
Java Language Fundamentals
- Programming Constructs
©2015 Manipal Global Education

manipalglobal
education services

INTRODUCTION

Programming Constructs

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Illustrate selection constructs
- Use iterative constructs
- Understand branching statements

Refer package **com.mgait.fundamentals** in the provided code base for demo programs on the topics covered in this presentation

Programming Constructs - Control Structure

- A control structure refers to the way in which the programmer specifies the order of executing the statements
- The following approaches can be chosen depending on the problem statement:
  - Sequential - In a sequential approach, all the statements are executed in the same order as it is written
  - Selectional - In a selectional approach, based on some conditions, different set of statements are executed
  - Iterational (Repetition) - In an iterational approach certain statements are executed repeatedly

CONCEPT

Selectional Constructs

SELECTION CONSTRUCT - OVERVIEW

- Selection constructs allows the programmer to control the flow of the program's execution based upon conditions evaluated during run time
- Java supports two selection statements:
  - if – else construct
  - switch - case construct

manipalglobal
education services

IF STATEMENT

- Conditional branching statement
- Can be used to route program execution through two different paths.
- Syntax

```
if (condition)
        statement1;
else
        statement2;
```

If the *condition* is true, then *statement1* is executed. Otherwise, *statement2* is executed.

```
if (condition){
        statement1;
        statement2;
}
else{
        statement3;
        statement4;
}
```

IF STATEMENT

DEMO
Class :

- The following snippet finds whether the given number is odd or even.

```
public static void main(String args[]) {
        int num = 13;
        if ( num % 2 == 0 )
                System.out.println(num + " is even ");
        else
                System.out.println(num + " is odd ");
}
```

NESTED IF STATEMENT

- An 'if-else' statement embedded within another 'if-else' statement is called as nested 'if'.

- Code snippet to find the maximum among three values

```
if ( a > b ) {
        if ( a > c}
                max = a;
        else
                max = b;
} else {
        if ( b > c}
                max = b;
        else
                max = c;
}
```

ELSE-IF LADDER CONSTRUCT

- The 'else if' statement is to check for a sequence of conditions
- When one condition is false, it checks for the next condition and so on
- When all the conditions are false the 'else' block is executed
- The statements in that conditional block are executed and the other 'if' statements are skipped

SYNTAX

```
if(condition)
   statement;
else if(condition)
   statement;
else if(condition)
   statement;
    ...
else
   statement;
```

manipalglobal
education services

ELSE-IF LADDER EXAMPLE

DEMO
Class :

- Code snippet to display the day of the week based on the week code

```
int weekCode = 5;

if ( weekCode == 1 )
  System.out.println("Monday");
else if ( weekCode == 2 )
  System.out.println("Tuesday");
else if ( weekCode == 3 )
.
.
.
else if ( weekCode == 7 )
  System.out.println("Sunday");
```

SWITCH – CASE CONSTRUCT

- The 'switch' statement is a selectional construct that selects a choice from the set of available choices
- Provides an alternative to if-else-if ladder
- The expression must be of type
  - Java 6 - byte, short, int, char or enum
  - Java 7 – Strings included
- *case* statements must have literals compatible with the expression

```
SYNTAX

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

SWITCH – CASE EXAMPLE

DEMO
Class :

- Code snippet to display the day of the week based on the week code

```
int weekCode = 5;
switch (weekCode) {
        case 1:
                System.out.println("Monday");
                break;
        case 2:
        .
        .
        case 7:
                System.out.println("Sunday");
                break;
        default:
                System.out.println("Invalid code");
}
```

SWITCH – FALLTHROUGH

- The break statement is optional
- If break is omitted, and a case matches, execution will continue to the next cases below the matched case statement

display week day or week end based on the week code

```
int weekCode = 5;
switch (weekCode) {
     case 1:
     case 2:
     case 3:
     case 4:
     case 5:
          System.out.println("Week Day");
     break;
     case 6:
     case 7:
          System.out.println("Week End");
          break;
     default:
          System.out.println("Invalid code");

}
```

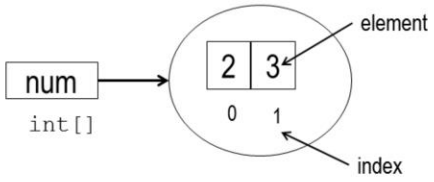
CONCEPT

Array Basics

ARRAY

- Container object that holds a fixed number of values of a same type
- Size of array is fixed after declaration and cannot be changed

