Intermediate Programming

Lecture #13 – Interfaces

What is an Interface?

- A Java interface specified a set of methods that any class that implements the interfacesmust have.
- An Interface is a type, which means that you can degine methods with parameters whose type is an interface.
- It can be considered an extreme form of an abstract class.

What is an Interface

- An interface specifies the headings for methods that the implementing classes must have.
- An interface contains only the interface's method header.
- Implementing an interface requires that we add implement **`InterfaceName`** to the end of the class's header.

Example: `Ordered.java`

```
// An interface called ordered


public interface Ordered
{
    //Does our object come before another object?
    public boolean precedes(Object other);

    // Does our object come after another object?
    public boolean follows(Object other);
}
```

How to Use an Interface

- Implementing an interface requires that we add
  ```
  implements InterfaceName;
  ```
  to the end of the class's header.
- If you wish a class to implement more than one class, you have to include them in class header:

```
public class MyClass implements SomeInterface,
                          AnotherInterface {
```

How to Use an Interface

- Classes must implement all the method headings in the interface definition.
- A class can be a derived class from a given base class AND an implementation of an interface.
- Ex

  -
    ```
    public class OrderedHourEmployee extends
              Hourly Employee implements Ordered;
    ```

OrderedHourlyEmployee.java

```
// OrderedHourlyEmployee is derived from the
// HourlyEmployee class, adding the methods
// precedes and follows.

public class OrderedHourlyEmployee
    extends HourlyEmployee implements Ordered  {
    public boolean precedes(Object other) {
        if (other == null)
            return false;
        else if (!(other instanceof
                   OrderedHourlyEmployee))
            return false;
```

```
    else {
       OrderedHourlyEmployee
            otherOrderedHourlyEmployee =
            OrderedHourlyEmployee) other;
       return (getPay() <
       otherOrderedHourlyEmployee.getPay());
    }
}
```

```
    public boolean follows(Object other) {
        if (other == null)
            return false;
        else if (!(other instanceof
                   OrderedHourlyEmployee))
            return false;
        else {
            OrderedHourlyEmployee
                  otherOrderedHourlyEmployee =
                  (OrderedHourlyEmployee) other;
            return (otherOrderedHourlyEmployee.
                        precedes(this));
        }
    }
}
```

More on Interfaces

- Interfaces are normally declared as **`public`**.
- An interface is a type, i.e., you can write a method with a parameters whose type is an interface.
- Interfaces serve a purpose that is similar to that of a base class, but it is NOT a base class.

More on Interfaces

- Previously we have seen regular classes be defined as implementing an interface. It is also possible for abstract classes to implement an interface.
- Interfaces can also be derived from base interfaces; this is called ***extending*** the interface.

`MyAbstractClass.java`

```
// An example of an abstract class that
// implements an interface

public abstract class MyAbstractClass
                  implements Ordered {
    private int number;
    private char grade;
```

```
   public boolean precedes(Object other) {
        if (other == null)
           return false;
        else if (!(other
                  instanceof HourlyEmployee))
           return false;
        else {
            MyAbstractClass
                  otherOfMyAbstractClass =
                    (MyAbstractClass) other;
            return (this.number <
                otherOfMyAbstractClass.number);
        }
    }
   public abstract boolean follow(Object other);
}
```

Example: Selection Sort

- A selection sort traverses an array and determines the smallest element and swaps it into its rightful place.
- It then repeats the process for the second smallest, third smallest, etc. until every element is in the correct place.

SelectionSort.java

```
public class SelectionSort  {
  // Precondition: numberUsed <= a.length;
  // The first numberUsed indexed variables
  // have values.
  // Action - sortds a so that a[0] <= a[1], etc.

  public static void sort(double[] a,
                              int numberUsed) {
    int index, indexOfNextSmallest;
```

```
     for (index = 0;  index < numberUsed - 1;
                                        index++)   {
       // Place the correct value in a[index]
       indexOfNextSmallest = indexOfSmallest
                             (index, a, numberUsed);
       interchange(index, indexOfNextSmallest, a);
     }
   }
```

```
// indexOfSmallest() - Returns the index of the
//                      smallest value that hasn't
//                      been placed in its proper
//                      spot.
public static int indexOfSmallest
             (int startIndex, double[] a,
                            int numberUsed) {
  double min = a[startIndex];
  int indexOfMin = startIndex;
  int index;
```

```
  for (index = startIndex + 1;
             index < numberUsed; index++)
    if (a[index] < min) {
      min = a[index];
      indexOfMin = index;
      // min is the smallest of a[startIndex]
      // through a[index]
    }
           return indexOfMin;
}
```

```
// Interchange - a[i] and a[j] switch values
// Precondition - i and j are valid indices
// Postcondition - a[i] and a[j] have exchanged
//                  values
public static void interchange(int i, int j,
                                double[] a) {
        double temp;

        temp = a[i];
        a[i] = a[j];
        a[j] = temp;

