**UNIT-I: Introduction to Java:**

1. Introduction to Object Oriented Paradigm
2. Concepts of OOP, Applications of OOP
3. History of Java, Java Features
4. JVM
5. Program Structure
6. Variables, Primitive Data Types, Constants
7. String class
8. Primitive type conversion and Casting
9. Control Structures.

**1.Introduction to Object Oriented Paradigm**

The word object-oriented is the combination of two words i.e. object and oriented. The dictionary meaning of the object is an article or entity that exists in the real world. The meaning of oriented is interested in a particular kind of thing or entity. In layman's terms, it is a programming pattern that rounds around an object or entity are called object-oriented programming.

**2.Concepts of OOP, Applications of OOP**

**Object**



Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

**Class**

*Collection of objects* is called class. It is a logical entity.A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

**Inheritance**

*When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



**Polymorphism**

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.In Java, we use method overloading and method overriding to achieve polymorphism.Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

**Abstraction**

*Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.In Java, we use abstract class and interface to achieve abstraction.



**Encapsulation**

*Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines.A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Advantage of OOPs over Procedure-oriented programming language**

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

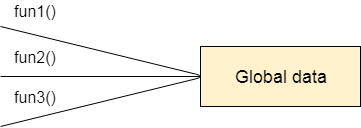


Figure: Data Representation in Procedure-Oriented Programming

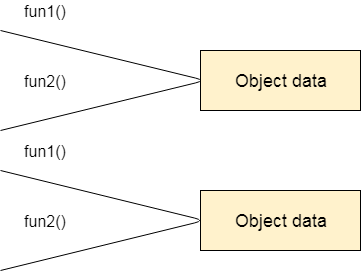


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

**3. History of Java, Java Features:**

# History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". [Java](https://www.javatpoint.com/java-tutorial) was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe the history of Java.

[**James Gosling**](https://www.javatpoint.com/james-gosling-father-of-java)**, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



**1. Simple:**

* Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:
* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

**2. Object-oriented**

* Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.
* Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

**3. Platform Independent**



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

* Runtime Environment
* API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

**4.Secured**

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

* No explicit pointer
* Java Programs run inside a virtual machine sandbox



* **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
* **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
* **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

**5.Robust**

The English mining of Robust is strong. Java is robust because:

* It uses strong memory management.
* There is a lack of pointers that avoids security problems.
* Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
* There are exception handling and the type checking mechanism in Java. All these points make Java robust.

**6.Architecture-neutral**

* Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
* In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

**7.Portable**

* Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

**8.High-performance**

* Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
* It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

**9. Distributed**

* Java is distributed because it facilitates users to create distributed applications in Java.
* RMI and EJB are used for creating distributed applications.
* This feature of Java makes us able to access files by calling the methods from any machine on the internet.

1. **Multi-threaded**

* A thread is like a separate program, executing concurrently.
* We can write Java programs that deal with many tasks at once by defining multiple threads.
* The main advantage of multi-threading is that it doesn't occupy memory for each thread.
* It shares a common memory area. Threads are important for multi-media, Web applications, etc.

1. **Dynamic**

* Java is a dynamic language. It supports the dynamic loading of classes.
* It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

**4.Java Virtual Machine (JVM)** is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Runtime Environment (JRE). In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

**Working of Java Virtual Machine (JVM)**

First, Java code is compiled into bytecode. This bytecode gets interpreted on different machines

