

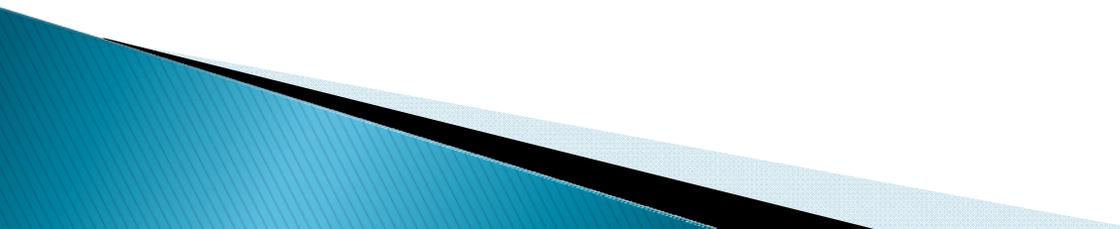
# CSSE 220 Day 3

Java Intro, Continued  
Exceptions, File I/O, Strings,  
Arrays, Object intro

# CSSE 220 Day 3

- ▶ Don't forget the **Python vs. Java** comparison document in the Resources folder on the Web.
- ▶ ANGEL quiz multiple submissions:  
Last submission is the one that counts.
  - (Except on Quiz 1, since I did not announce this for that quiz).
- ▶ I will not collect In-class Quiz 2. I incorporated the questions into Quiz 3.
- ▶ HW3 assignment and ANGEL Quiz 3 will be available this afternoon.

# Your questions about ...

- ▶ Syllabus
  - ▶ Java
  - ▶ Reading from the textbook
  - ▶ Homework
  - ▶ etc.
- 

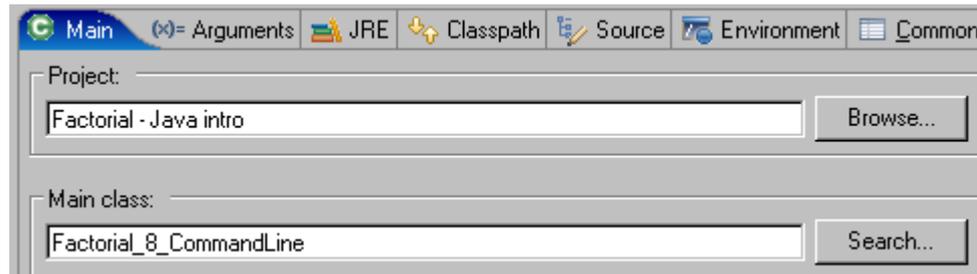
# HW1 Program Solutions

```
public static int maxOfThree (int a, int b, int c) {
    int larger = (a>b) ? a : b;
    return (larger > c) ? larger : c;
}

public static int monthsToReach(double target,
                                double monthlyDeposit,
                                double annualInterestRate) {
    double total = 0;
    double monthlyInterestRate = annualInterestRate / 12;
    int month = 0;
    while (total < target) {
        double interest = total * monthlyInterestRate;
        total = total + interest + monthlyDeposit;
        month++;
    }
    return month;
}
```

# 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.



# Java's five **Exception** keywords

- ▶ Discuss with the person next to you (for two or three minutes):
  - First, tell that person something she/he probably does not know about you.
- ▶ What does each of the following mean?
- ▶ What can you say about how it is used?
  - **try**
    - Run some code that might throw an exception
  - **catch**
    - if this type of exception is thrown, run the following code ... (i.e. **handle** the exception).
  - **finally**
    - Run this code at the end, whether there is an exception or not.
  - **throw**
    - I discovered something I don't know how to handle.  
Does anyone who called me know what to do?
  - **throws**
    - This method might throw this kind of checked exception.  
If it does, I'm not handling it!  
Documentation for the compiler and users!

