September 23, 2013: Announcements

Assignment 2 is due this Wednesday (9/25) at 9:00 pm. If you haven't started it, start it now! (Well, wait until after lecture…)

Keep using Piazza! When you post a new question, please search for similar questions first – things will be much more efficient that way! And please remember to treat everyone with the same level of respect that you would want to be treated with.

Office Hour Change: 1-3pm on Monday and Wednesday (4 total hours of office hours)

Lecture Notes are now posted by day (instead of the entire chapter together when we're done; I'll still do that too though)

Assignment 3 has been posted. Remember to rename the file I posted (from GeneratePassword.java.txt to GeneratePassword.java)

Remember to do your OELs by this Wednesday at (9/25) 11:59 pm; everyone needs to do the orientation lab, but only odd sections also have the other lab to do. Odd sections will also get two more labs after midnight on Wednesday (9/25).

Static Classes: The `Math` Class:

Reading: Section 3.11 and pages 677-678 (the Math class, in Appendix F).

The Math class is a *static* class.

Do **not** create an instance of `Math`.

The `Math` class has two constants:

```
System.out.println("pi is " + Math.PI);
pi is 3.141592653589793
System.out.println("e is " + Math.E);
e is 2.718281828459045
```

Note the all caps: by convention in Java, constants are written in all caps.

Static Classes: The **Math** Class (continued)

Example:

```
public class Circle
{
     public static void main(String[] args)
     {
        float radius = 4.25F;

        System.out.println("Area = " + Math.PI *
                                radius * radius);
        System.out.println("Circumference = " +
                                2 * Math.PI * radius);

     } // end of method main
} // end of class Circle
```

Static Classes: The **Math** Class (continued):

Absolute value, **`Math.abs()`**:

Can find the absolute value of any **`int`**, **`long`**, **`float`**, or **`double`**.

The return type will match the type of the argument:

```
int xray = 4783;
long zebra = -3928276268273986L;
int answerOne = Math.abs(xray); // returns an int since
                                // xray is an int
long willow = Math.abs(xray);
long answerTwo = Math.abs(zebra); // returns a long
                                  // since zebra is a long
int answerThree = Math.abs(zebra); // will not compile!
                                   // Why??
```

Static Classes: The **Math** Class (continued):

Finding the minimum/maximum of two values, **`Math.min()`** and **`Math.max()`**:

The return type will match the type of the arguments:

```
System.out.println("smaller is " + Math.min(278, 29876) );
System.out.println("larger is " +
                   Math.max(-2897.3897, -29837.83) );
int xray, yoke = 17;
xray = Math.min(yoke, yoke);
int zebra = 17;
xray = Math.max(yoke, zebra):
System.out.println("larger is " +
                   Math.max(-2897, -29837.83) );
```

Static Classes: The **Math** Class (continued):

Raise to a power, **`Math.pow()`**:

Java does not have an exponentiation operator that would allow statements like:
answer = $2^5 + 2^6$;

The function **`Math.pow()`** provides this feature.

It takes two arguments, both of which are a **`double`**:

The number to be raised.

The power to use.

Example:

```
double answer;
answer = Math.pow(2, 5) + Math.pow(2, 6);
System.out.println("The answer is " + answer);
```

Static Classes: The **Math** Class (continued):

Raise to a power, **`Math.pow()`**:

It can handle non-integer exponents; i.e., the following computes $613.73^{5/2}$

```
answer = Math.pow( 613.75, 2.5 );
System.out.println("The answer is " + answer);
answer = Math.sqrt( Math.pow( 613.75, 5.0 ) );
System.out.println("Gives the same answer: " +
answer);
```

Another **`Math.pow()`** example:

```
answer = Math.min( Math.pow(x, 3), Math.pow(y, 2) );
answer = Math.pow( xray, 1./3. );
```

Static Classes: The **Math** Class (continued):

Raise to a power, **`Math.pow()`**:

One (but not the other) of these also computes $613.75^{5/2}$ Which is correct, and which is wrong? Why is it wrong?

```
answer = Math.pow( (613.75 + 5.0), 7.0 / 2.0 + 3.2 );
answer = Math.pow( 613.75, 5 / 2 );
answer = Math.pow( 613.75, 5F / 2 );
```

<u>Flow of Control:</u>

Programs can control the order in which their instructions are executed.

A major feature of software and why it is useful!

Four types of flow:

Sequential:

Execute instructions in the order listed in the code. (This is mostly what you have been doing.)

Method calls:

Transfer flow control to the code inside the method; i.e.

```
String zap = "Once upon a time in a galaxy far, far
away...";
String word = zap.subString(22, 28);
System.out.println("word contains: " + word);
```

Control returns back to the point of the call. Some calls also return a value.

