UNIT-1

Introduction to Java

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

The term WORA, write once and run everywhere is often associated with this language. It means whenever we compile a Java code, we get the byte code (.class file), and that can be executed (without compiling it again) on different platforms provided they support Java. In the year 1995, Java language was developed. It is mainly used to develop web, desktop, and mobile devices. The Java language is known for its robustness, security, and simplicity features. That is designed to have as few implementation dependencies as possible.

Application Programs

The Implementation of an application program in Java application includes the following steps.

1. The program creation (writing the code)
2. The program compilation.
3. Executing the compiled code.

It is worth noting here that JDK (Java Development Kit) should be installed properly on the system, and the path should also be set.

The program Creation

The Java program can be written using a Text Editor (Notepad++ or NotePad or other editors will also do the job.) or IDE (Eclipse, NetBeans, etc.).

FileName: TestClass.java

```
1. public class TestClass
2. {
3. // main method
4. public static void main(String args [])
5. {
6. System.out.println("Hello World is my first Java Program.");
7. }
8. }
```

Write the above code and save the file with the name TestClass. The file should have the .java extension.

The program Compilation

Open the command prompt, and type javac TestClass.java. javac is the command that makes the Java compiler come to action to compile the Java program. After the command, we must put the name of the file that needs to be compiled. In our case, it is TestClass.java. After typing, press the enter button. If everything goes well, a TestClass.class file will be generated that contains the byte code. If there is some error in the program, the compiler will point it out, and TestClass.class will not be created.

Running / Executing the program

After the .class file is created, type java TestClass to run the program. The output of the program will be shown on the console, which is mentioned below.

Output:

Hello World is my first Java Program.

Name of Java

Java is also a name of an island in Indonesia where coffee (named Java Coffee) was produced. The name Java was chosen by James Gosling because he was having coffee near his office. Readers should note that Java is not an acronym. It is just a name.

Terminologies in Java

JVM (Java Virtual Machine): JVM is the specification that facilitates the runtime environment in which the execution of the Java bytecode takes place. Whenever one uses the command java, an instance of the JVM is created. JVM facilitates the definition of the memory area, register set, class file format, and fatal error reporting. Note that the JVM is platform dependent.

Byte Code: It has already been discussed in the introductory part that the Java compiler compiles the Java code to generate the .class file or the byte code. One has to use the javac command to invoke the Java compiler.

Java Development Kit (JDK): It is the complete Java Development Kit that encompasses everything, including JRE(Java Runtime Environment), compiler, java docs, debuggers, etc. JDK must be installed on the computer for the creation, compilation, and execution of a Java program.

Java Runtime Environment (JRE): JRE is part of the JDK. If a system has only JRE installed, then the user can only run the program. In other words, only the java command works. The compilation of a Java program will not be possible (the javac command will not work).

Garbage Collector: Programmers are not able to delete objects in Java. In order to do so, JVM has a program known as Garbage Collector. Garbage Collectors recollect or delete unreferenced objects. Garbage Collector makes the life of a developer/ programmer easy as they do not have to worry about memory management.

Classpath: As the name suggests, classpath is the path where the Java compiler and the Java runtime search the .class file to load. Many inbuilt libraries are provided by the JDK. However, if someone wants to use the external libraries, it should be added to the classpath.

Salient Features of Java

Platform Independent: Instead of directly generating the .exe file, Java compiler converts the Java code to byte code, and this byte code can be executed on different platforms without any issue, which makes Java a platform-independent language. Note that in order to execute the byte code, JVM has to be installed on the system, which is platform dependent.

Object-Oriented Programming Language: The concept of object-oriented programing is based on the concept of objects and classes. Also, there are several qualities that are present in object-oriented programming. A few of them are mentioned below.

- Abstraction
- Inheritance
- Polymorphism
- Encapsulation

The Java language also extensively uses the concepts of classes and objects. Also, all these features mentioned above are there in Java, which makes Java an object-oriented programming language. Note that Java is an object-oriented programming language but not 100% object-oriented.

Simple: Java is considered a simple language because it does not have the concept of pointers, multiple inheritances, explicit memory allocation, or operator overloading.

Robust: Java language is very much robust. The meaning of robust is reliable. The Java language is developed in such a fashion that a lot of error checking is done as early as possible. It is because of this reason that this language can identify those errors that are difficult to identify in other programming languages. Exception Handling, garbage collections, and memory allocation are the features that make Java robust.

Secure: There are several errors like buffer overflow or stack corruption that are not possible to exploit in the Java language. We know that the Java language does not have pointers. Therefore, it is not possible to have access to out-of-bound arrays. If someone tries to do so, ArrayIndexOutofBound Exception is raised. Also, the execution of the Java programs happens in an environment that is completely independent of the Operating System, which makes this language even more secure.

Distributed: Distributed applications can be created with the help of the Java language. Enterprise Java beans and Remote Method Invocation are used for creating distributed applications.

Multithreading: The Java language supports multithreading. The multithreading feature supports the execution of two or more parts of the program concurrently. Thus, the utilization of the CPU is maximized.

Portability: We know that Java is a platform-independent language. Thus, the byte code generated on one system can be taken on any other platform for execution, which makes Java portable.

High-Level Performance: The architecture of Java is created in such a fashion that it decreases runtime overhead. In some places, Java uses JIT (Just In Time) compiler when the code is compiled on a demand basis, where the compiler is only compiling those methods that are invoked and thus making the faster execution of applications.

Dynamic Flexibility: The Java language follows the Object-Oriented programming paradigm, which gives us the liberty to add new methods and classes to the existing classes. The Java language also supports functions mentioned in C/C++ languages and which are generally referred to as the native methods.

Sandbox Execution: It is a known fact that Java programs are executed in different environment, which gives liberty to users to execute their own applications without impacting the underlying system using the byte code verifier. The Byte code verifier also gives extra security as it checks the code for the violation of access.

Write Once Run Anywhere: The Java code is compiled by the compiler to get the .class file or the byte code, which is completely independent of any machine architecture.

Compiled and Interpreted Language: Most of languages are either interpreted language or the compiled language. However, in the case of the Java language, it is compiled as well as the interpreted language. The Java code is compiled to get the bytecode, and the bytecode is interpreted by the software-based interpreter.

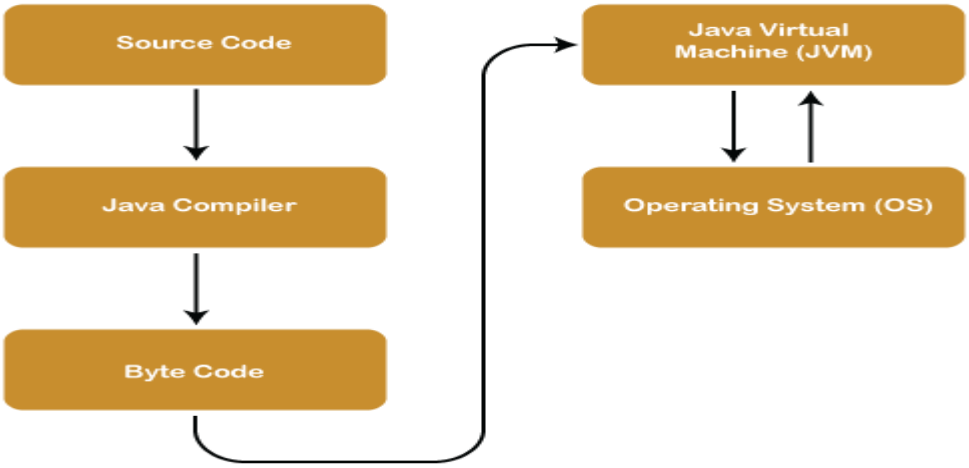
Java Architecture

Java Architecture is a collection of components, i.e., JVM, JRE, and JDK. It integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. Java Architecture explains each and every step of how a program is compiled and executed.

