Chapter 4 – Control Statements (do..while, while.., for..)

Executing statements one after another has limited use. To do more complex programs, we need to be able to loop a certain number of times and stop. This is one of the reasons computers are so powerful, they can do repetitive tasks very quickly. For instance, rendering one frame in a video game involves rendering tens of thousands of polygons; and this is done at 30 frames per second! Writing a good loop also means that we don't have to create as much code. If we don't stop the loop, then this is an infinite loop. The operating system is a program that runs from power on to shutdown, so it is an infinite loop on purpose. Beginning programmers might create an infinite loop accidentally, which is an error.

We have seen this before without explanation. So, here are the looping statements. We will start with *do..while* because we have already used this one to run a program until a sentinel is entered.

do..while statement syntax (for Java, Objective-C, and C++):

```
do
        statement;
while (condition);
```

Again, you can use a *block statement* if you need more than a single *statement*. The *do..while* statement will always execute the *statement* at least once and will continue to do so as long as the condition is true. The other way of stopping the loop is with a *break* statement (as shown in Example_2_4).

while statement syntax (for Java, Objective-C, and C++):

```
while (condition)
        statement;
```

The *while..* statement does not have to execute the statement at least once; if the *condition* starts off false, then it will never execute the *statement*. Again, we can stop looping with the *break* statement.

for statement syntax (for Java, Objective-C, and C++):

```
for (initializer; condition; increment/decrement)
        statement;
```

Note that the *initializer, condition,* and *increment/decrement* are all optional.

In theory, we can always stick with one looping method, but there is a convention for which one choose. If you know how many times to loop, then choose the *for..* statement: for example, calculating the average, maximum, or minimum of a list of numbers. If we don't know how many times to loop then we have to decide whether we need to loop at least one time or not to decide between *do..while* or *while..*: for example, the input loop, such as the console, file, or external device. The difficulty for beginner programmers is deciding how to start/stop a loop and how the condition is constructed. It is generally a good idea to write example situations on paper to figure this out.

> **Exercise 4.1: Write a program that calculates the first 100 triangular numbers. Triangular numbers are the sum of a sequence of natural numbers; eg 1 = 1, 1+2 = 3, 1+2+3 = 6, 1+2+3+4 = 10, 1+2+3+4+5 = 15. Use Example_4_1 as a template. Always do some calculations by hand to get the correct initializer and assignment. Sample printout:**

```
Number          Triangular Number
1               1
2               3
3               6
4               10
5               15
```

For any loop:

- You may or may not have an initializer.
- You should have something that changes the condition so that the loop will terminate.
- You may or may not need an increment/decrement for the condition.

Exercise 4.2: Write a program that calculates the first 100 pyramid numbers. Pyramid numbers are the sum of a sequence of squared numbers; eg $1^2 = 1$, $1^2+2^2 = 5$, $1^2+2^2+3^2 = 14$, $1^2+2^2+3^2+4^2 = 30$. Sample printout:

```
Number         Pyramid Number
1              1
2              5
3              14
4              30
5              55
```

Exercise 4.3: Write a program that prints the sum of a list of numbers. Use Example_4_3 as a template and remove any unnecessary variables. This shows how to form a nested (double) loop and the properly exit. Sample printout:

```
This program calculates the sum of a list of numbers.
Enter your numbers one at a time

Enter integer (C to calculate, Q to quit): 5
Enter integer (C to calculate, Q to quit): 8
Enter integer (C to calculate, Q to quit): 10
Enter integer (C to calculate, Q to quit): c
The sum is: 23

Enter integer (C to calculate, Q to quit): q
```

Exercise 4.4: Write a program that prints the minimum and maximum of a list of numbers. Sample printout:

```
This program calculates the minimum and maximum of a list of numbers.
Enter your numbers one at a time

Enter integer (C to calculate, Q to quit): 5
Enter integer (C to calculate, Q to quit): 8
Enter integer (C to calculate, Q to quit): 10
Enter integer (C to calculate, Q to quit): c
The minimum is: 5
The maximum is: 10

Enter integer (C to calculate, Q to quit): q
```

Exercise 4.5: Write a program that prints the pattern below using nested loops. Use Example_4_4 as a template and do not hard-code numbers, ie. get input for the box size. Sample printout:

```
Enter width (q-quit): 9
Enter height: 5
+++++++++
+++++++++
+++++++++
+++++++++
```

```
+++++++++
Enter width: q
```

Exercise 4.6: Write a program that prints the pattern below using nested loops. Make sure you use a suitable method and do not hard-code numbers. Sample printout:

```
+
++
+++
++++
+++++
++++++
```

Exercise 4.7: Write a program that prints the pattern below using nested loops. Example 4_5 may help. Make sure you use a suitable method and do not hard-code numbers. Hint: Use the absolute value method. Sample printout:

```
++++
+++
++
+
++
+++
++++
```

Exercise 4.8: Write a program that prints the pattern below using nested loops. Make sure you use a suitable method and do not hard-code numbers. Hint: Use exercise 4.7 as your template. Sample printout:

```
+
++
+++
++++
+++
++
+
```

