Chapter 6. Conditional Processing

A key part of any computer language is the ability to direct the execution *(or not)* of specific sections/blocks of code based on the result of boolean conditions. This chapter exercises your skill in using conditional processing to direct the execution of specific blocks of code by using the conditional processing statements `if`, `if/else`, `else` as well as the `switch`, `case`, `break` and `default` statements.

Conditional Operators: if *and* if/else

1. I have two integer variables, `x` and `y`, with values `5` and `7` respectively. Compare the two values and on comparison, print a statement that states which is greater. Use only `if` conditions to do the comparisons.

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

    boolean b01;
    boolean b02;
    int x = 5;
    int y = 7;

    b01 = x > y;
    b02 = x < y;

    if (b01 == true)
    {
      System.out.printf("%d is greater
                        than %d", x, y);
    }
    if (b02 == true)
    {
      System.out.printf("%d is greater
                        than %d", x, y);
    }
  }
}
```

OR

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

    int x = 5;
    int y = 7;

    if (x > y)
    {
      System.out.printf("%d is greater than
                         %d", x, y);
    }
    if (y > x)
    {
      System.out.printf("%d is greater than
                         %d", x, y);
    }
  }
}
```

Notes: On comparing the two solutions, it should be observed that the `boolean` variables `b01` and `b02` are not required, as the statement **`(x > y)`** or the statement **`(y > x)`** is a self-contained statement that returns a boolean result of `true` or `false`. The statement **`if(<whatever>)`** really means "**`if(<whatever is in this bracket is true>)`**, then run the commands in the braces following the `if`".

2. I have two floating point variables, `x` and `y`, with values `5.77` and `7.83` respectively. Compare the two values and if they are equal state so, otherwise state that they are not equal. Use an `if/else` construct.

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

    float x = 5.77f;
    float y = 7.83f;

    if (x == y) {
      System.out.println("The numbers are equal");
    }
    else{
      System.out.println("The numbers are not equal");
    }
  }
}
```

Explanation: the `else` condition can be described as follows:
Whatever is false about the `if` condition is handled by the `else` condition. Recall from the earlier example that **`if(<whatever>)`** means **`if(<whatever is in this bracket is true>)`**. The `else` then handles whatever failed

the `if` question.

3. I have two integer variables, `x` and `y`. Write a program which compares them and on comparison prints which is greater, otherwise if they are equal then the program should state that they are equal. Hardcode the values `5` and `7` for `x` and `y` to test your program.

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

    int x = 5;
    int y = 7;

    if (x > y) {
      System.out.printf("%d is > %d",x, y);
    }
    if (y > x){
      System.out.printf("%d is > %d", y, x);
    }
    if (x == y){
      System.out.printf("%d = %d", x, y);
    }
  }
}
```

OR

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

    int x = 5;
    int y = 7;

    if (x > y){
      System.out.printf("%d is > %d", x, y);
    }
    else if (y > x){
      System.out.printf("%d is > %d", y, x);
    }
    else if (x == y){
      System.out.printf("%d = %d", x, y);
    }
  }
}
```

4. What is the difference between the two solutions shown for the preceding exercise?

The 1st solution goes through and evaluates each of the conditions in the respective `if(<condition>)` statements. However, in this exercise where each of the conditions is mutually exclusive, there is no real need to test subsequent related conditions if a preceding condition has been evaluted to be true (for example in this particular exercise since we already know that `x > y` then there is no point in checking whether `y > x`, or whether `x == y`). Therefore, the 2nd solution which uses the `else if` conditions which are not evaluated if a preceding condition is true is more efficient.

5. Write a program, which based on a value given for the temperature gives the following output:
   1. If the temperature < 60 then output *"Cold enough to wear a coat"*
   2. If the temperature ≥ 60 but < 68 then output *"Cold enough to wear a jacket"*
   3. Otherwise simply output *"No outerwear required!"*

For testing purposes, hardcode the temperature value into the program.

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

    float temperature = 0f;
    // Set it to an initial value for our code. We can change it and recompile
    // in order to test the code for different temperatures.

    if (temperature < 60) {
      System.out.println("Cold enough to wear a coat");
    }
    else if (temperature < 68) {
      //We've already handled whatever is less than 60
      //so this is implicitly 60 to just under 68
      System.out.println("Cold enough to wear a jacket");
    }
    else {
      System.out.println("No outerwear required!");
    }
  }
}
```

Combining multiple conditions

6. Given two numerical variables `x` and `y`, explain what the following code means:

```
if ((x > 5) && (y > 20))
 {
   // Perform action...
 }
```

The code means, if `x` is greater than 5 ***AND*** `y` is greater than 20 then perform the actions in the braces.

7. Given two numerical variables `x` and `y`, explain what the following code means:

```
if ((x > 5) || (y < 20))
 {
   // Perform action...
 }
```

The code means, if `x` is greater than 5 ***OR*** `y` is less than 20 then perform the actions in the braces.

Note: Of course you know that logical OR means "at least one of the conditions is valid".

