

History of Java

- Java was developed by James Gosling, who is known as the father of Java, in 1995.
- The history of Java starts with the **Green Team**. Java team members (James Gosling, Mike Sheridan, and Patrick Naughton), initiated the java language project in June 1991. The small team of sun engineers called **Green Team**.
- Initially it was designed for small, embedded systems in electronic appliances like set-top boxes, televisions etc. However, it was best suited for internet programming.

History of Java

- The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic".
- Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe the history of Java.
- Firstly, it was called "**Greentalk**" by James Gosling, and the file extension was .gt .

History of Java



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

Why Java was named as "Oak"?

- Oak is a symbol of strength and chosen as its a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
- Java is an island in 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 a cup of coffee nearby his office.

History of Java

- Notice that Java is just a name, not an acronym.
- Finally JAVA was developed by James Gosling and his team at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called Java one of the Ten Best Products of 1995.
- 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.

Differences Between C, C++, and Java



- C is a general-purpose, structured, procedural, and high-level programming language developed by **Dennis MacAlistair Ritchie** in 1972 at **Bell Laboratories**.
- It is mainly used for system programming such as to develop the operating system, drivers, compilers, etc.
- The best-known example of the operating system that was developed using C language is Unix and Linux.

Features of C Language

- Machine independent and portable
- Modern Control Flow and Structure
- Rich set of operators
- Simple, Fast, and efficient
- Case-sensitive
- Low memory use
- Easily extendable
- Statically-typed (variable types are known at compile time)



C++

- C++ is an object-oriented, general-purpose, programming language developed by **Bjarne Stroustrup** at Bell Labs in 1979. It is based on C language or we can say that it is an extension of C language. It is used to develop high-performance applications.

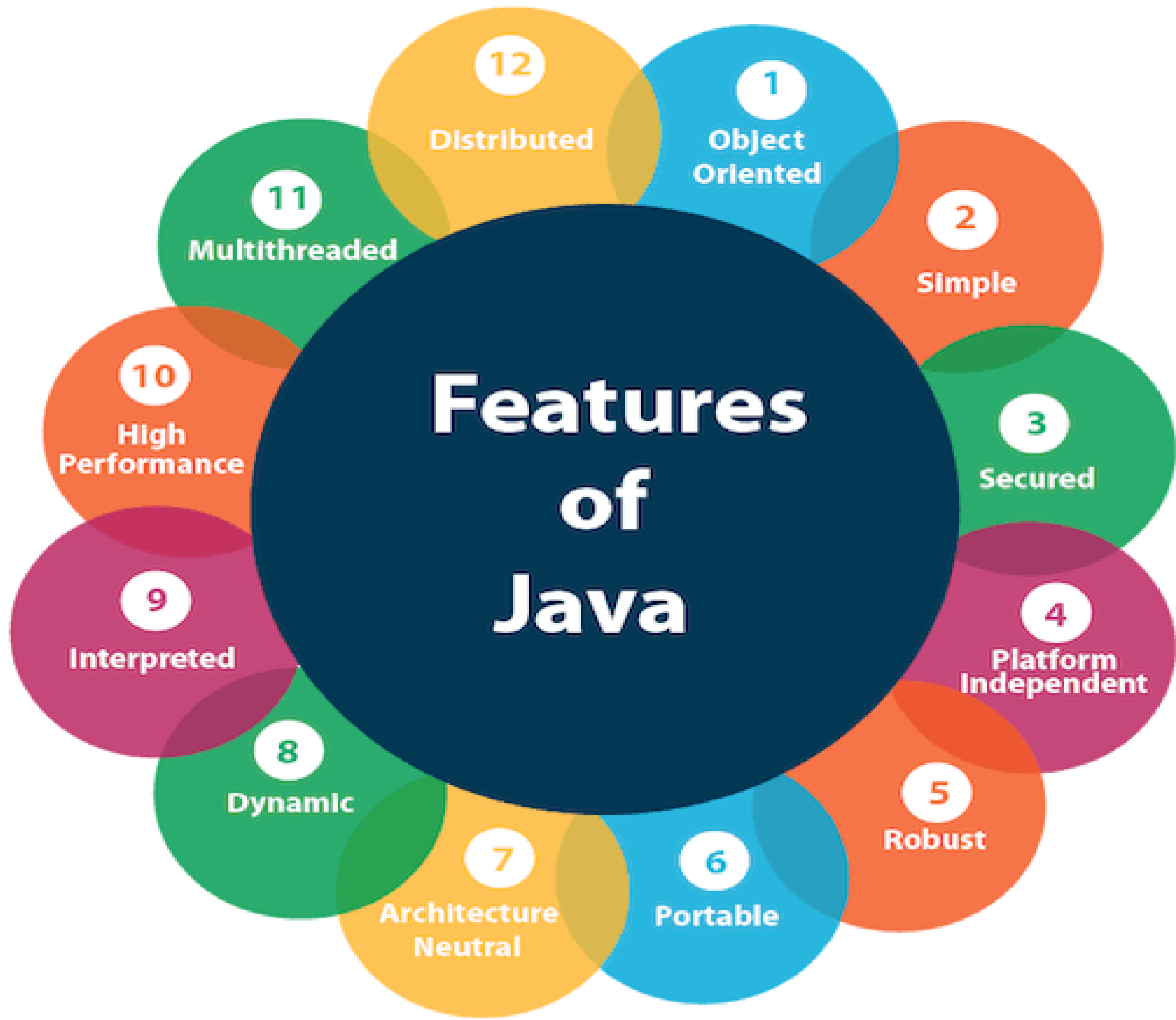
Features of C++ Language

- Case-sensitive
- Compiler based
- Platform-independent
- Portability
- Dynamic memory allocation



Java

- Java is also an object-oriented, class-based, static, strong, robust, safe, and high-level programming language. It was developed by **James Gosling** in 1995.
- Its code is compiled and interpreted. It is used to develop enterprise, mobile, and web-based applications.





Features of Java

➤ Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystems, 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.



Features of Java

➤ 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.

Java has a "binary code format" that's independent of hardware architectures, operating system interfaces, and window systems. The format of this system-independent binary code is architecture neutral.



Features of Java

➤ Object-oriented

Java is an object-oriented language and it incorporates the various characteristics of an object-oriented language, such as Class, Object, inheritance and polymorphism, abstraction, encapsulation.

