Introduction to JAVA

Java is an Object Oriented Programming Language.

History of JAVA:

In 1991, a Group of Sun Engineers (Patrick Naughton & James Gosling) wanted to design a **computer language** used to write programs for consumer electronic devices.

These devices do not have lot of **power** or **memory**,

So,The language had to be **small** & generate very **tight code**.

Also,Different manufacturer may choose **different CPU**.

So,It was important that the language is **independent** of CPU **architecture**.This project was named as **"Green"**.

Accordingly,

The Sun team constructs a model that generates an **intermediate code**.

Then,

This intermediate code could be used on any machine that had the correct **interpreter**.

The model is called as **JVM** (Java Virtual Machine).

The Sun peoples are come from **UNIX** background.

So,

Their new language based on C++ **i.e. Object Oriented**

James Gosling decided to call this language as **"Oak"**.

But later,

The team realized that **Oak** was the name of an existing language,

So They changed the name to **"JAVA"**.

In 1992,

The Green project delivered its first version, called **"*7**

Unfortunately No one was interested in producing this device at sun

And Also,

No any standard consumer electronics companies were interested in it.

The team spends all 1993 and half 1994 looking for people to buy its technology. But no one found.

During same period,

The **World Wide Web** (WWW) part of Internet was growing bigger and bigger.

The key to this web is "**Browser**" that translates the hypertext page to web screen.

In 1994,

Gosling Team realized that they could build a **browser** that needs some things to be wired

i.e.

Architecture Neutral,

Real Time, Reliable,

and Secure.

So,

Patrick Naughton & Team build the browser known as "**HotJava**" browser.

This browser was written in Java to show the power of Java.

This browser also had Power of **Applet** and it is capable of executing

code inside web pages.

FEATURES OF JAVA

1) Simple

2) Object Oriented

3) Distributed

4) Reliable or Robust

5) Secure

6) Architecture Neutral

7) Portable

8) Interpreted

9) High Performance

10) Multithreaded

11) Dynamics

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

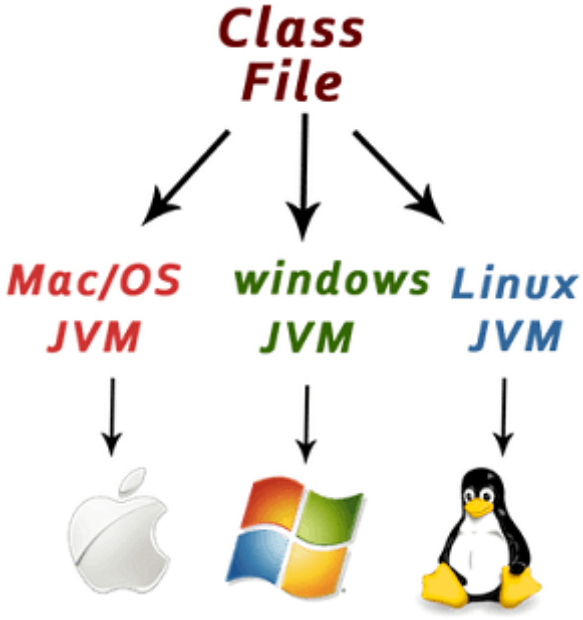
Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

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. A platform is the hardware or software environment in which a program runs.

