ABSTRACT CLASSES & METHODS

What does abstract mean?
Abstract in English means existing in thought or as an idea without concrete existence.

ABSTRACT CLASS

→ An abstract class cannot be instantiated
→ Java requires us to extend it if we want to access it.
→ It can include abstract & non-abstract methods.
→ If a class includes abstract methods, then the class itself must be declared abstract

E.g.:
```
public abstract class iPhone Model {
    abstract void switch off ();
        // code
    }
```

→ Abstract class are used when we want to achieve security & abstraction (hide certain details & only show necessary details)
→ When a abstract class is subclassed, ~~for e.g~~ the subclass usually provides implementations for all of the methods in parent class. If it doesn't it must be declared abstract.

Note: It is possible to create refernce of an abstract class.
→ It is not possible to create object of abstract class

```
abstract class Phone {
    abstract void on();
}

class SmartPhone extends Phone {
    void on() {
        sout("Turning on ---");
    }

p s v m ( ) {
    Phone obj = new Smartphone();
    obj.on();
    }
}
```

o/p
Turning on ---

ABSTRACT METHOD

→ A method declared without implementation is called a abstract method.
→ An abstract method can only be used inside an abstract class.
→ The body of the abstract method is provided by the class that inherits the abstract class in which the abstract method is present

Note: We can also assign reference of an abstract class to the object of a concrete subclass.

```
package com.company

abstract class Parent2{
    public Parent2(){
        System.out.println("Main Main2 ka constructor hoon");
    }

    public void sayHello(){
        System.out("Hello");
    }

    abstract public void greet();
    --||--  --||--  greet2();

}

class Child2 extends Parent2{
    @Override
    public void greet(){
        System.out.println("Good morning");
    }

    @Override
    public void greet2(){
        System.out("Good afternoon");
    }
}

abstract class Child3 extends Parent2
    public void th(){
        System.out("I am good");
    }

}
```

```
public class cwh_53_abstract{

    p s v m ( ){
    // Parent2 p = new Parent2(); ... error
    Child2 c = new Child2();
    // Child3 c3=new Child3(); ... error.

    }
}
```

o/p

Mai ~~Base2~~ .

Mai base2 ka constructor hoon

INTERFACES

→ Just like class in Java is a collection of the related methods, an interface in Java is a collection of abstract methods (with empty bodies)

→ The interface is one more way to achieve abstraction in Java.

→ An interface may also contain constants, default methods, & static methods.

→ All the methods inside an interface must have empty bodies except default methods & static methods.

→ We use interface keyword to declare an interface

→ There is no need to write abstract keyword before declaring methods in an interface because an interface is implicitly abstract.

→ An interface cannot contain a constructor (as it cannot be used to create objects)

→ In order to implement an interface, Java requires a class to use the implement keyword.

In english interface is a point where 2 systems meet & interact

E.g.

```
interface Bicycle {
    void apply_brake(int decrement);
    void speed_up(int increment);
}

class dven_cycle implements Bicycle {
    int speed = 7;
    void apply_brake(int decrement) {
        speed = speed - decrement;
    }
    void speedup(int increment) {
        speed = speed + increment;
    }
```

```
package com.company

interface Bicycle{
    int a=45;
void applyBrake(int decrement);
void speedup(int increment);
}

interface HornBicycle{
    int x=45;
 void blowHorn3g();
 void blowHornmhn();
}

class Avoncycle implements Bicycle,
                           HornBicycle{
    // public int x=5;

void blowHorn(){
    sout("Pee Pee Poo Poo");
}

public void applyBrake(int decrement){
    sout("Applying Brake");
}
public void speedUp(int increment){
    sout("applying SpeedUp");
}
public void BlowHorn3g(){
    sout("Kabhi khushi kabhie gum");
}
```

```
public void blowHornmhn (){
    sout("Main hoon na po po po");
}

public class Interfaces {
    p s v m ( ) {
    AvonCycle cycleHarry = new Avoncycle();
cycleHarry.applyBrake(1);
// You can create properties in Interfaces

soutln(cycleHarry.a);
soutln(cycleHarry.x);

// you cannot modify the properties in interfaces
as they are final
// cycleHarry.a=454;
// sout

cycleHarry.blowHorn123();
cycleHarry.blowHornmhn();
}
}
```

o/p :