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

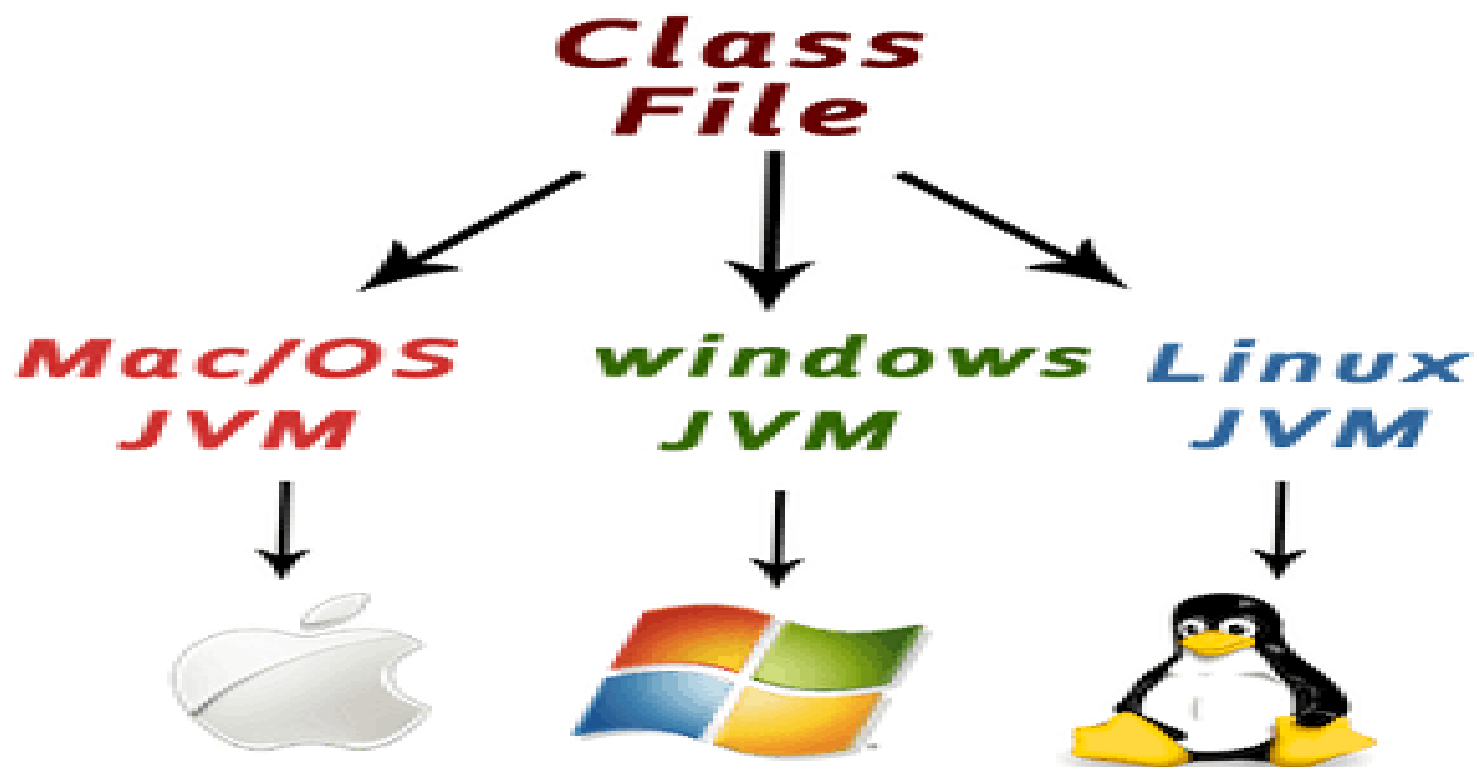
In Java, the entire program code must be encapsulated inside a class. Even the most basic program in Java must be written within a class.

➤ Platform independent

In Java, when you compile an error-free code, the compiler converts the program to a platform independent code, known as the bytecode. Thereafter, the Java Virtual Machine (JVM) interprets this bytecode into the machine code that can be run on that machine.

Features of Java

Converting a Java program into bytecode makes a Java program platform independent because any computer installed with the JVM can interpret and run the bytecode.



➤ Portable

Portability refers to the ability of a program to run on any platform without changing the source code of the program. The programs developed on one computer can run on another computer, which might have a different operating system. Java enables the creation of cross-platform programs by converting the programs into Java bytecode.

Features of Java

➤ Distributed

Java can be used for the development of those applications that can be distributed on multiple machines in a network, such as the Internet. These applications can be used by multiple users from multiple machines in a network. For this purpose, Java supports the various Internet protocols, such as Transmission Control Protocol and Internet Protocol (TCP/IP) and Hyper Text Transfer Protocol (HTTP).

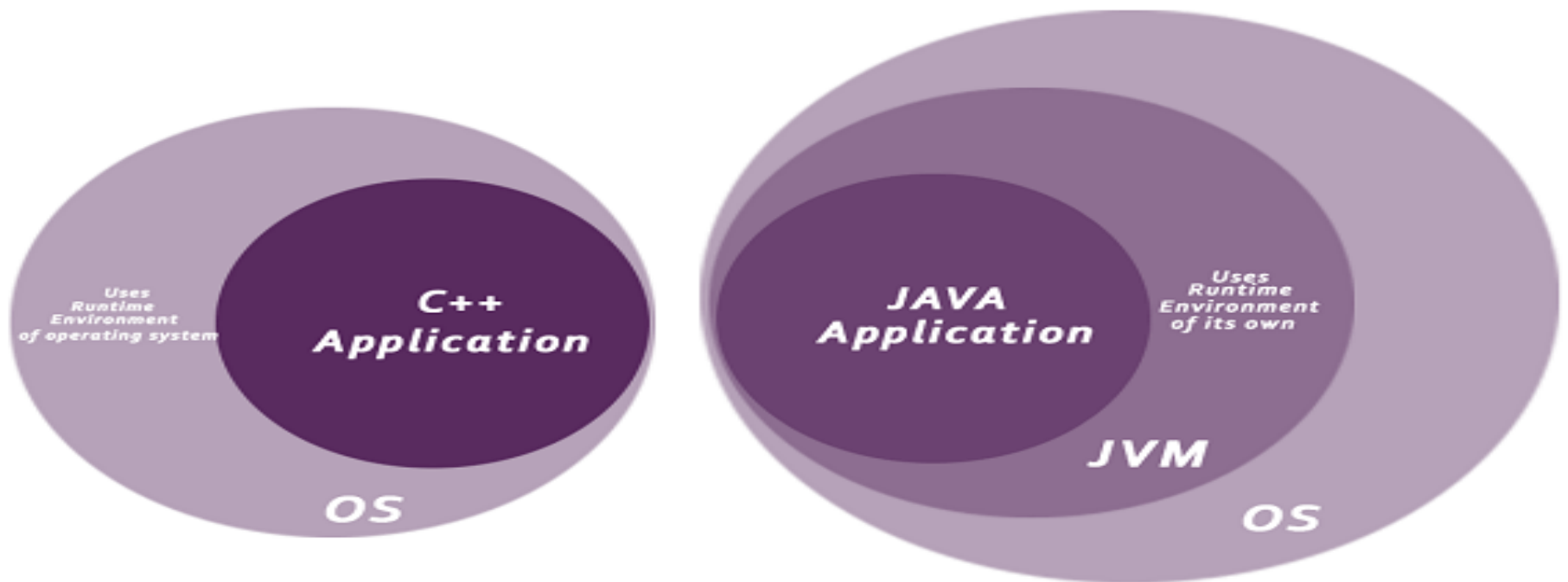
➤ Secure:

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

- ☐ **No explicit pointer**

- ☐ **Java Programs run inside a virtual machine sandbox**

Java program gains access to only those resources of a computer that are essential for its execution. This ensures that a Java program executing on a particular machine does not harm it



- **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.

