Chapter 3

3. Decision and Repetition Statements

3.1 Overview of Java statements

Statements and expressions

All tasks that you want to accomplish in a Java program can be broken down into a series of statements. In a programming language, a statement is a simple command that causes something to happen.

All the following are simple Java statements:

Example 1

```
int weight = 20;
int Length = 30;
int area= weight* Length;
System.out.println (area);
```

Some statements can convey a value, such as when you multiply or add two numbers together in a program or evaluate whether two variables are equal to each other. This kind of statement is called an expression.

An expression is a statement that produces a value. The value can be stored for later use in the program, used immediately in another statement, or disregarded. The value produced by a statement is called its return value.

Some expressions produce a numerical return value, as when two numbers are added together or multiplied. Others produce a Boolean value—true or false—or even can produce a Java object.

Although many Java programs contain one statement per line, this is a formatting decision that does not determine where one statement ends and another one begins. Each statement in Java is terminated with a semicolon character (;). A programmer can put more than one statement on a line, and it will compile successfully, as in the following example:

Example 2

```
int weight = 20;
int Length = 30;
```

Or

```
int weight = 20; int Length = 30;
```

Statements in Java are grouped using the opening brace ({) and closing brace (}). A group of statements organized between these characters is called a block or block statement

Example 3

```
class Main {
public static void main (String [] args) {
int weight = 20;
int Length = 30;
int area= weight* Length;
System.out.println (area);
}
}
Output =600
```

3.2 Decisions

Decision making statements are used to decide whether or not a particular statement or a group of statements should be executed.

The if statement

If statement is used to selectively execute a statement or a block of statements. The statement contains a condition which decides whether or not the statements would be executed. If the condition stated in the parentheses evaluates to true, then the Statements are executed, otherwise the statements in the if block are skipped. Conditions are built using conditional and relational operators. Following is the syntax of the if statement.

```
if (condition) {

  // block of code to be executed if the condition is true

// block of code to be skipped or terminated if the condition is false

}
```

And here is a small code snippet which displays the message "Hi" only if the value of the Boolean variable wish is true.

Example 4

```
class Main {
public static void main (String [] args) {
 int weight = 20;
int Length = 30;

if (Length > weight) {System.out.println("Hi");}
}
}
```

Output = Hi

Example 5

```
class Main {

public static void main (String [] args) {

 int weight = 30;

int Length = 20;

if ( Length > weight) { System.out.println("Hi");}

}

}
```

Output = none or skipped

Example 6

```
public class Main {
  public static void main(String[] args) {
    if (20 > 18) {
      System.out.println("20 is greater than 18"); // obviously
    }
  }
}
```

Output=20 is greater than 18

Example 7

```
public class Main {
  public static void main(String[] args) {
    if (18 > 20) {
      System.out.println("20 is greater than 18");
    }
  }
}
```

No output is displays because it is false statement

In the above code snippet, the braces could have been omitted. Braces are not required when The block consists of only a single statement. But we recommend that braces always be placed.

This ensures that mistakes won't creep into the code when you modify your program at a later time to add another statement to the if block forgetting to insert the necessary braces. Moreover the condition ' Length > weight is true' the output is displayed, but the condition ' Length > weight is false nothing displayed.

The if statement is known as **a single selection structure** since we have only a single statement that is either executed or not. No alternative statement would be executed if the condition evaluates to false. For such purpose, we have the if else double selection structure.

The if else statements

If the condition is true, the if block will be executed. Otherwise, the else block will be executed. Following is the syntax of if else structure.

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example 8

```
class Main {
  public static void main(String[] args) {
    int time = 20;
    if (time < 18) {
      System.out.println("Good day.");
    } else {
      System.out.println("Good evening.");
    }
  }
}
```

Output= Good evening

Example 9

```
   class If{
  public static void main(String[] args) {
      int num=10;
    if ( num%2==0){
    System.out.println("Number is even");
     }
   else
   {
System.out.println("Number is odd");
}
  }
}
```

Output= Number is even

And here is an example usage which states whether a number is an even number or an odd number. Even numbers are divisible by zero and hence leave a remainder of zero. The remain der on dividing the number by two can be obtained using the modulo (%) mathematical operator.

if else statements can be nested. That is the statements block of an if or else can in turn contain another if else structure. Look at the following code where if else structures have be nested.

If……Else if statements

Its Syntax is :

```
If(condition){
//code
}
Else if(condition){
//code
}
Else{
//code
}
```

Example: 10

```
class ElseifExample {
  public static void main(String[] args) {
    int time = 22;
    if (time < 10) {
      System.out.println("Good morning.");
    } else if (time < 20) {
      System.out.println("Good day.");
    }  else {
      System.out.println("Good evening.");
    }
  }
}
```

NOTE: The if…else if…else statement is the same as switch statement.

