
UNIT 3 DECISIONS AND LOOPS CONSTRUCTS

Structure

Page No.

- 3.0 Introduction
- 3.1 Objective
- 3.2 Decision Constructs
 - 3.2.1 The if statement
 - 3.2.2 The if then else statement
 - 3.2.3 The switch-case statement
 - 3.2.4 The jump statement
- 3.3 Loop Constructs
 - 3.3.1 while Loop
 - 3.3.2 do-while Loop
 - 3.3.3 for Loop
- 3.4 Array
- 3.5 Summary
- 3.6 Solutions/Answers to Check Your Progress
- 3.7 References/Further Readings

3.0 INTRODUCTION

The previous unit of this block introduced the basics of Java programming, such as data type, Variables, Operators and Statements/expressions. These basic building blocks of Java programming are used in writing any Java program. Now you are well aware of the basics. In this unit, we will discuss some more features of Java like decision statements, looping statements and arrays. Normally, you know that the statements are executed sequentially in the order they are written in the program, but sometimes situations arise in the program to execute a certain set(s) of code(s) on the basis of arising conditions in the program or repeat some section of code(s) based on the specified condition. For this purpose, Java supports selection statements, iteration statements, jump statements etc. The decision-making statements allow you to execute a particular section of program code based on the result of some boolean conditions in the program. Java supports two selection statements known as 'if' and 'switch'. Besides these, you will also learn some iterative statements such as do, do-while and for loop, which allows you to repeat a block of code under certain conditions. At the end of the unit, you will learn about arrays in Java language. You have already studied these concepts in the MCS-201 course of your first semester. That learning will help you pursue this unit. This unit consists of Java program examples which are executed in NetBeans IDE.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- used various kinds of decision statements in Java program,
- write Java programs using various iterative statements,
- Explain how to use array in programming, and
- differentiate between static array and dynamic array.

3.2 DECISION CONSTRUCTS

The program is not always executed in a sequential manner; sometimes, the flow of the program requires executing part of a program based on the condition or transferring control from one part to another part of the program. The Java language provides decision making statements to take a decision based upon the result of a specified condition and execute a set of codes based on that. The selection statement is the 'if' statement and the 'switch' statement. The condition defined with the selection statement is checked at the run time. Actually, the flow of programs and the output of a particular part of the program are decided by the conditions. Java supports two jump statements as 'break' and 'continue'. You will learn these selections and jump statements in the following section.

3.2.1 The if Statement

The functioning of 'if' statement in Java language is similar to the 'if' statement in any other programming language. It is syntactically identical to the 'if' statement in C and C++. The simplest form of the 'if' statement is shown here:

`if(condition) statement;`

where *condition* is a boolean expression, and it must be enclosed with parentheses. If the condition returns true value, then the statement is executed; otherwise, the statement is bypassed. For example:

`if(a > b) System.out.println("a is greater than b");`

The message *a is greater than b* will be printed only when `if(a>b)` returns true.

You can execute multiple statements when a single condition is true. For this, you can use a set of statements or block of statements which take the following form:

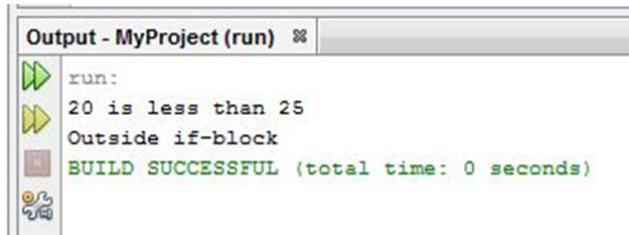
```
if(condition)
{
    Statement1
    Statement2
    ...
}
```

Example-1

Consider the following example for if statement. In this example, x is initialized with 20. Using 'if' statement, the condition is checked and if it is true then only "20 is less than 25" gets printed.

```
class IfExample
{
    public static void main(String args[])
    {
        int a = 20;
        if (a < 25)
            System.out.println("20 is less than 25");
        System.out.println("Outside if-block");
    }
}
```

