Introduction:

Object-oriented programming: As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOPs Concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Class
- Object
- Method
- Message Passing

Object-Oriented Programming paradigm

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object–oriented programming are −

- Bottom–up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

Grady Booch has defined object–oriented programming as *"a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships"*.

Let us learn about the different characteristics/basic concepts of an Object-Oriented Programming language:

1. **Polymorphism:** Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.
   For example:

```
// Java program to demonstrate Polymorphism

// This class will contain
// 3 methods with same name,
// yet the program will
// compile & run successfully
public class Sum {

   // Overloaded sum().
   // This sum takes two int parameters
   public int sum(int x, int y)
   {
      return (x + y);
   }

   // Overloaded sum().
   // This sum takes three int parameters
   public int sum(int x, int y, int z)
   {
      return (x + y + z);
   }

   // Overloaded sum().
   // This sum takes two double parameters
   public double sum(double x, double y)
   {
      return (x + y);
   }

   // Driver code
   public static void main(String args[])
   {
      Sum s = new Sum();
      System.out.println(s.sum(10, 20));
      System.out.println(s.sum(10, 20, 30));
      System.out.println(s.sum(10.5, 20.5));
   }
}
```

Output:

```
30

60

31.0
```

Polymorphism in Java are mainly of 2 types:

- Overloading in Java
- Overriding in Java

2. **Inheritance:** Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important terminology:

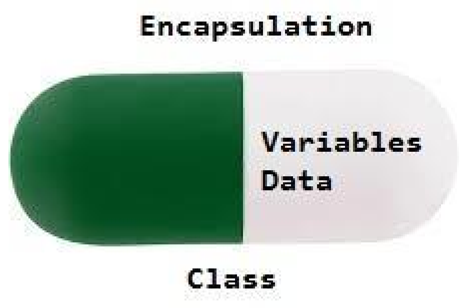
- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.

Syntax:

```
class derived-class extends base-class
{
  //methods and fields
}
```

3. **Encapsulation:** Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.
   - Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
   - As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
   - Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.



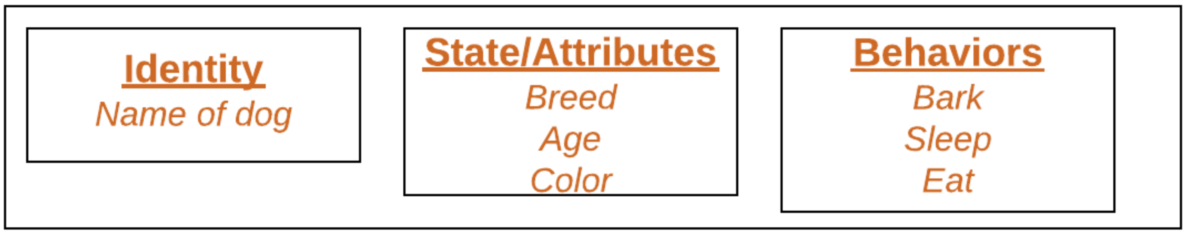
4. **Abstraction:** Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
   Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

5. **Class:** A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:
    0. **Modifiers**: A class can be public or has default access (Refer this for details).
    1. **Class name:** The name should begin with a initial letter (capitalized by convention).
    2. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
    3. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
    4. **Body:** The class body surrounded by braces, { }.
2. **Object:** It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:
    0. **State** : It is represented by attributes of an object. It also reflects the properties of an object.
    1. **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
    2. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

   Example of an object: dog

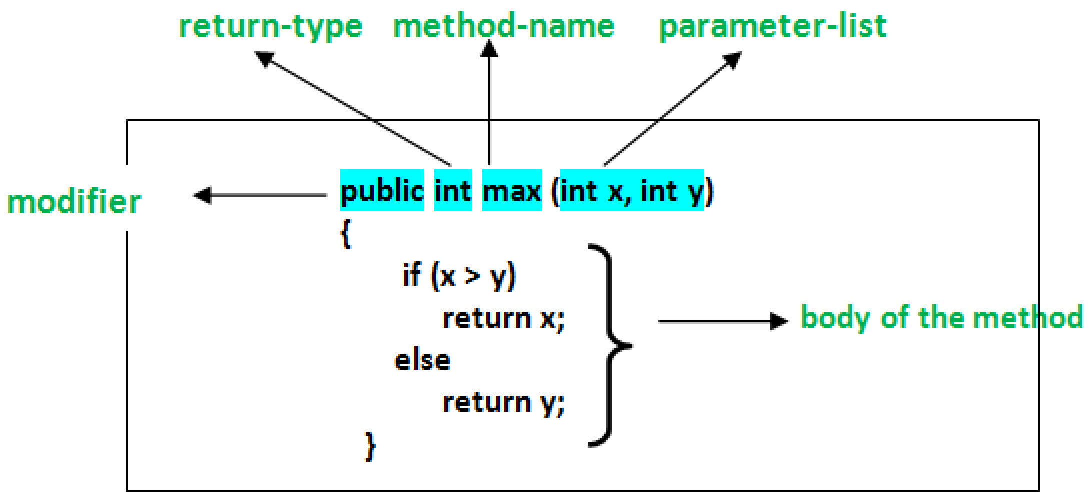


3. **Method:** A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python.
   Methods are **time savers** and help us to **reuse** the code without retyping the code.

   Method Declaration

   In general, method declarations has six components:

- **Access Modifier**: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  - **public:** accessible in all class in your application.
  - **protected:** accessible within the package in which it is defined and in its **subclass(es)(including subclasses declared outside the package)**
  - **private:** accessible only within the class in which it is defined.
  - **default (declared/defined without using any modifier):** accessible within same class and package within which its class is defined.
- **The return type**: The data type of the value returned by the method or void if does not return a value.
- **Method Name**: the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list**: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Exception list**: The exceptions you expect by the method can throw, you can specify these exception(s).
- **Method body**: it is enclosed between braces. The code you need to be executed to perform your intended operations.



```
public int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

2. **Message Passing:** Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Main application areas of OOP are:

- User interface design such as windows, menu.
- Real Time Systems
- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming

- Decision support and office automation systems etc.

Benefits of OOP*:*

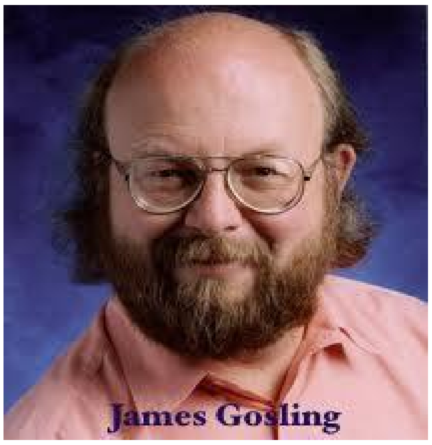
- It is easy to model a real system as real objects are represented by programming objects in OOP. The objects are processed by their member data and functions. It is easy to analyze the user requirements.
- With the help of inheritance, we can reuse the existing class to derive a new class such that the redundant code is eliminated and the use of existing class is extended. This saves time and cost of program.
- In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that can not be invaded by code in other part of the program.
- With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.
- Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
- It is possible to have multiple instances of an object to co-exist without any interference i.e. each object has its own separate member data and function.

2. JAVA EVOLUTION:

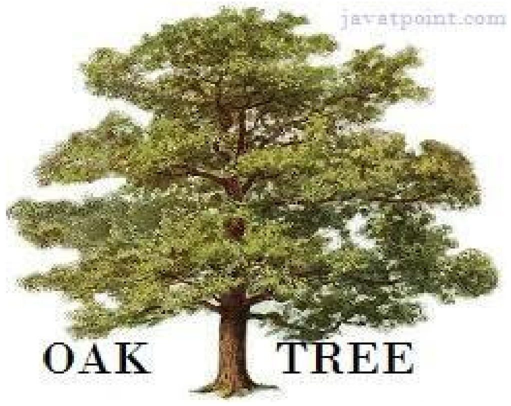
History of Java

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



Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There 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 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.

Why Java named "Oak"?
5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

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

Why Java Programming named "Java"?
7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc.

They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

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

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

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

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

12) JDK 1.0 released in(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 the new features in Java.

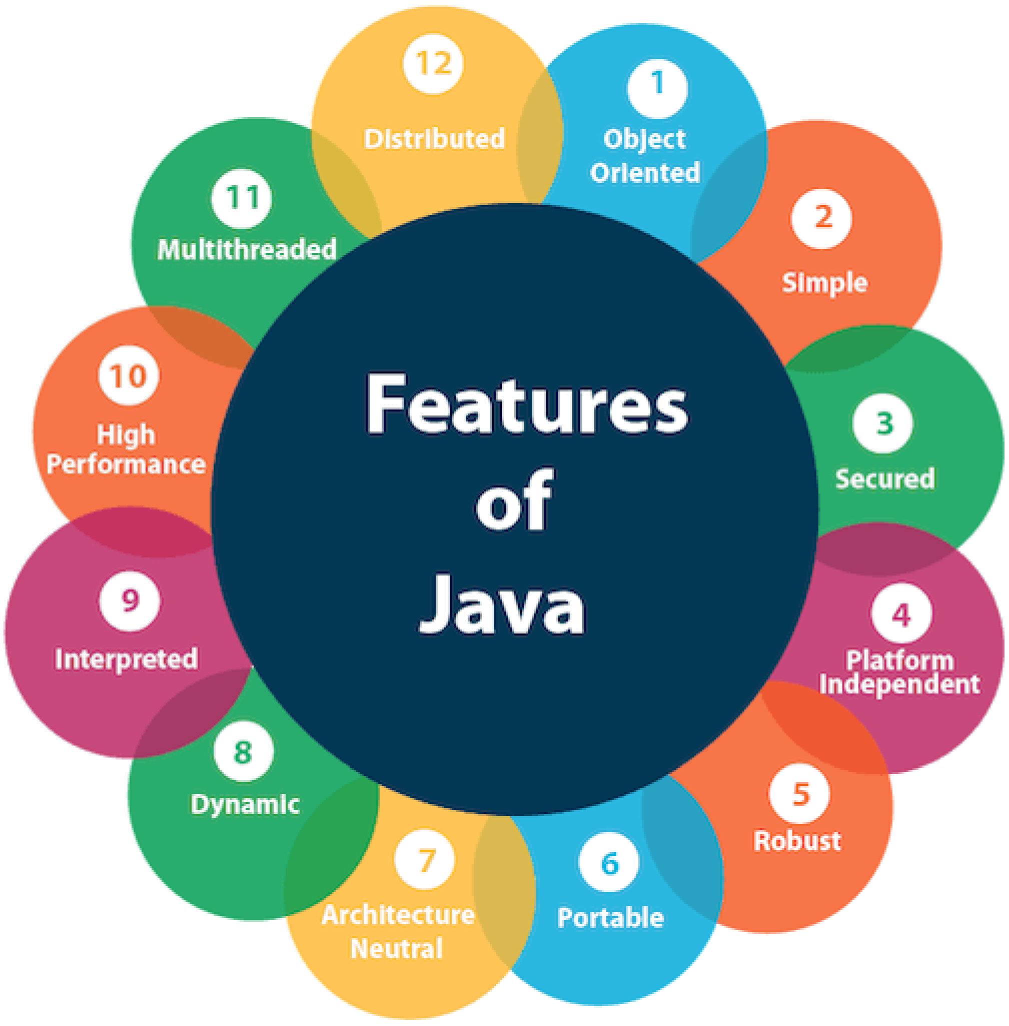
Java Version History
Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)

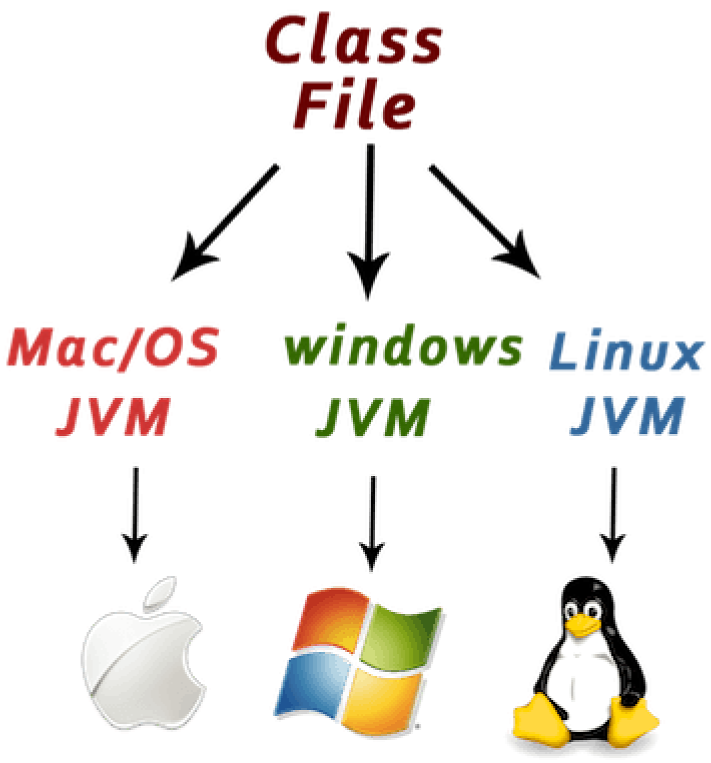
Features of Java

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 most important features of Java language is given below.



Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

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

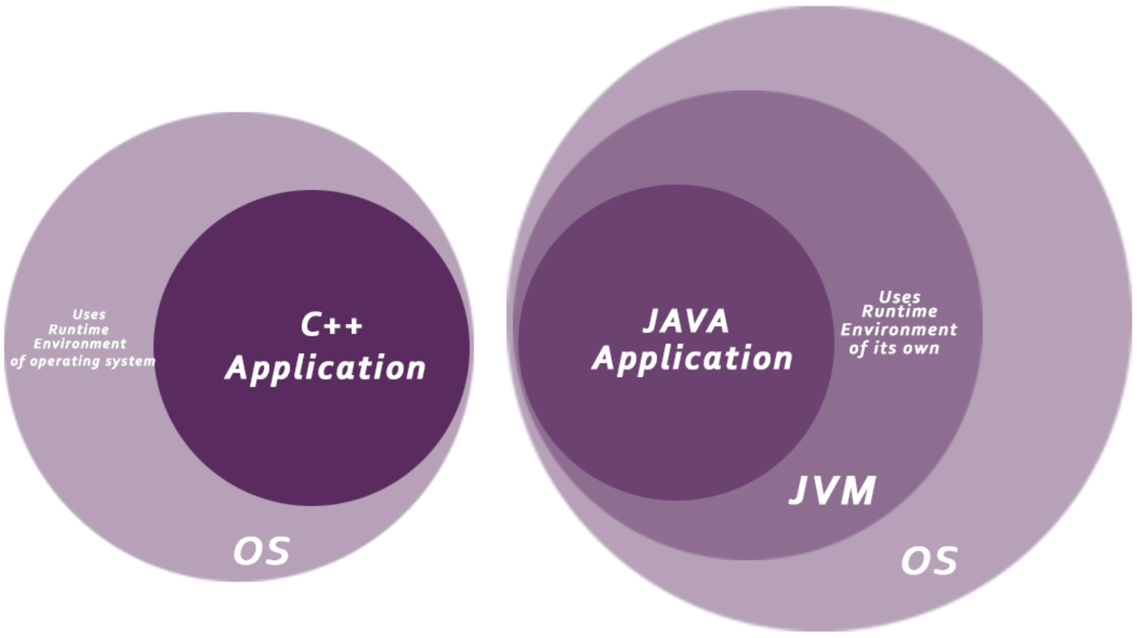
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

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

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

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


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

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

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

Robust

Robust simply means strong. Java is robust because:

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

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

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

High-performance

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

Distributed

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

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

Java supports dynamic compilation and automatic memory management (garbage collection).

How Java Differs From C:

Java and C

