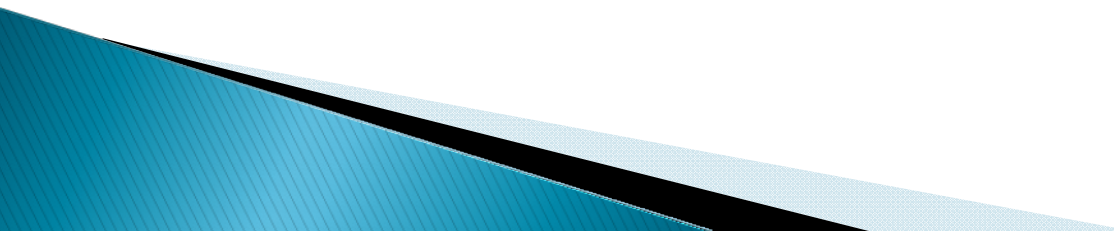


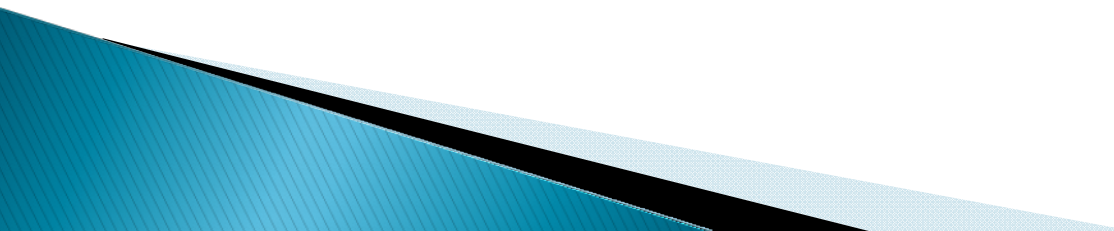
# CSSE 220 Day 1

Brief Course Intro  
Instructor Intro  
Java Intro

# Agenda

- ▶ Roll Call
  - ▶ A few administrative details(more next time)
  - ▶ Java vs. Python and C
  - ▶ A first Java program (calculate factorials)
  - ▶ Many factorial variations
  - ▶ File input/output
  - ▶ Primitive types, switch, ? :
  - ▶ A look at the homework
- 

# Daily Quizzes

- ▶ We will have them most days.
  - ▶ Help you interact with the lecture material.
  - ▶ Answers should be in the PowerPoint slides and class discussion.
  - ▶ When I return them, they should be notes for you.
  - ▶ A way for you to give me feedback, ask questions, or let me discover that we need more class time on a topic.
- 

# Roll Call

- ▶ If I mispronounce your name, or if you want to be called by another name than what the Registrar gave me, please tell me
- ▶ While I am calling the roll, register your attendance on ANGEL.
  - I'll write the PIN on the board.
  - Do this every class day.

# Contacting me

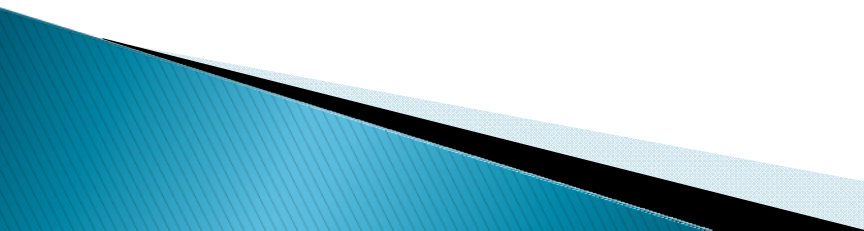
- ▶ Office (F-210)– I am there a lot, when I am not in meetings.
  - My Outlook Calendar is public, and linked from the Syllabus.
  - Sometimes in F217 helping students.
- ▶ Phone – x8331
- ▶ Email: [anderson@rose-hulman.edu](mailto:anderson@rose-hulman.edu)
- ▶ [csse220-staff@rose-hulman.edu](mailto:csse220-staff@rose-hulman.edu) will go to me and the student assistants.
- ▶ I want to help you (really!)
  - and I also hired a small army of students to help.

