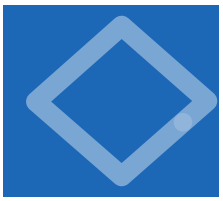


# 1.3 Conditionals and Loops

In the programs that we have examined to this point, each of the statements in the program is executed once, in the order given. Most programs are more complicated because the sequence of statements and the number of times each is executed can vary. We use the term *control flow* to refer to statement sequencing in a program. In this section, we introduce statements that allow us to change the control flow, using logic about the values of program variables. This feature is an essential component of programming.



Specifically, we consider Java statements that implement *conditionals*, where some other statements may or may not be executed depending on certain conditions, and *loops*, where some other statements may be executed multiple times, again depending on certain conditions. As you will see in numerous examples in this section, conditionals and loops truly harness the power of the computer and will equip you to write programs to accomplish a broad variety of tasks that you could not contemplate attempting without a computer.

**If statements** Most computations require different actions for different inputs. One way to express these differences in Java is the `if` statement:

```
if (<boolean expression>) { <statements> }
```

This description introduces a formal notation known as a *template* that we will use to specify the format of Java constructs. We put within angle brackets (`< >`) a construct that we have already defined, to indicate that we can use any instance of that construct where specified. In this case, *<boolean expression>* represents an expression that has a boolean value, such as one involving a comparison operation, and *<statements>* represents a *statement block* (a sequence of Java statements, each terminated by a semicolon). This latter construct is familiar to you: the body of `main()` is such a sequence. If the sequence is a single statement, the curly braces are optional. It is possible to make formal definitions of *<boolean expression>* and *<statements>*, but we refrain from going into that level of detail. The meaning

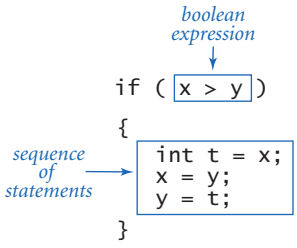
of an `if` statement is self-explanatory: the statement(s) in the sequence are to be executed if and only if the expression is `true`.

As a simple example, suppose that you want to compute the absolute value of an `int` value `x`. This statement does the job:

```
if (x < 0) x = -x;
```

As a second simple example, consider the following statement:

```
if (x > y)
{
   int t = x;
   x = y;
   y = t;
}
```




*Anatomy of an `if` statement*

This code puts `x` and `y` in ascending order by exchanging them if necessary.

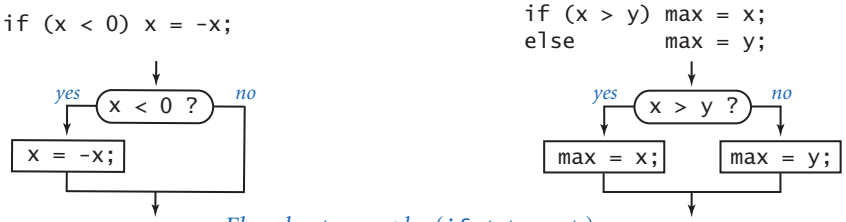
You can also add an `else` clause to an `if` statement, to express the concept of executing either one statement (or sequence of statements) or another, depending on whether the boolean expression is `true` or `false`, as in the following template:

```
if (<boolean expression>) <statements T>
else                      <statements F>
```

As a simple example of the need for an `else` clause, consider the following code, which assigns the maximum of two `int` values to the variable `max`:

```
if (x > y) max = x;
else       max = y;
```

One way to understand control flow is to visualize it with a diagram called a *flowchart*. Paths through the flowchart correspond to flow-of-control paths in the pro-




*Flowchart examples (`if` statements)*

| | |
|---|---|
| *absolute value* | `if (x < 0) x = -x;` |
| *put* x *and* y *into sorted order* | `if (x > y)`<br>`{`<br>`  int t = x;`<br>`  y = x;`<br>`  x = t;`<br>`}` |
| *maximum of* x *and* y | `if (x > y) max = x;`<br>`else  max = y;` |
| *error check for division operation* | `if (den == 0) System.out.println("Division by zero");`<br>`else  System.out.println("Quotient = " + num/den);` |
| *error check for quadratic formula* | `double discriminant = b*b - 4.0*c;`<br>`if (discriminant < 0.0)`<br>`{`<br>`  System.out.println("No real roots");`<br>`}`<br>`else`<br>`{`<br>`  System.out.println((-b + Math.sqrt(discriminant))/2.0);`<br>`  System.out.println((-b - Math.sqrt(discriminant))/2.0);`<br>`}` |

*Typical examples of using* `if` *statements*

gram. In the early days of computing, when programmers used low-level languages and difficult-to-understand flows of control, flowcharts were an essential part of programming. With modern languages, we use flowcharts just to understand basic building blocks like the `if` statement.

The accompanying table contains some examples of the use of `if` and `if-else` statements. These examples are typical of simple calculations you might need in programs that you write. Conditional statements are an essential part of programming. Since the *semantics* (meaning) of statements like these is similar to their meanings as natural-language phrases, you will quickly grow used to them.

PROGRAM 1.3.1 is another example of the use of the `if-else` statement, in this case for the task of simulating a coin flip. The body of the program is a single statement, like the ones in the table above, but it is worth special attention because it introduces an interesting philosophical issue that is worth contemplating: can a computer program produce *random* values? Certainly not, but a program *can* produce numbers that have many of the properties of random numbers.

### *Program 1.3.1 Flipping a fair coin*

```
public class Flip
{
   public static void main(String[] args)
   {  // Simulate a coin flip.
      if (Math.random() < 0.5) System.out.println("Heads");
      else                     System.out.println("Tails");
   }
}
```

*This program uses* `Math.random()` *to simulate a coin flip. Each time you run it, it prints either heads or tails. A sequence of flips will have many of the same properties as a sequence that you would get by flipping a fair coin, but it is not a truly random sequence.*

```
% java Flip
Heads
% java Flip
Tails
% java Flip
Tails
```

**While loops** Many computations are inherently repetitive. The basic Java construct for handling such computations has the following format:

```
while (<boolean expression>) { <statements> }
```

The `while` statement has the same form as the `if` statement (the only difference being the use of the keyword `while` instead of `if`), but the meaning is quite different. It is an instruction to the computer to behave as follows: if the expression is `false`, do nothing; if the expression is `true`, execute the sequence of statements (just as with `if`) but then check the expression again, execute the sequence of statements again if the expression is `true`, and *continue* as long as the expression is `true`. We often refer to the statement block in a loop as the *body* of the loop. As with the `if` statement, the braces are optional if a `while` loop body has just one statement. The `while` statement is equivalent to a sequence of identical `if` statements: