Learning Java - A Foundational Journey

Session: 3

Decision-Making and Loops

For Aptech Centre Use Only

Objectives

- List different types of decision-making statements
- Explain the if statement and various forms of if statement
- Explain switch-case statement
- Compare the if-else and switch-case statement
- List the different types of loops
- Explain the while statement and the associated rules
- Identify the purpose of the do-while statement
- State the need of for statement
- Describe nested loops
- Compare the different types of loops
- State the purpose of jump statements
- Describe break statement
- Describe continue statement

For Aptech Centre Use Only

Introduction

- A Java program consists of a set of statements executed sequentially in the order in which they appear.
- Three categories of control flow statements supported by Java are:

Conditional Statements	• These types of statements are also referred to as decision-making statements.
Iteration Statements	• These types of statements are also referred to as looping constructs.
Branching Statements	• These types of statements are referred to as jump statements.

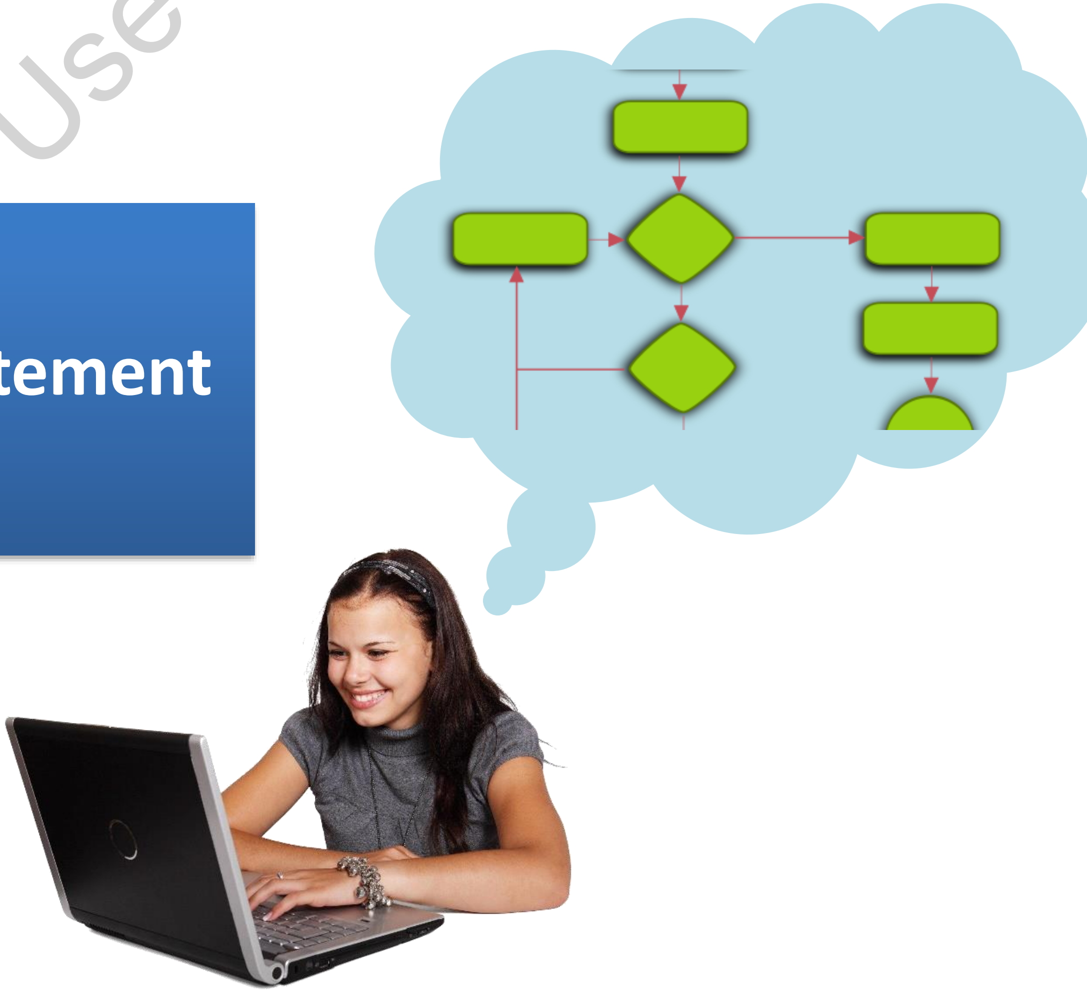
For Aptech Centre Use Only

Decision-making Statements

- Enable us to change flow of execution of a Java program.
- Evaluate a condition and based on the result of evaluation, a statement or a sequence of statements is executed
- Different types of decision-making statements:

`if` Statement

`switch-case` Statement



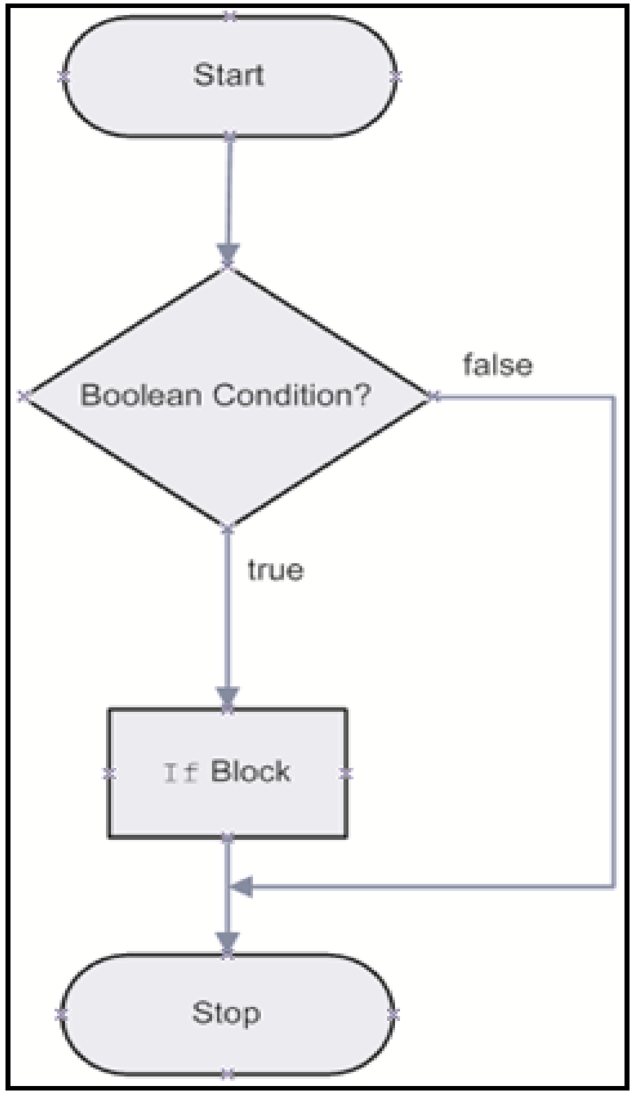
For Aptech Centre Use Only

‘if’ Statement

- It is the most basic form of decision-making statement.
- It evaluates a given condition and based on the result of evaluation executes a certain section of code.

Syntax

```
if (condition) {

    // one or more statements;

}
```



For Aptech Centre Use Only

'if-else' Statement

- `if-else` statement helps to define a block of statements to be executed when a condition evaluates to false.
- `if-else` Statement:
  - Begins with `if` block followed by `else` block
  - `else` block specifies a block of statements to be executed when a condition evaluates to false

Syntax

```
if (condition) {
     // one or more statements;
}
else {
     // one or more statements;
}
```

For Aptech Centre Use Only

Nested-if Statement

An `else` statement should always refer to the nearest `if` statement.

The `if` statement must be within the same block as `else` and it should not be already associated with some other `else` statement.

Syntax

```
if(condition) {

     if(condition)
       true-block statement(s);
     else
       false-block statement(s);
     }

else {
       false-block statement(s);
}
```

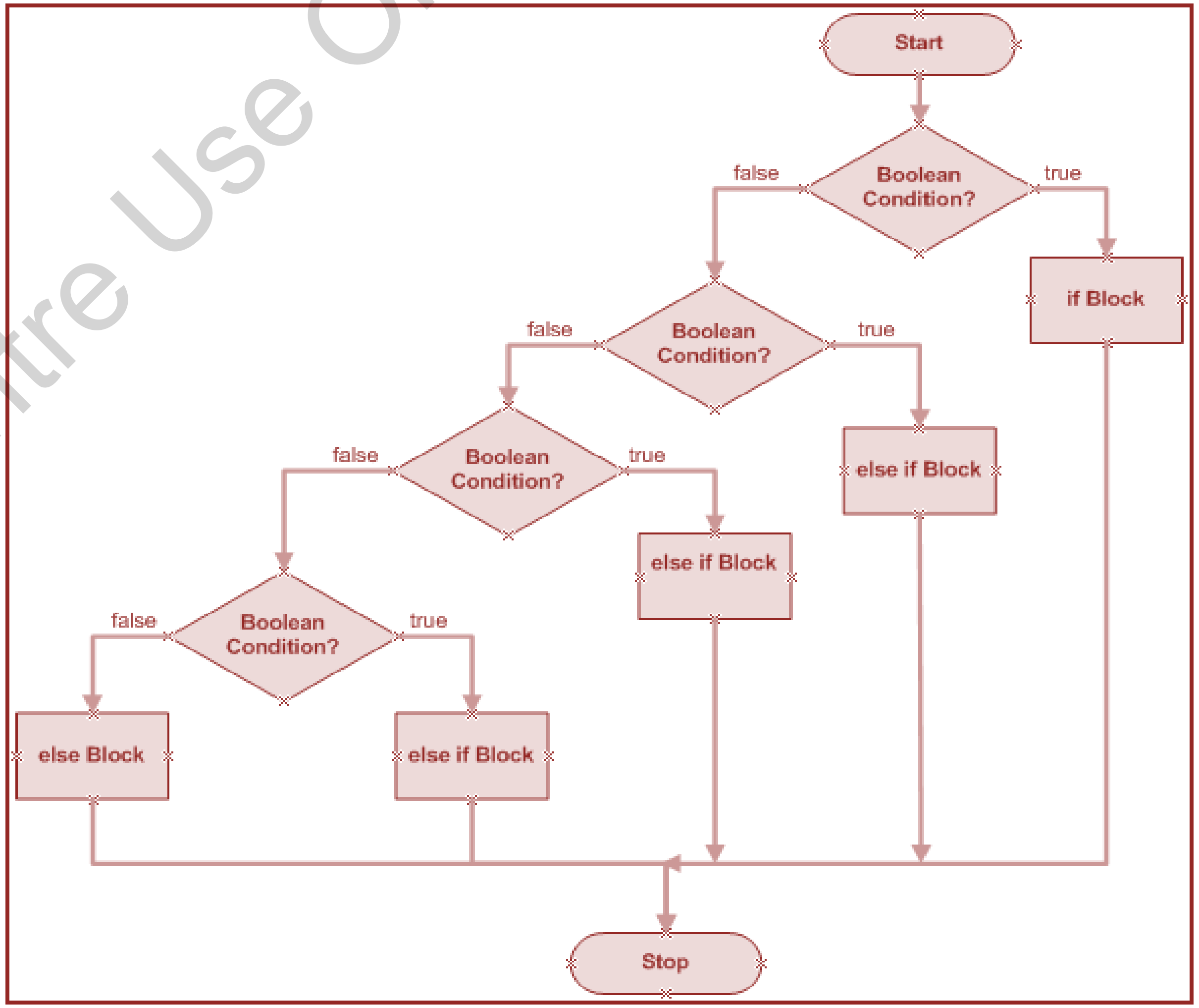
For Aptech Centre Use Only

‘if-else-if’ Ladder

- The multiple if construct is known as the `if-else-if` ladder.
- The conditions are evaluated sequentially starting from the top of the ladder and moving downwards.
- If none of the conditions is true, then the final `else` statement, also referred to as default statement, is executed.

Syntax

```
if(condition) {
    // one or more statements
}

else if (condition) {
    // one or more statements
}

else {
    // one or more statements
}
```



For Aptech Centre Use Only

‘switch-case’ Statement 1-3

- Alternative for too many `if` statements representing multiple selection constructs.
- Contains a variable as an expression whose value is compared against different values.
- Can evaluate different primitive data types, such as `byte`, `short`, `int`, and `char`.