# Email subject lines

- ▶ When you send me email:
  - Please **include 220** somewhere in your subject line
  - **And also** include a real subject line!
  - **Bad:** When's MineSweeper due?
  - **Bad:** CSSE 220
  - **Good:** CSSE 220: When's MineSweeper due?

Clifton, Curtis C	Re: csse220-examples repository	Sun 9/2/2007 10:1...	9 KB
Boutell, Matthew R	Evening Lab Hours	Sun 9/2/2007 9:15 PM	17 KB
Clifton, Curtis C	Re: Angel Quiz for HW2 posted	Sun 9/2/2007 5:4...	10 ...
clifton@rose-hulman.edu	CSSE120: r92 - in trunk: Public/Slides SlidesPPT	Sun 9/2/2007 5:4...	11 ...
Clifton, Curtis C	Angel Quiz for HW2 posted	Sun 9/2/2007 5:2...	7 KB

# Student Assistant Lab hours

- ▶ Place and Times
    - F-217
    - Sunday–Thursday 7–9 PM.
    - MTWR class periods 9 and 10 (and 7–8, we hope).
  - ▶ It's a great place to go to get help with software setup, programming problems, or course concepts.
  - ▶ Some students go there to work on their homework. In addition to the lab assistant, you may find other students to talk with.
- 

# A quick tour of the online course materials

- ▶ More materials will be added.
- ▶ More details will be filled in.
- ▶ This is a new version of the course!
- ▶ The posted schedule for the course is preliminary and ambitious.
  - There will be some adjustments as we go along.
- ▶ I will usually post my PowerPoint slides **after** each class meeting.
  - If I ever forget, feel free to remind me.




# Programming is not a spectator sport

And neither is this course.

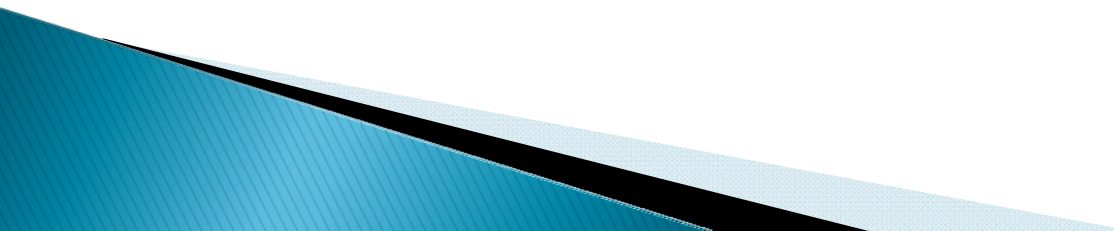
Ask, evaluate, respond, comment!

Is it better to ask a question  
and risk revealing your  
ignorance, or to remain silent  
and perpetuate your ignorance?

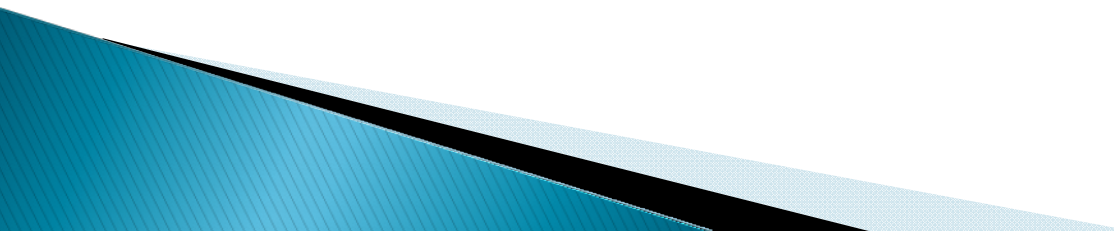
# Weiss Textbook

- Just the right topics for CSSE 220 and 230.
  - Good mix of theory and practice, design and implementation.
  - Lots of interesting language issues. He talks about Java, but applicable to other languages.
  - Challenging problems, a good place to go as you review for exams.
  - Read it!
  - Assignment for Day 2, read through Section 2.2.
  - By Day 3 class, finish Chapter 2 and part of Chapter 3.
  - Take the ANGEL quiz each day.
- 