Java is not lot like C but the major difference between Java and C is that Java is and object-oriented language and has mechanism to define classes and objects. In an effort to build a simple and safe language, the Java team did not include some of the C features in Java.

- Java does not include the C unique statement keywords **sizeof,** and **typedef.**
- Java does not contain the data type **struct** and **union.**
- Java does not define the type modifiers keywords **auto,extern,register,signed,** and **unsigned.**
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use # **define,** # **include,** and # **ifdef** statements.

- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the **void** keyword as done in C.
- Java adds new operators such as **instanceof** and >>>.
- Java adds labelled **break** and **continue** statements.
- Java adds many features required for object-oriented programming.

How Java Differs From C++ :

Java and C++

Java is a true object-oriented language while C++ is basically C with object-oriented extension. That is what exactly the increment operator ++ indicates. C++ has maintained backward compatibility with C. Is is therefore possible to write an old style C program and run it successfully under C++. Java appears to be similar to C++ when we consider only the "extensions" part of C++. However, some object -oriented features of C++ make the C++ code extremely difficult to follow and maintain.

Listed below are some major C++ features that were intentionally omitted from java or significantly modified.

- Java does not support operator overloading.
- Java does not have template classes as in C++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called "Interface".
- Java does not support global variables. Every variable and method is declared within classes and forms part of that class.
- Java does not use pointers.
- Java has replaced the destructor function with a finalize() function.
- There are no header files in Java.

Java also adds some new features. While C++ is a superset of C, Java is neither a superset nor a subset of C or C++. Java may be considered as a first cousin of C++ and a second cousin of C

Differences among C, C++ and Java Programming Languages: C vs C++ vs Java

The purpose of learning a programming language is to become a better programmer i.e. to become more effective at designing and implementing new systems and at maintaining old ones.

C, C++, and Java are the most popular programming languages used today at a broad level. They have a pretty similar syntax for basic concepts. Most of the basic constructs like if statements, loops, function syntax, switch case statements and concepts like recursion are still valid. Many other concepts like the syntax for comments, and the idea of static class variables, also held in both Java and C++.

Java uses the syntax of C and structure of C++ language.

Aspects	**C**	**C++**	**Java**

The developed year	1972	1979	1991
Developed By	Dennis Ritchie	Bjarne Stroustrup	James Gosling
Successor of	BCPL	C	C(Syntax) & C++ (Structure)
Paradigms	Procedural	Object Oriented	Object Oriented
Platform Dependency	Dependent	Dependent	Independent
Keywords	32	63	50 defined (goto, const unusable)
Datatypes: union, structure	Supported	Supported	Not Supported
Pre-processor directives	Supported (#include, #define)	Supported (#include, #define)	Not Supported
Header files	Supported	Supported	Use Packages (import)
Inheritance	No Inheritance	Supported	Multiple Inheritance not Supported
Overloading	No Overloading	Supported	Operator Overloading not Supported
Pointers	Supported	Supported	No Pointers
Code Translation	Compiled	Compiled	Interpreted
Storage Allocation	Uses malloc, calloc	Uses new, delete	uses garbage collector
Multithreading and Interfaces	Not Supported	Not Supported	Supported
Exception Handling	No Exception handling	Supported	Supported
Templates	Not Supported	Supported	Not Supported
Storage class: auto, extern	Supported	Supported	Not Supported
Destructors	No Constructor or Destructor	Supported	Not Supported
Database Connectivity	Not Supported	Not Supported	Supported

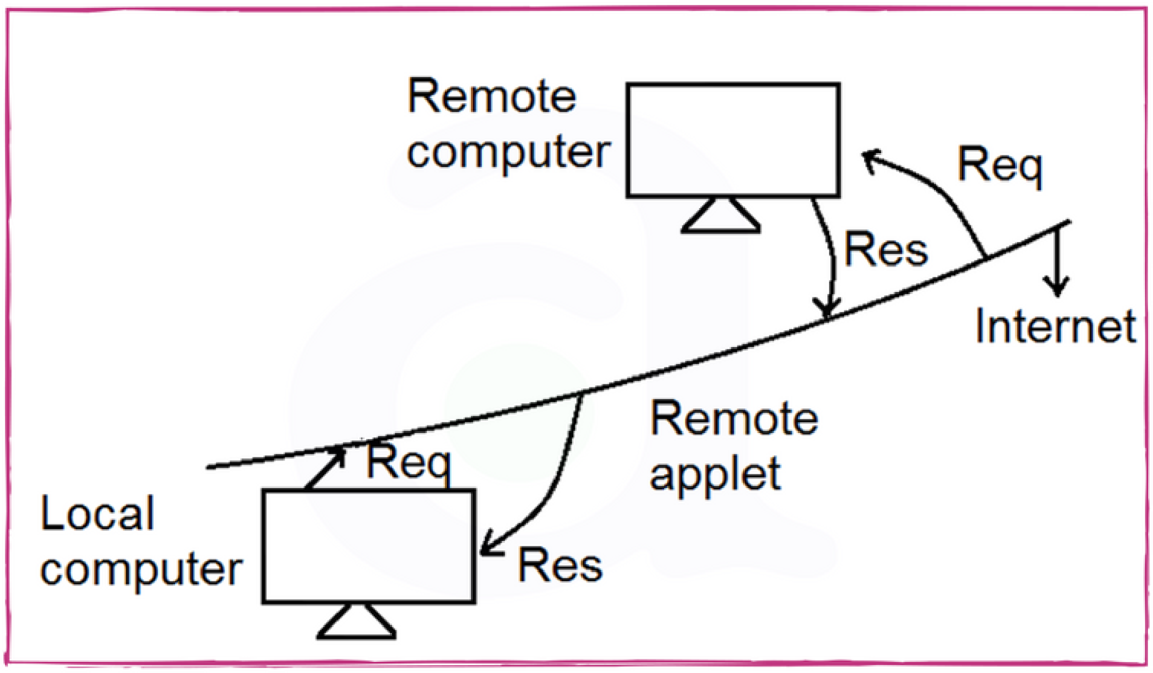
Java and the internet:

Java is strongly associated with the internet because of the first application program is written in Java was hot Java.

Web browsers to run applets on the internet.

Internet users can use Java to create applet programs & run then locally using a Java-enabled browser such as hot Java.

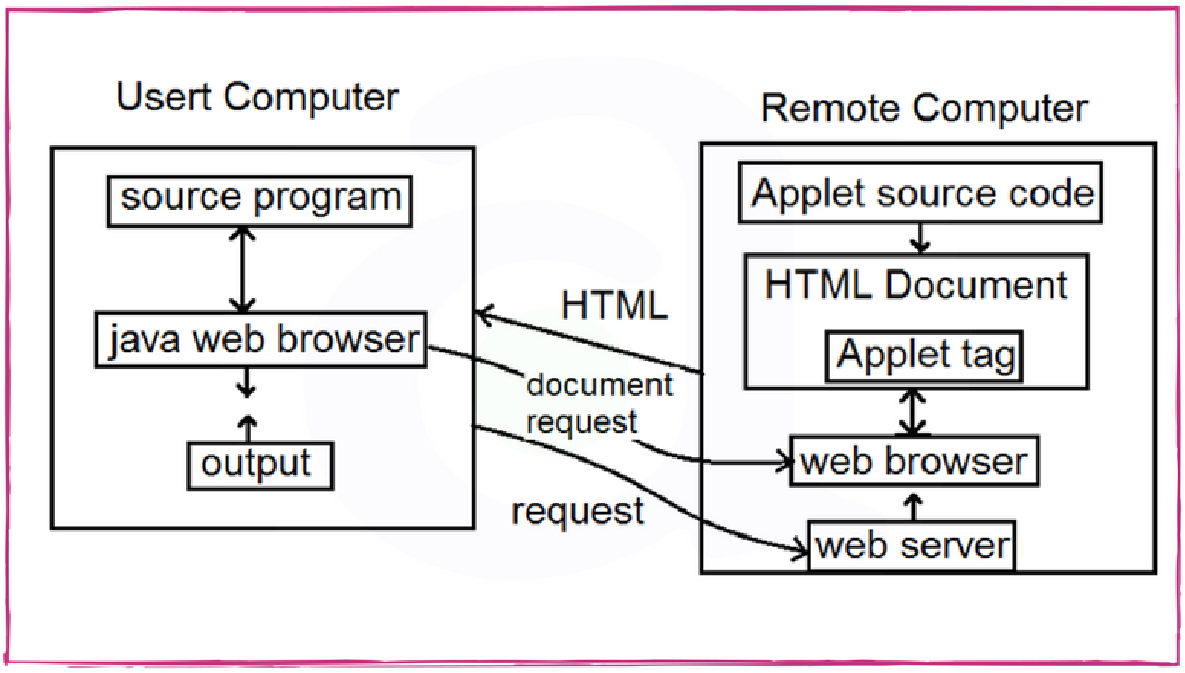
Java applets have made the internet a true extension of the storage system of the local computer.



Java and Internet

World wide web and internet

1. World wide web is a collection of information stored on internet computers.
