OBJECT ORIENTED PROGRAMMING

Unit 1

Chapter 1 – Introduction

The history of Java

Java was invented by James Gosling and his team at Sun Microsystems, in 1991.

Java is related to C++, which is inherited from the language C.

It took approx. Eighteen months to develop the first working version. It was first named as "Oak" but was renamed as "Java" in 1995. It is publicly announced in the spring of 1995.

The basic idea behind creating this language is to create a platform-independent language that is used to develop software for consumer electronic devices such as microwave ovens, remote controls, etc. Initially, it was not designed for Internet applications.

Other languages have the problem that they are designed to compile the code for a specific platform. Let us take the example of C++, it is possible to compile C++ code for any processor but to do so it requires a full C++ compiler targeted for that particular processor and platform. That makes it expensive and time-consuming. To overcome this, Gosling and others started working on a portable and platform-independent language, this leads to the creation of Java.

Java had an extreme effect on the Internet by the innovation of a new type of networked program called the Applet. An applet is a Java program that is designed to be transmitted over the internet and executed by the web browser that is Java-compatible. Applets are the small program that is used to display data provided by the server, handle user input, provide a simple function such as calculator etc.

Java solves the Security and the portability issue of the other language that is being used. The key that allows doing so is the Bytecode. Bytecode is a highly optimized set of instruction that is designed to be executed by the Java Virtual Machine (JVM). Java programs are executed by the JVM also helps to make Java a secure programming g language because the JVM contains the application and prevents it from affecting the external systems.

Evolution of Java

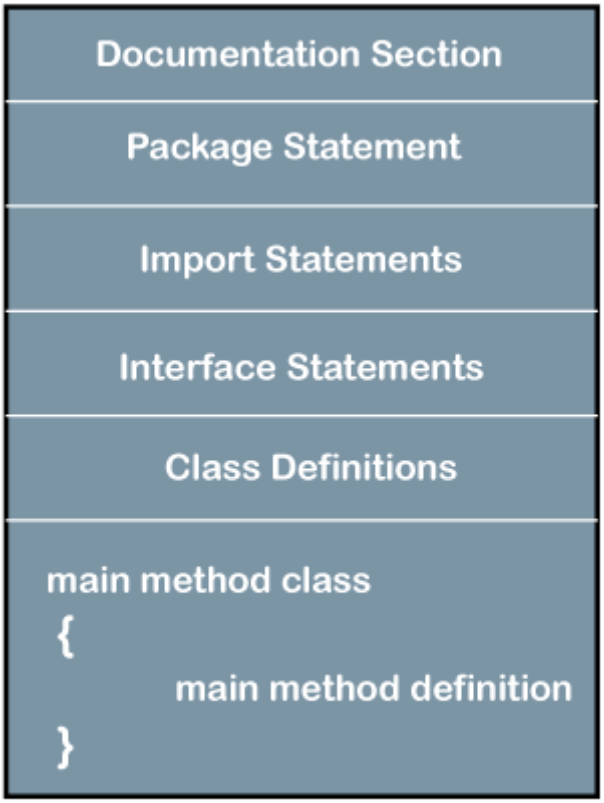
The versions of java are:

Version	Release Date	Features
JDK Beta	1995	—
JDK 1.0	January 1996	This is the first stable version.
JDK 1.1	February 1997	In this version, added many new library elements, redefined the way of handling events, and reconfigured most of the libraries of 1.0 and deprecated some features defined by 1.0. Added inner class, JavaBeans, JDBC, RMI, JIT (Just In time) compiler.
J2SE 1.2	December 1998	Added support for many features, such as Swing and Collection Framework. The methods suspend(), resume() and stop() of Thread class were deprecated.
J2SE 1.3	May 2000	A very small improvement, as it just improved the development environment.
J2SE 1.4	February 2002	It added some upgrades such as the new keyword assert, chained exception and a channel-based I/O subsystem. Also added some feature to the collection framework and the Networking classes.
J2SE 5.0	September 2004	The significant new features added to this version are – Generics, Annotation, Autoboxing and Auto-unboxing, Enumeration, for-

		each, variable-length argument, Static import, Formatted I/O, Concurrency utilities.
Java SE 6	December 2006	In this version the API libraries and several new packages got enhanced and offered improvements to the run time. It supports JDBC 4.0.
Java SE 7	July 2011	Added JVM support for dynamic language, String in the switch, Automatic resource management in try-statement, support for underscore in integers, binary integer literals etc…
Java SE 8	March 2014	Added Date and time API, Repeating annotation, JavaFX.
Java SE 9	September 2017	Added Java platform module system update, jshell, XML Catalog, jlink, and the JavaDB was removed from JDK
Java SE 10	March 2018	Added features are local variable type interface, Application class data sharing, Garbage collector interface, etc…
Java SE 11	September 2018	Feature added: Dynamic class file loader, HTTP client, and Transport layer security. JavaFX, Java EE, and CORBA modules have been removed from JDK.
Java SE 12	March 2019	Added Microbenchmark Suite, JVM Constant API, One AArch64 Port, Default CDS Archives etc.