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

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

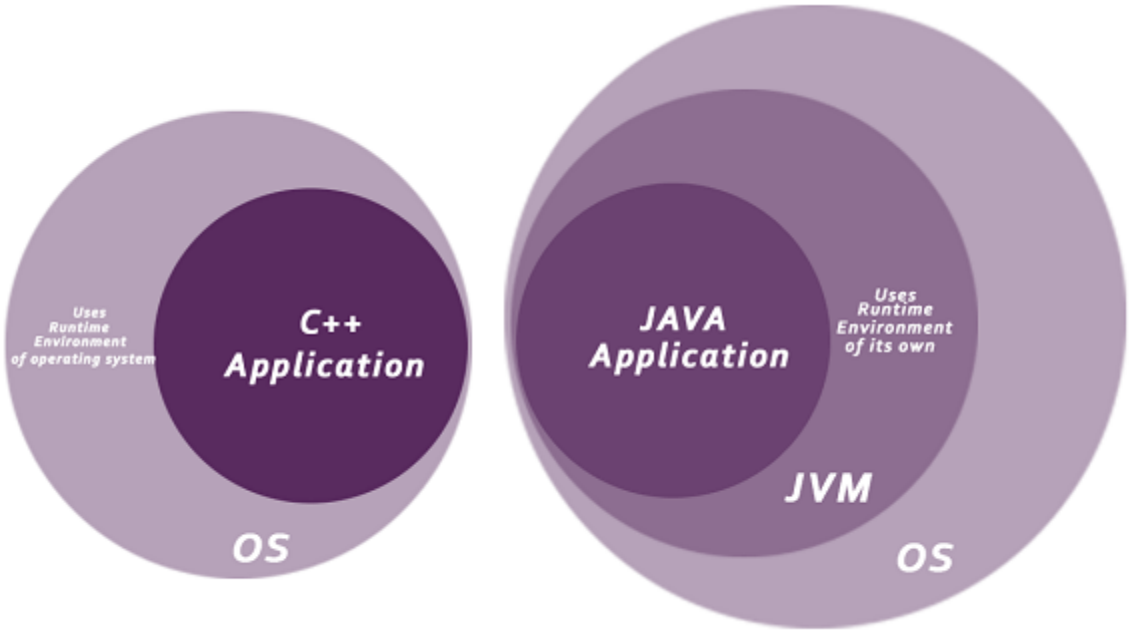
1. Runtime Environment
2. API(Application Programming Interface)

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

Secured

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

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



- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package

for the classes of the local file system from those that are imported from network sources.

- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

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

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.

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.