The switch statement

Given below is the syntax of the switch structure in Java.

```
switch ( expression )
{
 case value1 :
  // code
  break;
 case value2:
  // code
  break;
...
...
 case valueN:
  // code
  break;
...
...
 default:
  // code
  break;
}
```

The expression should evaluate to one of the following types: byte, short, int, char, String. This value is compared with each of the case values (value1, value2,... valueN). When a match is found, the statements under that particular case label are executed. Strings are checked for equality by invoking the equals() method and not by using the == operator which is the used for the other data types. If no match is found, the statements under the default label are executed. The break keyword is used to transfer control outside the switch block. If the break keyword is omitted, the statements under the other case labels are also excited whether or not the case value matches. This might be useful in certain situations which we will look into shortly. The last break under the default label is optional as on reaching the line, the control has to anyway move out of the switch block.

All the case values should be unique. Duplicates are not allowed. Also note that braces are not required even if we wish to include more than one statement in a particular case label. However, using them doesn't make any difference. Given below is an example which prints the day of the week based on the value stored in the int variable num. 1 stands for Sunday, 2 for Monday and so on:

```
int day = s.nextInt; // s is a Scanner object connected to System.in
String str; // stores the day in words
switch(day)
{
 case 1: str="Sunday";
           break;
  case 2: str="Monday";
           break;
    case 3:
        str="Tuesday";
        break;
    case 4:
        str="Wednesday";
        break;
    case 5:
        str="Thursday";
        break;
    case 6:
        str="Friday";
        break;
    case 7:
        str="Saturday";
        break;
    default:
        str=" Invalid day";
}
System.out.println(str);
```

Example 11

```
class WeekDays
{
   public static void main(String s[])
   {
      int day = 4;

      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;
         default:
            System.out.println("Weekend");
            break;
      }
   }
}
```

```
Thursday
```

Example 12

```
class SwitchExample {
public static void main(String[] args) {
    int number=30;
    switch(number){
    case 10: System.out.println("10");break;
    case 20: System.out.println("20");break;
    case 30: System.out.println("30");break;
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

```
30
```

3.3 Repetition

Repetition statements are used to execute a statement or a group of statements more than once. They are called loops. Java provides three repetitions structures: while, do while and for. These statements are also known as loops. We shall look into the syntaxes of these structures first.

while loops

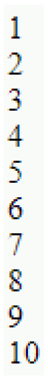
its syntax is:

```
while ( condition ) {
   // code
}
```

For example, the following code is used to display the numbers from 1 to 10.

Example:13

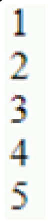
```
public class WhileExample {

public static void main(String[] args) {

   int i=1;

   while(i<=10){

      System.out.println(i);

   i++;

   }

}

}
```



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

Example:14

```
// Program to display numbers from 1 to 5

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

    // declare variables
    int i = 1, n = 5;

    // while loop from 1 to 5
    while(i <= n) {
      System.out.println(i);
      i++;
    }
  }
}
```



```
1
2
3
4
5
```

do...while loops

we have the exit control loops in which the condition is checked after executing the body of the loop because of which the loop is executed at least once.

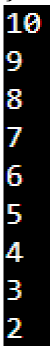
Following is the syntax of the do while loop:

```
do {
    // code
} while ( condition );
```

Note that there is a semicolon after the while statement. Following code uses the do while loop to print number on the screen.

Example: 15

```
class DoWhileLoopExample {
   public static void main(String args[]){
       int i=10;
       do{
           System.out.println(i);
           i--;
       }while(i>1);
   }
}
```



```
10
9
8
7
6
5
4
3
2
```

for loops

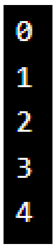
The last of the repetition statements is the for loop which provides a more convenient way to handle counters in loops. Following is the syntax of a for loop:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Following code prints the numbers from 0 to 4 using a for loop.

Example:16

```
class For {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
    }
  }
}
```



```
0
1
2
3
4
```

The following for header declares and initialises two variables i and j.for (int i= 0, j=10; // code; // code)

Here are the various formats of the initialisation part that are allowed:

< variable name > = <value >

< data type > < variable name > = < value >

< data type > < variable name 1 > = < value (optional) > , < variable name 2 > = < value (optional) >, ... <variable name n > = < value (optional).....

The increment decrement part can contain more than one statement. These are separated by a comma and not by a semicolon. Generally, we use this part only to increment or decrement v ariables, though other statements are also allowed.

