

CSSE 230 Day 4

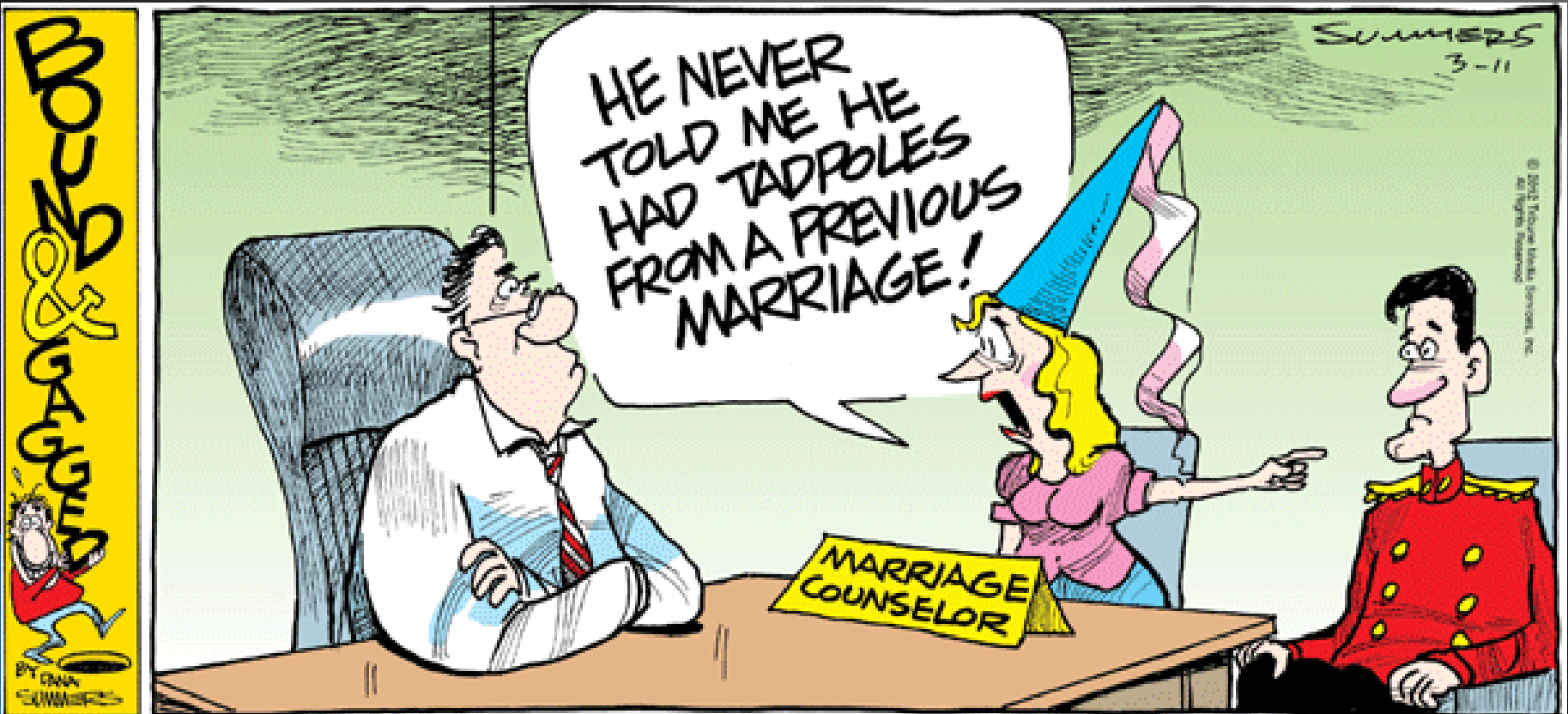
Data Structures grand tour continues
Diagnostic Quiz Review

Comparable, Comparator, and Function Objects

Check out from SVN: [DiagQuizReview](#)

Questions?