# 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");
        }
    }
}
```

# 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
        count = n;
        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

# 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 }
```

# A program that uses switch

```
// Adapted from Java Examples in a NutSheell 3rd Ed, by David Flanagan.
// The children's game FizzBuzz.

public class FizzBuzz {
    public static void main(String[] args) {
        for(int i = 1; i <= 100; i++) {
            // count from 1 to 100
            switch(i % 35) {
                // What's the remainder mod 35?
                // For multiples of 35... 1 2 3 4
                case 0:
                    // print "fizzbuzz".
                    System.out.print("fizzbuzz ");
                    break;
                // Don't forget this statement!
                // If the remainder is any of these
                // then the number is a multiple of 5
                case 5: case 10: case 15:
                case 20: case 25: case 30:
                    // so print "fizz".
                    System.out.print("fizz ");
                    break;
                // For any multiple of 7...
                case 7: case 14: case 21: case 28:
                    // print "buzz".
                    System.out.print("buzz ");
                    break;
                // For any other number...
                default:
                    // print the number.
                    System.out.print(i + " ");
                    break;
            }
            if (i%10 == 0) System.out.println();
        }
    }
}
```

```
1 2 3 4 fizz 6 buzz 8 9 fizz
11 12 13 buzz fizz 16 17 18 19 fizz
buzz 22 23 24 fizz 26 27 buzz 29 fizz
31 32 33 34 fizzbuzz 36 37 38 39 fizz
41 buzz 43 44 fizz 46 47 48 buzz fizz
51 52 53 54 fizz buzz 57 58 59 fizz
61 62 buzz 64 fizz 66 67 68 69 fizzbuzz
71 72 73 74 fizz 76 buzz 78 79 fizz
81 82 83 buzz fizz 86 87 88 89 fizz
buzz 92 93 94 fizz 96 97 buzz 99 fizz
```

# Primitives vs. Objects

- ▶ What is the main difference between primitive types and object types?
- ▶ Consider these two code snippets (assume that we have `import java.awt.Point;` at the top of the file.

```
int a = 3, b = 2;  
b = a;  
a = 4;  
System.out.println(a + " " + b);
```

```
Point p1 = new Point(4, 5), p3, p4;  
p3 = new Point(p1.x, p1.y);  
p4 = p1;  
System.out.println("p3==p1? " + (p3==p1) + " " +  
                    "p3.equals(p1)? " + p3.equals(p1));  
p3.y = 500;  
p4.x = 100;  
System.out.println(p1 + " " + p3 + " " + p4);
```

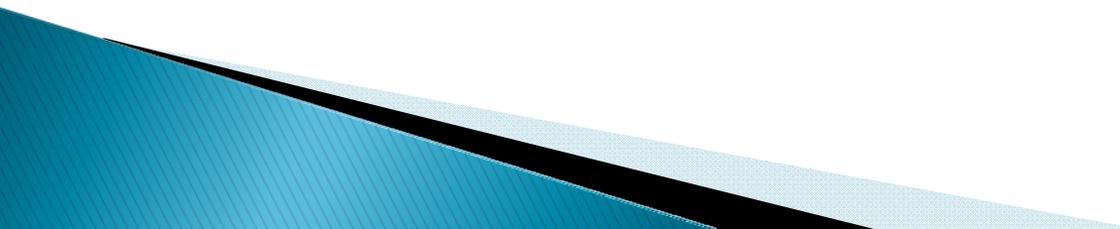
# Characters and Strings

- ▶ `char` is a (primitive) integer type that represents a single character.
- ▶ 

```
char c1='a', c2 ='\n', c3='\ \', c4=65;  
System.out.printf("%c%c%c%c%c\n",  
                  c1, c2, c3, c4, c1+1);
```
- ▶ output from the above:  

```
*a*  
\ *A*b*
```
- ▶ `String` is an object type that represents a sequence of zero or more characters.
- ▶ `'a'` vs `"a"`. Draw the pictures.

