

# Fundamentals of Java

## Session: 4

### Decision-Making Constructs





- ◆ Identify the need for decision-making statements
- ◆ List the different types of decision-making statements
- ◆ Explain the if statement
- ◆ Explain the various forms of if statement
- ◆ Explain the switch-case statement
- ◆ Explain the use of strings and enumeration in the switch-case statement
- ◆ Compare the if-else and switch-case statement

For Aptechnology Centre Use Only



- ◆ A Java program consists of a set of statements which are executed sequentially in the order in which they appear.
- ◆ The change in the flow of statements is achieved by using different **control flow statements**.
- ◆ Three categories of control flow statements supported by Java programming language are as follows:

## 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.



- ◆ Enable us to change the flow of execution of a Java program.
- ◆ Evaluates a condition and based on the result of evaluation, a statement or a sequence of statements is executed.
- ◆ Different types of decision-making statements supported by Java are as follows:

**if Statement**

**Switch-case  
Statement**

## 'if' Statement 1-5



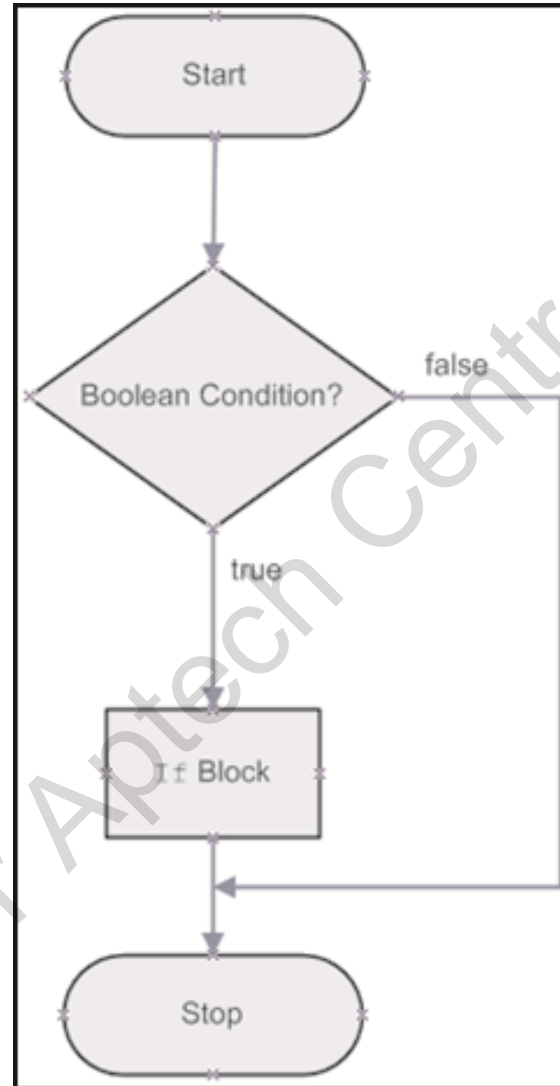
- ◆ It is the most basic form of decision-making statement.
- ◆ It evaluates a given condition and based on the result of evaluation executes a certain section of code.
- ◆ If the condition evaluates to `true`, then the statements present within the `if` block gets executed.
- ◆ If the condition evaluates to `false`, the control is transferred directly to the statement outside the `if` block.

For Apteck Centre Use Only

## 'if' Statement 2-5



- ◆ Following figure shows the flow of execution for the `if` statement:





- ◆ The syntax for using the `if` statement is as follows:

### Syntax

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

where,

`condition`: Is the boolean expression.

`statements`: Are instructions/statements enclosed in curly braces. These statements are executed when the boolean expression evaluates to `true`.