```
int[] num = {2,3};        // Array of 2 elements
```



- Elements are accessed using array index

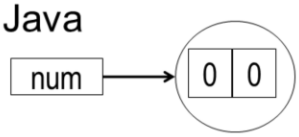
```
num[0] = 5;                    //setting element value using index

System.out.println(num[1]);    //accessing element using index
```

ARRAY – CREATING USING NEW OPERATOR

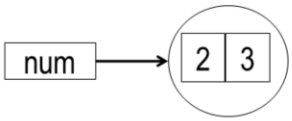
- **new** keyword is used to create an object in Java

```
int[] num = new int[2];
```



  - Elements of array are initialized with default values based on the data type

```
num[0] = 2;
num[1] = 3;
```



- length property gives the size of the array

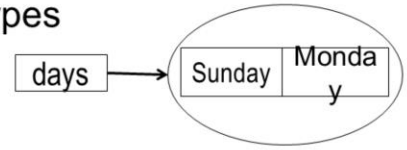
```
System.out.println(num.length);      // 2
```

manipalglobal
education services

ARRAY OF REFERENCE DATA TYPES

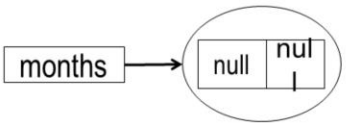
- Array can be of primitive or reference data types

```
String[] days = {"Sunday", "Monday"};
```

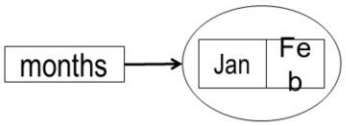


- Creating using new operator

```
String[] months = new String[2];
```




```
months[0] = "Jan";
months[1] = "Feb";
```



Refer following EduNxt Video for Two Dimensional Array

- M3L6L4_Demonstration_of_two_dimensional_arrays - Demo

CONCEPT

Iterational Constructs

ITERATIONAL CONSTRUCTS - OVERVIEW

- Iterational constructs allow a programmer to repeatedly execute the same set of instructions until a termination condition is met
- Are commonly called as *loops.*
- Java's iterational construct statements are
  - for
  - while
  - do-while

WHILE LOOP

DEMO
Class :

- Is the most fundamental loop statement.
- The boolean expression is evaluated first and the loop body repeats until the boolean expression is false
- Entry Controlled Loop

```
while(boolean expression) {

                // body of loop

        }
```

DO - WHILE LOOP

DEMO
Class

- Executes the body of a loop at least once, even if the boolean expression evaluates to false
- Can be used when the condition needs to be checked at the end of the loop rather than at the beginning.
- Exit Controlled Loop

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

FOR LOOP

- The general form of the **for** loop is given below:

  for(*initialization; condition; iteration*)

  statement;

- *The initialization* portion sets the loop control variable to an initial value.
- The *condition* is a boolean expression that checks the loop control variable. If the outcome is true, the loop continues to iterate. Otherwise, the loop terminates.
- The *iteration* expression determines how the loop control variable is changed during each iteration.

FOR LOOP

DEMO
Class :

- Code snippet to find the sum of first ten natural numbers

```
int sum  = 0;
for( int i = 1; i <= 10; i++){
    sum += i;
}
System.out.println(" The sum is " + sum );
```

CONCEPT

Branching Statements

BRANCHING STATEMENTS

- Transfer the control to another part of the program.
- Java supports following branching statements:
  - break
  - continue

BREAK

- The break statement is used to force the termination of a loop or exit a switch

Add the numbers entered by user until user enters -1

```
Scanner sc= new Scanner(System.in);
 int num = 0; int sum = 0;
 while (true) {
       System.out.println("Enter the number");
       num = sc.nextInt();
       sum += num;
       if(num == -1) break;
 }
 System.out.println(sum);
```

manipalglobal
education services

CONTINUE

DEMO
Class

- continue statement forces the next iteration of the loop to take place and skips the code between continue statement and the end of the loop.

