OOJ Lecture 7
Interfaces and Polymorphism

- Reading: Savitch, Chapter 7
- Reference: Big Java, Horstman, Chapter 9

Objectives

- To learn about and use interfaces
- To appreciate how interfaces may be used to decouple classes

Interface

- This is a Java--only feature.
- An interface is a contract between classes
- All methods in an interface are abstract
  - no implementation, can not be instantiated
- All methods in an interface are automatically public
- An interface does not have instance variables

Declaring an Interface

- Format:

```
[modifier]  interface  interface_name
  [extends interface1,interface2,...]
{      [declaring variables;]
       [declaring methods signatures;]
}
```

- All variables declared in an interface have "static" and "final" attributes. Hence all are constants, no instance variables
- All methods declared in an interface have "public" and "abstract" attributes.
- Their methods cannot be "static" since a static method can't be abstract.

Interfaces

- An ***interface*** is a type that specifies method headings.

  Example:

```
public interface Writeable
{
        public String toString();
        public void writeOutput();
}
```

- You can make a method more general by using an interface as a type for a parameter.
  - An object of any class that ***implements*** the interface can be passed as the parameter.

```
public void display(Writeable displayObj)
{
        displayObj.writeOutput();
}
```

Implementing an Interface

- Remember the way to define a class:

```
class A [extends B] implements
        interface-name1, interface2,…
  {
        [variable declaration;]
        [method implementations;]
  }
```

A class can implement more than one interface

Implementing an Interface

- A class that implements an interface must
  - contain complete definitions for all of the methods specified in the interface
  - be declared as implementing the interface

    `implements` *`Interface_Name`*
- Any class that implements the `Writeable` interface must have complete definitions of `toString` and `writeOutput`.
- There can be many different classes that implement an interface.

Examples

```
public interface A {
        int x = 0;          //static and final
        void f();           //abstract
}
```

- Note, we do not use keywords like static for the variable x and abstract for method f().

```
public Interface Simple1
{    int a=23;
     void absMethod1();  }
public Interface Simple2
{
   string str = "The string in Simple2";
   void absMethod2();    }
```

Interface Extension

```
interface Combined extends Simple1, Simple2
{
  void absMethod3();
}
```

```
/** ImpClass is required to implement absMehtod1(), absMethod2(),
 *  and  absMethod3(). Otherwise,  ImpClass would also need to be
 *  declared   as an abstract class */
```

```
class ImpClass implements Combined
{ public absMethod1()
  {   .... // defintion provided here   }
  public absMethod2()
  {   .... // defintion provided here   }
  public absMethod3()
  {    …    // defintion provided here   }
  other class methods
}
```

A concrete example

```
public interface Countable
{
   int X=20;    //declaring interface constants
   int Y=30;
   public void Counting(); //declaring a method
}
class Bigger implements Countable {
    private int x = X;
    private int y = Y;
    private int sum = 0;
    public void Counting()
    {   // implements interface method
        sum = x + y;
        System.out.println("Sum is "+sum);
    }
}
```

A concrete example – cont'd

```
class Smaller extends Bigger implements
  Countable{
   private int x = X, y = Y;
   private int subtraction = 0;
   public void Counting(){
      subtraction = y - x;
      System.out.println("Subtraction is"
                         + subtraction);
   }
}
public class ResultOfCount {
  public static void main(String args[])
  { Bigger A = new Bigger();
   A.Counting();
   Smaller B = new Smaller();
   B.Counting();
  }
}
```

Explanations

- The output of the program:

  Sum is 50

  Subtraction is 10

- Two classes implement (override) the `Counting()` method from `interface Countable` by using `implements Countable` in their class definitions.
- Note:
  - An interface may have many methods.
  - If a class implements an interface, but only implements some of its methods, then this class becomes an abstract method. It cannot be instantiated.

Developing Reusable Solutions

- We can define an Interface *Measurable* to declare a method getMeasure():

```
public interface Measurable
{
    double getMeasure();
}
```

- It can be implemented by any class

Class DataSet: An Interface as Parameter

```
//Reference: Horstman, Chapter 9
public class DataSet  //can calculate sum and maximum
{  . .
   public void add(Measurable x) //interface Measurable
   {   sum = sum + x.getMeasure();
       if (count == 0 ||
           maximum.getMeasure() < x.getMeasure())
       {   maximum = x;
           count++;                }
   }
   public Measurable getMaximum()
   {   return maximum;       }

   private double sum;
   private int count;
   private Measurable maximum;
}
```

