CSSE 220 Day 13

Function Objects and Comparators

Work on Paint

Exam 1 Statistics

	Overall	Angel	Prog.
A	11	7	15
B	3	7	1
C	4	4	3
D	6	7	2
F	4	3	7
Average	79%	75%	81%
Median	79%	75%	90%

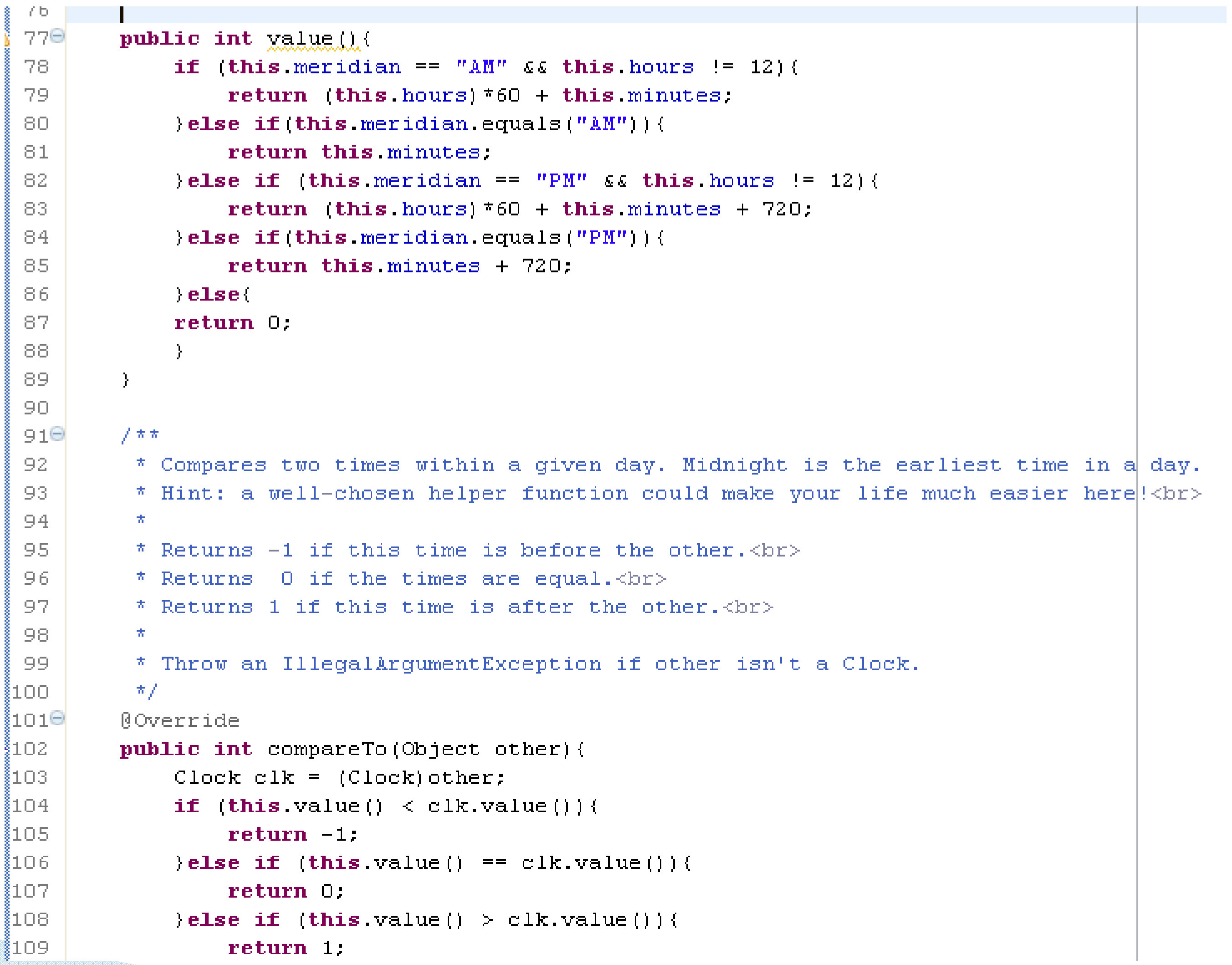
These are consistent with course averages so far… (82%)

Not trivial!

- Angel portion had some tricky questions
  - Review answers online and ask questions next class.
- Toggler
  - Required an array of buttons
  - With interactions between buttons
- Clock
  - tick() and compareTo() had lots of special cases!
- But insights help gain time…

