Section 9: Inner and Abstract Classes & Interfaces

15 July 2021 19:23

Section 9: Inner and Abstract Classes & Interfaces

Interfaces: So an interface, in java terms, specifies methods that a particular class implements the interface must implement.

Creating a java interface (file instead of class) where we define actual methods. For convention interface file name starts with capital I. Like IName.

```
public interface IName{
     // Methods Declaration just like .h file
     void methodOne();
}
```

public class Names implements Iname {

Don't write code in interface file, code will be written in java file.

```
// Should contain all methods (overridden) that are mentioned in Iname interface to have a valid class.

@override
Public void methodOne()
{
    // code here
}
```

Build in Interfaces by Java:

}

```
List<String> list = new ArrayList<>();
List<String> list = new LinkedList<>();
List<String> list = new Vector<>();
```

In Java, multiple inheritance is only available by implementing several interfaces.

```
Passing a int value to String List can be done by: "" + intValue; Getting int value from string list: Integer.parseInt(StringValue);
```

Inner Classes:

In java, it's possible to nest a class into another class.

There are 4 types of nested classes:-

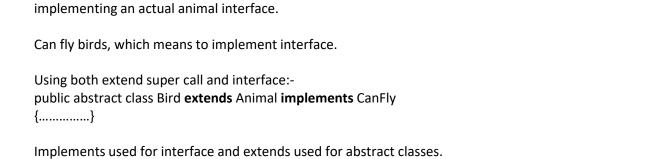
- 1) Static nested classes
- 2) Nonstatic nested class (we called than an inner class)
- 3) Local class (Inner class defined inside of a scope)
- 4) Anonymous class (Nested class without a class name)

Inner Classes Syntax:

```
public class mainClass {
```

```
private class subClass {
      }
}
In Main:
mainClass mc1 = new mainClass(6);
mainClass.subClass = sc1 = mc1.new subClass(1, 12.3);
Abstraction: Abstraction is when you define the required functionality for something without
actually implementing the details.
We focused on what needs to done, not on how's it's to be done.
Interfaces are by definition in java Abstract.
Abstract class implementation:
public abstract class Animal {
  private String name;
  public Animal(String name) {
    this.name = name;
  public abstract void eat();
  public abstract void breathe();
  public String getName() {
    return name;
}
public class Dog extends Animal {
  public Dog(String name) {
    super(name);
  }
  @Override
  public void eat() {
    System.out.println(getName() + " is eating");
  }
Sub class must implement those methods (as overridden) that are in super class
=> We need to check about the relationships:
  1) Is a,
  2) Has a,
  3) Can a
```

Dog is an animal, Bird is an animal -> That make sense to inherited from animal class rather than



If something is common to all, implement that in abstract class, as it's all methods are mandatory to be implemented in sub classes.

Interface cannot have constructors but abstract class have.