```
for ( // code; // code; i++, j++, k+=5 ) {
   //code
}
```

It is not necessary that all the three parts of the for header have to be present. One or all of them may be skipped. Following shows a for header in which only the condition part is specified. The other two parts are left empty.

```
for( ; i<=10; ) {
   // code
}
```

In all the above three decision making statements, if the condition is left empty,
it is taken as true. Such a loop keeps executing for ever and is known as an infinite loop.

```
while ( ) {
   // code
}
```

Infinite loops may also result even after specifying the condition as in the following example:

```
for ( int i=1; i <= 10 ;i--) {
   System.out.println(i);
}
```

Infinite loops are considered to be logical errors. There would be no reason as to why a programmer wishes to run a loop for ever. This would eventually cause the system to crash. Infinite loops result from trivial mistakes. For example, in the just preceding example, i++ was replaced with i—which caused the loop to run continuously.

Loops that do not have a body are known as empty loops. For example, the following while loop is an empty loop:

```
while ( ctr++<10 ) {
}
```

We may also write this loop in the following way:

```
while ( ctr++ < 10 ) ;
```

The above statement is equivalent to the following
which contains an empty statement. As stated
already, whitespaces do not matter in a program.

```
while ( ctr++< 10 );
```

Control statements may be nested within one another. An if else structure may be written Inside a for loop, a for loop can contain another for loop which in turn can contain one more for loop and so on....

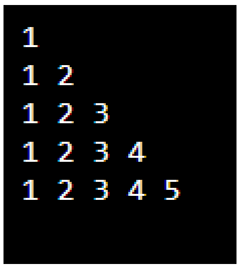
Nested Loops

Nested loops are quite helpful in processing information. We will learn how nested loops are used by taking the example of a pattern problem. If the input is seven,the following below pattern needs to be printed.

One thing that is obvious on observing this pattern is that loops are used to print the pattern. This is because, we are not aware in advance of the number of *'s that are to be printed. Further, we need nested loops: a for loop nested within another for loop. the outer for loop keeps track of the line number. Or, in other words, it is used to count the number of lines we are printing. The inner for loop is used to keep track of the number of *'s we are printing. Further, the number of stars on a particular line is equal to the number line number which in some way can be realted to the outer loop control variable.

Example: 17

```
class Nested {
  public static void main(String[] args) {
    int rows = 5;
    // outer loop
    for (int i = 1; i <= rows; ++i) {
      // inner loop to print the numbers
      for (int j = 1; j <= i; ++j) {
        System.out.print(j + " ");
      }
      System.out.println("");
    }
  }
}
```



```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Example: 18

```
class Nested {
        public static void main(String[] args) {
            int n=7;
           for(int i=1;i<=n;i++) {
               for(int j =1;j<=i;j++){
                   System.out.print("*");
               }
              System.out.println();
               }
           }
        }
```

```
*
**
***
****
*****
******
*******
```

Note that we have used the print() method and not the println() method to display the stars as we want the *'s to be displayed on the same line. And after the inner loop is executed, we go to the next line using the println() statement.

Jump Statements

The jumping statements are the control statements which transfer the program execution control to a specific statement.

Java has three types of jumping statements they are **break, continue**, and **return.** These statements transfer execution control to another part of the program.

Using break

It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

- We can use break statement in the following cases.
- Inside the switch case to come out of the switch block.
- Within the loops to break the loop execution based on some condition.
- Inside labeled blocks to break that block execution based on some condition.

Example:19

```
class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

```
0
1
2
3
```

This example stops the loop when i is equal to 4:

Example :20

```
class Test
{
public static void main(String[] args)
{
for(int j=0; j<10; j++)
{
if(j==5)
{
break;
}
System.out.println(j);
}
System.out.println("outside of for loop");
}
}
```

```
0
```

```
1
2
3
4
Outside of for loop
```

Using continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Example: 21

```
class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

```
0
1
2
```

```
3

5

6

7

8

9
```

This example skips the value of 4:

Example: 22 - To print odd numbers only.

```
class Test {

public static void main(String[] args) {

for(int j=1; j<=100; j++) {

if(j%2==0) {

continue; }

System.out.println(j);

}

}

}
```

```
1
3
5
7
.
```

```
.
99
```

Example: 23 To print Even numbers only.

```
class Test
{
public static void main(String[] args)
{
for(int j=1; j<=100; j++)
{
if(j%2!=0)
{
continue;
}
System.out.println(j);
}
}
}
```

```
    2
    4
    6
    8
    .
    .
```

```
.
100
```

Using return

The return statement is mainly used in methods in order to terminate a method in between and return back to the caller method. It is an optional statement. That is, even if a method doesn't include a return statement, control returns back to the caller method after execution of the method. Return statement may or may not return parameters to the caller method.

