



# SDE Readiness Training

Empowering Tomorrow's Innovators



PREPARATION

IS  
THE  
KEY  
TO  
SUCCESS

# Module I

*Java Software Development:  
Effective Problem Solving*

# Control Flow Statements

**Learning Level : Basics**

**DATE : 14.02.2025**



# Contents

01

Introduction

02

Branching  
Conditional

03

Looping

04

Nested Loop

05

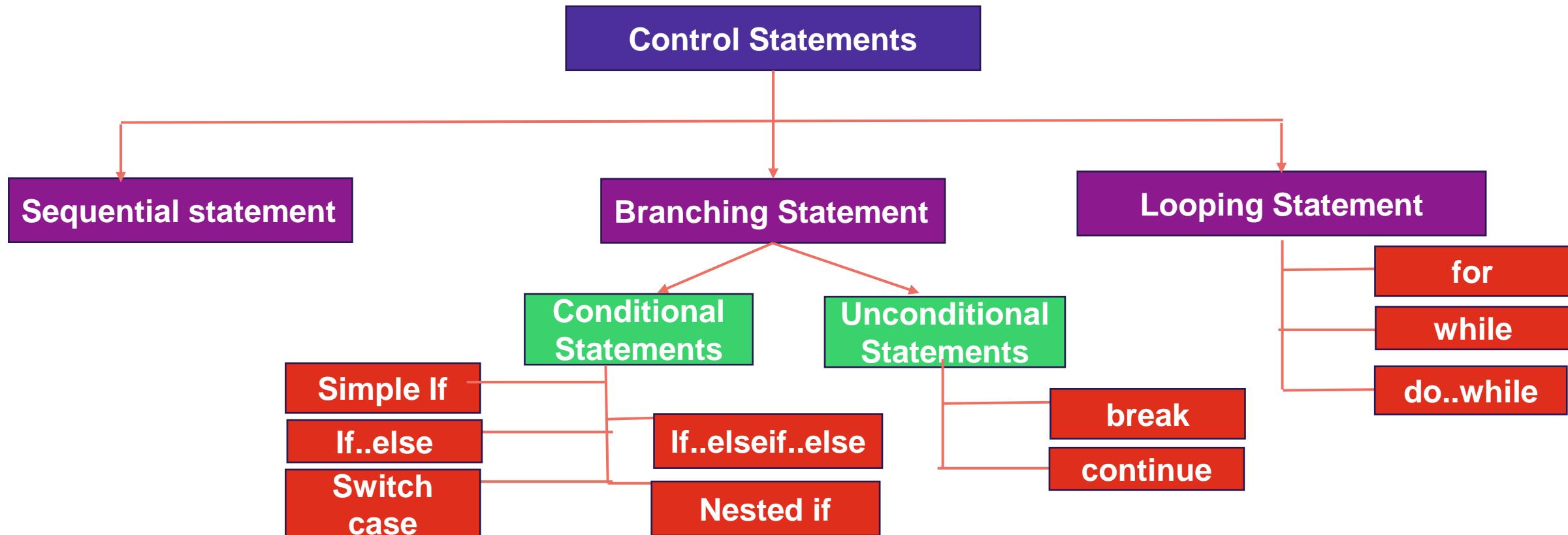
Branching -  
Unconditional

06

Quiz

## Introduction

- Control flow is the **order in which individual statements or instructions** of a program are executed or evaluated and **Control flow statements** is categorized as follows:



## Introduction

- **Sequential statements** describe a **sequence of actions** that a **program carries out one after another, unconditionally.**
- Execute a **list of statements in order.**

### Example:

```
import java.util.Scanner;
public class CircleArea {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        double area = Math.PI * radius * radius;
        System.out.println("The area of the circle with radius " + radius + " is: " + area);
        scanner.close();
    }
}
```

## Introduction

- **Branching (Conditional) /Decision Making** – To make **decisions** based on a given condition. If the condition evaluates to **True**, a **set of statements** is executed, **otherwise another set** of statements is executed. In Java, we have **five types** of decision making statements:
  - if
  - if...else
  - if-else-if
  - nested if
  - switch

## Decision-Making : simple if

- Simple if: If the condition is true, the body of the if statement is executed. If it is false, the body of the if statement is skipped.

- Syntax

if is a Java  
reserved word

The **condition** must be a boolean expression. It  
must evaluate to either **true or false**.

```
if ( condition ){  
    statement(s);  
}
```

If the **condition** is **true**, the **statement(s)** is/are executed.

If it is **false**, the **statement(s)** is/are skipped.

## Control Flow Statements

### Decision-Making : simple if

```
/**  
 * The IfControlStructure class implements an application that  
 * Illustrate the if decision Making control statement  
 */  
  
class IfControlStructure{  
    public static void main(String[] args){  
        boolean isMoving=true;  
        int currentSpeed=10;  
        if(isMoving){  
            System.out.println(currentSpeed);  
        }  
    }  
}
```

**Output:**

10

## Decision-Making : simple if

```
/*
 *The SimpleIf class implements an application that checks the seat availability status for movie ticket booking using
 *simple if decision Making control statement
 */
import java.util.Scanner;
public class SimpleIf {
    public static void main(String[] args){
        boolean seatAvailable = true;                      //Seat Available Status
        Scanner input = new Scanner(System.in);            //Scanner class object creation
        System.out.println("Enter the Seat Number : ");
        String SeatNumber = input.next();                  //get Seat Number from User
    }
}
```

## Control Flow Statements

### Decision-Making : simple if

```
if(seatAvailable){                                //Check the availability of seat
    System.out.println("Your have booked the seat number : "+SeatNumber);
}
input.close();
}
```

#### Output:

Enter the Seat Number : A32

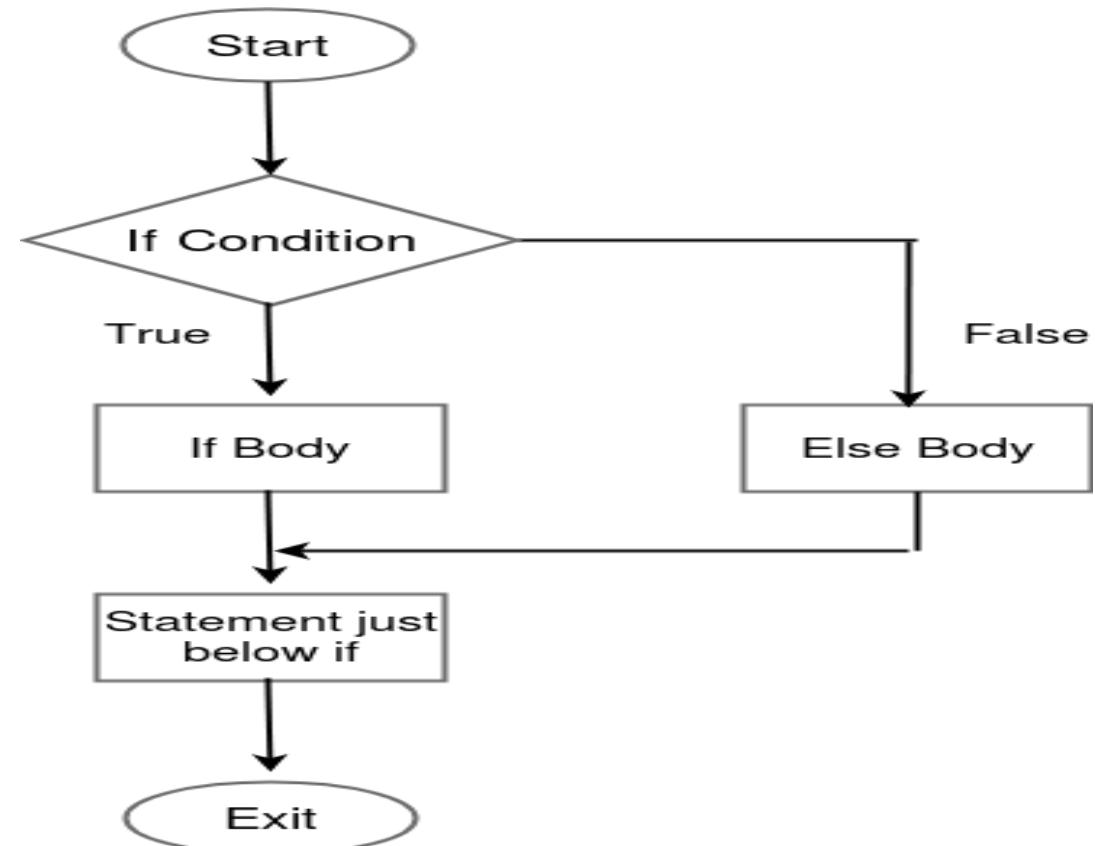
Your have booked the seat number : A32

## Control Flow Statements

## Decision-Making : simple if - else

- **if –else:** If the *condition* is true, the body of the if *statement* is executed. If it is false, the body of the false *statement* is executed.
- Syntax

```
if ( condition ){  
    If Body;  
}  
else{  
    else Body;  
}
```



