2 - DECISION MAKING - CONDITIONS

All the programming we have done so far consists of a set of statements, one after another. All statements of our programs have executed in sequence. For this reason these types of constructs are rightly called *sequential statements*.

A very powerful construct that is common to programming languages is the *selection construct*. It allows for conditions to be placed in the program and, if these conditions become true while the program is running, then the statements following the conditions are executed. Otherwise those statements are ignored and the execution of the program continues elsewhere. In this way, we can be selective about the code that gets executed in our program.

2.1 Selection: `if` and `if...else`

The first general form of the selection construct is

```
if (<condition>) {
    <statements>
}
```

The statements within the braces are executed if the condition is true. Program execution continues after the closing brace of the `if` construct *whether or not the condition is true*. The program below uses an `if` construct to implement the absolute value function,

```
import hsa.*;
public class AbsoluteFunction {
    public static void main(String[] args) {
        Stdout.print("Enter a number: ");
        int number = Stdin.readInt();
        Stdout.print("Absolute value of " + number + " is ");
        if (number < 0) {
            number = -1 * number;
        }
        Stdout.println(number);
    }
}
```

The `if` construct accepts an `else` part. If the condition within the `if` part is true, then the statements under the `if` are executed. Otherwise the statements under the `else` are executed. The general form is,

```
if (<condition>) {
    <statements>
}
else {
    <statements>
}
```

In C and in C++ the actual implementation of the selection construct is exactly like Java. Notice the following important points:

a) Statements within the `if` and within the `else` are always *indented*. Code needs to be indented to make programs readable and understandable. Code is indented usually 4 spaces
b) Opening and closing *parenthesis* `( )` are always used to enclose the condition. This is necessary for the program to compile.
c) Opening and closing *braces* `{ }` are used to contain the statements for the `if` part and for the `else` part. In Java these braces are optional if there is only one statement following the `if` or following the `else`. Having the braces regardless of the number of statements within the `if` or the `else` is a way to avoid possible future problems.

The following table summarizes the *logical operators* used in conditions and two (of many) *logical String methods*:

Operator	Name	Used to compare	Example
==	Equals	Numbers	if (distance == 200)
!=	Is not equal	Numbers	if (den != 0)
<	Less than	Numbers	if (age < 18)
>	More than	Numbers	if (height > 180)
<=	Less than or equal to	Numbers	if (month <= 6)
>=	More than or equal to	Numbers	if (side >= 20)
equals	equals	Strings	if (name.equals("John"))
compareTo	compareTo	Strings	if (name.compareTo("M") < 0)

(Note: The `char` and the `byte` are considered to be numeric types)

Here is an example of a program that determines whether the user has input an even or an odd number:

```
import hsa.*;
public class EvenOrOdd {
    public static void main(String[] args) {
        Stdout.print("Enter a number: ");
        int number = Stdin.readInt();

        if (number % 2 == 0) {
            Stdout.println(number + " is even");
        }
        else {
            Stdout.println(number + " is odd");
        }
    }
}
```

When the user types in a number, say 65, it is stored in the variable `number`. Then the condition `(number % 2 == 0)` is evaluated. If the result of `number % 2` is `0`, then the condition evaluates to `true` and thus `number` must be even. If the result of `number % 2` is `1`, then the condition evaluates to `false` and `number` is odd.

```
Enter a number: 45
45 is odd

Enter a number: 14
14 is even
```

2.1 Exercises

1. Write a Java program that asks the user for two numbers and prints out the larger of the two numbers

2. Write a program that asks the user for a mark. If the mark is 50 or more, the program displays the message

   `A mark of <mark> is a PASS`

   where `<mark>` is the actual number that the user typed in. For example,

   ```
   Mark: 79
   ```

```
A mark of 79 is a PASS

Mark: 87
A mark of 87 is a PASS
```

3. Add code to the previous program so that if the mark is something smaller than 50, the message displayed is

   ```
   With a mark of <mark> you do not pass the course
   ```

   For example,

   ```
   Mark: 75
   A mark of 75 is a PASS

   Mark: 46
   With a mark of 46 you do not pass the course

   Mark: 50
   A mark of 50 is a PASS
   ```