Written Assignment 2 Pascal Christmas Tree



Data Structures Grand Tour Continues

- » Some review
- Some new
- All will appear again

Array Lists and Linked Lists

Operations Provided	Array List Efficiency	Linked List Efficiency
Random access	$O(1)$	$O(n)$
Add/remove item	$O(n)$	$O(1)$

Stack

- ▶ A last-in, first-out (LIFO) data structure

- ▶ Real-world stacks

- Plate dispensers in the cafeteria
- Pancakes!

- ▶ Some uses:

- Tracking paths through a maze
- Providing “unlimited undo” in an application

```
public static void printInReverse(List<String> words) {  
    // TODO: implement  
    Stack<String> stack = new Stack<String>();  
    for (String w : words) {  
        stack.push(w);  
    }  
    while (!stack.isEmpty()) {  
        System.out.println(stack.pop());  
    }  
}
```

Operations Provided	Efficiency
Push item	O(1)
Pop item	O(1)

Implemented by
Stack, *LinkedList*,
and *ArrayDeque* in
Java

Queue

► first-in, first-out (FIFO)

data structure

► Real-world queues

- Waiting line at the BMV
- Character on Star Trek TNG

► Some uses:

- Scheduling access to shared resource (e.g., printer)

```
/**
 * Uses a queue to print pairs of words consisting of
 * a word in the input and the word that appeared five
 * words before it.
 *
 * @param words
 */
public static void printCurrentAndPreceding(List<String> words) {
    // TODO: implement
    ArrayDeque<String> queue = new ArrayDeque<String>();
    // Preloads the queue:
    for (int i = 0; i < 5; i++) {
        queue.add("NotAWord");
    }
    for (String w : words) {
        queue.add(w);
        String fiveAgo = queue.remove();
        System.out.println(w + ", " + fiveAgo);
    }
}
```

Operations Provided	Efficiency
Enqueue item	O(1)
Dequeue item	O(1)

Implemented by
LinkedList and
ArrayDeque in
Java

Set

- ▶ A collection of items **without duplicates** (in general, order does not matter)
 - If **a** and **b** are both in set, then ***!a.equals(b)***
- ▶ Real-world sets:
 - Students
 - Collectibles
- ▶ One possible use:
 - Quickly checking if an item is in a collection

```
public static void printSortedWords(List<String> words) {  
    TreeSet<String> ts = new TreeSet<String>();  
    for (String w : words) {  
        ts.add(w);  
    }  
    for (String s : ts) {  
        System.out.println(s);  
    }  
}
```

Example from 220

Operations	HashSet	TreeSet
Add/remove item	O(1)	O(lg n)
Contains?	O(1)	O(lg n)

Can hog space

Sorts items!

Q2-5

Map

How is a TreeMap like a TreeSet?
How is it different?

- ▶ Associate **keys** with **values**
- ▶ Real-world “maps”
 - Dictionary
 - Phone book
- ▶ Some uses:
 - Associating student ID with transcript
 - Associating name with high scores

Operations	HashMap	TreeMap
Insert key-value pair	$O(1)$	$O(\lg n)$
Look up the value associated with a given key	$O(1)$	$O(\lg n)$

Can hog space

Sorts items by key!

HashMap/HashSet Example (220)

```
public static void printWordCountsByLength(List<String> words) {
    HashMap<Integer, HashSet<String>> map =
        new HashMap<Integer, HashSet<String>>();

    for (String w : words) {
        int len = w.length();
        HashSet<String> set;
        if (map.containsKey(len)) {
            set = map.get(len);
        } else {
            set = new HashSet<String>();
            map.put(len, set);
        }
        set.add(w);
    }
    System.out.printf("%d unique words of length 3.%n", getCount(map, 3));
    System.out.printf("%d unique words of length 7.%n", getCount(map, 7));
    System.out.printf("%d unique words of length 9.%n", getCount(map, 9));
    System.out.printf("%d unique words of length 15.%n", getCount(map, 15));
}
```

```
public static int getCount(HashMap<Integer, HashSet<String>> map, int key) {
    if (map.containsKey(key)) {
        return map.get(key).size();
    } else {
        return 0;
    }
}
```

Priority Queue

Not like regular queues!

