What is 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.

History of Java

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

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

Following are given significant points that describe the history of Java.

1) **James Gosling, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

5) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

6) James gosling and his team did brainstorm session and after the session, they came up with several names such as "JAVA", "SILK", "DNA", "RUBY" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.

Java name was decided after much discussion since it was unique than other names.

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

8) Notice that Java is just a name, not an acronym.

9) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

10) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

11) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

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

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. High Performance
9. Multithreaded
10. Dynamic

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-oriented programming language. Everything in Java is an object.

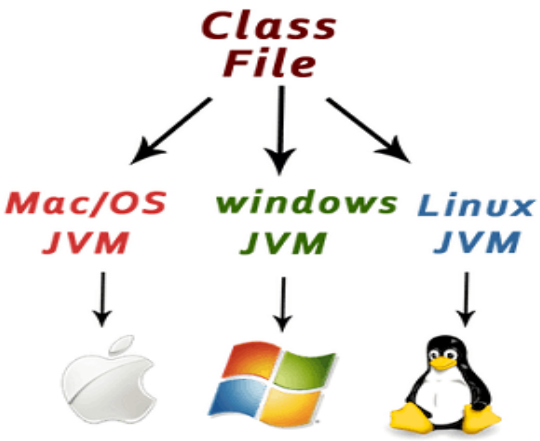
Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. 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**.

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)

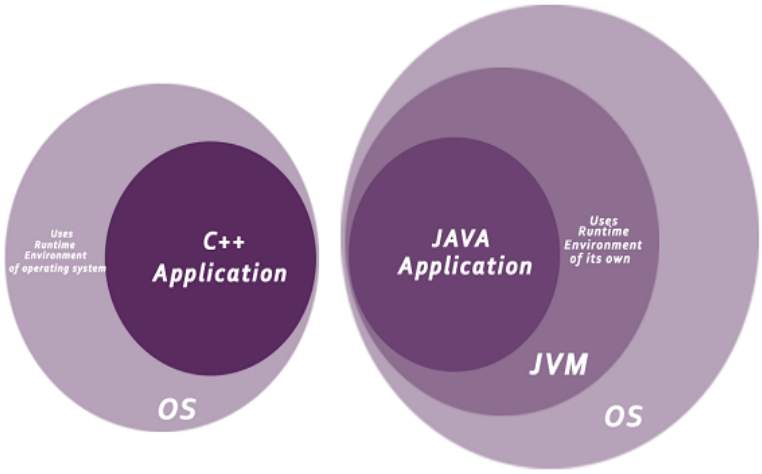
Java code can be executed on multiple platforms, for example, Windows, Linux, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-

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:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**
- **Classloader**
- **Bytecode Verifier**
- **Security Manager**



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 mining of Robust is strong. Java is robust because:

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

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.

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++.

C++ vs Java

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.
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	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-

	or not to override a function.	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.

1. JVM (Java Virtual Machine):

The JVM, or Java Virtual Machine, is an abstract computing machine that enables a computer to run Java programs.

It provides a runtime environment in which Java bytecode can be executed.

When you compile a Java source code file (.java file) using the Java compiler (javac), it's translated into bytecode (.class file), which is platform-independent.

The JVM then interprets or compiles this bytecode into machine code that is specific to the underlying hardware and operating system.

It handles memory management, garbage collection, security, and other runtime services required by Java applications.

The JVM essentially serves as an intermediary between the compiled Java code and the underlying hardware.

2. JRE (Java Runtime Environment):

The JRE, or Java Runtime Environment, is a package of software components that includes the JVM along with libraries and other files needed to run Java applications.

It provides the necessary runtime environment for executing Java bytecode.

Essentially, the JRE consists of the JVM plus the Java class libraries and other supporting files. When you want to run a Java application on your computer, you need to have the JRE installed.

The JRE allows Java applications to be executed on various platforms without needing to recompile the code for each specific system.

However, it doesn't include development tools like compilers or debuggers.

3. JDK (Java Development Kit):

The JDK, or Java Development Kit, is a comprehensive software development kit used for developing Java applications.