Enhancements to switch-case statement in Java SE 7 and later versions

- Supports use of strings in the `switch-case` statement.
- String variable can be passed as an expression for the `switch` statement.
- Supports use of objects from classes present in Java API.
- The classes whose objects can be used are `Character`, `Byte`, `Short`, and `Integer`.
- Supports use of enumerated types as expression.

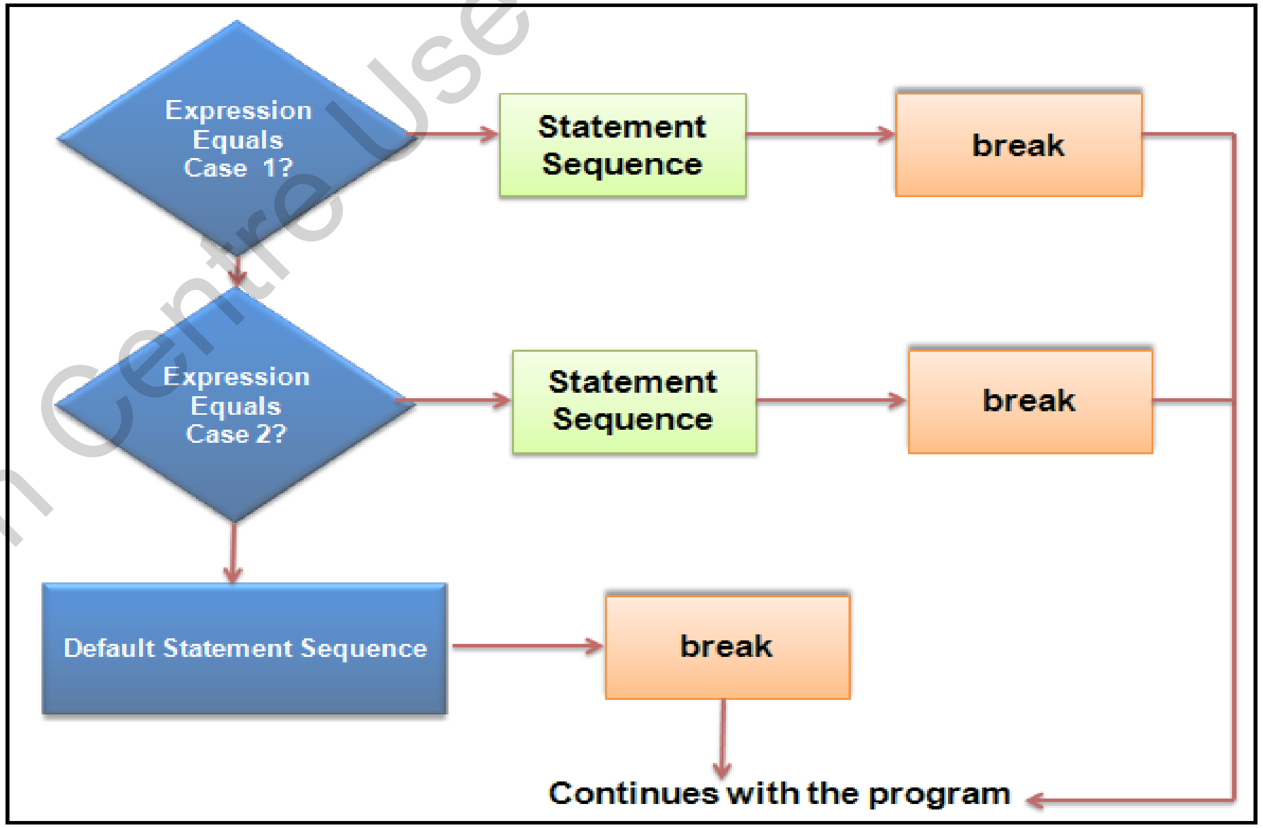
For Aptech Centre Use Only

‘switch-case’ Statement 2-3

Syntax

```
switch (<expression>) {
 case value1:
        // statement sequence
           break;
 case value2:
        // statement sequence
           break;
 . . .
 . . .
 . . .
 case valueN:
        // statement sequence
           break;
 default:
        // default statement sequence
}
```

Following figure shows the flow of execution for the switch-case statement:


Expression Equals Case 1?
Statement Sequence
break
Expression Equals Case 2?
Statement Sequence
break
Default Statement Sequence
break
Continues with the program

For Aptech Centre Use Only

‘switch-case’ Statement 3-3

The value of the expression specified with the `switch` statement is compared with each case constant value.

If any `case` value matches, the corresponding statements in that `case` are executed.

When the `break` statement is encountered, it terminates the `switch-case` block and control switches to the statements following the block.

The `break` statement must be provided as without it, even after the matching case is executed; all other cases following the matching case are also executed.

If there is no matching case, then the `default` case is executed.

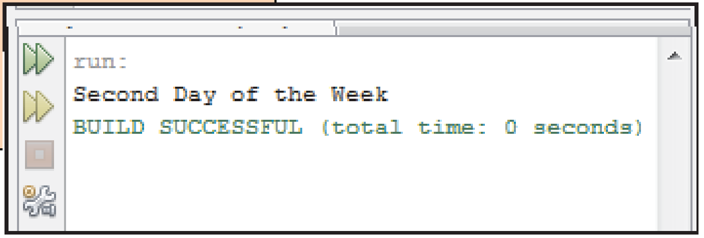
For Aptech Centre Use Only

String-based 'switch-case' Statement 1-2

- Java SE 7 and later versions support use of strings in the `switch-case` statement.
- A `String` is not a primitive data type, but an object in Java.