- ▶ Each **item** stored **has an** associated **priority**
 - Only item with “minimum” priority is accessible
 - Operations: *insert*, *findMin*, *deleteMin*
- ▶ Real-world “priority queue”:
 - Airport ticketing counter
- ▶ Some uses
 - Simulations
 - Scheduling in an OS
 - Huffman coding

```
PriorityQueue<String> stringQueue =  
    new PriorityQueue<String>();
```

```
stringQueue.add("ab");  
stringQueue.add("abcd");  
stringQueue.add("abc");  
stringQueue.add("a");
```

```
while(stringQueue.size() > 0)  
    System.out.println(stringQueue.remove());
```

Operations Provided	Efficiency
Insert	$O(\log n)$
Find Min	$O(\log n)$
Delete Min	$O(\log n)$

The version in Warm Up and Stretching isn't this efficient.

Trees, Not Just For Sorting

- ▶ Collection of nodes
 - One specialized node is the root.
 - A node has one parent (unless it is the root)
 - A node has zero or more children.
- ▶ Real-world “trees”:
 - Organizational hierarchies
 - Some family trees
- ▶ Some uses:
 - Directory structure on a hard drive
 - Sorted collections

Only if tree is
“balanced”

Operations Provided	Efficiency
Find	$O(\log n)$
Add/remove	$O(\log n)$


Graphs

- ▶ A collection of nodes and edges
 - Each edge joins two nodes
 - Edges can be directed or undirected
- ▶ Real-world “graph”:
 - Road map
- ▶ Some uses:
 - Tracking links between web pages
 - Facebook

Operations Provided	Efficiency
Find	$O(n)$
Add/remove	$O(1)$ or $O(n)$ or $O(n^2)$

Depends on
implementation
(time/space trade off)

Networks

- ▶ Graph whose edges have numeric labels
 - ▶ Examples (labels):
 - Road map (mileage)
 - Airline's flight map (flying time)
 - Plumbing system (gallons per minute)
 - Computer network (bits/second)
 - ▶ Famous problems:
 - Shortest path
 - Maximum flow
 - Minimal spanning tree
 - Traveling salesman
 - Four-coloring problem for planar graphs
- 

Common ADTs

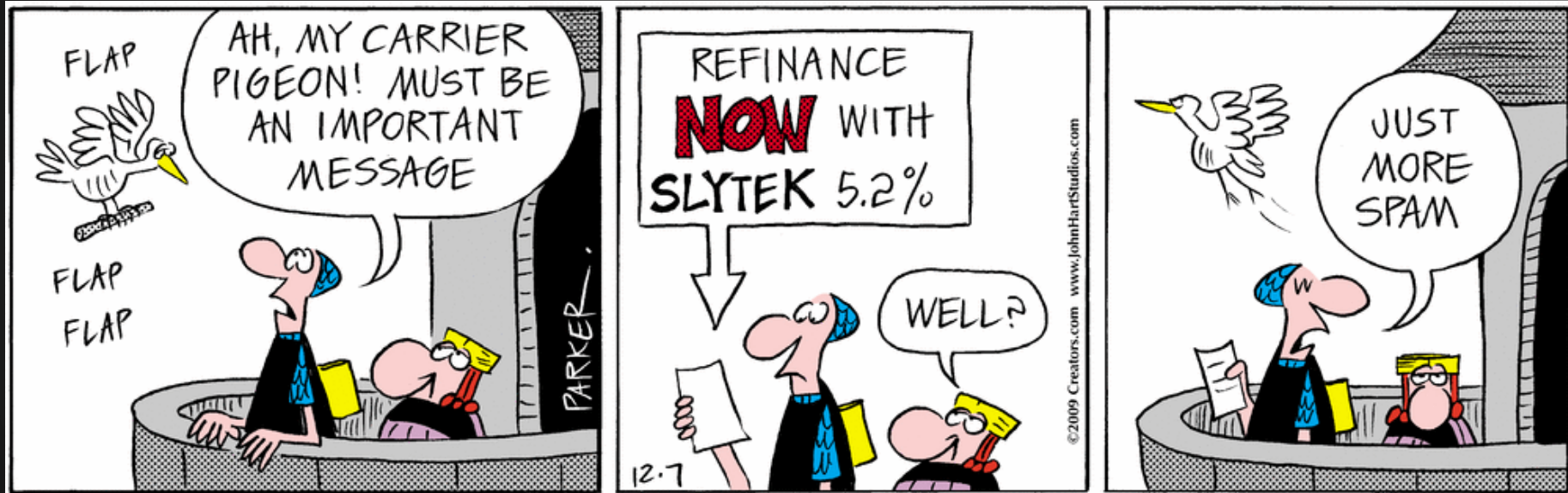
- ▶ Array
- ▶ List
 - Array List
 - Linked List
- ▶ Stack
- ▶ Queue
- ▶ Set
 - Tree Set
 - Hash Set
- ▶ Map
 - Tree Map
 - Hash Map
- ▶ Priority Queue
- ▶ Tree
- ▶ Graph
- ▶ Network

