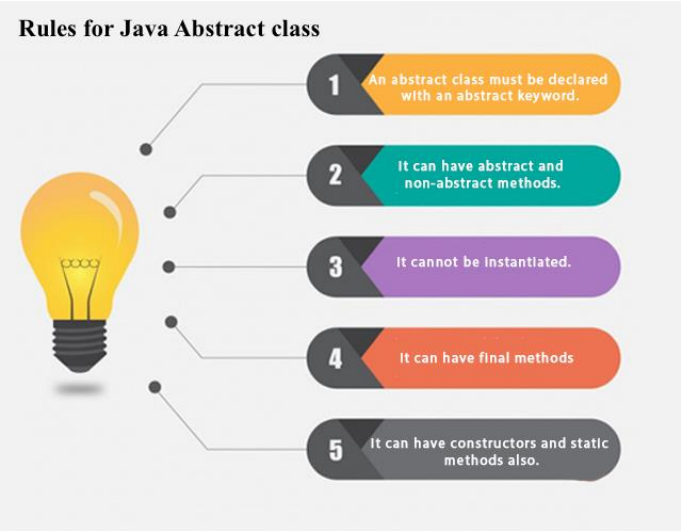
LEC17 OOP 2018-2019

Abstract Class

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Use the abstract keyword to declare a class abstract. The keyword appears in the class declaration somewhere before the class keyword. An abstract class is one that cannot be instantiated (غير مسموح لتعريف اي كائن ينتمي اليه). All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner. You just cannot create an instance of the abstract class. If a class is abstract and cannot be instantiated, the class does not have much use unless it is subclasses.



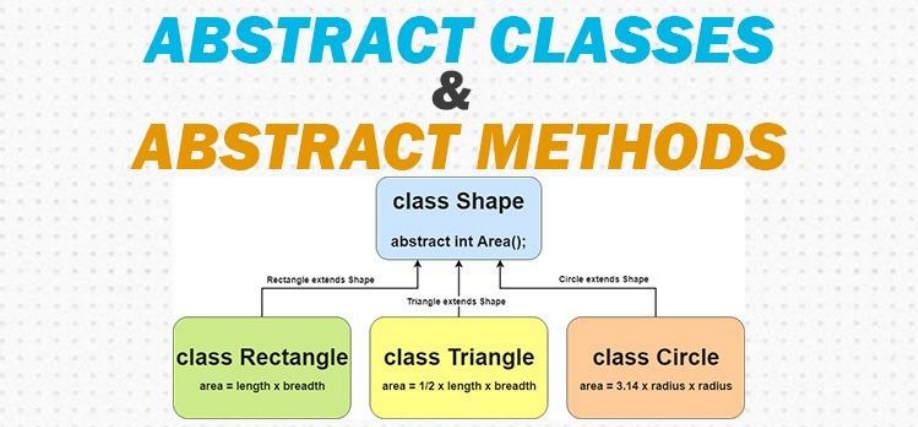
Example1

```
public abstract class First {
protected int x,y;
   public void setting(){x=100;y=200;} }

public class Second extends First {
   private int a,b;
   public void set(){ a=10;b=20;     System.out.println(a+ "    "+b);
                      setting();     System.out.println(x+"    "+y);    } }
public class Main {
   public static void main(String[] args) {
     //First f=new First(); error because it couldn't be instantiated
     Second s=new Second();
     s.set(e);    } }
```

LEC17 OOP 2018-2019

Abstract Method



- An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

abstract int area();
abstract int sum(int x,int y);
abstract int mul(int ,int);

- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, **if it does not, the subclass must also be declared `abstract`.**

- **If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.**

- The abstract keyword is also used to declare a method as abstract. Abstract methods consist of a method signature, but no method body.
- Abstract method would have no definition, and its signature is followed by a semicolon, not curly braces as follows:

```
public abstract class Employee  {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();

   //Remainder of class definition
}
```

Declaring a method as abstract has two results:

- The class must also be declared abstract. If a class contains an abstract method, the class must be abstract as well.
- Any child class must either override the abstract method or declare itself abstract.

A child class that inherits an abstract method must override it. If they do not, they must be abstract, and any of their children must override it. Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated. If Salary is extending Employee class then it is required to implement computePay() method as follows:

```
/* File name : Salary.java */
public class Salary extends Employee
{
   private double salary; //Annual salary

   public double computePay()
   {
      System.out.println("Computing salary pay for "
+ getName());
      return salary/52;
   }

   //Remainder of class definition
}
```

Example 2

```
Public abstract class Bike{
  abstract void run();   }
public class Honda4 extends Bike{
void run(){System.out.println("running safely..");}  }
public class Main{
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();  }  }
```

Conclusion points of abstract class:

1. Abstract class usually contains abstract methods.
2. Program can't instantiate an abstract class.
3. Abstract classes contain mixture of non-abstract and abstract methods.

Example 3

Define a base class called Polygon which has two integer attributes represent width and height of a Polygon. Set method is used for setting the width and the height. Derive three subclasses Rectangle, Triangle and Parallelogram. Write a main class to print the areas of the three Polygons using one polygon object. The output looks like:

```
public abstract class Polygon {
protected int d1,d2;

public  void set(int a,int b){
// protected  void set(int a,int b){

   d1=a;d2=b;}
public abstract int area();
public abstract void print(); }

public class Rectangle extends Polygon{
   public int area(){return (d1*d2);}
   public void print(){
      System.out.println("rectangle");
            System.out.println(d1+"     "+d2);     }
}
```

Protect correct too bcz they r in the same package **:Commented [l1]**

```
public class Triangle extends Polygon {
    public int area(){return (d1*d2/2);}
  public void print(){
       System.out.println("triangle");
             System.out.println(d1+"     "+d2);     }
}

public class Parallel extends Polygon {
public int area(){return (d1*d2);}
    public void print(){
       System.out.println("parallelogram");
             System.out.println(d1+"     "+d2);
   }
}

Public class Main{
Public static void main(String[] arq){
Polygon r=new Rectangle();
 r.set(4,5);  System.out.println(r.area());  r.print();

Polygon t=new Traingle();
 t.set(7,8); System.out.println(t.area()); t.print();

Polygon p=new Parallel();
p.set(5,6); System.out.println(p.area()); p.print();
```

20
rectangle
4 5

28
triangle
7 8

30
paralleloram
5 6

LEC17 OOP 2018-2019

Re write the previous main method using array to store the three polygons.

```
public class Main{
public static void main(String[] args) {
 Polygon [] a=new Polygon[3];
  for (int i=0;i<3;i++)
    switch (i) {
       case 0: a[i]=new Rectangle();a[i].set(5,4);break;
       case 1: a[i]=new Triangle();a[i].set(7,8);break;
       case 2: a[i]=new Parallel();a[i].set(5,6);break;
    }

  for(int i=0;i<3;i++){
     System.out.println(a[i].area());
     a[i].print();}
  }
}
```

The output is:

```
20
rectangle
5     4

28
triangle
7     8

30
paralleloram
5     6
```

More cases will be explained through the lecture