OOJ Lecture 8
Interfaces and Polymorphism

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

Objectives

- To be able to convert between supertype and subtype references

Converting Between Types

- Is that OK ?
  - if the definition is

    `public void add(`**`Measurable x`**`)`
  - But call it with a class object as follows?

    `Bankdata.add(new BankAccount(10000))`
- Yes, if the class BankAccount implemented the interface `Measurable`.

Converting Between Types

- It can convert from class type to realised interface type:

```
BankAccount account =
                        new BankAccount(10000);
Measurable x = account;      // OK
```

- Same interface type variable can hold reference to Coin

```
x = new Coin(0.1, "dime");  // OK
```

- Cannot convert between unrelated types

```
x = new Rectangle(5, 10, 20, 30); // ERROR
```

Reference: Horstman, Chapter 9

Converting Between Types

- an interface object cannot convert to a class object automatically

Casts

- Add coin objects to DataSet

```
//Reference: Horstman, Chapter 9
DataSet coinData = new DataSet();
coinData.add(new Coin(0.25, "quarter"));
coinData.add(new Coin(0.1, "dime"));
   ...
```

- Get largest coin with getMaximum method:

```
Measurable max = coinData.getMaximum();
```

Now, what can you do with the max reference?

Casts

- max is a realised *Measurable* object by Coin. But it is not of type Coin and cannot use Coin's method

```
String name = max.getName();
// ERROR since max doesn't have a method getName()
```

- We know it's a coin, but the compiler doesn't. We tell the compiler by applying an explicit cast:

```
Coin maxCoin = (Coin)max;
String name = maxCoin.getName();
```

Casts

- If you are wrong and max isn't a coin, the compiler throws an exception
- It is different for a cast:
  - between number types
    - Lose information
  - between object types
    - Take a risk of causing an exception
- Should test before casting
  - Use `instanceof` operator to test

The instanceof Operator

- Use instanceof for safe casts:

```
if (max instanceof Coin)
{
    Coin maxCoin = (Coin)max;
    . . .
}
```

The instanceof Operator

- *Syntax:*

```
object instanceof ClassName
```

- Example:

```
if (x instanceof Coin)
{
   Coin c = (Coin)x;
}
```

- Purpose:
  - To return true if the *object* is an instance of *ClassName* (or one of its subclasses), false otherwise

Class exercises: Interface concepts

- What is wrong with the following interface?

```
public interface SomethingIsWrong {
   public void aMethod(int aValue)
   {   System.out.println("Hi Mom");
   }
}
```

- Are the following interfaces valid?

```
public interface Marker {     }

public interface rabbit extends animal,
  Marker
{
    void method1(int x);
}
```

Interface

- We can add one more method into interface Measurable:

```
public interface Measurable
{   //Reference: Horstman, Chapter 9
    public int compareTo(Object other);
    public double getMeasure();
}
```

- We can rewrite the method add in DataSet class to use compareTo(..) for the maximum object and minimum object.
- We can add a method search(..) to find a given object from an array of objects

Interface

```
public class DataSet{  //modified DataSet
   //Reference: Horstman, Chapter 9
   public DataSet()
   {  sum = 0;
      count = 0;
      maximum = null; minimum = null;
   }

   public void add(Measurable x)
   {  sum = sum + x.getMeasure();
      if(minimum == null || x.compareTo(minimum)== -1)
         minimum = x;
      if(maximum == null || x.compareTo(maximum) > 0)
         maximum = x;
      count++;
   }
```

Interface

```
public double getAverage()
{
     if (count == 0) return 0;
     else return sum / count;
}

public Measurable getMinimum()
{
     return minimum;
}

public Measurable getMaximum()
{
     return maximum;
}
```

Interface

```
public static boolean Search(Measurable[] x, Object target)
{   int count = 0;
    int equal = 1; //1 is greater, 0 is equal, -1 is less
    while(count < x.length && (equal!=0))
    {     equal = x[count].compareTo(target);
                  //compareTo is a method of x
          count++;
    }
    if (equal == 0) return true;
    else return false;
}
private double sum;
private Measurable maximum;
private Measurable minimum;
private int count;
}
```

Interface

```
//Reference: Horstman, Chapter 9
public class Coin implements Measurable
{       public Coin(double aValue, String aName)
        {   value = aValue;
            name = aName;
        }
       public double getValue()
        {   return value;
        }
        public String getName()
        {   return name;
        }
        public double getMeasure()
        {   return value;
        }
}
```

Interface

```
/*  Compare Coin objects
    @param other the Object to be compared
    @return a negative integer, zero, or a positive integer as this
    object is less than, equal to, or  greater than the specified object  */
  public int compareTo(Object other)
  { //Reference: Horstman, Chapter 9
       if (other instanceof Coin)
       {
          Coin b = (Coin) other;
          if (value < b.value) { return -1; }
          else if (value > b.value) { return 1; }
          else { return 0; }
       } else  { return -999; }
  }
   private double value;
   private String name;
 } // end of class, from slide 13
```

Interface

```
//Reference: Horstman, Chapter 9
public class Purse implements Measurable
{
   public Purse()
   {   total = 0;
   }
   public void add(Coin aCoin)
   {   total = total + aCoin.getValue();
   }
   public double getTotal()
   {   return total;
   }
   public double getMeasure()
   {   return total;
   }
}
```

Interface

```
/*  Compare Purse objects
    @param other the Object to be compared
    @return a negative integer, zero, or a positive integer as this
    object  is less than, equal to, or  greater  than the specified object
   */
  public int compareTo(Object other)
  {   if (other instanceof Purse)
      {   Purse b = (Purse)other;
          if (total < b.total)
             return -1;
          else if (total > b.total)
             return 1;
          else
             return 0;                    }
      else return -999
  }
  private double total;
}
```

Interface

```
//Reference: Horstman, Chapter 9
public class TestCoinPurse
{
   public static void main(String[] args)
   {
       Purse p[] = new Purse[2];
       p[0] = new Purse();
       p[1] = new Purse();
       Coin c[] = new Coin[3];
       c[0] = new Coin(0.01, "pennies");
       c[1] = new Coin(0.05, "nickels");
       c[2] = new Coin(0.25, "quaters");
       p[0].add(c[0]);
       p[1].add(c[1]);
       p[1].add(c[2]);
```

Interface

```
Coin findCoin = c[1];
if ( DataSet.Search(c, findCoin))
{
    System.out.println("Found" +
                                findCoin);
}
else
{
    System.out.println("Not Found" +
                                findCoin);
}
}
```