Structured Programming

Lecture8

Dr. Obead Alhadreti

Outline

- **Abstraction**
- **Interfaces**

Abstraction

Abstraction

- Abstraction is the concept of hiding the internal details of a functionality and providing a simple representation for the same. So complex functionality can be made available to the outside world in a simple way.

- For example: We use mobile phone everyday but we don't know how the functionalities are designed inside it so that we receive the calls and send messages etc. These functionalities have been kept inside and we are just accessing them using the options provided in the mobile.

Abstraction

- Similarly, In java we can write a method to perform some functionality inside a class and we can expose it to outside world just by providing an option to call this method.

- Anyone who calls this method will **not be knowing the internal complexity** of the method but will be **knowing the functionality** of the method and hence he calls it and uses it.

- In this way, we **hide the internal implementation** and abstract it inside a method.

Abstraction

- We can achieve abstraction in Java using 2 ways:
  1) Abstract class (0 to 100%)

  2) Interface (100%)

1) Abstract class

- **Abstract class** in Java can be created using "**abstract**" keyword.

- If we make any **class as abstract** then it can not be instantiated which means we are **not able to create the object** of abstract class.

- **<u>Syntax</u> :**

```
abstract class class_name { }
```

Abstract Methods

- Inside **Abstract class**, we can declare **abstract methods** as well as **concrete methods (**non-abstract methods**).**
  A **concrete method** means, the **method** have complete definition (method with body), but it can be overridden in the inherited class.

- **An abstract method** is a **method** that is declared, but contains no implementation. The method body will be defined by its subclass. Abstract method can never be **final and static**. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

Abstract Methods

- **Syntax :**

  abstract return_type *function_name* ();

- **Example: Phone.java**

```
abstract class Phone {
abstract void receiveCall();
abstract void sendMessage();
}
```

- Now any **concrete class** which **extends** the above **abstract class** will provide the **definition of these abstract methods (overriding)**.

Example 1

```
1   class Samsung extends Phone{
2   public void receiveCall(){
3   System.out.println("Call received in Samsung");
4   }
5
6   public void sendMessage(){
7   System.out.println("Message sent in Samsung");
8   }
9
10  }
```

- **Samsung class** has provided the **concrete definition for abstract methods** declared inside **Phone** abstract class.
- Anyone who needs to **access this functionality** has to call the method using the subclass objects.

Example 1

```
public class AbstractTest{
public static void main(String[] args) {
Samsung s = new Samsung ();
s.receiveCall();
s.sendMessage();
}
}
```

Output

Call received in Samsung
Message sent in Samsung

Abstract Classes and Methods

- **When to use Abstract Methods & Abstract Class?**
- Abstract methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods.

- Abstract classes are used to define generic types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

Abstract Classes and Methods

- **Points to Remember:**
  1. An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
  2. Abstract classes can have Constructors, Member variables and Normal methods.
  3. Abstract classes are never instantiated.
  4. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

Example2

Test.java | Vehicle.java | TwoWheeler.java

Source | History

```java
package vehiclemanager;

abstract public class Vehicle {
String regNo;

void vehicle(){
System.out.println("Creating Vehicle");
}

public abstract void start();
public abstract void stop();

public void display(){
System.out.println("Vehicle reg no : "+regNo);
}
}
```

Test.java | Vehicle.java | TwoWheeler.java

Source | History

```java
package vehiclemanager;

public class TwoWheeler extends Vehicle{
    @Override
        public void start() {
            System.out.println("Two wheeler start logic");
        }

    @Override
        public void stop() {
        System.out.println("Two Wheeler stop logic");
        }
}
```

Example2

Test.java | Vehicle.java | TwoWheeler.java

Source | History

```java
package vehiclemanager;

public class Test {

    public static void main(String[] args) {
        TwoWheeler tw = new TwoWheeler();

        tw.regNo="HR01 Q2125";

        tw.start();
        tw.stop();
        tw.display();

    }

}
```

Output - VehicleManager (run)

```
run:
Two wheeler start logic
Two Wheeler stop logic
Vehicle reg no : HR01 Q2125
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example3

Test.java | Banks.java | SBI.java | PNB.java

Source | History

```
1   package bank;
2
3   abstract class Bank{
4
5       abstract int getRateOfInterest();
6   }
7
```

Test.java | Banks.java | SBI.java | PNB.java

Source | History

```
1   package bank;
2
3   public class SBI  extends Bank{
4
5       @Override
7       int getRateOfInterest(){return 7;}
7   }
8
```

Test.java | Banks.java | SBI.java | PNB.java

Source | History

```
1   package bank;
2
3   public class PNB  extends Bank{
4
5       @Override
7       int getRateOfInterest(){return 8;}
7   }
```

Example3

Test.java | Banks.java | SBI.java | PNB.java

Source | History

