


# Welcome to CSSE 220


- ▶ Please do not sit in the back row
  - ▶ Please sit:
    - Sit on the right side or as close to the front on the left side of the room as you can.
  - ▶ We are excited that you are here:
    - Start your computer and get ready for our first class session.
- 

# Course Introduction, Starting with Java

CSSE 220—Object-Oriented Software Development  
Rose-Hulman Institute of Technology



# Agenda

- ▶ Roll Call
  - ▶ A few administrative details
  - ▶ Verify Eclipse and Subclipse configuration
  - ▶ Java vs. Python and C
  - ▶ A first Java program (calculate factorials)
- 

# Daily Quizzes

- ▶ I expect you to answer every question.
- ▶ Stop me if I don't cover a question!


# Roll Call, Introductions

- ▶ Tell me what you prefer to be called
- ▶ For introductions give:
  - Name
  - Major
  - Hometown
  - Past programming experience


# A Tour of the On-line Course Materials

- ▶ ANGEL
- ▶ Syllabus
- ▶ Schedule

# Programming is not a spectator sport


- ▶ And neither is this course
  - ▶ Ask, evaluate, respond, comment!
  - ▶ Is it better to ask a question and risk revealing your ignorance, or to remain silent and perpetuate your ignorance?
- 

# Feel free to interrupt during class discussions

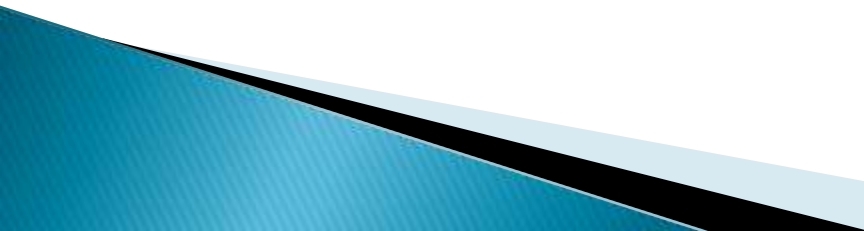
- ▶ Even with statements like, *“I have no idea what you were just talking about.”*
  - ▶ We want to be polite, but in this room learning trumps politeness.
  - ▶ I do not intend for classroom discussions to go over your head. Don't let them!
- 



# Things Java Has in Common with Python

- ▶ Classes and objects
  - ▶ Lists (but no special language syntax for them like Python)
  - ▶ Standard ways of doing graphics, GUIs.
  - ▶ A huge library of classes/functions that make many tasks easier.
  - ▶ A nicer Eclipse interface than C has.
- 

# Things Java Has in Common with C

- ▶ Primitive types: **int, char, long, float, double**
  - ▶ Static typing
  - ▶ Similar syntax and semantics for **if, for, while, break, ...**
  - ▶ Semicolons
  - ▶ Execution begins with **main()**
  - ▶ Comments: `//` and `/* ... */`
  - ▶ Arrays are *homogeneous*, and size must be declared at creation.
- 

# Why Java?

- ▶ Widely used in industry for large projects
  - From cell phones
    - including smart phones—Android platform
  - To global medical records
- ▶ Object-oriented (unlike C)
- ▶ “Statically type safe” (unlike Python, C, C++)
- ▶ Less complex than C++
- ▶ Part of a strong foundation
- ▶ Most popular language according to the TIOBE Programming Community Index [Feb 2011]

# Let's Get Started!

- ▶ Hopefully you have:
  - Java
  - Eclipse 3.5 (make sure you have this version!)
  - Subclipse
- ▶ **Go to Homework 1 and do: step 4, then step 5a–d.**
- ▶ This will:
  - Configure Eclipse
  - Create a Workspace for your Java projects
  - Set up your SVN repository in Eclipse
  - Check out today's SVN HW1 project
- ▶ **Figure out how to run HelloPrinter.java**

**Get help if you're stuck!**

THEN WE PROGRAM  
THE WEB SITE USING A  
FAST GUY IN TIGHTS  
AND A MOVIE ABOUT  
COFFEE.



www.dilbert.com scottadams@aol.com

CORRECT  
ME IF I'M  
WRONG.