Exercise 4.9: Write a program that prints the pattern below using nested loops. Make sure you use a suitable method and do not hard-code numbers. Hint: 3 loops are needed. Sample printout:

```
     +
    ++
   +++
  ++++
 +++++
++++++
```

Exercise 4.10: Write a program that prints Fahrenheit conversion from -50 °C to 150 °C with increments of 2 C°. Use `System.out.printf("%d\t%.1f\n", celsius, fahrenheit);` to print out your table. Add a comment line in your source code and tell me at what temperature does Celsius match Fahrenheit. Sample printout:

```
Celsius Fahrenheit
-50     -58.0
-48     -54.4
-46     -50.8
-44     -47.2
```

Exercise 4.11: Write a program that prints Celsius conversion from -50 °F to 150 °F with increments of 10 F°. Use `System.out.printf("%d\t%.1f\n", fahrenheit, celsius);` to print out your table.

Exercise 4.12: Write a program that prints the pattern below using nested loops. Sample printout:

```
Enter width (Q to quit): 8
Enter height: 6
++++++++
+      +
+      +
+      +
+      +
++++++++
Enter width: q
```

Exercise 4.13: Write a program that prints out a circle using math functions. If you want it to look round, you will need to multiply the x-coordinates by 2.2. Sample printout:

```
Enter radius (q to quit): 7
               *
        ***************
     ********************
   ************************
  ***************************
 *****************************
*********************************
*********************************
*********************************
 *****************************
  ***************************
   ************************
     ********************
        ***************
               *
Enter radius (q to quit):
```

Exercise 4.14: Write a program that prints out a hollow circle using math functions. This is only 3 more lines of code. Sample printout:

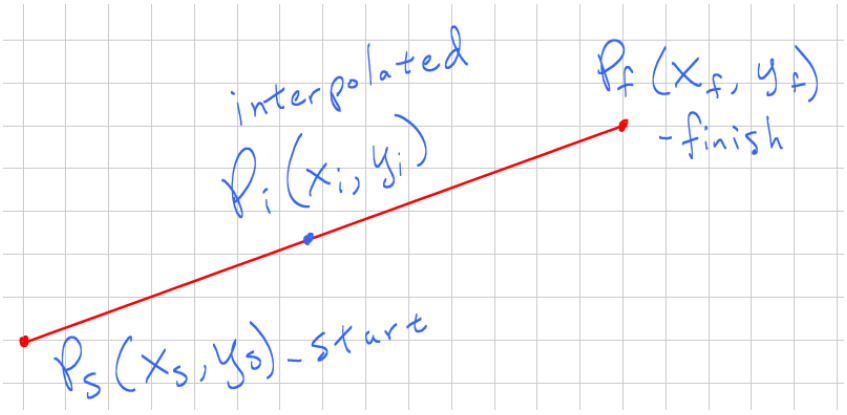
```
Enter radius (q to quit): 7
               *
        *             *
     *                   *
   *                       *
  *                         *
 *                           *
*                             *
*                             *
*                             *
 *                           *
  *                         *
   *                       *
     *                   *
        *             *
               *
Enter radius (q to quit):
```

Linear interpolation is a frequently used technique to reduce the size of a table of lookups. For instance, before there were calculators, trig tables were done for whole degrees or angles in tenths of a degree.

sin(24°)=.4067 and sin(26°)=.4384. If we want to find sin(24.7°), we do linear interpolation: $\frac{\sin(26)-\sin(24)}{26-24}(24.7-24)+.4067=\frac{.4384-.4067}{2}(.7)+.4067=.4178$. The actual value is sin(24.7°)=.4179, so this is a pretty good estimate. We use sine as the dependent variable, *y*; and angle as the independent variable, *x*. The subscript *s* is for start; *f* is for finish; and *i* is for interpolated. Then we have the following formula:

$$y_i=\frac{y_f-y_s}{x_f-x_s}(x_i-x_s)+y_s$$

Note that this is just the two-point form of a line.



Exercise 4.15: Write a program that does the linear interpolation of sine. Ask for two *double* angles and the interpolated decimal angle. Calculate the sine of the two angles, then calculate the interpolated sine. Compare this to the computed interpolated sine. Use the *Math* library to compute sine; you need to convert the angle into radians for the *Math* library to work, so multiply by Math.PI/180. Answer this as a comment in your source file: is sine more accurate when interpolated at 5 degrees or 85 degrees? Sample printout:

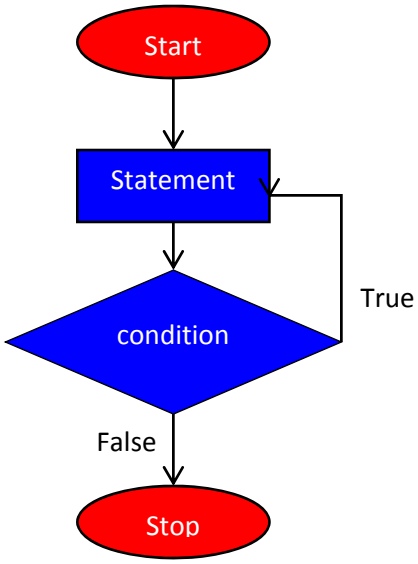
```
Linear Interpolation

Start (q to quit): 24
Finish: 26
Interpolated: 24.7
sin(24.0): 0.406737, sin(26.0): 0.438371
interpolated    sin(24.70): 0.417809
actual          sin(24.70): 0.417867

Start (q to quit):
```

