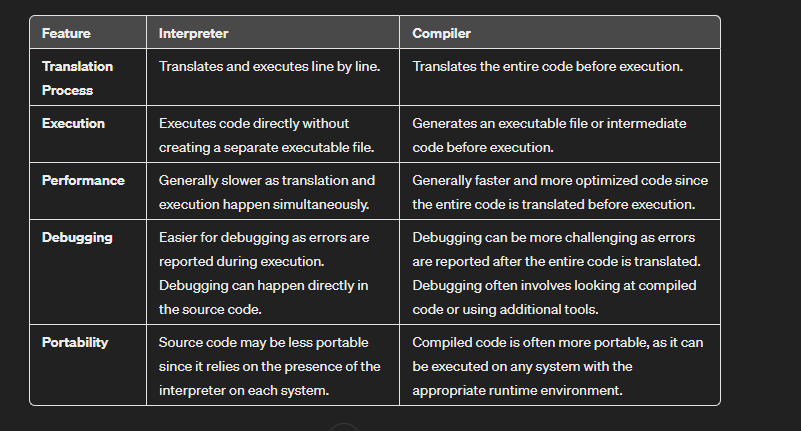
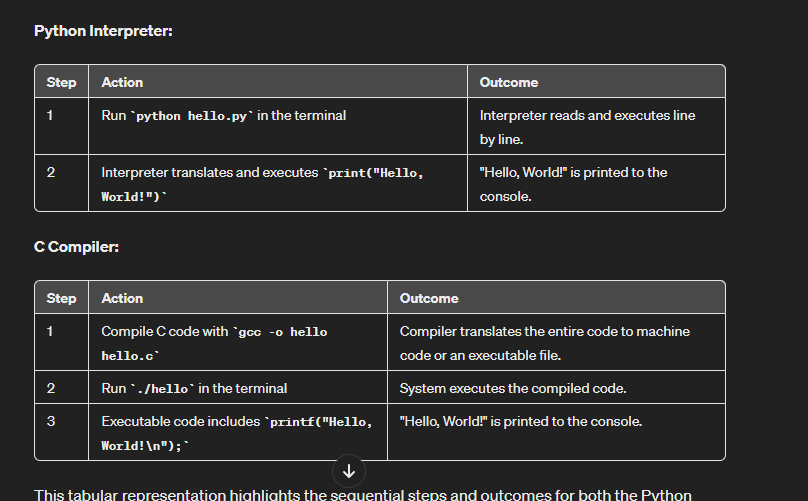
**15th Feb 2024**

## **Interpreter Vs Compiler :**



Example :



## Java Control Statements | Control Flow in Java

* Java compiler executes the code from top to bottom.
* The statements in the code are executed according to the order in which they appear.
* However, [Java](https://www.javatpoint.com/java-tutorial) provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

Java provides three types of control flow statements.

1. Decision Making statements
   * if statements
   * switch statement
2. Loop statements
   * do while loop
   * while loop
   * for loop
   * for-each loop
3. Jump statements
   * break statement
   * continue statement

##### 

##### **Decision-Making statements:**

* decision-making statements decide which statement to execute and when
* Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.
* Types : - if & Switch

1. **If statement :**

* In Java, the "if" statement is used to evaluate a condition.
* The condition of the If statement gives a Boolean value, either true or false.
* **Syntax :** 
  + - **if(condition) {**

**statement 1; //executes when condition is true**

**}**

* **if(condition) {**

**statement 1; //executes when condition is true**

**}**

**else{**

**statement 2; //executes when condition is false**

**}**

* **if-else-if statement**

**if(condition 1) {**

**statement 1; //executes when condition 1 is true**

**}**

**else if(condition 2) {**

**statement 2; //executes when condition 2 is true**

**}**

**else {**

**statement 2; //executes when all the conditions are false**

**}**

* **Nested if**

**if(condition 1) {**

**statement 1; //executes when condition 1 is true**

**if(condition 2) {**

**statement 2; //executes when condition 2 is true**

**}**

**else{**

**statement 2; //executes when condition 2 is false**

**}**

**}**

1. **switch statement :**

* In Java, [Switch statements](https://www.javatpoint.com/java-switch) are similar to if-else-if statements.
* The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.
* Syntax :
  + **switch** (expression){

case value1:

statement1;

break;

.