## Control Flow Statements

# Decision-Making : simple if - else

```
/** * The IfElseControlStructure class implements an application that Illustrate
the if ..else decision Making control statement */
class IfElseControlStructure{
    public static void main(String[] args){
        boolean isMoving=true;
        int currentSpeed=10;
        if(isMoving){
            currentSpeed--;
            System.out.println("The bicycle speed got reduced!");
        }
        else{
            System.out.println("The bicycle has already stopped!");
        }
    }
}
```

## Output:

The bicycle speed got reduced!

## Decision-Making : simple if - else

```
/** * The SimpleIfElse class implements an application that checks the seat availability status for movie ticket booking using the if ..else decision-Making control statement */

public class SimpleIfElse {

    public static void main(String[] args){

        boolean seatAvailable = false;          //Seat Available Status
        Scanner input = new Scanner(System.in); //Scanner class object creation
        System.out.println("Enter the Seat Number : ");
        String SeatNumber = input.next();        // get Seat Number from User
        if(seatAvailable){                    //Check the availability of seat
            System.out.println("Your have booked the seat number : "+SeatNumber);
        }
    }
}
```

## Control Flow Statements

### Decision-Making : simple if - else

```
else {                                //if seat not available then display
    System.out.println("Seat Numer "+SeatNumber+" is already booked");
}
input.close();
}
```

#### Output:

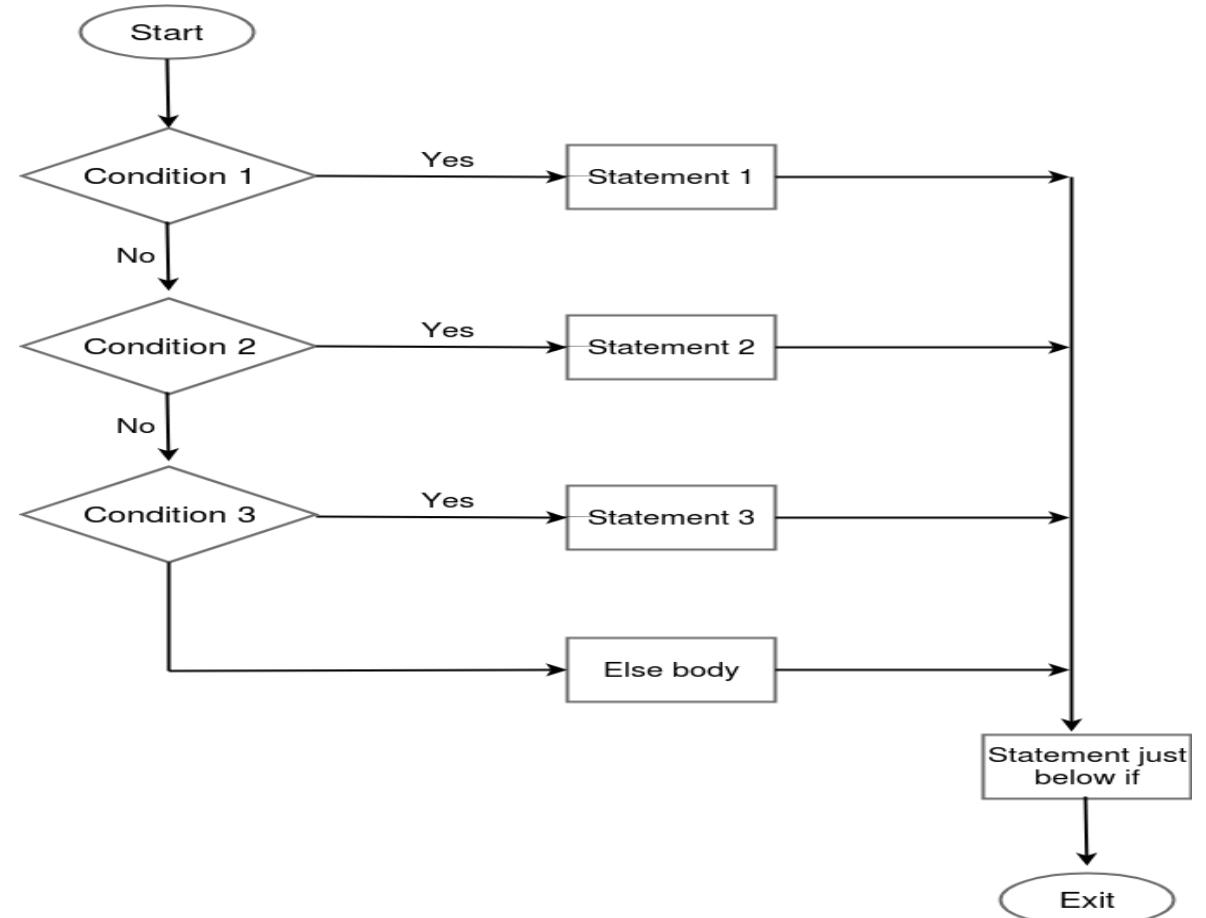
Enter the Seat Number : A22  
Seat Numer A22 is already booked

## Control Flow Statements

# Decision-Making : simple if – else – if

- **if-else-if:** It is also called **else-if ladder**. Here execute any one block of statements among many blocks.
- **Syntax**

```
if ( condition 1){
    statement 1;
}else if ( condition 2 ){
    statement 2;
}else if (condition 3){
    statement 3;
} else {
    else Body
}
```



## Control Flow Statements

### Decision-Making : simple if – else – if

```
/**  
 * The IfElseIFControlStructure class implements an application that  
 * Illustrate the if ..elseif decision Making control statement */  
  
class IfElseIFControlStructure{  
    public static void main(String[] args){  
        int colorValue=2;  
        if(colorValue==1)  
            System.out.println("Color Blue!");  
        else if(colorValue==2)  
            System.out.println("Color Red!");  
        else  
            System.out.println("Color Green!");  
    }  
}
```

**Output:**  
Color Red!

## Decision-Making : simple if – else – if

```
/**  
 * The IfElseif class implements an application that display the cost of the specific seat category in movie ticket  
 booking using if ..elseif decision Making control statement for fixing the cost based on Movie Type */  
import java.util.Scanner;  
  
public class IfElseif {  
    public static void main(String[] args){  
        String seattype;  
        System.out.println("Type of seats Available\nREGULAR\nPREMIUM\nEXECUTIVE\nVIP\n choose any one of the  
option : ");  
        Scanner input = new Scanner(System.in); //Scanner class object creation  
        seattype = input.next(); // get Seat Number from User  
        if (seattype.equals("REGULAR")) { //Display detail for REGULAR type  
            System.out.println("You have selected Executive Seat and cost is Rs.80");  
        }  
    }  
}
```