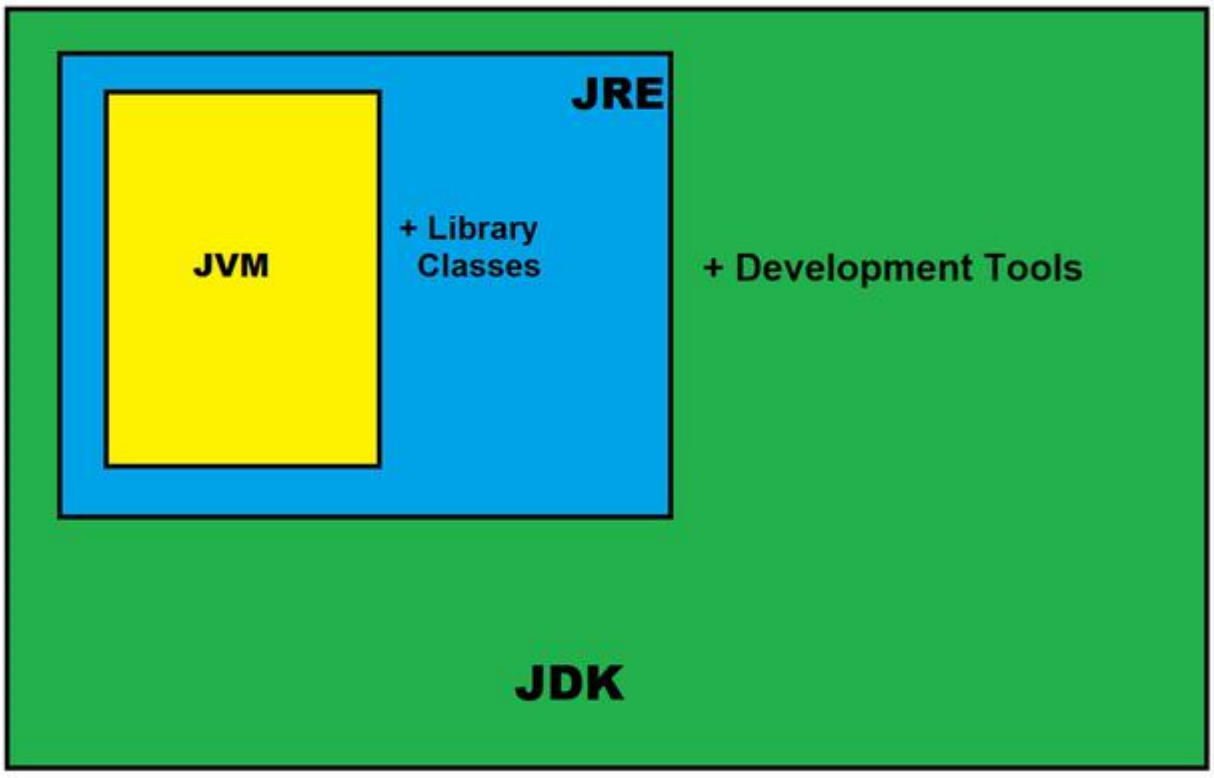
It includes the JRE, as well as development tools such as the Java compiler (javac), debugger (jdb), Java Archive Tool (jar), Java Documentation Generator (javadoc), and other utilities.

In addition to the runtime environment provided by the JRE, the JDK also offers everything a developer needs to write, compile, debug, and package Java applications.

It contains the necessary tools and libraries for developing Java programs from scratch.

Therefore, while the JRE is required for running Java applications, the JDK is necessary for both development and execution.

JDK=JRE+Development Tools



The Java Development Kit is an implementation of one of the Java Platform:

- Standard Edition (Java SE),
- Java Enterprise Edition (Java EE),
- Micro Edition (Java ME)

Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

How JVM Works – JVM Architecture?

JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a Java code. JVM is a part of JRE(Java Runtime Environment).

Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.