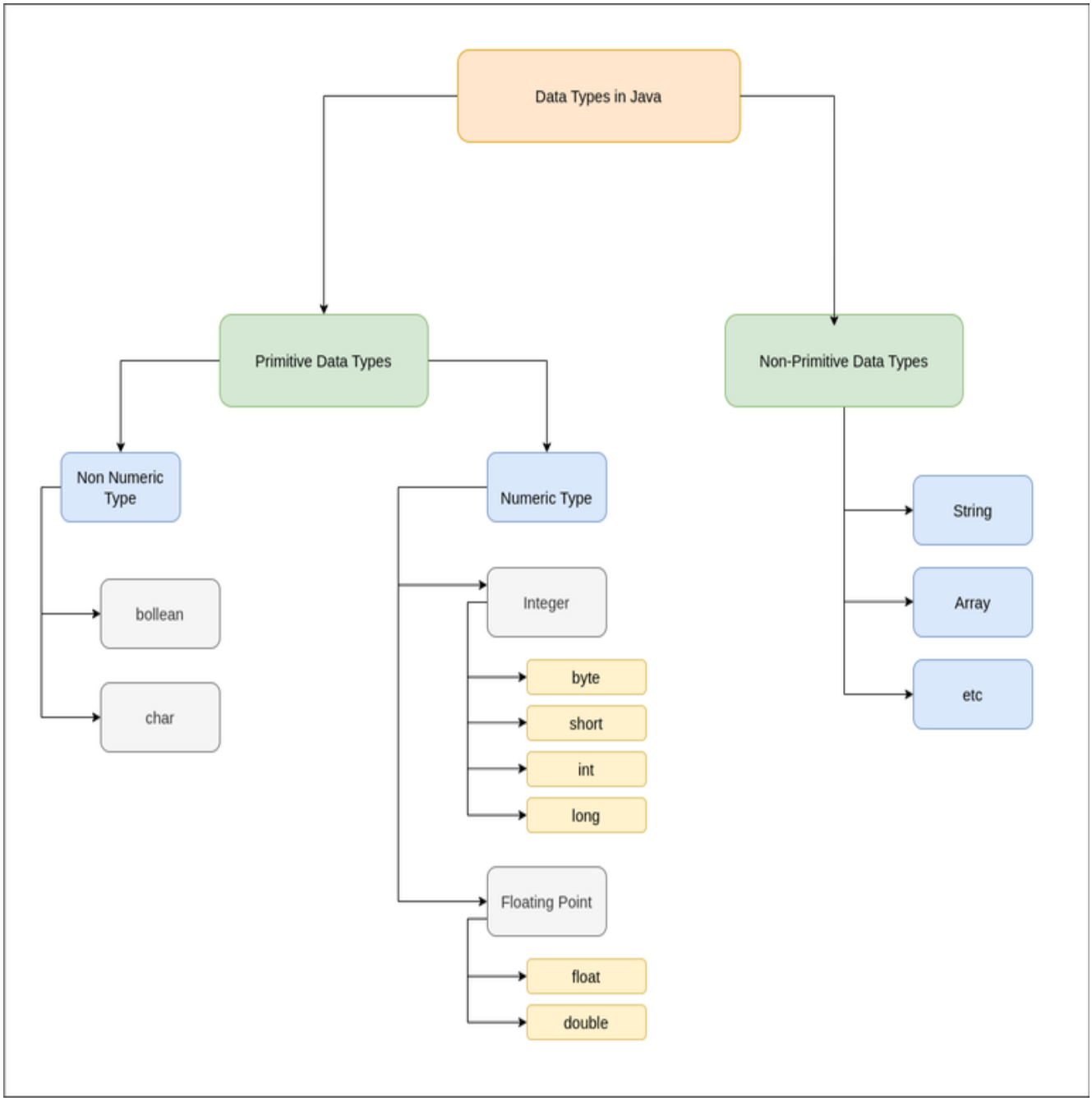
Dynamic

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

Java Data Types

- **Data types in Java** are of different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases.
  Java has two categories in which data types are segregated

1. **Primitive Data Type:** such as boolean, char, int, short, byte, long, float, and double
2. **Non-Primitive Data Type or Object Data type:** such as String, Array, etc.



Primitive Data Types in Java

Primitive data are only single values and have no special capabilities. There are 8 primitive data types. They are as follows:

Type	Description	Default	Size	Example Literals	Range of values
boolean	true or false	false	8 bits	true, false	true, false

Type	Description	Default	Size	Example Literals	Range of values
byte	twos-complement integer	0	8 bits	(none)	-128 to 127
char	Unicode character	\u0000	16 bits	‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’, ‘\n’, ‘β’	characters representation of ASCII values 0 to 255
short	twos-complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos-complement intger	0	32 bits	-2,-1,0,1,2	-2,147,483,648 to 2,147,483,647
long	twos-complement integer	0	64 bits	-2L,-1L,0L,1L,2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f , -1.23e-100f , .3f ,3.14F	upto 7 decimal digits
double	IEEE 754 floating	0.0	64 bits	1.23456e300d , -123456e-	upto 16 decimal digits

Type	Description	Default	Size	Example Literals	Range of values
	point			300d , 1e1d	

Non-Primitive Data Type or Reference Data Types

The **Reference Data Types** will contain a memory address of variable values because the reference types won't store the variable value directly in memory. They are strings, objects, arrays, etc.

1. Strings

Strings are defined as an array of characters.

The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char-type entities

. Unlike C/C++, Java strings are not terminated with a null character.

Syntax: Declaring a string

```
<String_Type> <string_variable> = "<sequence_of_string>";
```

Example:

```
// Declare String without using new operator
String s = "HelloABC";
// Declare String using new operator
String s1 = new String("HelloABC ");
```

2. 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.

3. Object

An Object is a basic unit of Object-Oriented Programming and represents real-life entities.

A typical Java program creates many objects, which as you know, interact by invoking methods.

4. Interface

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class..

- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

5. Array

An Array is a group of like-typed variables that are referred to by a common name.

Arrays in Java work differently than they do in C/C++.

