11 ITERATIONS

Terry Marris 21 April 2001

11.1 OBJECTIVES

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

- the while construct
- determinate loops
- indeterminate loops

11.2 PRE REQUISITES

The student should be comfortable with dry runs, boolean expressions and simple arithmetic expressions.

11.3 WHILE

while is a Java keyword. It introduces a loop. What is a loop? A loop is a block of statements that are repeatedly executed for as long as some condition remains met.

An iteration is some operation that is repeated.

11.4 DETERMINATE LOOPS

With a determinate loop you know in advance how many times the block of statements is to be executed. The program shown below uses a *while* loop to print

```
Help
Help
Help
```

on the screen.

```
/* HelpHelpHelp.java
   Terry Marris  22 April 2001
*/

public class HelpHelpHelp {
  public static void main(String[] s)
  {
    int i = 0;
    while (i < 3) {
      System.out.println("Help");
      i++;
    }
  }
}
```

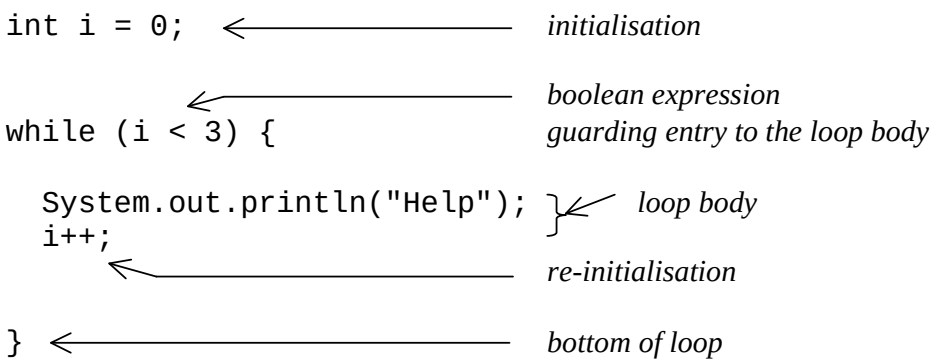
The *int i* is initialised to zero. Then for as long as *i* remains less than three, the block of statements

```
      System.out.println("Help");
      i++;
```

is repeatedly executed. Since with each execution *i* is incremented, there comes a time when it is no longer less than three, in which case the loop terminates.

11.5 ANATOMY OF A LOOP

Most loops have the following three elements: initialisation, boolean expression guarding entry to the loop body, re-initialisation.



The initialisation stage is done just once. It sets the loop control variable (*i* in our example) to a starting value.

Then the boolean expression guarding entry to the loop body is evaluated. The value stored in the loop control variable is tested. If the boolean expression is true, the entire loop body is executed.

The loop body contains a re-initialisation stage in which the loop control variable is updated.

Having reached the bottom of the loop we return to the boolean expression guarding entry to the loop body. If the boolean expression is still true, we execute the entire loop body again. If the boolean expression is false we terminate the loop by skipping to the statement following the bottom of the loop.

11.6 FLOW OF CONTROL

The initialisation stage is done just once. We repeatedly loop for as long as *i* remains less than three.