## Decision-Making : simple if – else – if

```
else if (seattype.equals("PREMIUM")) {           //Display detail for PREMIUM type
    System.out.println("You have selected Premium Seat and cost is Rs.100");
}

else if (seattype.equals("EXECUTIVE")) {           //Display detail for EXECUTIVE type
    System.out.println("You have selected Regular Seat and cost is Rs.120");
} else if (seattype.equals("VIP")) {                //Display detail for VIP type
    System.out.println("You have selected VIP Seat and cost is Rs.150");
} else {
    System.out.println("You have not selected any type");
}

input.close();
}
```

## Control Flow Statements

# Decision-Making : simple if – else – if

### Output:

Type of seats Available

REGULAR

PREMIUM

EXECUTIVE

VIP

choose any one of the option : PREMIUM

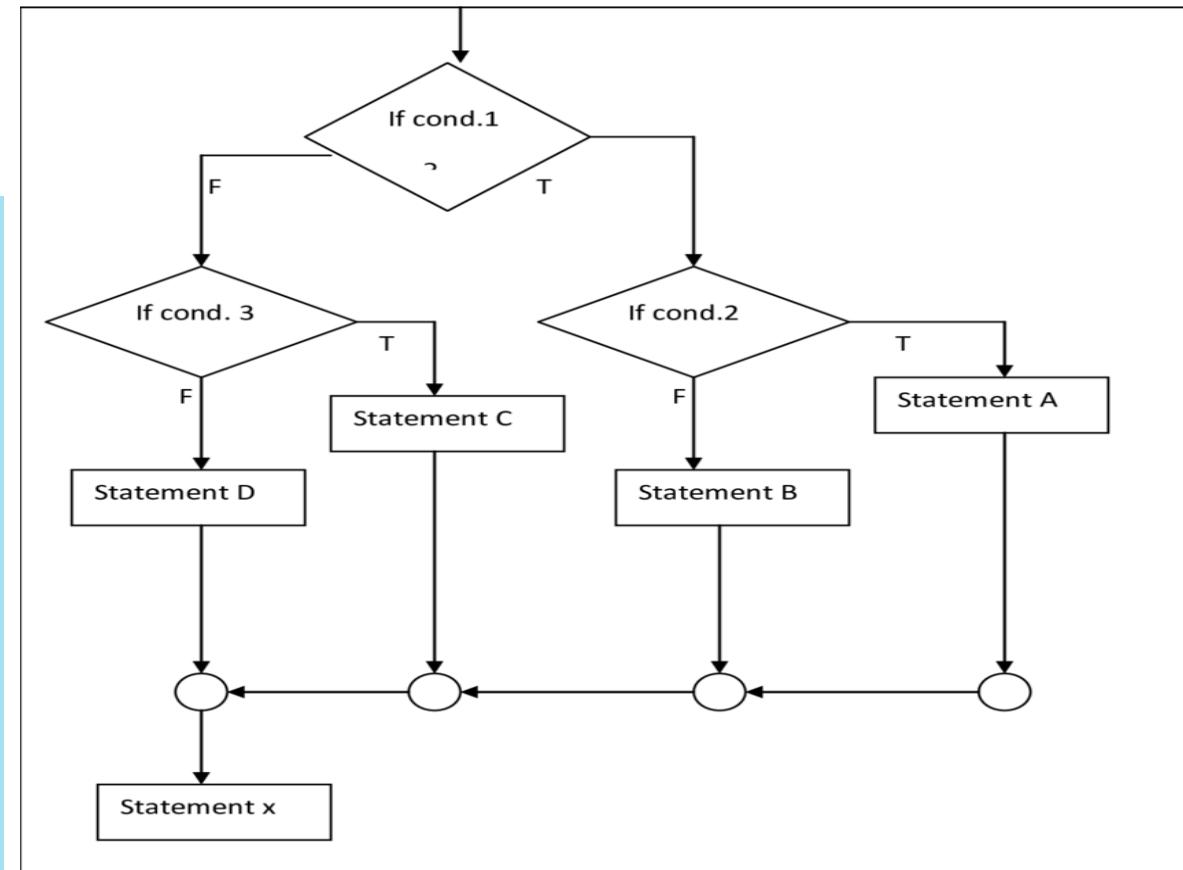
You have selected Premium Seat and cost Rs.100

## Control Flow Statements

# Decision-Making : Nested if

- **Nested if:** Use more than if statement inside another if statement. The outer if statement condition true means inside if statements execute.
- **Syntax**

```
if ( condition 1){
    if ( condition 2 ){
        Nested if Body
    }else{
        Nested else Body
    }
}else
    else Body
}
```



## Decision-Making : Nested if

```
/* * The NestedIfControlStructure class implements an application that
 * Illustrate the nestedif decision Making control statement */
class NestedIFControlStructure{
    public static void main(String[] args){
        int age=15;
        int weight=50;
        if(age>18) {
            if(weight>50)
                System.out.println("You are eligible to denote blood");
            else
                System.out.println("Not eligible because you are under weight");
        } else
            System.out.println("Not eligible because you are under age");
    }
}
```

### Output:

Not eligible because  
you are underage

## Decision-Making : Nested if

```
/* * The Booking class implements an application that validates the login and check for the seat availability
 * using nestedif decision Making control statement */

import java.util.*;

public class NestedIf {
    public static void main(String[] args){
        String username = "Sarvesh",password = "sarvesh@123",usernameentered,passwordentered;
        boolean seatAvailable = true;
        String seatNumber;
        Scanner input = new Scanner(System.in);          //Scanner class object creation
        System.out.println("Enter the Username : ");
        username = input.next();                         //getting the username
        System.out.println("Enter the Password : ");
        password = input.next();                        //getting the username
```

## Decision-Making : Nested if

```
if (usernameentered.equals(username) && password.equals(password)) {      //validate the user login
    System.out.println("You have logged in and you can book a ticket now");
    System.out.println("Enter the Seat Number : ");
    seatNumber = input.next();                      // get the seat number from the user
    if (seatAvailable) {                           // check for seat availability status
        System.out.println("Seat Number "+seatNumber+" you have chosen is available");
    } else{
        System.out.println("Your expected Seat Number "+seatNumber+" is not available");
    }
} else{
    System.out.println("You have to login for booking the ticket");
}     input.close();
}
```

## Decision-Making : Nested if

**Output:**

Enter the Username : Sarvesh

Enter the Password : sarvesh@123

You have logged in and you can book a ticket now

Enter the Seat Number : A10

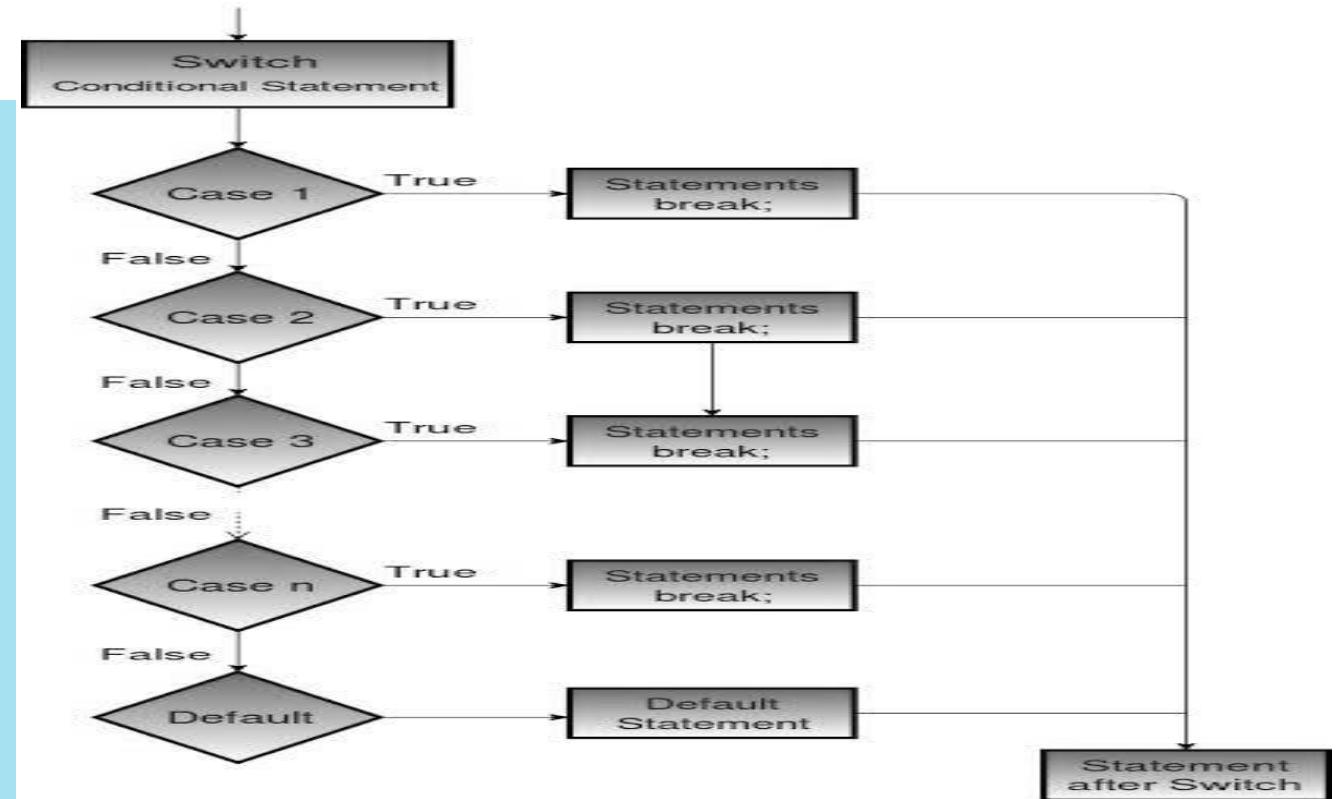
Seat Number A10 you have chosen is available