Code Snippet:

```
public class DayofWeek {
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
String day = "Monday";
// switch statement contains an expression of type String
switch (day) {
case "Sunday":
System.out.println("First day of the Week");
break;
case "Monday":
System.out.println("Second Day of the Week");
break;
case "Tuesday":
System.out.println("Third Day of the Week");
break;
case "Wednesday":
System.out.println("Fourth Day of the Week");
break;
case "Thursday":
System.out.println("Fifth Day of the Week");
break;
case "Friday":
System.out.println("Sixth Day of the Week");
break;
case "Saturday":
System.out.println("Seventh Day of the Week");
break;
default:
System.out.println("Invalid Day");
} // End of switch-case statement
}
}
```



```
run:
Second Day of the Week
BUILD SUCCESSFUL (total time: 0 seconds)
```

For Aptech Centre Use Only

String-based 'switch-case' Statement 2-2

- Following points are to be considered while using strings with the `switch-case` statement:

Null Values

- A runtime exception is generated when a `String` variable is assigned a `null` value and is passed as an expression to the `switch` statement.

Case-sensitive values

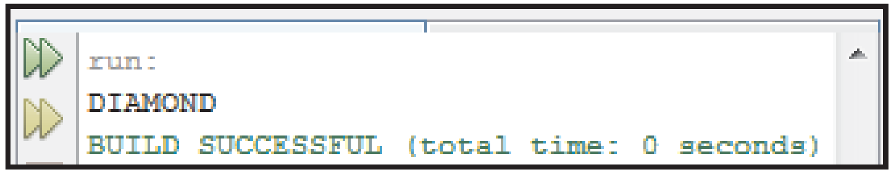
- The value of `String` variable that is matched with the case literals is case sensitive.
- Example: a `String` value "**`Monday`**" when matched with the case labeled "**`MONDAY`**"`:`, then it will not be treated as a matched value.

For Aptech Centre Use Only

Enumeration-based ‘switch-case’ Statement

- The `switch-case` statement supports the use of an enumeration (`enum`) value in the expression.
- The constraint with an `enum` expression is that:
  - All case constants must belong to the same `enum` variable used with the `switch` statement.

```
public class TestSwitchEnumeration {
/**
* An enumeration of Cards Suite
*/
enum Cards {
Spade, Heart, Diamond, Club
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
Cards card = Cards.Diamond;
// enum variable is used to control a switch
statement
switch (card) {
case Spade:
System.out.println("SPADE");
break;
case Heart:
System.out.println("HEART");
break;
case Diamond:
System.out.println("DIAMOND");
break;
case Club:
System.out.println("CLUB");
break;
} // End of switch-case statement
}
}
```

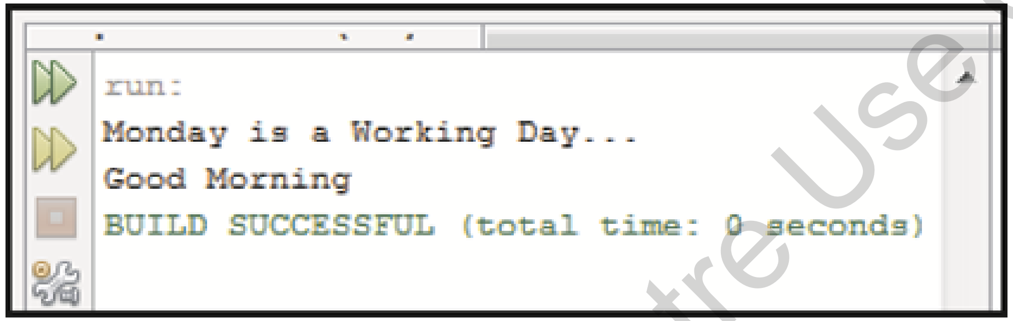


```
run:
DIAMOND
BUILD SUCCESSFUL (total time: 0 seconds)
```

For Aptech Centre Use Only

Nested ‘switch-case’ Statement

- A `switch-case` statement can be used as a part of another `switch-case` statement. This is referred to as nested `switch-case` statements.



```
run:
Monday is a Working Day...
Good Morning
BUILD SUCCESSFUL (total time: 0 seconds)
```

Three important features of `switch-case` statements are as follows:

- The switch-case statement differs from the if statement, as it can only test for equality.

- No two case constants in the same switch statement can have identical values, except the nested switch-case statements.

- A switch statement is more efficient and executes faster than a set of nested-if statements.

For Aptech Centre Use Only

Comparison Between if and switch-case Statement

if	switch-case
Each `if` statement has its own logical expression to be evaluated as `true` or `false`	Each case refers back to the original value of the expression in the `switch` statement
The variables in the expression may evaluate to a value of any type	The expression must evaluate to a `byte`, `short`, `char`, `int`, or `String`
Only one of the blocks of code is executed	If the `break` statement is omitted, the execution will continue into the next block

For Aptech Centre Use Only

Loops

- A computer program consists of a set of statements, which are usually executed sequentially.
- However, in certain situations, it is necessary to repeat certain steps to meet a specified condition.

- For example,

  1+2=3

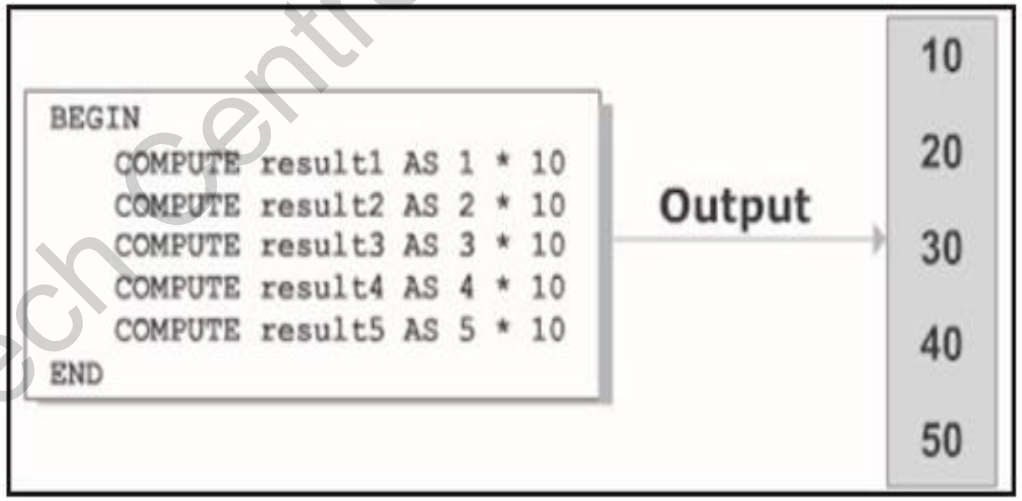
  3+3=6

  6+4=10

  10+5=15

  15+6=21

  . . . . and so on.



```
BEGIN
    COMPUTE result1 AS 1 * 10
    COMPUTE result2 AS 2 * 10
    COMPUTE result3 AS 3 * 10
    COMPUTE result4 AS 4 * 10
    COMPUTE result5 AS 5 * 10
END
```

```
10
20
30
40
50
```