4. Modify the program so that if the mark is less than zero or more than 100 it displays an error message. For example,

   ```
   Mark: 92
   A mark of 92 is a PASS

   Mark: 104
   ***Error: Mark cannot be more than 100

   Mark: 43
   With a mark of 43 you do not pass the course

   Mark: -4
   ***Error: Mark cannot be negative
   ```

5. Write a Java program that asks the user for two numbers. The program divides the first number by the second number and prints the result if and only if the second number is not zero. Otherwise, the program prints an error message. For example,

```
First number: 7
Second number: 2
7 / 2 = 3.5

Number 1: 4
Number 2: 0
Error: Division by zero is undefined
```

6. Write a Java program that asks the user for a number and determines whether or not the number divisible by 7. For example:

```
Number: 91
91 is divisible by 7

Number 22
22 is not divisible by 7
```

2.2 Selection: Multiway `if`

Many times we need to include a series of conditions in our program. For example, if we want to print the letter grade for a course mark, we may need to test whether the mark is larger than 80 but less than 90, or larger than 70 but less than 80. Here is a code fragment that handles the situation

```
double mark = Stdin.readDouble();
if (mark > 100) {
    c.println("***invalid mark: " + mark);
}
else if (mark > 90) {
    c.println("A+");
}
else if (mark > 80) {
   c.println("A");
}
else if (mark > 70) {
   c.println("B");
```

```
}
else if (mark > 60) {
   c.println("C");
}
else if (mark > 50) {
   c.println("D");
}
else if (mark > 0) {
   c.println("F");
}
else {
   c.println("***invalid mark: " + mark);
}
```

The general form is

```
if (<condition>) {
    <statements>
}
else if (<condition>) {
    <statements>
}
else if (<condition>) {
    <statements>
}
.
.
.
else {
    <statements>
}
```

In this construct, the last `else` is optional.

2.2 Exercises

1. Write a Java program that asks the user for how much they make in one hour. If the amount entered is `11` then print the words `MINIMUM WAGE`; if the amount entered is more than `11` then print `ABOVE MINIMUM`; otherwise print `***INVALID AMOUNT`. For example,

```
Hourly wage: 11
MINIMUM WAGE

Hourly wage: 4.75
***INVALID AMOUNT

Hourly wage: 17.25
ABOVE MINIMUM
```

2. Write a Java program that asks the user for three numbers and prints out the largest of the three numbers

3. Write a Java program that asks the user for three numbers and prints them out in ascending order

4. Write a Java program that asks for the user for a number from 1 to 12. The program prints the corresponding month as a word. For example:

```
Month: 10
October
```

5. Write a Java program that asks the user for a year and determines if the year is a leap year. Research the definition of leap. ***It is not enough that the year be divisible by 4***. For example, the year 2100 is divisible by 4 but it is not a leap year.

2.3 Selection: `switch...case`

In many instances we will need to make a decision based on one of many possible options. We have already done some of that, and it required many `if...else` constructs one after the other. Java offers a convenient alternative, i.e., the `switch` statement. It has its roots (yet again) in the C programming language that, for this particular construct, also shares the very same syntax as Java. Its syntax is:

```
switch(<expression>) {
     case <expression> :
          <statements>
          break;

     case <expression> :
          <statements>
          break;
     .
     .
     .
     default:
          <statements>
          break;
}
```

In the `switch` construct `<expression>` must be of type `byte, char, short` or `int`. Here is an example:

```
Stdout.print("Enter an option: ");
int option = Stdin.readInt();

switch(option) {
     case 0:
          Stdout.println("You chose option 0");
          break;
     case 1:
          Stdout.println("You chose option 1");
          break;
     case 2:
          Stdout.println("You chose option 2");
          break;
     default:
          Stdout.println("You chose neither of 0, 1, or 2");
          break;
}
```

Here is some sample runs:

```
Enter an option: 0
You chose option 0

Enter an option: 5
You chose neither of 0, 1, or 2
```

Things to keep in mind with the `switch` construct:

1. The expressions in the `switch` and `case` statements must be of an integer type: `byte`, `char`, `short`, or `int; long` is not accepted
2. The `break` statement is necessary to have the program jump from the end of the statements to the end of the `switch` construct. If the `break` statement is not included then the program will continue with the statements in the next `case`. This is something to be careful about; it is a cause for bugs in the program.
3. The `default` statement will cause its statements to run if none of the cases before it have been satisfied.