## Control Flow Statements

# Decision-Making : switch - case

- **switch-case:** Select one of many possible statements to execute. It gives alternate for long if..else..if ladders which improves code readable.
- **Syntax**

```
switch ( expression ) {
    case value1 :
        statement-list1;
        break;
    case value2 :
        statement-list 2;
        break;
    default:
        statement-list 3;
        break;
}
```



## Decision-Making : switch - case

### Note:

- In switch, Case value must be in **expression type** only and case values must be **unique**.
- Expression must be of **byte, short, int, long, enums and string**.
- Each case **break statement is optional**. It helps **terminate from switch expression**. If a break statement is not found, it executes the next case.
- The case value can have a **default label** which is **optional**.

## Control Flow Statements

# Decision-Making : switch - case

```
/** The SwitchControlStructure class implements an application that
 * Illustrate switch decision Making control statement */
class SwitchControlStructure{

    public static void main(String[] args){

        int letter='A';

        switch(letter){

            case 'a':
                System.out.println("Lowercase Letter");
                break;

            case 'A':
                System.out.println("Uppercase Letter");
                break;
        }
    }
}
```

## Control Flow Statements

### Decision-Making : switch - case

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

#### Output:

Uppercase Letter

## Decision-Making : switch - case

```
/** The SwitchCase class implements an application that demonstrate the movie searching by different types of languages
 * using switch decision Making control statement Searching the Movie detail by Title, Language, ReleaseDate, Genre*/
import java.util.Scanner;
public class SwitchCase {
    public static void main(String[] args){
        System.out.println("Enter the type to be search \n1. Search by Title \n2. Search by Language \n3. Search by Release
Date \n4. Search by Genre \nEnter the Choice (1/2/3/4)");
        Scanner input = new Scanner(System.in); //Scanner class object creation
        int choice = input.nextInt();           //getting the choice from user
        switch(choice){
            case 1:
                System.out.println("Your searching choice is Movies by Title");
                break;
            case 2:
```

## Decision-Making : switch - case

```
System.out.println("Your searching choice is Movies by Language");
break;
case 3:
    System.out.println("Your searching choice Movies by Release Date");
    break;
case 4:
    System.out.println("Your searching choice Movies by Genre");
    break;
default:
    System.out.print("Your choice is wrong. Please enter the correct choice ");
}
input.close();
}
```

## Decision-Making : switch - case

**Output:**

Enter the type to be search

1. Search by Title
2. Search by Language
3. Search by Release Date
4. Search by Genre

2

Your searching choice is Movies by Languagec

## Looping - Introduction

- **Loop/ iterative** statements are used for **executing a block of statements repeatedly** until a particular condition is satisfied.
- **In Java, we have following loop statements:**
  - while
  - do...while
  - for
  - for...each
  - Nested loops (for, while, do...while)

# Looping - Introduction

## Four Looping Elements

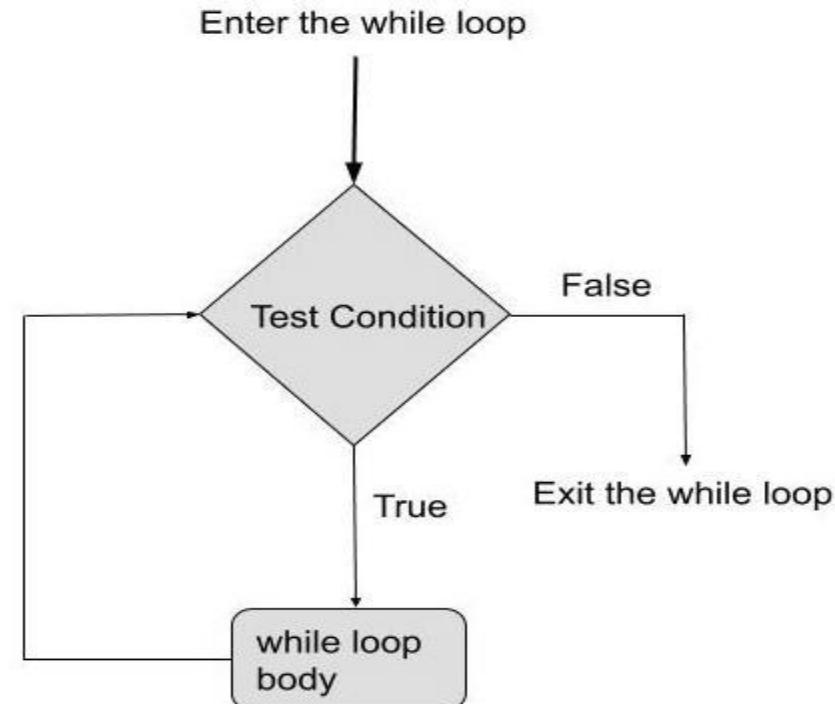
- **Initialization:** To set **initial value** to the iteration variable at the very start of the loop.
- **Condition:** Test condition is evaluate and give **boolean** ( 0 or 1) value as a result.
- **Loop-body:** If the test condition is true, the **body of the loop runs once**.
- **Updation: Increment/decrement** statement executes just after executing the body, and then the program goes back to the test condition.

## Control Flow Statements

# Looping - while

- **while:** It is used to repeat a **specific block of code**. It is **preferable** while we do **not know the exact number of iterations** of loop beforehand.
- **Syntax**

```
while (condition){\n    statement(s)\n}
```



## Control Flow Statements

### Looping - while

```
/**  
 * The WhileStructure class implements an application that  
 * Illustrate While Looping control statement  
 */  
  
class WhileStructure{  
    public static void main(String[] args){  
        int counter = 1;  
        while (counter < 11)  
        {  
            System.out.println("Count is: " + counter);  
            counter++;  
        }  
    }  
}
```

## Looping - while

**Output:**

Count is: 1

Count is: 2

Count is: 3

Count is: 4

Count is: 5

Count is: 6

Count is: 7

Count is: 8

Count is: 9

Count is: 10

## Looping - while

```
/**  
 * The ShowSeat class implements an application that display the current seat availability  
 * using While Looping control statement  
 */  
public class ShowSeat {  
    public static void main (String args[]){  
        int MaxSeatCount = 10, seatCount = 0;  
        while(seatCount < MaxSeatCount){  
            System.out.println("Current Seat Availability : "+(MaxSeatCount-seatCount));  
            seatCount++;  
        }  
        System.out.println("Seats are Filled");  
    }  
}
```