Output:



```
Output - MyProject (run) %
run:
20 is less than 25
Outside if-block
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.2.2 The if-then-else Statement

The if-then-else statement is represented in Java as *if-else* statement and is used to execute a certain part of codes. The 'if' statement is followed by a test condition, and when the specified condition evaluates to true, then the block of code of 'if' part is executed. If the specified condition evaluates to false, then control is transferred to *else* part of the program.

Following is the syntax of if...else statement:

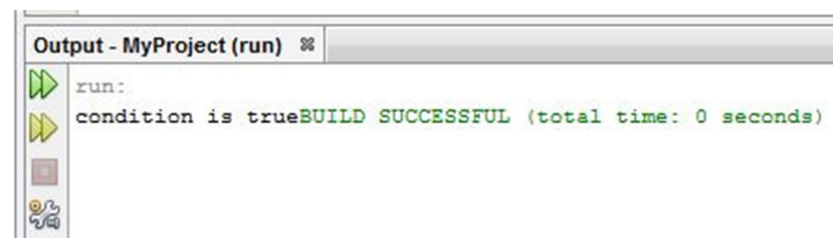
```
if(condition)
{
    // executes code when the condition is true
}
else
{
    // executes code when the condition is false
}
```

Example-2

Consider the following example for if-else statement:

```
class Example1
{
    public static void main(String args[])
    {
        int a = 50;
        if( a > 20 )
        {
            System.out.print("condition is true");
        }
        else
        {
            System.out.print("condition is false");
        }
    }
}
```

Output:



```
Output - MyProject (run) %
run:
condition is trueBUILD SUCCESSFUL (total time: 0 seconds)
```

3.2.3 The switch-case Statement

The if-else concept can be bulky when you have to work with multiple selections with many alternatives. For this purpose, Java provides *switch-case* statement. The basic purpose of the switch statement is used to select one statement (block) from the multiple code blocks for execution. If there are N number of case values for a switch condition, then each case will have a different value. Each case statement can have a break statement which is optional. When control comes at the break statement, it terminates the statement sequence inside the switch and control transfers after the switch statement. If a break statement is not there, it executes the next case of the switch statement.

The syntax of the switch statement in Java is as follows:

```
switch (condition)
{
case value1:
    // Java code
    break;
case value2:
    // Java code
    break;
    ...
    ...
default:
    // default statements
}
```

When you run your Java program with a switch statement, the condition is evaluated once and compared with the values of each case. If the condition matches with value1, the Java code of the case value1 is executed. In a similar manner, the code of case value2 is executed if the condition matches the value2. If there is no match, then the code of the default case is executed.

Example-3

The following Java example will ask the user for the input value in numbers and will display the output according to the value entered by the user.

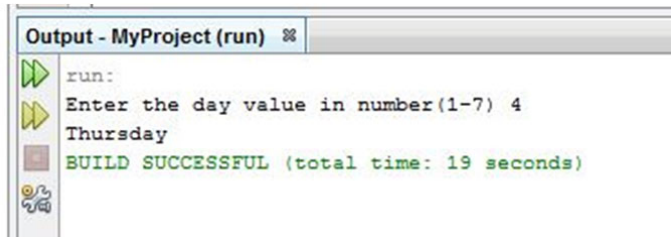
```
// Java program using switch statement
import java.util.Scanner;
class switchExample
{
    public static void main(String[] args)
    {
        int day;
        // Create a Scanner object for keyboard input.
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the day value in number(1-7) ");
        day = input.nextInt();
        switch (day)
        {
            case 1: System.out.println("Monday"); break;
            case 2: System.out.println("Tuesday"); break;
            case 3: System.out.println("Wednesday."); break;
            case 4: System.out.println("Thursday"); break;
```