- ◆ Following code snippet demonstrates the code that performs conditional check on the value of a variable:

```
public class CheckNumberValue {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        int first = 400, second = 700, result;  
        result = first + second;  
  
        // Evaluates the value of result variable  
        if (result > 1000) {  
            second = second + 100;  
        }  
        System.out.println("The value of second is " + second);  
    }  
}
```

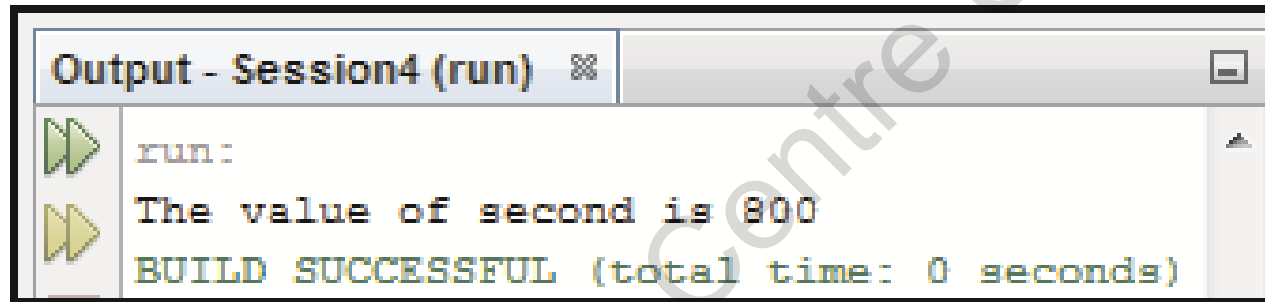
- ◆ The program tests the value of the variable, **result** and accordingly calculates value for the variable, **second**.
- ◆ If the value of **result** is greater than 1000, then the value of the variable **second** is incremented by 100.



## 'if' Statement 5-5



- ◆ If the evaluation of condition is `false`, the value of the variable **second** is not incremented.
- ◆ Finally, the value of the variable **second** gets printed on the console.
- ◆ Following figure shows the output of the code:



```
run:
The value of second is 800
BUILD SUCCESSFUL (total time: 0 seconds)
```

- ◆ If there is only a single action statement within the body of the `if` block, then use of opening and closing curly braces is optional.

# 'if-else' Statement 1-3



- ◆ Sometimes, it is required to define a block of statements to be executed when a condition evaluates to `false`.
- ◆ This is done by using the `if-else` statement.
- ◆ `if-else` Statement:
  - ◆ Begins with the `if` block followed by the `else` block.
  - ◆ `else` block specifies a block of statements that are to be executed when a condition evaluates to `false`.
- ◆ The syntax for using the `if-else` statement is as follows:

## Syntax

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

## 'if-else' Statement 2-3



- ◆ Following code snippet demonstrates the code that checks whether a number is even or odd:

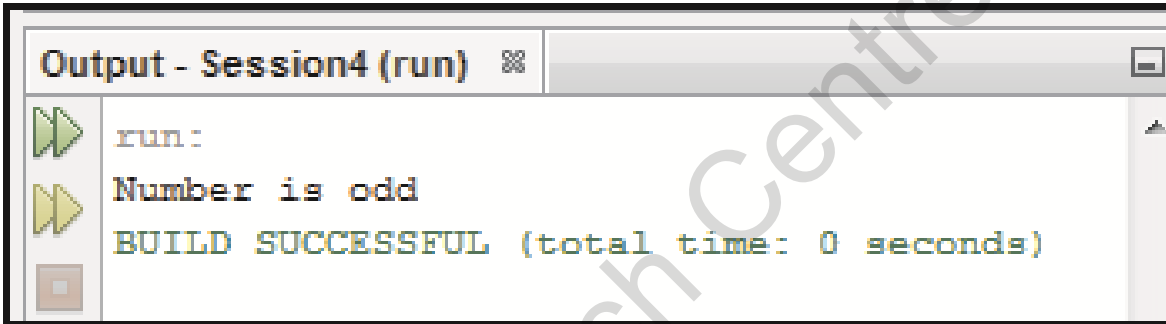
```
public class Number_Division {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        int number = 11, remainder;  
  
        // % operator to return the remainder of the division  
        remainder = number % 2;  
        if (remainder == 0) {  
            System.out.println("Number is even");  
        } else {  
            System.out.println("Number is odd");  
        }  
    }  
}
```

- ◆ In the code, the variable, **number** is divided by 2 to obtain the remainder of the division.