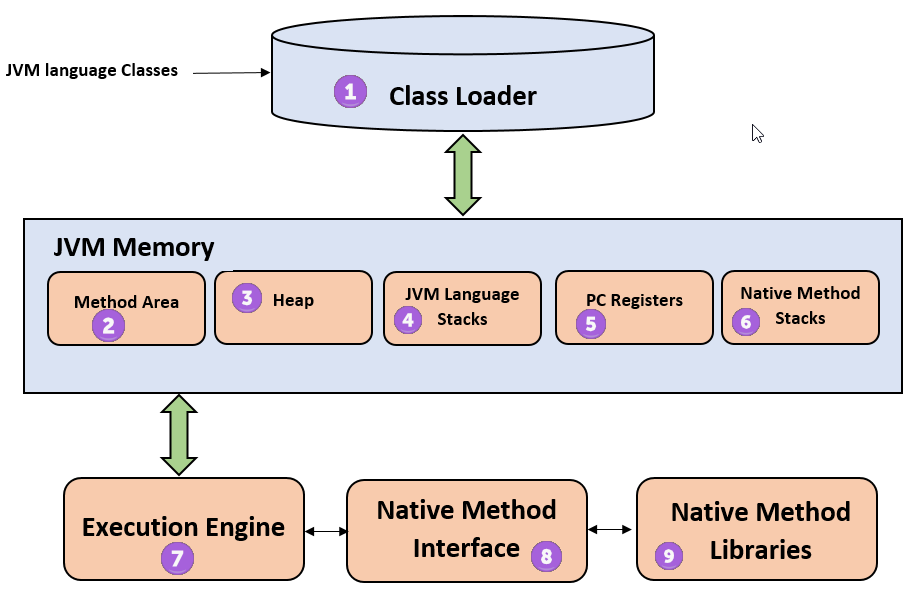
Between host system and Java source, Bytecode is an intermediary language.

JVM in Java is responsible for allocating memory space.



## JVM Architecture

Now in this JVM tutorial, let’s understand the Architecture of JVM. JVM architecture in Java contains classloader, memory area, execution engine etc.



**Java Virtual Machine Architecture**

**1) ClassLoader**

The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

**2) Method Area**

JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.

**3) Heap**

All the [Objects](https://www.guru99.com/java-oops-class-objects.html), their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

**4) JVM language Stacks**

Java language Stacks store local variables, and it’s partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.

**5) PC Registers**

PC register store the address of the Java virtual machine instruction which is currently executing. In Java, each thread has its separate PC register.

**6) Native Method Stacks**

Native method stacks hold the instruction of native code depends on the native library. It is written in another language instead of Java.

**7) Execution Engine**

It is a type of software used to test hardware, software, or complete systems. The test execution engine never carries any information about the tested product.

**8) Native Method interface**

The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.

**9) Native Method Libraries**

Native Libraries is a collection of the Native Libraries(C, C++) which are needed by the Execution Engine.

**Software Code Compilation & Execution process**

In order to write and execute a software program, you need the following

**1) Editor**– To type your program into, a notepad could be used for this

**2) Compiler**– To convert your high language program into native machine code

**3) Linker**– To combine different program files reference in your main program together.

**4) Loader**– To load the files from your secondary storage device like Hard Disk, Flash Drive, CD into RAM for execution. The loading is automatically done when you execute your code.

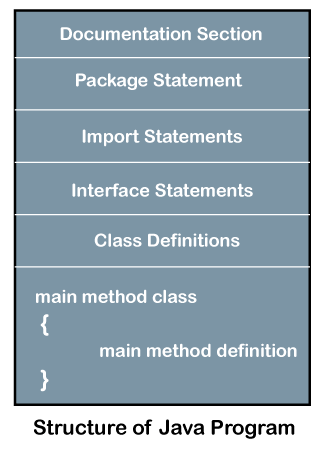
**5) Execution** – Actual execution of the code which is handled by your OS & processor.

Java is called platform independent because of Java Virtual Machine. As different computers with the different operating system have their JVM, when we submit a .class file to any operating system, JVM interprets the bytecode into machine level language.

* JVM is the main component of Java architecture, and it is the part of the [JRE (Java Runtime Environment)](https://www.w3schools.in/java/jre).
* A program of JVM is written in [C Programming Language](https://www.w3schools.in/c-programming), and JVM is Operating System dependent.
* JVM is responsible for allocating the necessary memory needed by the Java program.
* JVM is responsible for deallocating memory space.

**5. Program Structure**

Java is an object-oriented programming, **platform- independent,** and **secure** programming language that makes it popular.



Let's see which elements are included in the structure of a [Java program](https://www.javatpoint.com/java-programs). A typical structure of a [Java](https://www.javatpoint.com/java-tutorial) program contains the following elements:

* Documentation Section
* Package Declaration
* Import Statements
* Interface Section
* Class Definition
* Class Variables and Variables
* Main Method Class
* Methods and Behaviors

## Documentation Section

The documentation section is an important section but optional for a Java program. It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line,** and **documentation** comments.