Reuse - Common operations

- Notice that any class that implemented interface `Measurable` will be able to use the DataSet class to calculate sum and find the maximum value.
- Different classes can have different implementation for method *getMeasure()*
- If an objet x is a `Measurable` type, it has method *getMeasure* for the add method:

```
sum = sum + x.getMeasure();
 if (count == 0||
    maximum.getMeasure()
 < x.getMeasure())
   maximum = x;
```

Making BankAccount and Coin Classes as Measurable classes

In order to use DataSet for calculating sum …, all classes must implement Measurable interface:

```
class BankAccount implements Measurable
{     public double getMeasure()
      {
           return balance;
      }
      //additional methods and fields
}
class Coin implements Measurable
{     public double getMeasure()
      {
          return value;
      }
      // additional methods and fields
}
```

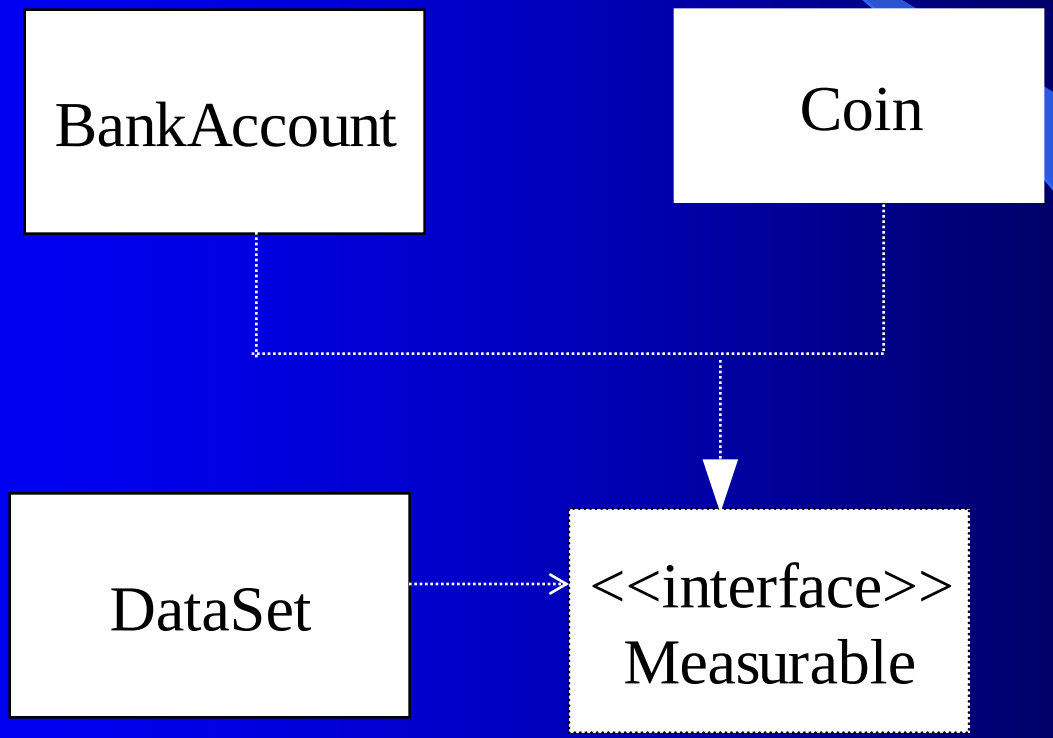
File DataSetTest.java

```
//Reference: Horstman, Chapter 9
public class DataSetTest
{ public static void main(String[] args)
   {
      DataSet bankData = new DataSet();
      bankData.add(new BankAccount(0));
      bankData.add(new BankAccount(10000));
      bankData.add(new BankAccount(2000));
      System.out.println("Average balance = "
        + bankData.getAverage());
      Measurable max = bankData.getMaximum();
      System.out.println("Highest balance = "
          + max.getMeasure());
```

File DataSetTest.java

```
   DataSet coinData = new DataSet();
   coinData.add(new Coin(0.25,"quarter"))
  ;
   coinData.add(new Coin(0.1, "dime"));
   coinData.add(new Coin(0.05, "nickel"))
  ;
   System.out.println("Average coin value
           = " + coinData.getAverage());
   max = coinData.getMaximum();
  System.out.println("Highest coin value
  =              "  + max.getMeasure());
  }
}
```

UML Diagram of DataSet and Related Classes



Summary Interface

- An ***interface*** is a type.
- An interface definition is stored in a .java file and compiled just the same as a class.
  - e.g, `Measurable.java`
- It is very useful for reuse programming that use interface objects as parameters of methods.
- It extends the sharing/acceptability of a method.
- An object of any class that ***implements*** the interface can be passed as the parameter to use the method.