# The String class

- ▶ A Java **String** object is immutable.
  - ▶ I.e., once created, you cannot change its length or the individual characters in the String.
  - ▶ String constants are enclosed in double quotes.
  - ▶ + is the concatenation operator
  - ▶ Every class has a `toString()` method, which returns a String representation of an object.
- 

# String Declarations and Operations

```
String s1 = new String();
System.out.println("**" + s1 + "**");
String s2 = "";
System.out.println("1 ==? " + (s1==s2));
String s3 = "abc";
String s4 = "ab" + "bc".substring(1);
String s5 = "ab".concat("c");
System.out.println("2 ==? " + (s3==s4));
System.out.println("3 ==? " + (s4==s5));
System.out.println("1 equals? " + (s1.equals(s2)));
System.out.println("2 equals? " + (s4.equals(s3)));
System.out.println("3 equals? " + (s4.equals(s5)));
s1 = "AbCdEfG";
System.out.println(s1.length() + " " + s1.charAt(3) + " " +
    s1.toLowerCase() + " " + s1.substring(2, 5) + "\n" +
    s1.replace("bc", "XYZ") + " " + s1.indexOf('C') + " " +
    s1.substring(0,3).equalsIgnoreCase(s3));
```

Later: look at the `String.format()` method.

It is like `printf()`, but it returns the formatted string instead of printing it.

# Some **static** String Methods to Write

- ▶ `printReverse(s)` prints the String `s` in reverse order.
- ▶ `reverse(s)` returns a String that is the reverse of `s`.
- ▶ `multiply(s, i)` returns a String that contains `i` copies of `s`, where  $i \geq 0$ .

# Array Basics

```
int [] nums = new int[5];  
for (int i=0; i<nums.length; i++)  
    nums[i] = i*2 + 1;  
System.out.println(nums);
```

```
for (int j : nums)  
    System.out.print(j + " ");
```

```
System.out.println();  
int [] moreNums = nums;  
moreNums[3] = 100;  
System.out.println(nums[3]);  
moreNums = nums.clone();  
moreNums[2] = 1000;  
System.out.println(nums[2]);  
int [] stillMore = {2, 4, 6, 8};
```

```
Point [] pts = {new Point(0,0), new Point(3,4), new Point(5, 6)};  
Point [] pts2 = new Point[6];  
pts[1].translate(-2, 2);  
System.out.println(pts[1]);
```

Write these methods:

```
public static void printIntArray(int[] a)
```

```
public static void reverseObjArray(Object[] a)
```

How many objects are created by the declaration of pts2?

# 2-dimensional and ragged arrays

```
int[][] table = new int[4][3];
    int[][] table2 = {{1, 4}, {2, 3}, {-2, 4}};
    int[][] ragged = new int[4][];
    ragged[0] = new int[2];
    ragged[0][1] = 4;
    ragged[1] = new int[1];
    ragged[1][0] = 6;
```

- ▶ Practice later: write methods to find the sum of the elements of
  - a 2D rectangular array of ints
  - a 2D ragged array of ints
- ▶ Why should there be two separate methods?

# "Resize" an array

- ▶ An array is inherently fixed-length.
- ▶ But we can get the effect of a "growable array":
  - Have two variables, `arr`, and `size`.
  - initialize `arr` to be an array of 5 elements
    - I choose 5 because that is what Mark Weiss does.
  - When we want to add a new element at the end:
    - if `size == arr.length`
      - call `resize` to give us an array twice as big.
    - Put the new element in `arr[size]` and increment `size`.
    - Code:

```
if (size == arr.length)
    arr = resize(arr, size, size*2);
arr[size++] = newValue;
```

Write  
`resize()`

Why \*2 instead of +1?

You'll answer that question mathematically on the first day of 230 (if not sooner)

# ArrayList: a class that implements a resizable array-like structure

- ▶ Full name: `java.util.ArrayList`
- ▶ Methods include
  - `add(element)`
  - `add(index, element)`
  - `get(index)`
  - `size()`
  - `clear()`
  - `remove(object)`
  - `remove(index)`
  - `set(index, element)`
  - `toArray()`
  - `trimToSize()`