2.3 Exercises

1. Write a Java program that asks the user for three assignment marks. Once the marks have been entered, the program displays a simple menu with 3 options:

    1) To print the average of the marks
    2) To print the highest mark
    3) To print the lowest mark

    Then the program asks the user for an option (1, 2, or 3). For each of the options, write the Java code needed to calculate the average, or highest, or lowest value. Here is a sample run,

```
Enter assignment 1 mark: 75
Enter assignment 2 mark: 80
Enter assignment 3 mark: 67

1. Print average
```

```
2. Print highest
3. Print lowest

Enter an option: 2
Highest mark is 80
```

Here is another sample run:

```
Enter assignment 1 mark: 75
Enter assignment 2 mark: 80
Enter assignment 3 mark: 67

1. Print average
2. Print highest
3. Print lowest

Enter an option: 1
Average mark is 74
```

2. Write a program that ask the user for a number from 1 to 12. The program then prints out the name of the month. For example,

```
Enter month: 2
February
```

3. Write a program that asks the user for two numbers: The first is a number from 1 to 12. This number is a month. The second number is a year. The program prints the number of days for the given month, taking into account that the year could be leap. Here are some examples,

```
Enter month: 5
Enter year: 2015
31

Enter month: 2
Enter year: 1999
28

Enter month: 11
Enter year: 2000
30

Enter month: 2
Enter year: 2016
29
```

2.4 Compound Conditions

All the conditions we have seen so far require only one test at a given time. If we had a case where two conditions needed to be met, we had to use a *nested* `if` statement, that is, an `if` statement inside another `if` statement. Java allows for more than one condition to be tested simultaneously in a single `if` statement through the use of *Boolean operators.* They are

i. AND (Java symbol is `&&`)
ii. OR (Java symbol is `||`)
iii. NOT (Java symbol is `!`)

These three operators are the foundation of all computer circuit design. All logic boards, processors, memory, and peripherals of computer systems are designed with blocks of these three operators. The most basic machine is the *half-adder* which allows for the addition of numbers using binary. In this course we will learn how to add two numbers with a half-adder by using these operators and we will explore its circuitry. From this block, full adders, multipliers, and all other operations of calculators and computers are built.

2.4.1 AND operator

AND is a *binary* operator that accepts *two operands* of type boolean. In this context, *binary* means that the operator accepts *two operands*. An operand is a value or an expression on which the operator operates. For example, in the expression *2 + 3,* the operator is the plus sign, and the operands are the numbers *2* and *3.* In this case the operands are numbers and the result is also a number, namely 5.

The *AND* operator accepts two boolean operands and the result is also boolean. For example:

if (mark is larger than 90 AND attendance is higher than 95 percent)

For this expression to evaluate to TRUE, *both* the left operand AND the right operand must be true. Thus, if it is the case that mark is larger than 90 and attendance is higher than 95%, then the whole expression is TRUE. If any of the two operands is false then the whole expression is FALSE. We can look at all the possible scenarios of this operator with a *truth table:*

Left Operand	AND	Right Operand
TRUE	**TRUE**	TRUE
TRUE	**FALSE**	FALSE
FALSE	**FALSE**	TRUE
FALSE	**FALSE**	FALSE

In Java, a condition using this operator requires the use of two ampersands: `&&`. For example,

```
if (mark > 90 && attendance > 0.95) {
    <statements>
}
```

A couple of points to be mindful of:

a) In English, the word *BUT* can be replaced by the AND operator to be used in logical and computer language

b) In Java, C and C++ the AND operator is implemented with two ampersands: `&&`. A single ampersand `&` has a different meaning: *bitwise-and* which we will learn soon.

