JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris July 2001

1 FOR LOOPS

1.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- state the essential elements of loop control
- understand how *for* loops function

1.2 INTRODUCTION

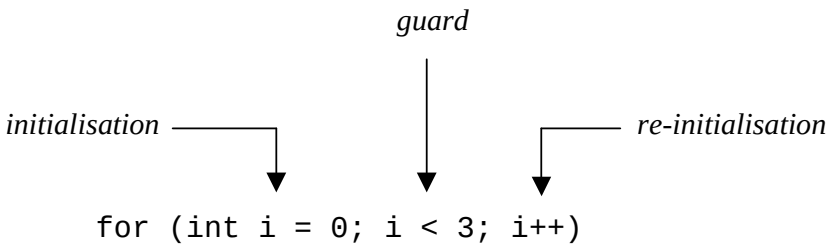
Iterations are an inescapable part of programming. In Part One - Fundamentals - we looked at *while* loops. In this chapter we look at *for* loops.

1.3 FOR LOOPS

In Part One we used *while* loops. This code fragment uses a *while* loop to sum the first three positive integers.

```
int sum = 0;
int i = 0;          <-- initialisation
while (i < 3) {     <-- guard
  sum += i;
  i++;              <-- re-initialisation
}
```

In a *while* loop the control elements (initialisation, guard and re-initialisation) are scattered. In a *for* loop construct the loop control elements are localised in one line.



Each loop control element is separated from the next by a semi-colon. Initialisation is done just once. The guard controls entry into the loop and is tested at the beginning of every loop. The loop is executed for as long as the guard remains true. Re-initialisation is done at the end of every loop.

The code fragment to sum the first five positive integers using a *for* loop is

```
int sum = 0;
for (int i = 0; i < 3; i++) {
  sum+= i;
}
```

Initially, *i* is zero.
i is less than three and so we go into the loop body where *i* is added to *sum*.
Then *i* is incremented; it is now one.

i is still less than three and so we go into the loop body where *i* is added to *sum*.
Then *i* is incremented again; it is now two.

i is still less than three and so we go into the loop body where *i* is added to *sum*.
Then *i* is incremented again; it is now three.

i is no longer less than three and so the loop terminates.

The pattern of a *for* loop is

```
for (initialisation; guard; re-initialisation) {
  statements to be repeatedly executed
}
```

Notice that there is no semi-colon following the *for(initiliasation, guard, re-initialisation)* line.

The brackets may be omitted if there is just one statement to be repeatedly executed.

1.4 REVIEW

1.5 FURTHER READING

In the next lesson we use *for* loops to process arrays.

1.6 EXERCISES

1 Explain each line of the code fragments shown below.

(a)
```
for (int i = 0; i < 3; i++) {
  System.out.println("Help");
}
```

(b)
```
int hour = 0;
for (int pop = 1; pop < 100; pop = pop * 2) {
  hour++;
}
System.out.println("The population reached 100 in " +
                   hour + " hours");
```

2 Dry run each of the code fragments shown below.

(a)
```
for (int i = 0; i < 5; i++) {
  System.out.println((i * 2 + 1));
}
```

(b)
```
for (int i = 0; i < 20; i += 5) {
  System.out.print(" " + i);
}
```

(c)
```
for (int j = 5; j > 0; j--) {
  System.out.print("*");
}
```

(d)
```
for (int i = 0; i > 5; i = i * i) {
  System.out.print(" " + i);
}
```

(e)
```
int p = 4;
int fp = 2;
int pnm;
int m = 6;
for (int i = 1; i <= m; i++) {
  pnm = p + fp;
  fp = p;
  p = pnp;
}
```

3 Dry run the Java code fragment shown below.

```
int n = 1;
for (int val = 100; val > 10; val = val / 2) {
  n++;
}
```

4 Dry run the Java code fragment shown below.

```
int pond = 100;
int day = 1;
for (int algae = 2; 2 * algae < pond; algae = algae * 2) {
  day++;
}
```

5 Carefully dry run this fragment of Java code

```
for (int i = 0; i < 5; i++) {
  for (int j  = 0; j < i; j++ {
    System.out.print("*");
  }
  System.out.println();
}
```

6 What do you think will happen here?

```
for ( ; ; )
  System.out.println("Help");
```