Java Architecture can be explained by using the following steps:

- There is a process of compilation and interpretation in Java.
- Java compiler converts the Java code into byte code.
- After that, the JVM converts the byte code into machine code.
- The machine code is then executed by the machine.

The following figure represents the Java Architecture in which each step is elaborate graphically.



Now let's dive deep to get more knowledge about Java Architecture. As we know that the Java architecture is a collection of components, so we will discuss each and every component into detail.

Components of Java Architecture

The Java architecture includes the three main components:

- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Java Development Kit (JDK)

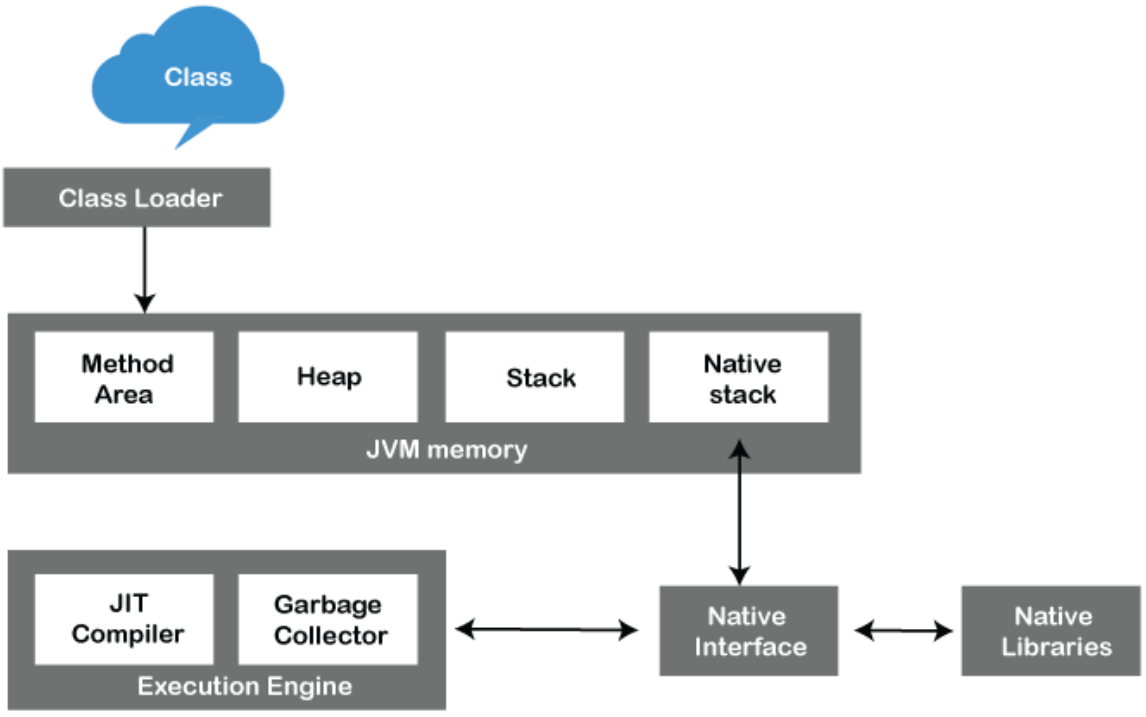
Java Virtual Machine

The main feature of Java is WORA. WORA stands for Write Once Run Anywhere. The feature states that we can write our code once and use it anywhere or on any operating system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.

JVM, first of all, loads the code into memory and verifies it. After that, it executes the code and provides a runtime environment. Java Virtual Machine (JVM) has its own architecture, which is given below:

JVM Architecture

JVM is an abstract machine that provides the environment in which Java bytecode is executed. The falling figure represents the architecture of the JVM.



Class Loader: Class Loader is a subsystem used to load class files. Class Loader first loads the Java code whenever we run it.

Class Method Area: In the memory, there is an area where the class data is stored during the code's execution. Class method area holds the information of static variables, static methods, static blocks, and instance methods.

Heap: The heap area is a part of the JVM memory and is created when the JVM starts up. Its size cannot be static because it increase or decrease during the application runs.

Stack: It is also referred to as thread stack. It is created for a single execution thread. The thread uses this area to store the elements like the partial result, local variable, data used for calling method and returns etc.

Native Stack: It contains the information of all the native methods used in our application.

Execution Engine: It is the central part of the JVM. Its main task is to execute the byte code and execute the Java classes. The execution engine has three main components used for executing Java classes.

- Interpreter: It converts the byte code into native code and executes. It sequentially executes the code. The interpreter interprets continuously and evens the same method multiple times. This reduces the performance of the system, and to solve this, the JIT compiler is introduced.
- JIT Compiler: JIT compiler is introduced to remove the drawback of the interpreter. It increases the speed of execution and improves performance.
- Garbage Collector: The garbage collector is used to manage the memory, and it is a program written in Java. It works in two phases, i.e., Mark and Sweep. Mark is an area where the garbage collector identifies the used and unused chunks of memory. The Sweep removes the identified object from the Mark.

Java Native Interface

Java Native Interface works as a mediator between Java method calls and native libraries.

Java Runtime Environment

It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it. To learn more about the Java Runtime Environment, click here.

Java Development Kit

It is a software development environment used in the development of Java applications and applets. Java Development Kit holds JRE, a compiler, an interpreter or loader, and several development tools in it. To learn more about the Java Development Kit, click here.

These are three main components of Java Architecture. The execution of a program is done with all these three components.