```java
package bank;

public class Test {

    public static void main(String[] args) {


        SBI s =new SBI();
        System.out.println("Rate of Interest is: "+s.getRateOfInterest()+" %");

      PNB p  =new PNB();
        System.out.println("Rate of Interest is: "+p.getRateOfInterest()+" %");
    }

}
```

Output - Bank (run)

```
run:
Rate of Interest is: 7 %
Rate of Interest is: 8 %
BUILD SUCCESSFUL (total time: 0 seconds)
```

Interfaces

Interfaces

- It is one of the ways to achieve abstraction in Java.
- They are used to achieve multiple inheritance and polymorphism.
- It will have only method declaration(abstract methods) and constant attributes in it.
- It cannot be instantiated like how we can't instantiate abstract class.

Interfaces

```
1  public interface Hello{
2  String str = "hello";
3  void sayHello();
4  }
```

- Note : All the variables inside Interface are **public , static and final** even if we don't specify anything.
- Also we **can't change** these default access modifiers **variables of Interfaces in Java**
- Since all variables inside interface are **static**, we can access it directly using interface name.

```
System.out.println(Hello.str);
```

Interfaces

- Also these variables are **final**, so we can't modify them.
- Do not use any access modifiers for interfaces.
- Do not use any access modifer for declared in interfaces.
- Since all the methods inside interface are **abstract**, they must be overridden in the implementing class.
- **Why can't we access methods using interface name ?**
- They are **not static** methods, so we need object to access them.

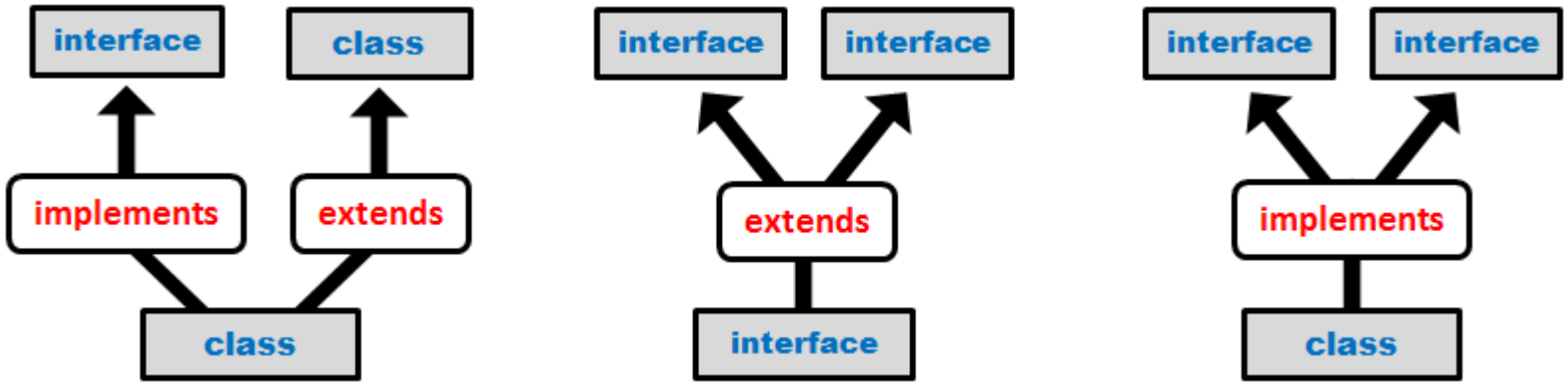
Methods of Interfaces in Java

- class B implements Hello {

```
@Override
  public void sayHello() {
   System.out.println("Hello");
   }
     }

        B b1 = new B();
       b1.sayHello();
```

Multiple Inheritance



Example

Test.java | A.java | B.java | C.java

Source | History

```java
package interfaces;

 interface A {

  void printA();

}
```

Test.java | A.java | B.java | C.java

Source | History

```java
package interfaces;

 interface B {

  void printB();

}
```

Test.java | A.java | B.java | C.java

Source | History

```java
package interfaces;

class C implements A, B {

    @Override
   public void printA() {
   System.out.println("C should Override the method printA()");
     }

   @Override
   public void printB() {
    System.out.println("C should Override the method printB()");
      }
}
```

Example

Test.java | A.java | B.java | C.java

Source | History

```java
package interfaces;

public class Test {

    public static void main(String[] args) {
    C c1 = new C ();

   c1.printA();
   c1.printB();

    }

}
```

Output - interfaces (run)

```
run:
C should Override the method printA()
C should Override the method printB()
BUILD SUCCESSFUL (total time: 0 seconds)
```