CSSE 220 Day 23

Generic methods and Function Objects

Mini-project intro

Check Out FunctionObjectsAndSorting project from SVN

Questions

- Exam
- Anything else?

- Day 23 HW is due Monday
  - Finish function objects exercise
  - Sorting Exercise
  - Finish Vector Graphics
  - Team member evaluation survey
  - Prepare a 5-minute demo for Monday's class.

Apply this limit property to the following pairs of functions

1. N and N^2
2. $N^2 + 3N + 2$ and N^2
3. $N + \sin(N)$ and N
4. $\log N$ and N
5. $N \log N$ and N^2
6. N^a and b^N
7. a^N and b^N $(a < b)$
8. $\log_a N$ and $\log_b N$ $(a < b)$
9. $N!$ and N^N

Q9

Generic Methods

Generic methods: the need

- Consider the following methods:

```
 public static void main(String[] args) {
     String [] ss = {"abc", "def", "ghij"};
     Integer [] ii = {new Integer(5), new Integer(6)};
     print(ss);
     print(ii);
  }
  public static void print(String[] strings){
     for (String s: strings)
        System.out.println(s);
  }

  public static void print(Integer[] ints){
     for (Integer i: ints)
        System.out.println(i);
  }
}
```

This code is in today's repository

- Can we write **print** in a generic way so we do not have to have a separate method for each type of array?

Generic method: simple solution

```
public static <T> void print (T[] a){
    for (T obj: a)
        System.out.println(obj);
}
```

- The type variable <T> before the method's return type tells the compiler: T will be a generic type for this method. Substitute for it the actual type of the argument.
- This method can be called with any array of objects.
- For some other methods, we need to constrain the generic type used (next slide)

Generic method: type constraint

- Suppose want a generic method to take an array as its only argument, and return the smallest item in the array.
- This only makes sense if the base type of the array implements the **Comparable** interface.

```
public static <T extends Comparable> T min (T[] a) {
     T smallest = a[0];
     for (int i=1; i<a.length; i++)
        if (smallest.compareTo(a[i]) > 0)
           smallest = a[i];
     return smallest;
  }
```

- This works, but gives a warning
  - Type safety: The method compareTo(Object) belongs to the raw type Comparable. References to generic type Comparable<T> should be parameterized
- How to fix it?

Generic method: fix the warning

```
public static <T extends Comparable<T>> T min (T[] a) {
     T smallest = a[0];
     for (int i=1; i<a.length; i++)
        if (smallest.compareTo(a[i]) > 0)
           smallest = a[i];
     return smallest;
  }
```

- Note that in this context "extends" means either "extends" or "implements".
- But this could be too restrictive. Perhaps we want to be able to be able to compare elements of a subclass with elements of a superclass (as in the Shape hierarchy from a couple of weeks ago).

Generic method: more generally

```
public static <T extends Comparable<? super T>> T min (T[] a) {
     T smallest = a[0];
     for (int i=1; i<a.length; i++)
        if (smallest.compareTo(a[i]) > 0)
           smallest = a[i];
     return smallest;
  }
```

- The ? is a "wild card". **<? super T>** says we can compare to an element of any superclass of T.
- For more on wild cards (optional) see *Java Generics and Collections* at Safari Books on-line, or http://www.devarticles.com/c/a/Java/Wildcards-and-Generic-Methods-in-Java/

Intro to Function Objects

- Sort example from other languages

  The difficulty of doing the same thing in Java

Limitations of Comparable!

- How would we write `compareTo()` for a Rectangle class? What would be the basis for comparison?
- There is more than one natural way to compare Rectangles!
- What if I don't want to commit to any particular method?
- It would be nice to be able to create and pass comparison methods to other methods …

Function Objects (a.k.a. Functors)

- We'd like to be able to pass a method as an argument to another method. (what is the role of arguments to methods in general?)
  - This is not a new or unusual idea.
  - You pass other functions as arguments to Maple's *plot* and *solve* functions all of the time (on a later slide).
  - C and C++ provide *qsort*, whose first argument is a comparison function.
  - Scheme has a *sort* function, which can take a function as its first argument.

```
Chez Scheme Version 7.4
Copyright (c) 1985-2007 Cadence Research Systems
> (sort > '(7 3 9 -2 5 -6 0 4 1 -8))
(9 7 5 4 3 1 0 -2 -6 -8)
> (sort (lambda (x y) (< (abs x) (abs y)))
        '(7 3 9 -2 5 -6 0 4 1 -8))
(0 1 -2 3 4 5 -6 7 -8 9)
```

