Announcements

- TA Schedule to be released soon
- Sample Code on Blackboard
- Top Hat
- Syllabus and Schedule change coming soon
- Program 1 due Monday 9/9
- Java Review (Zybooks Chapters 1-5) due Monday 9/9
- Zybooks chapter 7, sections 1-6 due Monday 9/9

Branches

```
Int x = 5;
If (x < 0) {
        System.out.println(“X is less than zero”);
}
Else if (x < 3) {
        System.out.println(“X is less than three but greater than zero”);
}
Else {
        System.out.println(“X is greater than Zero”);
}
```

Branches (cont.)

```
If (...){
        // Do some stuff
}
Else if (...) {
        // Do some stuff
}
Else if (...) {
        // Do some stuff
}
Else if (...) {
        // Do some stuff
}
```

Branches (cont.)

```
If (...) {
        If (...) {
                If (...) {
                        // Do some stuff
                }
                Else {
                        // Do some stuff
                }
        }
}
Else if (...) {
        // Do some stuff
}
```

Equality and Relational Operators

- == “is equal to”
- != “is not equal to”
- <= “is less than or equal to”
- >= “is greater than or equal to”
- < “is less than”
- > “is greater than”

Logical Operators

- && “Both options MUST be true”
- || “Either option can be true”

Do not mistake ‘&’ for ‘&&’ , or ‘|’ for ‘||’ , they are NOT the same!

Loops

```
boolean x = true;
int y = 0;
while (x) {
	if (y > 5) {
		System.out.println(“Y is now 6”);
		x = false;
	}
	else {
		System.out.println(“The value of y is: “ + y);
		y++;
	}
}
```

Is there a better way to write this while loop?

Loops (cont.)

```
for (int i = 0; i < 5; i++) {
        System.out.println(“The value of ‘i’ is: “ + i);
}
```

Format is:

1. Initialize loop variable (int i = 0)
2. Conditional Expression (i < 5)
    a. Can think of this as “While this conditional expression is TRUE, then keep looping)
3. Update the loop variable (i++)

What happens if you have i-- instead of i++ ?

Loops (cont.)

Nested loops:

```
for (int i = 0; i < 10; i++) {
	for (int j = 0; j < 10; j++) {
		System.out.println(“How many times do i print?”);
	}
}
```

Arrays at a higher level perspective

Arrays can be thought of as multiple **sequential** buckets. And these buckets hold exactly the same type of item.

Arrays (cont.)

Programmatically speaking, an Array stores a list of items. For example if I have three Strings:

- “Hello World”
- “Foobar”
- “Something else lame”

An array can hold all three of those strings...

Arrays (cont.)

```
String[ ] thisIsAnArray = { “Hello World”, “Foobar”, “Something else lame” };
```

OR

```
String[ ] thisIsAnArray = new String[ 3 ];

thisIsAnArray[ 0 ] = “Hello World”;

thisIsAnArray[ 1 ] = “Foobar”;

thisIsAnArray[ 2 ] = “Something else lame”;
```

Arrays (cont.)

To access the elements of an Array, you must use “**indexing**”.

```
int [ ] intArray = new Int[ ] { 1, 2 ,3 ,4 };

System.out.println(intArray[ 0 ]); // Acces the element at index zero

System.out.println(intArray[ 4 ]); // What happens when I execute this line of code?
```

Arrays (cont.)

Look at indexing into arrays from a higher level

Arrays (cont.)

The scary low level stuff about arrays…

- The “Buckets” or elements of an array, exist in **contiguous** (“sequential”) chunks of memory
- Indexing works because of the **immutable** size of primitive type’s

Arrays (cont.)

It is often useful to traverse the length of an array:

```
int[ ] intArray = { 1, 2, 3, 4 };

for (int i = 0; i < intArray.length; i++) {
        System.out.println(“The element at index: “ + i + “ is: “ + intArray[ i ]);
}
```

Methods

A method is a statement, or more specifically a block of code, that performs a specific task and (usually) returns the result to the caller of the method.

Some nice benefits of using methods includes:

- Modular code
- Code reuse (Removes redundancy)
- Increases readability of code

Methods (cont.)

Methods have:

- Parameters: What the method expects as input
- Arguments: What the method actually receives as input
- Output Type: What the method returns when it is done with its task
- Access Modifier: Who get’s to see / use the method?

Methods (cont.)

Here is the signature of a method:

public int addFive(int x)

Notice the placement:

- public (access modifier)
- int (return value type)
- addFive (method name)
- int x (parameter)

Methods (cont.)

```
public int addFive(int x) {
        x = x + 5;
        return x;
}
```

Methods(cont.)

How to use methods?