What it does

The JVM performs following operation:

- Loads code
- Verifies code

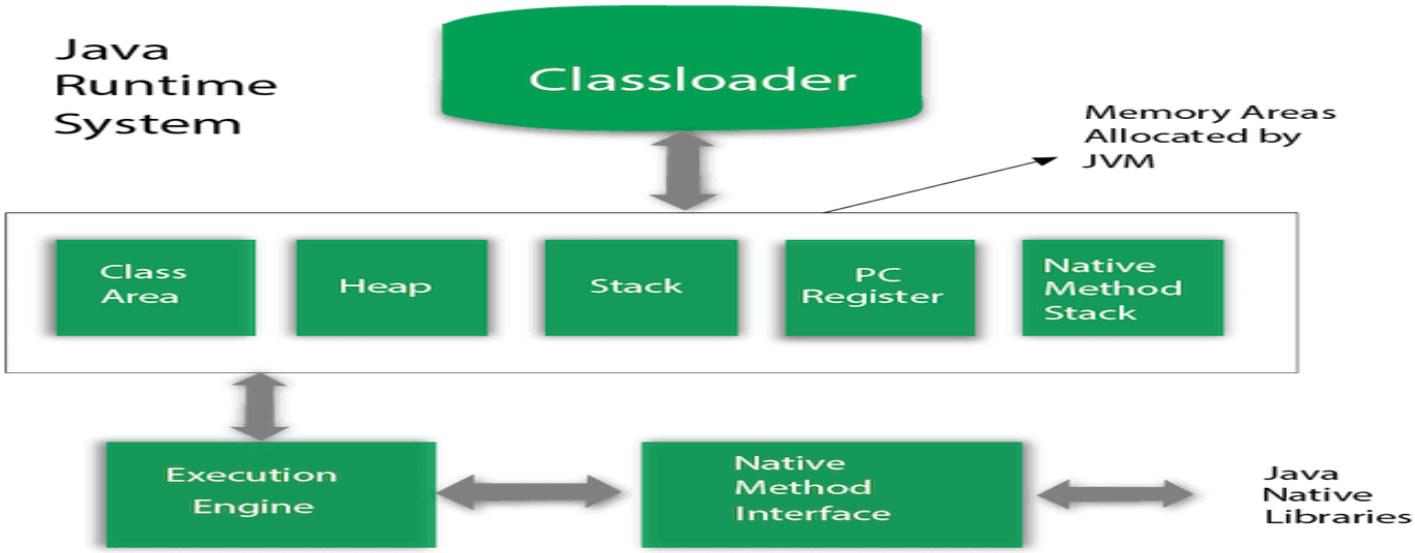
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

3) Heap

It is the runtime data area in which objects are allocated.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS librarie

---

Compiler and Interpreter are two different ways to translate a program from programming or scripting language to machine language.

Java Compiler

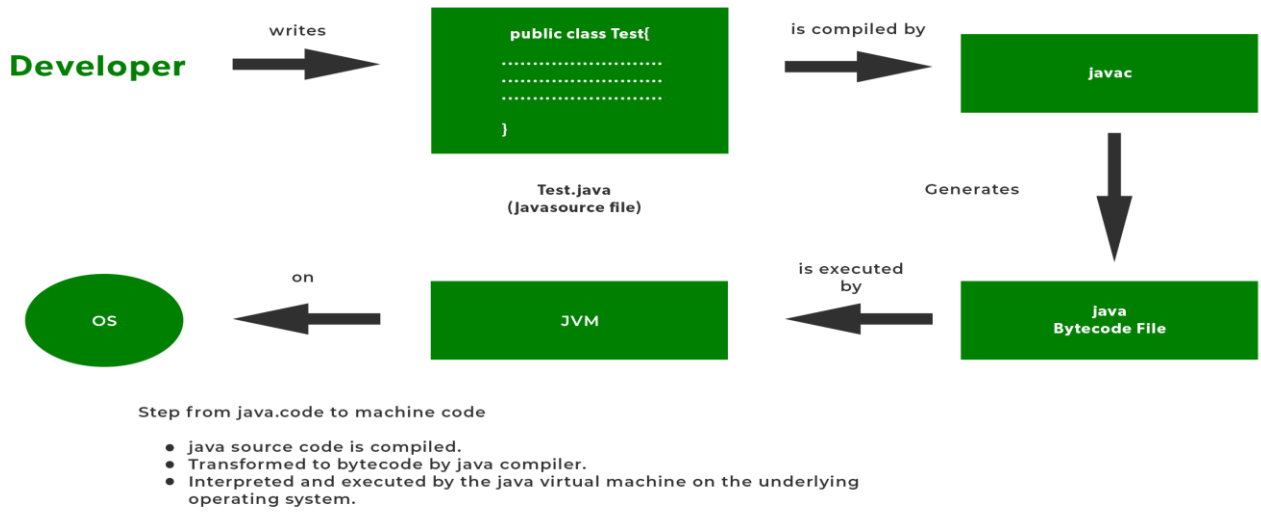
A compiler in Java translates the entire source code into a machine-code file or any intermediate code, and that file is then executed. It is platform-independent.

