Chapter 2
Control Structures

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:

```
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:

```
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 boolexpr that is true (or `stmtDefault`, when none of the boolexprs are true).

```
if (boolexpr_1) {
      stmt1;
}
else if (boolexpr_2) {
      stmt2;
}
```

```
else if (boolexpr_3) {
      stmt3;
}
else {
      stmtDefault;
}
```

Your Turn! Try to use `if-else`s to solve the following problems:

- Accept three numbers from the user. Print out "`the following ones were even: #2, #3`" (based on actual values), or "`none of them were even.`"
- Again with the three numbers, print out whether or not any two of them could be added up together to be a multiple of 5 (you don't need to say which ones). Try this using "else if" blocks of code, and also try to just do this all in if statement (with no `else`).
- `If-else`s will be useful in the later examples too, as part of the blocks that form the body of loops, the code inside a switch case, and really anywhere a statement is allowed.

Switch Statement

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

```
int cutoff = 0;
Scanner myScanner = new Scanner(System.in);
String s = myScanner.nextLine();
char gradeLetter = s.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`, enumerated type, 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`).

Your Turn!

- Write a switch statement that takes in an integer representing a month (1 – 12), and then prints out "months remaining in the year: ", followed by all the months that are not yet finished. Consider your break statements.
- Use a switch statement to print the following outputs for the following inputs. Try to rearrange cases and use breaks sparingly. To make it interesting (more difficult), try to only have one print statement in your code for each letter that shows up in outputs; you'll have to be careful about how to set up your control flow to always pass through the right piece of code! What other control flow can you use to make this possible?

Input:	Output:
1	A B C
5	B C D
2	A
0	A D

While Loop

A while loop (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).

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

Notes:

- 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.

Your Turn!

- Write a 'password checker' using a while loop. Try to make the wording of your first request friendly, and then all later requests should state say "wrong, try again!" .
- Extend the password example to ask them if the caps lock key is on, starting with the 5th attempt.
- *Take the longcut.* (meaning, let's do something in the not-best way, to learn what we can do). Add up the number of seconds in a day by using nested while-loops. The trick here is to not use multiplication, but actually add up each second individually. Silly constraints, yes, but see if you can figure out the logic and the nested nature of this approach. Print your answer when done. It might help to first use one while-loop to calculate the number of seconds in a minute (yes, it just counts up to 60), and then embed that in a while-loop that does this calculation once for each minute in an hour, and so on.

Do-While Loop

The do statement, also called the do-while loop (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).

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

Notes

- 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.

Your Turn!

- Try to turn your 'password checker' into a do-while loop. (It's perhaps easier when you aren't handling the 1st, 2nd, and 5th + cases differently…)
- Use a do-while loop to print some range of numbers, such as 1-100 inclusive.
- Print this range of numbers, including the {}'s and commas: {100, 200, 300,…2000}. Of course, fill in the ellipses, and use a do-while loop.

For Loop

For loops (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.

```
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 `i++` or something similar (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+=1) {
      System.out.println( i );
}
```

Your Turn!

- Try declaring i before the for-loop (`int i;`), and still assign it to zero in the init area: `for (i = 0; …)`
- Print the even numbers from 0 to a million.
- Print the following, braces/commas and all: {5, 11, 17, 23, … , 65}
- Use a for-loop to print the number 13, thirteen times. Do you still need a loop variable? (the `i` in the above code)

For-each Loop

The for-each loop lets us access each element of something known as an **Iterator**. Arrays are the first Iterator we will learn about. If you are already comfortable with arrays, do this section now. If not, go ahead and skip to the arrays section, and come back here.

```
for ( SomeType someIdentifier  :  iteratorOfSomeType ) {
      /* statement(s) that uses someIdentifier */
}
```

The for-each loop allows us to access each item in a sequence of values in a more regular way than manually managing an array index or carefully-yet-manually constructed conditions for a while loop. It should look similar to Python's for loops. The two following for-loops do the same thing:

```
int[ ] xs = {2,4,6,1,3,5,7};

for (int i = 0; i < xs.length; i++) {
      System.out.println( xs[i] );
}

for (int val : xs) {
      System.out.println( val );
}
```

Notes.

- The type ascription in the for-each loop (int, above) must match what is stored in the Iterator. We have an array of ints, so we must ascribe int here. When we learn more about the types we can create (subtyping and polymorphism and so on), it will become more apparent why we might need to write the type 'again'.
- The identifier, `val`, is a new identifier chosen by the programmer. The current item from the Iterator uses this name.

`val` is indeed the value inside the array at each location, and not the index of the value; if we wanted to print out the indexes (0 through 6 above), we would have to use the original for-loop (or just manually increment a variable to serve as a counter, at which point we also might as well have used the original formulation).

Your Turn!

- Use a for-each loop to calculate the sum of an array of doubles. Create your own array and fill it in with any values you choose.
- Use a for-each loop to find the largest number in an array of ints. Create your own starting array of integers, and test it a few times by moving the largest number around in the array.

Wrap Up

Your Turn!

- Print out all the elements in your arrays you've created. How will you handle the multiple dimensions?
- Use a Scanner and ask the user for 10 integers, storing them in an array. Print them back out.
- Generalize the Scanner example to be however many numbers they first tell you they want to enter. Does this affect where you declare your array? What about where you instantiate it?

- Try calculating the sum of an array of integers – what other storage might you need? What type should it be?
- Choose one of your arrays, and print it out its contents as we've done before, such as {2,4,6,1,3,5,7}, but based on whatever values happen to be in there.
- Report both the *index* of the maximum value in the array, as well as the *value* of the maximum value in the array.