2.4.2 OR operator

OR is a binary operator that accepts two *operands* of type boolean. When the expression is evaluated, the result is also boolean. For example:

if (assignment >= 80 OR quiz >= 85)

For this expression to be TRUE, *it is enough that one of the operands be TRUE*.

There exist two types of OR in languages: *Inclusive* and *exclusive*. In computer languages, the OR is *inclusive.* The truth table for inclusive OR is

Left Operand	OR	Right Operand
TRUE	**TRUE**	TRUE
TRUE	**TRUE**	FALSE
FALSE	**TRUE**	TRUE
FALSE	**FALSE**	FALSE

In Java, the OR operator is implemented with two vertical bars: `||`. For example,

```
if (mark < 50 || absences > 4) {
    Stdout.println("Mark is dropping...");
}
```

The truth table for *exclusive-OR* is

Left Operand	XOR	Right Operand
TRUE	**FALSE**	TRUE
TRUE	**TRUE**	FALSE
FALSE	**TRUE**	TRUE
FALSE	**FALSE**	FALSE

Exclusive-OR is not implemented in Java, but it can be easily implemented using the operators *AND, OR, NOT*. See exercises below.

We need to be careful to use two vertical bars. One vertical bar is a *bitwise-inclusive-or* that we will learn later. The *exclusive or* exists in Java only at the bit level and its operator is ^

2.4.3 NOT operator

NOT is a *unary* operator that toggles the value of the boolean expression. *Unary* means that it accepts only one operand. A value of TRUE is turned to FALSE and a value of FALSE is turned to TRUE. This is very useful when switching from one state to another, such as displaying or not displaying an object on the screen. The truth table for the NOT operator is:

Operand	NOT
TRUE	**FALSE**
FALSE	**TRUE**

In Java, a condition using this operator requires the use of the exclamation mark `!` which means *NOT*. For example,

```
boolean gameOver = false;
if (!gameOver) {
    <initialize game>
    <start game>
}
```

2.4.4 Order of operations

Boolean operators have an order of operations very much like BEDMAS. When an expression is given, the order is

1. Brackets ( )
2. Unary plus, unary minus, NOT + - !
3. Multiplication, division, modulus * / %
4. Addition, subtraction + -
5. Relational comparison < <= > >= ==
6. AND &&
7. OR ||
8. assignment =

If `test` is a boolean variable and we let `x = 4` and `y = -5` then the expression

```
test = x == 4 && (!(y < 0))
```

evaluates in this order:

a) `y < 0` is `true`
b) `!(y < 0)` is `false`
c) `(!(y < 0))` is `false`
d) `x == 4` is `true`
e) `x == 4 && (!(y < 0))` is `false`
f) `false` is assigned to the variable `test`

2.4 Exercises

1. Write a Java program that determines if a number is divisible by 6. In this program you need to use the '%' mod operator but you are restricted to using it only with 2 and 3. You may not use an expression that involves the number 6. The point of this is to use a compound condition.

2. Evaluate all possible cases of the expression `P && Q || R` by completing the following truth table. Remember to apply the correct order of operations:

P	**AND**	Q	**OR**	R
T		T		T
T		T		F
T		F		T
T		F		F
F		T		T
F		T		F
F		F		T
F		F		F

3. Evaluate all possible cases of the expression `!P || Q` by filling in all the possible combinations of TRUE and FALSE for *P* and *Q* in the

following table. Then evaluate the operators NOT and OR. Remember to apply the correct order of operations:

NOT	P	**OR**	Q

4. In the following compound sentences, let P stand for the sentence to the left of the binary operator, and Q stand for the sentence to the right of the binary operator. Then build the truth tables by assigning all possible values of TRUE and FALSE and evaluating the expressions:

    a) Peter has $7 in his pocket **OR** a movie ticket
    b) It is not the case that you are here **AND** I am awake
    c) Year is divisible by 4 **BUT** not by 100