A bytecode is basically an intermediate code generated by the compiler after the compilation of its source code.

Java compiler can be operated by using the “Javac.exe” command from the command prompt. Some of the compiler options are given below:

- -help: it prints a summary of standard options.
- -version: returns the compiler information.
- -verbose: verbose output
- -nowarn: it is used to turn off warnings.

Java Compiler Step By Step



Roles of Java Compiler

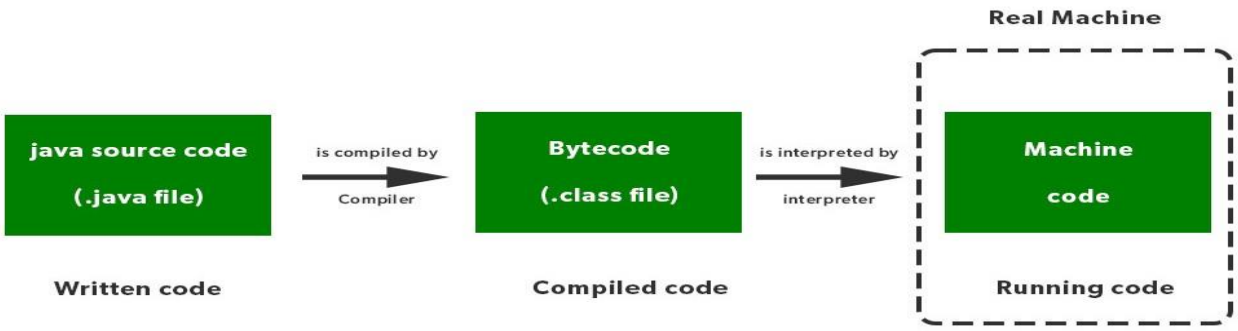
- It scans the complete source code in one go and then highlights the error.
- It requires more memory to produce the bytecode.
- It checks the correctness of the program by checking the type errors, syntax, etc.
- It also adds some additional code to our program if required.

Java Interpreter

Similarly, an Interpreter is a computer program that converts the high-level program statement into Assembly-level language. It converts the code into machine code when the program is run. Some of the interpreter options are given below:

- -version: displays interpreter version.
- -verbose: displays interpreter information.
- -help: displays interpreter options.

Java Interpreter Step By Step



Roles of Java Interpreter

- To convert the bytecode into the native code of the machine.
- This process is done line by line.
- If the error comes on any line, the process is stopped over there.

Here are some key differences between an interpreter and a compiler. They are as follows:

- The interpreter scans the program line by line and translates it into machine code whereas the compiler scans the entire program first and then translates it into machine code.
- The interpreter shows one error at a time whereas the compiler shows all errors and warnings at the same time.
- In the interpreter, the error occurs after scanning each line whereas, in the compiler, the error occurs after scanning the whole program.
- In an interpreter, debugging is faster whereas, in the compiler, debugging is slow.
- Execution time is more in the interpreter whereas execution time is less in the compiler.
- An interpreter is used by languages such as Java, and Python, and the compiler is used by languages such as C, C++, etc.

❖Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is used with a data type. Variable is a name of memory location.