We'll implement and use nearly all of these, some multiple ways. And a few other data structures.

Data Structure Summary

Structure	find	insert/remove	Comments
Array	$O(n)$	can't do it	Constant-time access by position
Stack	top only $O(1)$	top only $O(1)$	Easy to implement as an array.
Queue	front only $O(1)$	$O(1)$	insert rear, remove front.
ArrayList	$O(\log N)$	$O(N)$	Constant-time access by position
Linked List	$O(n)$	$O(1)$	$O(N)$ to find insertion position.
HashSet/Map	$O(1)$	$O(1)$	If table not very full
TreeSet/Map	$O(\log N)$	$O(\log N)$	Kept in sorted order
PriorityQueue	$O(\log N)$	$O(\log N)$	Can only find/remove smallest
Tree	$O(\log N)$	$O(\log N)$	If tree is balanced
Graph	$O(N*M) ?$	$O(M)?$	N nodes, M edges
Network			shortest path, maxFlow

Diagnostic Quiz Review



Some questions that many students missed

Expression questions

- ▶ Give a **very simple** Java expression that is equivalent to:

`!(x && !x)`

BTW: Never write something like
`if (a.isVisible() == true)`

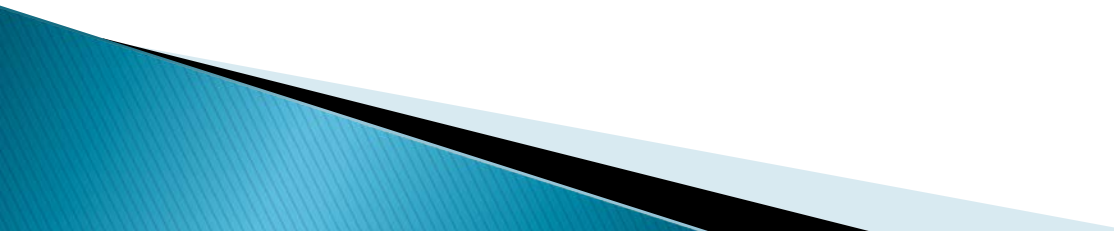
- ▶ What are the values of each of the following expressions, if `x==5` and `y==7` ?