2. World wide web is an information retrieval system designed to be used in the internet's distributed environment.
3. World wide web contains web pages that provide both information and controls.
4. Web pages contain HTML tags that enable us to find retrieve, manipulate and display documents world wide.
5. Before Java, the world wide web was limited to the display of still images & texts.
6. With the help of Java WWW is capable of supporting animation graphics, games and wide rage special effects.



Java and World Wide Web (www)

1. Java communicates with a web page through a special tag called
2. Java user sends a request for an HTML document to the remote computers net browser.
3. The web-browser is a program that accepts a request, processes the request and sends the required documents.
4. The HTML document is returned to that user browser.
5. The document contains the applet tag which identifies the applet. The corresponding applet is transferred to the user computer.
6. The Java enabled browser on the user's computer interprets the byte code and provide output.

If you have passion to learn Java programming and you want to write and test your own Java codes, then first of all you will require downloading a Java environment on your computer. Here we have brought you some of the essential hardware and software configuration environment which is required with any operating system.

Hardware Requirement for Java

Minimum hardware requirement to download Java on your Windows operating system as follows:

- Minimum Windows 95 software
- IBM-compatible 486 system
- Hard Drive and Minimum of 8 MB memory

- A CD-ROM drive
- Mouse, keyboard and sound card, if required

Software requirement for Java

Nowadays, Java is supported by almost every operating systems. whether it is a Windows, Macintosh and Unix all supports the Java application development. So you can download any of the operating system on your personal computer. Here are the minimum requirement.

- Operating System
- Java SDK or JRE 1.6 or higher
- Java Servlet Container (Free Servlet Container available)
- Supported Database and library that supports the database connection with Java.

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Following are the steps to set up the environment.

Java SE is freely available from the link Download Java. You can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories −

Setting Up the Path for Windows

Assuming you have installed Java in *c:\Program Files\java\jdk* directory −

- Right-click on 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc: export PATH = /path/to/java:$PATH'

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following −

- **Notepad** − On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** − A Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.

- **Eclipse** − A Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

3. OVERVIEW OF JAVA LANGUAGE:

Java programs structure

It is necessary to know the exact structure of the Java program, and this lesson contains a detailed description of it. This lesson is essential for you before proceeding to learn more advanced lessons of Java programming. Here, in this chapter, you will study the structure of the Java program. Such as how to create a simple Java program and what its different sections mean.

A Java program involves the following sections:

- Documentation Section
- Package Statement
- Import Statements
- Interface Statement
- Class Definition
- Main Method Class
  - Main Method Definition

Section	Description
Documentation Section	You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional, but we suggest you use them because they are useful to understand the operation of the program, so you must write comments within the program.
Package statement	You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors. Here, the package is a keyword that tells the compiler that package has been created. It is declared as: `package package_name;`

Import statements	This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program. Example: `import calc.add;`
Interface statement	Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
Class Definition	A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.
Main Method Class	Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements.

Here is an example of the Hello Java program to understand the class structure and features. There are a few lines in the program, and the primary task of the program is to print *Hello Java* text on the screen.

A Simple Java Program to Print "Hello Java"

Example:

```
//Name of this file will be "Hello.java"



public class Hello

{

   /* Author: www.javaexams.in

   Date: 2020-01-01

   Description:

   Writes the words "Hello Java" on the screen */

   public static void main(String[] args)

   {
        System.out.println("Hello Java");

    }

}
```

Program Output:

```
Hello Java
```

Java Tokens:

A **token** is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Special Symbols
5. Operators

1. **Keyword:** Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. **Java** language supports following keywords:

2.			
3.	**abstract**	**assert**	**boolean**
4.	**break**	**byte**	**case**
5.	**catch**	**char**	**class**
6.	**const**	**continue**	**default**
7.	**do**	**double**	**else**
8.	**enum**	**exports**	**extends**
9.	**final**	**finally**	**float**
10.	**for**	**goto**	**if**
11.	**implements**	**import**	**instanceof**
12.	**int**	**interface**	**long**
13.	**module**	**native**	**new**
14.	**open**	**opens**	**package**
15.	**private**	**protected**	**provides**
16.	**public**	**requires**	**return**
17.	**short**	**static**	**strictfp**
18.	**super**	**switch**	**synchronized**
19.	**this**	**throw**	**throws**
20.	**to**	**transient**	**transitive**
21.	**try**	**uses**	**void**
22.	**volatile**	**while**	**with**

Identifiers: Identifiers are used as the general terminology for naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements.

Examples of valid identifiers :

```
MyVariable
MYVARIABLE
myvariable
x
i
x1
i1
_myvariable
$myvariable
sum_of_array
geeks123
```

Examples of invalid identifiers :

1.
   My Variable // contains a space
2. 123geeks   // Begins with a digit
3. a+c // plus sign is not an alphanumeric character
4. variable-2 // hyphen is not an alphanumeric character
5. sum_&_difference // ampersand is not an alphanumeric character
6.
7. **Constants/Literals:** Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals.
   Constants may belong to any of the data type.
   Syntax:
   final data_type variable_name;
8. **Special Symbols:** The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.
   [] () {}, ; * =

   - **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
   - **Parentheses():** These special symbols are used to indicate function calls and function parameters.
   - **Braces{}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

- **comma (,):** It is used to separate more than one statements like for separating parameters in function calls.
- **semi colon :** It is an operator that essentially invokes something called an initialization list.
- **asterick (*):** It is used to create pointer variable.
- **assignment operator:** It is used to assign values.

9. **Operators:** Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-
   0. Arithmetic Operators
   1. Unary Operators
   2. Assignment Operator
   3. Relational Operators
   4. Logical Operators
   5. Ternary Operator
   6. Bitwise Operators
   7. Shift Operators
   8. instance of operator
   9. Precedence and Associativity

Types of Statements in Java

A statement specifies an action in a Java program, such as assigning the sum of x and y to z, printing a message to the standard output, writing data to a file, etc.

Statements in Java can be broadly classified into three categories:

- Declaration statement
- Expression statement
- Control flow statement

Declaration Statement

A declaration statement is used **to declare a variable**. For example,