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

1) Local Variable

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

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

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

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. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
public class A
{
  static int m=100;//static variable
  void method()
  {
    int n=90;//local variable
  }
  public static void main(String args[])
  {
    int data=50;//instance variable
  }
 }//end of class
```

❖ 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.

Rules For Defining Java Identifiers

There are certain rules for defining a valid Java identifier. These rules must be followed, otherwise, we get a compile-time error.

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '**$**'(dollar sign) and '_' (underscore).For example "geek@" is not a valid Java identifier as it contains a '@' a special character.
- Identifiers should **not** start with digits(**[0-9]**). For example "123geeks" is not a valid Java identifier.
- Java identifiers are **case-sensitive**.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- **Reserved Words** can't be used as an identifier. For example "int while = 20;" is an invalid statement as a while is a reserved word. There are **53** reserved words in Java.

❖ Literals

In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. **literal** is a notation that represents a fixed value in the source code

Types of Literals in Java

1. Integer Literal
2. Floating-point Litera;
3. Char Literal
4. String Literal
5. Boolean Literal

Integral literals

For Integral data types (byte, short, int, long), we can specify literals in 4 ways:-

Decimal literals (Base 10): In this form, the allowed digits are 0-9.

int x = 101;

Octal literals (Base 8): In this form, the allowed digits are 0-7. The octal number should be prefix with 0.

int x = 0146;

Hexa-decimal literals (Base 16): In this form, the allowed digits are 0-9, and characters are a-f. We can use both uppercase and lowercase characters as we know that java is a case-sensitive programming language, but here java is not case-sensitive.

The hexa-decimal number should be prefix with 0X or 0x.

int x = 0X123Face;

Binary literals: we can specify literal value even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.

int x = 0b1111;

Floating-Point literal

For Floating-point data types, we can specify literals in only decimal form, and we cant specify in octal and Hexadecimal forms.

Decimal literals(Base 10): In this form, the allowed digits are 0-9.

double d = 123.456;

Char literals

For char data types, we can specify literals in 4 ways:

Single quote: We can specify literal to a char data type as a single character within the single quote.

char ch = 'a';

Char literal as Integral literal: we can specify char literal as integral literal, which represents the Unicode value of the character, and that integral literal can be specified either in Decimal, Octal, and Hexadecimal forms. But the allowed range is 0 to 65535.

char ch = 062;

Unicode Representation: We can specify char literals in Unicode representation '\uxxxx'. Here xxxx represents 4 hexadecimal numbers.

char ch = '\u0061';

// Here /u0061 represent a.

Escape Sequence: Every escape character can be specified as char literals.

char ch = '\n';

String literals

String literal is a sequence of characters that is enclosed between **double** quotes ("") marks. It may be alphabet, numbers, special characters, blank space, etc. For example, "**Jack", "12345", "\n",** etc.

String s = "Hello";

Boolean literals

Boolean literals are the value that is either true or false. It may also have values 0 and 1

boolean b = true;

❖ Operators

Operators in Java are the symbols used for performing specific operations in Java. Operators make tasks like addition, multiplication, etc

Types of Operators in Java

There are multiple types of operators in Java all are mentioned below:

1. Arithmetic Operators

2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator

1. Arithmetic Operators

They are used to perform simple arithmetic operations on primitive data types.

- *** :** Multiplication
- **/ :** Division
- **% :** Modulo
- **+ :** Addition
- **– :** Subtraction

Example:

```
class GFG {

    public static void main (String[] args) {

        // Arithmetic operators

        int a = 10;

        int b = 3;

        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 % b = " + (a % b));

    }

}
```

Output

a + b = 13

a - b = 7

a * b = 30

a / b = 3

a % b = 1

2. Unary Operators

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

- **– : Unary minus**, used for negating the values.
- **+ : Unary plus** indicates the positive value (numbers are positive without this, however).
- **++ : Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.
    - **Post-Increment:** Value is first used for computing the result and then incremented.
    - **Pre-Increment:** Value is incremented first, and then the result is computed.
- **– – : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.
    - **Post-decrement:** Value is first used for computing the result and then decremented.
    - **Pre-Decrement: The value** is decremented first, and then the result is computed.

Example:

```
class GFG {

    public static void main(String[] args)

