Abstract Classes

Abstract Classes

- Java allows abstract classes
  - use the modifier `abstract` on a class header to declare an abstract class

```
 abstract class Vehicle
{  …  }
```

- An abstract class is a placeholder in a class hierarchy that represents a generic concept

```
// An example abstract class in Java
abstract class Shape {
            int color;

            // An abstract function (like a pure
virtual function in C++)
            abstract void draw();
}
```

Abstract Class: Example

```
abstract class Base {
   abstract void fun();
}
class Derived extends Base {
   void fun() { System.out.println("Derived fun() called");
   }
}
class Main {
   public static void main(String args[]) {

     // Uncommenting the following line will cause compiler
   error as the
     // line tries to create an instance of abstract class.
     // Base b = new Base();

     // We can have references of Base type.
     Base b = new Derived();
     b.fun();
   }
}
```

Abstract Classes

- An abstract class often contains *abstract methods*, though it doesn't have to
  - Abstract methods consist of only methods *declarations*, without any method body

- In Java, we can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated, but can only be inherited.
- An abstract class cannot be instantiated

  (why?) For any abstract java class we are not allowed to create an object

- Abstract classes can also have final methods (methods that cannot be overridden)

Java Interface

- A Java *interface* is a collection of constants and abstract methods
  - abstract method: a method header without a method body; we declare an abstract method using the modifier `abstract`
  - since all methods in an interface are abstract, the `abstract` modifier is usually left off

- Methods in an interface have public visibility by default

Interface: Syntax

interface is a reserved word

```
// A simple interface
interface Player
{
        final int id = 10;
        int move();
}
```

A semicolon immediately follows each method header

No method in an interface has a definition (body)

Implementing an Interface

- A class formally implements an interface by
  - stating so in the class header in the `implements` clause
  - a class can implement multiple interfaces: the interfaces are listed in the implements clause, separated by commas

- If a class asserts that it implements an interface, it must define all methods in the interface or the compiler will produce errors

Implementing Interfaces

```
// An example to show that interfaces can
// have methods from JDK 1.8 onwards
interface In1
{
          final int a = 10;
          default void display()
          {
                    System.out.println("hello");
          }
}

// A class that implements the interface.
class TestClass implements In1
{
          // Driver Code
          public static void main (String[] args)
          {
                    TestClass t = new TestClass();
                    t.display();
          }
}
```

implements is a reserved word

Each method listed in Doable is given a definition

Interfaces: An Example

```
import java.io.*;
// A simple interface
interface In1
{
// public, static and final
        final int a = 10;

        // public and abstract
        void display();
}
// A class that implements the interface.
class TestClass implements In1
{
        // Implementing the capabilities of
        // interface.
        public void display()
        {
                System.out.println("Geek");
        }
        // Driver Code
        public static void main (String[] args)
        {
                TestClass t = new TestClass();
                t.display();
                System.out.println(a);          }
}
```

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

More Examples

```
interface Inf1
{ public void method1();
}
interface Inf2 extends Inf1
{
public void method2();
}
public class Demo implements Inf2
{ /* Even though this class is only implementing the * interface Inf2, it has to implement all
the methods * of Inf1 as well because the interface Inf2 extends Inf1 */
public void method1()
{
System.out.println("method1");
}
 public void method2()
{
System.out.println("method2");

 }
public static void main(String args[])
{ I
nf2 obj = new Demo();
 obj.method2();
 }
}
```

Interface Hierarchies

- Inheritance can be applied to interfaces as well as classes
- One interface can be used as the parent of another
- The child interface inherits all abstract methods of the parent
- A class implementing the child interface must define all methods from both the parent and child interfaces
- Note that class hierarchies and interface hierarchies are distinct (they do not overlap)