JVM

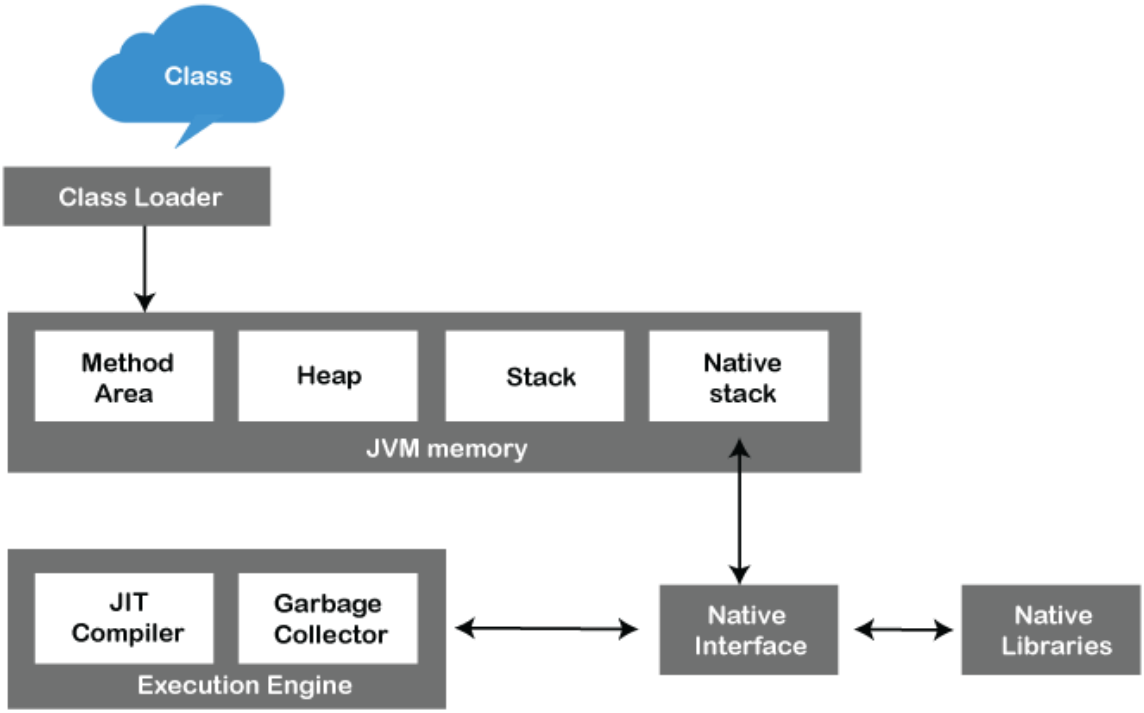
The main feature of Java is WORA. WORA stands for Write Once Run Anywhere. The feature states that we can write our code once and use it anywhere or on any operating

system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.

JVM, first of all, loads the code into memory and verifies it. After that, it executes the code and provides a runtime environment. Java Virtual Machine (JVM) has its own architecture, which is given below:

JVM Architecture

JVM is an abstract machine that provides the environment in which Java byte code is executed. The falling figure represents the architecture of the JVM.



Class Loader: Class Loader is a subsystem used to load class files. Class Loader first loads the Java code whenever we run it.

Class Method Area: In the memory, there is an area where the class data is stored during the code's execution. Class method area holds the information of static variables, static methods, static blocks, and instance methods.

Heap: The heap area is a part of the JVM memory and is created when the JVM starts up. Its size cannot be static because it increase or decrease during the application runs.

Stack: It is also referred to as thread stack. It is created for a single execution thread. The thread uses this area to store the elements like the partial result, local variable, data used for calling method and returns etc.

Native Stack: It contains the information of all the native methods used in our application.

Execution Engine: It is the central part of the JVM. Its main task is to execute the byte code and execute the Java classes. The execution engine has three main components used for executing Java classes.

- Interpreter: It converts the byte code into native code and executes. It sequentially executes the code. The interpreter interprets continuously and evens the same method multiple times. This reduces the performance of the system, and to solve this, the JIT compiler is introduced.
- JIT Compiler: JIT compiler is introduced to remove the drawback of the interpreter. It increases the speed of execution and improves performance.
- Garbage Collector: The garbage collector is used to manage the memory, and it is a program written in Java. It works in two phases, i.e., Mark and Sweep. Mark is an area where the garbage collector identifies the used and unused chunks of memory. The Sweep removes the identified object from the Mark

Java Native Interface

Java Native Interface works as a mediator between Java method calls and native libraries.

Scanner and System class

Java Scanner

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

The Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

Java Scanner Class Declaration

1. public final class Scanner
2. extends Object
3. implements Iterator<String>

How to get Java Scanner

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

1. Scanner in = new Scanner(System.in);

To get the instance of Java Scanner which parses the strings, we need to pass the strings in the constructor of Scanner class. For Example:

1. Scanner in = new Scanner("Hello Java");

Java System Class

The System class of java contains several useful class fields and methods. It also provides facilities like standard input, standard output, and error output Streams. It can't be instantiated.

The Java System class comes in the module of "java.base" & in the package of "java.lang".

In Java System Class, we have 3 different types of field and 28 different types of method.

Java System Class consists of following fields:-

SN	Modifier and Type	Field	Description
1	static PrintStrean	err	The "standard" error output stream.
2	static InputStream	in	The "standard" input stream.
3	static PrintStream	out	The "standard" output stream.

Comparison of Java vs C++

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Comparison of Java vs C#

There are many differences and similarities between Java and C#. A list of top differences between Java and C# are given below:

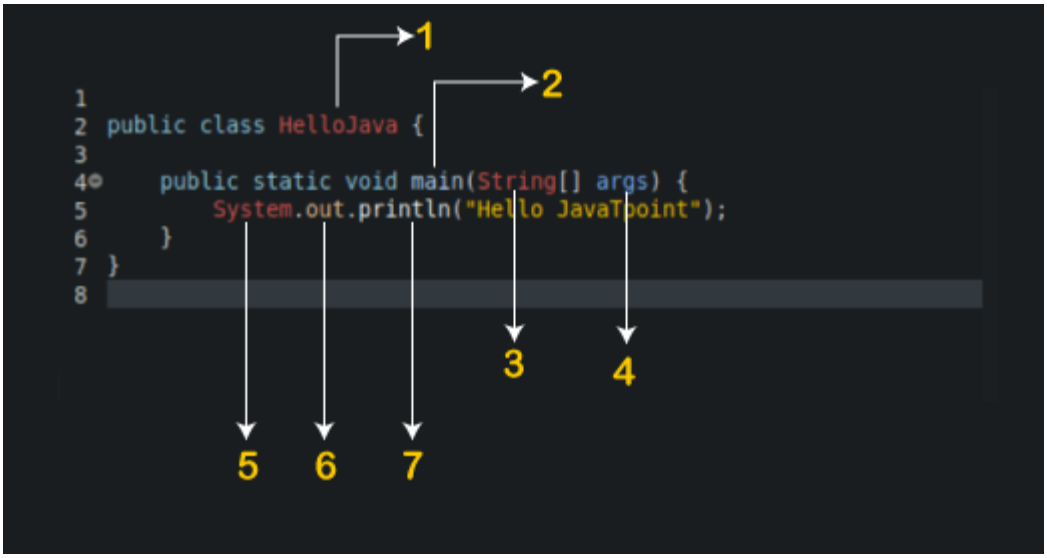
No.	Java	C#
1)	Java is a high level, robust, secured and object-oriented programming language developed by Oracle.	C# is an object-oriented programming language developed by Microsoft that runs on .Net Framework.
2)	Java programming language is designed to be run on a Java platform, by the help of Java Runtime Environment (JRE).	C# programming language is designed to be run on the Common Language Runtime (CLR).
3)	Java type safety is safe.	C# type safety is unsafe.
4)	In java, built-in data types that are passed by value are called primitive types.	In C#, built-in data types that are passed by value are called simple types.
5)	Arrays in Java are direct specialization of Object.	Arrays in C# are specialization of System.
6)	Java does not support conditional compilation.	C# supports conditional compilation using preprocessor directives.
7)	Java doesn't support goto statement.	C# supports goto statement.
8)	Java doesn't support structures and unions.	C# supports structures and unions.
9)	Java supports checked exception and unchecked exception.	C# supports unchecked exception.

