Lesson 3:

Introduction to the Java Basics: Control Flow Statements

Repetition Structures

THEORY

Variable Assignment

You can only assign a value to a variable that is consistent with the variable's declared type.

Example: Complete the following table.
Open your Java editor. Type in the following lines of code, placing
`System.out.print("x: " + x + " y: " + y)`
statements between each line to output the values of x and y after each operation.

Java code	***Value of x***	***Value of y***
`int x, y;`	?	?
x = 2;	2	?
y = x * 3;		
y = y + 1;		
x++; or ++x; or x = x+1;		
x = y – x;		
x = y/x;		
--y;		

Did the value change as you expected it would at each step?
On the line x = y/x; you'll notice that the value should have been 7/4 = 1.75, however, you noticed that x was actually 1. Java truncates everything after the decimal point in cases such as this, where you attempted to assign a real number to an integer variable type.

**Note: when incrementing or decrementing x, it is fine to use unary operators such as x++ or –-x. However, if you are assigning x to another variable in the same step, do not use unary operators (avoid something like y = x++; instead use y = x + 1;) because often other programmers will forget the behaviour of this code and will be unsure if the increment is done before or after the assignment statement.

Repetition Structures

Java has three types of repetition structures:

- `while`
- `do…while`
- `for`

While Loop

- The while statement is Java's most fundamental looping statement, and is the simplest to use
- It repeats a statement (or block of statements) while its controlling expression is true
- Syntax:
-

```
while (Boolean expression)
 {

   //statements to be executed (loop body!)

 }
```

- Braces are unnecessary if only a single statement is required, **however**, braces SHOULD ALWAYS BE PRESENT.
- Three things to coordinate with the while loop:
    1. Initial value(s) must be set up correctly – these may be used to exit the loop
    2. The Boolean expression which acts as the exit condition for the while loop must be set up correctly
    3. The change in variable(s) must be done properly

Boolean Expressions

- Boolean expressions in Java are expressions that evaluate to one of two outcomes: **true** or **false (not 0 or 1 as in other programming languages!)**
- Examples:
    - value1 < value2
    - value1 > value2
    - value1 == value 2 (two equal signs in a row compares two values to test if they are exactly equal or not)
    - value1 <= value2
    - value1 >= value2

While Examples

```
public class WhileExample
{
   public static void main(String [] args)
   {

         int counter = 0;

         while (counter < 5)
         {
               System.out.println("Hello");
               counter++;
         }
         System.out.println("Goodbye");

   }

}
```

Enter and run the above example. What is the output?
Notice that the three items described above are coordinated properly:

1. Initial value, counter, is set up outside the while loop (counter = 0)
2. Boolean expression exits properly (counter < 5)
3. Change in variable is performed (counter++)

```
public class WhileExample2
{

   public static void main(String [] args)
   {

         double pricePerKilo;    //Price of apples
         double kilosOfApples;   //Apples required
         double cost;            //Cost of apples

         pricePerKilo = 1.2;
         System.out.println("Cost of apples per kil: $" + pricePerKilo);

         System.out.println("Kilo's\tCost");
         kilosOfApples = 0.1;

         while(kilosOfApples <= 10.0)
          {
               cost = pricePerKilo * kilosOfApples;
               System.out.println(kilosOfApples + "\t$" + cost);
               kilosOfApples = kilosOfApples + 0.1;
          }

   }

}
```

Do…While Loop

- The do…while statement is very similar to the while looping structure, except that the code within the do…while is guaranteed to execute at least once
- The condition is tested at the end of each repetition
- Syntax:

```
do
{

   //code to be repeated here

} while (Boolean expression);
```

For Loop

- The for loop contains a way to initialize and update variables, in addition to a Boolean expression. You may declare variables in the initialization section.
- Syntax:

```
for (initialization; Boolean expression; update)
{

   //code to be repeated here

}
```

- Example:

```
// Calculate n factorial

public class Factorial
{

   public static void main(String [] args)
   {

         int n = 4;              // calculate 4!
         long factorial = 1;     // value of factorial

         for(int k = 2; k <=n; k++)
         {
               factorial = factorial * k;
         }

         System.out.println(n + "! = " + factorial);

   }
}
```

What is the result displayed by this program? What is the final value of k? Is k incremeneted after the condition is tested? If the condition is false, is k still incremented? Test this out by adding your own System.out.println() statements. We will discuss this during the next class.