    {

        int a = 10;

        int b = 10;

        System.out.println("Postincrement : " + (a++));

        System.out.println("Preincrement : " + (++a));

        System.out.println("Postdecrement : " + (b--));

        System.out.println("Predecrement : " + (--b));    }}
```

Output

Postincrement : 10

Preincrement : 12

Postdecrement : 10

Predecrement : 8

3. Assignment Operator

'=' Assignment operator is used to assign a value to any variable. It has right-to-left associativity.

The general format of the assignment operator is:

Variable=value;

the assignment operator can be combined with other operators to build a shorter version of the statement called a **Compound Statement**. For example, instead of a **=** a+5, we can write a **+=** 5.

- **+=**, for adding the left operand with the right operand and then assigning it to the variable on the left.
- **-=**, for subtracting the right operand from the left operand and then assigning it to the variable on the left.
- ***=**, for multiplying the left operand with the right operand and then assigning it to the variable on the left.

- **/=**, for dividing the left operand by the right operand and then assigning it to the variable on the left.
- **%=**, for assigning the modulo of the left operand by the right operand and then assigning it to the variable on the left.

Example:

```
class GFG {

    public static void main(String[] args)

    {

        int f = 7;

        System.out.println("f += 3: " + (f += 3));

        System.out.println("f -= 2: " + (f -= 2));

        System.out.println("f *= 4: " + (f *= 4));

        System.out.println("f /= 3: " + (f /= 3));

        System.out.println("f %= 2: " + (f %= 2));

            }}
```

Output

f += 3: 10

f -= 2: 8

f *= 4: 32

f /= 3: 10

f %= 2: 0

4. 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.

- **==, 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:

```
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

5. Logical Operators

These operators are used to perform “logical AND” and “logical OR” operations,. Used extensively to test for several conditions for making a decision.

Conditional operators are:

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.
- **!, Logical NOT:** returns true when a condition is false and vice-versa

Example:

```
class GFG {
    public static void main (String[] args) {
        boolean x = true;
        boolean y = false;
        System.out.println("x && y: " + (x && y));
        System.out.println("x || y: " + (x || y));
        System.out.println("!x: " + (!x));
    }}
```

Output

x && y: false

x || y: true

!x: false

6. Ternary operator

The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.

The general format is:

condition**?**iftrue**:**iffalse

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

Example:

```
public class operators {
   public static void main(String[] args)
   {
      int a = 20, b = 10, c = 30, result;
      result  = ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
      System.out.println("Max of three numbers = "+ result);  }}
```

Output

Max of three numbers = 30

7. Bitwise Operators

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

```
class GFG {

    public static void main(String[] args)

    {

        int d = 0b1010;

        int e = 0b1100;

        System.out.println("d & e: " + (d & e));

        System.out.println("d | e: " + (d | e));

        System.out.println("d ^ e: " + (d ^ e));

        System.out.println("~d: " + (~d));

        System.out.println("d << 2: " + (d << 2));

        System.out.println("e >> 1: " + (e >> 1));

        System.out.println("e >>> 1: " + (e >>> 1));

    }}
```

Output

d & e: 8

d | e: 14

d ^ e: 6

~d: -11

d << 2: 40

e >> 1: 6

e >>> 1: 6

8. Shift Operators

These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two. General format-

number**shift_op**number_of_places_to_shift;

- **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two.
- **>>, Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect to dividing the number with some power of two.
- **>>>, Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

```
// Java Program to implement

// shift operators

import java.io.*;

// Driver Class

class GFG {

    // main function

    public static void main(String[] args)

    {        int a = 10;

          // using left shift

        System.out.println("a<<1 : " + (a << 1));

        // using right shift

        System.out.println("a>>1 : " + (a >> 1));

    }}
```

Output

a<<1 : 20

a>>1 : 5

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.

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.