For Aptech Centre Use Only

Looping Statements

- A loop consists of statement or a block of statements that are repeatedly executed, until a condition evaluates to true or false.
- The loop statements supported by Java programming language are as follows:
  - `while` statement
  - `do-while` statement
  - `for` statement
  - `for-each` statement

For Aptech Centre Use Only

while Statement

- The `while` statement is the most fundamental looping statement in Java. It is used to execute a statement or a block of statements until the specified condition is true.

Syntax

```
while (expression) {
// one or more statements
}
```

where,

expression: Is a conditional expression which must return a boolean value, that is, true or false.

For Aptech Centre Use Only

Rules for Using while Loop

Following points should be noted when using `while` statement:

Value of the variables used in the expression must be set at some point before the while loop is reached.

This process is called the initialization of variables and has to be performed once before the execution of the loop. For example, num = 1;

The body of the loop must have an expression that changes the value of the variable which is a part of the loop's expression. For example, num++; or num--;

For Aptech Centre Use Only

Infinite Loop

- An infinite loop is one which never terminates.
- The loop runs infinitely when the conditional expression or the increment/decrement expression of the loop is missing.
- Any type of loop can be an infinite loop.

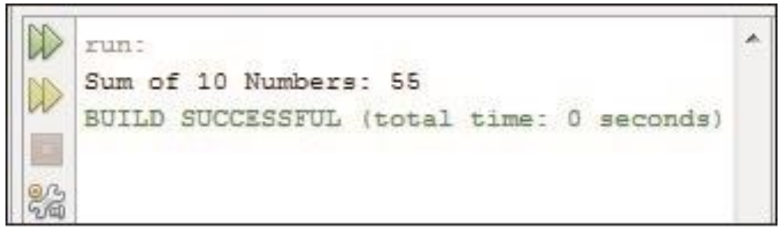
```
public class InfiniteWhileLoop {
     /**
*    @param args the command line arguments
     */
     public static void main(String[] args) {
         /*
*    Loop begins with a boolean value true and is executed
*    infinitely as the terminating condition is missing
         */
         while (true) {
             System.out.println("Welcome to Loops...");   } //End
        of the while loop
     }
}
```

For Aptech Centre Use Only

do-while Statement

- The `do-while` statement checks the condition at the end of the loop rather than at the beginning.
- The condition of the `do-while` statement usually comprises a condition expression that evaluates to a boolean value.

```
public class SumOfNumbers {
 /**
@param args the command line arguments
 */
public static void main(String[] args)
{
int num = 1, sum = 0;
 /* The body of the loop is executed first, then the condition is
evaluated*/
do {  sum = sum + num;  num++;
 } while (num <= 10);
 // Prints the value of variable after the loop terminates
System.out.printf("Sum of 10 Numbers: %d\n", sum);
 }
}
```



```
run:
Sum of 10 Numbers: 55
BUILD SUCCESSFUL (total time: 0 seconds)
```

For Aptech Centre Use Only

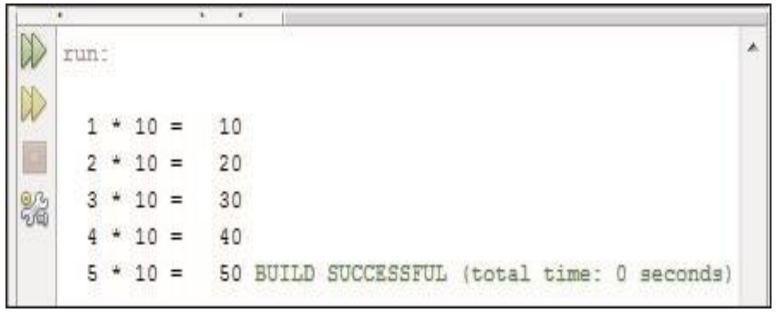
for Statement

- The `for` loop is especially used when user knows the number of times statements are required to be executed. It is similar to `while` statement in its function.

Syntax

```
for(initialization; condition; increment/decrement) {
          // one or more statements
         }
```

```
public class PrintMultiplesWithForLoop {
 /**
 * @param args the command line arguments
 */
public static void main(String[] args) {
   int num, product;
    // The for Loop with all the three declaration  parts
   for (num = 1; num <= 5; num++) {
   product = num * 10;;
   System.out.printf("\n % d * 10 = % d ", num, product);
   } // Moves the control back to the for loop
   }
}
```



```
run:

 1 * 10 =   10
 2 * 10 =   20
 3 * 10 =   30
 4 * 10 =   40
 5 * 10 =   50 BUILD SUCCESSFUL (total time: 0 seconds)
```

For Aptech Centre Use Only

Scope of Control Variables in for Statement

- Mostly control variables are used within `for` loops and may not be used further in the program.
- In such a situation, it is possible to restrict the scope of variables by declaring them at the time of initialization.