# Bonus points for reporting bugs

- ▶ In the textbook
  - ▶ In any of my materials.
  - ▶ Use the Bug Report Forum on ANGEL
  - ▶ More details in the Syllabus.
- 

# 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!
- 

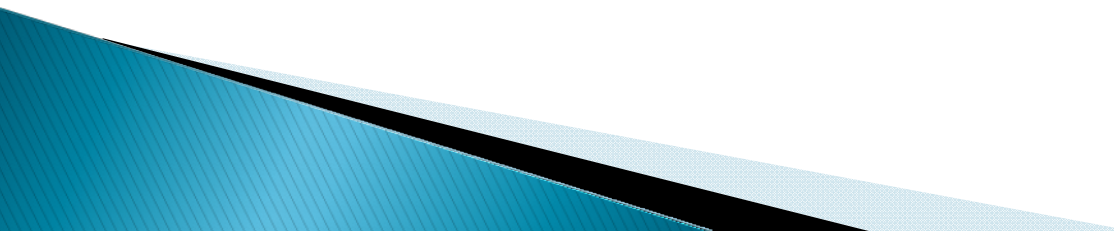
# More Administrivia Tomorrow

- ▶ After you have had a chance to read the syllabus.
  - If you have questions on it, write them down so you'll remember to ask about them in class.
  - Same thing for reading from the textbook.
- ▶ Also an introduction to me.

# Two Audiences for the Java Intro

- ▶ 1 / 3 of you know some Java from taking CSSE 120 previous to Fall, 2007.
  - Most of the Java intro will be review.
  - But don't go to sleep:
    - a few things are likely to be new,
    - or be rusty in your mind because it has been a while since you did Java programming.
- ▶ 2 / 3 of you know some Python and C.
  - I assume that Java is unknown to you.
  - We can move fast because of what you do know.
  - I'll sometimes compare/contrast Java with Python or C.
  - Folks from the other 1 / 3 should not need that analogy, but if you wish you can learn a little about Python and/or C in the process.

# 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

- ▶ Many similar primitive types: int, char, long, float, double, ....
- ▶ Static typing. Types of all variables must be declared.
- ▶ Similar syntax and semantics for if, for, while, break, continue, function definitions.
- ▶ Semicolons required mostly in the same places.
- ▶ Execution begins with the main() function.
- ▶ Comments: `//` and `/* ... */`
- ▶ Arrays are homogeneous, and size must be declared at creation.



# A First Java Program

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

Define a constant, MAX

Except for `public static`, everything about this function definition is identical to C.

Note the function signature for Java's `main()` .

We can declare the loop counter in for loop header.

`println` terminates the output line after printing; `print` does not.

`System.out` is Java's standard output stream. Note that this is the variable called `out` in the `System` class.

```
// Author: Claude Anderson. Nov 19, 2007.
```

```
public class Factorial_1_FirstJavaProgram {
```

```
    public static final int MAX = 17;
```

```
    /* Returns the factorial of n */
```

```
    public static int factorial (int n) {
```

```
        int product = 1;
```

```
        int i;
```

```
        for (i=2; i<=n; i++) {
```

```
            product = product * i;
```

```
        }
```

```
        return product;
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        for (int i=0; i <= MAX; i++) {
```

```
            System.out.print(i);
```

```
            System.out.print("! = ");
```

```
            System.out.println(factorial(i));
```

```
        }
```

```
    }
```

```
}
```

`System.out` is an object from the `PrintStream` class. `PrintStream` has methods called `print()` and `println()` .