Features of Java

➤ Robust:

The English meaning of Robust is strong.
Java is robust because:

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



Multithreaded:

Java provides the concept of multithreading that allows a program to simultaneously execute multiple tasks by using threads. A thread is the smallest unit of execution in a Java program.

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).

C vs C++ vs Java

The languages are based on each other but still, they are different in design and philosophy. The following table describes the major differences between C, C++, and Java. It will help you to select which language you have to learn.

S.N.	Basis	C	C++	Java
1	Origin	The C language is based on BCPL.	The C++ language is based on the C language.	The Java programming language is based on both C and C++.
2	Programming Pattern	It is a procedural language.	It is an object-oriented programming language.	It is a pure object-oriented programming language.
3	Approach	It uses the top-down approach.	It uses the bottom-up approach.	It also uses the bottom-up approach.
4	Dynamic or Static	It is a static programming language.	It is also a static programming language.	It is a dynamic programming language.

5	Code Execution	The code is executed directly.	The code is executed directly.	The code is executed by the JVM.
6	Platform Dependency	It is platform dependent.	It is platform dependent.	It is platform-independent because of byte code.
7	Translator	It uses a compiler only to translate the code into machine language.	It also uses a compiler only to translate the code into machine language.	Java uses both compiler and interpreter and it is also known as an interpreted language.
8	File Generation	It generates the .exe, and .bak, files.	It generates .exe file.	It generates .class file.
9	Number of Keyword	There are 32 keywords in the C language.	There are 60 keywords in the C++ language.	There are 52 keywords in the Java language.
10	Source File Extension	The source file has a .c extension.	The source file has a .cpp extension.	The source file has a .java extension.

11	Pointer Concept	It supports pointer.	It also supports pointer.	Java does not support the pointer concept because of security.
12	Union and Structure Datatype	It supports union and structure data types.	It also supports union and structure data types.	It does not support union and structure data types.
13	Pre-processor Directives	It uses pre-processor directives such as #include, #define, etc.	It uses pre-processor directives such as #include, #define, #header, etc.	It does not use directives but uses packages.
14	Constructor/ Destructor	It does not support constructor and destructor.	It supports both constructor and destructor.	It supports constructors only.
15	Exception Handling	It does not support exception handling.	It supports exception handling.	It also supports exception handling.

16	Memory Management	It uses the calloc(), malloc(), free(), and realloc() methods to manage the memory.	It uses new and delete operator to manage the memory.	It uses a garbage collector to manage the memory.
17	Overloading	It does not support the overloading concept.	Method and operator overloading can be achieved.	Only method overloading can be achieved.
18	goto Statement	It supports the goto statement.	It also supports the goto statement.	It does not support the goto statements.
19	Used for	It is widely used to develop drivers and operating systems.	It is widely used for system programming.	It is used to develop web applications, mobile applications, and windows applications.
20	Array Size	An array should be declared with size. For example, int num[10].	An array should be declared with size.	An array can be declared without declaring the size. For example, int num[].

Keywords: These are the **pre-defined** reserved words of any programming language. Each keyword has a special meaning. It is always written in lower case. Java provides the following keywords:

W

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. thro	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp



Java and Internet

Java is strongly associated with the Internet. Internet users can use Java to create applet programs and run them locally using a "Java-enabled browser" such as HotJava.

They can also use a Java-enabled browser to download an applet located on a computer anywhere in the Internet and run it on his local computer.

In fact, Java applets have made the Internet a true extension of the storage system of the local computer.

Internet users can also setup their websites containing java applets that could be used by other remote users of Internet. This feature made Java most popular programming language for Internet.

Java and World Wide Web

World Wide Web (WWW) is an open-ended information retrieval system designed to be used in the Internet's distributed environment. This system contains Web pages that provide both information and controls. Web system is open-ended and we can navigate to a new document in any direction. This is made possible with the help of a language called Hypertext Markup Language (HTML). Web pages contain HTML tags that enable us to find, retrieve, manipulate and display documents worldwide.

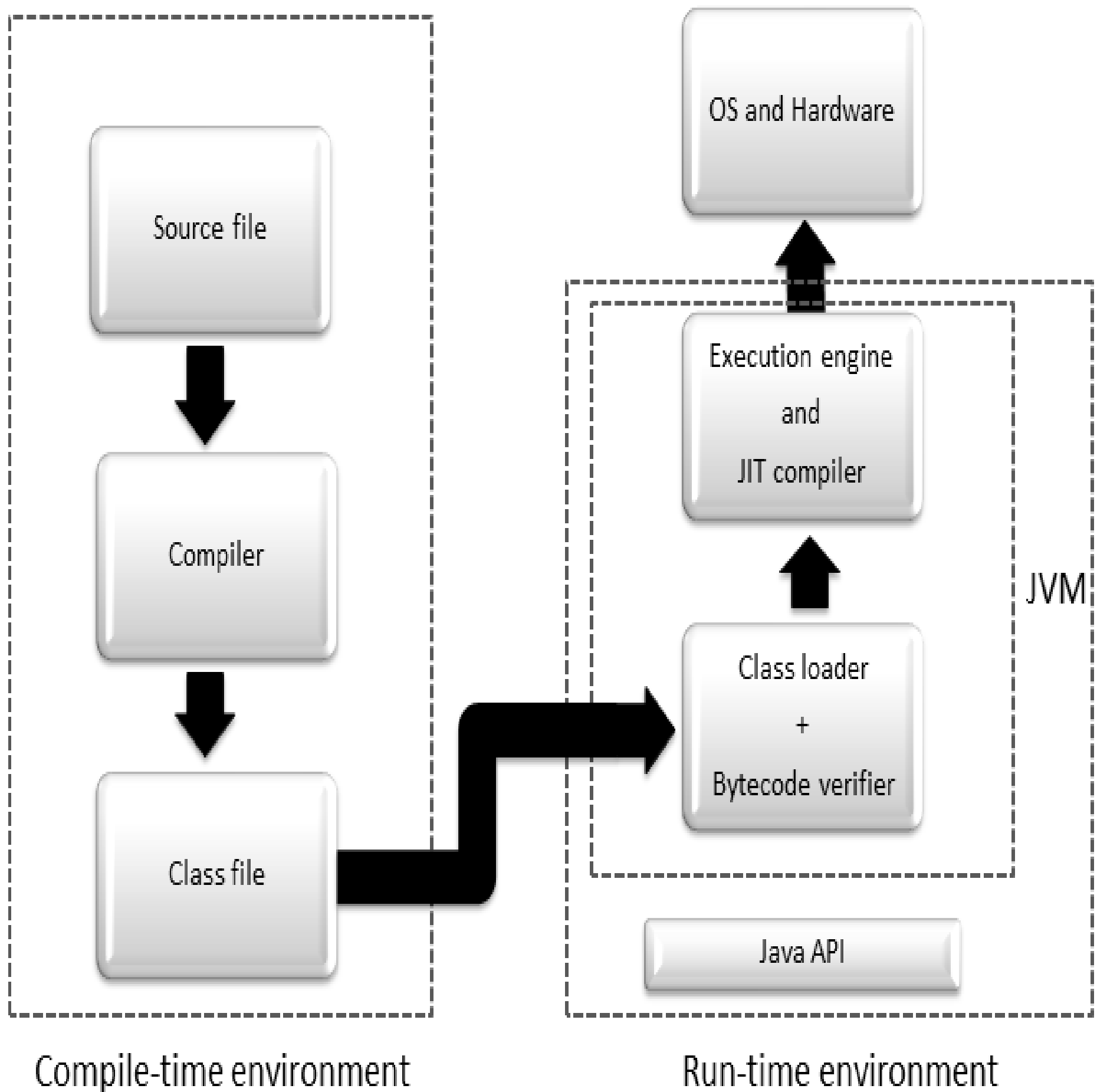
Java was meant to be used in distributed environments such as Internet. Java could be easily incorporated into the Web system. Before Java, the World Wide Web was limited to the display of still images and texts. However, the incorporation of Java into Web pages has made it capable of supporting animation, graphics, games, and a wide range of special effects.