Structure of Java Program

Java is an object-oriented programming, platform-independent, and secure programming language that makes it popular. Using the Java programming language, we can develop a wide variety of applications. So, before diving in depth, it is necessary to understand the basic structure of Java program in detail. In this section, we have discussed the basic structure of a Java program. At the end of this section, you will able to develop the Hello world Java program, easily.



Structure of Java Program

Path Setting

Z :\> path = C:\Program Files (x86)\Java\jdk1.6.0_10\bin

Z:\>javac

Or

Z :\> path= C:\Program Files \Java\jdk1.6.0_10\bin

Z:\>javac

Examples of simple java program

```
class example1
{
public static void main (String [] args)
{
System.out.println("Hello World");
}
}
```

Z:\ javac example1.java (Compiled by java file)

Z:\java example1 (Run by class file)

Output: Hello World

```
class example2
{
public static void main (String [] args)
{
System.out.println("Welcome to java");
}
}
```

Z:\ javac example2.java(Compiled by java file)

Z:\java example2 (Run by class file)

Output: Welcome to java

Examples 3:

```
class A
{
int a=10;
int b=20;
```

```
void method1()
{
System.out.println(a);
}
void method2()
{
System.out.println(b);
}
public static void main(String args[])
{
A m1= new A();
m1.method1();
A m2= new A();
m2.method2();
}
}
Z:\ javac A.java
Z:\java A
```

Output :

10

20

Examples 4:

```
class A1
{
int a=10;
int b=20;
void method1()
{
System.out.println(a);
```

```
}
void method2()
{
System.out.println(b);
}
}
class A2
{
public static void main(String args[])
{
A1 m1= new A1();
m1.method1();

A1 m2= new A1();
m2.method2();
}
}
```

```
Z:\ javac A2.java
Z:\java A2
```

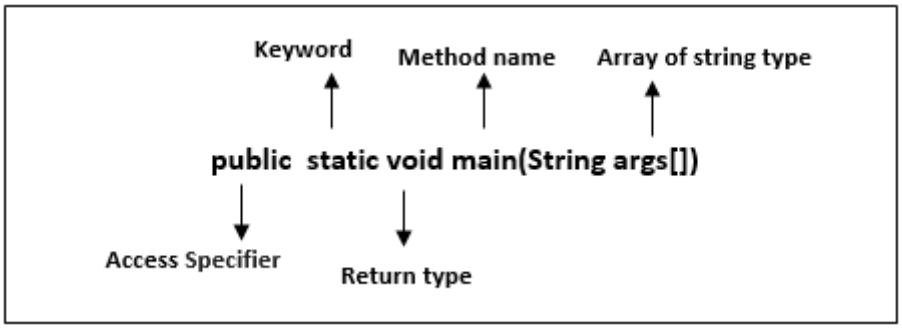
Output :

10

20

Java main () method

The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:



public: It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

static: You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

void: In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

main(): It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

String args[]: The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.

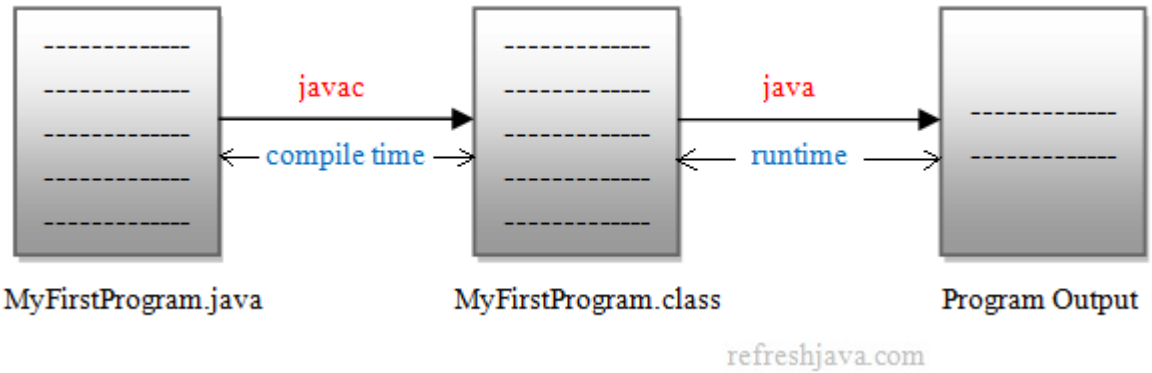
main(String args[])

Here, agrs[] is the array name, and it is of String type. It means that it can store a group of string. Remember, this array can also store a group of numbers but in the form of string only. Values passed to the main() method is called arguments. These arguments are stored into args[] array, so the name args[] is generally used for it.

