Code Handout to accompany final exam CSCI 455 [S13 Bono]

`Digits` class interface (from problem 1):

```
public class Digits {

   // create Digits version of num
   // PRE: num >= 0
   public Digits(int num) { . . . }

   // gets digit i of the number, such that digit 1 is the most significant
   //     digit, and digit numDigits() is the least significant digit
   // PRE: 1 <= i <= numDigits()
   public int getDigit(int i) { . . . }

   // the number of digits in the number
   public int numDigits() { . . . }

   // returns the integer as a whole
   public int getInt() { . . . }

   // . . .

}
```

`ArrayList<ElmtType>` class (selected methods)

```
ArrayList<Integer> arrList = new ArrayList<Integer>();
                                    // create empty arraylist that can hold ints
arrList.add(3);                     // add a value to the end of the arrList
int num = arrList.get(i);           // returns element in in arrList
int howMany = arrList.size());      // number of elements in arrList
boolean empty = arrList.isEmpty();  // true iff arrList has no elements
```

`Collections` class (selected methods)

The `Collections` class contains static methods that operate on collections. Note: `ArrayList` and `LinkedList` both implement the `List` interface used below.

`static void sort(List<ElmtType> list)`
Sorts the list into ascending order according to the natural ordering of its elements (i.e., using `compareTo`).

`static void sort(List<ElmtType> list, Comparator<ElementType> c)`
Sorts the list according to the order specified by the comparator.

`Comparator<Type>` interface

An object that can compare two objects of type `Type`. Has one method defined by the interface:

`int compare(Type object1, Type object2)`
Must return a negative number if `object1` should come before `object2`, `0` if `object1` and `object2` are equal, or a positive number if `object1` should come after `object2`.

`Comparable<Type>` interface

An object that can be compared to another object of the same type. Has one method defined by the interface:

`int compareTo(Type other)`
`a.compareTo(b)` must return a negative number if `a` should come before `b`, `0` if `a` and `b` are equal, and a positive number otherwise

Some Java classes that are `Comparable`: `String`, `Integer`, `Double`.
The Java class `Point` is not `Comparable`

[More other side]

`Point` class (selected methods)

The `Point` class is a subclass of the abstract class `Point2D`.
Note: the x and y coordinates are type `int` in all of the following.

`new Point(x, y)`
Constructs point object with given x and y values.

`new Point(p2)`
Constructs point object that has the same value as point `p2`.

`p.translate(dx, dy)`
Changes `x` and `y` values of `p` by `dx` and `dy`, respectively. I.e., if `p` had coordinates `(x, y)`, it's new value is a point with coordinates `(x+dx, y+dy)`

`p.getX()`
Returns the `x` coordinate of `p`.

`p.getY()`
Returns the `y` coordinate of `p`.

`Point2D` class (selected methods)

Abstract superclass of `Point` class.

`p.distance(p2)`
Returns the distance (a `double`) from `Point2D p` to a specified `Point2D`.

`p.distance(double px, double py)`
Returns the distance (a `double`) from `Point2D p` to a specified point.

`p.distanceSq(p2)`
Returns the square of the distance (a `double`) from `Point2D p` to a specified `Point2D`.

`p.distanceSq(double px, double py)`
Returns the square of the distance (a `double`) from `Point2D p` to a specified point.

C++ `Node` type and `ListType` (this is the only part of the code handout with C++ code):

```
struct Node {
  int data;
  Node * next;
  Node() { data = 0; next = NULL; }
  Node(int d) { data = d; next = NULL; }
  Node(int d, Node * n) { data = d; next = n; }
};

typedef Node * ListType;
```

[More other side]