AP CS Unit 7: Interfaces. Programs

You cannot use the less than (<) and greater than (>) operators with objects; it won’t compile because it doesn’t always make sense to say that one object is less than or greater than another. If you need to compare objects, Java provides the following interface:

```
public interface Comparable{
        public int compareTo( Object x );
}
```

The compareTo method returns a negative value if this object is less than the parameter; zero if they are equal, and a positive value if this object is greater than the parameter. The String and Integer classes both implement this interface.

Program 1a.

Complete the up method so that rearranges the elements in the array in increasing order.

Assume the length of the array is exactly 2.

You may not use Arrays.sort.

Do not cast anything.

Note. You will probably get a warning about an unchecked call to compareTo(T). Ignore it for now.

```
public class Runner1{
    public static void main( String [] args ){
        Integer [] a1 = { 8, 4 };
        up( a1 );
        for ( int n : a1 )
                System.out.println( n );

        String [] a2 = { "rock", "paper" };
        up( a2 );
        for ( String s : a2 )
                System.out.println( s );

        String [] a3 = { "ant", "boat" };
        up( a3 );
        for ( String s : a3 )
                System.out.println( s );
    }

    public static void up( Comparable[] c ){
        complete this method
    }
}
```

```
4
8
```

```
paper
rock
```

```
ant
boat
```

Program 1b.

Copy these and run the main method. They compile but crash. Why?

```
import java.util.*;

public class Runner2{
   public static void main(String [] args){
      Bat [] b1 = new Bat[ 2 ];
      b1[0] = new Bat( 34 );
      b1[1] = new Bat( 16 );
      Arrays.sort( b1 );
      for ( Bat b : b1 )
           System.out.println( b );
   }
}
```

```
public class Bat{
   private int n;

   public Bat( int n ){
      this.n = n;
   }

   public String toString(){
      return "bat " + n;
   }
}
```

Program 2.

Complete the max method so that it returns the largest value in the array.

And don't even think of using Arrays.sort.

```
public class Runner3{
        public static void main( String [] args ){
                String [] a1 = { "hat", "what", "why", "apple" };
                Comparable c = max( a1 );
                System.out.println( c );                    // why
                Integer [] a2 = { 56, 22, 109, 5, 8, 13 };
                c = max( a2 );
                System.out.println( c );                    // 109
        }

        public static Comparable max( Comparable[] c ){
                write the code
        }
}
```

Program 3. This program consists of 5 files.

1) Create a Product interface that has two methods:

```
        public double getPrice()
        public void setPrice( double x )
```

2) Create a Fruit class that implements Product.

- There are two instance variables:
```
        private double cost;
        private String name;
```
- The constructor has two parameters to initialize the instance variables.
- Override the equals method. If the prices and names are the same then return true, otherwise return false.
- Override the toString method so it returns the name and cost.
- Remember that you are implementing the Product interface.

3) Create a Book class that implements Product

- There are three instance variables:
```
        private double cost;
        private String title;
        private int pages;  // # pages in book
```
- The constructor has three parameters to initialize the instance variables.
- Override the equals method. If the titles are the same then return true, otherwise return false. The cost and number of pages do not matter.
- Override the toString method so that it shows the title, cost, and # of pages. Put quotes around the title.
- Write an accessor method that returns the number of pages.
- And I'm not going to remind you about those other two methods you must write.

4) Complete the Grocer class

```
public class Grocer{
    private Product [] inventory;                // contains all the products the grocer has

    public Grocer( int size ){
        inventory = new Product[ size ];
    }
    public void add( Product p ){
        Add p to the first null element found in inventory.  If there are no null elements, nothing
happens.  No casting needed.
    }
    public void remove( Product p ){
        Search for the first element that equals p and set that element to null.  No casting needed.
    }
    public void increasePrices( double p ){
        Increase the price of all products by p percent.  For example, if p is 0.2 then prices
increase by 20%.  Beware of elements that contain null values.  No casting needed.
    }
    public int totalPages(){
        Returns the number of pages in all the books in the inventory.
    }
    public String toString() {
        Return a string that contains the result of calling each non-null element's toString
method.  Put a new line character between elements.  No casting needed.
    }
}
```