Compile time and runtime in Java

Runtime and compile time, these are two programming terms that are more frequently used in java programming language. The programmers specially beginners find it little difficult to understand what exactly they are. So let's understand what these terms means in java with example.

In java running a program happens in two steps, compilation and then execution. The image below shows where does compile time and runtime takes place in execution of a program.



Compile time is a process in which java compiler compiles the java program and generates a .class file. In other way, in compile time java source code (.java file) is converted in to .class file using java compiler. While in runtime, the java virtual machine loads the .class file in memory and executes that class to generate the output of program.

What is JVM

- JVM, i.e., Java Virtual Machine.
- JVM is the engine that drives the Java code.
- Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine.
- When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform.
- Bytecode is an intermediary language between Java source and the host system.

- It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.
- **JVM's work can be explained in the following manner**
  - Reading Bytecode.
  - Verifying bytecode.
  - Linking the code with the library.
- **Diagram of JVM**

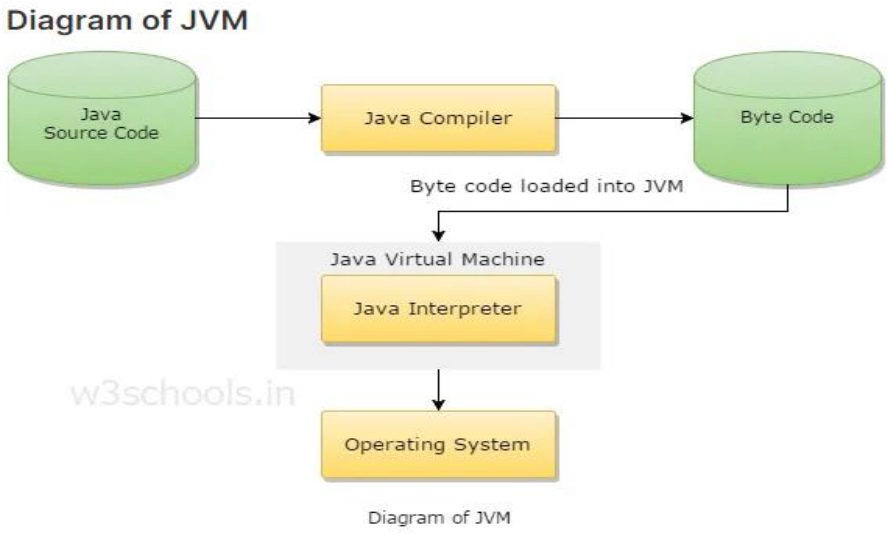


Diagram of JVM

- **NOTE :**Javac is the Java Compiler which Compiles Java code into Bytecode. JVM is Java Virtual Machine which Runs/ Interprets/ translates Bytecode into Native Machine Code.
- In Java though it is considered as an interpreted language, It may use JIT (Just-in-Time) compilation when the bytecode is in the JVM. The JIT compiler reads the bytecodes in many sections (or in full, rarely) and compiles them dynamically into machine code so the program can run faster, and then cached and reused later without needing to be recompiled. So JIT compilation combines the speed of compiled code with the flexibility of interpretation.
- **JVM is specifically responsible for converting bytecode to machine-specific code** and is necessary in both JDK and JRE. It is also platform-dependent and performs many functions, including memory management and security

Object-Oriented Programming

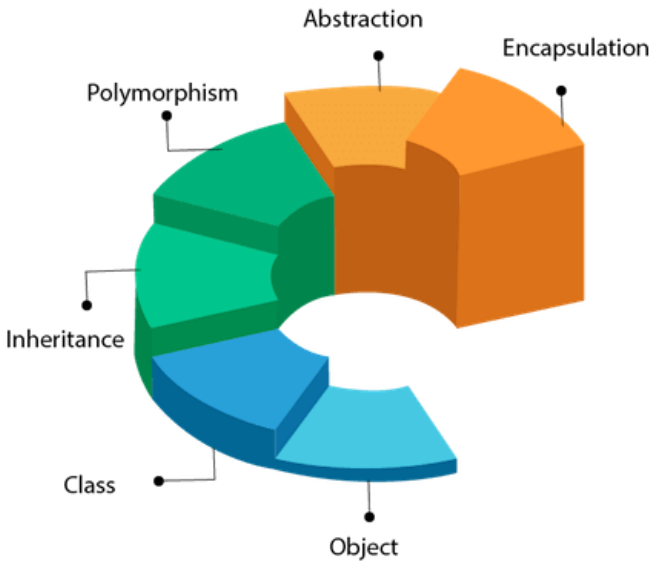
- OOP stands for Object-Oriented Programming.
- Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.
- **Object-oriented programming has several advantages over procedural programming:**
  - OOP is faster and easier to execute.
  - OOP provides a clear structure for the programs.
  - OOP makes the code easier to maintain, modify and debug.
  - OOP makes it possible to create full reusable applications with less code and shorter development time.
- The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System):