# Run the First Java Program

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
15! = 2004310016
16! = 2004189184
17! = -288522240
```

What happens when i gets to 14?

```
// Author: Claude Anderson. Nov 19, 2007.
```

```
public class Factorial_1_FirstJavaProgram {

    public static final int MAX = 17;

    /* Returns the factorial of n */
    public static int factorial (int n) {
        int product = 1;
        int i;
        for (i=2; i<=n; i++) {
            product = product * i;
        }
        return product;
    }

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

# Larger Factorials: the `long` type

It still overflows,  
but not as quickly.

```
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = -4249290049419214848
```

```
public class Factorial_2_WithLongs {
    public static final int MAX = 21;
```

```
    /* Return the factorial of n */
    public static long factorial (int n) {
        long product = 1;
        for (int i=2; i<=n; i++)
            product *= i;
        return product;
    }
```

`static`: Not associated  
with any particular object.

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

If either operand is a String, `+` is the  
concatenation operator.

If the other argument of `+` is not a string,  
that argument is automatically converted  
to a String (unlike in Python, where you  
must explicitly call `str()` to do the  
conversion).

A Java `int` is a 32-bit signed integer;  
a `long` is a 64-bit signed integer.

# Huge Factorials With BigInteger

Java's `BigInteger` is like Python's `long` type. There is set limit on how large a `BigInteger` can be. But calculations are less efficient than with Java's `int` or `long` types.

`new BigInteger(someString)` calls the `BigInteger` constructor that takes a `String` argument.

`multiply()` is a method of the `BigInteger` class that takes a `BigInteger` object as its argument, and returns the product as a new `BigInteger` object.

`final` means that the value of this variable can never change. So it is treated as a constant.

The `BigInteger` class is imported from the `java.math` package.

```
import java.math.BigInteger;

public class Factorial_3_BigInteger {

    public static final int MAX = 100;

    /* Return the factorial of n */
    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i=2; i<=n; i++)
            prod = prod.multiply(new BigInteger(i + ""));
        return prod;
    }

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

ONE is the name of a `BigInteger` constant (that represents the integer 1).

`i + ""` is a quick and easy way to get from a number to its `String` representation.

the `BigInteger` object returned by `factorial()` can be automatically converted to a `String` because `BigInteger` has a `toString()` method.

# Output in Columns: Fixed Width

```
0      1
1      1
2      2
3      6
4     24
5    120
6    720
7   5040
8  40320
9  362880
10 3628800
11 39916800
12 479001600
13 6227020800
14 87178291200
15 1307674368000
16 20922789888000
17 355687428096000
18 6402373705728000
19 121645100408832000
20 2432902008176640000
21 51090942171709440000
22 1124000727777607680000
23 25852016738884976640000
24 620448401733239439360000
25 15511210043330985984000000
```

```
import java.math.BigInteger;

public class Factorial_4_Printf {

    public static final int MAX = 25;

    /* Return the factorial of n */
    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i=1; i<=n; i++)
            prod = prod.multiply(
                new BigInteger(i + ""));
        return prod;
    }

    public static void main(String[] args) {
        for (int i=0; i <= MAX; i++)
            System.out.printf("%2d  %30s\n", i,
                               factorial(i));
    }
}
```

The syntax and semantics of printf in Java and C are identical for simple output formats. The format strings in Java and Python are also the same

# Output in Columns: Calculated Width

```
0      1
1      1
2      2
3      6
4     24
5    120
6    720
7   5040
8  40320
9  362880
10 3628800
11 39916800
12 479001600
13 6227020800
14 87178291200
15 1307674368000
16 20922789888000
17 355687428096000
18 6402373705728000
19 121645100408832000
20 2432902008176640000
21 51090942171709440000
22 1124000727777607680000
23 25852016738884976640000
24 620448401733239439360000
25 15511210043330985984000000
```