`x + ' ' + y`

`x + " " + y`

`x + y + " "`

Simple big-Oh questions

- ▶ What is the worst-case Big-Oh running time of an unsuccessful sequential search of an unordered array that contains N elements?
 - ▶ What is the worst-case Big-Oh running time of an unsuccessful binary search of an array that contains N
 - ▶ What is the Big-Oh running time of merge sort of an array that contains N elements?
- 

Method Selection

Overloading vs. Overriding

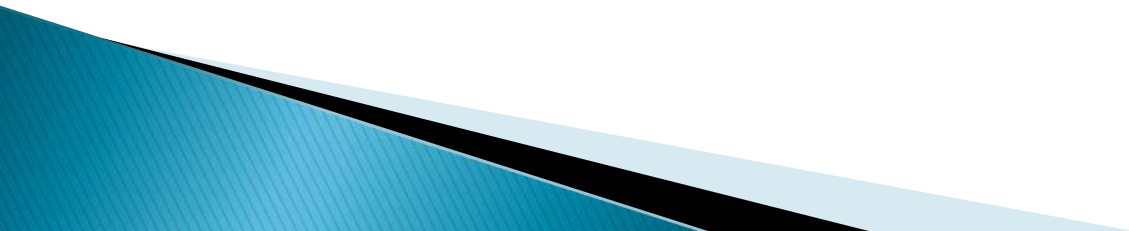
- ▶ In Eclipse, open:
examples.StaticParmsDemo
- ▶ from the *DiagQuizReview* project
- ▶ This is based on Figure 4.45, page 166 of Weiss.
- ▶ Section 4.9 begins:
 - "A common myth is that all methods and all parameters are bound at runtime. This is not true."
 - Methods that are static, final, or private.

Note that all of the code from the Weiss book is available on the course web site. You can run it, modify it, and experiment.

Interlude

- ▶ Computer Science is no more about computers than astronomy is about _____.

Donald Knuth



Interlude

- ▶ Computer Science is no more about computers than astronomy is about telescopes.

Donald Knuth

Aliasing

- ▶ How many objects are created in this code?

```
MyNumber a = new MyNumber();  
a.setNum(5);  
MyNumber b = new MyNumber();  
b.setNum(6);  
MyNumber c = a;  
System.out.println(c);
```

- ▶ What is “aliasing”?

Default Constructors

- ▶ What does Java do if no constructor is declared for a class?
 - How can we instantiate the class?
 - What values do the fields get?

```
class Jambalaya {  
    int beans;  
    double rice;  
    Insect crayfish;
```

```
    public String toString() {  
        return beans + " " + rice + " " + crayfish;  
    }  
}
```

Parameter Passing

- ▶ this code is available In Eclipse, open *examples.WhatIsX*

```
public static void main(String[] args) {  
    int x = 0;  
    f(x);  
    System.out.println(x);  
}
```

```
private static void f(int x) {  
    /*  
     * TODO: Without adding printing, can you change the body of  
     * this method to get this program to print:  
     *     3?  
     *     3.5?  
     *     Anything else?  
     */  
}
```


More Big-Oh Practice

```
for (int i = 0; i < n; i++)  
    for (int j = 0; j < i; j++)  
        sum++;
```

34% of students answered $N \log N$.
Where could the log come from?

```
for (int i = 0; i < n; i++)  
    for (int j = 0; j < n * n; j++)  
        for (int k = 0; k < j; k++)  
            sum++;
```

```
for (int i = 1; i < n; i = i * 2)  
    sum++;
```

One more distinction

- ▶ *throw* versus *throws*
 - Part of exception handling
 - Signal an error with: *throw new ExceptionType()*
 - Abdicate responsibility with:
void myMethod() throws ExceptionType {
...}

Function Objects and Generics

» Comparable and Comparator

Comparable review:

- ▶ *interface java.lang.Comparable<T>*
- ▶ Type Parameter: *T* – the type of objects that this object may be compared to
- ▶ *int compareTo(T o)*
 - Compares *this* with *o* 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
 - Primitive type comparison: *x < y*
 - Comparable comparison: *obj1.compareTo(obj) < 0*

Limitations of Comparable!