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

OOPs (Object-Oriented Programming System)



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

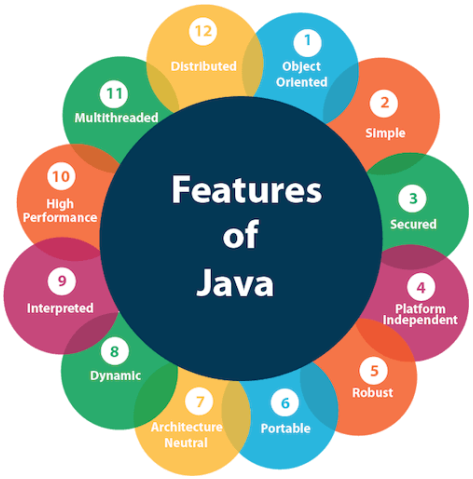
Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Java Buzz Words (OR) Features of Java

The features of Java are also known as Java buzzwords.A list of the most important features of the Java language is given below.

- Simple
- Platform Independent
- Architectural Neutral
- Dynamic
- Portable
- Multi-Threading
- Distributed
- Robust
- Secured
- High Performance
- Object Oriented



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.

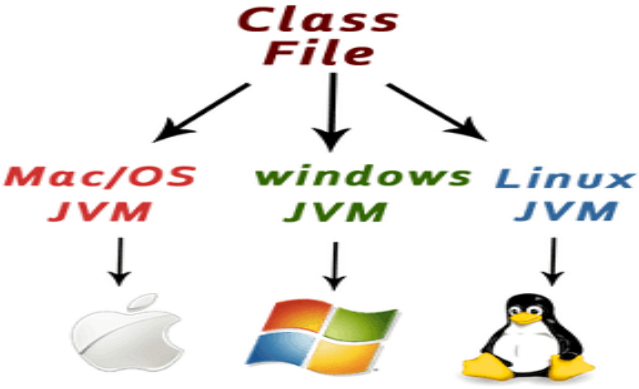
Object-oriented

Java is an object-orientedprogramming 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:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Platform Independent



Java is platform independent because it is different from other languages like C, C++, 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.

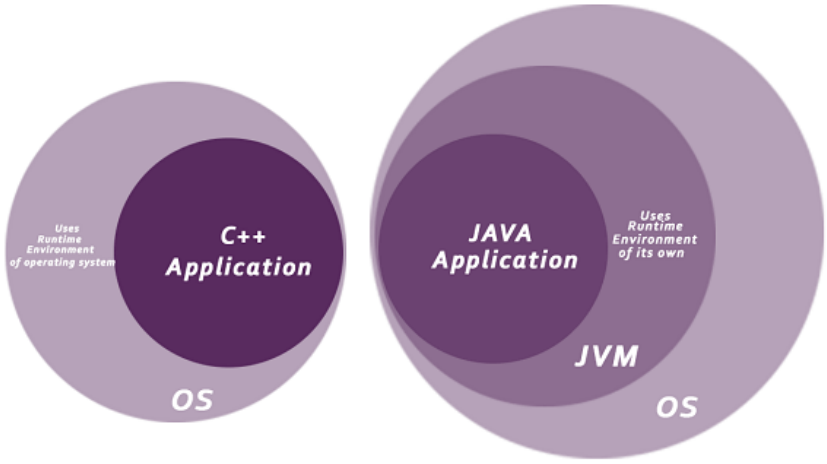
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. 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).

Secured

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



- Java Programs run inside a virtual machine sandbox.
- No explicit pointer.
- 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.

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

Robust

The English meaning 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.

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.

Portable

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

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.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI (Java **Remote Method Invocation)** and EJB (**enterprise java bean)** 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.

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.

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++

Definition of Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is the automatic process of reclaiming an unused memory in runtime.
- In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++.

What happens if the main() method is written without String args[]?