Q1

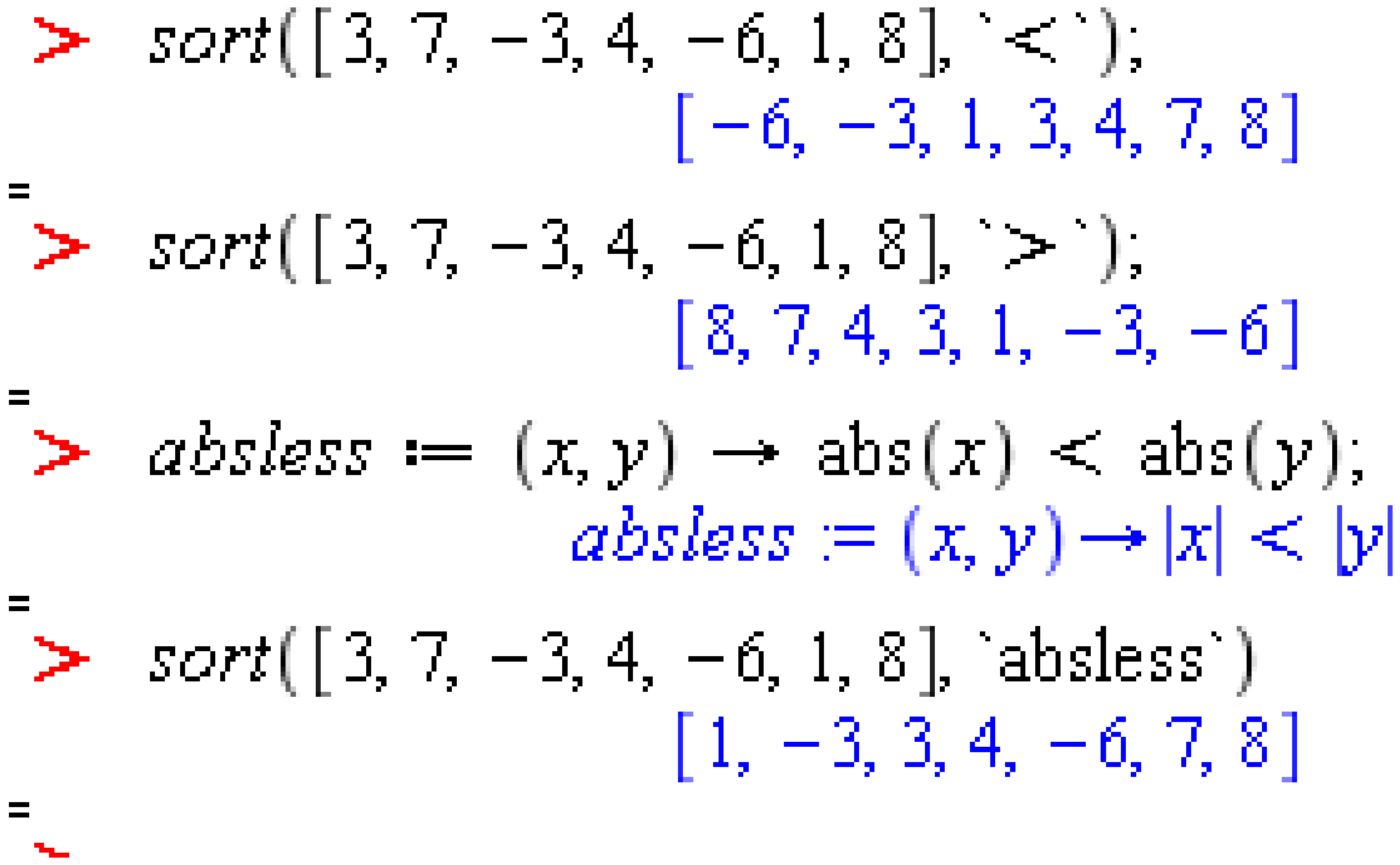
Similar example in Python

```
>>> list = [4, -2, 6, -1, 3, 5, -7]
>>> list.sort()
>>> list
[-7, -2, -1, 3, 4, 5, 6]
>>> def comp (a, b):
        return abs(a) - abs (b)

>>> list.sort(comp)
>>> list
[-1, -2, 3, 4, 5, 6, -7]
```

The **comp** function is passed as an argument to the **sort** method.