## Looping - while

**Output:**

Current Seat Availability : 10

Current Seat Availability : 9

Current Seat Availability : 8

Current Seat Availability : 7

Current Seat Availability : 6

Current Seat Availability : 5

Current Seat Availability : 4

Current Seat Availability : 3

Current Seat Availability : 2

Current Seat Availability : 1

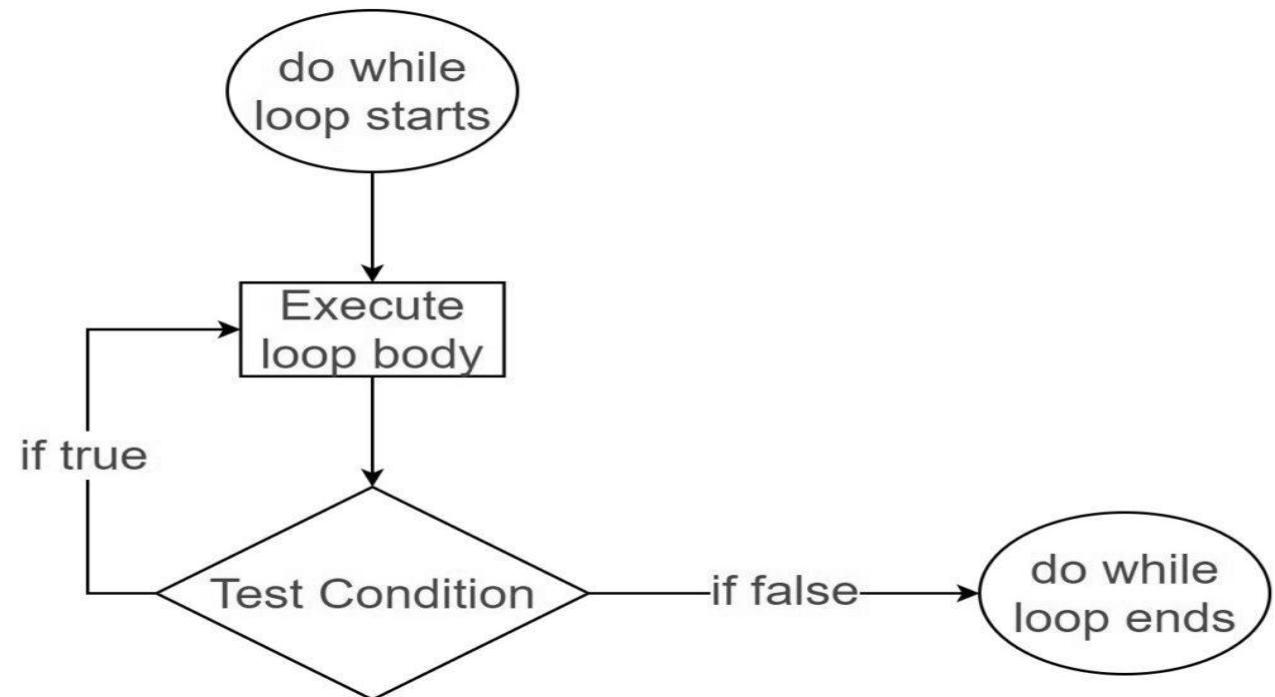
Seats are Filled

## Control Flow Statements

# Looping - do - while

- **do-while:** Executes the statement first and then checks for the condition. It is also called an **exit-controlled loop.**
- **Syntax**

```
do {  
    statement(s)  
} while (condition);
```



## Looping - do - while

```
/**  
 * The DoWhileStructure class implements an application that checks the seat availability  
 * status for movie ticket booking and illustrate do..while Looping control statement  
 */  
class DoWhileStructure{  
    public static void main(String[] args){  
        int counter = 1;  
        do {  
            System.out.println("Count is: " + counter);  
            counter++;  
        } while (counter < 11)  
    }  
}
```

## Looping - do - while

**Output:**

Count is: 1

Count is: 2

Count is: 3

Count is: 4

Count is: 5

Count is: 6

Count is: 7

Count is: 8

Count is: 9

Count is: 10

## Control Flow Statements

# Looping - do - while

```
/**  
 * The ShowSeat class implements an application that checks the seat  
 * availability status for movie ticket booking using do.. while Looping control statement */  
public class ShowSeat {  
    public static void main (String args[]){  
        int MaxSeatCount = 5, seatCount = 0;  
        do{  
            System.out.println("Current Seat Availability : "+(MaxSeatCount-seatCount));  
            seatCount++;  
        } while(seatCount < MaxSeatCount);  
        System.out.println("Seats are Filled");  
    }  
}
```

## Control Flow Statements

### Looping - do - while

#### Output:

Current Seat Availability : 5

Current Seat Availability : 4

Current Seat Availability : 3

Current Seat Availability : 2

Current Seat Availability : 1

Seats are Filled

## Control Flow Statements

### Looping - for

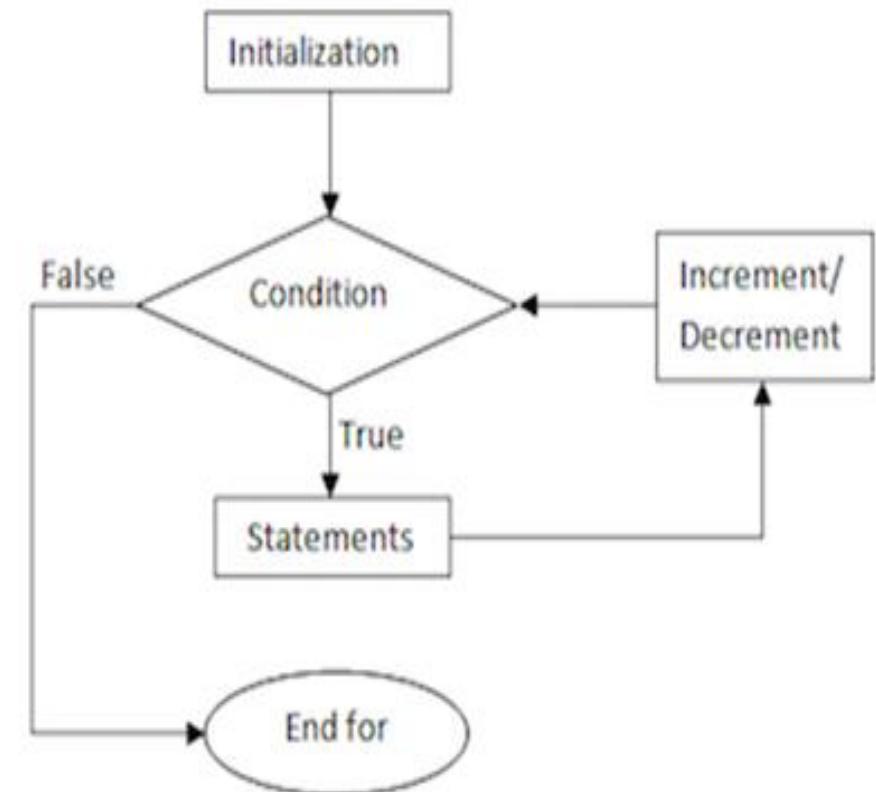
- **for:** When you know **exactly how many times** you want to loop through a block of code, use the for loop instead of a while loop
- **Syntax**

```
for (initialization; condition ; updation) {  
    statement(s)  
}
```

**Initialization** : Initializes the loop; it's executed once.

**Condition** : Evaluates till the condition becomes false.

**Updation** : Increment / Decrement the value. Executed (every time) after the code block is executed.



## Control Flow Statements

### Looping - for

```
/**  
 * The ForStructure class implements an application that  
 * Illustrate ForLooping control statement  
 */  
  
class ForStructure{  
  
    public static void main(String[] args) {  
  
        for(int count = 1; count<10; count++) {  
  
            System.out.println("Count is: " + count);  
  
        }  
  
    }  
}
```