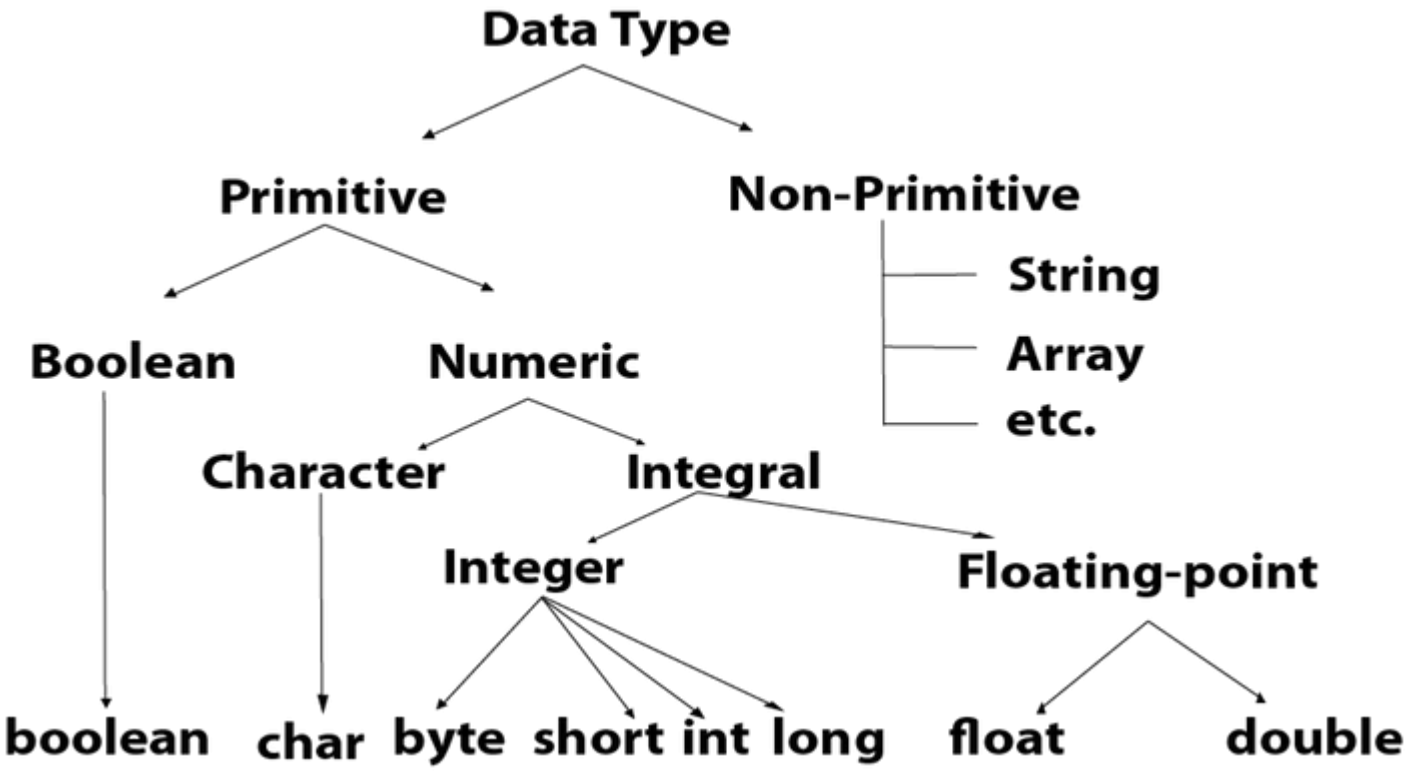
Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

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

Java Comments

The Java comments are the statements in a program that are not executed by the compiler and interpreter.

Why do we use comments in a code

- Comments are used to make the program more readable by adding the details of the code.

- It makes easy to maintain the code and to find the errors easily.
- The comments can be used to provide information or explanation about the variable, method, class, or any statement.
- It can also be used to prevent the execution of program code while testing the alternative code.

Types of Java Comments

There are three types of comments in Java.

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

1) Java Single Line Comment

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.

Single line comments starts with two forward slashes **(//)**. Any text in front of // is not executed by Java.

10 2) Java Multi Line Comment

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).

Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.