## 'if-else' Statement 3-3



- ◆ The `%` (modulus) operator which returns the remainder after performing the division.
- ◆ If the remainder is 0, the message **Number is even** is printed. Otherwise, the message **Number is odd** is printed.
- ◆ Following figure shows the output of the code:



```
run:
Number is odd
BUILD SUCCESSFUL (total time: 0 seconds)
```



- ◆ An `if` statement can also be used within another `if` statement forming a nested-if.
- ◆ A nested-if statement is an `if` statement that is the target of another `if` or `else` statement.
- ◆ The syntax to use the nested-if statements is as follows:

## Syntax

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



- ◆ Following code snippet checks whether a number is divisible by 3 as well as 5:

```
import java.util.*;
public class NumberDivisibility {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Scanner class is used to accept values from the user
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a Number: ");
        int num = input.nextInt();

        // Checks whether a number is divisible by 3
        if (num % 3 == 0) {
            System.out.println("Inside Outer if Block");

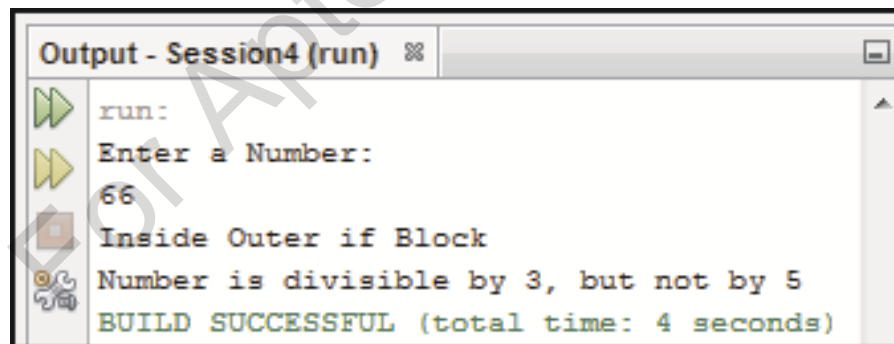
            // Inner if statement checks if number is divisible by 5
            if (num % 5 == 0) {
                System.out.println("Number is divisible by 3 and 5");
            } else {
                System.out.println("Number is divisible by 3, but not by 5");
            } // End of inner if-else statement
        }
    }
}
```

# Nested-if Statement 3-4



```
} else {  
    System.out.println("Number is not divisible by 3");  
} // End of outer if-else statement  
  
}  
  
}
```

- ◆ The code declares a variable **num** to store an integer value accepted from the user.
- ◆ Initially, the outer **if** statement is evaluated. If it evaluate to:
  - ◆ **false**, then the inner **if-else** statement is skipped and the final **else** block is executed.
  - ◆ **true**, then its body containing the inner **if-else** statement is evaluated.
- ◆ Following figure shows the output of the code:





- ◆ The important points to remember about nested-if statements are as follows:

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

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

For Aptech Centre Use Only





- ◆ 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.
- ◆ When a condition controlling the `if` statement is evaluated as `true`, then the associated statements associated are executed and all other `else-if` statements are bypassed.
- ◆ If none of the condition is `true`, then the final `else` statement also referred as default statement is executed.

For Aptech Centre Use Only



- ◆ The syntax for using the `if-else-if` statement is as follows:

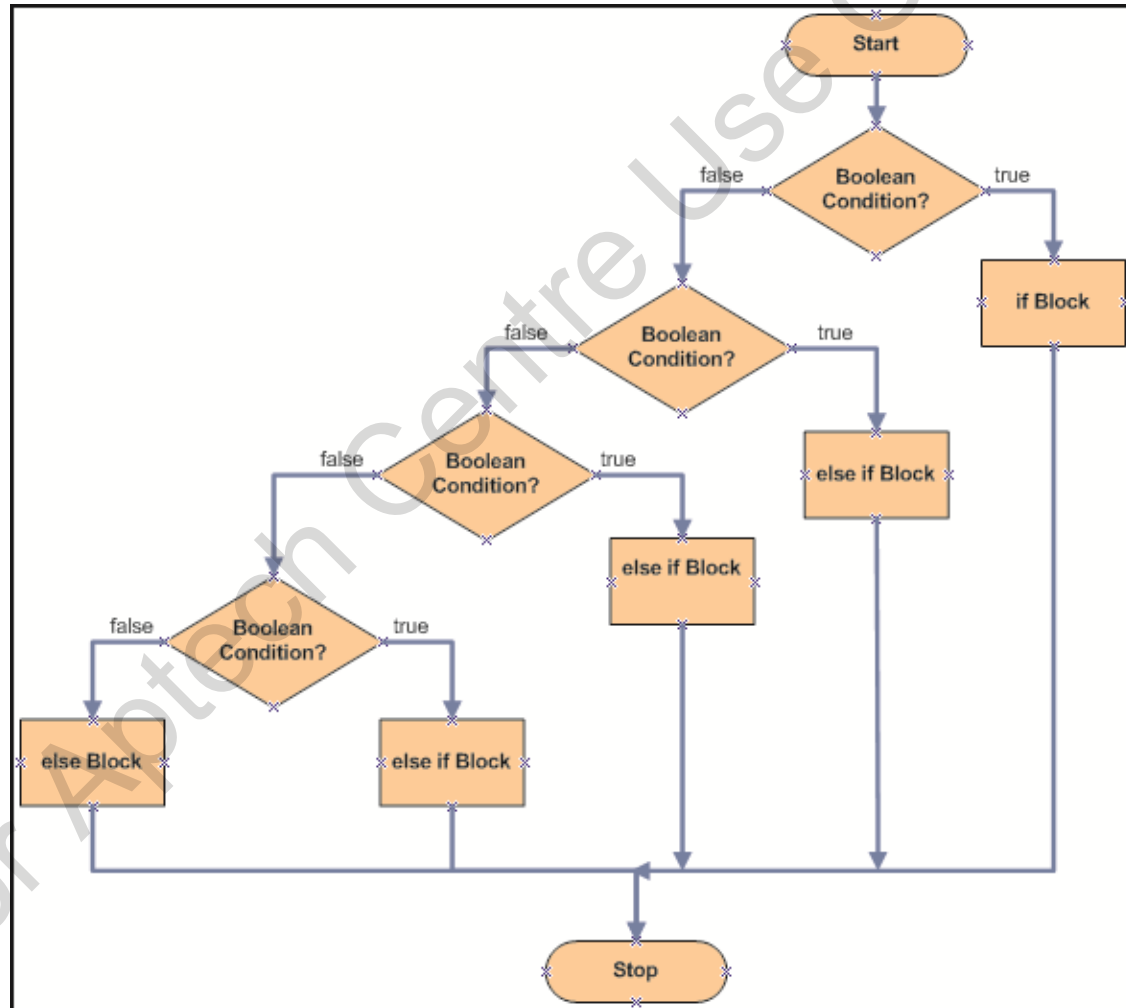
## Syntax

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

## 'if-else-if' Ladder 3-5



- ◆ Following figure shows the flow of execution for the `if-else-if` ladder:





- ◆ Following code snippet checks the total marks and prints the appropriate grade:

```
public class CheckMarks {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        int totalMarks = 59;  
        /* Tests the value of totalMarks and accordingly transfers  
         * control to the else if statement  
         */  
        if (totalMarks >= 90) {  
            System.out.println("Grade = A+");  
        } else if (totalMarks >= 60) {  
            System.out.println("Grade = A");  
        } else if (totalMarks >= 40) {  
            System.out.println("Grade = B");  
        } else if (totalMarks >= 30) {  
            System.out.println("Grade = C");  
        } else {  
            System.out.println("Fail");  
        }  
    }  
}
```