Identifiers in Java

Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java, There are some reserved words that cannot be used as an identifier.

For every identifier there are some conventions that should be used before declaring them. Let's understand it with a simple Java program:

```
1. public class HelloJava {
2.  public static void main(String[] args) {
3.  System.out.println("Hello JavaTpoint");
4.  }
5.  }
```



From the above example, we have the following Java identifiers:

1. HelloJava (Class name)
2. main (main method)
3. String (Predefined Class name)
4. args (String variables)
5. System (Predefined class)
6. out (Variable name)
7. println (method)

let's understand the rules for Java identifier:

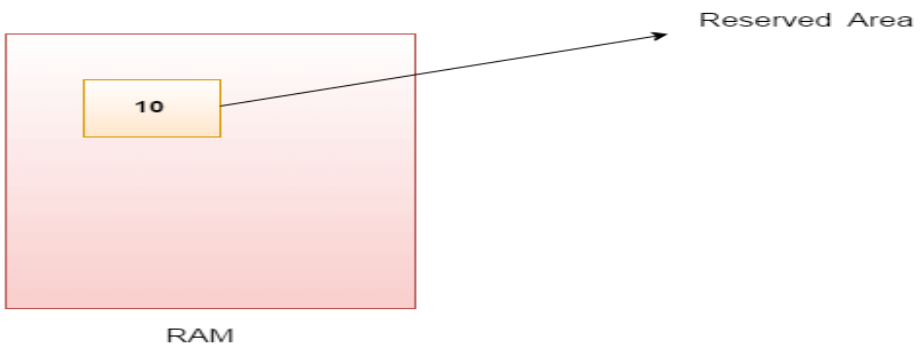
Rules for Identifiers in Java

There are some rules and conventions for declaring the identifiers in Java. If the identifiers are not properly declared, we may get a compile-time error. Following are some rules and conventions for declaring identifiers:

- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($). for example, @java is not a valid identifier because it contains a special character which is @.
- There should not be any space in an identifier. For example, java lab is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123java is an invalid identifier.
- An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
- We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.
- Java 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.
- There are two types of data types in Java: primitive and non-primitive.

Variables in java

- 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:

- 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. 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.

Java Variable Example: Add Two Numbers

```
1. public class Simple{
2. public static void main(String[] args){
3. int a=10;
4. int b=10;
5. int c=a+b;
6. System.out.println(c);
7. }
8. }
```

Output:

20

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, 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.

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:

1. Boolean one = false

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:

1. 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:

1. 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:

1. 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:

1. 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:

```
1. 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:

```
1. 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:

```
1. char letterA = 'A'
```

Java Non-Primitive Data Types

Class data type

A class in Java is a user defined data type i.e. it is created by the user. It acts a template to the data which consists of member variables and methods. An object is the variable of the class, which can access the elements of class i.e. methods and variables.

Example:

In the following example, we are creating a class containing the variables and methods (add() and sub()). Here, we are accessing the methods using the object of the Class obj**.**

Interface data type

An interface is similar to a class however the only difference is that its methods are abstract by default i.e. they do not have body. An interface has only the final variables and method declarations. It is also called a fully abstract class.

In the following example, we are creating the interface CalcInterface with two abstract methods (multiply() and divide()). Here, the class InterfaceExample implements the interface and further defines the methods of that interface. Then, the object of class is used to access those methods.

Arrays data type

An array is a data type which can store multiple homogenous variables i.e., variables of same type in a sequence. They are stored in an indexed manner starting with index 0. The variables can be either primitive or non-primitive data types.

Following example shows how to declare array of primitive data type int**:**

```
1. int [ ] marks;
```

Operators in Java

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,
- 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	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Unary Operator Example: ++ and --

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int x=10;
4. System.out.println(x++);//10 (11)
5. System.out.println(++x);//12
6. System.out.println(x--);//12 (11)
7. System.out.println(--x);//10
8. }}
```

Output:

```
10
12
12
10
```

Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. System.out.println(a+b);//15
6. System.out.println(a-b);//5
7. System.out.println(a*b);//50
8. System.out.println(a/b);//2
9. System.out.println(a%b);//0
```

```
10. }}
```

Output:

```
15
5
50
2
0
```

Shift operator

Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. System.out.println(10<<2);//10*2^2=10*4=40
4. System.out.println(10<<3);//10*2^3=10*8=80
5. System.out.println(20<<2);//20*2^2=20*4=80
6. System.out.println(15<<4);//15*2^4=15*16=240
7. }}
```

Output:

```
40
80
80
240
```

Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
1. public OperatorExample{
2. public static void main(String args[]){
3. System.out.println(10>>2);//10/2^2=10/4=2
4. System.out.println(20>>2);//20/2^2=20/4=5
5. System.out.println(20>>3);//20/2^3=20/8=2
6. }}
```

Output:

```
2
5
2
```

Java Shift Operator Example: >> vs >>>

```
1. public class OperatorExample{
2. public static void main(String args[]){
3.     //For positive number, >> and >>> works same
4.     System.out.println(20>>2);
5.     System.out.println(20>>>2);
6.     //For negative number, >>> changes parity bit (MSB) to 0
7.     System.out.println(-20>>2);
8.     System.out.println(-20>>>2);
9. }}
```

Output:

```
5
5
-5
1073741819
```

Relational Operators

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable relation_operator value

Some of the relational operators are-

- ==, Equal to returns true if the left-hand side is equal to the right-hand side.
- !=, Not Equal to returns true if the left-hand side is not equal to the right-hand side.
- <, less than: returns true if the left-hand side is less than the right-hand side.
- <=, less than or equal to returns true if the left-hand side is less than or equal to the right-hand side.
- >, Greater than: returns true if the left-hand side is greater than the right-hand side.
- >=, Greater than or equal to returns true if the left-hand side is greater than or equal to the right-hand side.