The program will compile, but not run, because JVM will not recognize the main() method. Remember JVM always looks for the main() method with a string type array as a parameter.

Execution Process

First, JVM executes the static block, then it executes static methods, and then it creates the object needed by the program. Finally, it executes the instance methods. JVM executes a static block on the highest priority basis. It means JVM first goes to static block even before it looks for the main() method in the program.

Example

```
class  Demo
{
static                //static block
{
System.out.println("Static block");
}
public static void main(String args[])  //static method
{
```

```
System.out.println("Static method");
}
}
```

Output:

```
Static block
Static method
```

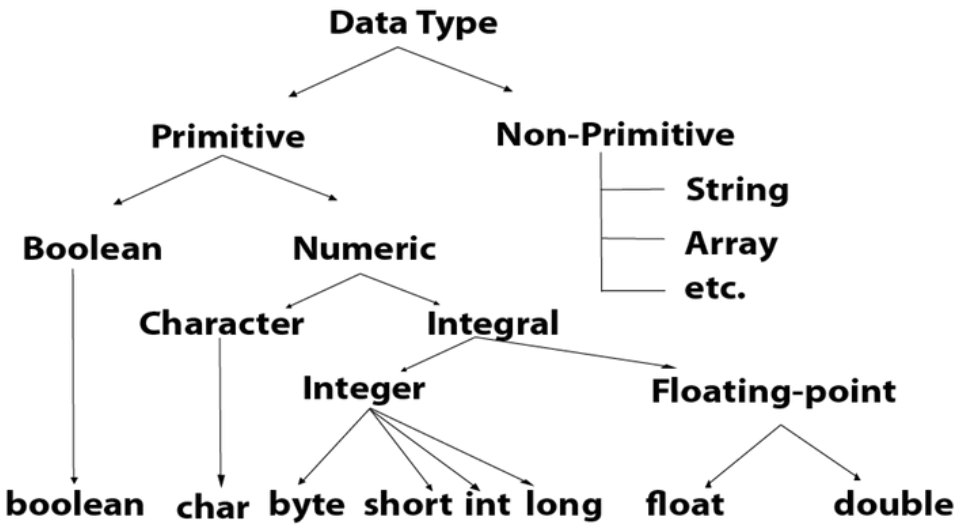
We observe that JVM first executes the static block, if it is present in the program. After that it searches for the main() method. If the main() method is not found, it gives error.

Data Types

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

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes,Interfaces,, and Arrays.



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.

Java is a statically-typed programming language. It means, all 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

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

Boolean one = false

or

Boolean one = true

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive).

Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807.

Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example:

char letter = 'A'

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.

Unicode System

- **Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.**
- **Java actually uses Unicode**, which includes ASCII and other characters from languages around the world.

Variables

A variable is a container which holds the value while the Java program is executed.

A variable is assigned with a data type.

Variable is a name of memory location.

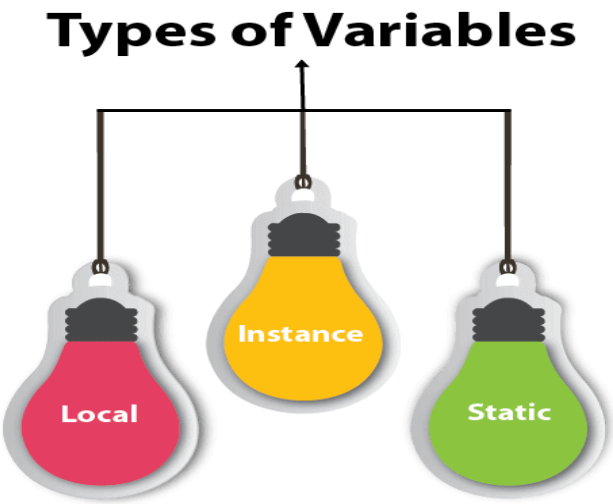
There are three types of variables in java: local, instance and static.

Example :int data=50;//Here data is variable

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable



Local Variables

- **Local variables are declared in methods, constructors, or blocks.**
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Instance Variables

- **Instance variables are declared in a class, but outside a method, constructor or any block.**
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null.

Class/Static Variables

- **Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.**

- Static variables are stored in the static memory.
- Static variables are created when the program starts and destroyed when the program stops.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null.

Example: Types of Variables in Java

```
class sample
{
static int a = 1; //static variable
int data = 99; //instance variable
void method()
{
int b = 90; //local variable
}
}
```

Operators

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<<>>>>
Relational	comparison	<><= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Control Statements

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.