The following are some important points about Java arrays.

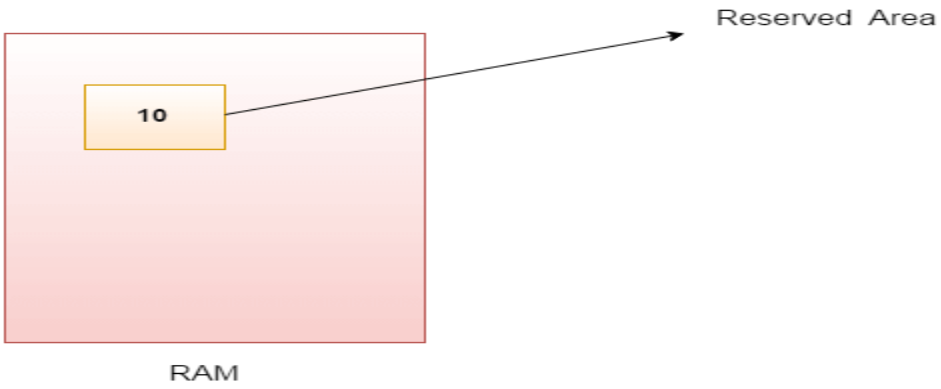
- In Java, all arrays are dynamically allocated. (discussed below)
- Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using size.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each has an index beginning with 0.
- Java array can also be used as a static field, a local variable, or a method parameter.
- The **size** of an array must be specified by an int value and not long or short.
- The direct superclass of an array type is Object.

Java Variables

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
- Variable is a name of memory location. There are three types of variables in java: local, instance and static.
- There are two types of data types in Java: primitive and non-primitive.

Variable

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

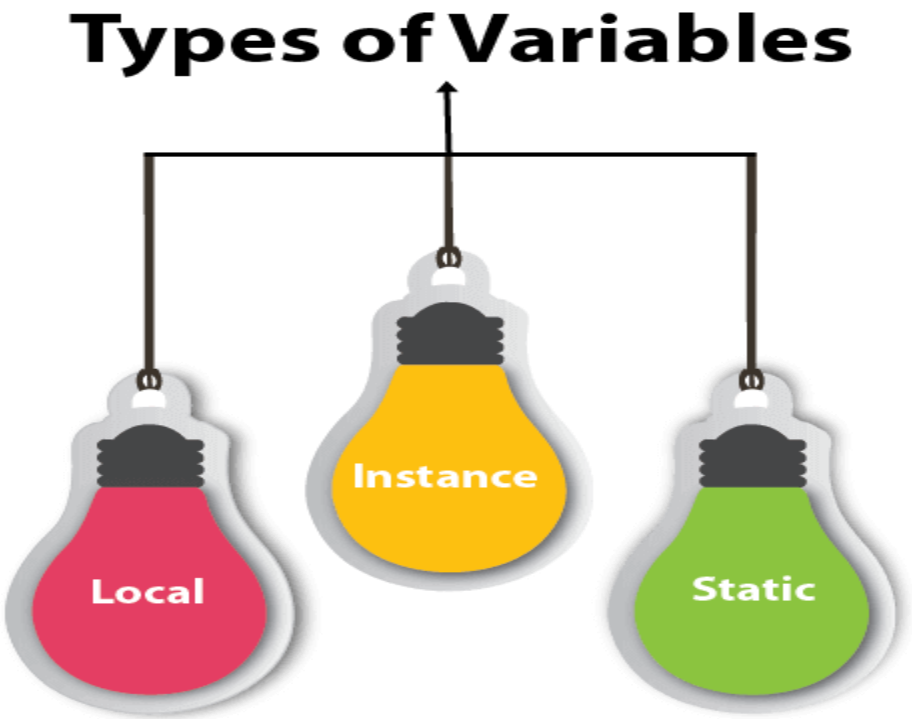


1. **int** data=50;//Here data is variable

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 an instance variable.
- It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

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

```
1. public class A
2. {
3.    static int m=100;//static variable
4.    void method()
5.    {
6.       int n=90;//local variable
7.    }
8.    public static void main(String args[])
9.    {
10.      int data=50;//instance variable
11.   }
12.}//end of class
```

Operators in Java

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

Java Operator Precedence

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

Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Structure of Java Program

It is essential to understand the structure of Java program.