```
for(...){
  for(...) {
        . . .
        if(cond1)
          continue;
         . . .
  }
  . . .
  if(cond2)
    continue;
  . . .
}
```

manipalglobal
education services

ENHANCED FOR LOOP

DEMO
Class :

➢ The enhanced for loop is a specialized for loop that simplifies looping through an array or a collection

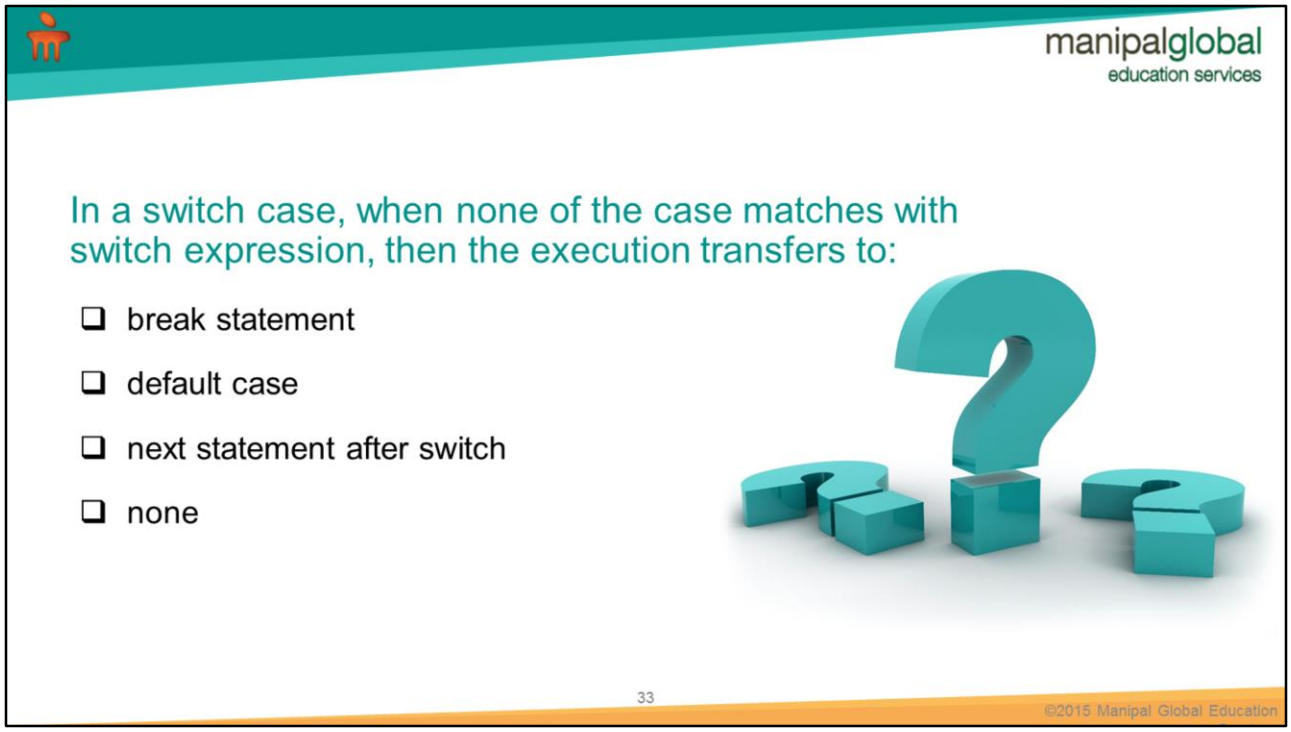
```
for(declaration : expression){
    ..
}
```