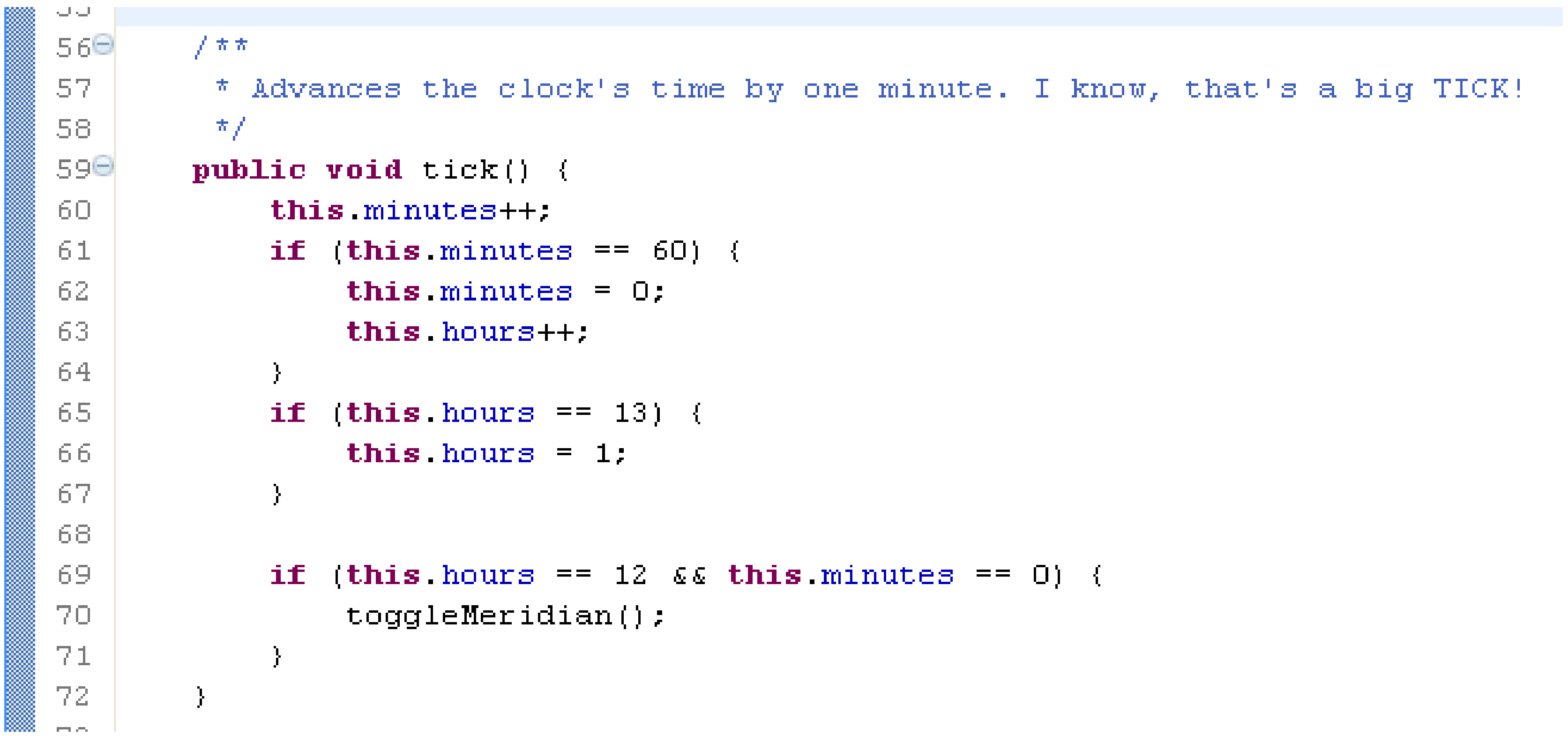
Value helper method calculates minutes from midnight