Structure of Java Program

Documentation Section

- The documentation section in the structure of a Java program serves as a vital but optional part of a Java program, providing essential details about the program.

- This includes the author's name, creation date, version, program name, company name, and a brief description.
- While these details enhance the program's readability, the Java compiler ignores them during program execution.
- To include these details, programmers typically use comments.
- Comments are non-executable parts of a program. .
- There are three different types of comments- single-line, multi-line, and documentation comments.

- **Single line comment**- It is a comment that starts with // and is only used for a single line.

```
//Single-line comment
```

- **Multi line comment**- It is a comment that starts with /* and ends with */ and is used when more than one line has to be enclosed as comments.

```
/* Multiline comment
in Java */
```

- **Documentation comment**- It is a comment that starts with /** and ends with */

```
/** Documentation comment */
```

Package Declaration

- Declaring the package in the structure of Java is optional.
- It comes right after the documentation section.
- You mention the package name where the class belongs.
- Only one package statement is allowed in a Java program and must come before any class or interface declaration.
- This declaration helps organize classes into different directories based on the modules they're used in.
- You use the keyword package followed by the package name. For instance:

```
package  Employee;
```

Import Statements

- Import statements are used to import classes, interfaces, or enums that are stored in packages or the entire package.
- A package contains many predefined classes and interfaces.
  We need to mention which package we are using at the beginning of the program.
- We do it by using the import keyword.
- We either import the entire package or a specific class from that package.
- The following is the description of how we can write the import statement.

```
import java.util.*; //imports all the classes in util package
```

Explanation-

We have imported *java.util* package .

Interface Section

- This is an optional section.
- The keyword interface is used to create an interface.
- An interface comprises a set of cohesive methods that lack implementation details., i.e. method declaration and constants.

```
interface Code {
   void write();
   void debug();
}
```

Explanation-

In the above code, we have defined an interface named Code, which contains two method declarations, namely write() and debug(), with no method body.

The body of abstract methods is implemented in those classes that implement the interface Code.

Class Definition

- This is a mandatory section in the structure of Java program.

- Each Java program has to be written inside a class as it is one of the main principles of Object-oriented programming that Java strictly follows, i.e., its Encapsulation for data security.
- There can be multiple classes in a program.
- Some conventions need to be followed to name a class.
- They should begin with an uppercase letter.

```
class Program{
   // class definition
}
```

Main Method Class

- This is a compulsory part of the structure of Java program.
- This is the **entry point** of the compiler where the execution starts.
- It is called/invoked by the Java Virtual Machine or JVM.
- The main() method should be defined inside a class.
- We can call other functions and create objects using this method.
- The following is the syntax that is used to define.

Syntax

```
public static void main(String[] args) {
   // Method logic
}
```

e added two numbers passed as parameters and printed the result after adding them. This is only executed when the method is called.

The resulting structure typically resembles the following format when incorporating the components above into a Java program.

```
import java.io.*;

public class Main{

public static void main(String[] args) {

   System.out.println("Hello, Java!");
 }

}
```

Output-

```
Hello, Java!
```

Explanation-

In the above program, we printed a line on the console
using System.out.println(), which gets displayed after the code is executed.

JAVA DEVELOPMENT KIT (JDK):

- The **Java Development Kit** (**JDK**) is a Sun Microsystems product for Java developers.
- The primary components of the JDK are a selection of programming tools, like…
- **java: -** The loader for Java applications.
  - This tool is an interpreter and can interpret the class files generated by the javac compiler.
- **javac: -** The compiler, which converts source code into Java bytecode.
- **jar: -** The archiver, which packages related class libraries into a single JAR file. This tool also helps managing jar files (archived java class files).
- The JDK also contains complete Java Runtime Environment.
- It contains **JVM** and all of the class libraries.

