ABSTRACT IN JAVA

by

DR. AMAL S. AJRASH

INTRODUCTION

Abstraction: is a process of hiding the implementation details and showing only functionality to the user. It shows only essential things to the user and hides the
internal details.

ABSTRACT CLASS & METHOD

• An abstract class is a class that is declared ”abstract” it may or may not include abstract methods.

• A method which is declared as “abstract” and does not have implementation is known as an abstract method.

• Syntax :

```
modifier abstract class className
{
    abstract dataType methodName();
}
modifier class childClass extends className
{
dataType methodName(){}
}
```

Abstract classes

Any class containing an abstract method is an abstract class, you must declare the class with the keyword abstract:

```
abstract class MyClass {...}
```

- An abstract class is *incomplete*
- It has “missing” method bodies

- If a class is declared abstract, it cannot be instantiated
- You can extend (subclass) an abstract class
- If the subclass defines all the inherited abstract methods, it is “complete” and can be instantiated
- If the subclass does *not* define all the inherited abstract methods, it too must be abstract
- You can declare a class to be abstract even if it does not contain any abstract methods This prevents the class from being instantiated.
- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();). But, if a class has at least one abstract method, then the class **must** be declared abstract.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it. To share code among several closely related classes.
- If classes that extend your abstract class have many common methods or fields or require access modifiers other than public (such as protected and private). You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

Why have abstract classes?

Suppose you wanted to create a class Shape, with subclasses Oval, Rectangle, Triangle, Hexagon, etc. You don’t want to allow creation of a “Shape” Only *particular* shapes make sense, not *generic* ones, If Shape is abstract, you can’t create a new Shape. You *can* create a new Oval, a new Rectangle, etc. Abstract classes are good for defining a general.

```
}
```

Example1:

```
abstract class Animal //Abstract Class Declaration
{
public abstract void sound(); //Abstract Method Declaration
}
public class Dog extends Animal //Dog inherits from Animal
{
public void sound()
{
System.out.println("Woof");
}
public static void main(String args[])
{
Animal obj = new Dog();
obj.sound();
}
}
```

OUTPUT : Woof

Example2:

```
abstract class MyClass
{
public void disp()
{
System.out.println("Concrete method of parent class");
}
abstract public void disp2();
```

```
}
class Demo extends MyClass
{
public void disp2()
{
System.out.println("overriding abstract method");
}
public class Main {
public static void main(String args[])
{
Demo obj = new Demo();
obj.disp2();
}
}
```

OUTPUT : overriding abstract method

Example3 of Abstract Class

```
public abstract class Employee
{
private String name;
private String address;
public Employee(String name, String address, int number)
{
System.out.println("Constructing an Employee");
this.name = name;
}
public double computePay()
{
System.out.println("Inside Employee computePay");
return 0.0;
}
}
```

Now lets try to instantiate the Employee class in the following way

```
public class Abstract Demo
{
public static void main(String [] args)
{
Employee e = new Employee("George W.“);
System.out.println("\n Call Employee ");
e.mailCheck();
}
}
```

Output:

```
Employee.java:46: Employee is abstract; cannot be instantiated
Employee e = new Employee("George W”);
^ 1 error
```

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 an abstract. **abstract** keyword is used to declare the method as abstract. You have to place the **abstract** keyword before the method name in the method declaration. Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

Example4 of the abstract method.

```
public abstract class Employee
{
private String name;
public abstract double computePay();
}
```

Declaring a method as abstract has two consequences –

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.
- Suppose Salary class inherits the Employee class, then it should implement computePay()

```
public class Salary extends Employee
{
private double salary;
double computePay()
{
System.out.println("Computing salary " + getName());
return salary/52;
}
}
```

Example5:

```
public abstract class A {
   int x;
   public abstract void print1();
}
public abstract class B extends A {
   public abstract void print2();
   // لا تنسى أن الكلاس B أيضاً ورث الدالة print1()
}
public class C extends B {
   // الكلاس C يجب أن يفعل Override لجميع الدوال التي نوعها abstract التي ورثها من الكلاس B
   @Override
   public void print1() {
      System.out.println("Class C should override the method print1()");
   }
   @Override
   public void print2() {
      System.out.println("Class C should override the method print2()");
   }
}
```

```
public class Main {

    public static void main(String[] args) {

        C c = new C();     // هنا قمنا بإنشاء كائن من الكلاس C

        c.print1();       // هنا قمنا باستدعاء الدالة print1() التي ورثها الكلاس C و فعل لها Override

        c.print2();       // هنا قمنا باستدعاء الدالة print2() التي ورثها الكلاس C و فعل لها Override

    }

}
```

Output:

```
Class C should override the method print1()
Class C should override the method print2()
```

Example6:

```
public abstract class A {     // الكلاس A نوعه abstract, إذاً لا يمكن إنشاء كائنات منه

    int x;

    public void print() {

        System.out.println("This is just an example.");

    }

}

public class Main {

    public static void main(String[] args) {

        A  a  =  new  A();        //  Incompatible  Type:  abstraction.A  is  abstract;  cannot  be
instantiated <-- [ سيظهر لك تحذير ]
```

```
    }
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - abstraction.A is abstract; cannot be instantiated
```

Example7:

```
public abstract class A {
    int x;
    public void print() {
        System.out.println("This is just an example.");
    }
}
public class B extends A {     // هنا قلنا أن الكلاس B يرث من الكلاس A. إذاً سيرث كل شيء موجود فيه
}
public class Main {
    public static void main(String[] args) {
        B b = new B();     // هنا قمنا بإنشاء كائن من الكلاس B
        b.print();         // هنا قمنا باستدعاء الدالة print() التي ورثها الكلاس B من الكلاس A
        b.x = 10;          // هنا قمنا بتغيير قيمة المتغير x الذي ورثه الكلاس B من الكلاس A
        System.out.println("b.x contain: " + b.x);      // هنا قمنا بعرض قيمة المتغير x
    }
}
```

Output:

```
This is just an example.
b.x contain: 10
```

Example8:

```
public abstract class A {
    int x;
    public void print() {
        System.out.println("This is just an example.");
    }
    public abstract void setX(int x);     // هنا قمنا بتعريف دالة نوعها abstract --> إذاً يجب أن يفعل لها Override كل كلاس يرثها
    public abstract int getX();           // هنا قمنا بتعريف دالة نوعها abstract --> إذاً يجب أن يفعل لها Override كل كلاس يرثها
}
public class B extends A {     // بما أن الكلاس B يرث من A يجب أن يفعل Override لأي دالة سيرثها من النوع abstract
    // هنا فعلنا Override للدالة setX()
    @Override
    public void setX(int x) {
        super.x = x;
    }
    // هنا فعلنا Override للدالة getX()
    @Override
    public int getX() {
        return super.x;
    }
}
public class Main {
    public static void main(String[] args) {
        B b = new B();     // هنا قمنا بإنشاء كائن من الكلاس B
        b.print();         // هنا قمنا باستدعاء الدالة print() التي ورثها الكلاس B من الكلاس A
        b.setX(55);   // هنا قمنا بتغيير قيمة المتغير x الذي ورثه الكلاس B من الكلاس A عن طريق الدالة setX()
        System.out.println("b.x contain: " + b.getX()); // هنا قمنا بعرض  x عن طريق الدالة getX()
قيمة المتغير
    }
}
```

Output:

```
This is just an example.
b.x contain: 55
```