```
int num;
int num2 = 100;
String str;
```

Expression Statement

An **expression with a semicolon at the end** is called an expression statement. For example,

```
/Increment and decrement expressions
num++;
++num;
```

```
num--;
--num;

//Assignment expressions
num = 100;
num *= 10;

//Method invocation expressions
System.out.println("This is a statement");
someMethod(param1, param2);
```

Control Flow Statement

By default, all statements in a Java program are executed in the order they appear in the program. Sometimes you may want to execute a set of statements repeatedly for a number of times or as long as a particular condition is true.

All of these are possible in Java using control flow statements. If block, while loop and for loop statements are examples of control flow statements.

Java Virtual Machine (JVM) & its Architecture

What is JVM?

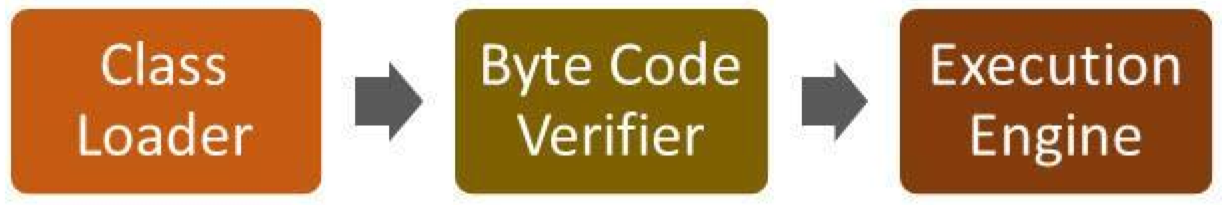
Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Run Environment (JRE). In other programming languages, the compiler produces machine code for a particular system. However,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

Here is how JVM works

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

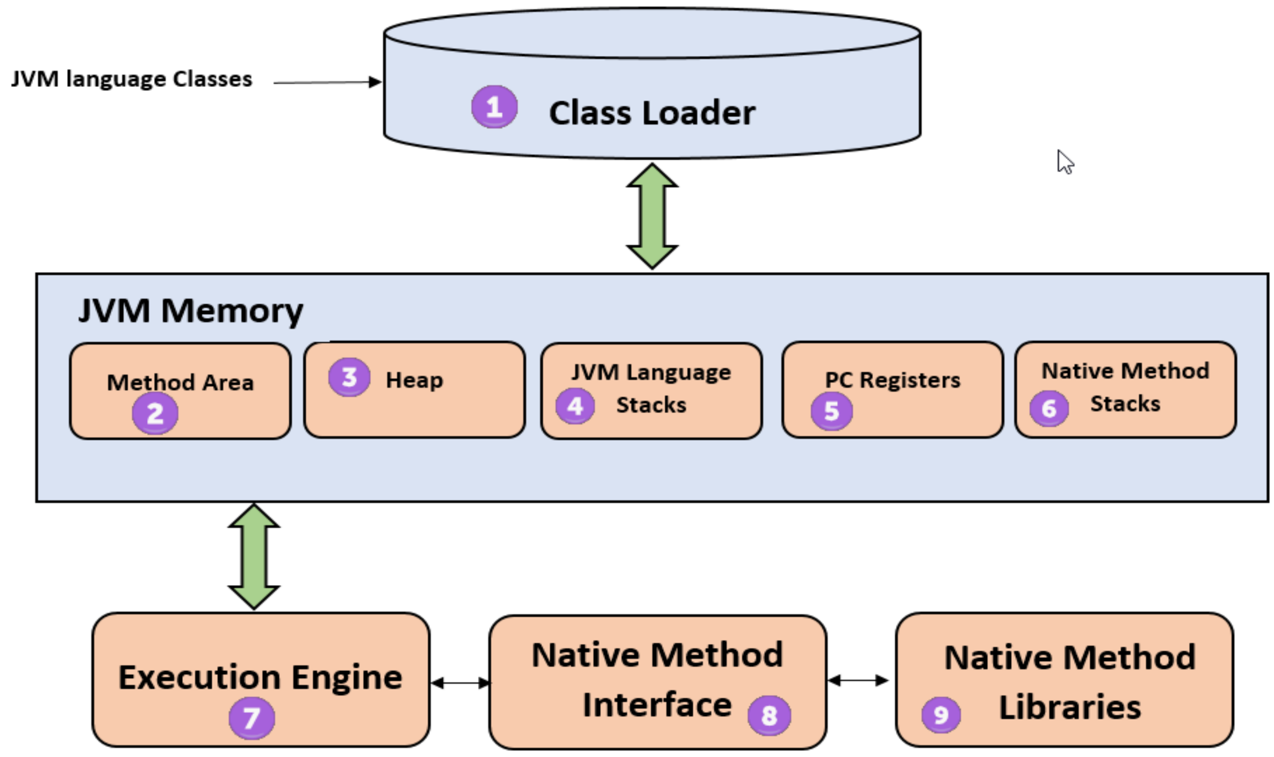
Between host system and Java source, Bytecode is an intermediary language.

JVM is responsible for allocating memory space.



JVM Architecture

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



1) ClassLoader

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

2) Method Area

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

3) Heap

All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.

4) JVM language Stacks

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

5) PC Registers

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

6) Native Method Stacks

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

7) Execution Engine

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

8) Native Method interface

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

9) Native Method Libraries

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

What is a Constant in Java?
A constant in Java is used to map an exact and unchanging value to a variable name.

Constants are used in programming to make code a bit more robust and human readable. Here's an example:

Imagine you are creating a program that needs to calculate areas and volumes of different shapes, it could look something like this, but this is an example of **WHAT NOT TO DO**:

```
public class AreasAndVolumes
{
  public double volumnOfSphere (double radius)
  {
    return (4/3) * Math.pow(3.14159 * radius, 3);
  }

  public double volumeOfCylinder (double radius, double height)
  {
    return Math.pow(radius * 3.14159, 2) * height;
  }

  public double areaOfCircle (double radius)
  {
    return Math.pow(radius * 3.14159, 2);
  }
}
```

So, this above code will get the job done, but there's something that can be improved. Can you guess how?

We should be using a **constant**! Look how many times we use the double value 3.14159, this value represents pi (π). We should create a constant that will assign the value of π to a variable name. Here's how we do it with some new code:

```
public class AreasAndVolumes
{
  // here we've declared a new variable called PI and assigned
  //  the value of 3.14159 to this new variable.
  private static final double PI = 3.14159;

  public double volumnOfSphere (double radius)
  {
    return (4/3) * Math.pow(PI * radius, 3);
  }

  public double volumeOfCylinder (double radius, double height)
  {
    return Math.pow(radius * PI, 2) * height;
  }

  public double areaOfCircle (double radius)
  {
    return Math.pow(radius * PI, 2);
  }
}
```

So now, we have a variable named PI declared in the instance variable declaration space. We've done this because we need to use this new constant value throughout our AreasAndVolumes class. This constant will function just like any other instance variable with one main exception… we've made the value final.

The final Keyword

Here is the magic behind a constant value. When you declare a variable to be final we are telling Java that we will NOT allow the variable's "pointer" to the value to be changed. That last sentence is key to understanding constants, did you read it thoroughly? If not, **re-read it**.

What the final keyword means is that once the value has been assigned, it cannot be re-assigned. So if you tried to put in some code later that tries to do this:

PI = 3.14159265359

You would get a compilation error, and your IDE would tell you that a new value cannot be assigned because the variable has been declared final.

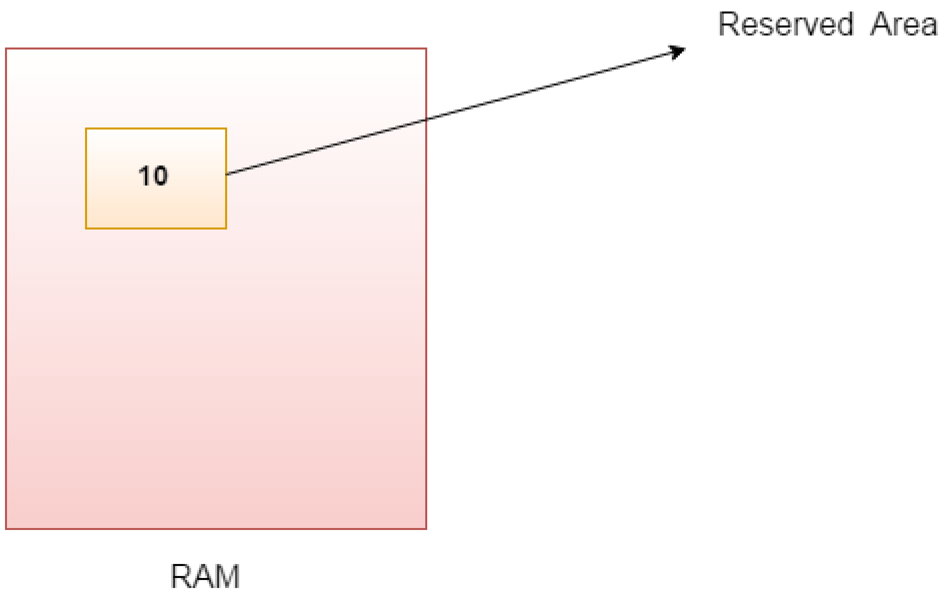
Java Variables

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

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.

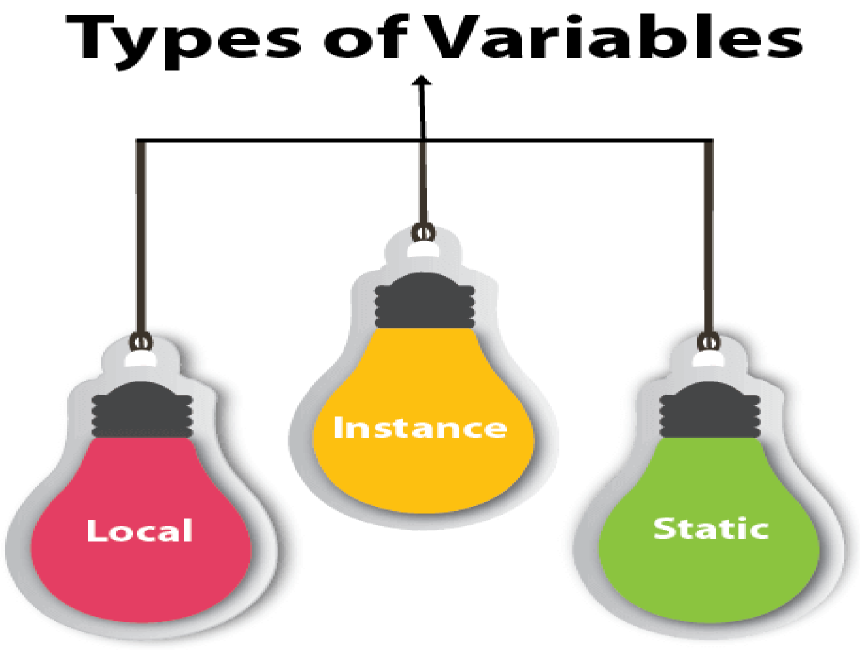
Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



1. **int** data=50;//Here data is variable

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 instance variable. It is not declared as static.

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

3) Static variable

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

Example to understand the types of variables in java

```java
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Java Variable Example: Add Two Numbers

```java
class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}}
```

Output:

```
20
```

Java Variable Example: Widening

```java
class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

```
10
10.0
```

Java Variable Example: Narrowing (Typecasting)

```
class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}}
```

Output:

```
10.5
10
```

Java Variable Example: Overflow

```
class Simple{
public static void main(String[] args){
//Overflow
int a=130;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
class Simple{
public static void main(String[] args){
byte a=10;
byte b=10;
//byte c=a+b;//Compile Time Error: because a+b=20 will be int
byte c=(byte)(a+b);
System.out.println(c);
}}
```

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.

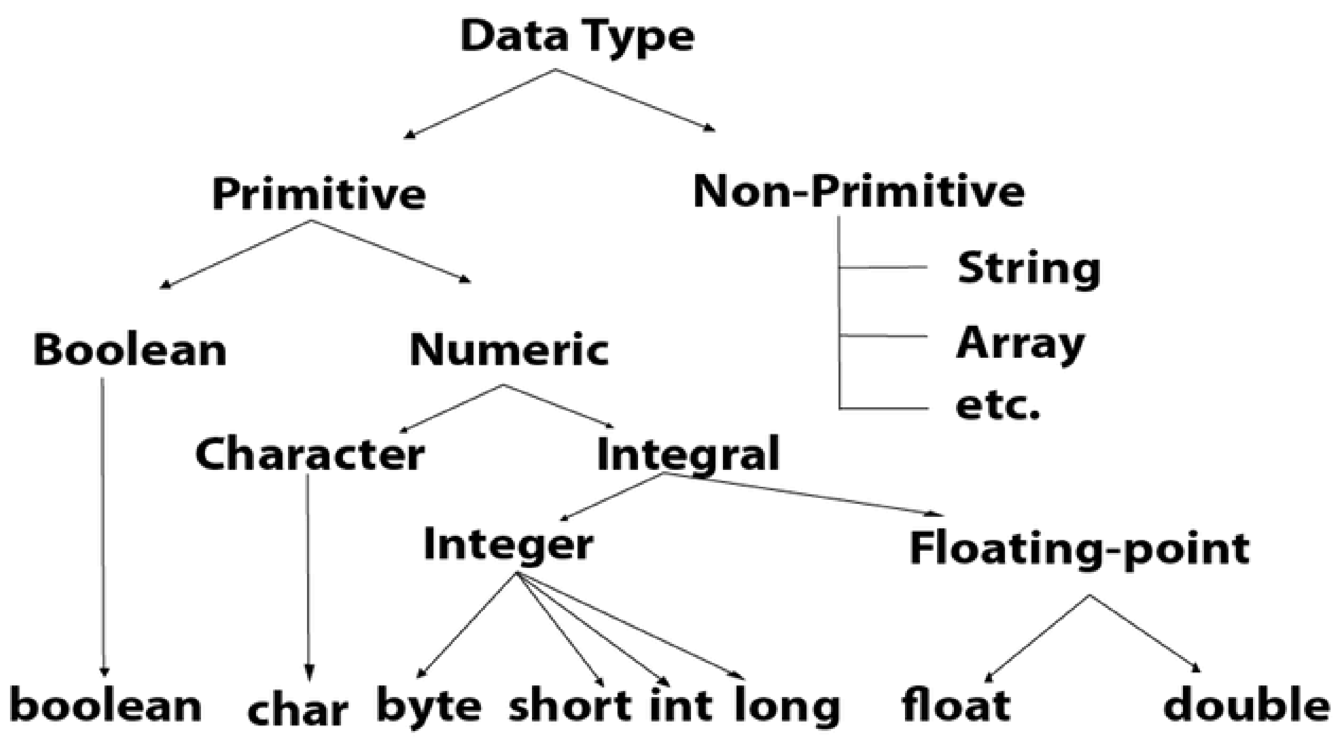
Java Primitive Data Types

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

> Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

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



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

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

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

Example: Boolean one = false

Byte Data Type

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

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

Example: byte a = 10, byte b = -20

Short Data Type

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

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

Example: short s = 10000, short r = -5000

Int Data Type

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

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

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

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

Float Data Type

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

Example: float f1 = 234.5f

Double Data Type

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

Example: double d1 = 12.3

Char Data Type

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

Example: char letterA = 'A'

Scope of Variables In Java

Scope of a variable is the part of the program where the variable is accessible. Like C/C++, in Java, all identifiers are lexically (or statically) scoped, i.e.scope of a variable can determined at compile time and independent of function call stack.
Java programs are organized in the form of classes. Every class is part of some package. Java scope rules can be covered under following categories.

Member Variables (Class Level Scope)

These variables must be declared inside class (outside any function). They can be directly accessed anywhere in class. Let's take a look at an example:

```
public class Test
{
   // All variables defined directly inside a class
   // are member variables
   int a;
   private String b
   void method1() {....}
   int method2() {....}
   char c;
}
```

- We can declare class variables anywhere in class, but outside methods.
- Access specified of member variables doesn't effect scope of them within a class.
- Member variables can be accessed outside a class with following rules

Modifier	**Package**	**Subclass**	**World**
public	Yes	Yes	Yes
protected	Yes	Yes	No
Default (no			

modifier)	Yes	No	No
private	No	No	No

Local Variables (Method Level Scope)

Variables declared inside a method have method level scope and can't be accessed outside the method.

```
public class Test
{
   void method1()
   {
     // Local variable (Method level scope)
     int x;
   }
}
```

Note : Local variables don't exist after method's execution is over.
Here's another example of method scope, except this time the variable got passed in as a parameter to the method:

```
class Test
{
   private int x;
   public void setX(int x)
   {
      this.x = x;
   }
}
```

The above code uses this keyword to differentiate between the local and class variables.
As an exercise, predict the output of following Java program.

```
public class Test
{
   static int x = 11;
   private int y = 33;
   public void method1(int x)
   {
      Test t = new Test();
      this.x = 22;
      y = 44;

      System.out.println("Test.x: " + Test.x);
      System.out.println("t.x: " + t.x);
      System.out.println("t.y: " + t.y);
```

```
        System.out.println("y: " + y);
    }

    public static void main(String args[])
    {
        Test t = new Test();
        t.method1(5);
    }
}
```

Output:

```
Test.x: 22

t.x: 22

t.y: 33

y: 44
```

Loop Variables (Block Scope)

A variable declared inside pair of brackets "{" and "}" in a method has scope withing the brackets only.

```
public class Test
{
    public static void main(String args[])
    {
        {
            // The variable x has scope within
            // brackets
            int x = 10;
            System.out.println(x);
        }

        // Uncommenting below line would produce
        // error since variable x is out of scope.

        // System.out.println(x);
    }
}
```

Output:

```
10
```

As another example, consider following program with a for loop.

```
class Test
{
    public static void main(String args[])
    {
        for (int x = 0; x < 4; x++)
        {
            System.out.println(x);
        }
        // Will produce error
        System.out.println(x);
    }
}
```

Output:

```
11: error: cannot find symbol
        System.out.println(x);                          ^
```

The right way of doing above is,

```
// Above program after correcting the error
class Test
{
    public static void main(String args[])
    {
        int x;
        for (x = 0; x < 4; x++)
        {
            System.out.println(x);
        }

       System.out.println(x);
    }
}
```

Output:

```
0
1
2
3
4
```

Let's look at tricky example of loop scope. Predict the output of following program. You may be surprised if you are regular C/C++ programmer.