Java Architecture

The Java architecture defines the components that are essential to carry out the creation and execution of code written in the Java programming language. The various components of the Java architecture are:

- Source file
- Class file
- JVM
- Application Programming Interface (API)

The various components in the Java architecture are shown in the following figure.





Source File

Java is a robust programming language that enables developers to build a wide variety of applications for varying purposes. In order to write a Java program or an application, you need to write certain code. A Java application contains the source code that enables an application to provide its desired functionalities. This source code is saved in a source file with the extension, .java.

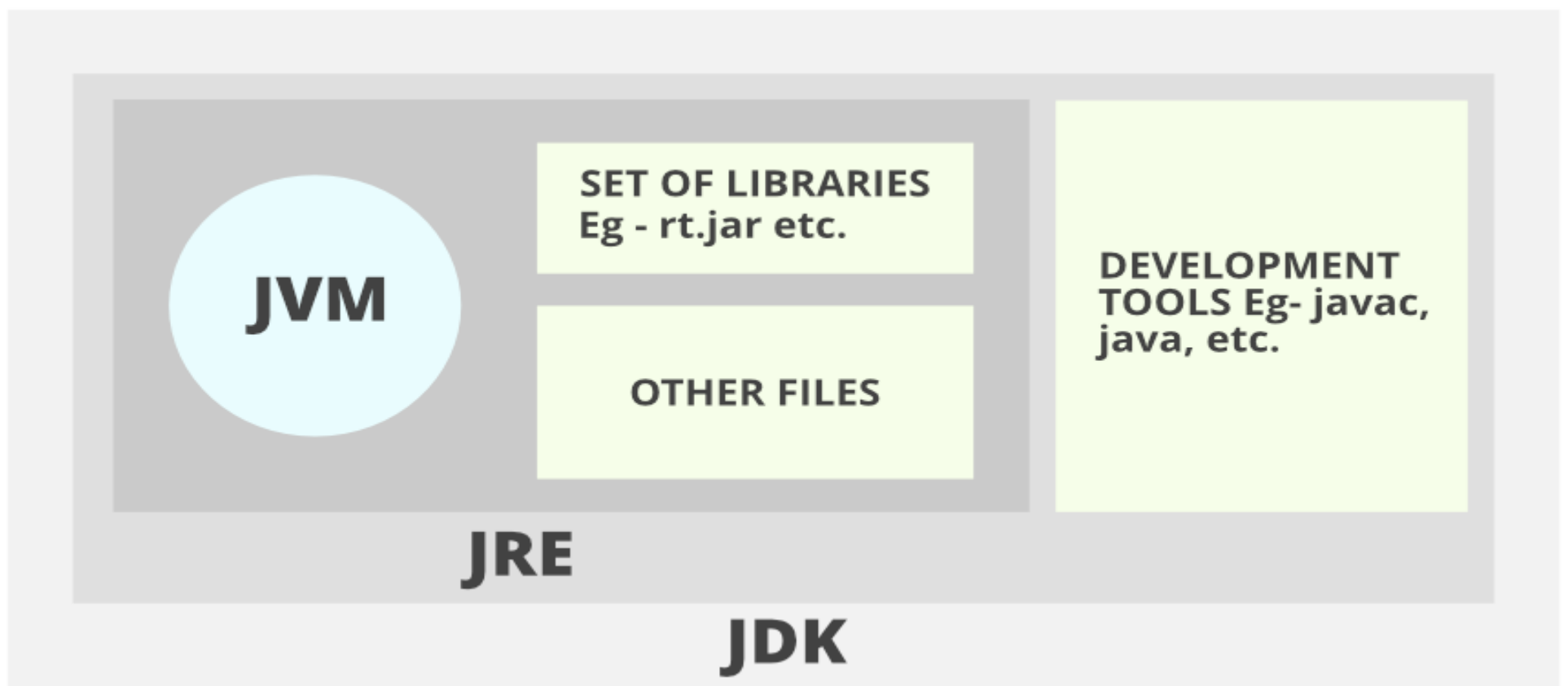
Class File

Once created, a .java file is compiled to generate the .class file. A class file contains the bytecode that is generated by the compiler. Further, this class file can be executed by any machine that supports the Java Runtime Environment (JRE).

Java Development Kit (JDK)

JDK is a software development environment used for developing Java applications and applets. **It includes :**

- Java Runtime Environment (JRE),
- an interpreter/loader (Java),
- a compiler (javac), an archiver (jar),
- a documentation generator (Javadoc), and other tools needed in Java development.



JRE (Java Runtime Environment)

Java Runtime Environment (JRE) is an open-access software distribution that has a Java class library, specific tools, and a separate JVM.

JRE is one of the interrelated components in the Java Development Kit (JDK). It is the most common environment available on devices for running Java programs. Java source code is compiled and converted to Java bytecode.

If we want to run this bytecode on any platform, you need JRE in turn JVM. The JRE loads classes, check memory access and get system resources. JRE acts as a software layer on top of the operating system.

JVM



The JVM is an application that is responsible for executing Java programs on a computer. It resides in the Random Access Memory (RAM) and is specific for every platform, such as Sun and Macintosh. The bytecode generated during the compilation of a program is the machine language of the JVM. JVM is responsible for executing the bytecode and generating the machine specific code for the machine on which the Java program needs to be executed.

The major components of JVM are:

- Class loader
- Execution engine and Just-In-Time (JIT) compiler
- Bytecode verifier



Class Loader

The class loader loads the class files required by a program that needs to be executed. The classes are loaded dynamically when required by the running program.

Execution Engine and JIT Compiler

The Java execution engine acts as an interpreter that interprets the bytecode one line after another and executes it. The line-by-line interpretation is a time-consuming task and reduces the performance of a program. To enhance the performance, the JIT compiler has been introduced in the JVM. The JIT compiler compiles the bytecode into machine executable code, as and when needed, which optimizes the performance of a Java program.



Bytecode Verifier

Before being executed by the JVM, the bytecode is verified by using the bytecode verifier. The bytecode verifier ensures that the bytecode is executed safely and does not pose any security threat to the machine.