- Student solution



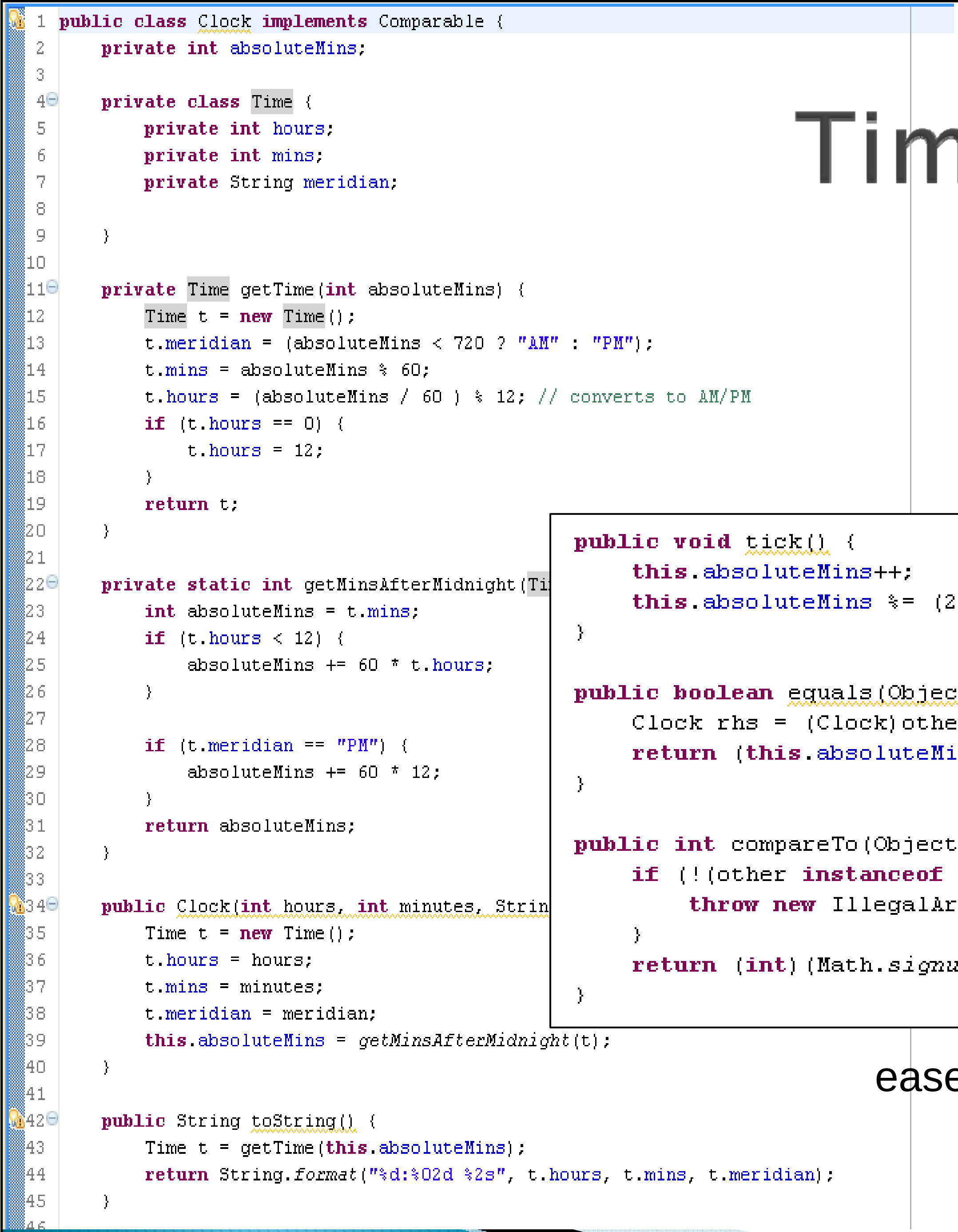
```
    public int value(){
        if (this.meridian == "AM" && this.hours != 12){
            return (this.hours)*60 + this.minutes;
        }else if(this.meridian.equals("AM")){
            return this.minutes;
        }else if (this.meridian == "PM" && this.hours != 12){
            return (this.hours)*60 + this.minutes + 720;
        }else if(this.meridian.equals("PM")){
            return this.minutes + 720;
        }else{
        return 0;
        }
    }

    /**
     * Compares two times within a given day. Midnight is the earliest time in a day.
     * Hint: a well-chosen helper function could make your life much easier here!<br>
     *
     * Returns -1 if this time is before the other.<br>
     * Returns  0 if the times are equal.<br>
     * Returns 1 if this time is after the other.<br>
     *
     * Throw an IllegalArgumentException if other isn't a Clock.
     */
    @Override
    public int compareTo(Object other){
        Clock clk = (Clock)other;
        if (this.value() < clk.value()){
            return -1;
        }else if (this.value() == clk.value()){
            return 0;
        }else if (this.value() > clk.value()){
            return 1;
```

My tick()

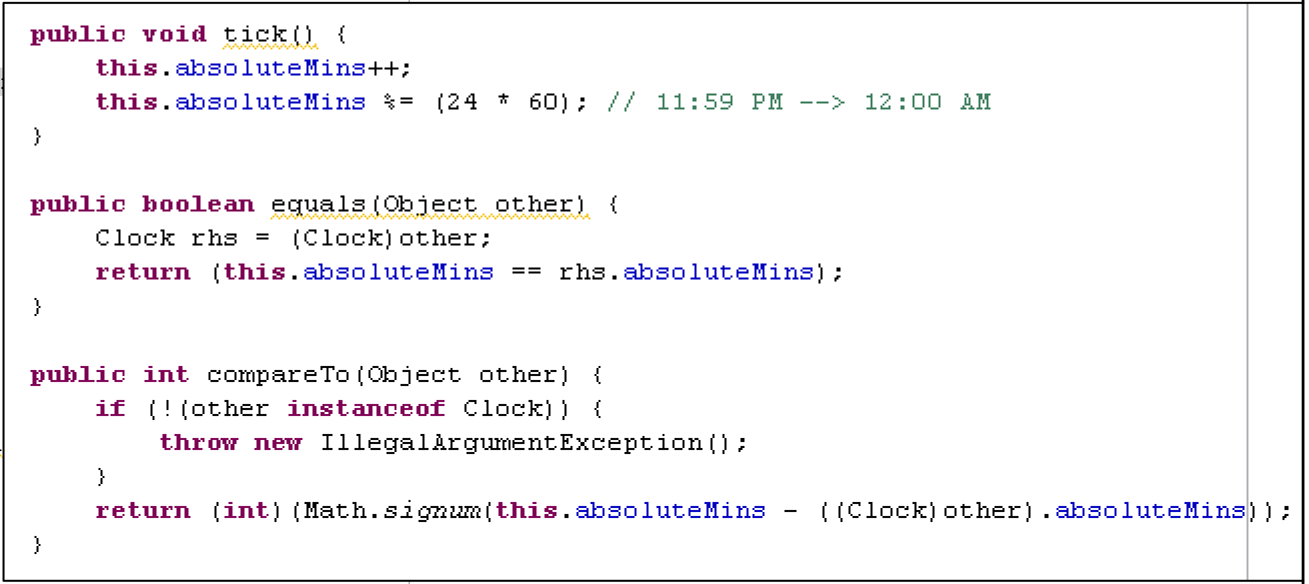


```
    /**
     * Advances the clock's time by one minute. I know, that's a big TICK!
     */
    public void tick() {
        this.minutes++;
        if (this.minutes == 60) {
            this.minutes = 0;
            this.hours++;
        }
        if (this.hours == 13) {
            this.hours = 1;
        }

        if (this.hours == 12 && this.minutes == 0) {
            toggleMeridian();
        }
    }
```