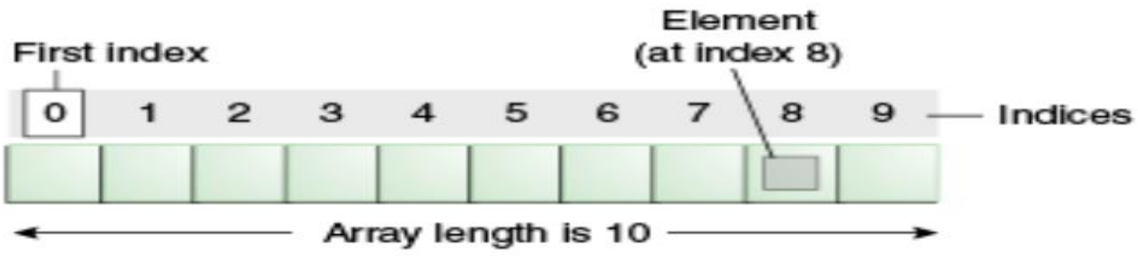
Example :24

```
class Test
{
public static void main(String[] args)
{
Test t = new Test();
int sum = t.addition(10,20);  //addition() method return integer value
System.out.println("Sum = "+sum);
t.show("To Samara");  //show() method does not return any value
}
int addition(int a,int b)
{
return a+b;
}
void show(String name)
{
System.out.println("Welcome "+name);
return; // not returning anything, it is optional
}
}
```

```
Sum = 30
Welcome To Samara
```

3.4. Arrays

Java Array Normally, array is a collection of similar type of elements that have contiguous memory location.
Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements.
We can store only fixed set of elements in a java array. Array in java is index based; first element of the array is stored at 0 indexes and the last element is stored n-1.
Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.



Advantage of Java Array

- Code Optimization: It makes the code optimized; we can retrieve or sort the data easily.
- Random access: We can get any data located at any index position.

Disadvantage of Java Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Syntax to Declare Single Dimensional Array in java

Define the datatype with **square brackets then array name;**

```
dataType[] arr; (or)
 dataType [] arr; (or)
dataType arr[];
```

Example: - int[] myNum;
int [] myNum;
int []myNum;

Initialization of Single Dimensional array

Define the datatype with **square brackets then array name = array value with braces then semicolon;**

```
int[] myNum = {10, 20, 30, 40};

      int [] myNum = {10, 20, 30, 40};

int  []myNum = {10, 20, 30, 40};
```

Instantiation of an Array in java

ArrayRefVar=new datatype [size];

int a[]=new int[5];//declaration and instantiation

Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare instantiate, initialize and traverse an array.

```
class Testarray {

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
 10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Single Dimensional Array

We can declare, instantiate and initialize the java array together by:

```
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
```

```
    }}
```

Output:

```
33

3

4

5
```

Access the Elements of Single Dimensional Array

- We access an array element by referring to the index number.

```
class Array {

 public static void main(String[] args) {

 int[] arrayname = {1,2,3,4,5};

System.out.println(arrayname[0]);

 }

}

1
```

This statement accesses the value of the first element in arrayname

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

1. dataType[][] arrayRefVar; (or)

2. dataType [][]arrayRefVar; (or)

3. dataType arrayRefVar[][]; (or)

4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

int[][] arr=new int[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in java

1. arr[0][0]=1;

2. arr[0][1]=2;

3. arr[0][2]=3;

4. arr[1][0]=4;

5. arr[1][1]=5;

6. arr[1][2]=6;

7. arr[2][0]=7;

8. arr[2][1]=8;

9. arr[2][2]=9;

Example of Multidimensional java array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3{

public static void main(String args[]){

//declaring and initializing 2D array

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
System.out.println();
}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Access the Elements of Multidimensional Array

```
class Array {
 public static void main(String[] args) {
 int[][] arrayname = {{1,2,3},{4,5,6}};
System.out.println(arrayname[0][1]);
 }
}
2
```

Addition of 2 matrices in java

Let's see a simple example that adds two matrices.

```
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};

int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices

int c[][]=new int[2][3];

//adding and printing addition of 2 matrices

for(int i=0;i<2;i++){

for(int j=0;j<3;j++){

c[i][j]=a[i][j]+b[i][j];

System.out.print(c[i][j]+" ");

}

System.out.println();//new line

}

}}
```

Output:
```
2 6 8

6 8 10
```