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:

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.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. 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
}
```

Consider the following example in which we have used the **if** statement in the java code.

Student.java

Student.java

```
public 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
}
```

Consider the following example.

Student.java

```
public 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
}
```

Consider the following example.

Student.java

```
public 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
}
}
```

Consider the following example.

Student.java

```
public 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;
}
```

Consider the following example to understand the flow of the switch statement.

Student.java

```
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.

1. for loop
2. while loop
3. do-while loop

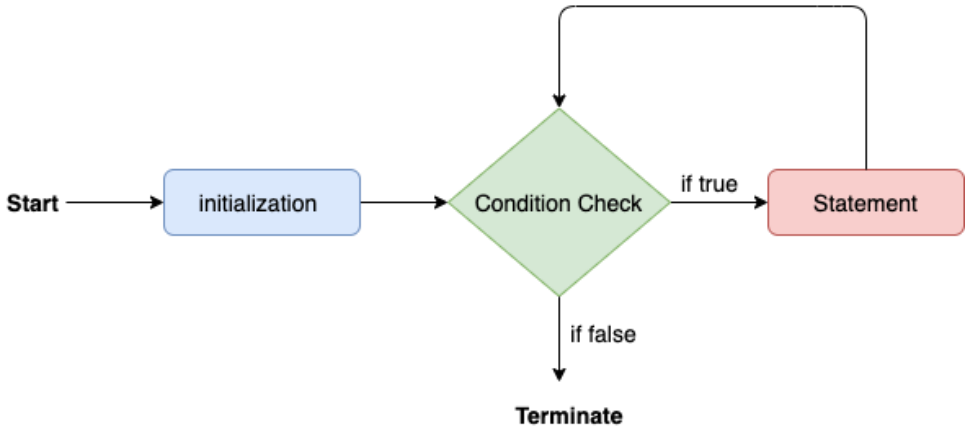
Let's understand the loop statements one by one.

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
}
```

The flow chart for the for-loop is given below.



Consider the following example to understand the proper functioning of the for loop in java.

Calculation.java

```
public 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.**

Example 1 :

```
for(data_type varname : array_name/collection_name)
{
//statements
}
```

Consider the following example to understand the functioning of the for-each loop in Java.

Calculation.java

```
class Calculation
{
public static void main(String[] args)
{
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
```

Example 2 :

for-each Loop Sytnax

The syntax of the Java **for-each** loop is:

```
for(dataType item : array) {
    ...
}
```

Here,

- **array** - an array or a collection
- **item** - each item of array/collection is assigned to this variable
- **dataType** - the data type of the array/collection

Example 1: Print Array Elements

```
// print array elements

class Main {
  public static void main(String[] args) {

    // create an array
    int[] numbers = {3, 9, 5, -5};

    // for each loop
    for (int number: numbers) {
      System.out.println(number);
    }
  }
}
```

Output

```
3
9
5
-5
```

Here, we have used the **for-each loop** to print each element of the `numbers` array one by one.

- In the first iteration, `item` will be 3.
- In the second iteration, `item` will be 9.
- In the third iteration, `item` will be 5.
- In the fourth iteration, `item` will be -5.

Example 2: Sum of Array Elements

```
// Calculate the sum of all elements of an array

class Main {
 public static void main(String[] args) {

   // an array of numbers
   int[] numbers = {3, 4, 5, -5, 0, 12};
   int sum = 0;

   // iterating through each element of the array
   for (int number: numbers) {
     sum += number;
   }

   System.out.println("Sum = " + sum);
 }
}
```

Output:

```
Sum = 19
```

Java while loop

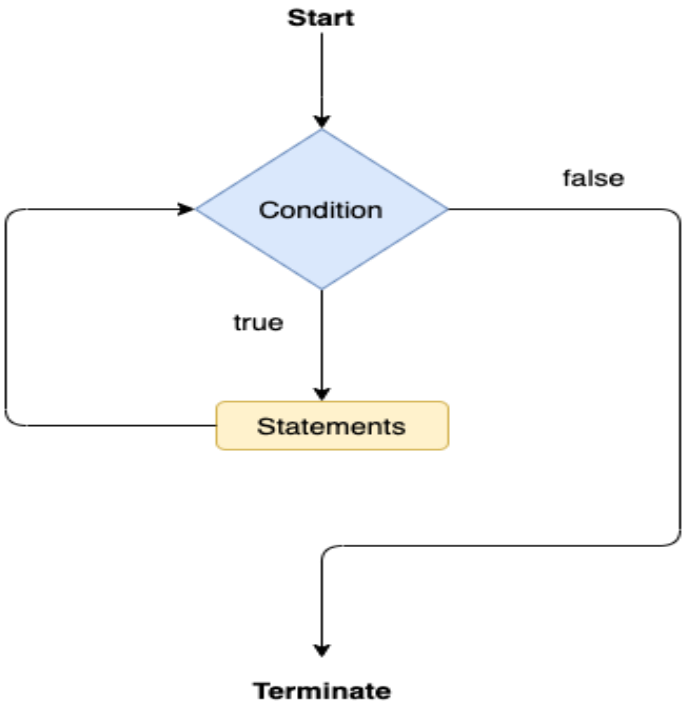
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while 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
}
```

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



Consider the following example.

Calculation .java