Time helper class



```
public class Clock implements Comparable {
    private int absoluteMins;

    private class Time {
        private int hours;
        private int mins;
        private String meridian;

    }

    private Time getTime(int absoluteMins) {
        Time t = new Time();
        t.meridian = (absoluteMins < 720 ? "AM" : "PM");
        t.mins = absoluteMins % 60;
        t.hours = (absoluteMins / 60 ) % 12; // converts to AM/PM
        if (t.hours == 0) {
            t.hours = 12;
        }
        return t;
    }

    private static int getMinsAfterMidnight(Ti
        int absoluteMins = t.mins;
        if (t.hours < 12) {
            absoluteMins += 60 * t.hours;
        }

        if (t.meridian == "PM") {
            absoluteMins += 60 * 12;
        }
        return absoluteMins;
    }

    public Clock(int hours, int minutes, Strin
        Time t = new Time();
        t.hours = hours;
        t.mins = minutes;
        t.meridian = meridian;
        this.absoluteMins = getMinsAfterMidnight(t);
    }

    public String toString() {
        Time t = getTime(this.absoluteMins);
        return String.format("%d:%02d %2s", t.hours, t.mins, t.meridian);
    }
```



```
public void tick() {
     this.absoluteMins++;
     this.absoluteMins %= (24 * 60); // 11:59 PM --> 12:00 AM
}

public boolean equals(Object other) {
     Clock rhs = (Clock)other;
     return (this.absoluteMins == rhs.absoluteMins);
}

public int compareTo(Object other) {
     if (!(other instanceof Clock)) {
          throw new IllegalArgumentException();
     }
     return (int)(Math.signum(this.absoluteMins - ((Clock)other).absoluteMins));
}
```

eases the work later!

Class this week

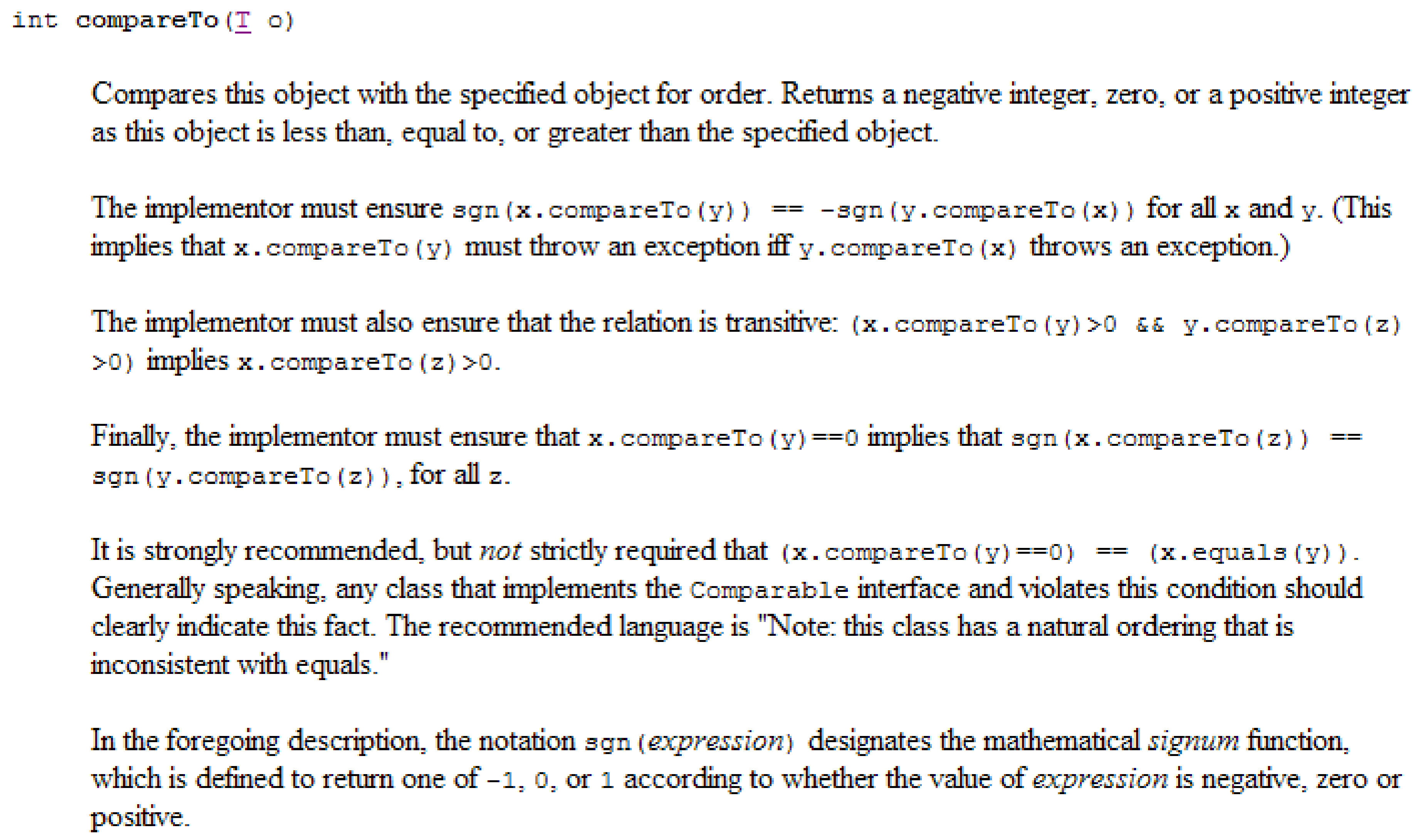
- Each class day this week.
  - Some time on new course content (function objects and algorithm analysis)
  - Some time to work on Paint (typically ~30 minutes).
- A progress report is due at the end of each class.
  - Easiest thing to do is keep your IEP updated, showing your progress on the phases that you outlined.
  - Commit it to your Paint repository.