Example:

```
import java.io.*;
class GFG {
   public static void main(String[] args)
   {
      int a = 10;
      int b = 3;
      int c = 5;
```

```
        System.out.println("a > b: " + (a > b));
        System.out.println("a < b: " + (a < b));
        System.out.println("a >= b: " + (a >= b));
        System.out.println("a <= b: " + (a <= b));
        System.out.println("a == c: " + (a == c));
        System.out.println("a != c: " + (a != c));
    }
}
```

Output

```
a > b: true
a < b: false
a >= b: true
a <= b: false
a == c: false
a != c: true
```

Logical and Bitwise Operator

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a<b&&a<c);//false && true = false
7. System.out.println(a<b&a<c);//false & true = false
8. }}
```

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a<b&&a++<c);//false && true = false
7. System.out.println(a);//10 because second condition is not checked
8. System.out.println(a<b&a++<c);//false && true = false
9. System.out.println(a);//11 because second condition is checked
10. }}
```

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a>b||a<c);//true || true = true
7. System.out.println(a>b|a<c);//true | true = true
8. //|| vs |
9. System.out.println(a>b||a++<c);//true || true = true
10. System.out.println(a);//10 because second condition is not checked
11. System.out.println(a>b|a++<c);//true | true = true
12. System.out.println(a);//11 because second condition is checked
13. }}
```

Output:

```
true
true
true
10
true
11
```

Ternary Operator

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

Java Ternary Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=2;
4. int b=5;
5. int min=(a<b)?a:b;
6. System.out.println(min);
7. }}
```

Output:

2

Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=20;
5. a+=4;//a=a+4 (a=10+4)
6. b-=4;//b=b-4 (b=20-4)
7. System.out.println(a);
8. System.out.println(b);
9. }}
```

Output:

14
16

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

List of Java Keywords

A list of Java keywords or reserved words are given below:

1. abstract: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. boolean: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. break: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. byte: Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. case: Java case keyword is used with the switch statements to mark blocks of text.
6. catch: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. char: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. class: Java class keyword is used to declare a class.
9. continue: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

10. default: Java default keyword is used to specify the default block of code in a switch statement.
11. do: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. double: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. else: Java else keyword is used to indicate the alternative branches in an if statement.
14. enum: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. extends: Java extends keyword is used to indicate that a class is derived from another class or interface.
16. final: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. finally: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. float: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. for: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. if: Java if keyword tests the condition. It executes the if block if the condition is true.
21. implements: Java implements keyword is used to implement an interface.
22. import: Java import keyword makes classes and interfaces available and accessible to the current source code.
23. instanceof: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. int: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. interface: Java interface keyword is used to declare an interface. It can have only abstract methods.
26. long: Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. native: Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. new: Java new keyword is used to create new objects.
29. null: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. package: Java package keyword is used to declare a Java package that includes the classes.
31. private: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. protected: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. public: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. return: Java return keyword is used to return from a method when its execution is complete.
35. short: Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. static: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. strictfp: Java strictfp is used to restrict the floating-point calculations to ensure portability.

38. super: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. switch: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. synchronized: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. this: Java this keyword can be used to refer the current object in a method or constructor.
42. throw: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. throws: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. transient: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. try: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. void: Java void keyword is used to specify that a method does not have a return value.
47. volatile: Java volatile keyword is used to indicate that a variable may change asynchronously.
48. while: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

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.

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.

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.

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
```

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

Student.java

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. int x = 10;
4. int y = 12;
5. if(x+y > 20) {
6. System.out.println("x + y is greater than 20");
7. }
8. }
9. }
```

Output:

x + y is greater than 20

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:

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
4. else{
5. statement 2; //executes when condition is false
6. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. int x = 10;
4. int y = 12;
5. if(x+y < 10) {
6. System.out.println("x + y is less than      10");
7. }   else {
8. System.out.println("x + y is greater than 20");
9. }
10. }
11. }
```

Output:

x + y is greater than 20

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.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. }
4. else if(condition 2) {
5. statement 2; //executes when condition 2 is true
6. }
7. else {
8. statement 2; //executes when all the conditions are false
9. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. String city = "Delhi";
```

```
4. if(city == "Meerut") {
5. System.out.println("city is meerut");
6. }else if (city == "Noida") {
7. System.out.println("city is noida");
8. }else if(city == "Agra") {
9. System.out.println("city is agra");
10. }else {
11. System.out.println(city);
12. }
13. }
14. }
```

Output:

Delhi

Nested if-statement

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.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. if(condition 2) {
4. statement 2; //executes when condition 2 is true
5. }
6. else{
7. statement 2; //executes when condition 2 is false
8. }
9. }
```

Consider the following example.

Student.java

```
1. public class Student {
2. public static void main(String[] args) {
3. String address = "Delhi, India";
4.
5. if(address.endsWith("India")) {
6. if(address.contains("Meerut")) {
7. System.out.println("Your city is Meerut");
8. }else if(address.contains("Noida")) {
9. System.out.println("Your city is Noida");
10. }else {
11. System.out.println(address.split(",")[0]);
12. }
13. }else {
14. System.out.println("You are not living in India");
15. }
16. }
```

```
17. }
```

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.

```
1. switch (expression){
2.     case value1:
3.      statement1;
4.      break;
5.     .
6.     .
7.     .
8.     case valueN:
9.      statementN;
10.     break;
11.    default:
12.     default statement;
13. }
```

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

Student.java

```
1. public class Student implements Cloneable {
2. public static void main(String[] args) {
3. int num = 2;
4. switch (num){
5. case 0:
6. System.out.println("number is 0");
7. break;
8. case 1:
```

```
9.  System.out.println("number is 1");
10. break;
11. default:
12. System.out.println(num);
13. }
14. }
15. }
```

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

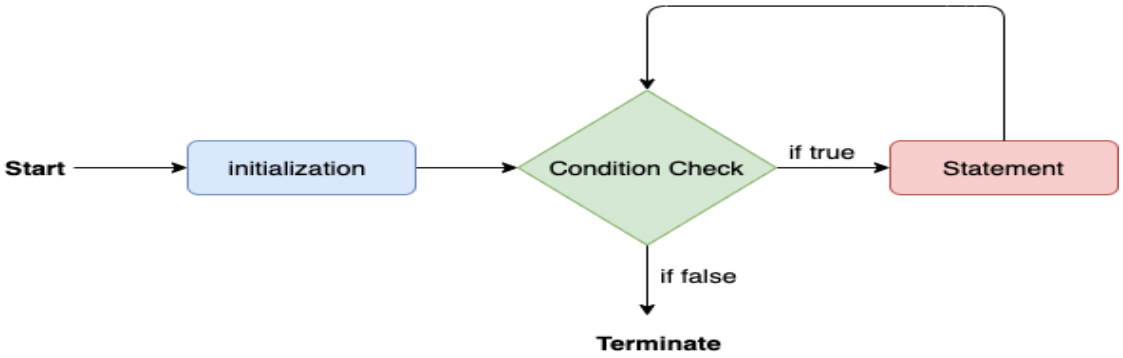
Let's understand the loop statements one by one.

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.

```
1. for(initialization, condition, increment/decrement) {
2. //block of statements
3. }
```

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

```
1. public class Calculattion {
2. public static void main(String[] args) {
3. // TODO Auto-generated method stub
4. int sum = 0;
5. for(int j = 1; j<=10; j++) {
6. sum = sum + j;
7. }
8. System.out.println("The sum of first 10 natural numbers is " + sum);
9. }
10. }
```

Output:

The sum of first 10 natural numbers is 55

Java while loop

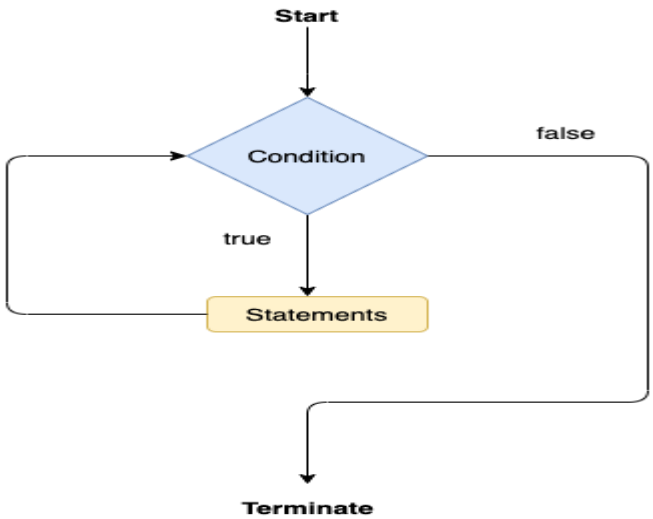
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while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

```
1. while(condition){
2. //looping statements
3. }
```

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



Consider the following example.