# 'if-else-if' Ladder 5-5



- ◆ If the code satisfies a given condition, then:
  - ◆ The statements within that `else if` condition are executed.
  - ◆ After execution of the statements, the control breaks.
  - ◆ Remaining `if` conditions are bypassed for evaluation.
- ◆ If none of the condition is satisfied, then:
  - ◆ The final `else` statement, also known as the default `else` statement is executed.
- ◆ Following figure shows the output of the code:

```
run:
Grade = B
BUILD SUCCESSFUL (total time: 0 seconds)
```

# 'switch-case' Statement 1-11



- ◆ Alternative for too many `if` statements representing multiple selection constructs.
- ◆ Contains a variable as an expression whose value is compared against different values.
- ◆ Results in better performance.
- ◆ Can have a number of possible execution paths depending on the value of expression provided with the `switch` statement.
- ◆ Can evaluate different primitive data types, such as `byte`, `short`, `int`, and `char`.

For Aptech Centre Use Only



## Enhancements to switch-case statement in Java SE 7

- Supports the 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 the Java API.
- The classes whose objects can be used are `Character`, `Byte`, `Short`, and `Integer`.
- Supports the use of enumerated types as expression.



- ◆ The syntax for using the switch-case statement is as follows:

## Syntax

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

where,

**switch:** The switch keyword is followed by an expression enclosed in parentheses.



# 'switch-case' Statement 4-11



`case`: The `case` keyword is followed by a constant and a colon. Each case value is a unique literal.

`default`: If no case value matches the switch expression value, execution continues at the `default` clause.

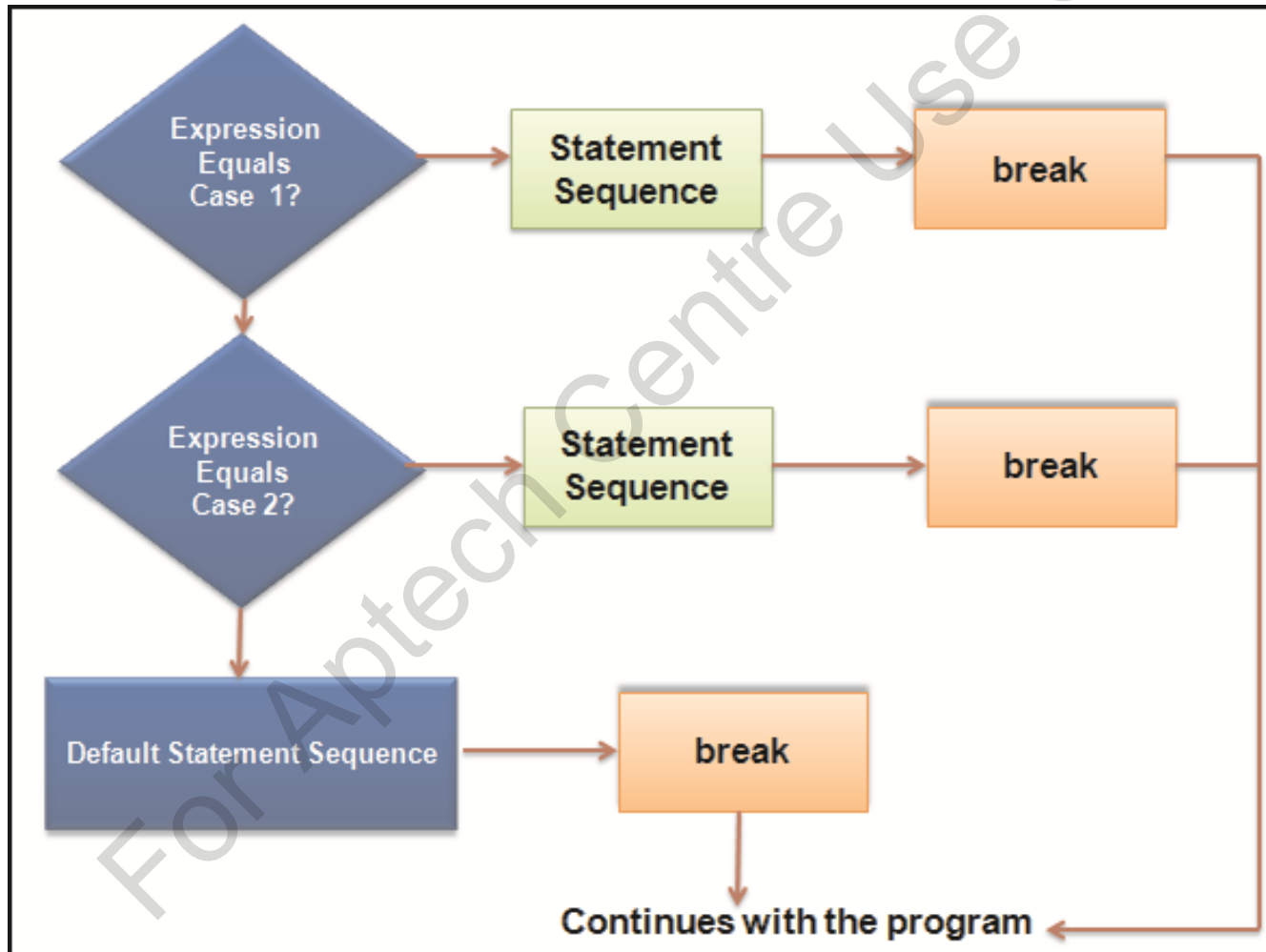
`break`: The `break` statement is used inside the `switch-case` statement to terminate the execution of the statement sequence. It is optional. If there is no `break` statement, execution flows sequentially into the next cases.