Announcements

- By now, everyone should know how to submit files to SVN repositories.
  - I have been rather lenient in the past if you didn't get it submitted correctly. By now you should be able to submit it to the right place on time.
- Another way to earn "bug-fixing" bonus points is to suggest changes to specs that are unclear.
- BallWorlds, BigRational back to you this week

- Today: Function Objects and Comparators
- Questions?

compareTo: the fine print



`int compareTo(T o)`

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive.

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.

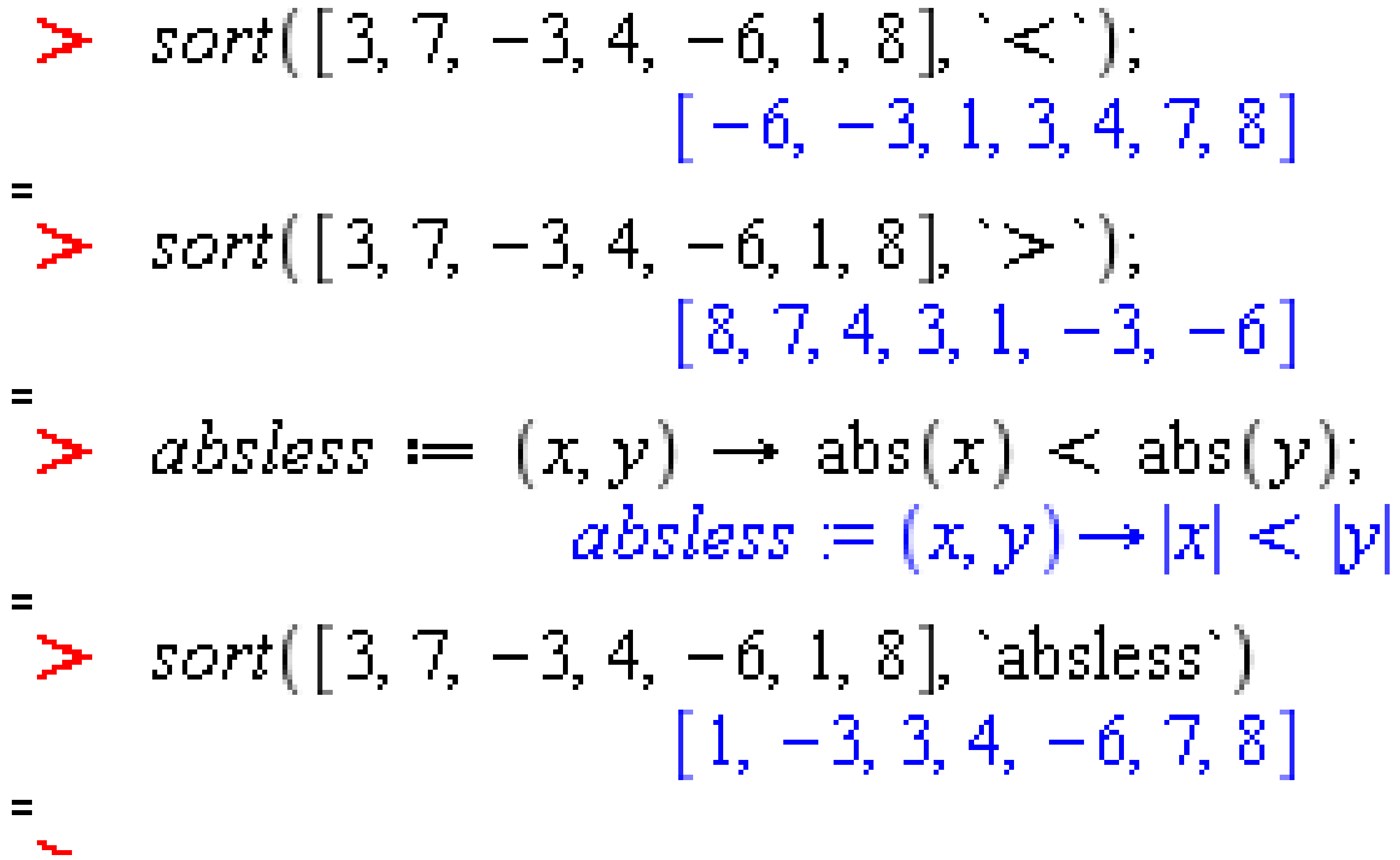
```
Chez Scheme Version 7.4
Copyright (c) 1985-2007 Cadence Research Systems
> (sort > '(7 3 9 -2 5 -6 0 4 1 -8))
(9 7 5 4 3 1 0 -2 -6 -8)
> (sort (lambda (x y) (< (abs x) (abs y)))
        '(7 3 9 -2 5 -6 0 4 1 -8))
(0 1 -2 3 4 5 -6 7 -8 9)
```