API

The Java API is a collection of software components, such as classes, interfaces, and methods, which provides various capabilities, such as GUI, Date and Time, and Calendar. The related classes and interfaces of the Java API are grouped into packages.

Simple Java Program Structure

- Documentation (Comment) Section
- Package Statement
- Import Statement
- Class Definition
- Main Method Class

Documentation (Comment) Section

It is used to improve the readability of the program. The compiler ignores these comments during the time of execution and is solely meant for improving the readability of the Java program.

There are three types of comments that Java supports

- Single line Comment
- Multi-line Comment
- Documentation Comment

Documentation (Comment) Section

/ a single line comment is declared like this

/* a multi-line comment is declared like this
and can have multiple lines as a comment */

/** a documentation comment starts with a
delimiter and ends with */

Package Statement

There is a provision in Java that allows you to declare your classes in a collection called package. There can be only one package statement in a Java program and it has to be at the beginning of the code before any class or interface declaration. This statement is optional. Example:
`package student;`

Import Statement

Many predefined classes are stored in packages in Java, an import statement is used to refer to the classes stored in other packages. An import statement is always written after the package statement but it has to be before any class declaration.

We can import a specific class or classes in an import statement. Take a look at the example to understand how import statement works in Java.

Ex:-

```
import java.util.Date; //imports the date class
```

```
import java.applet.*;
```

```
//imports all the classes from the java applet package
```



Class Definition

A Java program may contain several class definitions, classes are an essential part of any Java program. It defines the information about the user-defined classes in a program.

A class is a collection of variables and methods that operate on the fields. Every program in Java will have at least one class with the main method.



Main Method

The main method is from where the execution actually starts and follows the order specified for the following statements. Let's take a look at a sample program to understand how it is structured.

```
public class Example{  
    //main method declaration  
    public static void main(String[] args){  
        System.out.println("hello world");  
    }  
}  
public class Example
```

This creates a class called Example. You should make sure that the class name starts with a capital letter, and the public word means it is accessible from any other classes.



Comments

To improve the readability, we can use comments to define a specific note or functionality of methods, etc for the programmer.

Curly Braces

The curly brackets are used to group all the commands together. To make sure that the commands belong to a class or a method.



public static void main

- When the main method is declared **public**, it means that it can be used outside of this class as well.
- The word **static** means that we want to access a method without making its objects. As we call the main method without creating any objects.
- The word **void** indicates that it does not return any value. The main is declared as void because it does not return any value.
- **Main** is the method, which is an essential part of any Java program.



String[] args

It is an array where each element is a string, which is named as args. If you run the Java code through a console, you can pass the input parameter. The main() takes it as an input.

System.out.println();

The statement is used to print the output on the screen where the system is a predefined class, out is an object of the PrintWriter class. The method println prints the text on the screen with a new line. All Java statements end with a semicolon.

Example :

Java Application having two classes in one file.

```
package comp
public class Computer
{
    Computer()
    {
        System.out.println("Constructor of Computer class.");
    }
    void computer_method()
    {
        System.out.println("Power gone! Shut down your PC soon...");
    }

    public static void main(String[] args)
    {
        Computer c = new Computer();
        Laptop l = new Laptop();
        c.computer_method();
        l.laptop_method();
    }
}
```

```
class Laptop
{
    Laptop()
    {
        System.out.println("Constructor of Laptop class.");
    }
    void laptop_method()
    {
        System.out.println("99% Battery available.");
    }
}
```

What is token in Java?

In Java, the program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation.

These statements and expressions are made up of **tokens**. In other words, we can say that the expression and statement is a set of **tokens**. The tokens are the small building blocks of a Java program that are meaningful to the Java compiler.

The Java compiler breaks the line of code into text (words) is called Java tokens. These are the smallest element of the Java program. The Java compiler identified these words as tokens. These tokens are separated by the delimiters. It is useful for compilers to detect errors.



Defining a Class

A class defines the characteristics and behavior of an object.

For example, if you create a gaming application, each game that a user can play from the application can be considered as an object of the Games class. Each game has common characteristics, such as the number of players, game category, and score. These characteristics are known as the member variables, and the behavior is specified by methods.

Any concept that you need to implement in a Java program is encapsulated within a class. A class defines the member variables and methods of objects that share common characteristics.

Further, all the games have common methods, such as calculating score, starting the game, and displaying game instructions.



The following code snippet shows how to declare a class:

```
class <ClassName>
{
//Declaration of member variables
//Declaration of methods
}
```

In the preceding code snippet, the word, **class**, is a keyword in Java that is used to declare a class and **<ClassName>** is the name given to the class.

We can use the following code snippet to create the ClassicJumble class to develop the Classic Jumble Word game:

```
class ClassicJumble {
//Declaration of member variables
//Declaration of methods
}
```




There are certain rules that should be followed to name Java classes. A program will raise error if these rules are not followed. Some of these rules are:

- ☐ The name of a class should not contain any embedded space or symbol, such as `?`, `!`, `#`, `@`, `%`, `&`, `{}`, `[]`, `:`, `"`, and `/`.
- ☐ A class name must be unique.
- ☐ A class name must begin with a letter, an underscore (`_`), or the dollar symbol (`$`). Or, it must begin with an alphabet that can be followed by a sequence of letters or digits (0 to 9), `'$'`, or `'_'`.
- ☐ A class name should not consist of a keyword.

While naming a class, you should adhere to the following class naming conventions that improve the readability of a program:

- ☐ The class name should be a noun.
- ☐ The first letter of the class name should be capitalized.
- ☐ If the class name consists of several words, the first letter of each word should be capitalized.



After defining a class, you need to save the file before it can be executed. The following naming conventions should be followed for naming a Java file:

- A file name must be unique.
- A file name cannot be a keyword.
- If a class is specified as public, the file name and class name should be the same.
- If a file contains multiple classes, only one class can be declared as public. The file name should be the same as the class name that is declared public in the file.
- If a file contains multiple classes that are not declared public, any file name can be specified for the file.

Identifying Data Types

While working with an application, you need to store and manipulate varying data.

For example, in the Classic Jumble Word game, the game score is a numeric value and the player name is a string.

To handle such varying data in an application, Java supports various data types.

There are **eight** primitive data types in Java, which are further grouped into the following categories:



❑ **Integer type:** Can store integer values. The four integer data types are:

- byte
- short
- int
- Long

The following table lists the integer primitive data type, its data types, size, range, and default values.

Group	Data Type	Size	Range	Default Value
Integer	byte	1 byte	-2^7 to 2^7-1	0
	short	2 bytes	-2^{15} to $2^{15}-1$	0
	int	4 bytes	-2^{31} to $2^{31}-1$	0
	long	8 bytes	-2^{63} to $2^{63}-1$	0

The Details of the Integer Primitive Data Type



- ❑ **Floating point type:** Can store decimal numbers. The two floating point types are:
 - float
 - Double

The following table lists the floating point primitive data type, its data types, size, range, and default values.

