Chapter 7: Iterations

Objectives

Students should

- Be able to use Java iterative constructs, including *do-while*, *while*, and *for*, as well as the nested version of those constructs correctly.
- Be able to design programs that require the inclusion of iterations.

Repetitive Execution

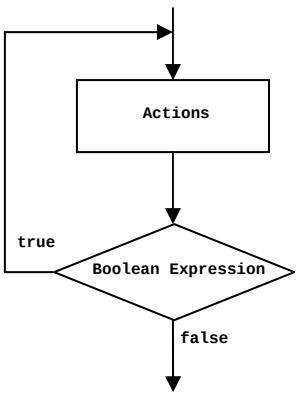
In writing most of useful computer programs, it is necessary to have the ability to execute a set of statements repeatedly for a certain number of iterations or until some conditions are met or broken. In Java, such ability is provided through three iterative constructs, namely *do-while*, *while* and *for* statements.

'do-while' Statement

A *do-while* statement is of the form:

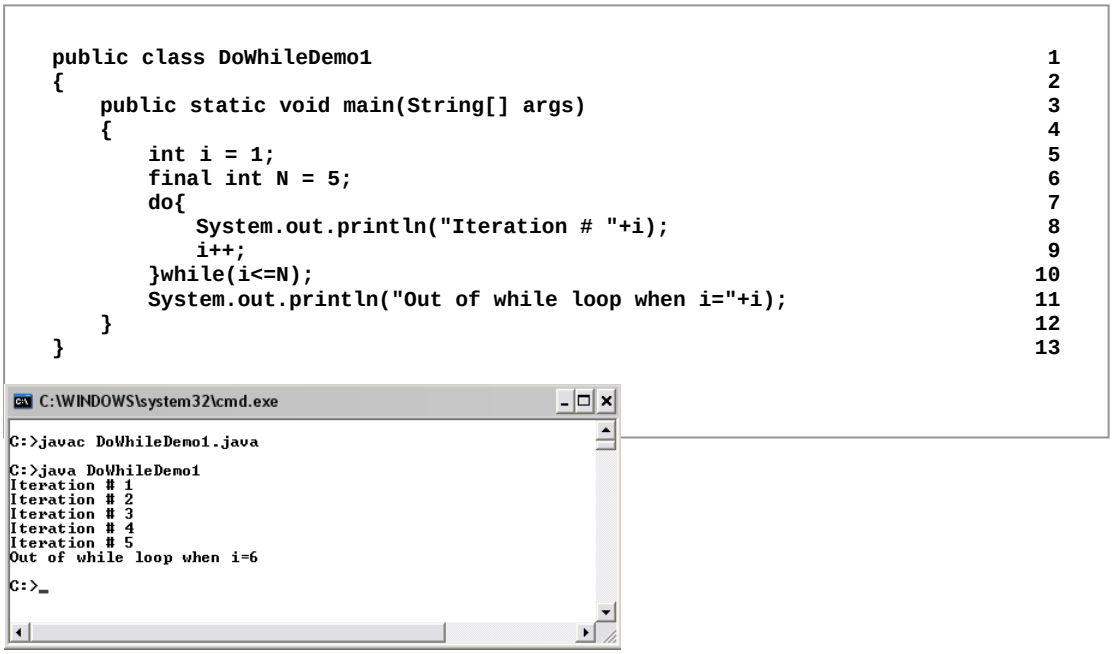
```
do{
    Actions
}while(Boolean Expression);
```

Its associated program flow can be shown in the following figure.



`Actions` can be one or more statements that will be repeatedly executed as long as **`Boolean Expression`** is evaluated to **`true`**. Once the program reaches the *do-while* statement, **`Actions`** will be execute first. Then, **`Boolean Expression`** is evaluated, and its value determines whether the program flow will loop back to repeat **`Actions`**, or finish the *do-while* statement.

Observe how it works by looking at the following program and its output.



```
public class DoWhileDemo1                                          1
{                                                                  2
    public static void main(String[] args)                        3
    {                                                              4
        int i = 1;                                                 5
        final int N = 5;                                           6
        do{                                                        7
            System.out.println("Iteration # "+i);                  8
            i++;                                                   9
        }while(i<=N);                                             10
        System.out.println("Out of while loop when i="+i);        11
    }                                                             12
}                                                                 13
```

```
C:\WINDOWS\system32\cmd.exe

C:>javac DoWhileDemo1.java

C:>java DoWhileDemo1
Iteration # 1
Iteration # 2
Iteration # 3
Iteration # 4
Iteration # 5
Out of while loop when i=6

C:>_
```

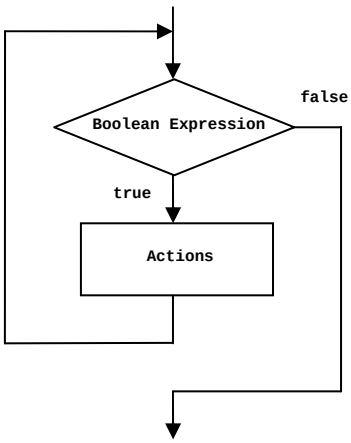
Initially, this program sets `i` to 1. This variable can be thought of as a counter of how many times the statements in the *do-while* block have already been executed. Each time this *do-while* block is entered, the program prints the iteration number from the variable `i`, which is increased by 1 at the end of every iteration (on line 9) before the program checks the **`boolean`** value of `i<=N`, where `N` equals 5. The program will exit the *do-while* statement after the 5th iteration, at the end of which the value of `i` is 6.