```

case 5: System.out.println("Friday"); break;
case 6: System.out.println("Saturday"); break;
default: System.out.println("Sunday");
}
}
}

```

Output:



3.2.4 The jump statement

While working with loops, it is sometimes desirable to skip some statements within the loop or stop the loop instantly without evaluating the test condition. For this purpose, Java provides jump statement. The jump statement is used to transfer the flow control from one part to another part of the program code. Mostly, two Jump statements i.e. break and continue statements in Java language, are used. As the name implies the meaning of both statements, the '*break*' keyword is used to designate break statements and '*continue*' keyword is used to designate the continue statement in Java programming. The third jump statement i.e. return, is used to terminate the current method. In the below section, both break and continue statements are discussed in detail with examples.

Break Statement

In Java, the break statement is broadly used with '*for*' loop, '*while*' loop, '*do-while*' loop and '*switch*' statement. The use of a *break statement in a switch statement terminates* the statement sequence. It can also be used to exit from loop execution and control transfer to the next statement immediately following the loop.

Example-4

Following is a simple Java program to demonstrate the use of break statement:

```

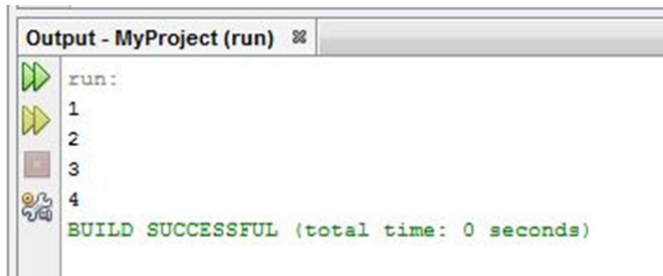
// Java program using break statement
class BreakExample
{
    public static void main(String[] args)
    {
        // for loop
        for (int x = 1; x <= 100; ++x)
        {
            // if the value of x is 5 the loop terminates
            if (x == 5)
            {
                break;
            }
            System.out.println(x);
        }
    }
}

```

```
}  
}
```

When you run the above program, it will print the value of x in each iteration using 'for' loop (You will read in detail about the 'for' loop in section 3.3.3 of this unit). When the value of x is equivalent to 5, the loop terminates. Hence you will get the output with values less than 5 only.

Output:



```
Output - MyProject (run) %  
run:  
1  
2  
3  
4  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Continue Statement

In Java, the continue statement is used with 'for' loop, 'while' loop and 'do-while' loop. It is used to skip the current iteration of a loop and control transfer to the next statement or end of the loop. It all depends on the condition that occurs in the program. Also, the continue statement is also used in decision-making statements like *if..else* statement.

Following is the syntax of the continue statement.

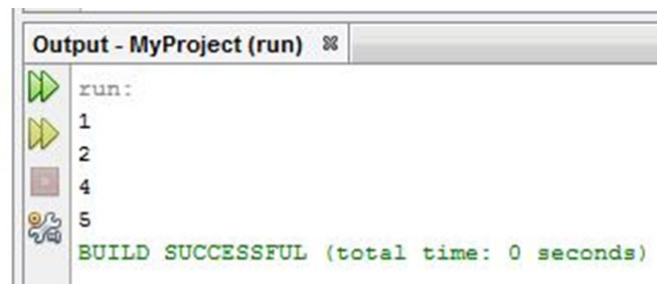
```
continue;
```

Example-5

Following is a simple Java program to demonstrate the use of the 'continue' statement:

```
// Java program for continue statement  
class ContinueExample  
{  
    public static void main(String[] args)  
    {  
        //for loop  
        for(int x=1;x<=5;x++)  
        {  
            if(x==3)  
            {  
                //using continue statement  
                continue;// here, it will skip the rest statement  
            }  
            System.out.println(x);  
        }  
    }  
}
```

When you run the above program, it will print the series like 1,2,4,5 and 3 will not be displayed on the console. It is because the loop is continued when it reaches to 3. After running the program, you will get the result like:



Check Your Progress- 1

1. Explain the decision-making statement in Java with examples.

2. What is a branching statement in Java? What are the types of branching statements in Java? Explain any one of them with an example.

3. What is the break statement? Explain its use in Java with an example.

3.3 LOOP CONSTRUCTS

In the previous section, you have learned about the decision-making statement. This section describes looping statements or iterating statements, which allow you to execute the same set of instructions repeatedly until a certain condition is fulfilled. Java supports three types of loops such as *while loop*, *do-while loop* and *for loop*. All three loops of Java are discussed below with examples.

