Fundamentals of Programming

Laboratory 4

Loop statements

LOOP STATEMETS

The purpose of loop statements is to *repeat Java statements* many times.

There are several kinds of loop statements:

- **while**
- **do … while**
- **for**

WHILE STATEMENT

“While” statement is used to repeat a block of statements while some condition is true. The condition must become false somewhere in the loop, otherwise it will never terminate

```
while (condition)
{
   statement (s);
}
```

WHILE STATEMENT

- The statement block will be executed as long as the condition in parentheses evaluates to true.
- If there is only a single statement inside the braces, you may omit the braces.
- For clarity, you should always use braces even when there is only one statement

WHILE STATEMENT

The following code prints integer numbers that are less than three:

```
int i = 0;
while (i < 3)
{
   System.out.println(i);
   i++; //equivalent to i=i+1;
}
```

An infinite loop:

```
while (true) { }
```

DO … WHILE STATEMENT

The *do … while* statement should be used when you want to test at the end to see whether something should be repeated.

```
do
{
    statement (s);
} while (condition);
```

DO … WHILE STATEMENT

- The do-while statement is like the while statement, except that the associated block always gets executed at least once

```
int i = 0;
do
{
   System.out.println(i);
   i++;
 } while (i < 3);
```

FOR STATEMENT

FOR STATEMENT

The *for* statement is Java's multipurpose loop controller. It is used for repeating code a known number of times. It is less error-prone than while.

```
for ( initialization ; condition ; update )
{
    statement (s);
}
```

FOR STATEMENT

```
for ( init ;  condition ; update )
{
        statement(s);
}
```

init is an initialization that will be performed once before the first iteration

condition is a boolean expression which will cause the execution of statement(s) if it evaluates to true

update is a statement that will be executed after the execution of the statement block

All the fields are optional:

```
for (;;) {} //an infinte loop
```

FOR Example:

```
for ( int i=0 ; i < 3 ; i++ )
{
     System.out.println(i);
}
```

break and **continue**

```
for ( int i=0 ; i < 3 ; i++ )
{
      System.out.println(i);
      if (i<2)
          continue; //jumps to the start of the loop
      else
          break; //jumps out of the loop
      i++;
}
```

QUESTION 1a

What is the exact output of the following code:

```
int i = 5;
while (i < 15)
{
    System.out.println(i);
    i = i+3;
}
```

ORAL
EXERCISE

QUESTION 1b

What is the exact output of the following code:

```
for (int i = 2; i <= 10; i = i + 3)
    System.out.print(i + " ");
```

ORAL
EXERCISE

QUESTION 1c

What is the exact output of the following code:

```
int i = 1;
do
{
    i = i + 2;
    System.out.println(i);
} while (i <= 11);
```

ORAL
EXERCISE

QUESTION 1d

What is the exact output of the following code:

```
int i = 0;
while (i <= 20)
{
    System.out.print(i + ", ");
    i = i*i + 1;
}
```

ORAL
EXERCISE

QUESTION 1e

What is the exact output of the following code:

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

ORAL
EXERCISE

QUESTION 2a

Check if the following statements are correct, and if not – correct them. The errors may be syntax or logic.

```
for (int i = 0, i < 10, i++)
     System.out.println(i);
```

ORAL EXERCISE

QUESTION 2b

Check if the following statements are correct, and if not – correct them. The errors may be syntax or logic.

```
int i = 0;
while (i < 2)
   System.out.println(i);
   i++;
```

ORAL
EXERCISE

QUESTION 2c

Check if the following statements are correct, and if not – correct them. The errors may be syntax or logic.

```
int i = 0;
do
{
      System.out.print(i + " ");
      i + 1;
} while (i < 5);
```

ORAL EXERCISE

QUESTION 2d

Check if the following statements are correct, and if not – correct them. The errors may be syntax or logic.

```
for (int i = 0; i > 0; i--)
     System.out.println(i & "; ");
```

ORAL EXERCISE

QUESTION 2e

Check if the following statements are correct, and if not – correct them. The errors may be syntax or logic.

```
int i = 5;
while (i >= 5)
{
      System.out.println(i)
      i++;
}
```

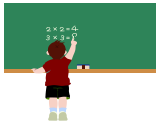
ORAL
EXERCISE

QUESTION 3

Calculate the n’th power of a number x

a) A **while** loop

b) A **for** loop



BOARD
EXERCISE

QUESTION 3-2

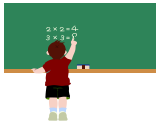
Calculate the factorials of numbers 1 to 5 (1!,2!...,5!) using

a) A **while** loop

b) A **for** loop

Output:

1, 2, 6, 24, 120



BOARD EXERCISE

EXERCISE 1

Implement the factorial program from the board on your computers.

Ask the user for N.

Print all the factorials up to N.

PROGRAMMING EXERCISE

EXERCISE 2

Use all 3 types of loops in one program, to write out all the even numbers from 2 to 50. Each loop should write the numberes again in one line.

Output:

2,4,6,…50

2,4,6,…50

2,4,6,…50

PROGRAMMING EXERCISE

EXERCISE 3

Write a program that computes the following sum:

sum = 1/2 + 1/4 + 1/8 + 1/16 + ... + 1/(2^N)

Where N will be an integer limit that the user enters.

Hint:

Use 1.0/N to obtain the correct result

PROGRAMMING EXERCISE

EXERCISE 4

Write a program that adds up the squares and the cubes of integers from 1 to N, where N is entered by the user. Do this by using just one loop that generates the integers.

Of course, if you really needed to calculate these sums you would use the appropriate formulas:

$$1^2 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n+1)/6$$

$$1^3 + 2^3 + 3^3 + \ldots + n^3 = n^2(n+1)^2/4$$

Add these formulas to your program and print out their results as well as that of the explicit summations.

PROGRAMMING EXERCISE

EXERCISE 5

Write a program that writes a wedge of stars.

Let the user input the initial number of stars. Each line has one few star than the previous line:

******

*****

****

***

**

*

PROGRAMMING EXERCISE

EXERCISE 6

A drug looses **x**% of its effectiveness every month it is in storage. When its effectiveness is below 50% it is considered expired and must be discarded. Write a program that determines how many years and months the drug can remain in storage when the user provides the value **x**.

Hint: *use the modulo operator %*

PROGRAMMING EXERCISE

EXERCISE 7

The Gregory series is perhaps the simplest, and slowest way to approximate π.

pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + ...

Write a program to check how many terms does it take before the sum is in the range 3.14159..3.14160? The series converges very slowly, so it may take over 100,000 terms

Hint: you can use a loop counter to break out of an infinite loop just in case you made some logic error:

```
if (counter > 500000) break;
```

PROGRAMMING EXERCISE

HOMEWORK

Write a program that outputs an “ASCII-art” square. The size of the square should be input by the user.

```
Square size = 0
++
++

Square size = 1
+++
+-+
+++

Square size = 2
++++
+--+
+--+
++++
```

```
Square size = 3
+++++
+---+
+---+
+---+
+++++

...
```

HOMEWORK
EXERCISE