Calculation .java

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

Output:

Printing the list of first 10 even numbers

0
2
4
6
8
10

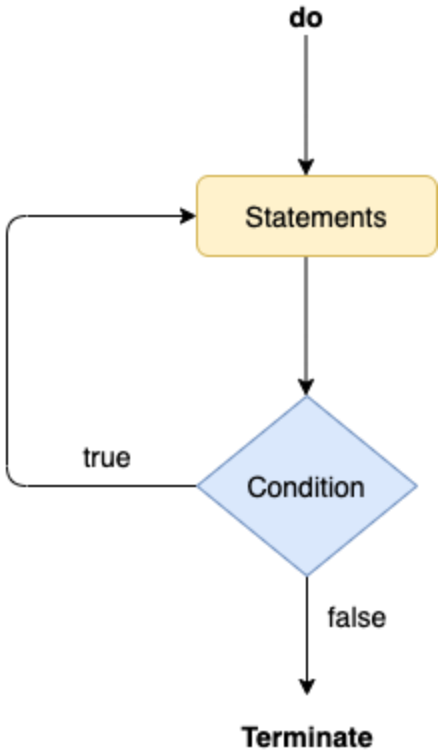
Java do-while loop

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.

```
1. do
2. {
3. //statements
4. } while (condition);
```

The flow chart of the do-while loop is given in the following image.



Consider the following example to understand the functioning of the do-while loop in Java.

Calculation.java

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

Output:

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

Jump Statements

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

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

```
1. public class BreakExample {
2.
3. public static void main(String[] args) {
4. // TODO Auto-generated method stub
5. for(int i = 0; i<= 10; i++) {
6. System.out.println(i);
7. if(i==6) {
8. break;
9. }
10. }
11. }
12. }
```

Output:

```
0
1
2
3
4
5
6
```

break statement example with labeled for loop

Calculation.java

```
1. public class Calculation {
2.
3. public static void main(String[] args) {
4. // TODO Auto-generated method stub
5. a:
6. for(int i = 0; i<= 10; i++) {
7. b:
8. for(int j = 0; j<=15;j++) {
9. c:
10. for (int k = 0; k<=20; k++) {
```

```
11. System.out.println(k);
12. if(k==5) {
13. break a;
14. }
15. }
16. }
17.
18. }
19. }
20.
21.
22. }
```

Output:

```
0
1
2
3
4
5
```

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.

```
1. public class ContinueExample {
2.
3. public static void main(String[] args) {
4. // TODO Auto-generated method stub
5.
6. for(int i = 0; i<= 2; i++) {
7.
8. for (int j = i; j<=5; j++) {
9.
10. if(j == 4) {
11. continue;
12. }
13. System.out.println(j);
14. }
15. }
16. }
17.
18. }
```