```
import java.math.BigInteger;

public class Factorial_5_CalculateWidth {
    public static final int MAX = 25;

    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i=1; i<=n; i++)
            prod = prod.multiply(new BigInteger(i + ""));
        return prod;
    }

    public static void main(String[] args) {
        int len = factorial(MAX).toString().length();

        for (int i=0; i <= MAX; i++)
            System.out.printf("%2d %" + len + "s\n",
                               i,
                               factorial(i));
    }
}
```

# Interlude

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"!!!



© Scott Adams, Inc./Dist. by UFS, Inc.

# Ask user for value (new way)

Import the Scanner class from the java.util package.

System.in is Java's standard input stream. Note that this means the variable called in in the System class.

Other Scanner methods include nextDouble(), nextLine(), nextBoolean(), hasNextInt(), hasNextline().

```
import java.math.BigInteger;
import java.util.Scanner;

public class Factorial_6_Scanner {
    public static final int MAX = 25;

    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i=1; i<=n; i++)
            prod = prod.multiply(new BigInteger(i + ""));
        return prod;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a nonnegative integer: ");
        int n = sc.nextInt();
        System.out.println(n + "! = " + factorial(n) );
    }
}
```

If we do not do the import, we can write

```
java.util.Scanner sc = new java.util.Scanner(System.in);
```

So import is a simple convenient shortcut



# Ask user for value (old way)

Using the new Scanner class is easier than this approach. But you will often see the old approach in other people's code (including Mark Weiss' code).

Think of this as the "magic incantation" for getting set up to read from standard input.

`readline()` returns the next line of input as a String.

Since `readline()` could generate an IO Exception, the try/catch is required.

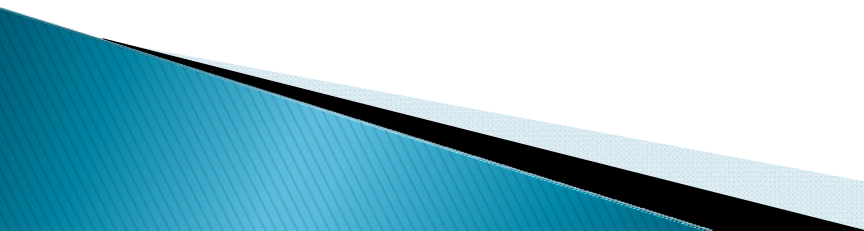
```
import java.math.BigInteger;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;

// omitted definition of the factorial method

public static void main(String[] args) {
    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(System.in));
    String line = "";
    System.out.print("Enter a positive integer: ");
    try {
        line = in.readLine();
    } catch (IOException e) {
        System.out.println("Could not read input");
    }
    int n = Integer.parseInt(line);
    System.out.println(n + "! = " + factorial(n) );
}
```

`parseInt()` takes a string that represents an integer and returns the corresponding int value. It is somewhat similar to Python's `int()` function.

# Command-line arguments

- ▶ Sometimes a program is not run standalone or from within an Integrated Development Environment.
  - ▶ Sometimes it is run as part of a script, from a command-line, such as a UNIX/Linux shell or a Windows DOS Prompt.
  - ▶ The `args` parameter of the Java `main()` method allows us to access the command-line arguments.
  - ▶ `args[0]` is the first command-line argument, `args[1]` the second, etc.
- 

# Add Java to your Windows Path

- ▶ You need to have the folder containing `javac.exe` in your **Path**\* variable:
- ▶ If you have not already added it:
  - Right-click My Computer, choose Properties
  - Advanced, then Environment Variables
  - Under System variables, click Path, then click Edit
  - Click in Variable value field, then press the End key
  - Add a semicolon, followed by the path to your Java bin folder, which should be something like  
**C:\Program Files\Java\jdk1.6.0\_01\bin**
  - Click OK several times to exit.

\***Path** is a list of folders in which the system looks for programs to run.



# Factorial with command-line argument