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



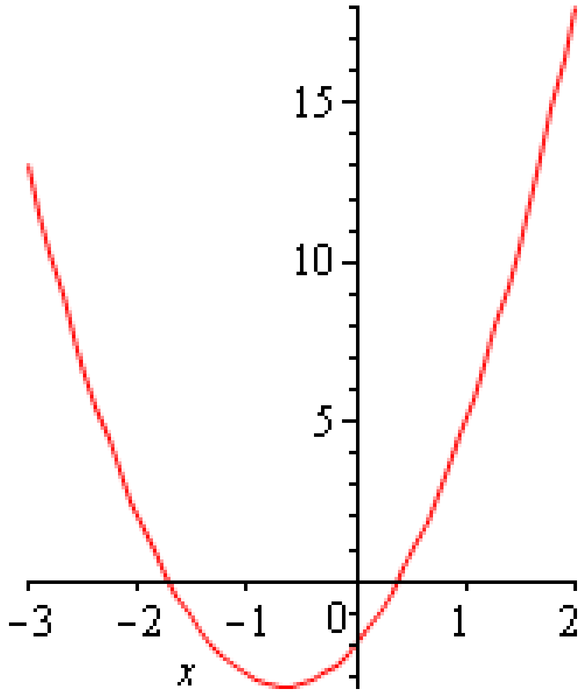
```
> sort([3, 7, -3, 4, -6, 1, 8], `<`);
                                        [-6, -3, 1, 3, 4, 7, 8]
=
> sort([3, 7, -3, 4, -6, 1, 8], `>`);
                                        [8, 7, 4, 3, 1, -3, -6]
=
> absless := (x, y) -> abs(x) < abs(y);
                                    absless := (x,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

More Maple functions as parameters

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

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

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



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

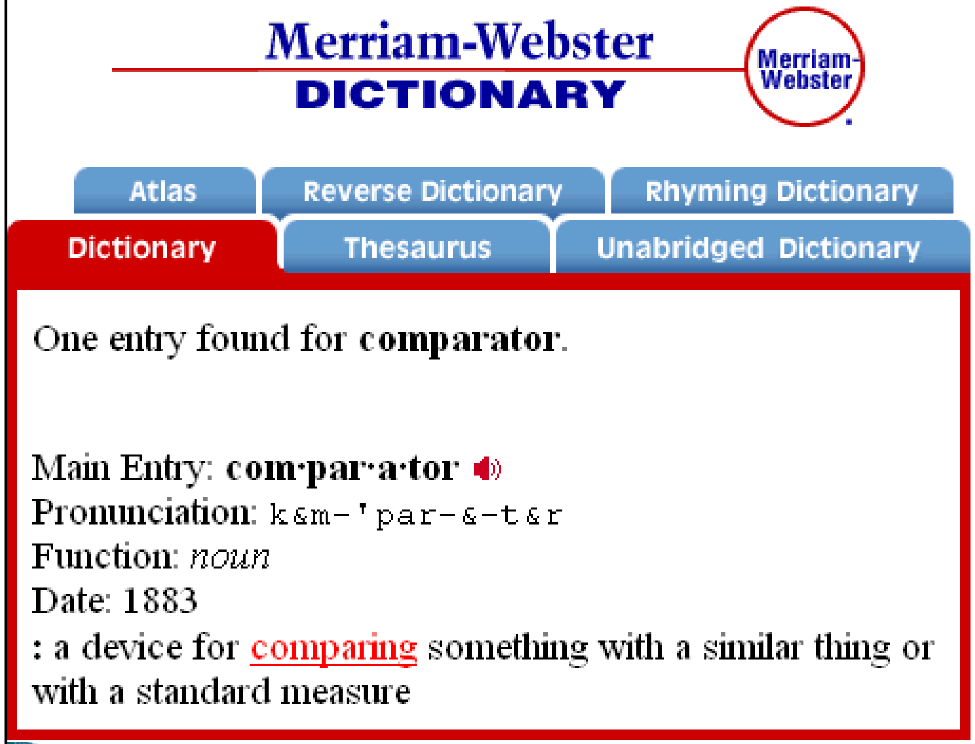
$$-\frac{2}{3} + \frac{\sqrt{10}}{3},\ -\frac{2}{3} - \frac{\sqrt{10}}{3}$$

Java Function Objects

- What's it all about?
  - 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* or *functionoids*.

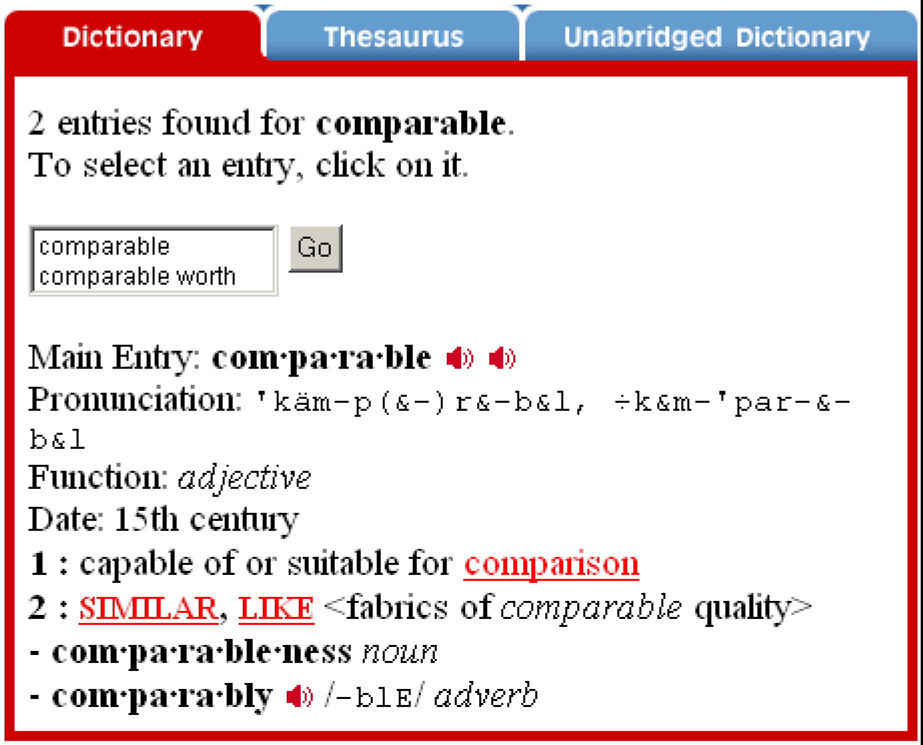
- Weiss DS book's example:
  - Uses **Comparator** objects (interface is defined in `java.util.Comparator`).
  - What is **Comparator** used for?
  - Why not just use **Comparable**?
    - OrderRectByWidth, SimpleRect, CompareTest

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*

Comparator Interface

Install this.

See HW 15