8. Given three numerical variables `x, y` and `z`, give an explanation of what the following code means:

```
if ((x > 5) || (y > 20) || (z == 0))
 {
   // Perform action...
 }
```

The code means, if any of `x` is greater than 5 ***OR*** `y` is greater than 20 **OR** `z` equals 0 then perform the actions in the braces.

Note: see the note in the preceding solution.

9. Given three integers `x, y` and `z`, give an explanation of what the following code means (pay close attention to the placement of the braces):

```
if ((x > 5) && ((y > 20) ^ (z == 0)))
 {
   // Perform action...
 }
```

The code is first evaluating two different sets of conditions separately, namely:

`(x > 5)`
AND
`((y > 20) ^ (z == 0))`

The `&&` condition is stating that "if both of the separate conditions are true", then perform the actions in the braces.

Note: Recall that `^` (XOR) means that only one of the conditions can be true. Contrast this with the functioning of OR (see the preceding exercise).

10. Rewrite this code fragment using the ternary operator:

```
int x = 50, y = 0;
if (x > 10)
  y = 5;
else
  y = 27;
System.out.println(y);
```

```
int x = 50, y = 0;
y = x > 10 ? 5 : 27;
System.out.println(y);
```

Note: You can think about the ternary operator in the following way:

```
value = question(the result is either true or false)? value if true: value if false;
```

The `switch` and `case` statements

11. Explain your understanding of how the `switch` statement works. Give an example that uses a `switch` statement.

A `switch` statement is a conditional processing construct which, based on the value of a variable under assessment, will execute specific blocks of code called *cases* according to the value of the variable. A `switch` construct uses the keywords `switch`, `case`, `break` and `default` to determine, bound and exit particular blocks of code within the switch construct. Each *case* is bounded by the keywords `case` and `break`. Any condition for which we do not have a `case/break` block can be handled by a `default` block at the end of the `switch` block.

For example, see the following code block:

```
int zz=10;
switch(zz)
{
  case 5:
     System.out.println("zz is equal to 5");
  break; // means end of this particular block. You exit the case (& thus the switch block)
         // at break statements.
  case 6:
  case 7:
     System.out.println("zz is equal to 6 or 7");
     // Observe that we can combine multiple cases in one block.
  break;
  case 8:case 9:case 10:
     System.out.println("zz is equal to 8, 9 or 10!");
  break;
  default:
  System.out.println("Default covers anything not covered by a case!");
}
```

12. Explain the output of the following switch block, given that `zz=5` and explain why it is so.

```
int zz=5;
switch(zz)
{
  case 5:
     System.out.println("zz is equal to 5");
  case 6:
  case 7:
     System.out.println("zz is equal to 6 or 7");
     // See that we can combine multiple cases in one block.
  break;
  default:
  System.out.println("Default covers anything not covered by a case!");
}
```

Output:

```
zz is equal to 5
zz is equal to 6 or 7
```

Reason: There was no `break` statement at the end of case 5; when no such statement appears, the switch simply keeps processing within the `switch` block until either it sees either a `break`, a `default` statement or the end of the block!
Sometimes this effect is intended, but in the case that it isn't we must be careful to put the `break` statement in.

13. Write a program, using a `switch` block to print out how many days any given month (specified as a string) has. Hardcode values with which to test your program.

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

    String month = "jan";
    month = month.toLowerCase(); // For uniform processing
    month = month.substring(0, 3); // 3-leftmost characters
    switch(month) {
      case "jan": case "mar": case "may": case "jul": case "aug": case "oct": case "dec":
        System.out.printf("This month has 31 days");
        break;
      case "apr": case "jun": case "sep": case "nov":
        System.out.println("This month has 30 days");
        break;
      case "feb":
        System.out.println("This month has 28 days");
        break;
      default:
        System.out.println("Invalid month. Enter at least the first 3 letters of a month");
      }
    }
```

```
  }
```

14. At our hotel, we have a reward card for frequent guests. There are three card levels, namely gold, silver and bronze, with bronze being the lowest level.

The benefits of each of the levels is as follows:

- bronze = free parking + newspaper
- silver = bronze level features + breakfast
- gold = silver level features + dinner for one

If you do not have a reward card, then by default all you get is a room.

Write a program which on assessing a `String` variable that contains the card level, prints out, using a `switch` block, all the benefits of each level. Ensure that you do not replicate code.

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

    String customerLevel = "bronze"; //hardcoded test value
    customerLevel = customerLevel.toLowerCase(); //for uniformity in processing

    System.out.println("Your benefits are:");
    switch (customerLevel)
    {
       case "gold":
          System.out.println("\t included dinner for 1");
       case "silver":
          System.out.println("\t included breakfast");
       case "bronze":
          System.out.println("\t free parking");
          System.out.println("\t included newspaper");
       default:
          System.out.println("\t room");
    }
  }
}
```

Note: Observe that to print out the aggregate benefits of each level, we put the cases in a particular order to take advantage of the fall-through functionality provided when we do not explicitly put a `break` in each `switch` block.