Similar example in Maple



```
> sort([3, 7, −3, 4, −6, 1, 8], `<`);
                                [−6, −3, 1, 3, 4, 7, 8]
=
> sort([3, 7, −3, 4, −6, 1, 8], `>`);
                                [8, 7, 4, 3, 1, −3, −6]
=
> absless := (x, y) → abs(x) < abs(y);
                         absless := (x, y) → |x| < |y|
=
> sort([3, 7, −3, 4, −6, 1, 8], `absless`)
                                [1, −3, 3, 4, −6, 7, 8]
=
```

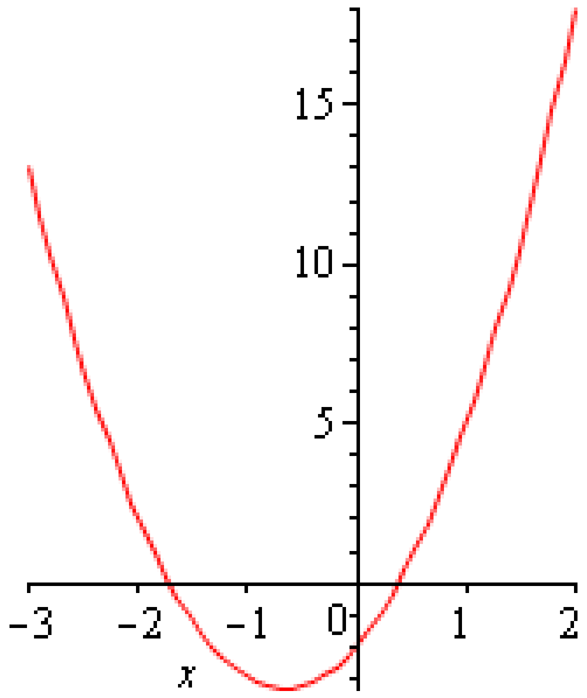
Q2

More Maple functions as parameters

```
> f := x->3*x^2 + 4*x - 2;
```

$$f := x \rightarrow 3\,x^2 + 4\,x - 2$$

```
=
> plot(f(x), x=-3..2);
```



```
=
> solve(f(x), x);
```

$$-\frac{2}{3}+\frac{\sqrt{10}}{3},\ -\frac{2}{3}-\frac{\sqrt{10}}{3}$$

Java Function Objects

- What's it all about?
  - Java (unlike Scheme, Maple, Python, C) does NOT allow methods to be passed as arguments.
  - We say that functions are first-class data in Scheme, Python, and Maple, but not in Java.
    - More about first-class data in CSSE 304.
  - But in Java, we can approximate "methods as parameters" by creating objects whose sole purpose is to provide a function for use by a method. They are called *function objects*, a.k.a. *functors*.

- The standard example:

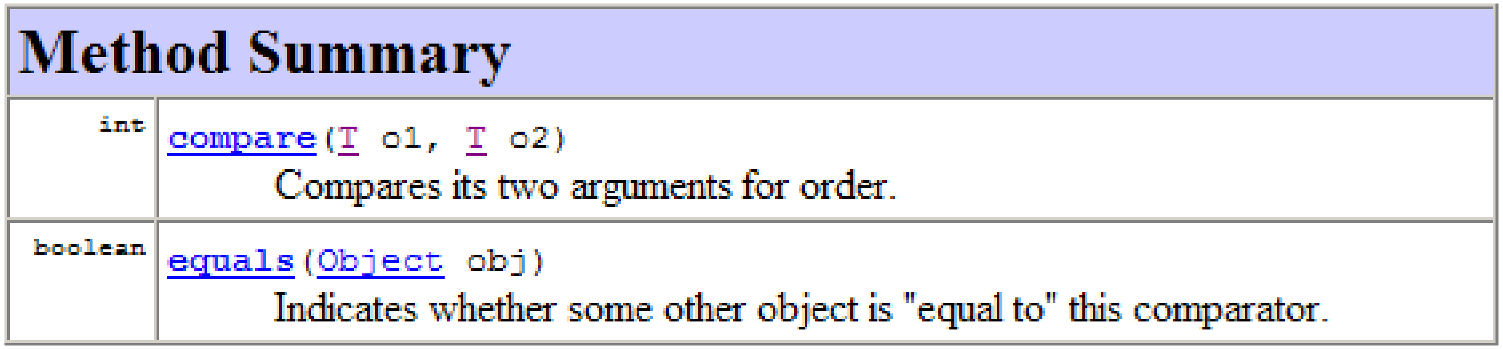
  java.util.Comparator

Function objects – summary so far

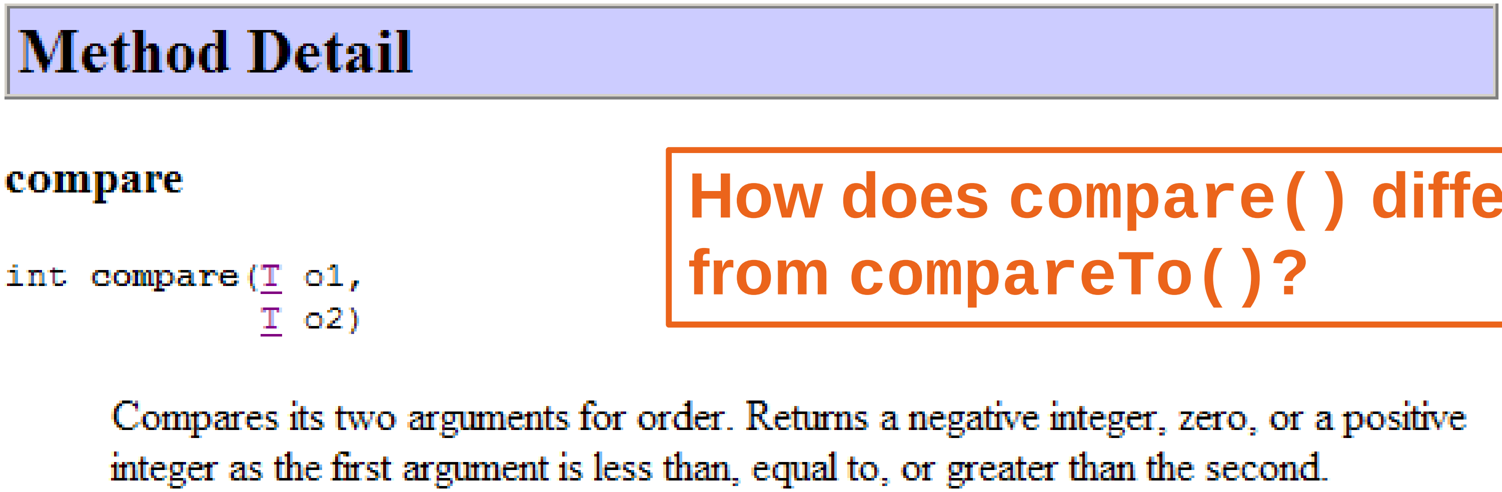
- The ability to pass functions as arguments to other functions can enable us to write code that is more flexible and generic
- Example that we examined in several different languages:
  - Pass a (built-in or user-defined) comparison function as one of the arguments to a sort function
- Unfortunately, Java (unlike C++) doesn't allow functions to be passed as arguments
- But we can create objects whose whole purpose is to pass a function into a method. They are called **function objects,** a.k.a. functors.
- For a (somewhat advanced, but worth skimming to get its flavor) overview of function objects in different languages:
  - http://en.wikipedia.org/wiki/Function_object
- Primary built-in Java example interface: **Comparator**

A built-in Function Object interface

- java.util.Comparator<T>



Method Summary	
`int`	`compare(T o1, T o2)` Compares its two arguments for order.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this comparator.