```
public 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

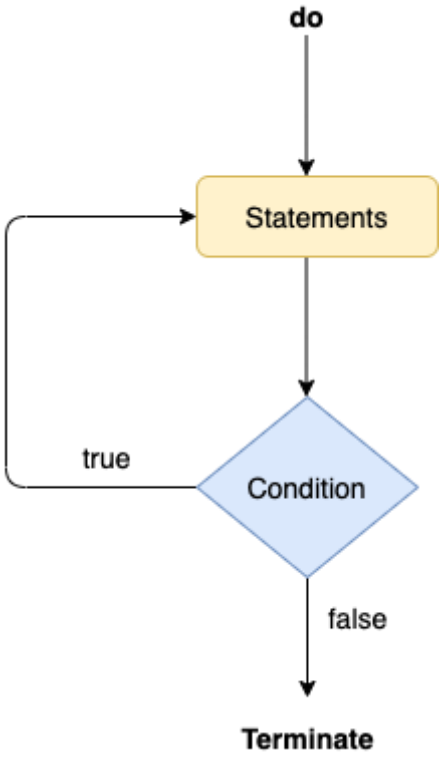
The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

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);
```

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

```
public 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.

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
```

break statement example with labeled for loop

Calculation.java

```
public 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 Keywords

Java has a set of keywords that are reserved words that cannot be used as variables, methods, classes, or any other identifiers:

Keyword	Description
abstract	A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)
assert	For debugging
boolean	A data type that can only store true and false values
break	Breaks out of a loop or a switch block
byte	A data type that can store whole numbers from -128 and 127

case	Marks a block of code in switch statements
catch	Catches exceptions generated by try statements
char	A data type that is used to store a single character
class	Defines a class
continue	Continues to the next iteration of a loop
const	Defines a constant. Not in use - use final instead
default	Specifies the default block of code in a switch statement
do	Used together with while to create a do-while loop
double	A data type that can store whole numbers from 1.7e−308 to 1.7e+308
else	Used in conditional statements
enum	Declares an enumerated (unchangeable) type
exports	Exports a package with a module. New in Java 9
extends	Extends a class (indicates that a class is inherited from another class)
final	A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override)
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
float	A data type that can store whole numbers from 3.4e−038 to 3.4e+038
for	Create a for loop

goto	Not in use, and has no function
if	Makes a conditional statement
implements	Implements an interface
import	Used to import a package, class or interface
instanceof	Checks whether an object is an instance of a specific class or an interface
int	A data type that can store whole numbers from -2147483648 to 2147483647
interface	Used to declare a special type of class that only contains abstract methods
long	A data type that can store whole numbers from -9223372036854775808 to 9223372036854775808
module	Declares a module. New in Java 9
native	Specifies that a method is not implemented in the same Java source file (but in another language)
new	Creates new objects
package	Declares a package
private	An access modifier used for attributes, methods and constructors, making them only accessible within the declared class
protected	An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses
public	An access modifier used for classes, attributes, methods and constructors, making them accessible by any other class
requires	Specifies required libraries inside a module. New in Java 9

return	Finished the execution of a method, and can be used to return a value from a method
short	A data type that can store whole numbers from -32768 to 32767
static	A non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class
strictfp	Restrict the precision and rounding of floating point calculations
super	Refers to superclass (parent) objects
switch	Selects one of many code blocks to be executed
synchronized	A non-access modifier, which specifies that methods can only be accessed by one thread at a time
this	Refers to the current object in a method or constructor
throw	Creates a custom error
throws	Indicates what exceptions may be thrown by a method
transient	A non-accesss modifier, which specifies that an attribute is not part of an object's persistent state
try	Creates a try...catch statement
var	Declares a variable. New in Java 10
void	Specifies that a method should not have a return value
volatile	Indicates that an attribute is not cached thread-locally, and is always read from the "main memory"
while	Creates a while loop

Note: true, false, and null are not keywords, but they are literals and reserved words that cannot be used as identifiers.