For Aptech Certified Use Only

# 'switch-case' Statement 5-11



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



# 'switch-case' Statement 6-11



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.



- ◆ Following code snippet demonstrates the use of the switch-case statement:

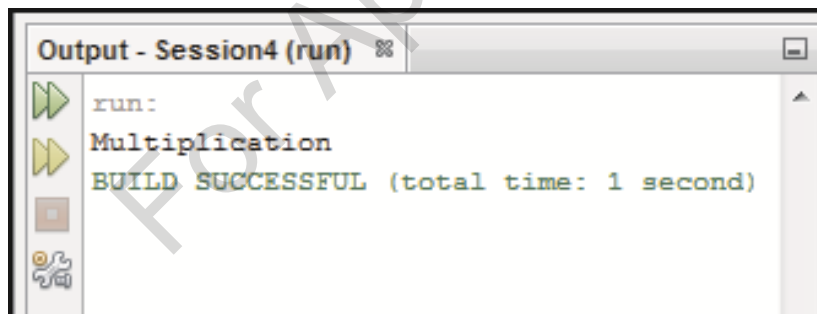
```
public class TestNumericOperation {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
  
        // Declares and initializes the variable  
        int choice = 3;  
  
        // switch expression value is matched with each case  
        switch (choice) {  
            case 1:  
                System.out.println("Addition");  
                break;  
            case 2:  
                System.out.println("Subtraction");  
                break;  
            case 3:  
                System.out.println("Multiplication");  
                break;  
        }  
    }  
}
```

# 'switch-case' Statement 8-11



```
case 4:
    System.out.println("Division");
    break;
default:
    System.out.println("Invalid Choice");
} // End of switch-case statement
}
```

- ❖ Value of the expression, **choice** is compared with the literal value in each of the case statement.
- ❖ Here, case 3 is executed, as its value is matching with the expression.
- ❖ The control moves out of the switch-case, due to the presence of the break statement.
- ❖ Following figure shows the output of the code:



# 'switch-case' Statement 9-11



- ◆ Sometimes, it is required to have multiple case statements to be executed without a break statement.
- ◆ Following code snippet demonstrates the use of multiple case statements with no break statement:

```
public class NumberOfDays {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
  
        int month = 5;  
        int year = 2001;  
        int numDays = 0;  
  
        // Cases are executed until a break statement is encountered  
        switch (month) {  
            case 1:  
            case 3:  
            case 5:  
            case 7:
```