Group	Data Type	Size	Range	Default Value
Floating point	float	4 bytes	3.4e-038 to 3.4e+038	0.0
	double	8 bytes	1.7e-308 to 1.7e+308	0.0



Boolean type: Can store only the values, **true** and **false**.

Character type: Can store a single character, such as a symbol, letter, and number. In the character type, there is one data type, **char**. To a character data type, 2 bytes of memory is allocated.

Note :- To store string values, you need to use built-in classes, such as, **String**, **StringBuilder**, and **StringBuffer**.

Wrapper Classes

Variables that are declared by using the primitive data types are not objects. The primitive data types, such as `int` and `char`, are not a part of the object hierarchy. Therefore, in order to use the primitive data types as objects, Java provides wrapper classes. A wrapper class acts like an object wrapper and encapsulates the primitive data types within the class so it can be treated like an object.

The various wrapper classes, such as **`Boolean`**, **`Byte`**, **`Character`**, **`Integer`**, **`Short`**, **`Long`**, **`Double`**, and **`Float`**, are provided by the `java.lang` package. These classes are useful when you need to manipulate primitive data types as objects.

Autoboxing and Unboxing

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing.

Advantages of Autoboxing and Unboxing

No need of conversion between primitives and Wrappers manually so less coding is required.

```
class BoxingExample1{  
    public static void main(String args[]){  
        int a=50;  
  
        Integer a2=new Integer(a);//Boxing  
  
        Integer a3=5;//Boxing  
        //Integer a3=Integer.valueOf(5)  
        System.out.println(a2+" "+a3);  
    }  
}
```


Example of Unboxing in java

The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing. Let's see the example of unboxing:

```
class UnboxingExample1{
    public static void main(String args[])
    {
        Integer i=new Integer(50);
        int a=i;
        //int a=i.intValue();
        System.out.println(a);
    }
}
```

Identifying Class Members

In Java, a class can contain the following members:

- Variables
- Methods
- Objects
- Inner classes

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.

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.

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

Variable

A **variable** needs to be declared before it is accessed. The declaration of a variable informs the compiler about the variable name and the type of value stored in it. The following code snippet shows how to declare a variable:

```
<type> <variablename>; // Single variable of given type.
```

```
<type> <variable1name,variable2name.....variable_n_name>;  
// Multiple variables of given type.
```

```
int choice;  
int num1, num2;
```

The following code snippet shows how to assign values to a variable:

```
<type> <variablename>=<value>; // During declaration.  
<variablename>=<value> // After declaration.
```



For example, you can use the following code snippet to assign values to the variables, num1 and num2:

```
int num1, num2; // Declaration of variables.  
num1 = 5; // Assigning values to the variables.  
num2 = 10;
```

The following code snippet can be used to assign a value to a variable at the time of its declaration.

```
int num1=5;
```

You can also assign the same value to more than one variable in a single statement. The following code snippet shows the assignment of the same value to more than one variable:

```
a=b=c=3;
```

In the preceding code snippet, the integer value, 3, is first assigned to c, then to b, and finally to a.

Literal in a JAVA Program

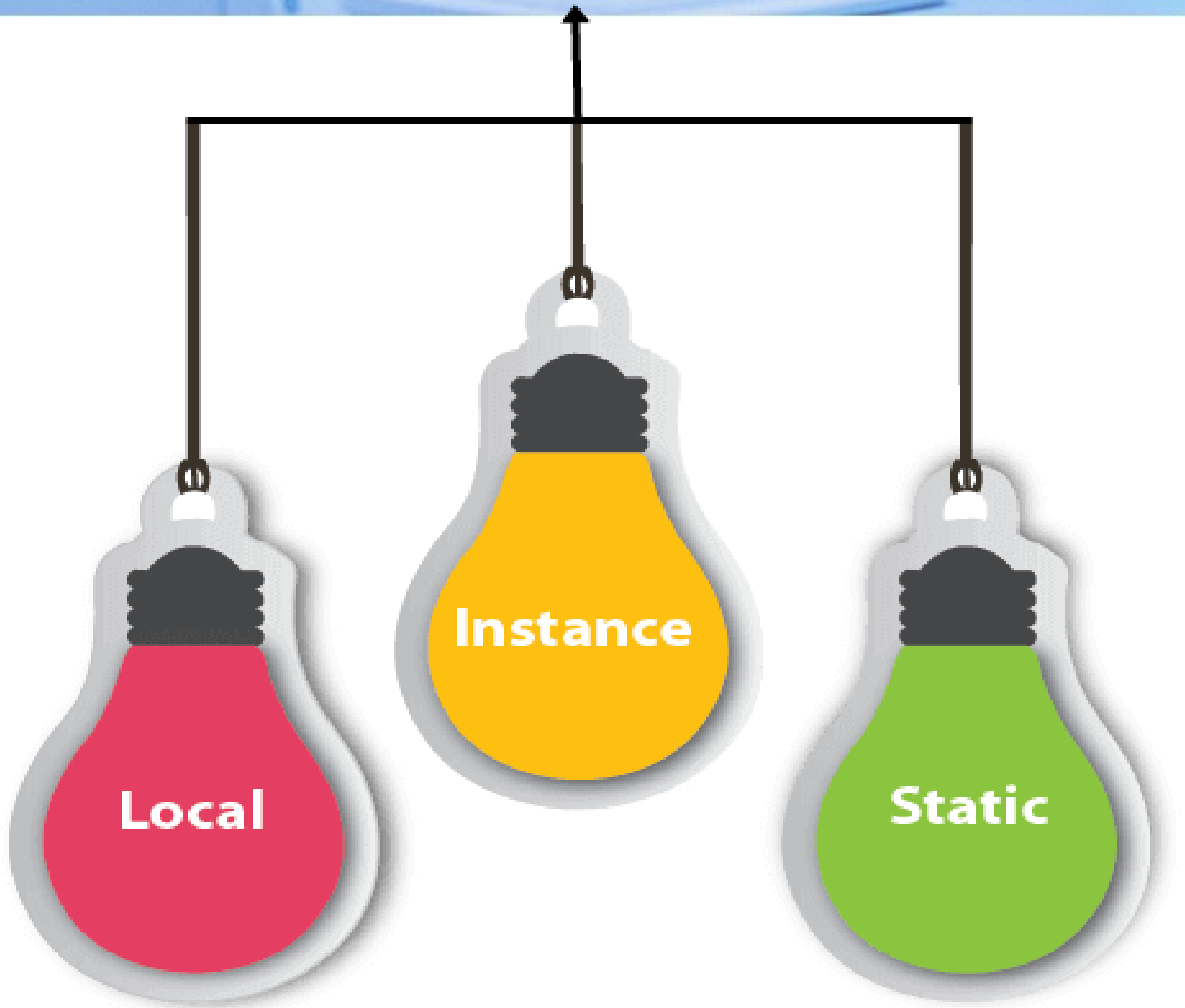
To understand the type of values that a variable can contain, you need to understand the concept of literals. **A literal is a value that is assigned to a variable or constants in a Java program.** A literal contains a sequence of characters, such as digits, alphabets, or any other symbol, which represents the value to be stored. The various types of literals in Java are:

- ❑ **Integer literals:** Are non-fractional numeric values. The numerical values can be represented in the decimal, octal, and hexadecimal notation. In order to represent an octal number, it is preceded by zero. In order to represent hexadecimal numbers, they are prefixed by 0x. For example, `n=0567` is integer literal represented in the octal notation whereas `n=0x124` represents an integer literal in hexadecimal notation.