```
package weiss.util;   // It's in java.util also.

import java.io.Serializable;

/**
 * Comparator function object interface.
 */
public interface Comparator extends Serializable
{
    /**
     * Return the result of comparing lhs and rhs.
     * @param lhs first object.
     * @param rhs second object.
     * @return < 0 if lhs is less than rhs,
     *           0 if lhs is equal to rhs,
     *         > 0 if lhs is greater than rhs.
     * @throws ClassCastException if objects
     *      cannot be compared.
     */
    int compare( Object lhs, Object rhs )
                 throws ClassCastException;
}
```

Weiss provides code for several classes that are equivalent to those in **java.util,** so we can see how parts of the **java.util** classes might be implemented.

Generics would make this code slightly more complicated; we'll most likely deal with that later.

Example: Rectangles

```
public class SimpleRectangle {
    public SimpleRectangle( int l, int w ) {
      length = l; width = w;
    }

    public int getLength( ) {
      return length;
    }

    public int getWidth( ) {
      return width;
    }

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

    private int length;
    private int width;
}
```

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

FindMax Uses a Comparator object

```
public class CompareTest
{
    public static Object findMax( Object [ ] a, Comparator 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 )
    {
        Object [ ] rects = new Object[ 4 ];
        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.

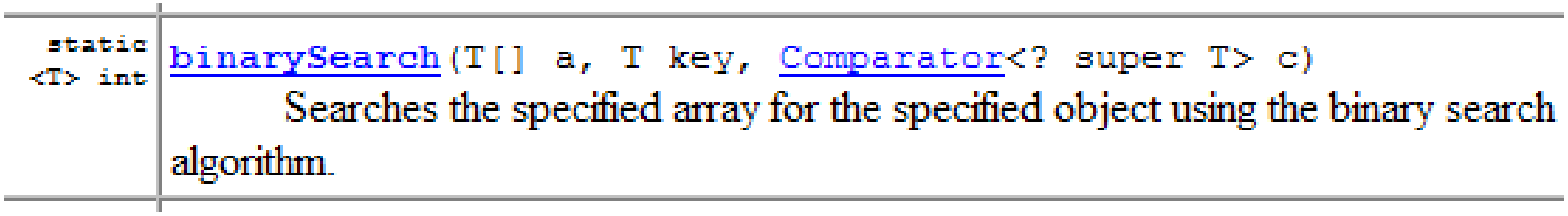
Without something like Comparators, we would need separate `findMax` functions for finding the max using different comparison criteria