'while' statement

Another way to execute a set of statements repeatedly until a specified condition is met is to use a *while* statement. A *while* statement is of the form:

```
while(Boolean Expression){
    Actions
};
```

Its associated program flow can be shown in the following figure.

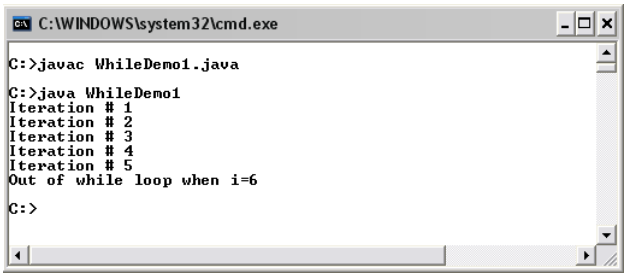


`Actions` can be one or more statements that will be repeatedly executed as in the *do-while* case. However, before the *while* block is entered, **`Boolean Expression`** is checked. If its value is

`false`, the statements in the *while* block will not be executed. If its value is true, **`Actions`** will be executed, and after that, the program flow loops back to checking **`Boolean Expression`**.

The following program (WhileDemo1.java) performs the same task as what is performed by the previous example (DoWhileDemo1.java) but with the use of a *while* statement instead of the *do-while* one.

```
public class WhileDemo1                                                  1
{                                                                        2
    public static void main(String[] args)                              3
    {                                                                    4
        int i = 1;                                                       5
        final int N = 5;                                                 6
        while(i<=N){                                                     7
            System.out.println("Iteration # "+i);                        8
            i++;                                                         9
        };                                                              10
        System.out.println("Out of while loop when i="+i);              11
    }                                                                   12
}                                                                       13
```

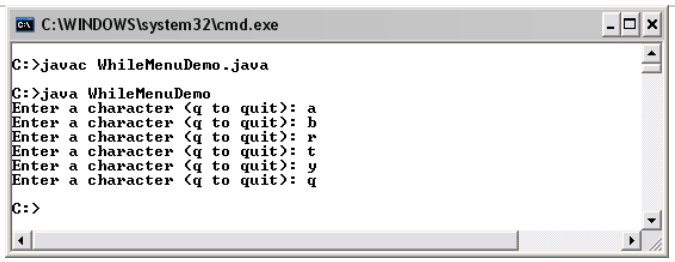


Both programs give out the same outputs. The only difference between using a do-while statement and a while statement is that the statements in the *do-while* block is always executed at least once since the condition checking is done after the *do-while* block, while the checking is done prior to ever entering the *while* block. Thus, the statements in the *while* block may never be executed.

Example

The following example how to use keep prompting for character input from the user repeatedly until a specified character is entered.

```
import java.io.*;                                                        1
public class WhileMenuDemo                                              2
{                                                                        3
    public static void main(String[] args) throws IOException           4
    {                                                                    5
        boolean done = false;                                            6
        char command;                                                    7
        BufferedReader stdin =                                           8
            new BufferedReader(new InputStreamReader(System.in));        9
        while(!done){                                                   10
            System.out.print("Enter a character (q to quit): ");       11
            command = stdin.readLine().charAt(0);                       12
            if(command == 'q') done = true;                             13
        }                                                               14
    }                                                                   15
}                                                                       16
```

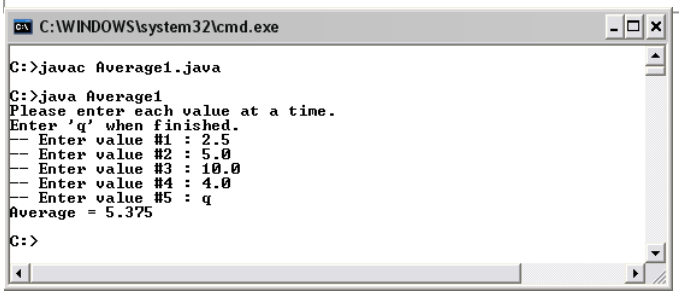


On line 6, a **boolean** variable called **done** is created and initialized to **false**. This variable is used in the condition checking of the *while* statement so that the statements in the *while* block will be iteratively executed as long as **done** is **false**. **done** has to be set to **true** at some point to avoid infinite loop (i.e. the situation when the iteration repeats forever!), and in this program, it is when the **char** value that the user enters equals **'q'**, as on line 13.

Example

The following example finds average of a number of values entered by the user. The program iteratively asks the user to enter each value at a time, until a character 'q' is entered.

```
import java.io.*;                                                          1
public class Average1                                                      2
{                                                                          3
    public static void main(String[] args) throws IOException             4
    {                                                                      5
        double sum = 0, i = 0;                                             6
        boolean doneInputing = false;                                      7
        String input;                                                      8
        BufferedReader stdin =                                             9
            new BufferedReader(new InputStreamReader(System.in));         10
        System.out.println("Please enter each value at a time.");         11
        System.out.println("Enter \'q\' when finished.");                 12
        while(!doneInputing){                                             13
            System.out.print("-- Enter value #"+(int)(i+1)+" : ");       14
            input = stdin.readLine();                                     15
            if((input.charAt(0) == 'q')&&(input.length()==1)){            16
                doneInputing = true;                                      17
            }else{                                                        18
                i++;                                                      19
                sum += Double.parseDouble(input);                         20
            }                                                             21
        }                                                                 22
        System.out.println("Average = "+(sum/i));                         23
    }                                                                     24
}                                                                         25
```



