1. Sorting Algorithms :: Comparable Interface

The Java Class Library contains a **generic** interface named *java.lang.Comparable*<*T*> where *T* is any class of objects that can be compared to determine if:

1. One object *obj1* is less than another object *obj2*, or if
2. *obj1* is greater than *obj2*, or if
3. *obj1* is equal to *obj2*.

The interface declares one abstract method that must be implemented by classes that implement the interface:

```
int compareTo(T obj2)
```

The method is to return:

-1	If *this* object (the one on which the method is invoked) is less than *obj2*.
0	If *this* object is equal to *obj2*.
1	If *this* object is greater than *obj2*.

1. Sorting Algorithms :: Comparable Interface (continued)

For example suppose we declare a class *Coordinate* to represent an (x, y) coordinate in the Cartesian plane. Furthermore, we define a *Coordinate c1* to be less than another *Coordinate c2* if *c1* is closer to the origin than *c2*.

```
public class Coordinate implements Comparable<Coordinate> {
  private double mX;
  private double mY;
  private double calcDistanceToOrigin() {
    return Math.sqrt(mX * mX + mY * mY);
  }
  public int compareTo(Coordinate pCoord) {
    double thisDist = calcDistanceToOrigin();
    double coordDist = pCoord.calcDistanceToOrigin();
    if (thisDist < coordDist) {
      return -1;
    } else if (thisDist > coordDist) {
      return 1;
    } else {
      return 0;
    }
  }
}
```

1. Sorting Algorithms :: Comparable Interface (continued)

One of the primary uses of the *Comparable<T>* interface is to compare objects that are being sorted. Most sorting algorithms work by comparing elements to determine if they are "out of order". Therefore, for any class that we declare for which we intend to sort objects of that class, we need to implement the *Comparable<T>* interface.

Note that the *java.lang.String*, *java.lang.Integer*, and *java.lang.Double* classes each implement the *Comparable<String>*, *Comparable<Integer>*, and *Comparable<Double>* interfaces. For example,

```
String name1 = "Fred";
String name2 = "Betty";
String name3 = "fred";

int x = name1.compareTo(name2);  // x is assigned 1 since "Fred" > "Betty"
int y = name2.compareTo(name3);  // x is assigned -1 since "Betty" < "fred"
int z = name1.compareTo(name3);  // x is assigned -1 since "Fred" < "fred"
```