```
import java.math.BigInteger;

public class Factorial_8_CommandLine {
    public static final int MAX = 25;

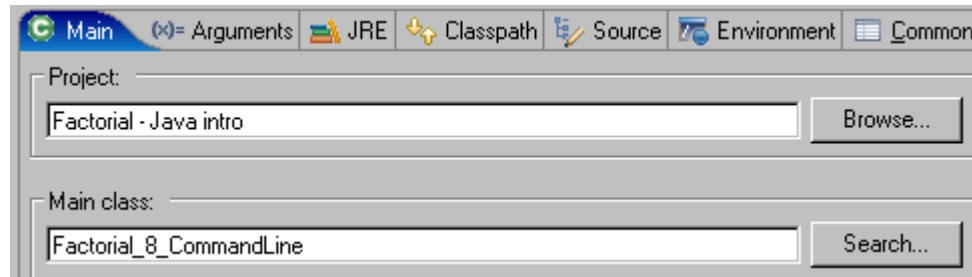
    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i=1; i<=n; i++)
            prod = prod.multiply(new BigInteger(i + ""));
        return prod;
    }

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        System.out.println(n + "! = " + factorial(n));
    }
}
```

args[0] is the first  
command-line argument.

# Command-line arguments in Eclipse

- ▶ Run as ... Run ...
- ▶ On Main tab, make sure the class you want to run is selected. If not, use Search ...



- ▶ Click **Arguments** tab.
- ▶ Enter the Arguments under **Program Arguments**.
- ▶ Click Run.



# What if a user types something wrong?

If any exception gets thrown by the code in the try clause, the catch clauses are tested in order to find the first one that matches the actual exception type.

If none match, the exception is thrown back to whatever method called this one.

If it is never caught, the program crashes.

```
import java.math.BigInteger;

public class Factorial_9_InputErrors {

    public static BigInteger factorial(int n) {
        if (n < 0)
            throw new IllegalArgumentException();
        BigInteger prod = BigInteger.ONE;
        for (int i = 1; i <= n; i++)
            prod = prod.multiply(new BigInteger(i + ""));
        return prod;
    }

    public static void main(String[] args) {
        try {
            int n = Integer.parseInt(args[0]);
            System.out.println(n + "! = " + factorial(n));
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Command-line arg required");
        } catch (NumberFormatException e) {
            System.out.println("Argument must be an integer");
        } catch (IllegalArgumentException e) {
            System.out.println("Argument cannot be negative");
        }
    }
}
```

# Factorial recursive

Recursive factorial definition:

$$n! = 1 \quad \text{if } n = 0$$
$$n! = (n-1)! n \quad \text{if } n > 0$$

```
import java.math.BigInteger;
```

```
public class Factorial_10_Recursive {
    public static final int MAX = 30;

    /* Return the factorial of n */
    public static BigInteger factorial(int n) {
        if (n < 0)
            throw new IllegalArgumentException();
        if (n == 0)
            return BigInteger.ONE;
        return new BigInteger(n+ "").multiply(factorial(n-1));
    }

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

Recursive basically means:  
The method calls itself.



# Speed up Factorial Calculation with Caching

Store previously-computed values in an array called `vals`

```
import java.math.BigInteger;

public class Factorial_11_Caching {
    public static final int MAX = 30;
    static int count = 0; // How many values have we cached so far?
    static BigInteger [] vals = new BigInteger[MAX+1]; // the cache
    static { vals[0] = BigInteger.ONE; } // Static initializer

    /* Return the factorial of n */
    public static BigInteger factorial(int n) {
        if (n < 0 || n > MAX)
            throw new IllegalArgumentException();
        if (n <= count) // If we have already computed it ...
            return vals[n];
        BigInteger val =
            new BigInteger(n+ "").multiply(factorial(n-1));
        vals[n] = val; // Cache the computed value before returning it
        return val;
    }

    // Code for main()omitted. Same as in previous example.
}
```

