final int speedlimit=90;  //final variable

 void run()

{

  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike obj=new  Bike();
 obj.run();
 }
}
```

OUTPUT

```
Output:Compile Time Error
```

2) Java final method

If you make any method as final, you cannot override it.

CODING:

```
class Bike
{
  final void run()
{
System.out.println("running");
}
}

class Honda extends Bike
{
   void run()
{
System.out.println("running safely with 100kmph");
}
    public static void main(String args[])
{
   Honda honda= new Honda();
   honda.run();
 }
}
```

OUTPUT

```
Output:Compile Time Error
```

3) Java final class

If you make any class as final, you cannot extend it

CODING:

```
final class Bike
{}
 class Honda1 extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
  public static void main(String args[])
{
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

OUTPUT

```
Output:Compile Time Error
```

<u>**INHERITANCE OF FINAL :**</u>

EXAMPLE CODING 1:

```
class Bike

{

  final void run()

{

System.out.println("running...");

}

}

class Honda2 extends Bike

{

   public static void main(String args[])

{

    new Honda2().run();

   }

}
```

OUTPUT

```
running...
```

EXAMPLE:

EXAMPLE CODING 2:

```
abstract class Shape

{

          private double width;

          private double height;

          public Shape(double width, double height)
```

```
        {
                this.width = width;
                this.height = height;
        }
                public final double getWidth()
        {
                return width;
        }
                public final double getHeight()
        {
                return height;
        }
        abstract double getArea();
}
class Rectangle extends Shape
{
                public Rectangle(double width, double height)
        {
                                super(width, height);
        }
        final double getArea()
        {
                return this.getHeight() * this.getWidth();
        }
}
class Square extends Shape
{
                public Square(double side)
        {
```

```
                                super(side, side);
        }
                final double getArea()
        {
                return this.getHeight() * this.getWidth();
        }
}
class Main
{
        public static void main(String[] args)
        {
                Shape s1 = new Rectangle(10, 20);
                Shape s2 = new Square(10);
                System.out.println("width of s1 : "+ s1.getWidth());
                System.out.println("height of s1 : "+ s1.getHeight());
                System.out.println("width of s2 : "+ s2.getWidth());
                System.out.println("height of s2 : "+ s2.getHeight());
                System.out.println("area of s1 : "+ s1.getArea());
                System.out.println("area of s2 : "+ s2.getArea());
        }
}
```

OUTPUT

```
width of s1 :10
height of s1 :20
width of s2 :10
height of s2:10
area of s1 :200
area of s2 :100
```