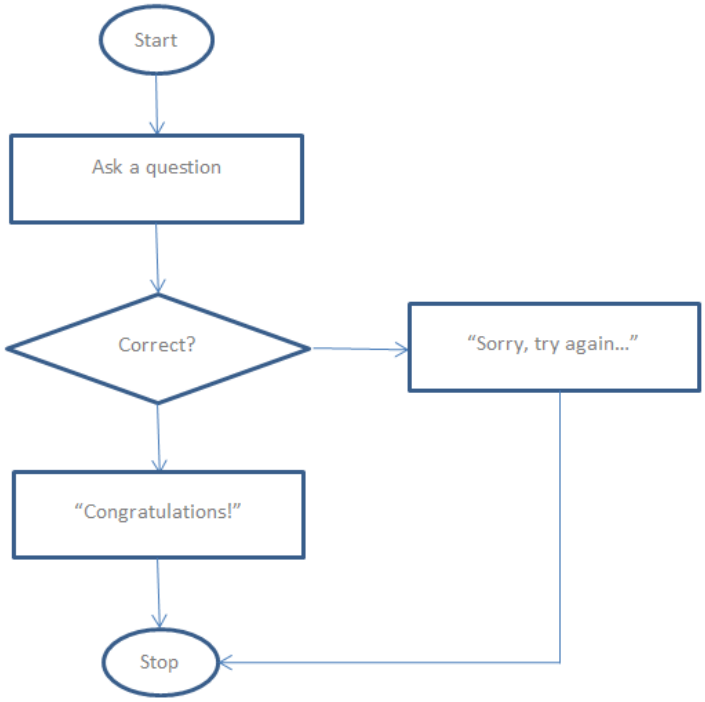
Making Decisions in Java

Why Computer Programs need to make Decisions

Assume you are writing a very simple quiz program, where the user is asked a question. If the user answers the question correctly, she is congratulated, otherwise she is asked to try again.

This scenario can be graphically represented as a flowchart:



We can also represent the game in pseudocode:

1. Start
2. Ask user a question
3. Prompt user for an answer
4. IF the answer is correct, display the message "Congratulations!"
   ELSE display the message "Sorry, wrong answer. Please try again"
5. Stop

Making Decisions in Java

Like most high-level languages, Java uses a structure known as the `if…else` structure for decision making. The syntax for the `if…else` statement is as follows:

```
if(a == b)
{
    <statement 1>
}
else
{
    <statement 2>
}
```

The code explained:

Firstly, note that the expression `a == b` is not and assignment statement. We use a single equals sign for assignment, as shown in the following example:

```
a = 5
total = 100
```

Expression `a == b` is what we call a **comparison statement**. In other words, it is comparing the values `a` and `b` to see if they are equal. The double equals sign (==) is known as a **relational operator**.

Based on the result of the comparison, Java will execute either `statement 1` or `statement 2`. If the comparison returns true, `<statement 1>` will be executed, and `<statement 2>` will be completely ignored.

On the other hand, if the comparison returns false, `<statement 1>` will be skipped and `<statement 2>` will be executed.

`Statement 1` is known as the *true* branch, and `Statement 2` is known as the *false* branch.

You will understand all this much more clearly via an example. Below is the complete Java code for the simple quiz game which we discussed above:

```
1 import javax.swing.JOptionPane;
2
3 public class QuizGame
4 {
5    public static void main(String[] args)
6    {
7       String answerString;
8       int answerInt;
9       String name = "Tom";
10
11      answerString = JOptionPane.showInputDialog("What is 5 + 4?");
12      answerInt = Integer.parseInt(answerString);
13
14      if(answerInt == 9)
15      {
16         JOptionPane.showMessageDialog(null, "Congratulations!\nCorrect answer");
17      }
18      else
19      {
20         JOptionPane.showMessageDialog(null, "Sorry, wrong answer\nPlease try again.");
21      }
22      System.exit(0);
23   }
24 }
```

The code explained:

- Line 7 and 8: In these lines we declare a `String` to store the input from the user, and an `int` to store the answer
- Line 10 and 11: In these lines we are getting a `String` as input from the user, and converting the `String` into an integer.

If you are uncertain about what is going on in the previous lines, take a look at Section 3.5 for a full explanation.

- Line 13: In this line the user's answer (stored in `answerInt`) is being compared to the correct answer, which is "9". If the value entered by the user is equal to 9, then line 15 is executed, otherwise line 19 is executed.

Activity 4.1
Modify the **QuizGame** program to ask a different math question.

Boolean Expressions

As mentioned before, the statement within the braces after the `if` keyword is always a comparison, and always returns one of two values: true or false. The term used to describe statements like this is **Boolean Expression**.