3.2.1 while Loop

The while loop is a most fundamental looping statement in Java which repeats a statement or block of code until its condition is true. Below is its general syntax:

```
while(condition)
{
    // body of loop
}
```

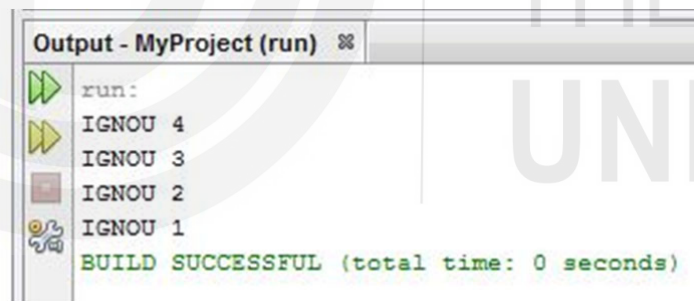
The condition can be any boolean expression, and the body of the loop will be repeated until the condition is true. When the condition turns false, the control transfers to the next line of program code immediately after the while loop.

Example-6

Here is an example to show the use of the while loop. This program prints " IGNOU " words with numbers in decreasing order four times.

```
// Demonstration of while loop
class whileLoop
{
    public static void main (String args[])
    {
        int i = 4;
        while(i > 0)
        {
            System.out.println("IGNOU"+" "+ i);
            i--;
        }
    }
}
```

Output:



```
Output - MyProject (run) x
run:
IGNOU 4
IGNOU 3
IGNOU 2
IGNOU 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.2.2 do-while Loop

The do-while loop in Java is used to execute a part of the program code repeatedly until the specified condition is true. The statements within do-while loop are executed once, and thereafter the condition gets evaluated; if this condition is true, then the loop will repeat. Otherwise, the loop terminates, and control is transferred to the next statement after do-while loop.

A do-while loop in Java is similar to the while loop with one exception. The while loop checks the condition first and then executes the statement(s), whereas do-while loop will execute the statement(s) at least once, and then the condition is checked.

The syntax of do-while loop is as follows:

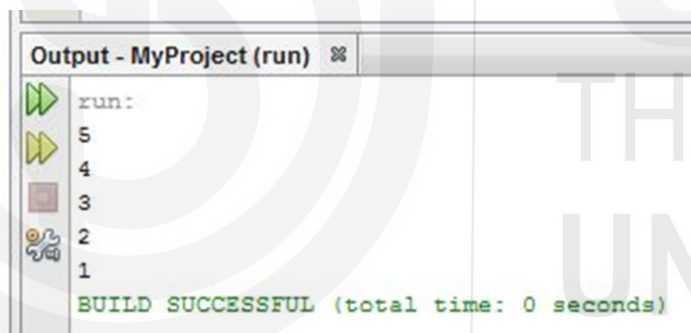

```
do
{
// body of loop
}while (condition);
```

Example-7

Code Snippet to demonstrate the use of do-while loop:

```
// Demonstration of do-while loop
class doWhileExample
{
public static void main(String args[])
{
int x=5;
do
{
System.out.println(x);
x--;
}while(x>0);
}
}
```

When you run the above program, it will display the following output:



3.2.3 for Loop

The 'for' loop allow(s) you to execute a block of statement repetitively, a fixed number of times on the basis of the decided pre-condition. The for loops is one of the very commonly used looping constructs in programming. The 'for' loop is a very general construct of looping that supports the iteration of a block of code controlled by a counter or some updated variable after every iteration.

The basic syntax of for loop is as follows:

```
for(initialization; condition; increment/decrement)
{
//body of loop
}
```

When the 'for' loop first starts, the initialization part is executed only once. Next, the condition is evaluated. This condition must be a boolean expression. It checks the loop

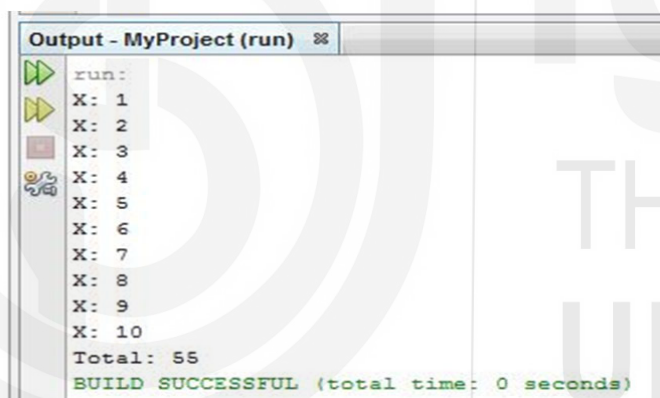
control variable against a target value. If this test condition is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the increment/decrement or iteration part of the loop is executed.

Example-8

Consider the following Java program, which prints the series of the first 10 natural numbers and also calculates the total sum of the numbers using for loop.

```
class Total
{
public static void main(String[] args)
{
int total = 0;
for(int x=1; x<=10; x++)
{
System.out.println("X: " + x);
total += x;
}
System.out.println("Total: " + total);
}
}
```

Output:



```
Output - MyProject (run)
run:
X: 1
X: 2
X: 3
X: 4
X: 5
X: 6
X: 7
X: 8
X: 9
X: 10
Total: 55
BUILD SUCCESSFUL (total time: 0 seconds)
```

Check Your Progress - 2

1. Write a Java program using while loop to display numbers from 1 to 'N', where the program will ask the user to input the number 'N'.

2. What is the difference between while and do-while loop?

-
3. Write a Java program which prints odd numbers between 0 and 10 using ‘for’ loop.
-
-
-
-

3.4 ARRAY

There is always a need to store multiple data items of the same data types in programming. For example, storing marks of one subject of all the students in a class. For this purpose, Java uses an array. An array is a collection of similar data type items which are stored in contiguous memory locations. These multiple values are stored in a single variable rather than declaring separate variables for each value.

One-Dimensional Arrays

A one-dimensional array is a list of variables of similar/same data type. For declaring an array, you must define the variable type with square brackets. The general form of one-dimensional array declaration is as follows:

type var_name[];

Here, type indicates the *data type* of array. For example, a variable named ‘Bus’ is declared as an array of strings.

String[] Bus;

You can use an array literal to insert values to it and place the values in a comma-separated list within curly braces, as shown below.

String[] Bus= {"Volvo", "TATA", " Ashok Leyland", " Mahindra & Mahindra"};

Accessing the value of an array element is done using the index number. The index of an array variable determines on which value of the array you want to access or work on. The following example shows how to access the value of the first element in the ‘vehicle’ array:

Example-

```
String[] vehicle = {"car", "bus", "scooter", "motorcycle"};
System.out.println(vehicle[0]);
// Outputs: car
```

In the above array- vehicle, there are four values stored, and the value stored at index 0 is “car”; similarly, at index 3 value "motorcycle" is stored. Note that indexing start from 0 and goes up to N-1, where N is the total elements in the array.

In a similar way, you can create an array of integers.

int[] Marks = {1, 2, 3, 4};

It is noted that Java uses zero-based indexing i.e. indexing of arrays in Java starts with zero(0). In the array variable Marks, given above if we access Marks[2] we will get the value 3.

Creating an Array using New Operator

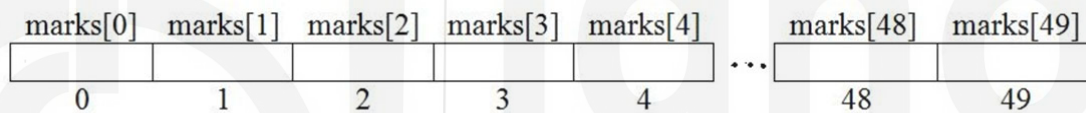
The main objective of the *new operator* is to allocate memory for a variable or an object during the run time. The array length need not be a constant: `new int [n]` creates an array of length n. Like the following, statement declares and initializes an array of 100 integers. This is general-form of the *new operator* which applies to one-dimensional array and appears as follows:

```
int [] a = new int [100] ;
```

For **example**, you may declare and initializes an integer type of 50 elements of marks as an array.