# File Input/Output

```
import java.util.*;
import java.io.*;

public class FileIOTest {
    /* Copy an input file to an output file, changing all letters to uppercase.
    This approach can be used for input processing in almost any program. */
    public static void main(String[] args) {
        String inputFileNames = "sampleFile.txt";
        String outputFileNames = "upperCasedFile.txt";
        try {
            Scanner sc = new Scanner(new File(inputFileNames));
            PrintWriter out = new PrintWriter(new FileWriter(outputFileNames));
            while (sc.hasNextLine()) { // process one line
                String line = sc.nextLine();
                line = line.toUpperCase();
                for (int i= 0; i< line.length(); i++)
                    // normally we might do something with each character in the line.
                    out.print(line.charAt(i));
                out.println();
            }
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# More File Input/Output

Essentially the same as before

```
import java.util.Scanner;
import java.io.*;

public class TryFileInputOutput {

    public static void main(String[] args) {
        String inFileName=null ,outFileName = "outFile.txt";
        Scanner fileScanner;
        PrintWriter out;
```

```
        try {
            Scanner sc = new Scanner(System.in);
            while (true) // until we get a valid file.
                try {
                    System.out.print("Enter input file name: ");
                    inFileName = sc.nextLine();
                    fileScanner = new Scanner(new File(inFileName));
                    break; // we have a valid file, so exit the loop.
                } catch (FileNotFoundException e) {
                    System.out.println("Did not find file " + inFileName + ". Try again!");
                }
        }
```

Keep looping until user enters the name of an input file that we can actually open.

```
        out = new PrintWriter(new FileWriter(outFileName));
        while (fileScanner.hasNextLine()){ // process one line
            String line = fileScanner.nextLine();
            line = line.toUpperCase();
            for (int i=0; i<line.length(); i++)
                out.print(line.charAt(i)); // process each char on the line
            out.println();
        }
        out.close();
        fileScanner.close();
        System.out.println("Done!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Essentially the same as before

# Primitive types

Primitive Type	What It Stores	Range
byte	8-bit integer	-128 to 127
short	16-bit integer	-32,768 to 32,767
int	32-bit integer	-2,147,483,648 to 2,147,483,647
long	64-bit integer	$-2^{63}$ to $2^{63} - 1$
float	32-bit floating-point	6 significant digits ( $10^{-46}$ , $10^{38}$ )
double	64-bit floating-point	15 significant digits ( $10^{-324}$ , $10^{308}$ )
char	Unicode character	
boolean	Boolean variable	false and true

**figure 1.2**

The eight primitive types in Java

# Java *switch* statement

**figure 1.5**

Layout of a switch statement

```
1 switch( someCharacter )
2 {
3     case '(':
4     case '[':
5     case '{':
6         // Code to process opening symbols
7         break;
8
9     case ')':
10    case ']':
11    case '}':
12        // Code to process closing symbols
13        break;
14
15    case '\n':
16        // Code to handle newline character
17        break;
18
19    default:
20        // Code to handle other cases
21        break;
22 }
```

# Ternary conditional operator ? :

```
1 public class MinTest
2 {
3     public static void main( String [ ] args )
4     {
5         int a = 3;
6         int b = 7;
7
8         System.out.println( min( a, b ) );
9     }
10
11     // Method declaration
12     public static int min( int x, int y )
13     {
14         return x < y ? x : y;
15     }
16 }
```

**figure 1.6**

Illustration of method declaration and calls

# To do for tomorrow

- ▶ Read the syllabus.
    - You can ask questions in the next class meeting.
  - ▶ Read Weiss chapter 1 and sections 2.1–2.2.
  - ▶ Take the ANGEL quiz on that material.
  - ▶ Install Subclipse if you do not already have it.
  - ▶ Write a short Java program (two static methods)
  - ▶ Turn in that program on AFS.
  - ▶ Details are in the `hw01.html` document.
- 