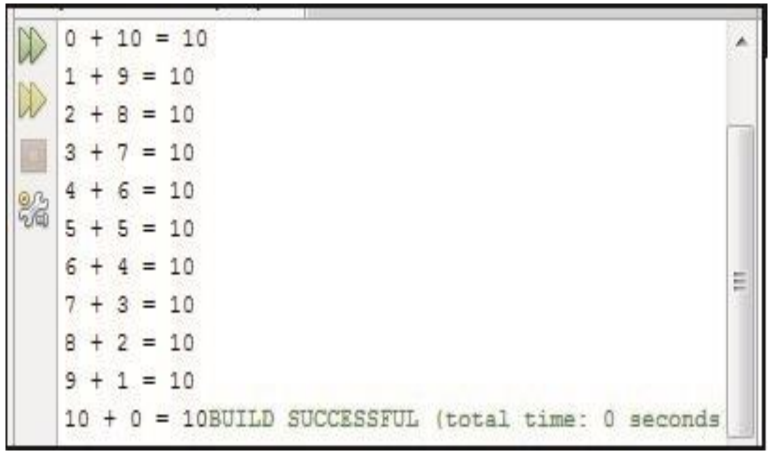
```
public class ForLoopWithVariables {
  /**
  * @param args the command line arguments
  */
  public static void main(String[] args) {
    int product;
    // The counter variable, num is declared inside the for loop
    for (int num = 1; num <= 5; num++) {
          product = num * 10;;
          System.out.printf("\n %d * 10 = %d ", num, product);
    } // End of the for loop
  }
}
```

For Aptech Centre Use Only

Use of Comma Operator in for Statement

- The `for` statement can be extended by including more than one initialization or increment expressions in the `for` loop specification. The expressions are separated by using the 'comma' (,) operator and evaluated from left to right.

```
public class ForLoopWithComma {
 /**
 * @param args the command line arguments
 */  public static void main(String[] args) {
 int i, j;  int max = 10;
 /* The initialization and increment/decrement section includes more
than one variable */
    for (i = 0, j = max; i <= max; i++, j--) {
 System.out.printf("\n%d + %d = %d", i, j, i + j);
 }
 }
}
```

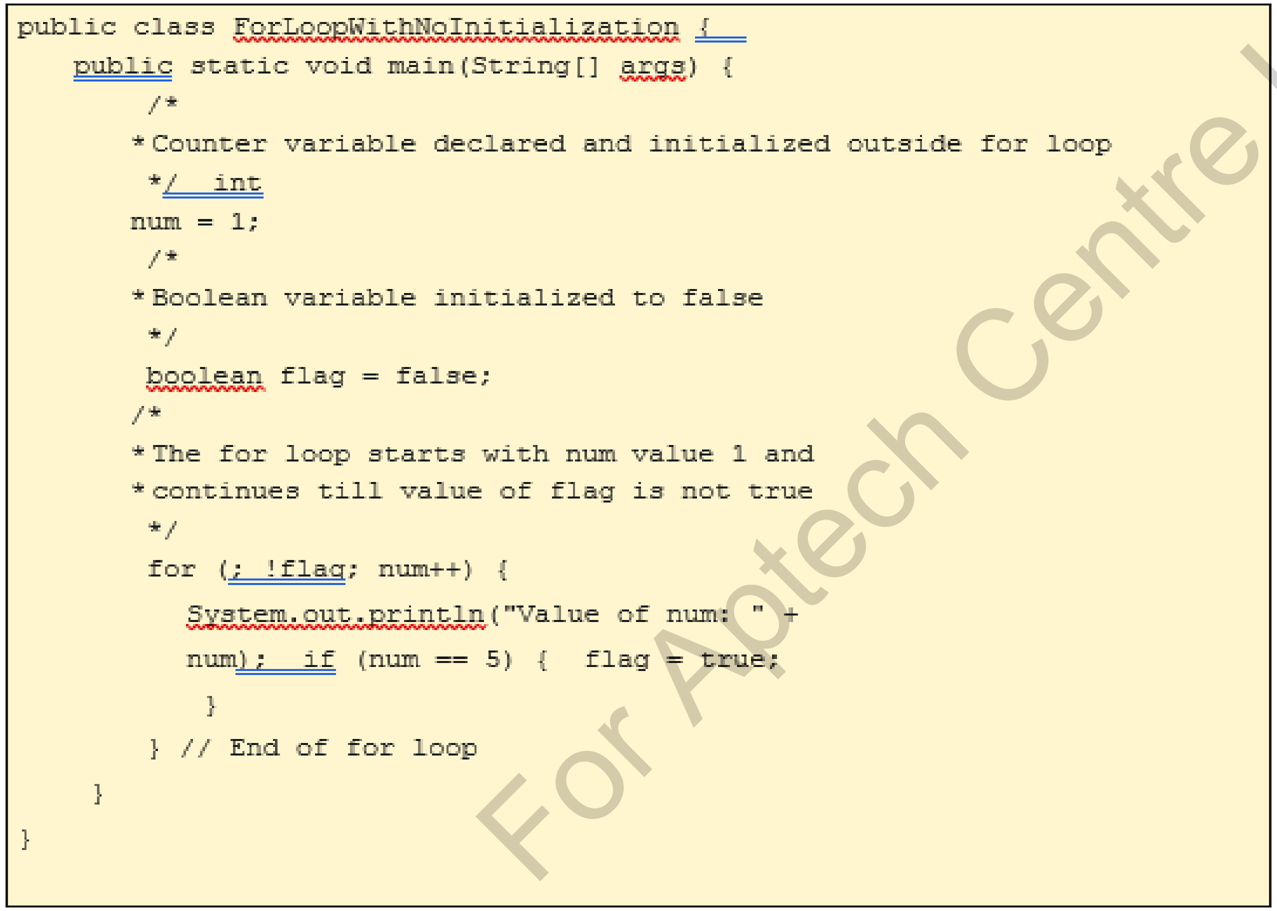


```
0 + 10 = 10
1 + 9 = 10
2 + 8 = 10
3 + 7 = 10
4 + 6 = 10
5 + 5 = 10
6 + 4 = 10
7 + 3 = 10
8 + 2 = 10
9 + 1 = 10
10 + 0 = 10BUILD SUCCESSFUL (total time: 0 seconds
```