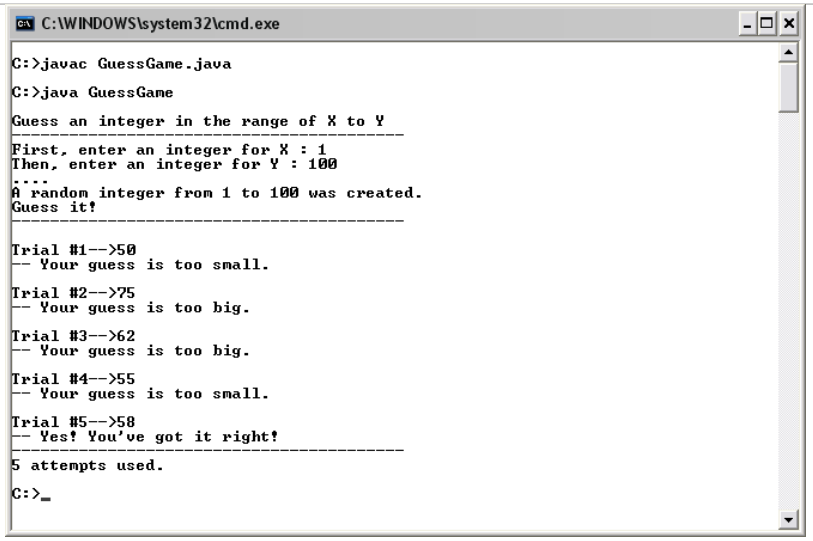
Try for yourself to write a rather similar program that finds the average of the values input by the user, but the program asks how many values will be entered first. Then, if the user specifies that there will be **n** values, the program iteratively prompts the user for each input **n** times before calculating the average of those values and shows the result on the screen. Use a *while* statement or a *do-while* statement.

Example

The following program is called GuessGame.java. The user of this program will play a game in which he/she needs to guess a target number, which is a number that the program has randomly picked in the range that the user chooses. The program will repeatedly prompt for the guessed number and provide a clue whether the guessed number is bigger or smaller than the target number, until the guessed number equals the target number.

Many things that we have learned so far are used in this program, including method calling, conditional constructs, data type casting, iterative constructs, and etc. Thus, you should observe the source code and be able to understand every statement used in this program.

```
import java.io.*;                                                          1
public class GuessGame                                                     2
{                                                                          3
    public static void main(String[] args) throws IOException              4
    {                                                                      5
        int x=0, y=0, target, nTrial=0, guess;                             6
        boolean validBound = false;                                        7
        boolean notCorrect = true;                                         8
        String comparison;                                                 9
                                                                          10
        BufferedReader stdin =                                            11
            new BufferedReader(new InputStreamReader(System.in));         12
        System.out.println("\nGuess an integer in the range of X to Y");   13
        System.out.println("----------------------------------------");    14
                                                                          15
        while(!validBound){                                               16
            System.out.print("First, enter an integer for X : ");        17
            x = Integer.parseInt(stdin.readLine());                       18
            System.out.print("Then, enter an integer for Y : ");         19
            y = Integer.parseInt(stdin.readLine());                       20
            if(y>x){                                                      21
                validBound = true;                                        22
            }else{                                                        23
                System.out.println("-- !! Y must be greater than X.");    24
            }                                                             25
        }                                                                 26
                                                                          27
        target = (int)Math.round(x+Math.random()*(y-x));                  28
                                                                          29
        System.out.println("....");                                       30
        System.out.println("A random integer from "+x+" to "+y+" was created.");
        System.out.println("Guess it!");                                  32
        System.out.println("----------------------------------------");    33
                                                                          34
        while(notCorrect){                                                35
            nTrial++;                                                     36
            System.out.print("\nTrial #"+nTrial+"-->");                   37
            guess = Integer.parseInt(stdin.readLine());                   38
            if(guess == target){                                          39
                notCorrect = false;                                       40
                System.out.println("-- Yes! You've got it right!");       41
            }else{                                                        42
                comparison = (guess>target)? "big.":"small.";             43
                System.out.println("-- Your guess is too "+comparison);    44
            }                                                             45
        }                                                                 46
                                                                          47
        System.out.println("----------------------------------------");    48
        System.out.println(nTrial+" attempts used.");                     49
                                                                          50
    }                                                                     51
}                                                                         52
```



'for' statement

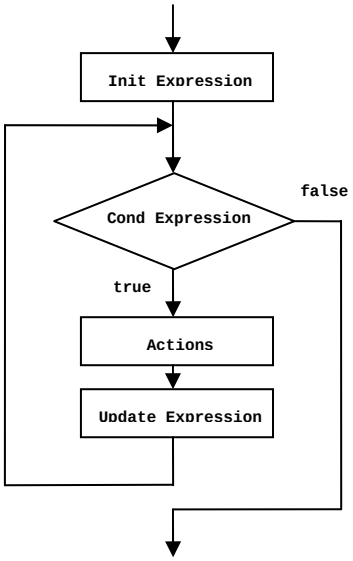
Another form of an iterative construct is the *for* statement, which is of the form:

```
for(Init Expression; Cond Expression; Update Expression){
    Actions
};
```

,which is equivalent to:

```
Init Expression;
while(cond Expression){
    Actions
    Update Expression;
};
```

Its associated program flow can be shown in the following figure.