Boolean expressions are put together using relational operators. We've already seen one relational operator, the double equals (==) operator, which checks for equality. The other relational operators are:

<	Less than	4 < 5
>	Greater than	a > b
<=	Less than or equal	4 <= (a + b)
>=	Greater than or equal	b >= 9

!=	Not equals	3 != 4

Activity 4.2
Take a look at the following Boolean expressions:

1. 6 < 7 returns true
2. (9 * 2) > (6 * 3) returns false because 9 * 2, which equates to 18, is not greater than 6 * 3, which also equates to 18. In other words, 18 is not greater than 18.
3. 4 + 2 != 3 * 4 returns true

Try to determine what the following Boolean expressions will evaluate to:

1. 5 == 3 + 2
2. 4 * 6 >= 45 - 14
3. (((2 * 5) / (18 – 15)) * 2) >= ((120 / 25) – (63 % 2))

Comparing Strings

The double equals operator works fine for numbers, but you cannot use it to compare Strings. If you attempt to use it to compare Strings, as in the following code snippet:

```
String name = "Tom";
if(name == "Tom")…
```

This will be legal as far as Java goes, but the results you get will not be what you would expect. The correct way to compare Strings is to use the `equals()` method:

```
if(name.equals("Tom"))…
```

Nested if-else Statements

You can nest an if statement within another:

```
1 if(a == b)
2 {
3     if(c == d)
4     {
5         do something
6     }
7 }
```

The `if` statement in line 3 is 'nested' within the main `if` statement. You may wonder why you would ever need to do this; it does have its uses though, as in the following example:

Suppose you need to write a program which takes in a student's percentage mark, and allocate a symbol corresponding to that mark. The symbols will be allocated as follows:

- Mark greater than 79% - A
- Mark between 69% and 80% - B

- Mark between 59% and 70% - C
- Mark between 49% and 60% - D
- Mark between 39% and 50% - E
- Mark less than 40% - F

The pseudo code will look like this:

```
IF the mark is greater than 79%
        The symbol is A
ELSE
        IF the mark is greater than 69%
                The symbol is B
        ELSE
                IF the mark is greater than 59%
                        The symbol is C
                ELSE
                        IF the mark is greater than 49%
                                The symbol is D
                        ELSE
                                IF the mark is greater than 39%
                                        The symbol is E
                                ELSE
                                        IF the mark is less than 40%
                                                The symbol is F
```

The pseudo code can be translated into Java as follows:

```
int mark;
char symbol;

<CODE TO READ IN MARK GOES HERE>

if(mark > 79)
   symbol = 'A';
else
   if(mark > 69)
      symbol = 'B';
   else
      if(mark > 59)
         symbol = 'C';
      else
         if(mark > 49)
            symbol = 'D';
         else
            if(mark > 39)
               symbol = 'E';
            else
               if(mark < 40)
                  symbol = 'F';
```

Did you notice that in this example, unlike the previous ones, there are no curly braces after the lines with the `if` statements. This is possible where there is only one statement to be executed after the `if` statement, although it is always best practice to use the curly braces.

Activity 4.3
Complete the above program

The Logical Connectives

Often the scenario will arise where two or more conditions have to be fulfilled in order for a Boolean expression to be true, that is:

IF ((a == b) **AND** (c == d))

That is, the above expression will only evaluate to true if both (a==b) is true *and* (c == d) is true. If either one is false, the entire expression will be false.

An example of this is a user login system for a website. A user will be granted access to the site only if he inputs the correct username and the correct password. If only one is correct, it will not be good enough; both will have to be correct in order to gain access.

The pseudo code will look like this:

```
if the username is correct AND the password is correct
  grant access
else
  deny access
```

Done in Java, it will look like this:

```
if((username.equals("tron") && (password.equals("abc1234"))
{
    //grant access
}
else
{
    //do not grant access
}
```

The double ampersand above performs the **AND** function, and is known as a **logical connective**. Logical connectives are used to join Boolean expressions to make larger, more complex Boolean expressions.

Another logical connective you will encounter is the **OR** connective. With the **OR** connective:

IF ((a == b) **OR** (c == d))

The above expression will evaluate to true if either (a == b) is true **OR** (c == d) is true. They don't both have to be true.

Let's look at an example. A restaurant gives a 25% discount to children under the age of 12 and to pensioners who are over 60 years old. So if a person fulfils any one of the following conditions, he will be entitled to a 25% discount:

a) Younger than 12 years old
b) Older than 60 years old

Written in pseudo code:

If customer's age is less than 12 OR customer's age is greater than 60
    Give a 25% discount
Else
    Do not give a discount

This can be translated into Java as follows:

```
int age;

<CODE TO READ IN AGE GOES HERE>

if((age < 12) || (age > 60))
{
   //Give a 25% discount
}
else
{
   //Do not give a discount
}
```

The OR operator is represented in Java with two pipe symbols (||). The pipe symbol is typed by pressing the shift button and the backspace button together (SHIFT + \) on your keyboard.

Activity 4.4
The examples given above for the AND and OR operators are code snippets, not complete Java programs.

Write complete Java programs which incorporate the code in the examples.

Operator Precedence Revisited

In the previous chapter we looked at operator precedence, which determines the order in which the mathematical operators are evaluated. Have you wondered where the relational operators and logical connective discussed in this chapter fit into the precedence levels?

The table below shows all the operators you've covered thus far, and there precedence levels:

Operator Precedence	
1	()
2	!
3	* / %
4	+ -
5	< <= > >= ==
6	++ !=
7	&&
8	||

Activity 4.5
Consider the following declarations:

```
int a = 3;
int b = 4;
int c = 5;
```

Now work out what the values of the following expressions will be:

1. (a <= 1) && (b > 3)
2. (a == 1) || (b >= 5)
3. !(a == 1) && (b > a)
4. !((a < c) && ((b >= c) || !(a == b)))