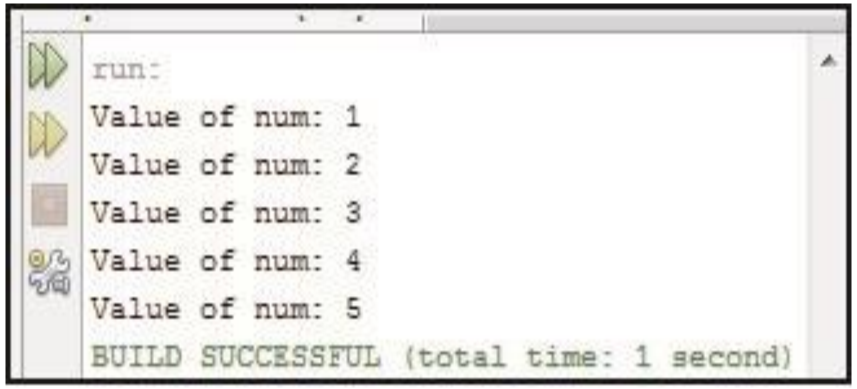
For Aptech Centre Use Only

Variation in for Loop

- The `for` loop is very powerful and flexible in its structure.
- The most common variation involves the conditional expression. Mostly, the conditional expression is tested with the targeted values, but, it can also be used for testing boolean expressions.



```
public class ForLoopWithNoInitialization {
    public static void main(String[] args) {
         /*
        * Counter variable declared and initialized outside for loop
         */  int
        num = 1;
         /*
        * Boolean variable initialized to false
         */
         boolean flag = false;
        /*
        * The for loop starts with num value 1 and
        * continues till value of flag is not true
         */
         for (; !flag; num++) {
           System.out.println("Value of num: " +
           num);  if (num == 5) {  flag = true;
             }
         } // End of for loop
      }
}
```



```
run:
Value of num: 1
Value of num: 2
Value of num: 3
Value of num: 4
Value of num: 5
BUILD SUCCESSFUL (total time: 1 second)
```

For Aptech Centre Use Only

Infinite for Loop

- If all the three expressions are left empty, then it will lead to an infinite loop.
- The infinite `for` loop will run continuously because there is no condition specified to terminate it.

```
.....
for( ; ; ) {
 System.out.println("This will go on and on");
}
....
.
```

For Aptech Centre Use Only

Enhanced for Loop

- Java SE 5 onwards supports enhanced `for` loop to increase the readability of the loop.
- The enhanced for loop is designed to retrieve or traverse through a collection of objects, such as an array.
- The classes are defined in the collection framework and are used to store objects.

Syntax

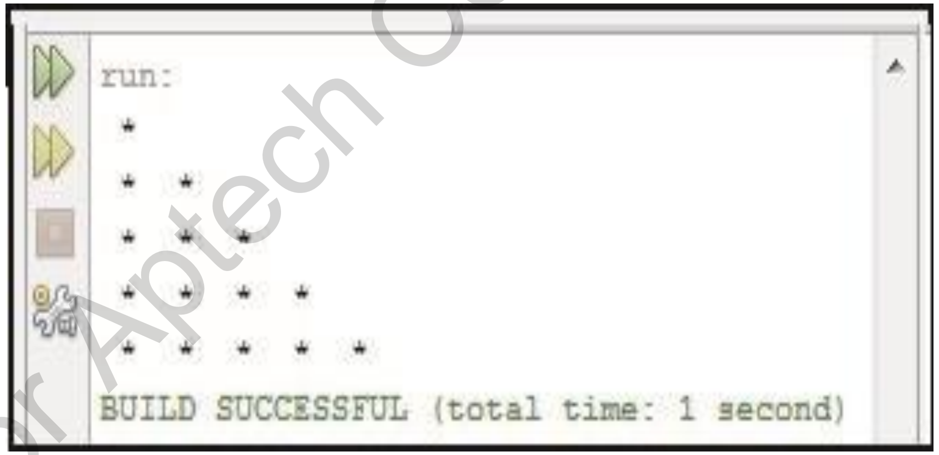
```
for (type var: collection) {
    // block of statement
    }
```

for Loop	Enhanced for Loop
`type var; for (int i = 0; i < arr. length; i++) { var = arr[i]; . . . }`	`for (type var : arr){ . . . // Body of the loop . . . }`

For Aptech Centre Use Only

Nested Loops

- The placing of a loop statement inside the body of another loop statement is called nesting of loops.
- For example, a `while` statement can be enclosed within a `do-while` statement and a `for` statement can be enclosed within a `while` statement.
- When you nest two loops, the outer loop controls the number of times the inner loop is executed.



```
run:
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
BUILD SUCCESSFUL (total time: 1 second)
```

For Aptech Centre Use Only

Comparison of Loops

while/for	do-while
Loop is pre-tested. The condition is checked before the statements within the loop are executed.	Loop is post-tested. The condition is checked after the statements within the loop are executed.
The loop does not get executed if the condition is not satisfied at the beginning.	The loop gets executed at least once even if the condition is not satisfied at the beginning.

For Aptech Centre Use Only

Jump Statements

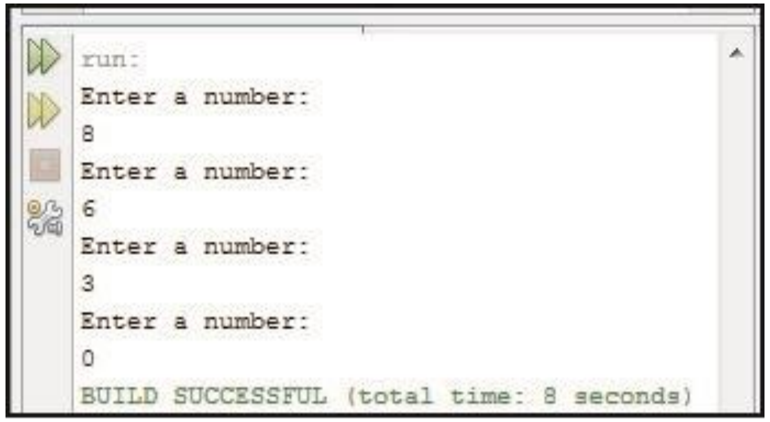
- Java supports jump statements that unconditionally transfer control to locations within a program known as target of jump statements.
- Java provides two keywords: `break` and `continue` that serves diverse purposes.
- However, both are used within loops to change the flow of control based on conditions.