The process starts with the execution of **`Init Expression`**. This is usually for assigning an initial value to a variable, which is often of type **`int`**. Then, the program checks whether **`Cond Expression`** is evaluated to **`true`**. If so, the program executes **`Actions`**. If not, the program goes out of the *for* statement. **`Cond Expression`** usually involves checking the value of the variable

initialized in **`Init Expression`**. Once the program finishes the execution of **`Actions`**, **`Update Expression`** is executed. **`Update Expression`** typically involves changing the value of the variables used in **`Cond Expression`**. Variables determining how many times **`Actions`** will be executed are called *index variables*.

Here are some examples of *for* loops.

```
for(int i=1; i<=10; i++){
    System.out.println(i);
}//for loop A

for(int i=10; i>0; i--){
    System.out.println(i);
}//for loop B

for(int i=0; i<=10; i += 2){
    System.out.println(i);
}//for loop C

for(int i=1; i<100; i *= 2){
    System.out.println(i);
}//for loop D

for(char c='A'; c<='Z'; c++){
    System.out.println(c);
}//for loop E
```

for loop A prints the values of **`i`** from 1 to 10. *for* loop B prints the values of **`i`**, starting from 10 down to 1. *for* loop C prints 0, 2, 4, 6, 8, and 10. *for* loop D prints 1, 2, 4, …, 32, 64. And, *for* loop E prints A to Z.

If needed, there can be more than one **`Init Expression`**'s as well as more than one **`Update Expression`**'s. Each of them is separated using commas. Look at such an example below. Notice that the initialization part contains two assignments, **`i=0`** and **`j=10`**, while the updating part contains **`i++`** and **`j--`**.

```
for(int i=0, j=10; i<=j; i++, j--){
    System.out.println(i+","+j);
}
```

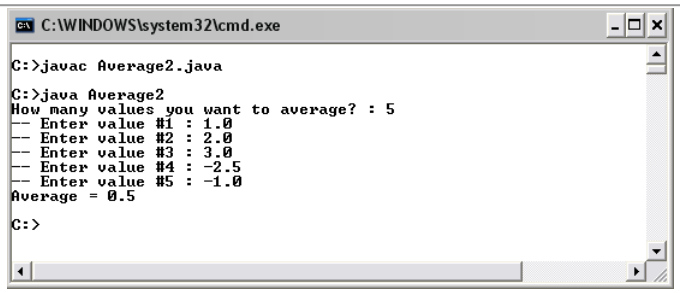
The above *for* loop causes the program to print:

```
0,10
1,9
2,8
3,7
4,6
5,5
```

Example

The following Java program finds the average of a number of values input by the user using a *for* loop. The number of values to be averaged is entered and stored in the variable **`n`** on line 11. Then, the *for* loop, starting on line 12, executes the statement located inside the loop (on line 13 and on line 14) for **`n`** iterations. Notice how the variable **`n`** and the variable **`i`** are used in order to iteratively execute the statements inside the loop **`n`** times.

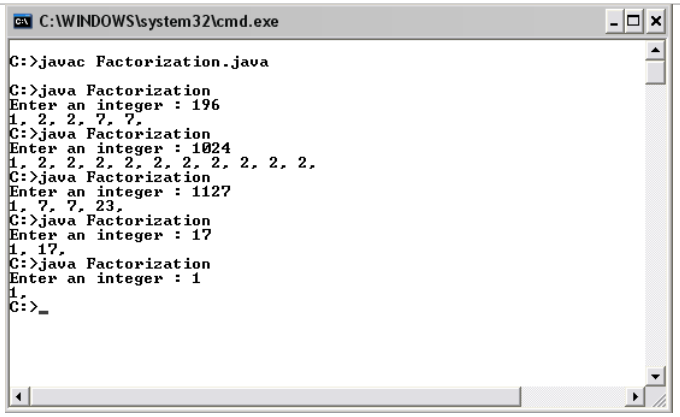
```
import java.io.*;                                                          1
public class Average2                                                      2
{                                                                          3
    public static void main(String[] args) throws IOException             4
    {                                                                      5
        double sum = 0, n = 0;                                             6
        String input;                                                      7
        BufferedReader stdin =                                             8
            new BufferedReader(new InputStreamReader(System.in));          9
        System.out.print("How many values you want to average? : ");     10
        n = Integer.parseInt(stdin.readLine());                           11
        for(int i=1;i<=n;i++){                                            12
            System.out.print("-- Enter value #"+i+" : ");                13
            sum += Double.parseDouble(stdin.readLine());                  14
        }                                                                 15
        System.out.println("Average = "+(sum/n));                         16
    }                                                                     17
}                                                                         18
```



Example

Let's look at a program called Factorization.java which is a program that finds prime factors of an integer (e.g. the prime factorization of 120 is 2×2×2×3×5). The algorithm used here (which is by no mean the best) is that we will iteratively factor the smallest factor out. Let the integer to be factorized be `n`. An integer `i` is a factor of `n` when `n%i` equals 0. A *for* loop is used to perform the iteration, starting from using the index variable of 2. In each iteration of the *for* loop, all factors equal the index variable are factored out via a *while* statement. In the program, a variable `m` is used for storing the resulting integer of the partially factorization. The *for* loop continues as long as the index variable `i` is still less than `m`.

```
import java.io.*;                                                          1
public class Factorization                                                 2
{                                                                          3
    public static void main(String[] args) throws IOException             4
    {                                                                      5
        int n, m;                                                          6
        BufferedReader stdin =                                             7
            new BufferedReader(new InputStreamReader(System.in));          8
        System.out.print("Enter an integer : ");                         9
        n = Integer.parseInt(stdin.readLine());                           10
        m = n;                                                            11
        System.out.print("1, ");                                          12
        for(int i=2;i<=m;i++){                                            13
            while(m%i == 0){                                              14
                System.out.print(i+", ");                                 15
                m = m/i;                                                  16
            }                                                             17
        }                                                                 18
    }                                                                     19
}                                                                         20
```