5) Test your code with this class. The output is to the right.

```
public class RunStore{
    public static void main( String [] args ){
        Grocer g = new Grocer( 5 );
        g.add( new Book( "Hey", 2, 25 ) );
        g.add( new Fruit( "Lime", 0.5 ) );
        g.add( new Book( "There", 2.5, 100 ) );
        System.out.println( g );
        System.out.println( g.totalPages() + " total pages\n" );
        g.increasePrices( 0.1 );
        System.out.println( g );
        g.remove( new Book( "Hey", 399, 1 ) );
        System.out.println( g );
        g.add( new Fruit( "Lime", 0.5 ) );
        g.add( new Fruit( "Lime", 0.5 ) );
        System.out.println( g );
        g.remove( new Fruit( "Lime", 0.5 ) );
        System.out.println( g );
    }
}
```

```
Inventory:
"Hey", 25 pages, $2.0
Lime $0.5
"There", 100 pages, $2.5

125 total pages

Inventory:
"Hey", 25 pages, $2.2
Lime $0.55
"There", 100 pages, $2.75

Inventory:
Lime $0.55
"There", 100 pages, $2.75

Inventory:
Lime   $0.5
Lime $0.55
"There", 100 pages, $2.75
Lime $0.5

Inventory:
Lime $0.55
"There", 100 pages, $2.75
Lime $0.5
```

Program 4. Different classes can implement the List interface. The following program runs a comparison between two different implementations: the ArrayList and LinkedList classes. Copy and run the following program. Be patient, it may take a minute or so to complete.

```
import java.util.*;

public class RunSpeedTests{
    public static void main( String [] args ){
        List<Integer> x = new ArrayList<Integer>();
        List<Integer> y = new LinkedList<Integer>();

        long time = addToFront( x );
        System.out.println("Adding to Front of an ArrayList: " + time + " ms" );
        time = addToFront( y );
        System.out.println("Adding to Front of a LinkedList: " + time + " ms" );

        time = accessElement( x, x.size()/2 );
        System.out.println("Accessing an element of an ArrayList: " + time + " ms" );
        time = accessElement( y, y.size()/2 );
        System.out.println("Accessing an element of a LinkedList: " + time + " ms" );
    }

    private static long addToFront( List<Integer> list ){
        long start_time = System.currentTimeMillis();
        for ( int n = 0; n < 200000; n++ ){
            int num = (int)(100*Math.random());
            list.add( 0, num );
        }
        long stop_time = System.currentTimeMillis();
        return stop_time - start_time;
    }

    private static long accessElement( List<Integer> list, int index ){
        long start_time = System.currentTimeMillis();
        for ( int n = 0; n < 100000; n++ ){
            int num = list.get( index );
        }
        long stop_time = System.currentTimeMillis();
        return stop_time - start_time;
    }
}
```

Calculates the time in ms to add 200,000 numbers to the front of a list.

Calculates the time in ms to access an element at a particular index 100,000 times

	On school computer	On my home computer*
Adding to Front of an ArrayList:		
Adding to Front of a LinkedList:		
Accessing an element of an ArrayList:		
Accessing an element of a LinkedList:		

* My times were 10, 60, 9150, and 43724 but not in that order.

Access times for LinkedList can vary greatly depending on where the element is. We will talk a little about this in class.

The point of program 4 is emphasize the idea that an interface allows us to specify what something should do without specifying how it should do it. The List interface describes what a linear list should be able to do. The ArrayList and LinkedList classes take very different approaches to implementing the List interface. In some situations an ArrayList is much faster; in other situations a LinkedList is the more efficient implementation.

Program 5. Create a file and complete this program.

```
import java.util.*;

public class Runner{
        public static void main(String [] args) {
                String [] words = { "once", "upon", "a", "time", null };
                List<String> myList = fillList( words );
                for ( int n = 0; n < myList.size(); n++ )
                        System.out.print( myList.get(n) + ", " );
                System.out.println();
```

This should print:

```
once, upon, a, time, null,
```

```
                List<String> list2 = new ArrayList<String>();
                list2.add( "Apples" );
                list2.add( "are" );
                list2.add( null );
                list2.add( "red" );
                String [] phrases = fillArray( list2 );
                for ( int n = 0; n < phrases.length; n++ )
                        System.out.print( phrases[n]  + ", " );
                System.out.println();
```

This should print:

```
Appless, ares, s, reds,
```

```
                ArrayList<String> list3 = new ArrayList<String>();
                list3.add( null );
                list3.add(  null  );
                list3.add( "ok" );
                list3.add(  null  );
                list3.add( null );
                System.out.println( "Size of list3: " + list3.size() );
                removeNulls( list3 );
                System.out.println( "Size of list3: " + list3.size() );
        }
```

This should print:

```
Size of list3: 5
Size of list3: 1
```

```
        public static List<String> fillList( String [] s ){
                Returns a List that is a copy of the data in the array.  Use an ArrayList.
        }

        public static String[] fillArray( List<String> list ){
                Returns an array of Strings that consists of the original strings except that an “s”
has been added to the end of each string. If an element is null, then replace it with the letter “s”
        }

        public static void removeNulls( List<String> s ){
                Remove all null values (and only null values) from the list
        }
}
```