Exercise 4.16: Write a program similar to the previous. Ask for two *double* angles and the number of interpolated values. You may want to use Example_4_2 as a template. Print the interpolated cosine value. Sample printout:

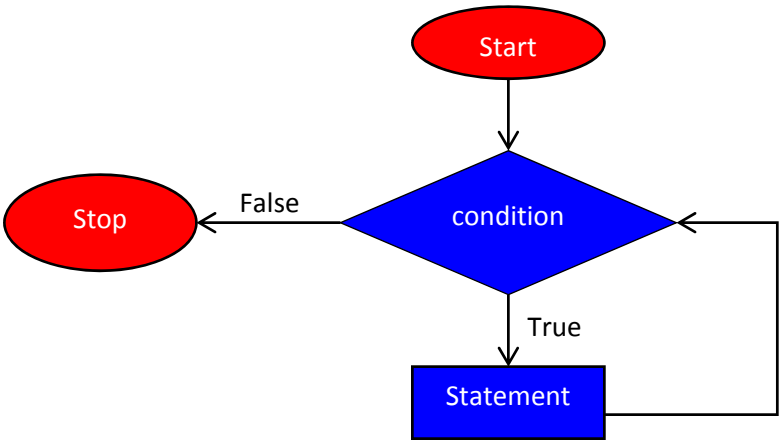
```
Linear Interpolation

Start (q to quit): 5
Finish: 7
Steps: 10
interpolated cos(5.00) = 0.996195               actual cos(5.00) = 0.996195
interpolated cos(5.20) = 0.995830               actual cos(5.20) = 0.995884
interpolated cos(5.40) = 0.995465               actual cos(5.40) = 0.995562
interpolated cos(5.60) = 0.995100               actual cos(5.60) = 0.995227
interpolated cos(5.80) = 0.994735               actual cos(5.80) = 0.994881
interpolated cos(6.00) = 0.994370               actual cos(6.00) = 0.994522
interpolated cos(6.20) = 0.994006               actual cos(6.20) = 0.994151
interpolated cos(6.40) = 0.993641               actual cos(6.40) = 0.993768
interpolated cos(6.60) = 0.993276               actual cos(6.60) = 0.993373
interpolated cos(6.80) = 0.992911               actual cos(6.80) = 0.992966
interpolated cos(7.00) = 0.992546               actual cos(7.00) = 0.992546
```

Here is how we flowchart the 3 different loops:

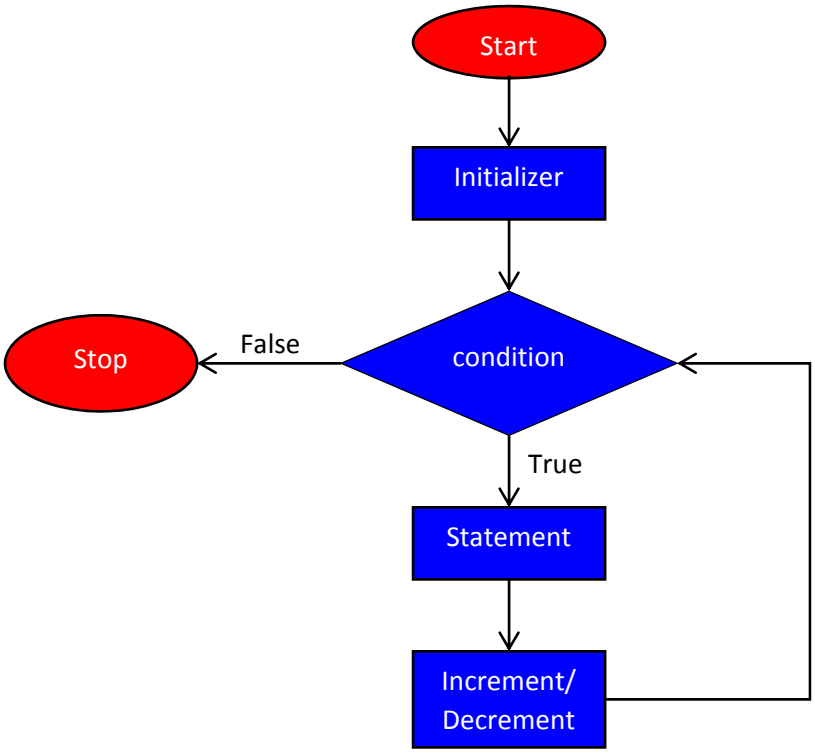
do..while:



while..:



for..:



Email me the exercise source files (not the whole folder) when you are done.