Example

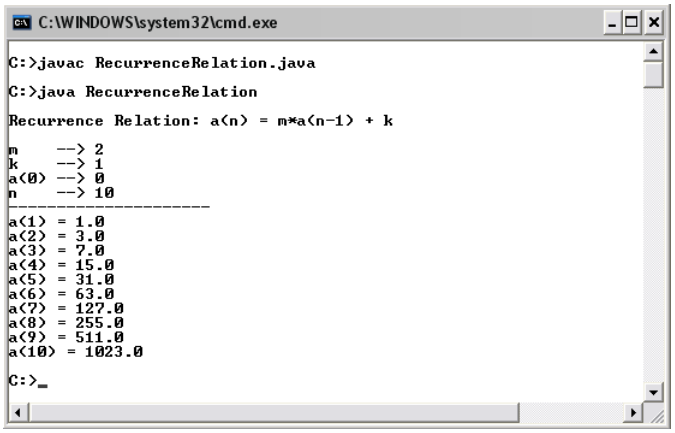
A sequence of number $\{a_n\} = a_0, a_1, a_2, a_3, \ldots$ can be defined using recurrent relation, in which the value of the n^{th} term of the sequence (a_n) is described based on preceding terms. For the first-order recurrent relation, the only preceding term required to compute the value of a_n is a_{n-1}. Thus, the first-order recurrent relation can be written as:

$$a_n = ma_{n-1} + k,$$

where m and k are two constant values defining the relation. We can compute the value of a_n in the sequence described by a first-order recurrent relation for any positive integer n from its initial condition, which is the value of a_0, and its associated recurrent relation.

The following program finds the value of a_1 to a_n for any positive integer n defined by the user from a first-order recurrent relation and its initial condition. A *for* loop is used for computing a_i from a_{i-1} for i equals 1 to the specified n.

```
import java.io.*;                                                    1
public class RecurrenceRelation                                     2
{                                                                    3
    public static void main(String[] args) throws IOException       4
    {                                                                5
        double an, an_1, k, m, a0;                                   6
        int n;                                                       7
        BufferedReader stdin =                                       8
            new BufferedReader(new InputStreamReader(System.in));    9
                                                                    10
        System.out.print("\nRecurrence Relation:");                 11
        System.out.println(" a(n) = m*a(n-1) + k\n");                12
                                                                    13
        System.out.print("m    --> ");                             14
        m = Double.parseDouble(stdin.readLine());                    15
        System.out.print("k    --> ");                             16
        k = Double.parseDouble(stdin.readLine());                    17
        System.out.print("a(0) --> ");                             18
        a0 = Double.parseDouble(stdin.readLine());                   19
        System.out.print("n    --> ");                             20
        n = Integer.parseInt(stdin.readLine());                      21
        System.out.println("---------------------");                22
                                                                    23
        an_1 = a0;                                                   24
        for(int i=1;i<=n;i++){                                       25
            an = m*an_1+k;                                           26
            System.out.println("a("+i+") = "+an);                    27
            an_1 = an;                                               28
        }                                                            29
    }                                                                30
}                                                                    31
```



'break' and 'continue'

We have seen in the previous chapter that a *break* statement causes the program to jump out of the current conditional construct and continue executing statements following that construct. If a *break* statement is put inside an iterative construct, it causes the program to jump out of that construct; no matter how many times the loop is left to be executed.

The following program is a trivial program that persistently prompts the user to enter some texts. It will keep prompting the user for infinitely many times unless the user enters **`Java`**.

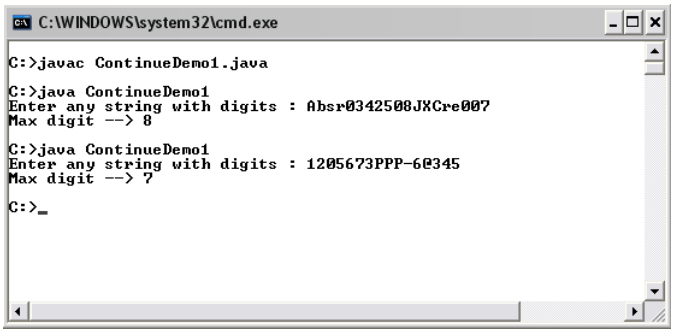
```
import java.io.*;                                                          1
public class BreakDemo1                                                    2
{                                                                          3
    public static void main(String[] args) throws IOException             4
    {   String s;                                                          5
        BufferedReader stdin =                                             6
            new BufferedReader(new InputStreamReader(System.in));          7
        while(true){                                                       8
            System.out.print("Say the magic word\n>>");                    9
            s = stdin.readLine();                                         10
            if(s.equals("Java")) break;                                   11
        }                                                                 12
        System.out.println(":)");                                         13
    }                                                                     14
}                                                                         15
```

We can see on line 8 that the condition for the *while* loop is always **`true`**. Thus, the while loop repeats forever unless the input *String* is "Java", in which case the *break* statement is executed, resulting in the program jumping out of the *while* loop.

A *continue* statement causes the current iteration to terminate immediately. However, unlike what happens with a *break* statement, a *continue* statement pass the program flow to the start of the next iteration.