.

.

case valueN:

statementN;

break;

default:

default statement;

}

**NOTE :**

* The case variables can be int, short, byte, char, or enumeration.
* String type is also supported since version 7 of Java
* Cases cannot be duplicate. In case of duplicate value, it renders compile-time error.
* Default statement is executed when any of the case doesn't match the value of expression. It is optional.
* Break statement terminates the switch block when the condition is satisfied.  
  It is optional, if not used, next case is executed.
* While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value
* While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.
* There can be *one or N number of case values* for a switch expression.

##### **Loop statements :**

* sometimes we need to execute the block of code repeatedly while some condition evaluates to true.
* However, loop statements are used to execute the set of instructions in a repeated order.
* The execution of the set of instructions depends upon a particular condition.
* In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

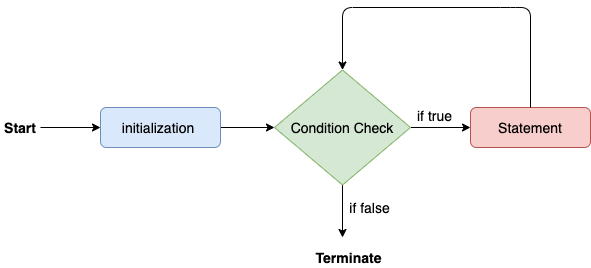
1. for loop
2. for - each loop
3. while loop
4. do-while loop
5. **for loop :** 
   * It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
   * We use the for loop only when we exactly know the number of times, we want to execute the block of code.
   * Syntax :

**for**(initialization, condition, increment/decrement) {

//block of statements

}

* Flow :



1. **For- each loop :** 
   * Java provides an enhanced for loop to traverse the data structures like array or collection.
   * In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

for(data\_type var : array\_name/collection\_name){

//statements

}

Example :

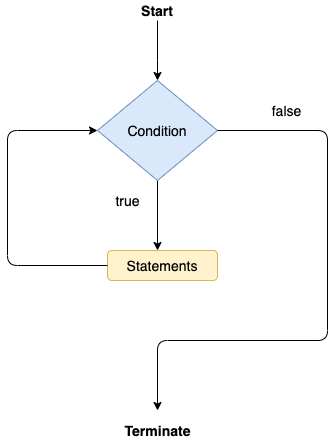
1. **public** **class** Calculation {
2. **public** **static** **void** main(String[] args) {
3. // TODO Auto-generated method stub
4. String[] names = {"Java","C","C++","Python","JavaScript"};
5. System.out.println("Printing the content of the array names:\n");
6. **for**(String name:names) {
7. System.out.println(name);
8. }
9. }
10. }

1. **while loop :** 
   * The [while loop](https://www.javatpoint.com/java-while-loop) is also used to iterate over the number of statements multiple times.
   * However, if we don't know the number of iterations in advance, it is recommended to use a while loop.
   * Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.
   * It is also known as the entry-controlled loop since the condition is checked at the start of the loop.
   * If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.
   * The syntax of the while loop is given below.

**while**(condition){

//looping statements

}

* + The flow chart for the while loop is given in the following image.
  + 

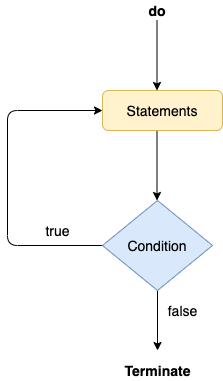
1. **do-while loop :** 
   * The [do-while loop](https://www.javatpoint.com/java-do-while-loop) checks the condition at the end of the loop after executing the loop statements.
   * When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.
   * It is also known as the exit-controlled loop since the condition is not checked in advance.
   * The syntax of the do-while loop is given below.