The Actual Function Objects

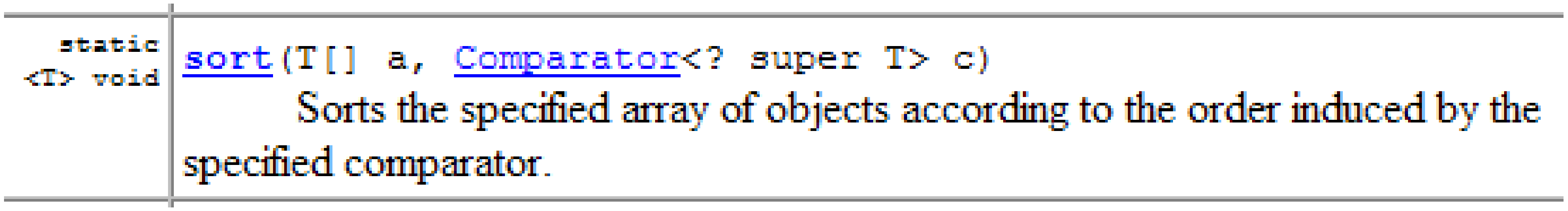
```
class OrderRectByArea implements Comparator
{
    public int compare( Object obj1, Object obj2 )
    {
        SimpleRectangle r1 = (SimpleRectangle) obj1;
        SimpleRectangle r2 = (SimpleRectangle) obj2;

        return( r1.getWidth()*r1.getLength() -
                r2.getWidth()*r2.getLength() );
    }
}

class OrderRectByWidth implements Comparator
{
    public int compare( Object obj1, Object obj2 )
    {
        SimpleRectangle r1 = (SimpleRectangle) obj1;
        SimpleRectangle r2 = (SimpleRectangle) obj2;

        return( r1.getWidth() - r2.getWidth() );
    }
}
```

Two Comparator classes.

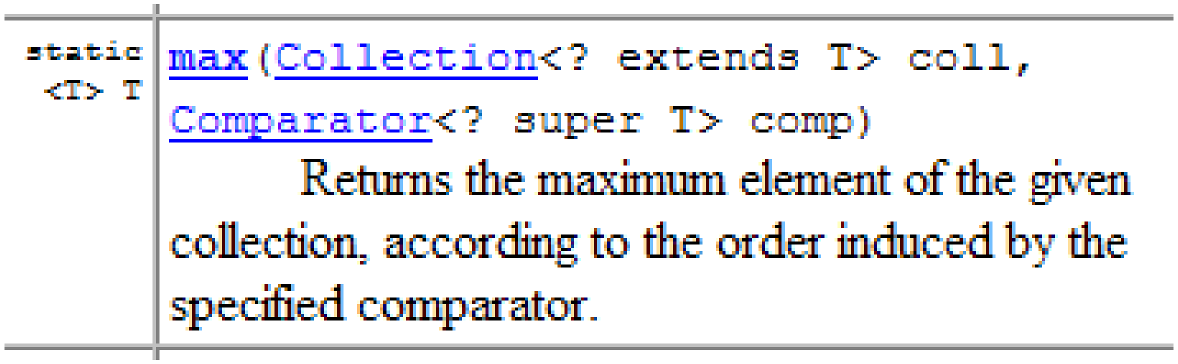
Examples: Arrays and Collections



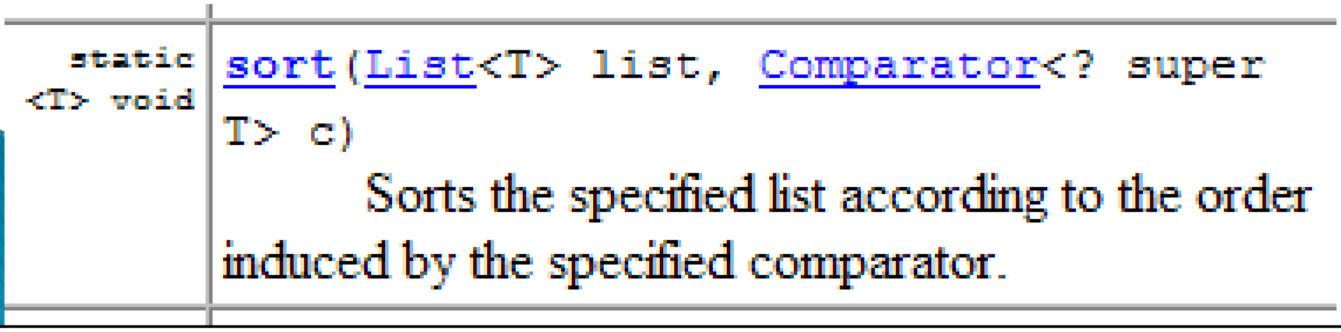
`static <T> int`	`binarySearch(T[] a, T key, Comparator<? super T> c)` Searches the specified array for the specified object using the binary search algorithm.



`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.

In-class Assignment

- You can (and should) talk to your neighbors, the student assistants, and me, but you should submit your own work.
- Starting code is in your individual **SVN** repository.
  Project name: Weiss4_29and4_30
- It includes JUnit tests that you should get to run successfully.
- Weiss problems 4.29, 4.30 (statements are on a very small handout).
- **EqualsZero** (problem 29c) should implement the interface from problem 29a. I called that interface **Matchable** and its method **test**
- 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.

Interlude

- Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
  --Martin Golding

Paint

- Discuss UML, spec

Paint

- Careful with opening spreadsheets in Eclipse
- Paint bucket (fill) moved to optional section.
- 75% minimum for basic stuff applies to functionality only
  - You’ll also get points for design (UML and IEP), code style, and documentation
  - So final grade can vary depending on these documents
- Hints: JColorChooser, Stroke object.
- See Java Swing Tutorial on Sun’s site.

Work on Paint

- Don't forget to commit your progress report (IEP) to the repository before the end of class.