TRY IT:

- Add some code (another for loop!) that will print out the factorials of numbers from 1 to 20.

- Other For Loop Variations:
  - Can use the comma operator in the initialization or update sections of the for loop to initialize or update multiple variables.
  - Example:

```
for(a=1, b=4; a < b; a++, b--)
{
      System.out.println("a = " + a);
}
```

- You can omit any or all of the expressions in the loop; if all parts are left empty, you have an infinite loop
- Example:

```
for( ; ; )
{

}
```

Comparing While and For Loops:

```
n = 1;
while (n <= 10)
{
   //code repeated
   n++;
}

for (n = 1; n <= 10; n++)
{
   //code to be repeated here
}
```

It is important to note that any loop can be rewritten using the other two loop structures. That is, there is nothing I can do with a for loop that I cannot rewrite using a while or a do…while loop.

Break and Continue

Break

- Java provides a `break` statement which alters the flow of execution
- This statement is used for two main purposes:
  - Exit in a `switch` statement (studied in the next lesson)
  - Exit a loop, moving execution to the next statement after the loop
- Example:

```
for(int i = 0; i < 100; i++)
{

   if(i == 10)
   {
         break;
   }
   System.out.println("i = " + i);

}
System.out.println("The next statement that executes after the loop!");
```

What effect did the break statement have on the loop? How many times did the loop execute before it terminated? What statement was executed after the break statement caused the loop to terminate?

Continue

- The `continue` statement forces the early iteration of a loop. It skips the current iteration of any loop.
- The control flow skips to the end of the innermost loop's body and evaluates the `boolean` expression that controls the loop.
- Example:

```
long factorial;
long limit=20;

for(int n = 1; n <= limit; n++)
{
   if(n % 2 == 1)
   {
         continue;
   }

   factorial=1;
   for(int k = 2; k <=n; k++)
   {
         factorial = factorial * k;
   }

   System.out.println(n + "! = " + factorial);

}
```

What effect did the continue statement have on the loop?

Which factorials were computed?

What would happen if you removed the continue statement?

When the continue statement is used, is the increment step still done in a for loop?

When is the condition tested?

PROGRAMMING PRACTISE

1. Write a program that outputs the first 20 powers of 10.

2. Write a program to find the smallest value of n for which the $\sum_{k=1}^{n} k^2 \geq 12$

3. Write a program that uses a do…while loop to determine how many days it will take for 50% of a lawn to die, if 6% of the lawn dies each day and you started with 100 units of grass.

4. Write a program that simulates rolling a pair of dice. You can simulate rolling one die by choosing one of the integers 1, 2, 3, 4, 5, or 6 at random (hint: use code: `(int)(Math.random()*6) + 1` which selects a random integer between 1 and 6). The random number generated represents the value of the die roll. Assign this value to a variable to represent one of the dice that are being rolled. Do this twice and add the results together to get the total roll. Your program should report the number showing on each die as well as the total roll. For example:

    The first die comes up 3
    The second die comes up 5
    Your total roll is 8

(Note: The word "dice" is a plural, as in "two dice." The singular is "die.")

5. This exercise builds on the previous exercise. Every time you roll the dice repeatedly, trying to get a given total, the number of rolls it takes can be different. The question naturally arises, what's the average number of rolls? Write a program that performs the experiment of rolling to get a given total 10000 times. The program make a table of the results, displaying the average number of rolls for each dice total, from 2 - 12, something like:

```
Total On Dice     Average Number of Rolls
-------------     -----------------------
       2               35.8382
       3               18.0607
       .                 .
       .                 .
```

Answer the following questions about loops in Java:

1. Why are repetition structures important in a programming language? What limitations would programs have without them?

2. In your opinion, why does Java provide a variety of repetition structures, if every loop can be rewritten as any other loop?

3. Describe scenarios where you believe each of the three repetition structures would be most effective. When is it most convenient to use a while loop? A for loop? A do…while loop?