# Java Variables

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

#### 1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

#### 2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called an instance variable because its value is instance-specific and is not shared among instances.

#### 3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

|  | data is instance variable.it cannot be static.  m is a static variable and belongs to the object of a class.  N is the local variable accessible inside the method and it cannot be static. |
| --- | --- |

### Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

### **Important java keywords**

1. [**abstract**](https://www.javatpoint.com/abstract-keyword-in-java)**:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. [**break**](https://www.javatpoint.com/java-break)**:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
3. [**continue**](https://www.javatpoint.com/java-continue)**:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
4. [**enum**](https://www.javatpoint.com/enum-in-java)**:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
5. [**final**](https://www.javatpoint.com/final-keyword)**:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
6. [**private**](https://www.javatpoint.com/private-keyword-in-java)**:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
7. [**protected**](https://www.javatpoint.com/protected-keyword-in-java)**:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
8. [**public**](https://www.javatpoint.com/public-keyword-in-java)**:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
9. [**return**](https://www.javatpoint.com/return-keyword-in-java)**:** Java return keyword is used to return from a method when its execution is complete.
10. [**static**](https://www.javatpoint.com/static-keyword-in-java)**:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
11. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

**FLOW CONTROL IN JAVAA**

| **public class Student {**  **public static void main(String[] args) {**  **String city = "Delhi";**  **if(city == "Delhi") {**  **System.*out*.println("city is meerut");**  **return;**  **}else if (city == "Noida") {**  **System.*out*.println("city is noida");**  **}else if(city == "Agra") {**  **System.*out*.println("city is agra");**  **}else {**  **System.*out*.println(city);**  **}**  **System.*out*.println("hiii");**  **}**  **}** |
| --- |

In the above program first we come in main →then if city==delhi then go inside loop and then print the sopln.

Return will make you go out of the method means the rest of lines will not be executed

public class Student {

public static void main(String[] args) {

String city = "Delhi";

if(city == "Delhi") {

System.*out*.println("city is meerut");

}else if (city == "Noida") {

System.*out*.println("city is noida");

}else if(city == "Agra") {

System.*out*.println("city is agra");

}else {

System.*out*.println(city);

}

System.*out*.println("hiii");

}

}

When there is no return then the program will move in if condition then execute rest of code below it and then go out of the method.

### 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.

### Java break statement

As the name suggests, 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

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* statement is used to break loop or [switch](https://www.javatpoint.com/java-switch) statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as [for loop](https://www.javatpoint.com/java-for-loop), [while loop](https://www.javatpoint.com/java-while-loop) and [do-while loop](https://www.javatpoint.com/java-do-while-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;

}

}

System.*out*.println("hello");

}

}

Here when the value of the i==6 then we will move out of the loop and eit the loop immediately.

### Java 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.

Example

public class ContinueExample {

public static void main(String[] args) {

// *TODO Auto-generated method stub*

for(int i = 0; i<= 8; i++) {

if(i == 4) {

continue;

}

System.*out*.println(i);

}

}

}

Here when i==4 then the rest below aprt of loop will no tbe executed the control will go above to the loop statement.

JAVA  **SWITCH STATEMENT**

# Java Switch Statement

**The Java *switch statement* executes one statement from multiple conditions. It is like** [**if-else-if**](https://www.javatpoint.com/java-if-else) **ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use** [**strings**](https://www.javatpoint.com/java-string) **in the switch statement**

Example

public class SwitchExample {

public static void main(String[] args) {

//Declaring a variable for switch expression

int number=20;

//Switch expression

switch(number){

//Case statements

case 10: System.*out*.println("10");

break;

case 20: System.*out*.println("20");

break;

case 30: System.*out*.println("30");

break;

//Default case statement

default:System.*out*.println("Not in 10, 20 or 30");

}

}

}

Now since the case is 20 so it will go in case 20 execute the lines in tha case and then break from there.

## Java Switch Statement with String

Java allows us to use strings in switch expression since Java SE 7. The case statement should be string literal.

**Example:**

public class SwitchStringExample {

public static void main(String[] args) {

//Declaring String variable

String levelString="Expert";

int level=0;

//Using String in Switch expression

switch(levelString){

//Using String Literal in Switch case

case "Beginner": level=1;

break;

case "Intermediate": level=2;

break;

case "Expert": level=3;

break;

default: level=0;

break;

}

System.*out*.println("Your Level is: "+level);

}

}

ENUMS

An enum in Java is a special data type that represents a group of constants (unchangeable variables, like final variables). Using enums can make your code more readable and less error-prone by providing a predefined set of values that a variable can have.

### **Why Use Enums**

1. **Type Safety:** Enums provide compile-time checking, ensuring that the variable can only take one of the predefined values.
2. **Readability:** Enums make the code more readable and understandable by providing meaningful names for a set of related values.
3. **Maintainability:** Enums help in managing related constants in a centralized way, making it easier to update and maintain.

### **Example Usage**

Let's consider a real-world scenario where enums can be useful: representing the days of the week.

### **Defining an Enum**

We make an enum as below:

public enum Day {

*SUNDAY*,

*MONDAY*,

*TUESDAY*,

*WEDNESDAY*,

*THURSDAY*,

*FRIDAY*,

*SATURDAY*

}

Now we use it in our code as below:

public class EnumExample1 {

public static void printDayMessage(Day day) {

switch (day) {

case *MONDAY*:

System.*out*.println("Mondays are tough!");

break;

case *FRIDAY*:

System.*out*.println("Fridays are great!");

break;

case *SUNDAY*:

System.*out*.println("Sundays are relaxing.");

break;

default:

System.*out*.println("Midweek days are busy.");

break;

}

}

public static void main(String[] args) {

Day today = Day.*MONDAY*;

*printDayMessage*(today);

// Iterating over all enum values

for (Day day : Day.*values*()) {

System.*out*.println(day);

}

}

}

In the above the method public static void printDayMessage(Day day) {  
Will ensure that the valid value is passed to the function which is of type day only.

### **Issues with Not Using Enum**

1. **Type Safety:**
   * Without enum, you lose the compile-time type checking. Any string or integer could be passed, increasing the risk of errors.
2. **Readability:**
   * Code using strings or integers to represent days is less readable and less intuitive.
3. **Maintainability:**
   * Managing related constants becomes harder as the values are spread throughout the codebase, making it difficult to update and maintain.