

# CL309

## Object Oriented Analysis and Design Lab

### Lab # 08

## Object-Oriented Programming: Strings & Interfaces

### Objectives

Understanding the concepts of

- Interfaces
- Strings

### 5.1. Interface

In Java, this multiple inheritance problem is solved with a powerful construct called **interfaces**. Interface can be used to define a generic template and then one or more abstract classes to define partial implementations of the interface. Interfaces just specify the method declaration (implicitly public and abstract) and can only contain fields (which are implicitly public static final). Interface definition begins with a keyword interface. An interface like that of an abstract class cannot be instantiated. All methods in an interface are by default public and abstract. All variables are final and static.

Multiple Inheritance is allowed when extending interfaces i.e. one interface can extend none, one, or more interfaces. Java does not support multiple inheritance, but it allows you to extend one class and implement many interfaces.

If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class.

**Example 1:** Below is an example of a Shape interface

```
interface Shape {  
  
    public double area();  
    public double volume();  
}
```

Below is a Point class that implements the Shape interface.

```
public class Point implements Shape{
    int x, y;
    public Point() {
        x = 0;
        y = 0;
    }
    public double area() {
        return 0;
    }
    public double volume() {
        return 0;
    }
    public void show() {
        System.out.println("point: " + x + "," + y);
    }
    public static void main(String args[]) {
        Point p = new Point();
        p.show();
    }
}
```

### 5.1.1. Implementing multiple interfaces

A concrete class can only extend one super class, but it can implement multiple Interfaces. The Java programming language does not permit multiple inheritance but interfaces provide an alternative. All abstract methods of all interfaces have to be implemented by the concrete class. A concrete class extends one super class but implements multiple Interfaces:

```
public class ComputerScienceStudent
extends Student
implements PersonInterface, AnotherInterface, Thirddinterface
{
    // All abstract methods of all interfaces
    // need to be implemented.
}
```

Interfaces are not part of the class hierarchy. However, interfaces can have inheritance relationship among themselves.

```
public interface PersonInterface
{
    void doSomething();
}
public interface StudentInterface
extends PersonInterface
{
    void doExtraSomething();
}
```

### Example:

```
public interface Relation {

    public boolean isGreater(Object a);
    public boolean isLess(Object a);
    public boolean isEqual(Object a);

}

public class Line implements Relation{
    private int x1, x2, y1, y2;

    public Line(int x1, int x2, int y1, int y2) {
        super();
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
    }

    public double getLength()
    {
        return Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
    }

    public boolean isEqual(Object b) {
        double len1=getLength();
        double len2=((Line)b).getLength();
        if(len1==len2)
            return true;
        return false;
    }

    public boolean isGreater(Object b) {
        double len1=getLength();
        double len2=((Line)b).getLength();
        if(len1>len2)
            return true;
    }
}
```

```
        return false;
    }

    public boolean isLess( Object b) {
        double len1=getLength();
        double len2=((Line)b).getLength();

        if(len1<len2)
            return true;
        return false;
    }
}
```

### 5.1.2. Polymorphism using Interfaces

**Polymorphism** means one name, many forms. Polymorphism allows a reference to denote objects of different types at different times during execution. A super type reference exhibits polymorphic behavior, since it can denote objects of its subtypes.

There are 3 distinct forms of Java Polymorphism;

- Method overloading (Compile time polymorphism)
- Method overriding through inheritance (Run time polymorphism)
- Method overriding through the Java interface (Run time polymorphism)

```
interface Shape {

    public double area();
    public double volume();
}

class Cube implements Shape {

    int x = 10;
    public double area( ) {

        return (6 * x * x);
    }

    public double volume() {
        return (x * x * x);
    }

}

class Circle implements Shape {

    int radius = 10;
```

```

    public double area() {
        return (Math.PI * radius * radius);
    }
    public double volume() {
        return 0;
    }
}

public class PolymorphismTest {

    public static void main(String args[]) {
        Shape[] s = { new Cube(), new Circle() };
        for (int i = 0; i < s.length; i++) {
            System.out.println("The area and volume of " + s[i].getClass()
                               + " is " + s[i].area() + " , " + s[i].volume());
        }
    }
}

```

### 5.1.3. Interfaces Can Be Extended

One interface can inherit another by use of the keyword `extends`. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

Following is an example:

```

// One interface can extend another.
interface A {
    void meth1();
    void meth2();
}
// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
    void meth3();
}
// This class must implement all of A and B
class MyClass implements B {
    public void meth1() {
        System.out.println("Implement meth1().");
    }
    public void meth2() {
        System.out.println("Implement meth2().");
    }
    public void meth3() {
        System.out.println("Implement meth3().");
    }
}
class IFExtend {
    public static void main(String arg[]) {
        MyClass ob = new MyClass();
        ob.meth1();
        ob.meth2();
        ob.meth3();
    }
}

```

}

#### 5.1.4. Interface vs. Abstract Class

1. All methods of an Interface are abstract methods while some methods of an Abstract class are abstract methods

- Abstract methods of abstract class have **abstract** modifier

2. An interface can only define constants while abstract class can have fields

3. Interfaces have no direct inherited relationship with any particular class, they are defined independently

- Interfaces themselves have inheritance relationship among themselves