# 'switch-case' Statement 10-11



```
case 8:
case 10:
case 12:
    numDays = 31;
    break;

case 4:
case 6:
case 9:
case 11:
    numDays = 30;
    break;

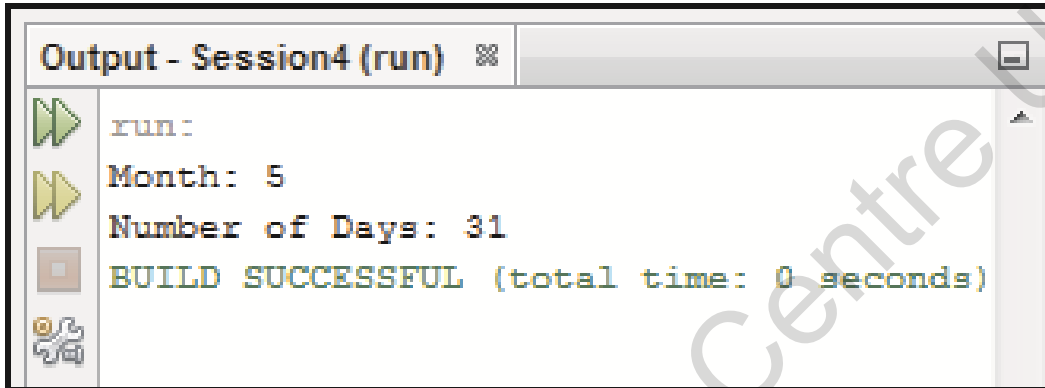
case 2:
    if (year % 4 == 0) {
        numDays = 29;
    } else {
        numDays = 28;
    }
    break;

default:
    System.out.println("Invalid Month");
} // End of switch-case statement
System.out.println("Month: " + month);
System.out.println("Number of Days: " + numDays);
}
}
```

# 'switch-case' Statement 11-11



- ◆ The value of expression, **month** is compared through each case, till a **break** statement or end of the **switch-case** block is encountered.
- ◆ Following figure shows the output of the code:



```
run:
Month: 5
Number of Days: 31
BUILD SUCCESSFUL (total time: 0 seconds)
```

For Aptech Centre Use Only



# String-based 'switch-case' Statement 1-4



- ◆ Java SE 7 supports the use of strings in the switch-case statement.
- ◆ A `String` is not a primitive data type, but an object in Java.
- ◆ To use strings for comparison, a `String` object is passed as an expression in the switch-case statement.
- ◆ Following code snippet demonstrates the use of strings in the switch-case statement:

```
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;  
        }  
    }  
}
```

# String-based 'switch-case' Statement 2-4

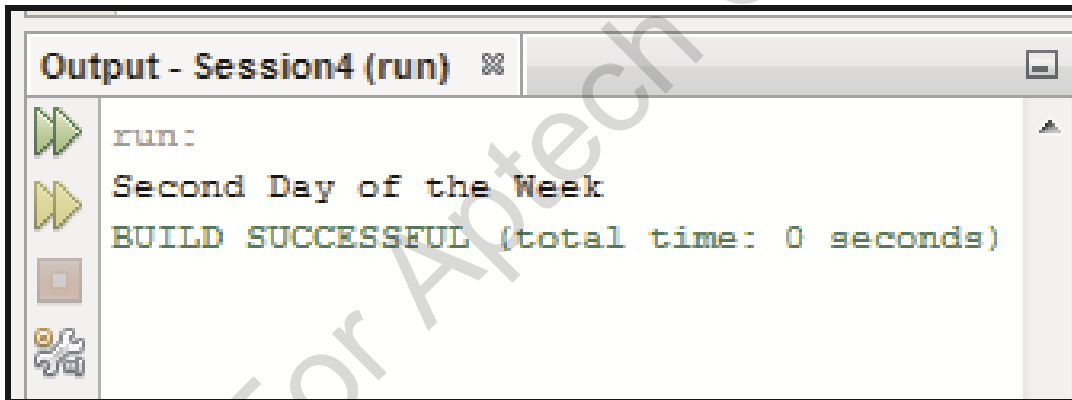


```
case "Monday":
    System.out.println("Second Day of the Week");
    break;
case "Tuesday":
    System.out.println("Third Day of the Week");
    break;
case "Wednesday":
    System.out.println("Fourth Day of the Week");
    break;
case "Thursday":
    System.out.println("Fifth Day of the Week");
    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
}
```