5. Exclusive-OR, *XOR,* is a binary operator that takes two binary operands and evaluates to TRUE if and only if one of the operands is true (See the XOR table above in section 6.4.2) If P is the first operand and Q is the second operand, XOR can be expressed as follows:

```
(P or Q) but not both
```

This can be further refined to

```
(P or Q) but not (P and Q)
```

Write the Java expression that implements *XOR* and verify your expression by writing a Java program that uses it with the four possible different cases: true true, true false, false true, false false. The result of your program should look like this:

```
p         q         (p   xor   q)
-------------------------------
true      true          false
true      false         true
false     true          true
false     false         false
```

6. **Extending:** XNOR is the negation of XOR. In other words, XNOR is the same as NOT XOR. Build the truth table for XNOR using two operands, P and Q in a Java program. Your program output should be

```
p        q        (p   xnor   q)
-------------------------------
true     true          true
true     false         false
false    true          false
false    false         true
```

7. Write a Java program with the class name `IsPointInRectangle`. The program checks if a given point is inside a given rectangle. Proceed as follows:

    a) Use the `Stdin()` method to ask the user for the coordinates of the left upper corner of a rectangle, its width and its height
    b) Use the `Stdin()` method to ask the user for the coordinates of a point
    c) Use an `if` construct to determine if the given point lies inside the rectangle. If it does, then print a statement like

    ```
    Point is in the rectangle
    ```

    Otherwise print a statement like,

    ```
    Point is not in the rectangle
    ```

    d) On a separate window, draw the given rectangle and the point

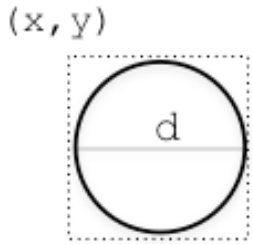
    Does your printed statement agree with what you drew in the other console?

8. Write a Java program with the class name `OverlappingRectangles` that asks the user for the coordinates of the left upper corner of a rectangle, its width and its height. Then, the program asks the user for the coordinates of the left upper corner of another rectangle, its width and its height. The program needs to determine if the rectangles overlap.

9. The boundaries of a Java console window are (0, 0) at the top left corner and (639, 499) at the bottom right. When drawing a circle we use the method:

   ```
   c.drawOval(x, y, d, d)
   ```

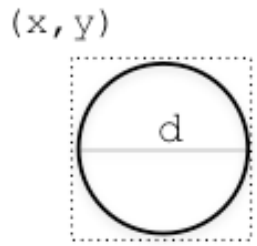
   Java places the circle like this

   where the coordinates `(x, y)` are the upper left corner of an imaginary square that circumscribes the circle.

   Write a Java program with the class name `CircleInConsole` that asks the user for a position `(x, y)` and a diameter `d`. Your program has to determine if the circle will be fully drawn within the Java console window. Then draw the circle and confirm that your program works correctly.

10. Given the circle



and the screen boundaries (0, 0) and (639, 499) , write a Java program with the class name `CircleTouchingFrame` that asks the user for the coordinates `(x, y)`. Then the program determines whether or not the circle is touching the frame of the window. If the circle is touching the frame of the window, print a message that says that the circle is touching on the left or on the right or above or below.

11. Evaluate each of the expressions below if

```
int a = -10;
int b = 5;

double c  = 25;
boolean x = false;
```

a) `b > 0 && b < 10`
b) `c >= 75 || c <= 30 || c == 15`
c) `!x && (x || c == 30) && (x || b < 10)`
d) `-a * 4 < 0 && 22 + b >= c`
e) `c % 19 / 2 >= 5 || !(28 / b == b)`
f) `x = !x`
g) `x = -b == c / -b && !x || false`

12. Use the expression in part b) of the previous exercise. Let the letters P, Q, R stand for each one of the conditions of the expression. Then, re-write the expression using those three letters and work out the complete truth table.

13. Use the expression in part e) of exercise 11, let the letters P, and Q stand for each of the conditions and work out the full truth table.