WE USE  
FLASH  
AND  
JAVA-  
SCRIPT



11-15-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.

I SAID,  
"IF"!!!



# Checkout project for today

- ▶ Go to SVN Repository view, at bottom of the workbench
  - If it is not there,  
Window → Show View → Other → SVN  
→ SVN Repositories
- ▶ Browse SVN Repository view for **HW1** project
- ▶ Right-click it, and choose **Checkout**
  - Accept default options
- ▶ Expand the **HW1** project that appears in Package Explorer (on the left-hand-side)

# HelloPrinter.java

- ▶ To run a Java program:
  - Right-click it in the Package Explorer view
  - Choose **Run As → Java Application**
- ▶ Change the program to say hello to a person next to you
- ▶ Introduce an error in the program
  - See if you can come up with a different error than the person next to you
- ▶ Fix the error that the person next to you introduced

# A First Java Program

In Java, all variable and function definitions are inside *class* definitions

main is where we start

```
public class HelloPrinter {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

**System.out** is Java's standard output stream. This is the variable called **out** in the **System** class.

**System.out** is an *object* from the **PrintStream** class. **PrintStream** has a *method* called **println()**.



# A Second Java Program

Define a constant, MAX

```
public class Factorial {
    public static final int MAX = 17;

    public static int factorial(int n) {
        int product;

        product = 1;
        for (int i = 2; i <= n; i++) {
            product = product * i;
        }

        return product;
    }

    public static void main(String[] args) {
        for (int i = 0; i <= Factorial.MAX; i++) {
            System.out.print(i);
            System.out.print("! = ");
            System.out.println(factorial(i));
        }
    }
}
```

Except for **public static** and the declaration of the loop counter *inside* the **for** header, everything about this function definition is identical to C.

This *class* is called *Factorial*. It has one *field* called MAX and two *methods*: *factorial* and *main*.

`println` (below) terminates the output line after printing; `print` doesn't.

Make a new class (File ~ New ~ Class) called *Factorial* (check the box to let Eclipse type *main* for you). Enter & run the Factorial code. What happens when  $i = 14$ ? Why?

# Javadoc comments

```
/**
 * Has a static method for computing n!
 * (n factorial) and a main method that
 * computes n! for n up to Factorial.MAX.
 *
 * @author Claude Anderson et al.
 */
public class Factorial {
    /**
     * Biggest factorial to compute.
     */
    public static final int MAX = 17;

    /**
     * Computes n! for the given n.
     *
     * @param n
     * @return n! for the given n.
     */
    public static int factorial (int n) {
        ...
    }

    ...
}
}
```

We left out something important on the previous slide – comments!

Java provides Javadoc comments (they begin with `/**`) for both:

- Internal documentation for when someone reads the code itself
- External documentation for when someone re-uses the code

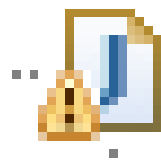
Comment your own code now, as indicated by this example. Don't forget the `@author` tag in `HelloPrinter`.

# Writing Javadocs

- ▶ Written in special comments: `/** ... */`
- ▶ Can come before:
  - Class declarations
  - Field declarations
  - Constructor declarations
  - Method declarations
- ▶ Eclipse is your friend!
  - It will generate Javadoc comments automatically
  - It will notice when you start typing a Javadoc comment

# In all your code:

- ▶ Write appropriate comments:
  - Javadoc comments for public fields and methods.
  - Explanations of anything else that is not obvious.
- ▶ Give self-documenting variable and method names:
  - Use name completion in Eclipse, Ctrl-Space, to keep typing cost low and readability high
- ▶ Use Ctrl-Shift-F in Eclipse to format your code.
- ▶ Take care of all auto-generated TODO's.
  - **Then delete the TODO comment.**
- ▶ Correct ALL compiler warnings. Quick Fix is your friend!



# Homework Due Next Session

»» HW1, linked from schedule