# String-based 'switch-case' Statement 3-4



- ◆ The statement **String day="Monday"** creates an object named day of type `String` and initializes it.
  - ◆ The object is passed as an expression to the `switch` statement.
  - ◆ The value of this expression, that is "Monday", is compared with the value of each `case` statement.
  - ◆ If no matching statement is found, then the statement associated with the default clause is executed.
- ◆ Following figure shows the output of the code:



```
Output - Session4 (run)
run:
Second Day of the Week
BUILD SUCCESSFUL (total time: 0 seconds)
```



- ◆ 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.

# Enumeration-based 'switch-case' Statement 1-3



- ◆ 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.
- ◆ Following code snippet demonstrates the use of enumerations in the `switch-case` 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;  
    }  
}
```

## Enumeration-based 'switch-case' Statement 2-3



```
// 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
}
```

- ❖ The enum, **card** is passed as an expression to the `switch` statement.
- ❖ Each case statement has an enumeration constant associated with it and does not require it to be qualified by the enumeration name.

# Enumeration-based 'switch-case' Statement 3-3



- ◆ Following figure shows the output of the code:

A screenshot of an IDE's output window titled "Output - Session4 (run)". The window contains three lines of text: "run:", "DIAMOND", and "BUILD SUCCESSFUL (total time: 0 seconds)". The first two lines are preceded by green double arrow icons, indicating they are the primary output of the program. The third line is preceded by a yellow double arrow icon, indicating it is a secondary or informational message. The text is displayed in a monospaced font with syntax highlighting: "run:" is in black, "DIAMOND" is in blue, and "BUILD SUCCESSFUL" is in green, with "(total time: 0 seconds)" in black.

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

For Aptech Centre Use Only

# Nested 'switch-case' Statement 1-4



- ◆ A switch-case statement can be used as a part of another switch-case statement. This is referred to as nested switch-case statements.
- ◆ Following code snippet demonstrates the use of nested switch-case statements:

```
public class Greeting {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        // String declaration  
        String day = "Monday";  
        String hour = "am";  
  
        // Outer switch statement  
        switch (day) {  
            case "Sunday":  
                System.out.println("Sunday is a Holiday...");  
                // Inner switch statement  
                switch (hour) {
```



# Nested 'switch-case' Statement 2-4



```
        case "am":
            System.out.println("Good Morning");
            break;
        case "pm":
            System.out.println("Good Evening");
            break;
    } // End of inner switch-case statement
    break; // Terminates the outer case statement

case "Monday":
    System.out.println("Monday is a Working Day...");
    switch (hour) {
        case "am":
            System.out.println("Good Morning");
            break;
        case "pm":
            System.out.println("Good Evening");
            break;
    } // End of inner switch-case statement
    break;
default:
    System.out.println("Invalid Day");
} // End of the outer switch-case statement
}
```

## Nested 'switch-case' Statement 3-4



- ◆ The variable, **day** is used as an expression with the outer `switch` statement.
  - ◆ If the value of **day** variable matches with “**Sunday**” or “**Monday**”, then the inner `switch-case` statement is executed.
  - ◆ The inner `switch` statement compares the value of **hour** variable with case constants “**am**” or “**pm**”.
- ◆ Following figure shows the output of the code:

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

# Nested 'switch-case' Statement 4-4



- ◆ The 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.

# Comparison Between if and switch-case Statement



- ◆ Following table lists the differences between `if` and `switch-case` statement:

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



- ◆ 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 needs to be compared against different values. Java SE 7 supports 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.