## Looping - for

**Output:**

Count is: 1

Count is: 2

Count is: 3

Count is: 4

Count is: 5

Count is: 6

Count is: 7

Count is: 8

Count is: 9

## Looping - for

```
/**  
 * The ShowSeat class implements an application that checks the seat availability status for  
 * movie ticket booking using For Looping control statement  
 */  
class ShowSeat{  
    public static void main (String args[]){  
        int MaxSeatCount = 5, seatCount = 0;  
        for(seatCount=0;seatCount < MaxSeatCount;seatCount++){  
            System.out.println("Current Seat Availability : "+(MaxSeatCount-seatCount));  
        }  
        System.out.println("Seats are Filled");  
    }  
}
```

## Control Flow Statements

### Looping - for

#### Output:

Current Seat Availability : 5

Current Seat Availability : 4

Current Seat Availability : 3

Current Seat Availability : 2

Current Seat Availability : 1

Seats are Filled

## Looping - for - each

- **for-each:** It's **commonly used** to iterate over an **array or a Collections class**. It is also called **enhanced for loop**.
- **Syntax:**

```
for (type var : arrayName) {  
    statements;  
}
```
- It is used **to iterate over the elements** of a collection **without** knowing the **index** of each element.
- It is **immutable** which means the values which are **retrieved during the execution** of the loop are **read only**

## Looping - for - each

```
/**  
 * The ForEachApp class implements an application that  
 * illustrate foreach looping Structure  
 */  
class ForEachApp{  
    public static void main (String args[]){  
        int[] marks = { 125, 132, 95, 116, 110 };  
        int maxSoFar = marks[0];  
        //for each loop
```

## Control Flow Statements

### Looping - for - each

```
for (int indexValue : marks){  
    if (indexValue > maxSoFar){  
        maxSoFar = indexValue;  
    }  
}  
System.out.println("The highest score is " +maxSoFar);  
}  
}
```

#### Output:

The highest score is 132

## Looping - for - each

```
/**  
 * The ForEach class implements an application that list the movie based on their Genre  
 * using foreach looping Structure  
 */  
  
import java.util.Scanner;  
  
public class ForEach {  
    public static void main(String[] args) {  
        String MovieName[] = {"AAA","BBB","CCC","DDD"};  
        String MovieGenre[] = {"ACTION","COMEDY","THRILLER","ACTION"};  
        Scanner input = new Scanner(System.in);      //Scanner class object creation  
        System.out.println("Enter the Genre to be searched : ");  
        String Genre = input.next();  
        int counter = 0;
```

## Control Flow Statements

### Looping - for - each

```
System.out.println(Genre + " Movies are");
for (String M : MovieGenre ) {
    if(M.equals(Genre)) {
        System.out.println(MovieName[counter]);
    }
    counter++;
}
input.close();
}
```

#### Output:

Enter the Genre to be searched :  
ACTION  
ACTION Movies are  
AAA  
DDD

## Nested Loop

- **Nested Loop:** A loop inside another loop is called a nested loop. The number of loops depend on the requirement of a problem. It contains outer loop and inner loop. For each iteration of outer loop the inner loop executes completely.
- **Syntax**

```
while(condition) {  
    statement(s)  
    while(condition) {  
        statement(s)  
        .....  
    }  
    .....  
}
```

```
for (initialization; condition; updation) {  
    .....  
    for (initialization; condition; updation) {  
        statement(s)  
        .....  
    }  
    .....  
}
```

```
do {  
    .....  
    statement(s)  
} do {  
    .....  
    statement(s)  
    .....  
} while(condition);  
.....  
} while(condition);
```

## Nested Loop

```
/**  
 * The NestedWhileApp class implements an application that  
 * illustrate nested while loop */  
class NestedWhileAPP{  
    public static void main (String args[]){  
        int outerLoop=1,innerLoop=1;  
        whlie(outerLoop<=5){  
            while(innerLoop<=5){  
                System.out.print("*");  
                innerLoop++;  
            }  
        }  
    }  
}
```

## Control Flow Statements

# Nested Loop

```
System.out.println(" ");
    outerLoop++;
    innerLoop=1;
}
}
}
```

## Output:

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

## Nested Loop

```
/**  
 * The NestedWhile class implements an application that demonstrate the seat availability while the seats are getting  
 booked in multiple screens using nested while loop */  
  
class NestedWhile{  
    public static void main (String args[]){  
        int MaxSeatCount = 5, TotalScreenCount = 2, seatCount = 0, screenCount = 0;  
        while(screenCount < TotalScreenCount){  
            seatCount = 0;  
            System.out.println("Screen "+(screenCount+1)+" Availability details");  
            while(seatCount < MaxSeatCount){  
                System.out.println("Current Seat Availability : "+(MaxSeatCount-seatCount));  
                seatCount++;  
            }  
        }  
    }  
}
```

## Nested Loop

```
        System.out.println("Seats Filled in Screen "+(screenCount+1));  
        screenCount++;  
    }  
}  
}
```

### Output:

Screen 1 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 1  
Screen 2 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 2

## Control Flow Statements

# Nested Loop

```
/**  
 * The Nested DoWhileApp class implements an application that illustrate nested do...while loop */  
class NestedDoWhileApp {  
    public static void main (String args[]){  
        int outerLabel= 1;  
        do {  
            int innerLabel = 1;  
            do{  
                System.out.print(outerLabel);  
                innerLabel++;  
  
            } while (innerLabel <= 3);  
            outerLabel++;  
        } while (outerLabel <= 3);  
    }  
}
```

## Output:

111222333

## Nested Loop

```
/**  
 * The NestedDoWhile class implements an application that demonstrate the seat availability while the seats are  
 getting booked in multiple screens using nested do...while loop */  
  
class NestedDoWhile{  
    public static void main (String args[]){  
        int MaxSeatCount = 10, TotalScreenCount = 2, seatCount = 0, screenCount = 0;  
        do{  
            System.out.println("Screen "+(screenCount+1)+" Availability details");  
            seatCount = 0;  
            do{  
                System.out.println("Current Seats Availability : "+(MaxSeatCount-seatCount));  
                seatCount++;  
            } while(seatCount < MaxSeatCount);  
            screenCount++;  
        } while(screenCount < TotalScreenCount);  
    }  
}
```

## Nested Loop

```
        System.out.println("Seats Filled in Screen "+(screenCount+1));
        screenCount++;
    } while(screenCount < TotalScreenCount);
}
}
```

### Output:

Screen 1 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 1  
Screen 2 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 2

## Nested Loop

```
/**  
 * The Nested ForApp class implements an application that  
 * illustrate nested for looping structure  
 */  
  
class NestedForApp  
{  
    public static void main (String args[])  
    {  
        int rows = 5;  
        // outer loop
```

## Control Flow Statements

### Nested 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("");  
}
```

#### Output:

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

## Nested Loop

```
/**  
 * The NestedFor class implements an application that demonstrate the seat availability while the seats are getting  
 booked in multiple screens and illustrate nested for looping structure  
 */  
  
class NestedFor{  
  
    public static void main (String args[]){  
  
        int MaxSeatCount = 5, TotalScreenCount = 2, seatCount = 0, screenCount = 0;  
  
        System.out.println("Screen "+(screenCount+1)+" Availability details");  
  
        for(screenCount=0;screenCount < TotalScreenCount;screenCount++){  
  
            for(seatCount=0;seatCount < MaxSeatCount;seatCount++){  
  
                System.out.println("Current Seat Availability "+(MaxSeatCount-seatCount));  
            }  
        }  
    }  
}
```

## Nested Loop

```
    }  
  
    System.out.println("Seats Filled in Screen "+(screenCount+1));  
}  
}  
}
```

**Output:**

Screen 1 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 1  
Screen 2 Availability details  
Current Seat Availability : 5  
Current Seat Availability : 4  
Current Seat Availability : 3  
Current Seat Availability : 2  
Current Seat Availability : 1  
Seats Filled in Screen 2

## Branching - Unconditional

- **Branching (Unconditional)** - To transfers the control from **one part of the program** to another part **without any condition**.
- In Java, we have the following **unconditional statements**:
  - break
  - labelled break
  - continue
  - labelled continue

## Branching - Unconditional

- **break:** When a **break** statement is encountered inside a loop, the **loop** is immediately **terminated** and the program control resumes at the **next statement** following the **loop**.
- The Java break is used to **break loop or switch statement**. It **breaks the current flow of the program** at specified condition. In case of **inner loop**, it **breaks only inner loop**.
- Break statement use **all types of loops** such as for loop, while loop and do-while loop.
- **Syntax**