**do**

{

//statements

} **while** (condition);

* + The flow chart of the do-while loop is given in the following image.
    - * 
  + Consider the following example to understand the functioning of the do-while loop in Java.

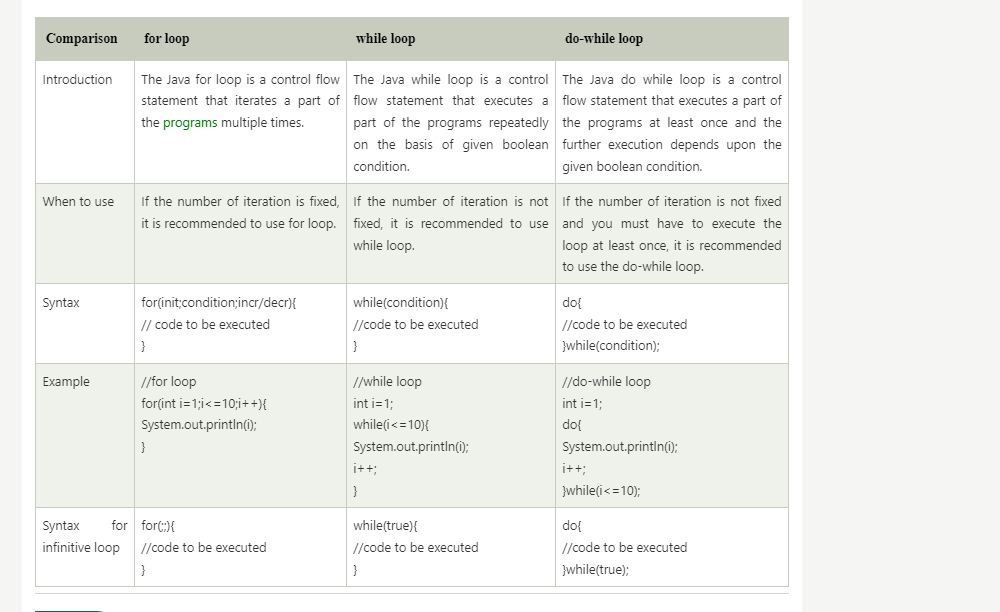
**Calculation.java**

1. **public** **class** Calculation {
2. **public** **static** **void** main(String[] args) {
3. // TODO Auto-generated method stub
4. **int** i = 0;
5. System.out.println("Printing the list of first 10 even numbers \n");
6. **do** {
7. System.out.println(i);
8. i = i + 2;
9. }**while**(i<=10);
10. }
11. }

**Output:**

* + Printing the list of first 10 even numbers
  + 0
  + 2
  + 4
  + 6
  + 8

10



##### **Jump statements :**

* Jump statements are used to transfer the control of the program to the specific statements.
* In other words, jump statements transfer the execution control to the other part of the program.
* There are two types of jump statements in Java, i.e., break and continue.

1. **Break Statement:**

* the [break statement](https://www.javatpoint.com/java-break) is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.
* The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.
* **The break statement example with for loop**

Consider the following example in which we have used the break statement with the for loop.

**BreakExample.java**

**public** **class** BreakExample {

**public** **static** **void** main(String[] args) {

// TODO Auto-generated method stub

**for**(**int** i = 0; i<= 10; i++) {

System.out.println(i);

**if**(i==6) {

**break**;

}

}

}

}

* **Output:**

0

1

2

3

4

5

6

1. **Continue statement :**

Unlike break statement, the [continue statement](https://www.javatpoint.com/java-continue) doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