The following program should give you an example of how *continue* works. The program is used for finding a maximal-valued digit from a string of character. Each character in the input string is not restricted to being a digit. For example, if the input is "abc12D81", the maximal-valued digit is 8.

```
import java.io.*;                                                          1
public class ContinueDemo1                                                 2
{                                                                          3
    public static void main(String[] args) throws IOException             4
    {   int len, max = 0;                                                  5
        String s;                                                          6
        BufferedReader stdin =                                             7
            new BufferedReader(new InputStreamReader(System.in));          8
        System.out.print("Enter any string with digits : ");              9
        s = stdin.readLine();                                             10
        len = s.length();                                                 11
        for(int i=0; i<len; i++){                                         12
            char c = s.charAt(i);                                         13
            if(!(c>='0' && c<='9')) continue;                             14
            int digit = Character.digit(c,10);                            15
            if(digit > max) max = digit;                                  16
        }                                                                 17
        System.out.println("Max digit --> "+max);                         18
    }                                                                     19
}                                                                         20
```



A *for* statement starting on line 12 goes through every position of the input *String*. In each position, if the character at that position does not fall between '0' and '9' inclusively, that character is not a digit. Thus, if `!(c>='0' && c<='9')` is `true`, there is no need to do anything else to the character in that position. Therefore, *continue* is used in order for the program to start the next iteration right away, as seen on line 14.

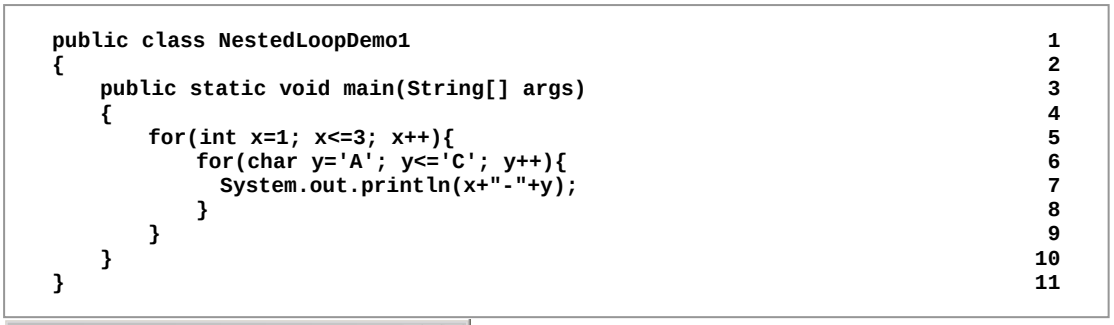
Nested 'for' Loops

A *for* statement can be placed inside another *for* statement, causing what we called *nested for loops*. Observe the following code segment.

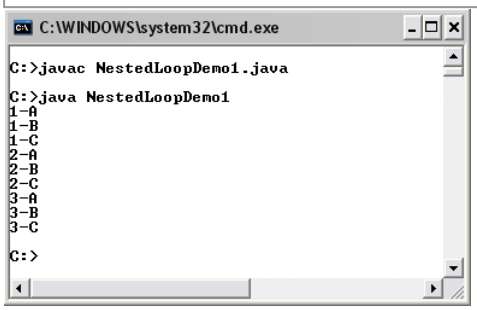
```
for(int i=1; i<=n; i++){
    for(int j=1; j<=m; j++){
        Actions
    }
}
```

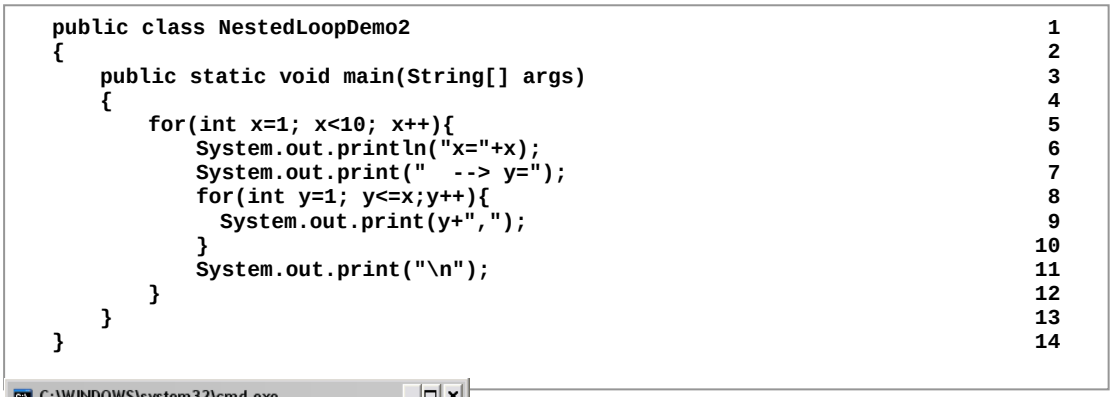
The outer *for* statement is associated with an index variable `i` that runs from 1 to `n`, resulting in `n` iterations of the inner *for* statement. For each iteration of the outer *for* statement, the inner *for* statement iterates `m` times. So, this results in `Actions` being executed `n` × `m` times.

Observe the following programs and their outputs.