```
String[] months = {"jan","feb","mar"};
for (String name : months) {
        System.out.println(name);
}
```

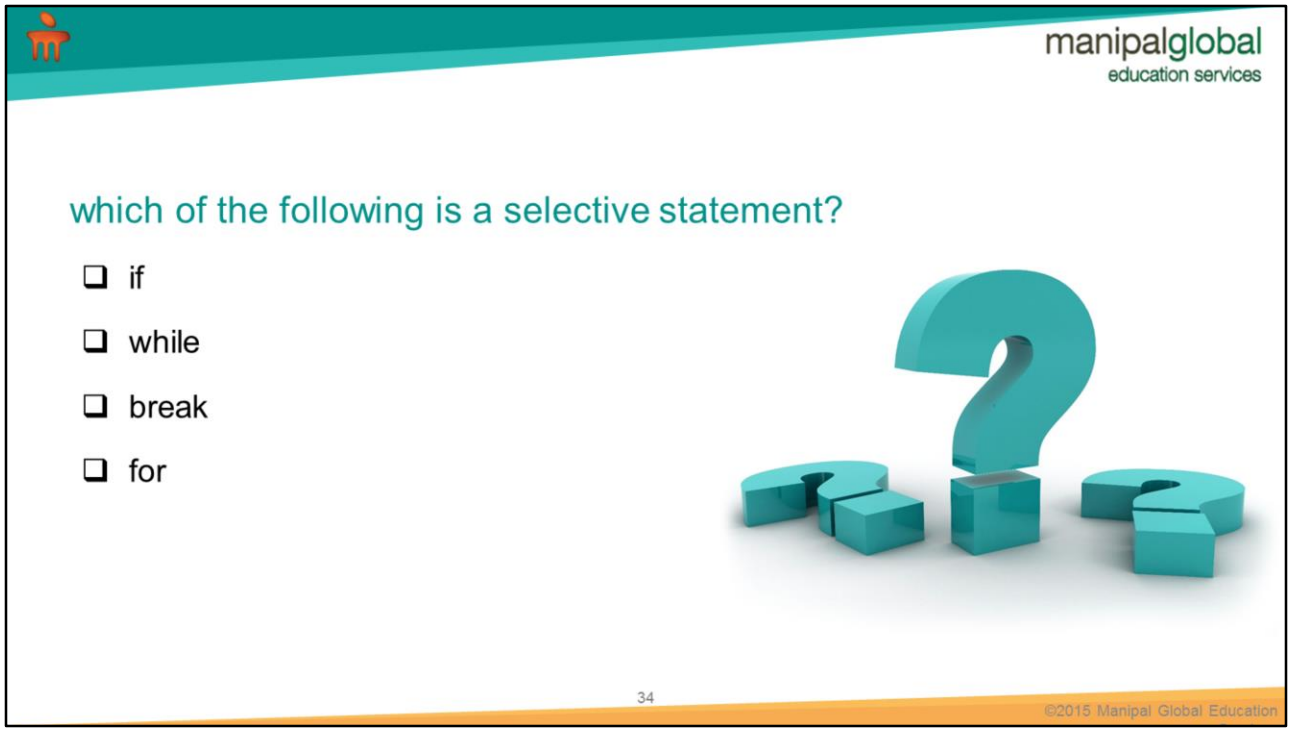
- expression:
  - Must be an array or Collection which has to be looped through
  - Could be a method call that returns an array or collection
- declaration:
  - Variable, of a type compatible with the elements of the array or collection

Refer the below class for programming constructs demonstration

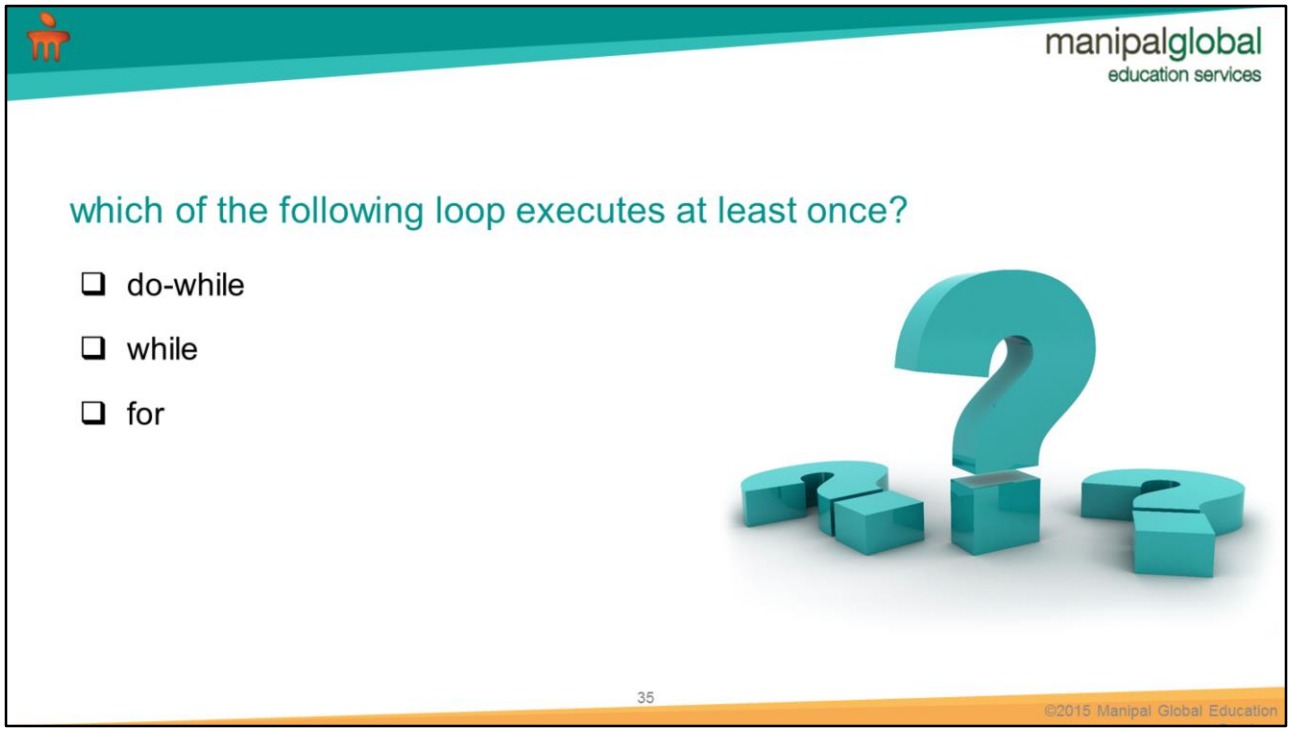
