Abstract class

There is a situation in which you want to define a super class that declares the structure of a given abstraction without providing a complete implementation of every method. That is sometimes you want to create a super class or parent class that only defines a generalized method and that method you want to share by all of its sub-classes or child classes, leaving it to be each sub class to implement method. Abstract class determines the nature of the methods that the sub-classes must implement.

Abstract class means a class which is declared with the "**abstract**" keyword and abstract class may or may not include **abstract method**.

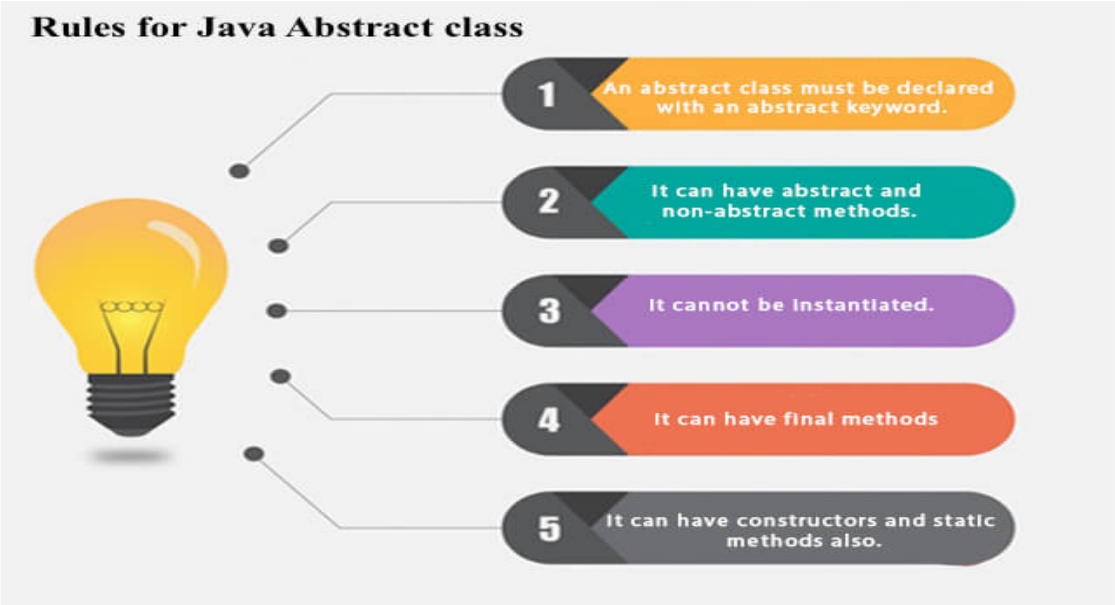
Abstract methods means a method which are only declared but not defined (means abstract method does not have body).

Abstract class needs to be extended and its method implemented. Means the abstract methods which are there in super class must be override in sub-class otherwise it will give you error message.

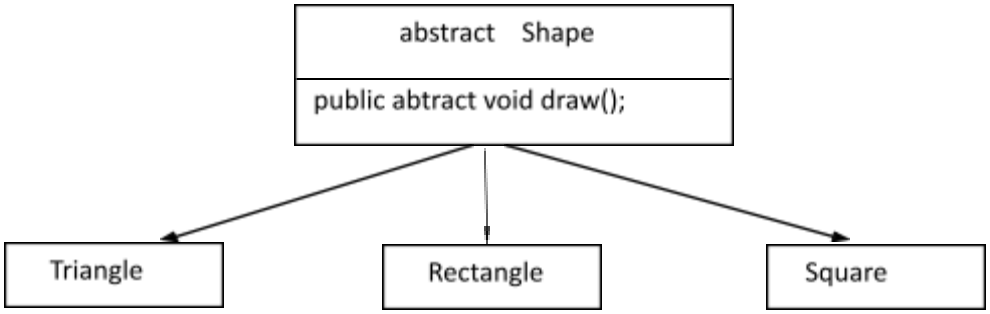
We can't create an instance of an abstract class because it is an incomplete class.

Points to Remember

- 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.



Eg:



In this example, Shape is the abstract class, and its implementation is provided by the Triangle Rectangle and Square classes.

Syntax to create abstract class:

<abstract> <class> <calss_name)
{
//body
}

Syntax to create abstract method:

<access-modifier> <abstract> <retruen-type> <method_name>()
{
 	
}

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle");}
}

class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Square extends Shape{
void draw(){System.out.println("drawing square");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Triangle t=new Triangle();
t.draw();
Rectangle r=new Rectangle();
r.draw();
}
}
```

Eg:

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor.

```
//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();
   void changeGear(){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 Test{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Output:

```
bike is created
running safely..
gear changed
```