JAVA VIRTUAL MACHINE (JVM):

- A **Java Virtual Machine** (**JVM**) is a set of software programs
- and data structures which used for the execution of other computer programs and scripts.
- The model used by a JVM accepts a form of computer intermediate language commonly referred to as **Java bytecode**.
- JVM operate on Java bytecode, generated from Java source code.
- JVM can also be used to implement programming languages other than Java.
- The JVM is a crucial component of the Java Platform.
- The use of the same bytecode for all platforms allows Java to be described as "compile once, run anywhere“.

- The Java Virtual Machine Specification defines an abstract machine or processor.
    - Once a Java virtual machine has been implemented for a given platform, any Java program (which, after compilation, is called bytecode) can run on that platform.
    - A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real processor instruction)

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
1. class CommandLineExample{
2. public static void main(String args[]){
3. System.out.println("Your first argument is: "+args[0]);
4. }
5. }
```

1. compile by > javac CommandLineExample.java
2. run by > java CommandLineExample abc

Output: Your first argument is: abc

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```
1. class A{
2. public static void main(String args[]){
```

```
3.
4. for(int i=0;i<args.length;i++)
5. System.out.println(args[i]);
6.
7. }
8. }
```

1. compile by > javac A.java
2. run by > java A Hello abc 1 3

```
Output: Hello
        abc
        1
        3
```

Java Arrays

Java array is an object which contains elements of a similar data type.

Additionally, The elements of an array are stored in a contiguous memory location.

It is a data structure where we store similar elements.

We can store only a fixed set of elements in a Java array.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array

An array that has **only one subscript** or one dimension is known as a single-dimensional array.

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=**new** datatype[size];

Example:

```
2. class Testarray1{
3. public static void main(String args[]){
4. int a[]={33,3,4,5};//declaration, instantiation and initialization
5. //printing array
6. for(int i=0;i<a.length;i++)//length is the property of array
7. System.out.println(a[i]);
8. }}
```

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

1. int[][] arr=new int[3][3];//3 row and 3 column

Example:

```
1. class Testarray3{
2. public static void main(String args[]){
3. //declaring and initializing 2D array
4. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
5. //printing 2D array
```

```
6.  for(int i=0;i<3;i++){
7.   for(int j=0;j<3;j++){
8.     System.out.print(arr[i][j]+" ");
9.   }
10. System.out.println();
11.}
12.}}
```

StringBuffer Class

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

> Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors of StringBuffer Class

Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

Method:

1) append()

- The append() method concatenates the given argument with this String.
- It is used to append the specified string with this string.
- The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

Syntax:

append(String s)

Example:

```
1. class StringBufferExample{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }
```

Output:

```
Hello Java
```

2) StringBuffer insert() Method

- It is used to insert the specified string with this string at the specified position.
- The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

Syntax

```
insert(int offset, String s)
```

Example:

```
1. class StringBufferExample2{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

Output:

```
HJavaello
```

3) replace()

- It is used to replace the string from specified startIndex and endIndex-1.

Syntax:

replace(int startIndex, int endIndex, String str)

Example:

```
1. class StringBufferExample3{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

Output:

HJavalo

4)delete() Method

The delete() method of the StringBuffer class deletes the String from the specified beginIndex to endIndex-1.

Syntax:

```
delete(int startIndex, int endIndex)
```

StringBufferExample4.java

```
1. class StringBufferExample4{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

Output:

```
Hlo
```

5) reverse()

The reverse() method of the StringBuilder class reverses the current String.

Syntax:

reverse()

StringBufferExample5.java

```
1. class StringBufferExample5{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

Output:

```
olleH
```

6)capacity()

- The capacity() method of the StringBuffer class returns the current capacity of the buffer.
- The default capacity of the buffer is 16.
- If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

Syntax:

capacity()

StringBufferExample6.java

```
1. class StringBufferExample6{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. }
10.}
```

Output:

```
16
16
34
```