1. **public** **class** ContinueExample {
3. **public** **static** **void** main(String[] args) {
4. // TODO Auto-generated method stub
6. **for**(**int** i = 0; i<= 2; i++) {
8. **for** (**int** j = i; j<=5; j++) {
10. **if**(j == 4) {
11. **continue**;
12. }
13. System.out.println(j);
14. }
15. }
16. }
18. }

**Output:**

ADVERTISEMENT

0

1

2

3

5

1

2

3

5

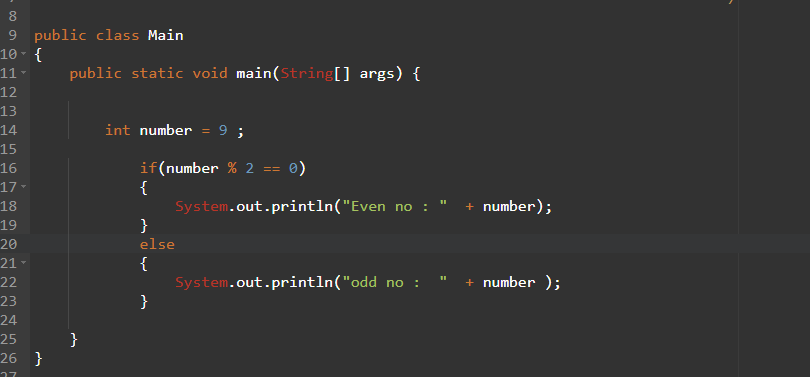
2

3

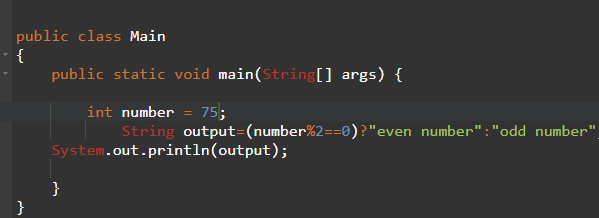
5

## **JAVA Practice Questions :**

**Q1 : It is a program of odd and even number.**

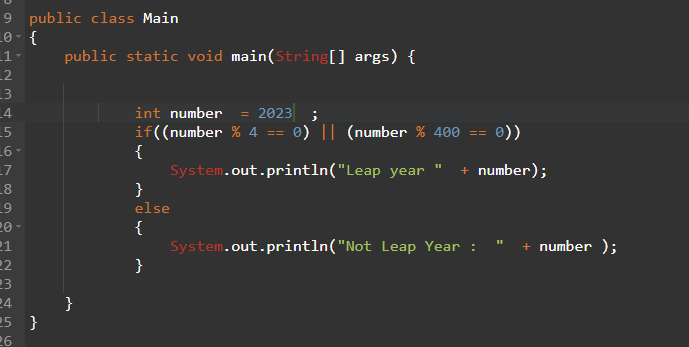
****

**Using ternary operator :**



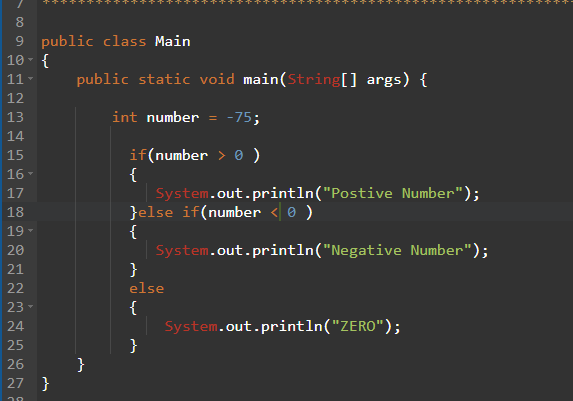
**Q2 : Leap Year Example:**

**ANS : A year is leap, if it is divisible by 4 and 400. But, not by 100.**



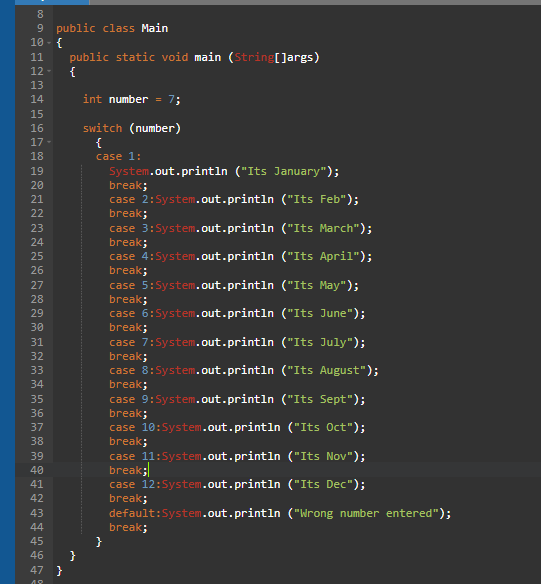
Q3 : **Program to check POSITIVE, NEGATIVE or ZERO:**

**ANS :**

****

**Q4 : Finding Month Example:**

**Ans :**

****

**Q5 : Program to check Vowel or Consonant:**

**ANS : If the character is A, E, I, O, or U, it is vowel otherwise consonant. It is not case-sensitive.**

****

**Q5 : Java Program to demonstrate the use of Java Switch statement with String**

**ANS :**

****

## Java Enum in Switch Statement

## **Java Enums :**

* In Java, Enumerations or Java Enum serve the purpose of representing a group of named constants in a programming language.
* Java Enums are used when we know all possible values at compile time, such as choices on a menu, rounding modes, command-line flags, etc.
* The set of constants in an enum type doesn’t need to stay fixed for all time.
* A Java enumeration is a class type. Although we don’t need to instantiate an enum using new, it has the same capabilities as other classes. This fact makes Java enumeration a very powerful tool. Just like classes, you can give them constructors, add instance variables and methods, and even implement interfaces.
* Enum Example:
* The 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit. Other examples include natural enumerated types (like the planets, days of the week, colors, directions, etc.).
* One thing to keep in mind is that, unlike classes, enumerations neither inherit other classes nor can get extended(i.e become superclass). We can also add variables, methods, and constructors to it. The main objective of an enum is to define our own data types(Enumerated Data Types).
* Declaration of enum in Java
* Enum declaration can be done outside a Class or inside a Class but not inside a Method.
* 1. Declaration outside the class
* Java
* // A simple enum example where enum is declared
* // outside any class (Note enum keyword instead of
* // class keyword)
* enum Color {
* RED,
* GREEN,
* BLUE;
* }
* public class Test {
* // Driver method
* public static void main(String[] args)
* {
* Color c1 = Color.RED;
* System.out.println(c1);
* }
* }
* Output
* RED
* 2. Declaration inside a class
* Java
* // enum declaration inside a class.
* public class Test {
* enum Color {
* RED,
* GREEN,
* BLUE;
* }
* // Driver method

## public static void main(String[] args)

## {

## Color c1 = Color.RED;

## System.out.println(c1);

## }

## }

## 

## Output

## RED

## 

## 

## The first line inside the enum should be a list of constants and then other things like methods, variables, and constructors.

## According to Java naming conventions, it is recommended that we name constant with all capital letters

## Properties of Enum in Java

## There are certain properties followed by Enum as mentioned below:

## Every enum is internally implemented by using Class.

## Every enum constant represents an object of type enum.

## Enum type can be passed as an argument to switch statements.

## Every enum constant is always implicitly public static final. Since it is static, we can access it by using the enum Name. Since it is final, we can’t create child enums.