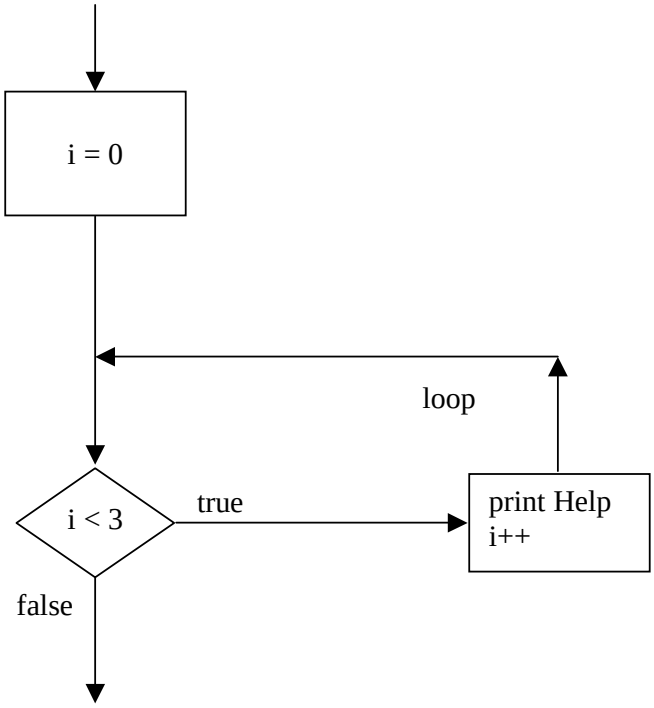


Figure 11.1 *Flow of control for a while loop*

11.7 INDETERMINATE LOOPS

With an indeterminate loop you do not know in advance how many times the statement block is to be executed.

The program shown below calculates how many hours it takes for a population of bacteria to reach 100 if it doubles in size every hour (like in a fridge at 4° C.)

```
/* PopulationGrowth.java
   Terry Marris  22 April 2001
*/

public class PopulationGrowth {
  public static void main(String[] s)
  {
    int hour = 0;
    int pop = 1;
    while (pop < 100) {
      hour++;
      pop = pop * 2;
    }
    System.out.println("The population reached 100 in " +
                        hour + " hours");
  }
}
```

pop is initialised to 1. We loop for as long as *pop* remains less than 100. Every time the loop body is executed we multiply *pop* by 2. So there eventually comes a time when *pop* is no longer less than 100. Then the loop terminates and the *println()* message is sent to the *System.out* object.

The output from the program is

```
The population reached 100 in 7 hours
```

To check this result we construct a table showing how *pop* changes each time the loop body is executed.

hour	pop	pop < 100
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

11.8 LAYOUT

Notice that

- the opening brace for the start of the block is on the same line as the *while(booleanExpression)*
- the block of statements to be repeatedly executed is indented under the while
- the closing brace fore the end of the block is vertically beneath the *w* in *while*.
- in general, there is no semi colon immediately following *while(booleanExpression)*

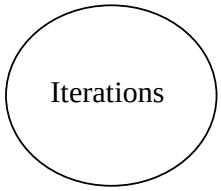
```
while (pop < 100) {
  hour++;
  pop = pop * 2;
}
```

11.9 FURTHER READING

HORSTMANN & CORNELL Core Java 2 Volume 1 Fundamentals pp 78.

11.10 REVIEW

block of statements repeatedly executed for as along as some condition is met



determinate loop - know in advance how many repetitions there are going to be

```
int i = 0;
while (i < 5) {
  System.out.println(i + " x 2 = " + (i * 2));
  i++;
}
```

indeterminate loop - do not know in advance how many repetitions there are going to be

```
int year = 1;
int activity = 100;
while (activity > 10) {
  year++;
  activity = activity / 2;
}
System.out.println("Activity decreased " +
                   " to 10 in " + year + " years");
```

11.11 EXERCISES

1 Dry run and state the output (if any) from each of the Java code fragments shown below.

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

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

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

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

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

2 Dry run the Java code fragment shown below.

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

3 Dry run the Java code fragment shown below.

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

4 Dry run this code fragment three times when **(a)** target = 3, **(b)** target = 4, **(c)** target = 5.

```
int target;
int lo = 0;
int hi = 9;
while (lo <= hi) {
  int mid = (lo + hi) / 2;
  if (mid == target)
    return "success";
  else if (target < mid)
    hi = mid - 1;
  else if (target > mid)
    lo = mid + 1;
}
return "failure";
```

5 Carefully dry run this fragment of Java code

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

6 Dry run this code fragment three times when **(a)** m = 9.0, **(b)** m = 4.0, **(c)** m = 2.0

```
double m;
double e = 1.0;
double a = m / 2.0;
double b = m;
double d = Math.abs(a - b);
while (d > e) {
  b = a;
  a = (b + m / b) / 2.0;
  d = Math.abs(a - b);
}
System.out.println(a);
```

Note: Math.abs(double) returns the absolute value of its argument, that is, removes the minus sign, if any, that its argument may have. For example, Math.abs(-7.0) returns 7.0. and Math.abs(7.0) also returns 7.0.