```
int [ ] marks = new int [50] ;
```

The memory allocation for the array variable marks is given as following:



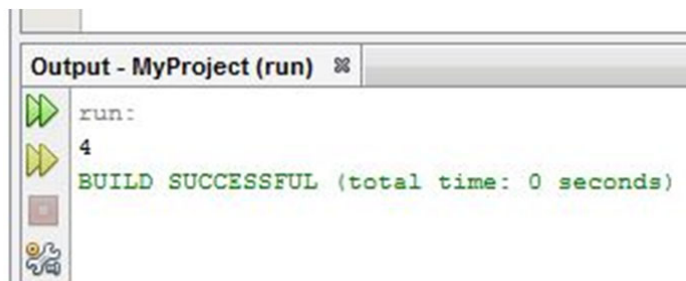
In the above figure, array indexes are from 0 to 49, and the array locations are accessed using marks [0], marks [1]...marks [49]. Since a single name refers to all the locations, the values can be stored and retrieved more easily rather than individual variables.

You can use the length property of array in Java to find out the total number of elements contained by all the dimensions of that array.

Example-10

```
public class ArrayLength
{
    public static void main(String[] args)
    {
        int[] a = {1,2,3,4};
        System.out.println(a.length);
    }
}
```

When you run the above program, it will display the following output:



Multidimensional Array

A multidimensional array is an array of arrays. Such arrays use more than one index to access the array elements. The general syntax of multidimensional array is as under:

```
data_type [1st dimension][2nd dimension][ ].....[Nth dimension]
```

```
array_name = new data_type[size1][size2].....[sizeN];
```

Where `data_type` represents the type of data to be stored in the array, such as `int`, `char`, etc., and the *dimension* indicates the dimension of the array created, i.e. 2D, 3D etc., **array_name** is the name of the array variable and **size1, size2, ..., sizeN** shows sizes of the dimensions respectively.

Two-dimensional arrays

The two-dimensional array is the simplest form of a multidimensional array. In a two-dimensional array, data is stored in rows and columns. You can access the elements using both the row index and column index. It is noted that Java uses zero-based indexing i.e. indexing of arrays in Java starts with zero(0). Following statement shows that two-dimensional array consists of 12 elements, i.e. 3 rows and 4 columns.

```
int[ ][ ] TwoD = new int[3][4]; // it can store a total of (3*4) = 12 elements.
```

The following table shows the memory allocation of 12 elements of two-dimensional array:

	Column 1	Column 2	Column 3	Column 4
Row 1	2D[0][0]	2D[0][1]	2D[0][2]	2D[0][3]
Row 2	2D[1][0]	2D[1][1]	2D[1][2]	2D[1][3]
Row 3	2D[2][0]	2D[2][1]	2D[2][2]	2D[2][3]

When you will create a two-dimensional array, add each array within its own set of curly braces as shown below:

```
int[ ][ ] NumArray = { {1, 2, 3, 4, 5}, {6, 7, 8, 9} };
```

`NumArray` is two dimensional array. For accessing the elements of the `NumArray`, you can specify two indexes: one for the array and another for the element inside the array.

Three-dimensional Arrays

Following three dimensional integer arrays is accomplished to store 6000 elements:

```
int[ ][ ][ ] ThreeDArray = new int[10][20][30]; // it can store a total of (10*20*30) = 6000 elements
```

Example-11

The following Java program demonstrates the two-dimensional array in which each element displays the number values from left to right and top to bottom:

```
// demonstrates the two-dimensional array
```

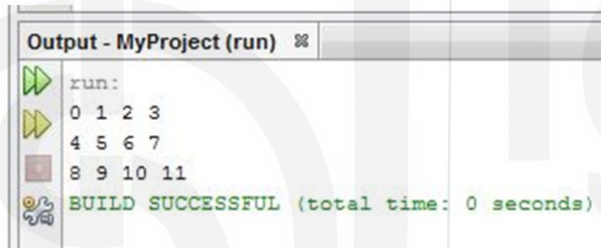
```
class TwoDArray
{
    public static void main(String[] args)
```

```

{
// Array consists of 12 elements i.e. 3 row and 4 column
int[][] TwoDim = new int[3][4];
int x, y, z = 0;
    for (x = 0; x < 3; x++)
        for (y = 0; y < 4; y++)
        {
            TwoDim[x][y] = z;
            z++;
        }
    for (x = 0; x < 3; x++)
    {
        for (y = 0; y < 4; y++)
            System.out.print(TwoDim[x][y] + " ");
        System.out.println();
    }
}
}

```

When you run the above program, it will display the following output:



```

Output - MyProject (run)
run:
0 1 2 3
4 5 6 7
8 9 10 11
BUILD SUCCESSFUL (total time: 0 seconds)

```

Variable-length array (Dynamic Array)

As yet, you have learned fixed-sized or static array. Static arrays are declared before runtime and are assigned values while writing the code. It means that static arrays are allocated memory at compile time. The variable-length array or dynamic array is declared while writing the code, and memory is allocated at the runtime. The memory is allocated from the heap. In other words, the dynamic arrays can be assigned values in run-time. If you want to create dynamic arrays of variable length, you can use collections like *array list*. Java also provides the facility to specify the size of the remaining dimensions separately, except for the first dimension. The following program explains this concept in which the second dimension size to array variable is allocated manually.

Example-12

```

class ArrayDSize
{
public static void main (String args[])
{
int x, y, z=0;
int twoDim [ ][ ] = new int [4][ ];
twoDim[0] = new int[1];
twoDim[1] = new int[2];
twoDim[2] = new int[3];
twoDim[3] = new int[4];
for ( x= 0; x<4 ; x++)

```

```

{
for (y = 0; y<x+1; y++)
{
twoDim[x][y] = z + z*4;
z++;
}
}
for ( x= 0; x <4 ; x++)
{
for (y = 0; y< x+1; y++)
System.out.print(twoDim[x][y] + " ");
System.out.println();
}
}
}

```

When you run the above program, it will display the following output:

```

Output - MyProject (run)
run:
0
5 10
15 20 25
30 35 40 45
BUILD SUCCESSFUL (total time: 0 seconds)

```

Check Your Progress- 3

1. What is an array in Java? Also, define a multidimensional array in JAVA.

2. Write a Java program to print the sum of all the array elements.

3. What is the difference between a static array and a dynamic array?

3.5 SUMMARY

In this unit you have been explained the Java decision and looping statements. Selection statements allow a Java program to check numerous conditions and execute a block of code based on the condition given. You have learned here *if*, *if-else*, *switch* statements and *jump* statements. Java developer mainly uses two jump statements, i.e. *break* and *continue* statement. The *break* statement is used to terminate the statement sequence. The *continue* statement is used to skip the current iteration of loops (i.e. *for*, *while*, or *do-while* loop). You can make multiple-choice decisions using a *switch* statement based on a single condition value. When the *switch* test condition evaluates true, control transfers to the matching case and executes the appropriate case statement.

Java supports three loops constructs such as *for*, *do* and *do-while*. The ‘*for*’ loop allows you to execute a block of the statement(s) repetitively for a fixed number of times based on the condition given. Additionally, you have also learned the *while* loop and *do-while* loop. Also, in this unit, you also learned about the array, such as declaring or creating an array and inserting values into the array. This unit also discussed the multidimensional array in Java program.

3.6 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress -1

- 1) Usually, the program is written in a sequential manner and executed in the order in which they appear, but the program appearance is not always sequential. Sometimes as per need of program, you have to define conditions. Using Java decision-making statements, you can define conditions on which you can make a decision and get a result based on that condition. It happens only when the jumping of statements or repetition of specific code is necessary. The decision-making statements allow you to execute specific parts of the program on the basis of a specified condition. Java supports some control statements such as *if*, *if-else*, and *switch-case* statements.
- 2) Branching statements are used to transfer the flow of execution from one section of a source code to another section. This kind of statement is widely used within the control statements. Java has mainly three branching statements, i.e., *break*, *continue*, and *return*, but programmers mostly use two of them (*break*, *continue*). The branching statements allow you to exit from a control statement when certain conditions meet.

The *continue* statement is used with *for loop*, *while loop* and *do-while loop*. This statement is used to skip the current iteration of a loop. For the example, you can refer to section 3.2.4 of this unit.

- 3) The *break* statement is broadly used with *for loop*, *while loop*, *do-while loop* and *switch statement*. The use of a *break* statement within the *switch* statement terminates the statement sequence.

Example Program