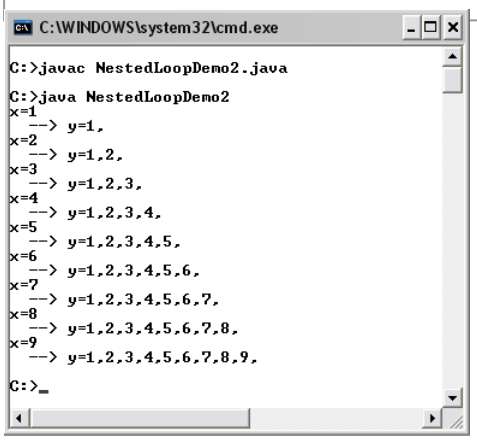


```
public class NestedLoopDemo1                                    1
{                                                               2
    public static void main(String[] args)                     3
    {                                                           4
        for(int x=1; x<=3; x++){                                5
            for(char y='A'; y<='C'; y++){                       6
              System.out.println(x+"-"+y);                      7
            }                                                   8
        }                                                       9
    }                                                           10
}                                                               11
```





```
public class NestedLoopDemo2                                    1
{                                                               2
    public static void main(String[] args)                     3
    {                                                           4
        for(int x=1; x<10; x++){                                5
            System.out.println("x="+x);                         6
            System.out.print("  --> y=");                       7
            for(int y=1; y<=x;y++){                             8
              System.out.print(y+",");                          9
            }                                                   10
            System.out.print("\n");                           11
        }                                                       12
    }                                                           13
}                                                               14
```



Index Variable Scope

If index variables are declared in the initialization part of a *for* statement (i.e. in **`Init Expression`**), they are known only inside the block associated with that *for* loop. Or, we can say that the scope of these variables is just inside that *for* statement. If you attempt to use these variables outside, a compilation error will occur. However, if a variable is declared in a block where a *for* statement is nested inside, immediately or not, that variable is known in that *for* loop as well.

The following code segment cannot be compiled successfully since the variable k is not known outside the *for* loop.

```
public static void main(String[] args)
{   int lastIndex;
    for(int k = 0; k < 10; k++){
        // Some statements
    }
    lastIndex = k; // This line causes a compilation error.
}
```

Example

{*x*,*y*,*z*} is a solution of $a^2+b^2+c^2 = 200$ if the equation is true when a = *x*, b = *y*, and c = *z*. How many unique solutions are there if we require that a, b, and c can only take non-negative integers? We can write a Java program utilizing nested *for* loops to count the number of all possible solutions to the equation that meet the non-negative integer requirement.

The idea is to try all possible combinations of three non-negative integers and see whether which ones of them satisfy $a^2+b^2+c^2 = 200$. Due to the non-negative integer requirement, each variable from a, b, and c can only be as small as 0, while each of them cannot exceed $\lfloor\sqrt{200}\rfloor$. Thus, we use three-level nested *for* loops, each of which is associated with a variable that runs from 0 to $\lfloor\sqrt{200}\rfloor$. Whether each combination of the three non-negative integers is a solution of the equation is tested in the innermost *for* loop, where the solution is also printed out on the screen.

Here is a Java program that do the mentioned task.

```
public class SolutionsCount                                                  1
{                                                                            2
    public static void main(String[] args)                                  3
    {   int a,b,c,nSolutions=0;                                              4
        int maxPossible = (int)Math.floor(200);                              5
        for(a = 0; a < maxPossible; a++){                                    6
            for(b = 0; b < maxPossible; b++){                                7
              for(c = 0;c < maxPossible; c++){                               8
                if(a*a+b*b+c*c == 200){                                      9
                  nSolutions++;                                             10
                  System.out.print("Soltn #"+nSolutions);                   11
                  System.out.println("("+a+","+b+","+c+")");                 12
                }                                                           13
              }                                                             14
            }                                                               15
        }                                                                   16
        System.out.println("# of non-negative integer solutions for ");     17
        System.out.println("a^2 + b^2 + c^2 = 200 is "+nSolutions);         18
    }                                                                       19
}                                                                           20
```

```
C:\WINDOWS\system32\cmd.exe

C:>javac SolutionsCount.java

C:>java SolutionsCount
Soltn #1(0,2,14)
Soltn #2(0,10,10)
Soltn #3(0,14,2)
Soltn #4(2,0,14)
Soltn #5(2,14,0)
Soltn #6(6,8,10)
Soltn #7(6,10,8)
Soltn #8(8,6,10)
Soltn #9(8,10,6)
Soltn #10(10,0,10)
Soltn #11(10,6,8)
Soltn #12(10,8,6)
Soltn #13(10,10,0)
Soltn #14(14,0,2)
Soltn #15(14,2,0)
# of non-negative integer solutions for
a^2 + b^2 + c^2 = 200 is 15

C:>_
```

Exercise

1. If `n` is an integer greater than 0, how many times is *boohoo()* executed in the following code segment?

```
int i = 0;
do{
    i++;
    booHoo();
}while(i<=n);
```

2. What is the output of the following code segment?

```
int k = 0, m = 6;
while(k<=3 && m>=4){
    System.out.println(k+","+m);
    k++;
    m--;
}
```

3. What is the value of `n` after the following code segment is executed?

```
int n = 1, i = 100;
while(n++<i--);
```

4. What are the values of `n` and `m` after the following code segment is executed?

```
int n=0, m=0, i=0,maxItt=300;
while(i++<maxItt) n++;
i=0;
while(++i<maxItt) m++;
```

5. Rewrite the code segment in the last problem by using *for* loops instead of the two *while* statements.

6. What is the value of `n` after the following code segment is executed?

```
int n = 0;
for(int i = 1;i<=100;i++)
    for(int j = 1;j<=100;j++)
        n++;
```

7. What is the value of `n` after the following code segment is executed?

```
int n = 0;
for(int i = 50;i>0;i--)
    for(int j = 40;j>0;j--)
        n++;
```

8. What is the output of the following code segment?

```
int k=0;
for(int i=100;i>1;i/=2,k++);
System.out.println(k);
```

9. Explain why the following program cannot be compiled successfully.

```
public class Ex7Test
{
    public static void main(String[] args)
    {
        int n = 10, k = 0;
        for(int i=0;i<n;i++){
            k++;
        }
        System.out.println("i="+i+", k="+k);
    }
}
```

10. Use a *for* statement to write a Java program to calculate the value of 8!

11. Repeat the previous problem using a *while* statement instead of the *for* statement.

12. Write a Java program that calculates *n*! from the integer *n* obtained from keyboard. If the value of the input *n* causes an overflow to occur, report it to the user. Otherwise, show the value of *n*! on screen.