- ❑ **Floating point literals:** Are numeric values that contain a fractional part. The floating point literals contain a decimal and an integer part. For example, `z=2.2` represents a floating point literal.
- ❑ **Character literals:** Are represented inside a pair of single quotation marks. For example, `x='k'` is a character literal.
- ❑ **String literals:** Are enclosed in a pair of double quotation marks. For example, `x="James"` is a string literal.
- ❑ **Boolean literals:** Are the literals that can contain the values, true or false. For example, `x= false` is a boolean literal.
- ❑ **Binary literals:** Are the literals that can be used to express the primitive integer data types into binary form. In order to specify these literals, you need to prefix the binary number with `0b` or `0B`. For example, `0b101`.

Types of Variables





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.

// Java Program to Illustrate Working Of Function Calling

// Class

class LocalVar {

public static int findSum(int a, int b)

{

int sum = a + b; //Local Variable

return sum;

}

public static void main(String[] args)

{

int a = 10, b = 20; //Local Variable

int c = findSum(a, b);

System.out.print(c);

}

}

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.

Features of an instance variable

The life of an instance variable depends on the life of an Object, i.e., when the object is created, an instance variable also gets created.

```
public class Student
{
    String name;
    int marks;
    Student (String stuName) {
        name = stuName;
    }
    public void setMarks(int stuMar) {
        marks = stuMar;
    }
}
```

// This method prints the student details.

```
public void printStudent() {
    System.out.println("Name: " + name );
    System.out.println("Marks:" + marks);
}
```

```
public static void main(String args[]) {
    Student StudentOne = new Student("Ankit");
    Student StudentTwo = new Student("Akshat");
    Student StudentThree = new Student("Vikram");
```

```
    StudentOne.setMarks(98);
    StudentTwo.setMarks(89);
    StudentThree.setMarks(90);
```

```
    StudentOne.printStudent();
    StudentTwo.printStudent();
    StudentThree.printStudent();
}
}
```


Java Scanner Class

The `Scanner` class of the `java.util` package is used to read input data from different sources like input streams, users, files, etc.

The `Scanner` class provides various methods that allow us to read inputs of different types.

Method

`nextInt()`

`nextFloat()`

`nextBoolean()`

`nextLine()`

`next()`

`nextByte()`

`nextDouble()`

`nextShort()`

`nextLong()`

Description

reads an int value from the user

reads a float value from the user

reads a boolean value from the user

reads a line of text from the user

reads a word from the user

reads a byte value from the user

reads a double value from the user

reads a short value from the user

reads a long value from the user

```
import java.util.Scanner;
```

```
class Scanner1 {
```

```
    String name,gender;
```

```
    Byte age;
```

```
    Integer qty;
```

```
    Double rate;
```

```
    Boolean status;
```

```
    public void input()
```

```
    {
```

```
        // creates an object of Scanner
```

```
        Scanner input = new Scanner(System.in);
```

```
        System.out.print("Enter your name: ");
```

```
        // takes input from the keyboard
```

```
        name = input.nextLine();
```

```
        System.out.print("Enter your Gender: ");
```

```
        gender=input.next();
```

```
        System.out.print("Enter your Age: ");
```

```
        age=input.nextByte();
```

```
        System.out.print("Enter Item Quantity :");
```

```
        qty=input.nextInt();
```

```
        System.out.print("Enter Rate of Item (Double) :");
```

```
        rate=input.nextDouble();
```

```
        System.out.print("Are you above 18?- :(True/False)");
```

```
        status = input.nextBoolean();
```

```
    }
```

```
public void display()
{
    // prints the name
    System.out.println("\n\n\n\n\nMy name is   :" + name);
    System.out.println("My Gender is  :" + gender);
    System.out.println("My Age is    :" + age);
    System.out.println("Rate of Item  :" + rate);
    System.out.println("Item Quantity : " + qty);
    if (status == true)
    {
        System.out.println("You are over 18");
    }

    else if (status == false)
    {
        System.out.println("You are under 18");
    }
}

public static void main(String[] args)
{
    Scanner1 obj=new Scanner1();
    obj.input();
    obj.display();
}
}
```



Static Members

Variables and methods declared using keyword **static** are called static members of a class. We know that non-static variables and methods belong to instance.

But static members (variables, methods) belong to class. Static members are not part of any instance of the class.

Static members can be accessed using class name directly, in other words, there is no need to create instance of the class specifically to use them.

Static members can be of two types:

- ❑ **Static Variables**
- ❑ **Static Methods**

Static Variable

- A static variable is associated with a class rather than an instance. A static field is declared using the static keyword.
- Static variables are shared across all instances of a class.
- There is only one copy of a static variable per class.
- Non-static variables cannot be called inside static methods.
- If any instance of a class modifies the value of a static variable, the change is reflected across all instances of the class.
- Static variables are memory-efficient as they are not duplicated for each instance.



Static Variables

Static variables are also called **class variables** because they can be accessed using class name, whereas, non static variables are called instance variables and can be accessed using instance reference only.

Static variables occupy single location in the memory. These can also be accessed using instance reference.

These are accessible in both static and non-static methods, even non-static methods can change their values.



While declaration, there is no need to initialize the static variables explicitly because they take default values like other non-static variables (instance variables).

Now, the question is, why to use static variables?

Let us look at a reason for using one. Suppose you want to keep record about the number of running instances of a class. In this situation, it would be better to declare a static variable.

Static Methods / Class Methods

Static methods are also called class methods. A static method belongs to class, it can be used with class name directly. It is also accessible using instance references.

Static methods can use static variables only, whereas non-static methods can use both instance variables and static variables.

Implementation:

The following program uses a static counter variable and static method `showCount()` method to track the number of running instances of the class `student` in the memory.

```
class Student
{
    private static int count=0; //static variable
    Student()
    {
        count++; //increment static variable
    }
    static void showCount() //static method
    {
        System.out.println("Number Of students : "+count);
    }
}

public class StaticDemo
{
    public static void main(String[] args)
    {
        Student s1=new Student();
        Student s2=new Student();
        Student.showCount(); //calling static method
        Student s3=new Student();
        Student s4=new Student();
        s4.showCount(); //calling static method
    }
}
```

```
class Country {
```

```
// Static variable
```

```
static int countryCounter;
```



```
String name;
```

```
int counter;
```

```
Country() //Constructor function
```

```
{
```

```
    countryCounter++;
```

```
}
```

```
public static void main(String[] args)
```

```
{
```

```
    Country ob1 = new Country();    // Creating first instance
```

```
    // Assigning values to object's data variables.
```

```
    ob1.name = "India";
```

```
    ob1.counter = 1;
```

```
    Country ob2 = new Country(); // Creating second instance
```

```
    ob2.name = "South Africa";
```

```
    ob2.counter = 1;
```

```
    System.out.println("ob1.countryCounter = " + ob1.countryCounter);
```

```
    System.out.println("ob2.countryCounter = " + ob2.countryCounter);
```

```
    System.out.println("Country.countryCounter = " + Country.countryCounter);
```

```
}
```

```
}
```

Java Command Line Arguments

The command-line arguments in Java allow the programmers to pass the arguments during the execution of a program. The users can pass the arguments during the execution by passing the command-line arguments inside the `main()` method.

