Widgets

# Resource classes typically have a standard structure. When you write a resource class from scratch, you should follow the standard Java conventions. In this lab, you will write a Widget class. Widget objects take two values and can be sorted by your sorting algorithms.

# Implementing an Interface

# If you recall, you wrote very similar overloaded methods in each of your sorts, one set for doubles and the other set for Comparables. If you are going to sort Widgets, you need to make Widget “isa” Comparable. Your first idea might be to make the Widget class extend Comparable. However, Java does not do it that way. Instead, Java wrote Comparable as an interface. In this lab you will make Widget *implement the Comparable interface* so that Widgets can be sorted. The Comparable interface has exactly one abstract method compareTo.

Widget

Object

Comparable

# Inheriting from Object

It does not seem useful to make Widget extend another class, except for java.lang.Object which all Java classes automatically extend. Of course, Widget then inherits all of Object's methods.

The API for Object lists several useful instance methods. In this lab

you will override both toString and equals. (Later in the year you will   
override hashCode in this same Widget class.)

An Object’s default toString returns the object's type, plus an @ sign, plus the object's hexadecimal address. For example, the call widget.toString()might return the string "Widget@6ea71b". When writing a resource class, you typically override the default toString method to return a string consisting of the object’s private data, nicely labeled.

w

aWidget

The Object’s default equals method compares two Objects' memory locations. Thus, w.equals(v) returns true if w and v both reference the same memory location as shown. Usually this is not what you want. It is usually more useful to override equals to refer to the object’s private data, as shown in a.equals(b) . An easy way to accomplish this is ensure that the result of equals agrees with the result of compareTo. If those two methods returned conflicting results, you could be creating a nasty bug.

a

b



Widget

5, 6

Widget

5, 6

v

**Java Constructors**

You should know that every Java constructor contains a hidden default super as its very first call. As you know, the purpose of that hidden default super is to initialize the superclass’ fields, if there are any.

As you know, constructors initialize the object’s private data. When you are designing a class, you may want to provide several useful constructors, including a default constructor, constructors with one, two, or more arguments, and a copy constructor. The Widget’s copy constructor creates a copy of the other Widget, with identical fields-with-values, when that other Widget is passed as an argument.

# Private

In Java, private means private at the class level, not at the object level. Thus, any Widget has direct access, without going through an accessor method, to any other Widget's private fields. The fields are private only to objects instantiated from another class. As a real world example, consider Student objects with a private ID number. Every Student object has direct access to every other Student's ID number. However, those private ID numbers cannot be accessed by other objects, such as Teacher objects.

**Assignment**



A Widget models something that is defined by two measurements called “cubits” and “hands.” Since the two measurements are incommensurable, there is no way to convert cubits to hands, or vice versa. There is no formula to use! You will have to deal with it.

We have described the features of a standard resource class above. You are to write a Widget class that has those features, namely, implements Comparable, has two private fields, has three constructors (default, two-arg, and copy), has accessors and modifiers for each field, and overrides both equals and compareTo (take care that they agree!) and a nicely-labeled toString (e.g. 64 cubits 13 hands ).

When writing compareTo, take advantage of the fact that each Widget object has direct access to the private data of every other Widget object. In other words, a Widget object has direct access to the private field other.myCubits. You don’t need to use the accessor method other.getCubits().

Widgets\_Driver has (almost) been written for you. You will have to figure out how to call each of your four sorts. The widgets.txt file is in the directory. Complete the Widget class. You will submit Widget.

**Input Sample Run**

widgets.txt

Default widget A: 0 cubits 0 hands  
2-arg constructor B: 23 cubits 10 hands  
Copy constructor C: 23 cubits 10 hands  
C's cubits = 23 and hands = 10  
Test the equals methods:   
 A equals B false  
 B equals C true  
C is reset to 3 cubits 4 hands  
  
Test each sort on 57 Widgets  
 1 Selection Sort  
 2 Insertion Sort  
 3 Merge Sort  
 4 Quick Sort  
Choose your sort: 2  
  
0 cubits 14 hands  
1 cubits 3 hands  
2 cubits 14 hands  
5 cubits 14 hands  
10 cubits 14 hands  
11 cubits 11 hands  
12 cubits 0 hands  
12 cubits 7 hands  
13 cubits 9 hands  
15 cubits 12 hands  
. . .

99 cubits 5 hands

64

13

15

12

67

3

10

14

32

1

5

14

<etc.>