Hello!
Today we will focus on reviewing the various basic control structures available in Java. We also have links to "The Java Tutorials" (http://docs.oracle.com/javase/tutorial/java/index.html) throughout, in case you want another perspective.

Through this chapter, create a file named "Lab_02.java" and edit it. (Note that your class's name inside this file will also be Lab_02.java.) Just add code for each "Your Turn!" as you reach them, one after another. Remember you can add comments /*...*/ around all the parts that you've completed but don't want to see running over and over each time, without having to delete all your progress.

A note on braces
The curly braces {} let us group multiple statements together into one group. Although they are not required for any of these control structures (except switch), we will show them here because it is very good practice to just always use them.

if-else Branch
Java's branching control uses the if-else structure (http://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html) for selecting which blocks of code should run. The if-statement by itself lets us choose whether or not to run a block of code:

```java
if ( boolExpr ) {
    // run these statements only if boolExpr was true; otherwise, skip.
}
```

The if-else structure lets us select which block of code to run:

```java
if (boolExpr) {
    //run when boolExpr is true
}
else {
    // run when boolExpr is false
}
```

There is no "elif" structure – we can just glue if-else statements together. Since if and else both grab "the next statement" as the branch body, the following if-else statement becomes the else's entire branch.
Below, exactly one of `stmt1`, `stmt2`, `stmt3`, and `stmtDefault` will run, based on the first boolean that is true (or `stmtDefault`, when none of the boolexprs are true).

```java
if (boolexpr_1) {
    stmt1;
} else if (boolexpr_2) {
    stmt2;
} else if (boolexpr_3) {
    stmt3;
} else {
    stmtDefault;
}
```

---

**Switch Statement**
The switch statement ([http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html)) allows us some peculiar control flow possibilities. First, look at an example:

```java
int cutoff = 0;
Scanner myScanner = new Scanner(System.in);
String string = myScanner.nextLine();
char gradeLetter = string.charAt(0); // grabs the first letter.

switch (gradeLetter) {
    case 'A':
        cutoff = 90;
        break;
    case 'B':
        cutoff = 80;
        break;
    case 'C':
        cutoff = 70;
    case 'D':
        cutoff = 60;
        break;
    default:
        cutoff = 0;
}
```

**Notes:**
- The expression at the start can be a byte, short, char, int, and even a String (as of Java 7 – you would need to have version 1.7 or higher installed).
- Each value of a case must be of the same type as the starting expression.
- The expression after the case keyword must be an actual literal value (like 5, 'G', and in Java 7 it can also be strings).
- By putting a break statement at the end of each case, they are isolated from each other and actually behave like "else if" blocks. (There's one break missing above – what will this effect be? case 'C' will run both cutoff=70 and then immediately run cutoff=60).
While Loop
A **while loop** ([http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html)) is like an if-statement that enjoys success so much, it keeps on running again and again until its boolean condition finally fails (at which point the party’s over).

```java
while (thisIsStillTrue) {
    runThisCodeBlock;
}
```

- Remember that if the boolean expression is false the first time, the loop's body might not run at all! A while loop provides for **zero or more iterations** of the loop's body.

Do-While Loop
The do statement, also called the **do-while loop** ([http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html)), is the same in functionality as a while loop except that its body is guaranteed to run at least once. This is why its loop body comes before the condition (boolean expression).

```java
do {
    thisCodeBlock;
} while (thisIsStillTrue);
```

- The body of the block (**thisCodeBlock above**) will always run at least once, because we don't test the condition (**thisIsStillTrue**) until after the block has run.
- The semicolon is required syntax after the parenthesized condition.

For Loop
**For loops** ([http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html)) are excellent for iteration with well-defined changes between values between each iteration, such as the "print this range of numbers" example above, and as we will see, generating the indexes to use when accessing each element in an array.

```java
for (init; guard; update) {
    someCodeBlock
}
```

**Notes.**
The **init**, **guard**, and **update** portions are used as follows:
- **init**: only run once, at the very beginning of the entire for-loop.
- **guard**: the boolean expression used **before** each loop iteration – must be true to run next iteration.
- **update**: statement that is run after each loop iteration. Often it is increment i by 1, but it's your chance to do a single assignment to your loop variable in any amount, direction, or what have you.

Example:
for ( int i = 0; i<10; i=i+1) {
    System.out.println( i );
}