Output:

```
0
1
```

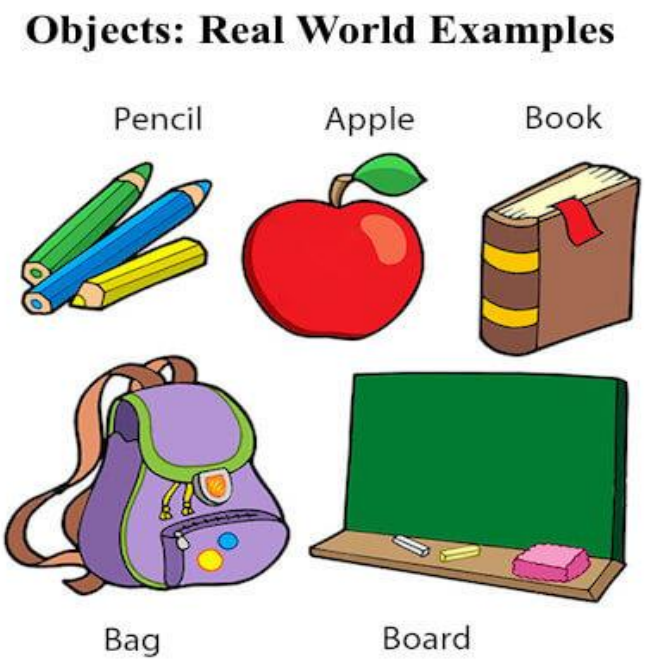
2
3
5
1
2
3
5
2
3
5

Objects and Classes in Java

In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

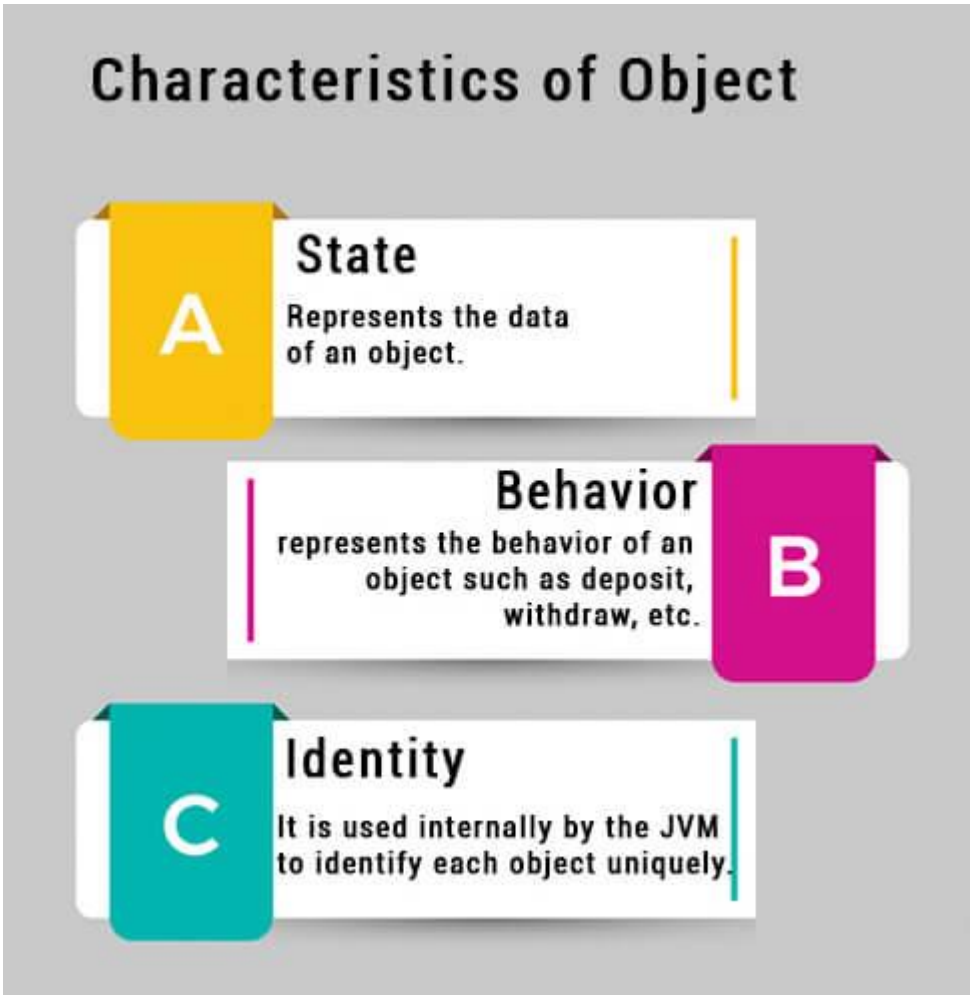
object in Java



An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- State: represents the data (value) of an object.
- Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.



For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

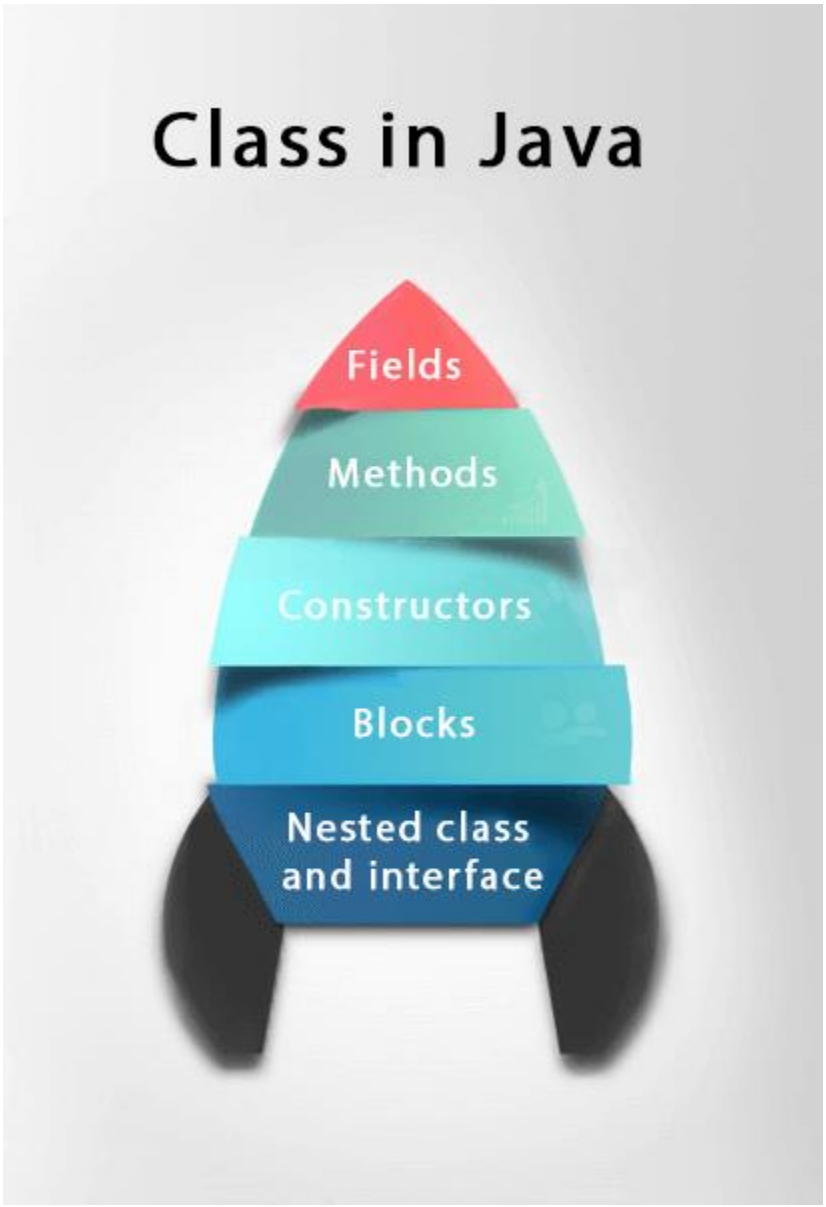
- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behavior.
- The object is an instance of a class.

class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface



Syntax to declare a class:

```
1. class <class_name>{
2.     field;
3.     method;
4. }
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

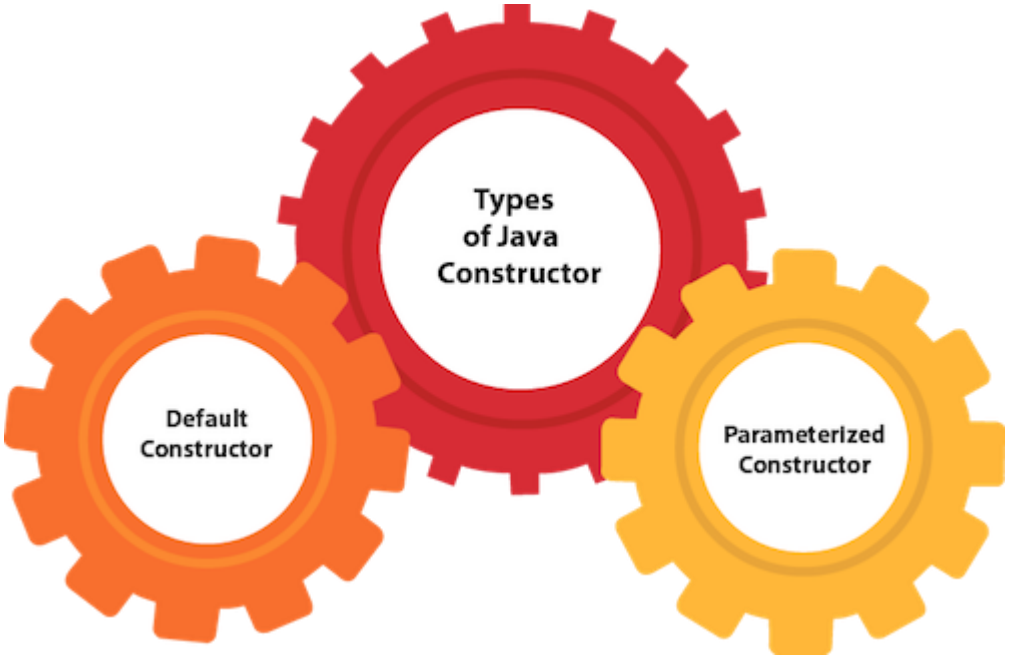
There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. <class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
1. //Java Program to create and call a default constructor
2. class Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. public static void main(String args[]){
7. //calling a default constructor
8. Bike1 b=new Bike1();
9. }
10. }
```