Note: Usually // is used for short comments and / */ is used for longer comments.*

3) Java Documentation Comment

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

To create documentation API, we need to use the **javadoc tool**. The documentation comments are placed between /** and

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

switch case

```
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);
}
}
}
```

Java for-each loop

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

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

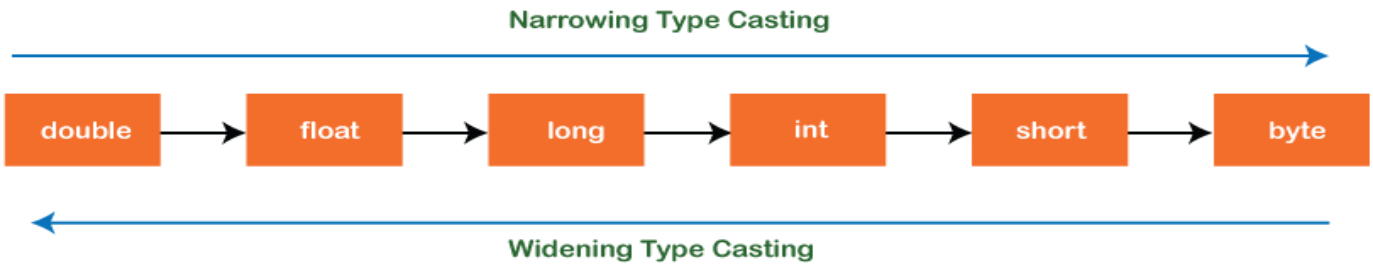
Consider the following example to understand the functioning of the for-each loop in Java.

Calculation.java

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

Type Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.



Type Casting in Java

Type casting

Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting

There are two types of type casting:

- Widening Type Casting
- Narrowing Type Casting

Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

1. **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

WideningTypeCastingExample.java

```
1. public class WideningTypeCastingExample
2. {
3. public static void main(String[] args)
4. {
5. int x = 7;
6. //automatically converts the integer type into long type
7. long y = x;
8. //automatically converts the long type into float type
9. float z = y;
10. System.out.println("Before conversion, int value "+x);
11. System.out.println("After conversion, long value "+y);
12. System.out.println("After conversion, float value "+z);
13. }
14. }
```

Output

```
Before conversion, the value is: 7
After conversion, the long value is: 7
After conversion, the float value is: 7.0
```

In the above example, we have taken a variable x and converted it into a long type. After that, the long type is converted into the float type.

Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

NarrowingTypeCastingExample.java

```
1. public class NarrowingTypeCastingExample
2. {
3. public static void main(String args[])
4. {
5. double d = 166.66;
6. //converting double data type into long data type
7. long l = (long)d;
8. //converting long data type into int data type
9. int i = (int)l;
10. System.out.println("Before conversion: "+d);
11. //fractional part lost
12. System.out.println("After conversion into long type: "+l);
13. //fractional part lost
14. System.out.println("After conversion into int type: "+i);
15. }
16. }
```

Output

```
Before conversion: 166.66
After conversion into long type: 166
After conversion into int type: 166
```

1 . if

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

Syntax of if Statement

if(condition)

{ // Statements to execute if condition is true }

2. if-else

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else when the condition is false? Here comes the else statement. We can use the else statement with the if statement to execute a block of code when the condition is false.

Syntax :

```
if (condition)
{
   // Executes this block if condition is true
}
else
{
   // Executes this block if condition is false
}
```

3. Nested if-else in C

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement.

Syntax of Nested if-else

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition2)
  {
    // Executes when condition2 is true
  }
  else
```

```
  {
      // Executes when condition2 is false
}
```

4. if-else-if Ladder in C

The if else if statements are used when the user has to decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

Syntax of if-else-if Ladder

```
if (condition)
   statement;
else if (condition)
   statement;
.
.
else
   statement;
```

5. switch Statement in C

The switch case statement is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.

Syntax of switch

```
switch (expression) {
   case value1:
      statements;
   case value2:
```

```
    statements;
  ....
  ....
  ....
  default:
    statements;
}
```

7. Jump Statements in C

A) break

This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

B) continue

This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

for Loop

for loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times.

for (initialize expression; test expression; update expression)

```
{
   // body of for loop
}
```