For Aptech Centre Use Only

break Statement

- The `break` statement in Java is used in two ways.
- It terminates a case in the `switch` statement and then, bypasses the loop’s normal conditional test.

```
import java.util.Scanner; public

class AcceptNumbers {
     /**

     * @param args the command line arguments

     */

   public static void main(String[] args) {

   int count, number; // count variable is a counter variable

   for (count = 1, number = 0; count <= 10; count++) {
         // Scanner class is used to accept data from the keyboard

         Scanner input = new Scanner(System.in);

         System.out.println("Enter a number: ");

         number = input.nextInt();

         if (number == 0) {

          // break statement terminates the loop  break;
         } // End if statement
       } // End of for statement

     }

}
```

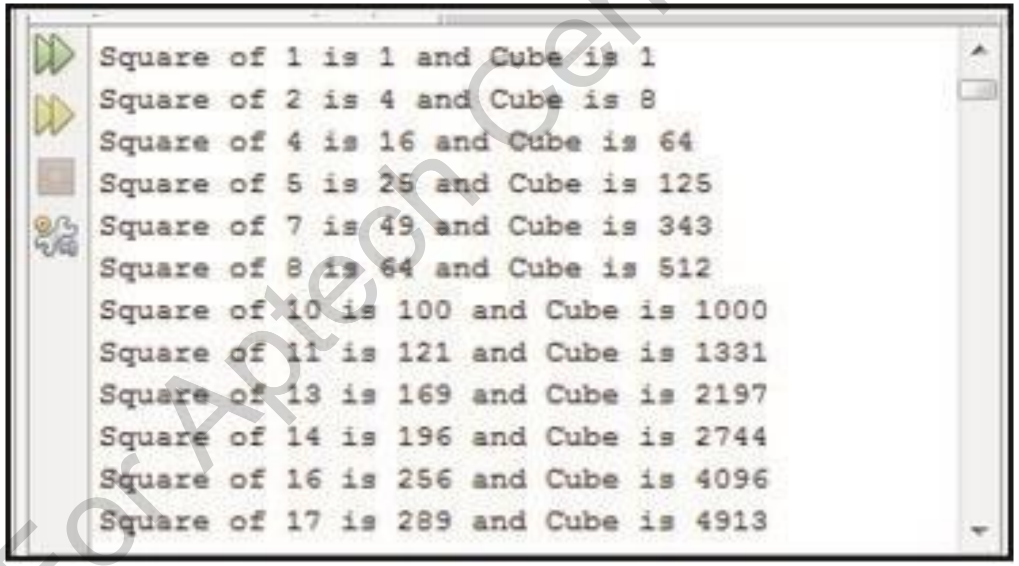


```
run:
Enter a number:
8
Enter a number:
6
Enter a number:
3
Enter a number:
0
BUILD SUCCESSFUL (total time: 8 seconds)
```

For Aptech Centre Use Only

continue Statement

- Java provides another keyword named `continue` to skip statements within a loop and proceed to the next iteration of the loop.
- In `while` and `do-while` loops, a continue statement transfers the control to the conditional expression which controls the loop.



```
Square of 1 is 1 and Cube is 1
Square of 2 is 4 and Cube is 8
Square of 4 is 16 and Cube is 64
Square of 5 is 25 and Cube is 125
Square of 7 is 49 and Cube is 343
Square of 8 is 64 and Cube is 512
Square of 10 is 100 and Cube is 1000
Square of 11 is 121 and Cube is 1331
Square of 13 is 169 and Cube is 2197
Square of 14 is 196 and Cube is 2744
Square of 16 is 256 and Cube is 4096
Square of 17 is 289 and Cube is 4913
```

For Aptech Centre Use Only

Labeled Statements 1-2

- Java does not support `goto` statements, as they are difficult to understand and maintain.
- It can be used within constructs to control the flow of statements.
- For example, to exit from a deeply nested set of loops, `goto` statement can be useful.
- Java defines an expanded form of `break` and `continue` statements which can be used within any block.
- It is not necessary that the blocks must be part of loop or a `switch` statement.

Syntax

```
break label;
```

where,

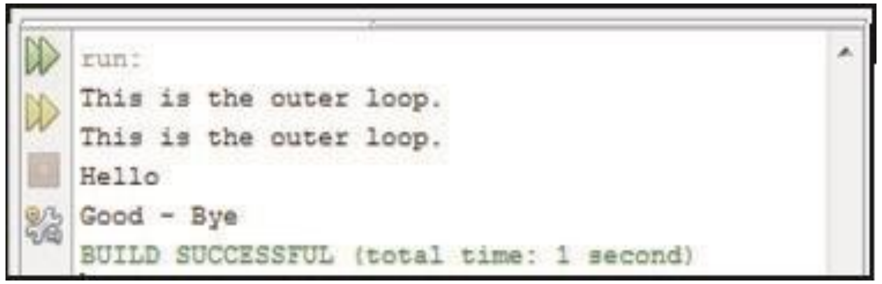
label: Is an identifier specified to put a name to a block. It can be any valid Java identifiers followed by a colon.

For Aptech Centre Use Only

Labeled Statements 2-2

```
public class NumberPyramid {
    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        outer:
        for (int i = 1; i < 5; i++) {
            for (int j = 1; j < 5; j++) {
                if (j > i) {
                    System.out.println();
                    /* Terminates the loop counting j and continues the
                    * next iteration of the loop counting i
                    */
                    continue outer;
                } // End of if statement
                System.out.print(j);
            } // End of inner for loop
            System.out.println("\nThis is the outer loop.");
        } //End of outer for loop
    System.out.println("Good-Bye");
```

For Aptech Centre Use Only



```
run:
This is the outer loop.
This is the outer loop.
Hello
Good - Bye
BUILD SUCCESSFUL (total time: 1 second)
```

Summary

- A Java program is a set of statements, which are executed sequentially in the order in which they appear.
- The three categories of control flow statements supported by Java programming language include: conditional, iteration, and branching statements.
- The if statement is the most basic decision-making statement that evaluates a given condition and based on result of evaluation executes a certain section of code.
- The if-else statement defines a block of statements to be executed when a condition is evaluated to false.
- The multiple if construct is known as the if-else-if ladder with conditions evaluated sequentially from the top of the ladder.
- The switch-case statement can be used as an alternative approach for multiple selections. It is used when a variable must be compared against different values. Java SE 7 and higher support strings and enumerations in the switch-case statement.
- A switch statement can also be used as a part of another switch statement. This is known as nested switch-case statements.

For Aptech Centre Use Only