- ProgrammingConstructsDemo



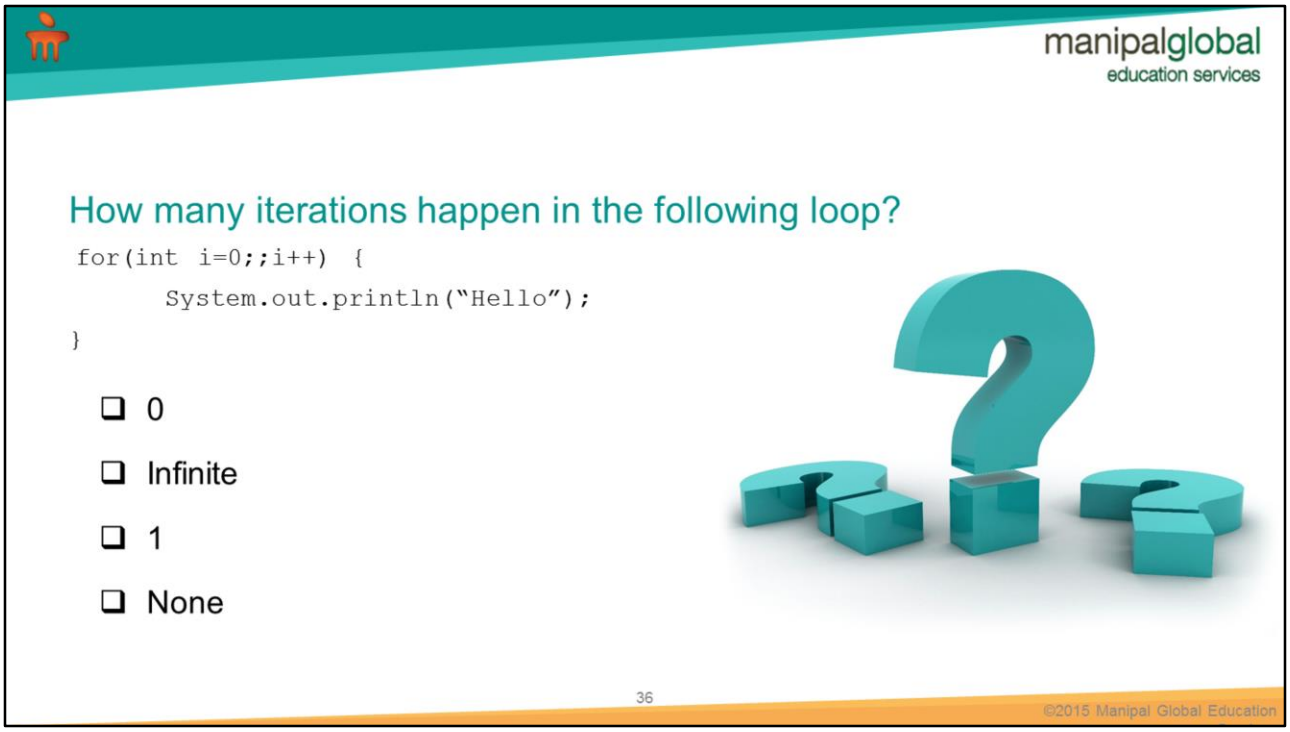
Answer: a default case


manipalglobal
education services
which of the following is a selective statement?
if
while
break
for
34
©2015 Manipal Global Education

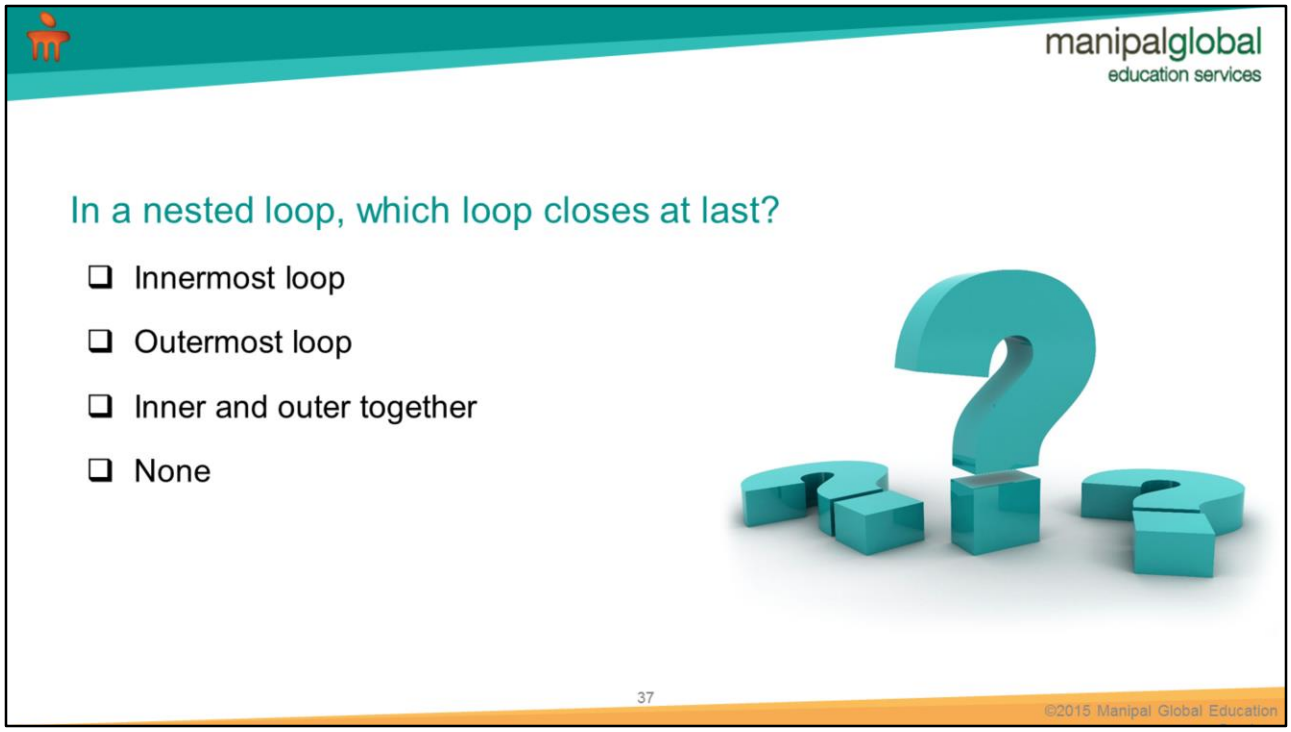
Answer: if



Answer: do while



Answer: Infinite



Answer: outermost loop

References

- Refer following demo videos on EduNxt
  - M3L3L2_Demonstration_of_if-else_construct – Demo
  - M3L3L4_Demonstration_of_switch-case_construct – Demo
  - M3L4L2_Demonstration_of_do-while_loop – Demo
  - M3L4L4_Demonstration_of_for_loop – Demo
  - M3L4L6_Demonstration_of_while_loop – Demo
  - M3L5L2_Demonstration_of_branching_statements – Demo
  - M3L6L6_Demonstration_of_enhanced_for_loop - Demo

SUMMARY

Programming Constructs

SUMMARY

In this lesson, you've learned to:

- Illustrate selectional constructs
- Use iterational constructs
- Understand branching statements