## We can declare the main() method inside the enum. Hence we can invoke the enum directly from the Command Prompt.

## Below is the implementation of the above properties:

## 

## Java

## // A Java program to demonstrate working on enum

## // in switch case (Filename Test. Java)

## 

## import java.util.Scanner;

## 

## // An Enum class

## enum Day {

## SUNDAY,

## MONDAY,

## TUESDAY,

## WEDNESDAY,

## THURSDAY,

## FRIDAY,

## SATURDAY;

## }

## 

## // Driver class that contains an object of "day" and

## // main().

## public class Test {

## Day day;

## 

## // Constructor

## public Test(Day day) { this.day = day; }

## 

## // Prints a line about Day using switch

## public void dayIsLike()

## {

## switch (day) {

## case MONDAY:

## System.out.println("Mondays are bad.");

## break;

## case FRIDAY:

## System.out.println("Fridays are better.");

## break;

## case SATURDAY:

## case SUNDAY:

## System.out.println("Weekends are best.");

## break;

## default:

## System.out.println("Midweek days are so-so.");

## break;

## }

## }

## 

## // Driver method

## public static void main(String[] args)

## {

## String str = "MONDAY";

## Test t1 = new Test(Day.valueOf(str));

## t1.dayIsLike();

## }

## }

## 

## Output

## Mondays are bad.

## 

## Java Enum Programs

## 1. Main Function Inside Enum

## We can declare a main function inside an enum as we can invoke the enum directly from the Command Prompt.

## 

## Below is the implementation of the above property:

## 

## Java

## // A Java program to demonstrate that we can have

## // main() inside enum class.

## 

## enum Color {

## RED,

## GREEN,

## BLUE;

## 

## // Driver method

## public static void main(String[] args)

## {

## Color c1 = Color.RED;

## System.out.println(c1);

## }

## }

## 

## Output

## RED

## 

## 2. Loop through Enum

## We can iterate over the Enum using values( ) and loop. values() function returns an array of Enum values as constants using which we can iterate over the values.

## 

## Below is the implementation of the loop through Enum:

## 

## Java

## // Java Program to Print all the values

## // inside the enum using for loop

## import java.io.\*;

## 

## // Enum Declared

## enum Color {

## RED,

## GREEN,

## BLUE;

## }

## 

## // Driver Class

## class GFG {

## 

## // Main Function

## public static void main(String[] args)

## {

## // Iterating over all the values in

## // enum using for loop

## for (Color var\_1 : Color.values()) {

## System.out.println(var\_1);

## }

## }

## }

## 

## Output

## RED

## GREEN

## BLUE

## 

## 3. Enum in a Switch Statement

## Java

## // Java Program to implement

## // Enum in a Switch Statement

## import java.io.\*;

## 

## // Driver Class

## class GFG {

## // Enum Declared

## enum Color {

## RED,

## GREEN,

## BLUE,

## Yellow;

## }

## 

## // Main Function

## public static void main(String[] args)

## {

## Color var\_1=Color.Yellow;

## 

## // Switch case with Enum

## switch(var\_1){

## case RED:

## System.out.println("Red color observed");

## break;

## case GREEN:

## System.out.println("Green color observed");

## break;

## case BLUE:

## System.out.println("Blue color observed");

## default:

## System.out.println("Other color observed");

## }

## }

## }

## 

## Output

## Other color observed

## 

## Enum and Inheritance

## All enums implicitly extend java.lang.Enum class. As a class can only extend one parent in Java, an enum cannot extend anything else.

## toString() method is overridden in java.lang.Enum class, which returns enum constant name.

## enum can implement many interfaces.

## Enum and Constructor

## Enum can contain a constructor and it is executed separately for each enum constant at the time of the enum class loading.

## We can’t create enum objects explicitly and hence we can’t invoke the enum constructor directly.

## Enum and Methods

## Enum can contain both concrete methods and abstract methods. If an enum class has an abstract method, then each instance of the enum class must implement it.