Flow of Control: Selection

<u>Reading</u>: Chapter 5

Forming conditions.

`if` statements

Comparing floating-point numbers.

Comparing Objects.

The **`switch`** statement.

The Conditional Operator.

Flow of Control:

Programs can control the order in which their instructions are executed.

A major feature of software and why it is useful!

Four types of flow:

Sequential:

Execute instructions in the order listed in the code. (This is mostly what you have been doing.)

Method calls:

Transfer flow control to the code inside the method; i.e.

```
String zap = "Once upon a time in a galaxy far, far
away...";
String word = zap.subString(22, 28);
System.out.println("word contains: " + word);
```

Control returns back to the point of the call. Some calls also return a value.

Flow of Control:

Programs can control the order in which their instructions are executed.

A major feature of software and why it is useful!

Four types of flow:

Selection: (Chapter 5 — now!)

Which set of instructions are executed depends on the data.

Looping: (Chapter 6)

Repeat a set of instructions, changing some of the data each time through the set of instructions.

Flow of Control:

When would we use flow of control? What would that look like?

Write a program that would decide which line people should use at the grocery store based on how many items they have in their cart.

- Ask the user for how many items they have
- Evaluate the input: print a message that is appropriate for the data entered
  - If they have 15 items or less, tell the user to go to the checkout line
  - If they have more than 15 items, tell the user to go in the normal line

What are some other scenarios where we might want to do something like this?

My example had only two options, so that it could be short. That is not a requirement of Flow of Control. Later on we'll see examples that are much more complex.

Comparison Operators:

Reading: Section 5.1

In the previous example, I said things like, "If the user has 15 items or less…" What does that look like in code?

We need ways to compare values. So, I need to be able to take the temperature that the user gave me and look at it and decide what range it's in, so that I can tell the user what kind of clothes are appropriate.

We will then use these techniques to make choices about which instructions to execute.

All of these techniques have *one goal*: to compute a **`boolean`** value (**`true`** or **`false`**).

Comparison Operators:

Equality, or the lack thereof:

There are two “equality” operators:

`==` compares two values and returns **`true`** if one is equal to the other.

`!=` compares two values and returns **`true`** if one is not equal to the other.

Example:

```
boolean answer1, answer2;
Scanner inputScan = new Scanner( System.in );
int aNum, bNum;
System.out.print("Enter two integers: ");
aNum = inputScan.nextInt();
bNum = inputScan.nextInt();
answer1 = (aNum == bNum);
answer2 = (aNum != bNum);
```

Comparison Operators (continued):

Equality Operators (continued):

Do not confuse these two:

`==` for comparing two values.

`=` for assigning a value to a variable.

Examples:

```
int aNum = 42, bNum = 96;
boolean answer;
answer = (aNum = bNum);
```

Gives the error: “Incompatible types”. Why?

```
int zap;
zap = (aNum = bNum);
System.out.println("zap is " + zap);
```

Compiles without error.
Runs without error.
What is printed?

Comparison Operators (continued):

Relational Operators:

Result is boolean: **`true`** or **`false`**

Relational Operators	Type	Meaning
`<`	binary two operands	is less than
`<=`	binary two operands	is less than or equal to
`>`	binary two operands	is greater than
`>=`	binary two operands	is greater than or equal to

Comparison Operators (continued):

Relational Operators (continued):

Example:

```
import java.util.Scanner;

public class Relational
{
   public static void main(String[] args)
   {
      boolean answer;
      Scanner inputScan = new Scanner( System.in );
      int aNum, bNum;

      System.out.print("Enter two integers: ");
      aNum = inputScan.nextInt();
      bNum = inputScan.nextInt();
```

Comparison Operators (continued):

Relational Operators (continued):

Example (continued):

```
      System.out.println("Result of (aNum < bNum)  is " +
                         (aNum < bNum));
      System.out.println("Result of (aNum <= bNum) is " +
                         (aNum <= bNum));
      System.out.println("Result of (aNum > bNum)  is " +
                         (aNum > bNum));
      System.out.println("Result of (aNum >= bNum) is " +
                         (aNum >= bNum));

   } // end of method main
} // end of class Relational
```

Comparison Operators (continued):

Relational and Equality Operators (continued):

The two-character operators (**==**, **!=**, **<=**, **>=**):

No space in-between the two characters.

The order of the two characters does matter (except for **==**, of course :-):

```
boolean result = (aNum => bNum);
```

Will not compile, gives the error: “illegal start of expression”