Method Detail

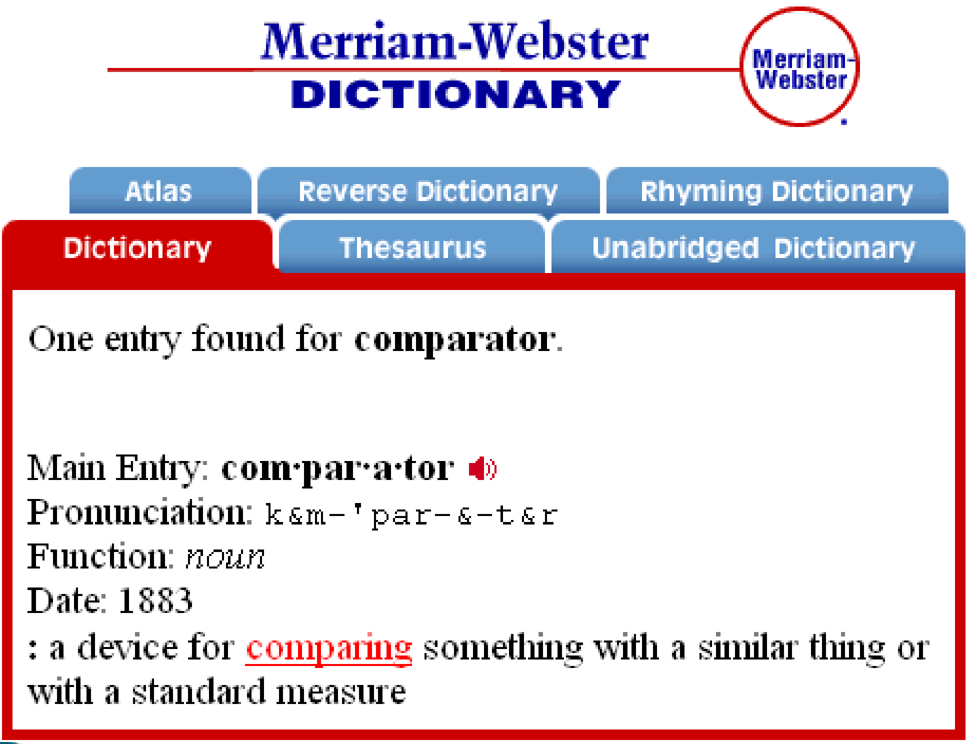
compare

```
int compare(T o1,
            T o2)
```

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

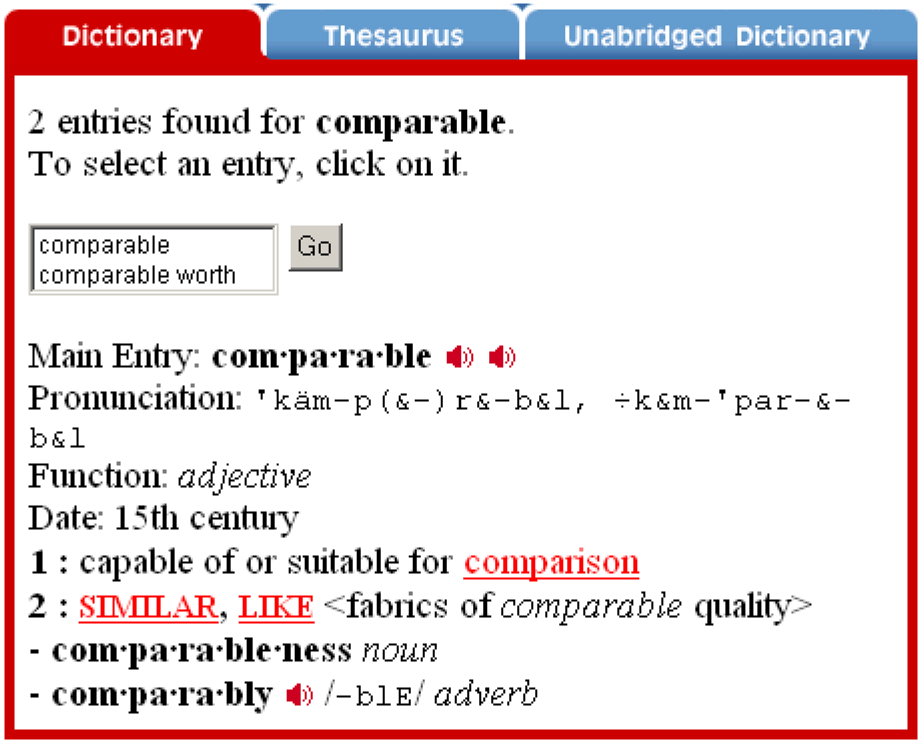
How does `compare()` differ from `compareTo()`?

How to pronounce Comparator, Comparable



One entry found for **comparator**.