## 

## Java

## // Java program to demonstrate that enums can have

## // constructor and concrete methods.

## 

## // An enum (Note enum keyword inplace of class keyword)

## enum Color {

## RED,

## GREEN,

## BLUE;

## 

## // enum constructor called separately for each

## // constant

## private Color()

## {

## System.out.println("Constructor called for : "

## + this.toString());

## }

## 

## public void colorInfo()

## {

## System.out.println("Universal Color");

## }

## }

## 

## public class Test {

## // Driver method

## public static void main(String[] args)

## {

## Color c1 = Color.RED;

## System.out.println(c1);

## c1.colorInfo();

## }

## }

## 

## Output

## Constructor called for : RED

## Constructor called for : GREEN

## Constructor called for : BLUE

## RED

## Universal Color

## 

## FAQs on Enum in Java

## Q1. Can we create the instance of Enum by the new keyword?

## Ans:

## 

## No, we can’t create the instance of the Enum keyword because it contains private constructors only.

## 

## Q2. Can we have an abstract method in the Enum?

## Ans:

## 

## Yes, we have an abstract method in Enum.

## 

## Q3. What is the purpose of the values() method in the enum?

## Ans:

## 

## In Java, the values( ) method can be used to return all values present inside the enum.

## 

## Q4. What is the purpose of the valueOf() method in the enum?

## Ans:

## 

## The valueOf() method returns the enum constant of the specified string value if exists.

## 

## Q5. What is the purpose of the ordinal() method in the enum?

## Ans:

## 

## By using the ordinal() method, each enum constant index can be found, just like an array index.

## 

## Q6. Write a program in Java to describe the use of values(), valueOf(), and ordinal() methods in the enum.

## Ans:

## 

## Java

## // Java program to demonstrate working of values(),

## // ordinal() and valueOf()

## 

## enum Color {

## RED,

## GREEN,

## BLUE;

## }

## 

## public class Test {

## public static void main(String[] args)

## {

## // Calling values()

## Color arr[] = Color.values();

## 

## // enum with loop

## for (Color col : arr) {

## // Calling ordinal() to find index

## // of color.

## System.out.println(col + " at index "

## + col.ordinal());

## }

## 

## // Using valueOf(). Returns an object of

## // Color with given constant.

## // Uncommenting second line causes exception

## // IllegalArgumentException

## System.out.println(Color.valueOf("RED"));

## // System.out.println(Color.valueOf("WHITE"));

## }

## }

## 

## Output

## RED at index 0

## GREEN at index 1

## BLUE at index 2

## RED

## 

## 

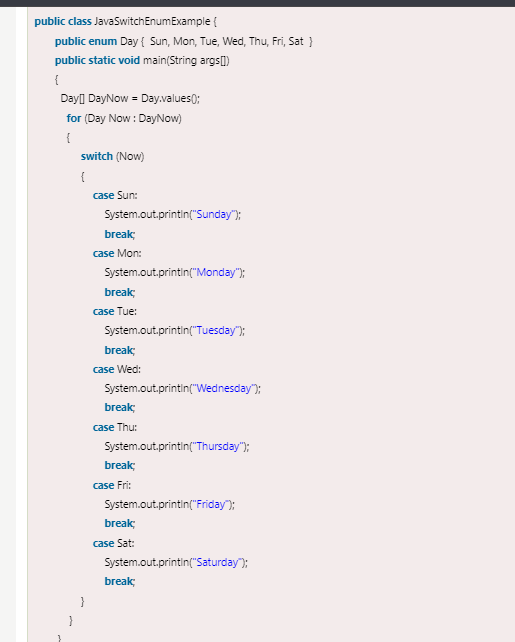
## 

## 

## Java Enum in Switch Statement

* Java allows us to use enum in switch statement. Java enum is a class that represent the group of constants. (immutable such as final variables). We use the keyword enum and put the constants in curly braces separated by comma.