Applying Brakes

45

45

Kabhie khushi kabhie gum

Main hoon naa

Abstract Classes vs. Interfaces

Abstract class	Interface
→ It can contain abstract & non-abstract methods	→ It can only contain abstract methods. We do not need to use the "abstract" keyword in interface methods because the interface is implicitly abstract
→ Abstract keyword is used to declare an abstract class.	→ Interface keyword is used to declare an interface
→ A subclass extends the abstract class by using "extends" keyword.	→ The implements keyword is used to implement an interface.
→ A abstract class in Java can have members like private, protected etc.	→ Members of Java interface are public by default
→ Abstract class doesn't support multiple inheritance	→ Multiple inheritance is achieved in Java by using interface

Interfaces are meant for dynamic method dispatch & run-time polymorphism

Why multiple inheritance is not supported in Java?

→ Multiple inheritance faces problems when there exists a method with the same signature in both ~~classes~~ the superclasses.

→ Due to such a problem, Java doesn't support multiple ~~the~~ inheritance directly, but the similar concept can be achieved using interfaces.

→ A class can implement multiple interfaces & extend a class at the same time.

Note:→ Interface in Java is a bit like class but with a significant difference.
→ An interface can only have method signatures, constant fields & default methods.
→ The class implementing an Interface needs to declare the methods (not fields)
→ You can create a reference of Interfaces but not the object.
→ Interface methods are public by default.

Default methods in Java:

→ An interface can have default & static methods

→ Default methods enable us to add new functionality to existing interfaces.

→ This feature was introduced in Java 8 to ensure backward compatibility while updating a interface.

→ A class implementing the interface need not implement default methods.

→ Interfaces can also include private methods

→ You can easily override a default method like any other method of a interface.

→ Remember that interface cannot implement another interface, only classes do that.

E.g.

```
Interface Animal {
// Default method

default void say() {
    Sout("Hello, this is default method");
}

// Abstract method
  void bark();
}

public class CWH implements Animal {
@Override
public void bark() {
    Sout("Dog ~~bark~~ barks!");
}
```

```
p s v m(String[] args) {
    CWH obj1 = new CWH();
    obj1.bark();
    obj1.say();
    }
}
```

o/p

~~Dog to~~ Dog barks!
Hello, this is default method

For e.g.

```
package com.company

interface MyCamera{
  void takeSnap();
  void recordVideo();
  private void greet(){
        sout("Good Morning");
        }
  default void record4KVideo(){
      greet();
      sout("Recording in 4K....");
      }
}

interface MyWifi{
    String[] getNetworks();
  void connectToNetwork(String network);
}
```

```
class MyCellphone {
    void callNumber(int phoneNumber){
        sout("Calling " + phoneNumber);
    }

    void pickCall(){
        sout("Connecting ---");
    }
}

class MySmartphone extends MyCellphone
implements MyWifi, MyCamera {
    public void takeSnap(){
        sout("Taking snap");
    }

    public void recordVideo(){
        sout("Taking snap");
    }

    public String[] getNetworks(){
        Soutln("Getting list of Att Networks");
        String[] networkList={"Harry", "Prashanth",
                              "AngaliSG"};
        return networkList;
    }

    public void connectToNetwork(String network){
        Soutln("Connecting to" + network);
    }
}
```

cam1.getNetworks(); // → Not allowed

cam1.sampleMeth(); --> Not allowed

Date
Page

```
public class cntt{
    p s v m ( ){
MySmartPhone ms = new MySmartPhone();
ms.record4KVideo();
// ms.greet(); → shows error
String[] as = ms.getNetworks();
MyCamera2 cam1 = new MySmartphone2();
for(String item: as){
        sout(item);
    }
}
}
```

Allowed, This is a smartphone but use as a camera

```
c/p
Good Morning
Recording in 4K.....
Getting List of Networks
Harry
Prashanth
AngaliSG
```

Inheritance in Interfaces

Interfaces can extend other interfaces as shown below.

```
public interface Interface1{
    void meth1();
}
```

```
public interface Interface2 extends
                    Interface1{
    void meth2();
}
```