Main Entry: **com·par·a·tor**
Pronunciation: k&m-'par-&-t&r
Function: *noun*
Date: 1883
: a device for comparing something with a similar thing or with a standard measure



2 entries found for **comparable**.
To select an entry, click on it.

Main Entry: **com·pa·ra·ble**
Pronunciation: 'käm-p(&-)r&-b&l, ÷k&m-'par-&-b&l
Function: *adjective*
Date: 15th century
1 : capable of or suitable for comparison
2 : SIMILAR, LIKE <fabrics of *comparable* quality>
- com·pa·ra·ble·ness *noun*
- com·pa·ra·bly /-blE/ *adverb*

Example: Rectangles

```
// Example class for use with  Comparators.
// by Mark Allen Weiss, modified by Claude Anderson

public class SimpleRectangle {
    private int length, width;

    public SimpleRectangle(int len, int wid) {
        length = len; width = wid;
    }

    public int getLength( ) { return length; }

    public int getWidth( ) { return width; }

    public String toString( ){
        return "Rectangle " +
        getLength( ) + " by " +
        getWidth( );
    }
}
```

The **SimpleRectangle** class does *not* implement **Comparable**, because there is not a single "natural" way to order **SimpleRectangle** objects.

Q3

FindMax Uses a Comparator object

```
public class CompareTest {
  public static <AnyType> AnyType findMax( AnyType [ ] a,
                                           Comparator<AnyType> cmp ) {
    int maxIndex = 0;
    for( int i = 1; i < a.length; i++ )
      if( cmp.compare( a[ i ], a[ maxIndex ] ) > 0 )
        maxIndex = i;
    return a[ maxIndex ];
    }

  public static void main( String [ ] args ) {
    SimpleRectangle [ ] rects = new SimpleRectangle[
    rects[ 0 ] = new SimpleRectangle( 1, 10 );
    rects[ 1 ] = new SimpleRectangle( 20, 1 );
    rects[ 2 ] = new SimpleRectangle( 4, 6 );
    rects[ 3 ] = new SimpleRectangle( 5, 5 );

    System.out.println( "MAX WIDTH: "
       + findMax( rects, new OrderRectByWidth( ) ) )
    System.out.println( "MAX AREA: "
       + findMax( rects, new OrderRectByArea( ) ) );
    }
}
```

vs. a[i].compareTo(a[maxIndex])

Note that **java.util.Collections.max** has the functionality of this **findMax** method.

Without something like Comparator, we would need separate `findMax` functions for finding the max using different comparison criteria

Construct Comparator objects, pass them to findMax

The Function Object Classes

```
class OrderRectByArea implements
                       Comparator<SimpleRectangle> {
  public int compare(SimpleRectangle r1,
                      SimpleRectangle r2){
     return r1.getWidth( ) * r1.getLength( )
           - r2.getWidth( ) * r2.getLength( );
    }
}

class OrderRectByWidth implements
                    Comparator<SimpleRectangle>{
   public int compare(SimpleRectangle r1,
                      SimpleRectangle r2){
      return( r1.getWidth() - r2.getWidth() );
    }
}
```

Two Comparator classes

Examples: Arrays and Collections

static `<T> int`	`binarySearch(T[] a, T key, Comparator<? super T> c)` Searches the specified array for the specified object using the binary search algorithm.

static `<T> void`	`sort(T[] a, Comparator<? super T> c)` Sorts the specified array of objects according to the order induced by the specified comparator.

static `<T> T`	`max(Collection<? extends T> coll, Comparator<? super T> comp)` Returns the maximum element of the given collection, according to the order induced by the specified comparator.

static `<T> void`	`sort(List<T> list, Comparator<? super T> c)` Sorts the specified list according to the order induced by the specified comparator.

Count Matches Exercise

- You can (and should) talk to your neighbors, the student assistants, and me, but you should submit your own work
- Starting code is in today's project
- It includes JUnit tests that you should get to run successfully.
- The second paramater of countMatches is a function object that returns a boolean value
- **EqualsZero** *and* **EqualsK** implement the **Matchable** interface
- Unit tests should help you discern the interface
- Analogy with our Rectangle example:
  - **countMatches** (corresponds to findMax in the example) is the method that takes an array and a function object as parameters
  - **EqualsZero** (corresponds to OrderRectsByWidth) is a specific "function object" class
  - **Matchable** (corresponds to Comparator) is the function object interface; you get to pick the name for its method.