14. Work out the complete truth tables for these two expressions. Use `0` instead of `false` and `1` instead of `true` in each possible case:

    a) `(P || Q) && !(P && Q)`
    b) `P && Q`

15. Write a Java program that asks the user for three integer values: `xc, yc, r`. These are the coordinates of the centre of a circle and its radius. The program then asks for two other integer values: `x, y`. These are the coordinates of a point. The program then determines if the point is inside the circle. If the point is on the circumference, we consider it inside the circle. Then, draw both the circle and the point on the screen to confirm the output of your program. Keep in mind that the `drawOval()` method does not take the centre and the radius as parameters. Also, remember to cast your calculations as late as possible.

16. Write a Java program that asks the user for the centres and the radii of two circles. The program determines if the circles overlap. Draw both circles on the screen to confirm your result.

17. Write a Java program that asks the user for the center and the radius of a circle. Then, the program asks for the upper left coordinates and the side of a square:
    a. The program determines if the circle is inside the square. Draw the circle and the square to confirm your result.
    b. Modify the program to determine if the square is inside the circle. Draw the circle and the square to confirm your result

2.5 Program indentation

Now that we write programs that contain conditions, it becomes very useful to indent our source code. Doing so gives us a good sense of the program flow. We will revisit code indentation as we make progress in learning Java. Here are the first set of guidelines:

1. When indenting, we do it with a consistent number of spaces: 4 spaces is the standard, but indentation of 8 spaces or 2 spaces is also acceptable as it may be more comfortable for some people.

2. The statement that contains the name of the class is not indented. It is written at the same level as the `import` statement, and the closing brace of the class is at the same level as the word `public` of the class:

```
import hsa.*;
import java.awt.*;

public class Numbers {
}
```

3. A method within a class is indented one level, and the closing brace of the method is at the same level as the word `public` of the method:

```
import hsa.*;
import java.awt.*;
public class Numbers {
    public static void main(String[] args) {
    }
}
```

4. Code within a method is indented one level. Sequential statements stay at the same level:

```
import hsa.*;
import java.awt.*;
public class Numbers {
    public static void main(String[] args) {
        Console window = new Console("Computer Science");
        window.drawLine(0, 0, window.maxx(), window.maxy());
    }
}
```

5. Decision statements are also at the same level as sequential statements, but the code within the braces of an `if` or an `else` is indented one level. The `else` statement is at the same level as the `if`: They are *aligned*. This is essential for us to know how the execution of the program flows. Also, the closing braces are at the same level as the `if` and the `else`:

```
import hsa.*;
import java.awt.*;
public class Numbers {
    public static void main(String[] args) {
        Console window = new Console("Computer Science");
        window.drawLine(0, 0, window.maxx(), window.maxy());

        int x = window.readInt();

        if (x < window.maxx() / 2) {
            window.print("Left");
        }
        else {
            window.print("Right");
        }

        window.setCursor(window.maxrow(), 1);
        window.print("End of program");
    }
}
```

6. When we need to have an `if` statement inside another `if` statement, we indent one more level to create a *nested* `if`. Notice how the `else` is aligned with the `if` to which it (the `else`) belongs:

```
import hsa.*;
import java.awt.*;
public class Numbers {
    public static void main(String[] args) {
        Console window = new Console("Computer Science");
        window.drawLine(0, 0, window.maxx(), window.maxy());

        int x = window.readInt();
        int y = window.readInt();

        if (x < window.maxx() / 2) {
            if (y < window.maxy() / 2) {
                window.print("Left upper");
            }
            else {
                window.print("Left lower");
            }
        }
        else {
            if (y < window.maxy() / 2) {
                window.print("Right upper");
            }
            else {
                window.print("Right lower");
            }
        }

        window.setCursor(window.maxrow(), 1);
        window.print("End of program");
    }
}
```

(next page)