* **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:

//First Java Program

**Multi-line Comment:** It starts with a **/\*** and ends with **\*/.** We write between these two symbols. For example:

/\*It is an example of

multiline comment\*/

**Documentation Comment:** It starts with the delimiter **(/\*\*)** and ends with **\*/**. For example:

/\*\*It is an example of documentation comment\*/

## Package Declaration

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the **package name** in which the class is placed. Note that there can be **only one package** statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory. We use the keyword **package** to declare the package name. For example:

**package** javaivsem; //where javaivsem is the package name

**package** com.javaivsem; //where com is the root directory and javaivsem is the subdirectory

## Import Statements

The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement. We use the import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements. For example:

**import** java.util.Scanner; //it imports the Scanner class only

**import** java.util.\*; //it imports all the class of the java.util package

## Interface Section

It is an optional section. We can create an **interface** in this section if required. We use the **interface** keyword to create an interface. An [interface](https://www.javatpoint.com/interface-in-java) is a slightly different from the class. It contains only **constants** and **method** declarations. Another difference is that it cannot be instantiated. We can use interface in classes by using the **implements** keyword. An interface can also be used with other interfaces by using the **extends** keyword. For example:

**interface** car

{

**void** start();

**void** stop();

}

## Class Definition

In this section, we define the class. It is **vital** part of a Java program. Without the [class](https://www.javatpoint.com/object-and-class-in-java), we cannot create any Java program. A Java program may conation more than one class definition. We use the **class** keyword to define the class. The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method. For example:

class Student //class definition

{

}

## Class Variables and Constants

In a Java program, the variables and constants are defined just after the class definition. The variables and constants store values of the parameters. It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

**class** Student //class definition

{

String sname;  //variable

**int** id;

**double** percentage;

}

## Main Method Class

In this section, we define the **main() method.** It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

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

{

}

For example:

**public** **class** Student //class definition

{

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

{

//statements

}

}

## Methods and behavior

The methods are the set of instructions that we want to perform. These instructions execute at runtime and perform the specified task. For example:

**public** **class** Demo //class definition

{

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

{

**void** display()

{

System.out.println("Welcome to java");

}

//statements

}

}

**6.Variables, Primitive Data Types, Constants**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

**Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

**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.

**Example to understand the types of variables in java**

public class A

{

static int m=100;//static variable

void method()

{

int n=90;//local variable

}

public static void main(String args[])

{

int data=50;//instance variable

}

}//end of class

**Constant:** In Java, to declare any variable as constant, we use static and final modifiers. It is also known as non-access modifiers. According to the Java naming convention the identifier name must be in capital letters.

**Static and Final Modifiers**

* The purpose to use the static modifier is to manage the memory.
* It also allows the variable to be available without loading any instance of the class in which it is defined.
* The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.

**The syntax to declare a constant is as follows:**

static final datatype identifier\_name=value;

**For example, price is a variable that we want to make constant.**

static final double PRICE=432.78;

# Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

## Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in [Java language](https://www.javatpoint.com/java-tutorial).

Java is a statically-typed programming language. It means, all [variables](https://www.javatpoint.com/java-variables) must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

* boolean data type
* byte data type
* char data type
* short data type
* int data type
* long data type
* float data type
* double data type



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

**Why char uses 2 byte in java and what is \u0000 ?**

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

**7.String class**

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

char[] ch={'j','a','v','a' };

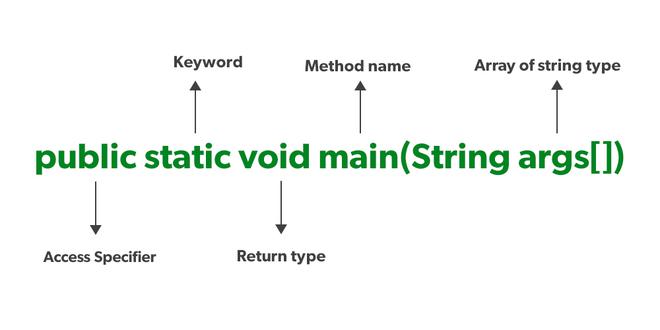
String s=new String(ch);

is same as:

String s="java";

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

**Main Method:**



1. **Public :** It is an Access modifier, which specifies from where and who can access the method. Making the main() method public makes it globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.
2. **Static:** It is a keyword that is when associated with a method, making it a class-related method. The main() method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the main() method by the JVM.
3. **Void :** It is a keyword and is used to specify that a method doesn’t return anything. As the main() method doesn’t return anything, its return type is void. As soon as the main() method terminates, the java program terminates too. Hence, it doesn’t make any sense to return from the main() method as JVM can’t do anything with the return value of it.
4. **Main:** It is the name of the Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It’s not a keyword.
5. **String[] args  :** It stores Java command-line arguments and is an array of type java.lang.String class. Here, the name of the String array is args but it is not fixed and the user can use any name in place of it.

**8.Java Type Casting**

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

**Widening Casting (automatically)** - converting a smaller type to a larger type size

byte -> short -> char -> int -> long -> float -> double

**Narrowing Casting (manually)** - converting a larger type to a smaller size type

double -> float -> long -> int -> char -> short -> byte

**Widening Casting**

Widening casting is done automatically when passing a smaller size type to a larger size type:

**Example:**

public class Main {

public static void main(String[] args) {

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

}

}

**Narrowing Casting**

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

**Example**

public class Main {

public static void main(String[] args) {

double myDouble = 9.78d;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

}

}

**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 provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

Decision Making statements

* if statements
* switch statement

Loop statements

* do while loop
* while loop
* for loop
* for-each loop

Jump statements

* break statement
* continue statement

**Decision-Making statements:**

As the name suggests, 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. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

* Simple if statement
* if-else statement
* if-else-if ladder
* Nested if-statement

Let's understand the if-statements one by one.

**1) Simple if statement**:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

if(condition) {

statement 1; //executes when condition is true

}

class Student {

public static void main(String[] args) {

int x = 10;

int y = 12;

if(x+y > 20) {

System.out.println("x + y is greater than 20");

}

}

}

Output:

x + y is greater than 20

**2) if-else statement**

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

if(condition) {

statement 1; //executes when condition is true

}

else{

statement 2; //executes when condition is false

}

class Student {

public static void main(String[] args) {

int x = 10;

int y = 12;

if(x+y < 10) {

System.out.println("x + y is less than 10");

} else {

System.out.println("x + y is greater than 20");

}

}

}

Output:

x + y is greater than 20

**3) if-else-if ladder:**

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

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

}

class Student {

public static void main(String[] args) {

String city = "Delhi";

if(city == "Meerut") {

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);

}

}

}

Output:

Delhi

**4. Nested if-statement**

In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

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

}

}

class Student {

public static void main(String[] args) {

String address = "Delhi, India";

if(address.endsWith("India")) {

if(address.contains("Meerut")) {

System.out.println("Your city is Meerut");

}else if(address.contains("Noida")) {

System.out.println("Your city is Noida");

}else {

System.out.println(address.split(",")[0]);

}

}else {

System.out.println("You are not living in India");

}

}

}

Output:

Delhi

**Switch Statement:**

In Java, Switch statements 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. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

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
* 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.

The syntax to use the switch statement is given below.

switch (expression){

case value1:

statement1;

break;

.

.

.

case valueN:

statementN;

break;

default:

default statement;

}

public class Student implements Cloneable {

public static void main(String[] args) {

int num = 2;

switch (num){

case 0:

System.out.println("number is 0");

break;

case 1:

System.out.println("number is 1");

break;

default:

System.out.println(num);

}

}

}

Output:

2

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.

**Loop Statements**

In programming, 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.

