Abstract class in JAVA

Abstract class is a class in which contains at least one abstract function[does not contain function body]

Any class that contains one or more abstract methods must also be declared abstract.

To declare a class abstract use the abstract keyword before the class keyword at the beginning of the class declaration.

An abstract class is not fully defined so we can not access whole class functionalities because one or more than one functions may not contain function body.

- An abstract class cannot be directly instantiated with the new operator , so we can not create objects of an abstract class.

- Such objects would be useless, because an abstract class is not defined fully.

- We can not declare abstract constructors[or constructor must contain its body]

- We can not declare abstract static methods.

- Any subclass of an abstract class must either implement all of the abstract methods declared in the superclass, or subclass be itself declared abstract.

```
abstract class A
{
int a,b;
A(int i,int j)
{a=i; b=j;}
abstract void add();
void sub() // concrete methods are still allowed in abstract classes
{ System.out.println("This is a concrete method.“+(a-b));
 }
}
```

```
class B extends A {
B(int i,int j)
{super(i,j);}
void add()
{ System.out.println("B's implementation of add“+(a+b));
} }
class demo
{ public static void main(String a[])
{ B b=new B(34,56);
System.out.println(“This is the output from base class”);
 b.add();
```

```
 abstract class shape
{
double d1,d2;

shape(double a, double b) { d1 = a; d2 = b; }
 // area is now an abstract method

abstract double area();
}

class Rect extends shape
{ Rect(double a, double b) { super(a, b); //call to abstract class constructor}

 // override area for Rect
double area()
{ System.out.println("Inside Area for Rectangle.");
return (d1 * d2); }
 }
```

```
class Tri extends shape
{
 Tri(double a, double b)
{ super(a, b); //call to the abstract class constructor}

double area()
 { System.out.println("Inside Area for Triangle.");
return d1 * d2 / 2;
 } }
class Areas
{
 public static void main(String args[])
{
// shape s = new shape(23, 44); // illegal now  not allowed because shape is abstract

Rect r = new Rect(9, 5);
Tri t = new Tri(10, 8);
 shape s; // this is OK, no object is created
 s = r;
System.out.println("Area is " + s.area());
 s = t;
System.out.println("Area is " + s.area()); } }
```

- As the comment inside main( ) indicates,
- it is no longer possible to declare objects of type shape.
- since it is abstract. And, all subclasses of shape must override area( ).
- If we try creating a subclass that does not override area( ).
- A compile-time error is generated
- It is not possible to create an object of type shape
- Reference variable of type shape may be created.
- The variable ‘s’ is declared as a reference to shape,
- which means that it can be used to refer to an object of any class derived from shape.
- It is through superclass reference variables that overridden methods are resolved at run time. Using final with Inheritance