    }
}
```

SelectionSortDemo.java

```
public class SelectionSortDemo  {
  public static void main(String[] args) {
    double [] b = {7.7, 5.5, 11, 3, 16, 4.4,
                 20, 14, 13, 42};
    System.out.println
          ("Array contents before sorting:");
    int i;
    for (i = 0;  i < b.length; i++)
      System.out.print(b[i] + " ");
    System.out.println();

    SelectionSort.sort(b, b.length);
    System.out.println("Sorted array values:");
    for (i = 0;  i < b.length;  i++)
      System.out.print(b[i] + " ");
    System.out.println();
  }
}
```

The Comparable Interface

- The selection sort that we wrote sorts arrays of doubles and could be tweeked to sort integers, characters, strings or even objects. But having to make these changes can a nuisance.
- The Comparable interface has only one method that must be written: **`compareTo()`**, which allows us to write a more general method to do the sorting.

compareTo()

- The header for **compareTo()** is:

```
public int compareTo(Object other);
```

- **compareTo()** returns:
  - a negative number if the calling object "comes before" the parameter **other**.
  - zero if the calling object "equals" the parameter **other**.
  - a positive number if the calling object "comes after" the parameter **other**.

Generalized SelectionSort.java

```
public class GeneralizedSelectionSort
{
  // Precondition: numberUsed <= a.length;
  // The first numberUsed indexed variables have
  // values.
  // Action - sortds a so that a[0] <= a[1], etc.

  public static void sort(Comparable [] a,
                             int numberUsed) {
    int index, indexOfNextSmallest;
    for (index = 0;  index < numberUsed - 1;
                                   index++)  {
      // Place the correct value in a[index]
      indexOfNextSmallest =
          indexOfSmallest(index, a, numberUsed);
      interchange(index, indexOfNextSmallest, a);
    }
  }
```

```
// indexOfSmallest() - Returns the index of the
//                      smallest value that
//                      hasn't been placed in
//                      its proper spot.
public static int indexOfSmallest
         (int startIndex, Comparable[] a,
                           int numberUsed) {
  Comparable min = a[startIndex];
  int indexOfMin = startIndex;
  int index;
```

```
    for (index = startIndex + 1;
                 index < numberUsed; index++)
      if (a[index].compareTo(min) < 0) {
        min = a[index];
        indexOfMin = index;
        // min is the smallest of a[startIndex]
        // through a[index]
      }
      return indexOfMin;
    }
```

```
    // Interchange - a[i] and a[j] switch values
    // Precondition - i and j are valid indices
    // Postcondition - a[i] and a[j] have
    //                 exchanged values
    public static void interchange(int i, int j,
                                Comparable[] a) {
      Comparable temp;

      temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }
}
```

ComparableDemo.java

```
// Demonstrates sorting arrays for classes that
// implement the Comparable interface
public class ComparableDemo
{
  public static void main(String[] args) {
    Double[] d = new Double[10];
    for (int i = 0; i < d.length; i++)
      d[i] = new Double(d.length - i);

    System.out.println("Before sorting:");
    int i;
```

```
    for (i = 0;  i < d.length; i++)
      System.out.print(d[i].doubleValue() +
                                          ", ");
    System.out.println();

    GeneralizedSelectionSort.sort(d, d.length);

    System.out.println("After sorting:");
    for (i = 0;  i < d.length; i++)
      System.out.print(d[i].doubleValue() +
                                          ", ");
    System.out.println();

    String[] a = new String[10];
    a[0] = "dog";
    a[1] = "cat";
    a[2] = "cornish game hen";
    int numberUsed = 3;

    System.out.println("Before sorting:");
    for (i = 0;  i < numberUsed; i++)
      System.out.print(a[i] + ", ");
    System.out.println();

    GeneralizedSelectionSort.sort(a, numberUsed);
    System.out.println("Before sorting:");
    for (i = 0;  i < numberUsed; i++)
      System.out.print(a[i] + ", ");
    System.out.println();
  }
}
```

Interfaces and Defining Constants

- An interface can be used to define constants:

```
public interface MonthNumbers  {
  public static final int JANUARY = 1,
    FEBRUARY = 2, MARCH = 3, APRIL = 4, MAY = 5,
    JUNE = 6, JULY = 7, AUGUST = 8,
    SEPTEMBER = 9, OCTOBER = 10, NOVEMBER = 11,
    DECEMBER= 12;
}
```

Interfaces and Constants

- Any class implementing the MnthNumbers interface automatically has access to these constants:

```
public class DemoMonthNumbers
            implements MonthNumbers  {
  public static void main(String[] args) {
    System.out.println
      ("The number for January is " + JANUARY);
    }
}
```