Output:
Bike is created

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
1. //Let us see another example of default constructor
2. //which displays the default values
3. class Student3{
4. int id;
5. String name;
6. //method to display the value of id and name
7. void display(){System.out.println(id+" "+name);}
8.
9. public static void main(String args[]){
10. //creating objects
11. Student3 s1=new Student3();
12. Student3 s2=new Student3();
13. //displaying values of the object
14. s1.display();
15. s2.display();
16. }
17. }
```

Output:
0 null
0 null

Explanation:In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of the parameterized constructor.
2. class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+" "+name);}
12.
13.    public static void main(String args[]){
14.    //creating objects and passing values
15.    Student4 s1 = new Student4(111,"Karan");
16.    Student4 s2 = new Student4(222,"Aryan");
17.    //calling method to display the values of object
18.    s1.display();
19.    s2.display();
20.   }
21. }
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
1. //Java program to overload constructors
2. class Student5{
3.     int id;
4.     String name;
5.     int age;
6.     //creating two arg constructor
7.     Student5(int i,String n){
8.     id = i;
9.     name = n;
10.    }
11.    //creating three arg constructor
12.    Student5(int i,String n,int a){
13.    id = i;
14.    name = n;
15.    age=a;
16.    }
17.    void display(){System.out.println(id+" "+name+" "+age);}
18.
19.    public static void main(String args[]){
20.    Student5 s1 = new Student5(111,"Karan");
21.    Student5 s2 = new Student5(222,"Aryan",25);
22.    s1.display();
23.    s2.display();
24.   }
25. }
```

Output:

```
111 Karan 0
222 Aryan 25
```

Java Object finalize() Method

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

Syntax

```
1. protected void finalize() throws Throwable
```

Throw

Throwable - the Exception is raised by this method

Example 1

```
1. public class JavafinalizeExample1 {
2. public static void main(String[] args)
3. {
4. JavafinalizeExample1 obj = new JavafinalizeExample1();
```

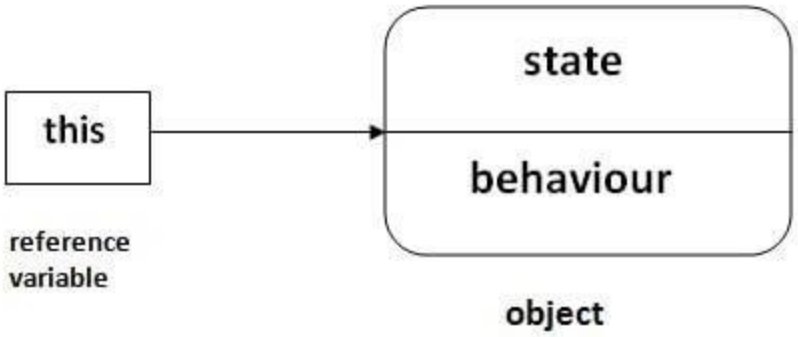
```
5.  System.out.println(obj.hashCode());
6.   obj = null;
7.   // calling garbage collector
8.   System.gc();
9.   System.out.println("end of garbage collection");
10.  }
11.  @Override
12.  protected void finalize()
13.  {
14.   System.out.println("finalize method called");
15.  }
16.  }
```

Output:

```
2018699554
end of garbage collection
finalize method called
```

this keyword in Java

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object.

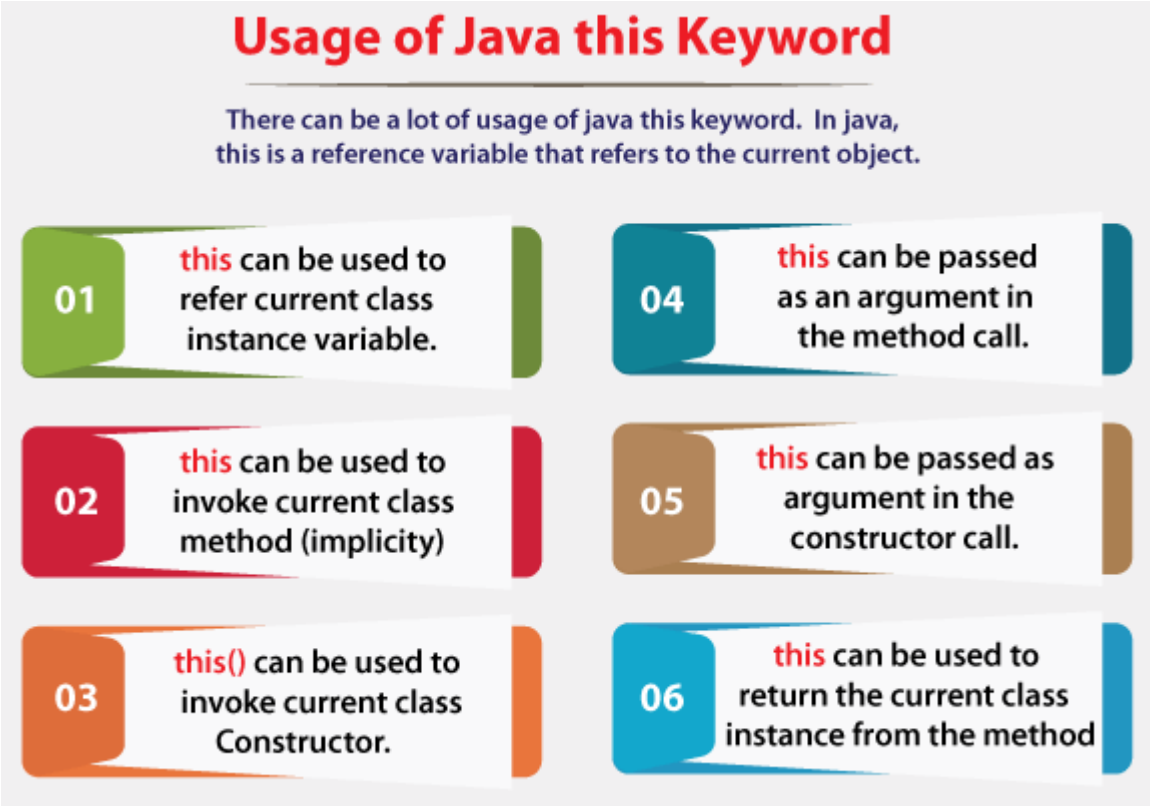


Usage of Java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Suggestion: If you are beginner to java, lookup only three usages of this keyword.



1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. rollno=rollno;
7. name=name;
8. fee=fee;
9. }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12. class TestThis1{
13. public static void main(String args[]){
14. Student s1=new Student(111,"ankit",5000f);
15. Student s2=new Student(112,"sumit",6000f);
16. s1.display();
17. s2.display();
18. }}
```

Output:

0 null 0.0
0 null 0.0

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. this.rollno=rollno;
7. this.name=name;
8. this.fee=fee;
9. }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12.
13. class TestThis2{
14. public static void main(String args[]){
15. Student s1=new Student(111,"ankit",5000f);
16. Student s2=new Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}
```

Output:

111 ankit 5000.0
112 sumit 6000.0