- ▶ There is more than one natural way to compare Rectangles!
- ▶ What if we want to compare using
 - Height?
 - Width?
 - Closeness of aspect ratio to the golden ratio, ϕ
- ▶ It would be nice to be able to create and pass comparison methods to other methods ...

$$\phi = \frac{a+b}{a} = \frac{a}{b} = \frac{1+\sqrt{5}}{2}$$

Function Objects (a.k.a. Functors)

- ▶ Why do methods have arguments in the first place?
- ▶ We'd like to be able to pass a method as an argument to another method
- ▶ **This is not a new or unusual idea.**
 - You pass other functions as arguments to Maple's *plot* and *solve* functions (on a later slide).
 - C and C++ provide *qsort*, whose first argument is a comparison function.
 - Scheme and Python also have *sort* functions that can take a comparison function as an argument.

In Scheme

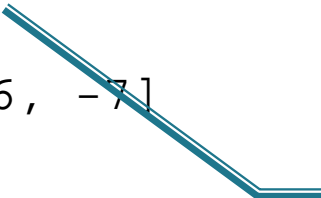
- ▶ Scheme has a sort function that takes a function as an argument:

```
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]  
=
```

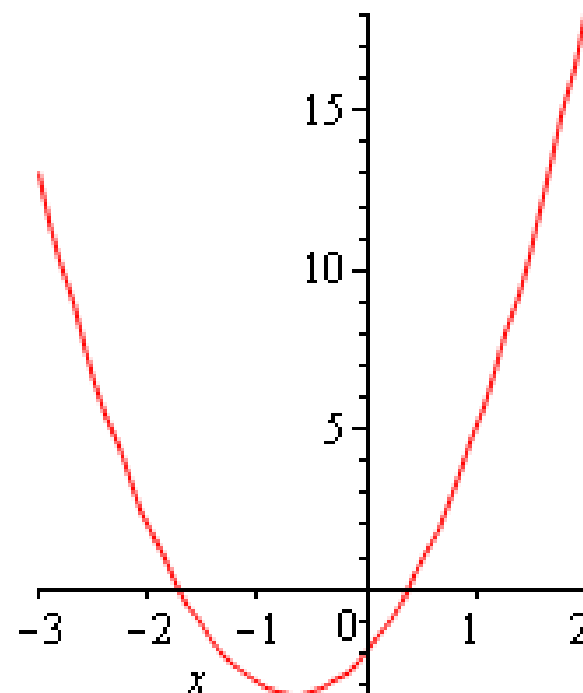
More Maple

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

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

=

```
> 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 doesn't (yet) allow passing functions as arguments.
 - So, we create objects whose sole purpose is to pass a function into a method
 - Called **function objects**
 - a.k.a. functors, functionoids, more fun than a barrel of monkeys
- ▶ Prime function object example: *Comparator*

You say "tomato", I say "toe-mah-toe"

Merriam-Webster
DICTIONARY



Atlas

Reverse Dictionary

Rhyming Dictionary

Dictionary

Thesaurus

Unabridged Dictionary

One entry found for **comparator**.

Main Entry: **com·par·a·tor** 🔊

Pronunciation: kəm-ˈpar-ə-tər

Function: *noun*

Date: 1883

: a device for comparing something with a similar thing or with a standard measure

Java: "imposed" ordering

Dictionary

Thesaurus

Unabridged Dictionary

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

comparable
comparable worth

Go

Main Entry: **com·para·ble** 🔊 🔊

Pronunciation: ˈkäm-p(ə)rə-bəl, ðkəm-ˈpar-ə-bəl

Function: *adjective*

Date: 15th century

1 : capable of or suitable for comparison

2 : SIMILAR, LIKE <fabrics of *comparable* quality>

- **com·para·ble·ness** *noun*

- **com·para·bly** 🔊 /-bəl/ *adverb*

"natural" ordering

Sorting Arrays and Collections

- ▶ *java.util.Arrays* and *java.util.Collections* are your friends!
- ▶ On Written Assignment 2
 - The **CountMatches** implementation problem asks you to write code that creates and uses function objects.

**Add an anonymous
Comparator to main().**