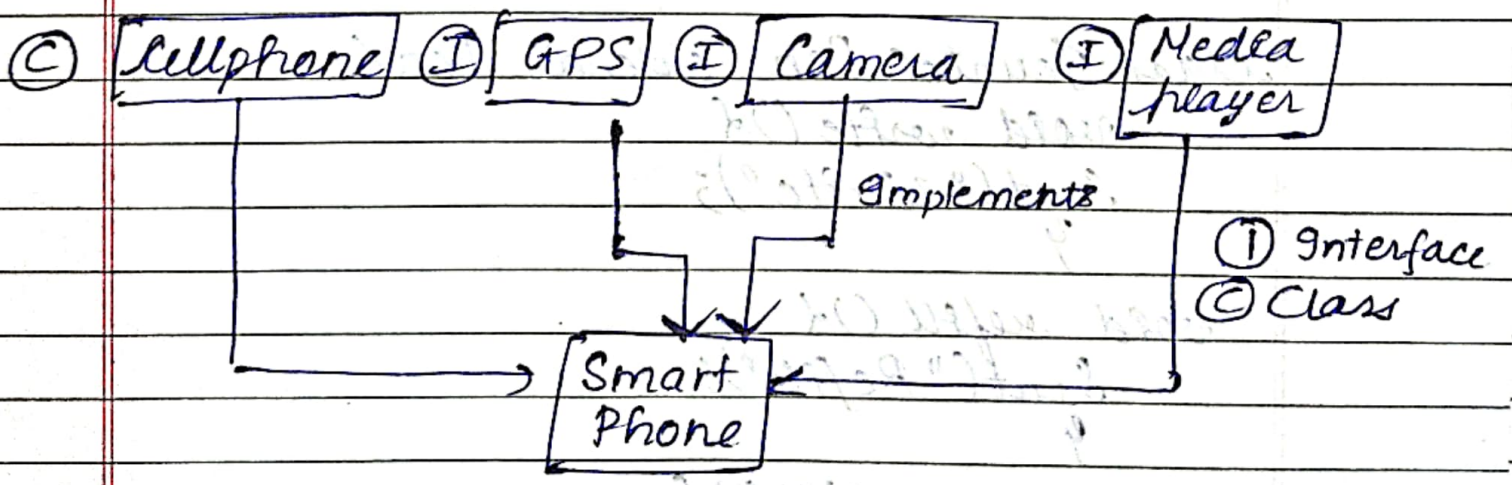
Note: Remember that interfaces cannot implement other interface only classes do that

For eg.

```
interface sample Interface{
    void meth1();
    void meth2();
}

interface childsample Interface extends
            sample Interface{
    void meth3();
    void meth4();
}

class Mysampleclass implements
            childsample Interface{
    public void meth1(){
        sout("meth1");
    }
    public void meth2(){
        sout("meth2");
    }
    public void meth3(){
        sout("meth3");
    }
```

```
public void meth4(){
    cout("meth 4");
}
}

public class CWH{

    p s v m ( ... ) {
MySampleClass obj = new MySampleClass();
    obj.meth1();
    obj.meth2();
    obj.meth3();
}
}
o/p
meth1
meth2
meth3
```

Polymorphism in Interfaces



GPS g = new Smartphone(); Can only use GPS methods

Smartphone s = new Smartphone(); can only use Smartphone methods

Practice set

1) Create a abstract class pen with methods write() & refill() as abstract methods, create fountain pen with method changeNib()

2) Create a class monkey with jump() & bite methods. Create a class human which inherits the monkey class & implements basic animal interface with eat() & sleep methods

```
package com.company

abstract class Pen {
    abstract void write();
    abstract void refill();
}

class FountainPen extends Pen {
    void write() {
        Sout("Write");
    }

    void refill() {
        Sout("Refill");
    }

    void changeNib() {
        Sout("Changing Nib");
    }
}
```

```
class Monkey {

    void jump() {
        Sout("Jumping....");
    }

    void bite() {
        Sout("Biting----");
    }

}

Interface BasicAnimal {
    void eat();
    void sleep();
}

class Human extends Monkey ~~extends~~ implements
                                    BasicAnimal {
    void speak() {
        Sout("Hello sir");
    }

    @Override
    public void eat() {
        Sout("Eating");
    }

    @Override
    public void sleep() {
        Sout("Sleeping");
    }
}
```

```
public class ABS {

    p s v m ( ) {

        FountainPen pen = new FountainPen();
        pen.changeNibe();

        Human harry = new Human();
        harry.sleep();

        Monkey m1 = new Human();
        m1.jump();
        m1.bite();
        // m1.speak(); Cannot use speak
        method because the reference of monkey
which does not have speak method

        BaseAnimal lovish = new Human();
        // lovish.speak(); → error
        lovish.eat();
        lovish.sleep();

    }
}
```

o/p

```
Changing the nibe
Sleeping
Jumping---
Biting-----
Eating
Sleeping
```