CS–313
Object Oriented Analysis and Design (OOA&D)
Lab # 05

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, Thirdinterface
{
      // 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

Lab Assignment

Task 1:

Create a class Circle with radius. This class implements the Relation interface discussed in the lab to check if a Circle bigger, smaller, or equal to another Circle.

Task 2:

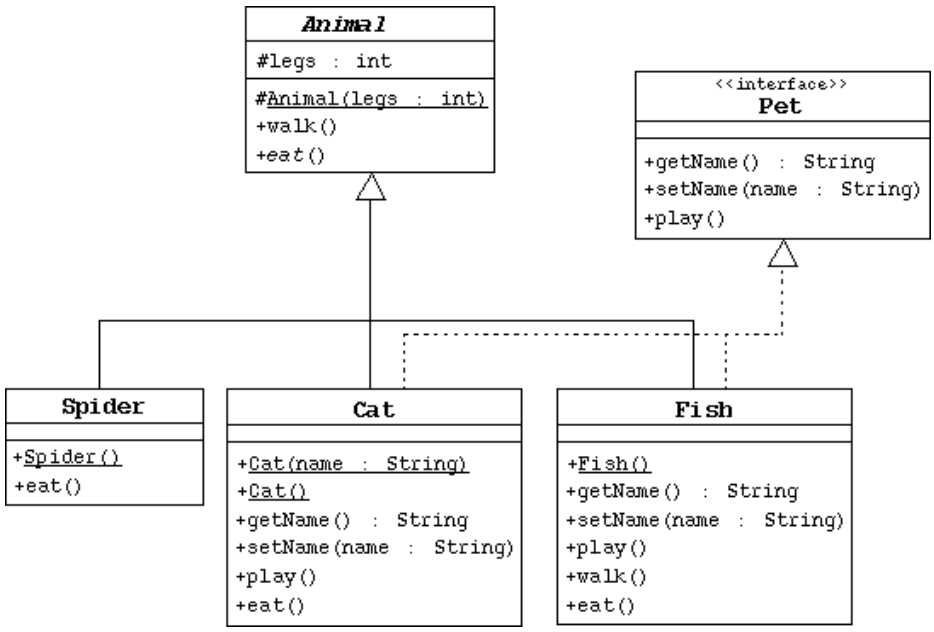
Take two random strings from user, and show the following.

- Length of both strings
- Concatenation of both strings
- Convert to upper case
- Convert to Lower case
- Take out a substring from both strings
- Should be able to replace a word in a string
- Check whether two strings are equal
- Take out a char from the string at a given index.

Home Assignment

Task 1:

In this exercise you will create a hierarchy of animals that is rooted in an abstract class Animal. Several of the animal classes will implement an interface called Pet. You will experiment with variations of these animals, their methods, and polymorphism.



1. Create the `Animal` class, which is the abstract superclass of all animals.

a) Declare a protected integer attribute called `legs`, which records the number of legs for this animal.
b) Define a protected constructor that initializes the `legs` attribute.
c) Declare an abstract method `eat`.
d) Declare a concrete method `walk` that prints out something about how the animals walks (include the number of legs).

2. Create the `Spider` class.

a) The `Spider` class extends the `Animal` class.
b) Define a default constructor that calls the superclass constructor to specify that all spiders have eight legs.
c) Implement the `eat` method.

3. Create the `Pet` interface specified by the UML diagram.
4. Create the `Cat` class that extends `Animal` and implements `Pet`.

a) This class must include a `String` attribute to store the name of the pet.
b) Define a constructor that takes one `String` parameter that specifies the cat's name. This constructor must also call the superclass constructor to specify that all cats have four legs.
c) Define another constructor that takes no parameters. Have this constructor call the previous constructor (using the `this` keyword) and pass an empty string as the argument.
d) Implement the `Pet` interface methods.
e) Implement the `eat` method.

5. Create the `Fish` class. Override the `Animal` methods to specify that fish can't walk and don't have legs.

6. Create a `TestAnimals` program. Have the `main` method create and manipulate instances of the classes you created above. Start with:

```
Fish d = new Fish();
Cat c = new Cat("Fluffy");
Animal a = new Fish();
Animal e = new Spider();
Pet p = new Cat();
```

Experiment by: a) calling the methods in each object, b) exhibiting polymorphism using interfaces and inheritance, and c) using `super` to call super class methods.

Lab Test:

Create four strings containing the values “object”, “ oriented”, “ analysis”, “ and design”. Create an integer containing the value 2012.

1. Create a fifth string that contains the concatenation of all the above four strings and the integer.

2. Create a sixth string that contains the uppercase representation of the fifth string.

3. Create a seventh string from the sixth string that contains only the substring “analysis and design”.

4. Compare the fifth and the sixth strings and display a message whether both are equal or not. Do it first considering the case of the strings and second time ignoring the case of strings.

5. Create an eight string from the fifth string, replacing the occurrence of “and” by “&” using the Java String functions.