**Java allows us to use enum in switch statement. Java enum is a class that represent the group of constants. (immutable such as final variables). We use the keyword enum and put the constants in curly braces separated by comma.**

****

## Java Labeled For Loop

**We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.**

#### Note: The break and continue keywords breaks or continues the innermost for loop respectively.

**Syntax:**

1. **labelname:**
2. **for(initialization; condition; increment/decrement){**
3. **//code to be executed**
4. **}**

**Example:**

**LabeledForExample.java**

1. **//A Java program to demonstrate the use of labeled for loop**
2. **public class LabeledForExample {**
3. **public static void main(String[] args) {**
4. **//Using Label for outer and for loop**
5. **aa:**
6. **for(int i=1;i<=3;i++){**
7. **bb:**
8. **for(int j=1;j<=3;j++){**
9. **if(i==2&&j==2){**
10. **break aa;**
11. **}**
12. **System.out.println(i+" "+j);**
13. **}**
14. **}**
15. **}**
16. **}**

**Output:**

**1 1**

**1 2**

**1 3**

**2 1**

**If you use break bb;, it will break inner loop only which is the default behaviour of any loop.**

**LabeledForExample2.java**

1. **public class LabeledForExample2 {**
2. **public static void main(String[] args) {**
3. **aa:**
4. **for(int i=1;i<=3;i++){**
5. **bb:**
6. **for(int j=1;j<=3;j++){**
7. **if(i==2&&j==2){**
8. **break bb;**
9. **}**
10. **System.out.println(i+" "+j);**
11. **}**
12. **}**
13. **}**
14. **}**

**Output:**

**1 1**

**1 2**

**1 3**

**2 1**

**3 1**

**3 2**

**3 3**

## **Java Comments**

**The** [**Java**](https://www.javatpoint.com/java-tutorial) **comments are the statements in a program that are not executed by the compiler and interpreter.**

**Why do we use comments in a code?**

* Comments are used to make the program more readable by adding the details of the code.
* It makes easy to maintain the code and to find the errors easily.
* The comments can be used to provide information or explanation about the [variable](https://www.javatpoint.com/java-variables), method, [class](https://www.javatpoint.com/object-and-class-in-java), or any statement.
* It can also be used to prevent the execution of program code while testing the alternative code.

**Types of Java Comments**

There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

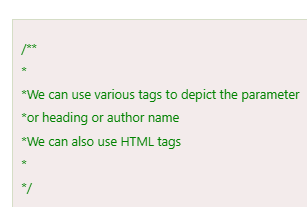
**Syntax:**

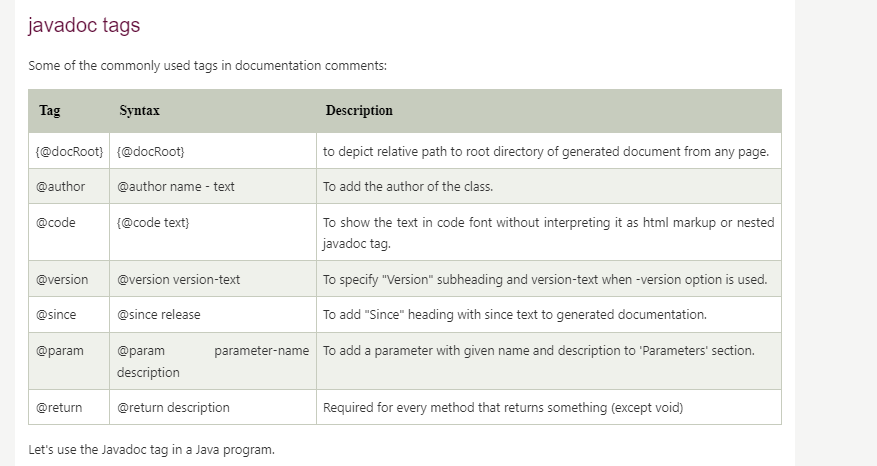
1. // → This is single line comment
2. 

3. Documentation comment : Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.

To create documentation API, we need to use the [**javadoc tool**](https://www.javatpoint.com/creating-api-document). The documentation comments are placed between /\*\* and \*/.

Syntax:





Example :