```
class Test
{
    public static void main(String args[])
    {
        for (int x = 0; x < 10; x++)
```



```

{
    if (x == 6)
    {
        break; //break statement
    }
    System.out.println(x);
}
}
}

```

Check Your Progress - 2

1. Program code which displays the numbers from 1 to N.

```

import java.util.Scanner;
class exampleWhile
{

    public static void main(String[] args)
    {
        int num;
        Scanner input = new Scanner(System.in); //use of Scanner class to take input
        System.out.print("Enter the N number : ");
        num = input.nextInt();
        int x = 1;
        // while loop from 1 to 10
        while(x <= num)
        {
            System.out.println(x);
            x++;
        }
    }
}

```

2. A do-while loop in Java is similar to the while loop with one exception. In while loop the condition is checked first and then executes the statement(s), whereas do-while loop will execute the statement(s) at least once and then checks the condition.
3. The 'for' loop allows you to execute a block of a statement(s) repetitively a fixed number of times on the basis of the condition. When the 'for' loop starts, the initialization part is executed only once. Next, the *condition* is evaluated. This *condition* must be a boolean expression. It checks the loop control variable against a target value. If this test condition is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the *loop's increment/decrement or iteration part* is executed.

Example Program

```

Class forLoopExample
{

    public static void main(String[] args)
    {

```

```

for (int x = 0; x<=10; x=x+3)
{
System.out.println(x);
}
}
}

```

Check Your Progress- 3

- 1) An array provide you chance to create variable which is a collection of similar data type items, and are stored in contiguous memory locations. These multiple values are stored in a single variable rather than declaring separate variables for each value. A one-dimensional array is a list of the same type of variables. A multidimensional array is an array of arrays. In multidimensional arrays, more than one indexes are used to access the array elements.
- 2) The Java program given below prints the sum of all the elements of the given array.

```

class ArrayExample
{
public static void main(String[] args)
{
//Initialize array
int [] A = new int [] {10, 20, 30, 40, 50};
int sum = 0;
//For' Loop for calulating the sum of elements
for (int x = 0; x < A.length; x++)
{
sum = sum + A[x];
}
System.out.println("Sum of elements of an array: " + sum);
}
}

```

- 3) Static Arrays are declared before runtime and are assigned values while writing the code. It means that static arrays are allocated memory at compile time. The variable-length array or dynamic array is declared while writing the code and can be assigned values in runtime. In other words, the dynamic arrays are allocated memory at the runtime, and the memory is allocated from the heap. If you want to create dynamic arrays of variable length, you can use collections *like array list*.

3.7 FURTHER READINGS

- Herbert Schildt, “Java The Complete Reference”, McGraw-Hill, 2017.
- Savitch, Walter, “Java: An introduction to problem solving & programming”, Pearson Education Limited, 2019.
- S.Sagayaraj, R. Denis, P.Karthik and D.Gajalakshmi, “Java Programming for Core and Advanced Learners”, University Press, 2018.

- Neil, O. , “Teach yourself JAVA”, Tata McGraw-Hill Education, 1999.
- <https://www.w3schools.com/Java/>
- <https://docs.oracle.com/javase/tutorial/Java/nutsandbolts/arrays.html>