* for loop
* while loop
* do-while loop

**Java for loop**

In Java, for loop is similar to C and C++. 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.

for(initialization, condition, increment/decrement) {

//block of statements

}

class Calculattion {

public static void main(String[] args) {

// TODO Auto-generated method stub

int sum = 0;

for(int j = 1; j<=10; j++) {

sum = sum + j;

}

System.out.println("The sum of first 10 natural numbers is " + sum);

}

}

Output:

The sum of first 10 natural numbers is 55

**Java 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

}

public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

String[] names = {"Java","C","C++","Python","JavaScript"};

System.out.println("Printing the content of the array names:\n");

for(String name:names) {

System.out.println(name);

}

}

}

Output:

Printing the content of the array names:

Java

C

C++

Python

JavaScript

**Java while loop**

The 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

}

class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

while(i<=10) {

System.out.println(i);

i = i + 2;

}

}

}

Output:

Printing the list of first 10 even numbers

0

2

4

6

8

10

**Java do-while loop**

The 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);

class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

do {

System.out.println(i);

i = i + 2;

}while(i<=10);

}

}

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.

**Java break statement**

As the name suggests, the break statement 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.

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

class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

a:

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

b:

for(int j = 0; j<=15;j++) {

c:

for (int k = 0; k<=20; k++) {

System.out.println(k);

if(k==5) {

break a;

}

}

}

}

}

}

Output:

0

1

2

3

4

5

**Java continue statement**

Unlike break statement, the continue statement 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.

public class ContinueExample {

public static void main(String[] args) {

// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {

continue;

}

System.out.println(j);

}

}

}

}

Output:

0

1

2

3

5

1

2

3

5

2

3

5

### Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | [char charAt(int index)](https://www.javatpoint.com/java-string-charat) | It returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | It returns string length |
| 3 | [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns a formatted string. |
| 4 | [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns formatted string with given locale. |
| 5 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index. |
| 6 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index and end index. |
| 7 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | It returns true or false after matching the sequence of char value. |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 10 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | It checks the equality of string with the given object. |
| 11 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | It checks if string is empty. |
| 12 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | It concatenates the specified string. |
| 13 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified char value. |
| 14 | [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified CharSequence. |
| 15 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | It compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex. |
| 17 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex and limit. |
| 18 | [String intern()](https://www.javatpoint.com/java-string-intern) | It returns an interned string. |
| 19 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index. |
| 20 | [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index starting with given index. |
| 21 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index. |
| 22 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index starting with given index. |
| 23 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase. |
| 24 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase using specified locale. |
| 25 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase. |
| 26 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase using specified locale. |
| 27 | [String trim()](https://www.javatpoint.com/java-string-trim) | It removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | It converts given type into string. It is an overloaded method. |

**Java String compare**

We can compare String in Java on the basis of content and reference.

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

**1) By Using equals() Method**

The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

Play Video

* public boolean equals(Object another) compares this string to the specified object.
* public boolean equalsIgnoreCase(String another) compares this string to another string, ignoring case.

class Teststringcomparison1{

public static void main(String args[]){

String s1="Sachin";

String s2="Sachin";

String s3=new String("Sachin");

String s4="Saurav";

System.out.println(s1.equals(s2));//true

System.out.println(s1.equals(s3));//true

System.out.println(s1.equals(s4));//false

}

}

**Output:**

true

true

false

In the above code, two strings are compared using equals() method of String class. And the result is printed as boolean values, true or false.

Teststringcomparison2.java

class Teststringcomparison2{

public static void main(String args[]){

String s1="Sachin";

String s2="SACHIN";

System.out.println(s1.equals(s2));//false

System.out.println(s1.equalsIgnoreCase(s2));//true

}

}

**Output:**

false

true

**2) By Using == operator:**

The == operator compares references not values.

class Teststringcomparison3{

public static void main(String args[]){

String s1="Sachin";

String s2="Sachin";

String s3=new String("Sachin");

System.out.println(s1==s2);//true (because both refer to same instance)

System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)

}

}

**Output:**

true

false

**3) By Using compareTo() method:**

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two String objects. If:

s1 == s2 : The method returns 0.

s1 > s2 : The method returns a positive value.

s1 < s2 : The method returns a negative value.

class Teststringcomparison4{

public static void main(String args[]){

String s1="Sachin";

String s2="Sachin";

String s3="Ratan";

System.out.println(s1.compareTo(s2));//0

System.out.println(s1.compareTo(s3));//1(because s1>s3)

System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

}

}

**Output:**

0

1

-1