13. Write a Java program that randomly picks an English alphabet (A-Z) and keeps asking the user to guess the alphabet until he/she has got it right. Also report the number of trials.

14. Determine the output of the following code segment.

```
int i,j;
for(i=0,j=0;i<=100;i++){
    if(i%2==0)
        i++;
    else
        j++;
}
System.out.println(j);
```

15. Determine the output of the following code segment.

```
int i=0,a=0,n=0;
while((i=a+=2)<=100){
    n++;
};
System.out.println(i+","+a+","+n);
```

16. What is the value of `k` after the following code segment is executed?

```
int k=0, j=0;
while(true){
    if(j<20){
        k++; j++;
    }else{
        break;
    }
    if(k>5){
        continue;
    }else{
        j++;
    }
}
```

17. Use nested loops to write a Java program that generates a multiplication table as shown below.

```
       |   2   3   4   5   6   7   8   9   10  11  12
-------------------------------------------------------
2      |   4   6   8   10  12  14  16  18  20  22  24
3      |   6   9   12  15  18  21  24  27  30  33  36
4      |   8   12  16  20  24  28  32  36  40  44  48
5      |   10  15  20  25  30  35  40  45  50  55  60
6      |   12  18  24  30  36  42  48  54  60  66  72
7      |   14  21  28  35  42  49  56  63  70  77  84
8      |   16  24  32  40  48  56  64  72  80  88  96
9      |   18  27  36  45  54  63  72  81  90  99  108
10     |   20  30  40  50  60  70  80  90  100 110 120
11     |   22  33  44  55  66  77  88  99  110 121 132
12     |   24  36  48  60  72  84  96  108 120 132 144
```

18. Write a code segment using nested loops that displays a daily timetable of the form:

```
0:00AM    _____________________
0:30AM    _____________________
1:00AM    _____________________
1:30AM    _____________________
  ……
11:30AM   _____________________
12:00PM   _____________________
12:30PM   _____________________
 1:00PM   _____________________
  ……
11:00PM   _____________________
11:30PM   _____________________
```

19. Write a Java program that reverses the order of the characters in the string input from keyboard and show them on screen.

20. Write a Java program that finds the value of $f(x,n)$, where x can be any real value, n can only be a non-negative integer, and $f(x,n)$ is defined as:

$$f(x,n) = \sum_{i=0}^{n} x^i$$

Your program must check whether the value of n input by the user is valid or not. (i.e. n must have an integer value and be non-negative.) Use *Math.pow()* to find the value of x^i.

21. Repeat the previous problem without using any methods provided by the *Math* class.

22. Write a Java program that calculates and shows the sum of all even integers from 0 to **n**, where **n** is specified by the user via keyboard.

23. An imaginary webmail service called "veryhotmail.com" is kind enough to let its users choose their own passwords. However, a valid password must comply with the following (somewhat strange) rules:

- The length must be between 6 to 15 characters, inclusive.
- Each character must be either one of the 26 English alphabets. Both uppercase and lower case letters are allowed.
- The difference between the number of the uppercase letters and the lowercase letters cannot be greater than 20% of the password length.

Write a code segment that can validate the password stored in a *String* reference **s**.

24. Write a Java program to shows the value of a_0, a_1, …, a_n that is corresponding to the recurrence relation $a_k = k^2a_{k-1}-a_{k-2}+3^k$ where k = 2, 3, 4 ,… The values of n and the initial conditions (a_0 and a_1) are specified by the user at run-time via keyboard.

25. Write a Java program to find the number of solutions to the equation $x+y+z = 30$ where x, y, and z are non-negative integers.

26. Write a Java program to find the number of solutions to the equation $x+y+z = n$ where n is a constant integer supplied by the user of the program and x, y, and z are integers between $-b$ and b. Also, b is an integer defined by the user.

27. Write a Java program that receives an English sentence from keyboard. The program encodes the sentence by reversing the order of letters in each word appearing in that sentence while keeping the original word order and shows the result on screen. Assuming that words are always separated by a space. For example, "`We are the champions.`" is encoded as "`eW era eht .snoipmahc`".

28. Write a code segment that replaces any number of multiple spaces connected together in a *String* reference `s` with single spaces.

    For example, if `s` contains:

    "`This does not contain multiple spaces.`",

    it should be changed to:

    "`This does not contain multiple spaces.`"

29. An integer *n* is prime if its only factors are 1 and itself. Write a code segment for checking whether the value of an `int` *n* is prime.

30. An integer *n* is perfect if the sum of its factors is *n*. Write a code segment for checking whether the value of an `int` *n* is perfect.

31. Write a Java program that displays positive integer values less than 10 million in words. The values are received from keyboard. For example, the value 1500 is displayed as "`one thousand five hundred`". The program should keep asking for another value until the user input -1.