Benefits of OOP

Benefits of OOP include:

- **Modularity.** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability.** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
- **Productivity.** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
- **Easily upgradable and scalable.** Programmers can implement system functionalities independently.
- **Interface descriptions.** Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
- **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

Arrays

- Normally, an array is a collection of similar type of elements which has contiguous memory location.
- Java array **is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements.**
- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with square brackets:
  - **String[] cars;**

- We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:
- **String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};**

- To create an array of integers, you could write:
- **int[] myNum = {10, 20, 30, 40};**

- **<u>Access the Elements of an Array</u>**
- You access an array element by referring to the index number.
- This statement accesses the value of the first element in cars:
- **Example:**

```
Class sample
{
public static void main(String[] args)
{
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
System.out.println(cars[1]);
System.out.println(cars[2]);
System.out.println(cars[3]);
}
}
```

Output:

```
Volvo
BMW
Ford
Mazda
```

Note: **Array indexes start with 0: [0] is the first element. [1] is the second element, etc.**

- **Change an Array Element**
- To change the value of a specific element, refer to the index number:
- **Example:**
- class Main
- {

  public static void main(String[] args)

  {

  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

  cars[0] = "Opel";

  System.out.println(cars[0]);

  }

  }

  Output:

  Opel
- **Array Length**
- To find out how many elements an array has, use the length property:
- **Example:**

  class sample

  {

  public static void main(String[] args)

  {

  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

  System.out.println(cars.length);

  }

  }

  Output:

  4

- **Loop Through an Array**
- You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.
- The following example outputs all elements in the cars array:
- **Example:**

```
class Main
{
public static void main(String[] args)
{
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i<cars.length; i++)
{
System.out.println(cars[i]);
}
}
}
```

Output:

```
Volvo
BMW
Ford
Mazda
```

- **Loop Through an Array with For-Each**
- There is also a "for-each" loop, which is used exclusively to loop through elements in arrays:
- **Syntax**

```
for (type variable :arrayname)
{
  ...
}
```

- The following example outputs all elements in the cars array, using a "for-each" loop:
- **Example**

```
class Main
{
public static void main(String[] args)
{
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars)
{
System.out.println(i);
}
}
}
```

Output:

Volvo

BMW

Ford

Mazda

Note : The example above can be read like this: **for each** String element (called **i** - as in **i**ndex) in **cars**, print out the value of **i**.

If you compare the for loop and **for-each** loop, you will see that the for-each method is easier to write, it does not require a counter (using the length property), and it is more readable.

- **Multidimensional Arrays**
- A multidimensional array is an array of arrays.
- A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself
- **Example**

  int[][] a = new int[3][4];
- Here, we have created a multidimensional array named a.
- It is a **2-dimensional array**, that can hold a maximum of 12 elements,
- **Example**
- Let's take another example of the multidimensional array.

- This time we will be creating a **3-dimensional array**. For example,

  String[][][] data = new String[3][4][2];
- Here, data is a 3d array that can hold a maximum of 24 (3*4*2) elements of type String.
- **How to initialize a 2d array in Java?**
- Here is how we can initialize a 2-dimensional array in Java.
- int[][] a = {

    {1, 2, 3},

    {4, 5, 6, 9},

    {7},

  };
- **Example: 2-dimensional Array**

  class MultidimensionalArray

  {

  public static void main(String[] args)

  {

  // create a 2d array

  int[][] a = {

  {1, 2, 3},

  {4, 5, 6, 9},

  {7},

  };

  // calculate the length of each row

  System.out.println("Length of row 1: " + a[0].length);

  System.out.println("Length of row 2: " + a[1].length);

  System.out.println("Length of row 3: " + a[2].length);

  }

  }
- **Output:**

  Length of row 1: 3

  Length of row 2: 4

Length of row 3: 1

- **Example: Print all elements of 2d array Using Loop**

```
class MultidimensionalArray
{
public static void main(String[] args)
{
int[][] a = {
{1, -2, 3},
{-4, -5, 6, 9},
{7},
};
for (int i = 0; i<a.length; ++i)
{
for(int j = 0; j < a[i].length; ++j)
{
System.out.println(a[i][j]);
}
}
}
}
```

- **Output:**

```
1
-2
3
-4
-5
6
9
7
```

- **We can also use the for...each loop to access elements of the multidimensional array. For example,**

```
class MultidimensionalArray
```

```
{
public static void main(String[] args)
{
// create a 2d array
int[][] a = {
{1, -2, 3},
{-4, -5, 6, 9},
{7},
};

// first for...each loop access the individual array
// inside the 2d array
for (int[] innerArray: a)
{
// second for...each loop access each element inside the row
for(int data: innerArray)
{
System.out.println(data);
}
}
}
}
```

- **Output:**

```
1
-2
3
-4
-5
6
9
7
```