```
break;
```

## Branching - Unconditional

```
/**  
 * The BreakApp class implements an application that  
 * illustrate Break branching statement  
 */  
  
class BreakApp{  
    public static void main (String args[]){  
        for(int count = 1;count<10;count++) {  
            if(count ==5)  
                break; //exit from the current loop  
            System.out.println("Count is: " + count);  
        }  
    }  
}
```

**Output:**

Count is: 1

Count is: 2

Count is: 3

Count is: 4

## Branching - Unconditional

```
/**  
 * The UnconditionalBreak class implements an application that demonstrate the seat availability while the seats are  
 * getting booked assuming that the VIP seats are already reserved using Break branching statement  
 */  
class UnconditionalBreak {  
    public static void main (String args[]){  
        int premiumSeat = 5, vipSeat = 5, seatBooked = 0;  
        int totalSeat = premiumSeat + vipSeat;  
        for(seatBooked = 0;seatBooked < totalSeat;seatBooked++) {  
            if(seatBooked > premiumSeat) {  
                break;  
            }  
            System.out.println("Seat Booked");  
        }  
    }  
}
```

## Branching - Unconditional

```
        System.out.println("All PREMIUM Seats Booked ");

        System.out.println("All VIP Seats 1 to 5 are Reserved ");
        break; //exit from the current loop

    }

    else {
        System.out.println("PREMIUM Seat Availability : "+(premiumSeat - seatBooked));
    }
}

}
```

## Branching - Unconditional

**Output:**

PREMIUM Seat Availability : 5

PREMIUM Seat Availability : 4

PREMIUM Seat Availability : 3

PREMIUM Seat Availability : 2

PREMIUM Seat Availability : 1

All PREMIUM Seats Booked

All VIP Seats 6 to 10 are Reserved

## Branching - Unconditional

- **continue:** When you need to **jump to the next iteration** of the loop immediately.
- It **continues the current flow** of the program and **skips the remaining code** at the specified condition.

In case of an inner loop, it continues the inner loop only.

- Continue statement use **all types of loops** such as for loop, while loop and do-while loop.
- Syntax:

```
continue;
```

## Branching - Unconditional

```
/**  
 * The ContinueApp class implements an application that  
 * illustrate Continue branching statement  
 */  
  
class ContinueApp {  
    public static void main (String args[]) {  
        for(int count = 1;count<10;count++) {  
            if(count ==5)  
                continue;  
            System.out.println("Count is: " + count);  
        }  
    }  
}
```

**Output:**

Count is: 1  
Count is: 2  
Count is: 3  
Count is: 4  
Count is: 6  
Count is: 7  
Count is: 8  
Count is: 9

## Branching - Unconditional

```
/**  
 * The UnconditionalContinue class implements an application that demonstrate the seat availability while the seats are  
 getting booked assuming that the VIP seats are already reserved using Continue branching statement  
 */  
class UnconditionalContinue {  
    Public static void main (String args[]){  
        int executiveSeat = 5, premiumSeat = 5, vipSeat = 5, seatBooked = 0;  
        int totalSeat = regularSeat + executiveSeat + premiumSeat + vipSeat;  
        for(seatBooked = 0;seatBooked < totalSeat;seatBooked++) {  
            if(seatBooked < (vipSeat)){  
                System.out.println("All VIP Seats 1 to 5 are Reserved ");  
                continue;  
            }  
        }  
    }  
}
```

## Branching - Unconditional

```
if(seatBooked < vipSeat + premiumSeat) {  
    System.out.println("PREMIUM Seat No : "+(seatBooked+1));  
}  
  
if(seatBooked < (vipSeat + premiumSeat + executiveSeat)) {  
    System.out.println("EXECUTIVE Seat No : "+(seatBooked+1));  
}  
}
```

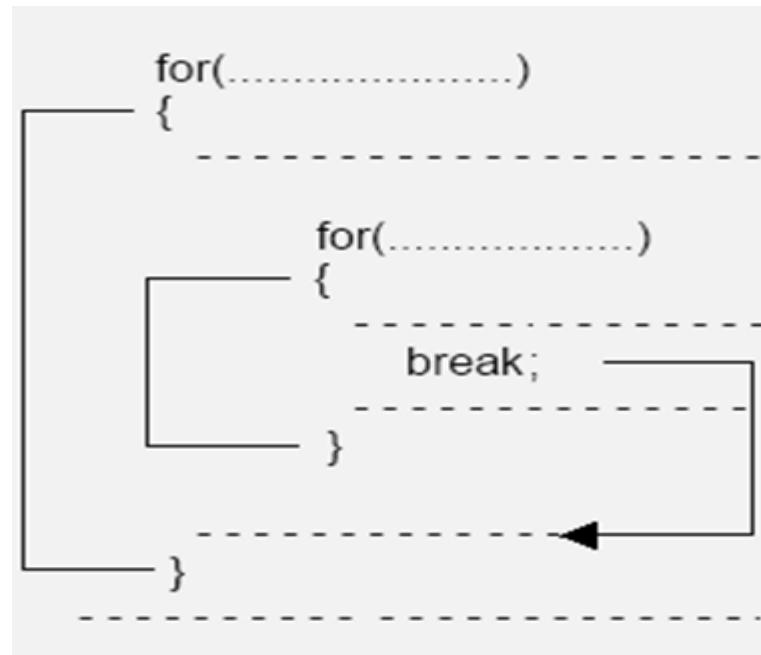
**Output:**

All VIP Seats 1 to 5 are Reserved  
PREMIUM Seat No : 6  
PREMIUM Seat No : 7  
PREMIUM Seat No : 8  
PREMIUM Seat No : 9  
PREMIUM Seat No : 10  
Executive Seat No : 11  
Executive Seat No : 12  
Executive Seat No : 13  
Executive Seat No : 14  
Executive Seat No : 15

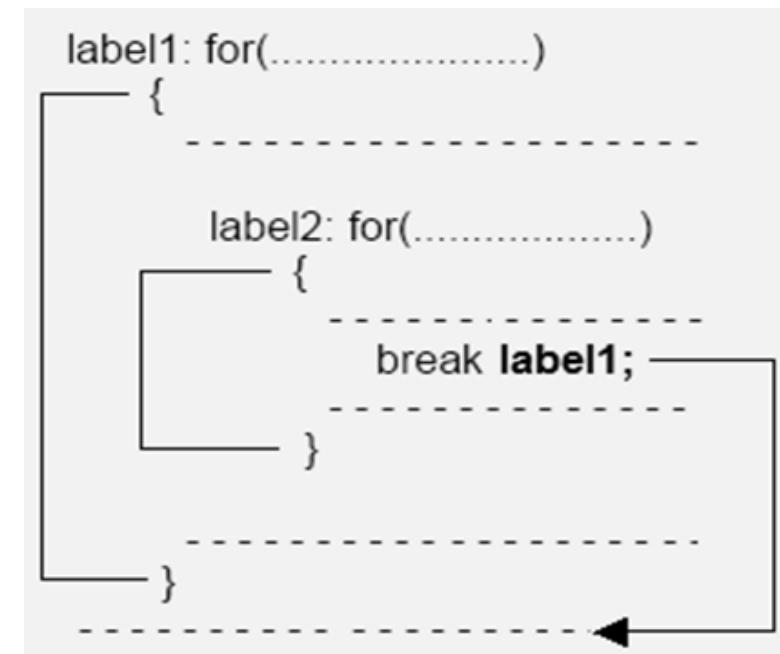
## Branching - Unconditional

- **Labeled Loop:** In java, use labels with break and continue. A Label is used to identify a block of code.  
In case of multiple loop involved use labeled loop to transfer any specific loop with help of labels.

### Loop without Label



### Loop with Label



## Branching - Unconditional