7. The `switch` statement is not indented but the `case` labels are indented one level. Within each `case` the statements are aligned after the label. All `case` labels are aligned, as well as the `default` label at the end. The closing brace of the `switch` statement is aligned with the word `switch`:

```
import hsa.*;
import java.awt.*;
public class Numbers {
    public static void main(String[] args) {
        int number = Stdin.readInt();

        switch(number) {
            case 10 : Stdout.println("You entered 10");
                      break;
            case 20 : Stdout.println("You entered 20");
                      break;
            case 30 : Stdout.println("You entered 30");
                      break;
            default : Stdout.println("I don't know");
                      break;
        }
    }
}
```

2.5 Exercises

1. Using the above guidelines, manually indent the following code:

```
import hsa.*;
public class Section2_5Exercise_1 {
public static void main(String[] args) {
Console c = new Console();

c.print("Name 1: ");
String name1 = c.readLine();
c.print("Name 2: ");
String name2 = c.readLine();
c.print("Name 3: ");
String name3 = c.readLine();
c.print("Name 4: ");
String name4 = c.readLine();
c.print("Name 5: ");
String name5 = c.readLine();

if (name1.compareTo("A") >= 0 && name1.compareTo("B") <= 0) {
c.println(name1);
}
if (name2.compareTo("A") >= 0 && name2.compareTo("B") <= 0) {
c.println(name2);
```

```
}
if (name3.compareTo("A") >= 0 && name3.compareTo("B") <= 0) {
c.println(name3);
}
if (name4.compareTo("A") >= 0 && name4.compareTo("B") <= 0) {
c.println(name4);
}
if (name5.compareTo("A") >= 0 && name5.compareTo("B") <= 0) {
c.println(name5);
}
}
}
```

2. Consider the previous program:
   a. Can it be written using a `switch` statement? Why or why not?
   b. What does the program do?
   c. Every `if` statement is executed when the program runs. Why is this necessary? How would the program behave differently if, starting on the second `if`, we wrote `else if` instead of just `if`? How might we have introduced a bug in the program by doing so?

3. Create a small Java program that uses a multiway-`if` with four conditions and an `else` statement at the end. Then, rewrite the program using a `switch` statement. Both programs must achieve the very same result. Be sure to have the correct indentation.

2.6 The alternative short `if`

Java, C and C++, all have a special short form of the `if` statement. In its simplest form, the alternative short `if` can be used when

1. The if and the else are present, ***and***
2. There is only one statement inside each of the if and the else[1], ***and***
3. The statement is an assignment[2]

Here is an example. The code

[1] Multiple statements are also possible

[2] `return` statements also accept this form

```
if (mark >= 50) {
    grade = "PASS";
}
else {
    grade = "FAIL";
}
```

can be written in one statement as follows,

```
grade = mark >= 50 ? "PASS" : "FAIL";
```

The variable `grade` will be assigned the string `PASS` if `mark` is larger or equal to 50. Otherwise `grade` will be assigned the string `FAIL`. Here is another example. The code

```
if (name.compareTo("M") >= 0) {
    inAlphabet = "Above L";
}
else {
    inAlphabet = "Below M";
}

inAlphabet = name.compareTo("M") >= 0 ? "Above L" : "Below M";
```

The general form of the short `if` construct is

```
variable = condition ? value if condition true : value if false;
```

2.6 Exercises

1. For the following, assume that the value of `int number` is 19. Predict the value of `int result` when each of these is executed:

   a) `result = number == 19 ? 0 : 1;`
   b) `result = number >= 10 && number <= 35 ? -100 : 100;`
   c) `result = (-1 * number) < -10 ? number : 2 * number;`
   d) `result = number % 4 < 2 ? 0 : number % 4;`
   e) `result = number / 2 == 9 ? number + 1 : number - 1;`

2. Rewrite the following code using the short `if` form:

```
if (xCoordinate <= 0) {
    xCoordinate = 0;
}
else {
    xCoordinate = xCoordinate - 1;
}
```

3. Rewrite the following code using the short `if` form:

```
if (phone.equals("416-393-1420")) {
    school = "BLOOR CI";
}
else {
    school = "OTHER";
}
```

4. Rewrite exercises 1a), 1c), and 1e) using the standard `if` form.