Programming 1

Tutorial 4

Activity 1

Write a program named `BasicForLoop` which uses a `for` loop to print out the numbers from 1 to 10. Also write a program named `BasicWhileLoop` which does the same thing but uses a `while` loop.

Expected program output

```
1
2
3
4
5
6
7
8
9
10
```

Hints

With the `while` loop, you need to declare an `int` variable to hold the number which will be printed in the loop.

Deliverable

BasicForLoop.java

BasicWhileLoop.java

Activity 2

Write a Java program named `FactorialCalculator` to calculate factorial of a user-entered integer using loop (you can choose to use any type of loop). Factorial is defined as follows:

$0! = 1$ (factorial of 0 is 1)

$n! = n \times (n - 1)!$
(factorial of n is n times factorial of $n - 1$)

Sample program output:

```
What integer you want to calculate factorial for?
11
Factorial of 15 is 39916800
```

Deliverable

FactorialCalculator.java

Activity 3

Write a Java program named `PrimeCheck` to find out if a positive number (entered by user) is prime. A number is called prime if it is divisible only by either itself or 1. Number c is divisible by d if $c \% d = 0$. We say that d is a divisor of c. To make sure that the number c doesn't have any other divisor besides 1 and itself, you need to check that $c \% n \neq 0$ for all n $(1 < n < c)$.

(*I hope that my explanation is sufficient and clear enough for you to understand the method to find out whether a number is a prime number!*)

Hint

Your goal is to check that $c \% n \neq 0$ for all n $(1 < n < c)$. You should let user enter the number c by using `Scanner`. The number n is not a single number, in fact, n represents all the numbers from 2 to $c - 1$. In your program, you should let n start with the value of 2 and gradually increase the value of n so that n will become 3, 4, 5, … and stop when n is $c - 1$. This can be easily done with a `for` loop whose loop variable is n. It is also possible to use a `while` loop to to this.

You should also create a `boolean` variable (`isPrime`, for example) and use this `isPrime` variable to reflect the fact that c is a prime number or not. At first, you should set `isPrime` to `true` because you haven't proven that c is not a prime! As soon as you encounter any value of n which is a divisor of c, you can conclude that c is NOT a prime, and set `isPrime` to `false`. After this point, it doesn't matter if you continue to check other values of n (one divisor is enough to draw the conclusion). Recall that you can use the `break` keyword to prematurely exit a loop.

Expected result

```
Enter a positive integer: 17
It is a prime number.

Enter a positive integer: 15
It is not a prime number.
```

Deliverable

PrimeCheck.java

Activity 4

(optional)

Write a program named `SavingsCalculator` to calculate your savings account balance after *y* years with an interest rate of *x* percent per year. The interest is re-invested after each year.

Expected result

```
How much money? 15300
How many years do you want to deposit your money? 10
What's the interest rate (%)? 6.8
After 10 years, you'll receive 29539.55
```

Hint

The formula to calculate received money after *y* years, with *r* percent rate and original amount *m*:

$$m * \left(1 + \frac{r}{100}\right)^{y}$$

Deliverable

SavingsCalculator.java

Activity 5

(optional)

Write a program in Java to print the Fibonacci series up to a given number (entered by user).

Fibonacci is a popular number series in which, starting from the 3rd number, the number is equal to the sum of the previous two numbers. The first two numbers are 0 and 1.

Deliverable

Fibonacci.java

Activity 6

(optional)

An Armstrong's number is 3-digit number for which sum of cube of its digits are equal to itself. For example, 371 is an Armstrong number because of 3*3*3 + 7*7*7 + 1*1*1 = 371). Write Java program to find and print out all Armstrong's numbers between 100 and 999.

Hint

Try to extract the 3 digits from a number first.

Deliverable

AmstrongNumber.java

Activity 7

(optional)

Write a program that reads an integer and prints how many digits the number has, by checking whether the number is `≥ 10`, `≥ 100`, and so on (assume that all integers are less than ten billion). If the number is negative, first multiply it with `-1`.

Deliverable

CountDigits.java

Submission

Submit a **zip** file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.