```
/** * The LabeledBreakApp class implements an application that * illustrate Labeled Break */
class LabeledBreakApp{
    public static void main(String args[]){
        first: // First label
        for (int i = 0; i < 3; i++) {
            second: // Second label
            for (int j = 0; j < 3; j++) {
                if (1==i && 1 == j) {
                    // Using break statement with label
                    break first;
                }
                System.out.println(i + " " + j);
            }
        }
    }
}
```

**Output:**

```
0 0
0 1
0 2
1 0
```

## Branching - Unconditional

```
/** * The LabeledBreakApp class implements an application that illustrate Labeled Break */
public class LabelledBreak {
    public static void main (String args[]){
        int MaxSeatCount = 10, TotalScreenCount = 2, seatCount = 0, screenCount = -1;
        Start:
        //System.out.println(screenCount);
        while(screenCount < TotalScreenCount){
            screenCount++;
            System.out.println("Screen "+(screenCount+1)+" Seat Booked detail");
            seatCount=0;
            while(seatCount < MaxSeatCount){
                if(seatCount >3 && screenCount == 1) {
                    System.out.println("Seat No 4 & 5 are Reserved");
                }
                seatCount++;
            }
        }
    }
}
```

## Branching - Unconditional

```
        break Start;  
    }  
    else  
        System.out.println("Seats No Booked : "+(seatCount+1));  
    seatCount++;  
}  
System.out.println("All Seats Filled in Screen "+(screenCount+1));  
}  
}
```

## Branching - Unconditional

**Output:**

Screen 1 Seat Booked detail

Seats No Booked : 1

Seats No Booked : 2

Seats No Booked : 3

Seats No Booked : 4

Seats No Booked : 5

All Seats Filled in Screen 1

Screen 2 Seat Booked detail

Seats No Booked : 1

Seats No Booked : 2

Seats No Booked : 3

Seat No 4 & 5 are Reserved

## Control Flow Statements

# Branching - Unconditional

```
/** The LabeledContinueApp class implements an application that illustrate LabeledContinue loop */
class LabeledContinueApp {
    public static void main(String args[]){
        first: // First label
        for (int i = 0 ; i < 3; i++) {
            second: // Second label
            for (int j = 0; j < 3; j++) {
                if (1 == i && 1 == j) {
                    // Using continue statement with label
                    continue second;
                }
                System.out.println(i + " " + j);
            }
        }
    }
}
```

**Output:**  
0 0  
0 1  
0 2  
1 0  
2 0  
2 1  
2 2

## Branching - Unconditional

```
/** The LabeledContinueApp class implements an application that illustrate LabelledContinue loop */
public class LabelledContinue {
    public static void main (String args[]){
        int MaxSeatCount = 5, TotalScreenCount = 2, seatCount = 0, screenCount = 0;

        SkipScreen:
        //System.out.println(screenCount);
        while(screenCount < TotalScreenCount){

            System.out.println("Screen "+(screenCount)+" Ticket Booking detail");
            seatCount=0;
            SkipSeat:
            while(seatCount < MaxSeatCount){

                seatCount++;
```

## Branching - Unconditional

```
if((seatCount == 2 || seatCount == 3) && screenCount == 1) {  
    System.out.println("Seat No "+seatCount+" is Reserved");  
    continue SkipSeat;  
}  
else  
    System.out.println("Seats No Booked : "+(seatCount));  
  
}  
System.out.println("All Seats Filled in Screen "+(screenCount+1));  
screenCount++;  
}  
}  
}
```

## Branching - Unconditional

**Output:**

Screen 1 Ticket Booking detail

Seats No Booked : 1

Seats No Booked : 2

Seats No Booked : 3

Seats No Booked : 4

Seats No Booked : 5

All Seats Filled in Screen 1

Screen 2 Ticket Booking detail

Seats No Booked : 1

Seat No 2 is Reserved

Seat No 3 is Reserved

Seats No Booked : 4

Seats No Booked : 5

All Seats Filled in Screen 2

## Quiz



1. What do you call a statement that executes a certain statement(s) ,if the condition is true, and executes a different statement(s) if the condition is false?

a) if statement

b) if else statement

c) if else if statement

d) None of the Above

b) if else statement

## Quiz



2. From the given if-else statement, what will be displayed if the value of num is 6?

```
if (num>0)
    System.out.println("Positive Number\n");
else
    System.out.println("Negative Number\n");
    System.out.println("The number is %d",num);
```

a) Positive Number

b) Negative Number

c) Positive Number  
The number is 6

d) The number is 6

c) Positive Number  
The number is 6

## Quiz



3. In switch case statement, what will be executed if there is no case matched?

- a) case
- b) break
- c) default
- d) All the above

c) default

## Quiz



4. What is a java keyword used in switch statements that causes immediate exit or that terminates the switch statement?
- a) The case keyword
  - b) The switch keyword
  - c) The default keyword
  - d) The break keyword
  - d) The break keyword

## Quiz



5. Which of the following loops will execute the statements at least once, even though the condition is false initially?

a) while

b) do...while

c) for

d) All the options

b) do...while

## Quiz



6. What value is stored in index at the end of this loop?

```
for(int index =1;index<=10;index++)
```

a) 1

b) 9

c) 10

d) 11

b) 11

## Quiz



7. Which of the following are true about the enhanced for loop?

- A. It can iterate over an array or a Collection but not a Map.
- B. Using an enhanced for loop prevents the code from going into an infinite loop
- C. Using an enhanced for loop on an array may cause infinite loop.
- D. An enhanced for loop can iterate over a Map.
- E. You cannot find out the number of the current iteration while iterating.

a) A,B,E

b) A,B,C

c) B,C,E

d) A,D,E

a) A,B,E

## Quiz



8. What is printed as a result of the following code segment?

```
for (int k = 0; k < 20; k+=2)
{
    if (k % 3 == 1)
        System.out.print(k + " ");
}
```

a) 0 2 4 6 8 10 12 14 16 18

b) 4,6

c) 4,10,16

d) 0,6,12,18

c) 4,10,16

## Quiz



**9. What is the output after the following code has been executed?**

```
class FlowControl1 {  
  
    public static void main(String[] args){  
  
        int value1= 10, value2 = 20;  
  
        if (value1 < value2 ){  
  
            if (value2 > value2 )  
  
                System.out.println("Hello Friend");  
  
            else  
  
                System.out.println(" Happy Day!");  
  
        }  
  
    }  
  
}
```

## Quiz



**10. What is stored in the variable result after the following code has been executed?**

```
class FlowControl2{  
    public static void main(String args[]){  
        int index = 0;  
        int result = 1;  
        while ( true ){  
            ++index;  
            if ( index % 2 == 0 )  
                continue;  
            else if ( index % 5 == 0 )  
                break;  
            result *= 3;  
        }  
    }  
}
```

# Quiz



**11. What is the output after the following code has been executed?**

```
class FlowControl3 {  
  
    public static void main(String[] args){  
  
        boolean b = true;  
  
        if(!b) {  
  
            System.out.println("HELLO");  
  
        } else {  
  
            System.out.println("BYE");  
  
        }  
  
    }  
}
```

## Control Flow Statements

# Quiz



**12. What is the output after the following code has been executed?**

```
class FlowControl4 {  
  
    public static void main(String[] args){  
  
        for (int i = 0; ; i++) {  
  
            System.out.println("HIII");  
  
        }  
  
        System.out.println("BYE");  
  
    }  
}
```

# Quiz



13. What is stored in the result after the following code has been executed?

```
class Test {  
  
    public static void main(String[] args){  
  
        do {  
  
            System.out.print(1);  
  
            do {  
  
                System.out.print(2);  
  
            } while (false);  
  
        } while (false);  
  
    }  
}
```

”

Action is the  
foundational key to  
all success

- Pablo Picasso

# THANK YOU