We need to pass the arguments as space-separated values. We can pass both strings and primitive data types(`int`, `double`, `float`, `char`, etc) as command-line arguments to JVM. These arguments get stored as Strings in a String array that is passed to the `main()` function.

We can use these command-line arguments as input in our Java program.

// Java Program to Check for Command Line Arguments

```
class CommandLine {
```

```
// Main driver method
```

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

```
    // Checking if length of args array is greater than 0
    if (args.length > 0) {
```

```
        // Print statements
```

```
        System.out.println("The command line arguments are:");
```

```
        // Iterating the args array
```

```
        // using for each loop
```

```
        for (String val : args)
```

```
            // Printing command line arguments
```

```
            System.out.println(val);
```

```
        }
```

```
    else
```

```
        // Print statements
```

```
        System.out.println("No command line "+ "arguments found.");
```

```
    }
```

```
}
```



```
package mypack;  
  
public class CommandLine  
{  
  
    public static void main(String []args)  
    {  
        System.out.println("Welcome to java Class");  
        System.out.println("Command Line Arguments are:");  
  
        for (String s : args)  
            System.out.println("Command Line Arguments are :->" + s);  
        }  
    }
```



```
public class Arguments  
{  
    public static void main(String args[])  
    {        int sum=0;  
            System.out.println("Sum of all numbers are");  
            for(int i=0;i<args.length;i++)  
            {  
                sum=sum+Integer.parseInt(args[i]);  
            }  
            System.out.println("Sum of numbers is "+sum);  
    }  
}
```



```
public class Tables
{
    public static void main(String args[])
    {
        int n=0;
        System.out.println("Table of "+args[0]);
        for(int i=1;i<=10;i++)
        {
            n=Integer.parseInt(args[0]);
            System.out.println(n+" * "+i+" = "+n*i);
        }
    }
}
```

Java Constant

- **Constant** is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers **static** and **final**.
- In Java, to declare any variable as constant, we use static and final modifiers. It is also known as **non-access** modifiers. According to the Java naming convention the identifier name must be in **capital letters**.

Static and Final Modifiers

- The purpose to use the static modifier is to manage the memory.
- It also allows the variable to be available without loading any instance of the class in which it is defined.
- The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.


```
import java.util.Scanner;
public class ConstantExample1
{
//declaring constant
private static final double PRICE=234.90;
```

```
public static void main(String[] args)
{
int unit;
double total_bill;
PRICE=900;
```

```
System.out.print("Enter the number of units you have used: ");
Scanner sc=new Scanner(System.in);
unit=sc.nextInt();
```

```
total_bill=PRICE*unit;
```

```
System.out.println("The total amount you have to deposit is:
"+total_bill);
}
}
```

Method or Function

A *method* is a set of statements that is intended to perform a specific task.

For example,

a compute() method can be used for computing the score.

Moreover, methods provide encapsulation and are also essential to refer to the data members and access them. A method consists of two parts, method declaration and method body.

The syntax for defining a method is:

```
<Access specifier> <Return type> <Method name>(Parameter list)
//method declaration
{
<Method body> // body of the method
}
```



In the preceding syntax, the first line depicts the method declaration and the contents within the curly braces depict the method body. According to the preceding syntax, the various elements of a method are:

- ❑ **Access specifier:** Determines the extent to which a method can be accessed from another class.
- ❑ **Return type:** A method can also return a value. The return type specifies the data type of the value returned by a method. Some methods do not return any value. Such methods should use the void return type. The value to be returned by the method is specified with the keyword, return.
- ❑ **Method name:** Is a unique identifier and is case-sensitive.



❑ **Parameter list:** A method may or may not contain a parameter list depending upon its functionality. The parameter list consists of a list of parameters separated by commas. Parameters are used to pass data to a method. The parameter list is written between parentheses after the method name. The parentheses are included even if there are no parameters.

❑ **Method body:** This contains the set of instructions that perform a specific task. The variables declared inside the method body are known as local variables.



While adding a method to a class, you should adhere to the following method naming conventions:

- ❑ The method names should be verb-noun pairs.
- ❑ The first letter of the method should be in lowercase.
- ❑ If the method name consists of several words, the first letter of each word, except the first word, should be capitalized.

Example :

```
public int startGame()
```

Constructor in Java

Constructor is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

- ☐ numeric data types are set to 0
- ☐ char data types are set to null character('\0')
- ☐ reference variables are set to null

Rules for creating a Java Constructor



- 1) It has the same name as the class
- 2) It should not return a value not even *void*
- 3) They don't require manual calling because they are automatically called when an object is created.
- 4) Constructors are crucial for initializing objects with default or initial states.

Types of constructor in Java

There are three types of constructors in Java, which are

1. Default constructor in Java
2. No arguments constructor in Java
3. Parameterized constructor in Java

Default Constructor

If you do not create any constructor in the class, Java provides a **default constructor** that initializes the object with default values.

2. No-Args (No Argument) Constructor

As the name specifies, the No-argument constructor does not accept any argument. By using the No-Args constructor you can initialize the class data members and perform various activities that you want on object creation.

Example: No-Args Constructor



```
class Demo{
    int value1;
    int value2;
    Demo(){
        value1 = 10;
        value2 = 20;
        System.out.println("Inside Constructor");
    }

    public void display(){
        System.out.println("Value1 === "+value1);
        System.out.println("Value2 === "+value2);
    }

    public static void main(String args[]){
        Demo d1 = new Demo();
        d1.display();
    }
}
```

3. Parameterized Constructor

A constructor with one or more arguments is called a parameterized constructor.

Most often, you will need a constructor that accepts one or more parameters.

Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

```
class Demo{
    int value1;
    int value2;
    Demo(){ //Default Constructor
        value1 = 10;
        value2 = 20;
        System.out.println("Inside 1st Constructor");
    }
    Demo(int a){ //One Argument Constructor
        value1 = a;
        System.out.println("Inside 2nd Constructor");
    }
    Demo(int a,int b){ //One Argument Constructor
        value1 = a;
        value2 = b;
        System.out.println("Inside 3rd Constructor");
    }
}
```



```
public void display()
{
    System.out.println("Value1 === "+value1);
    System.out.println("Value2 === "+value2);
}

public static void main(String args[]){
    Demo d1 = new Demo();
    Demo d2 = new Demo(30);
    Demo d3 = new Demo(30,40);
    d1.display();
    d2.display();
    d3.display();
}
}
```

Cloneable interface in JAVA

Cloneable is an interface that is used to create the exact copy of an object. It exists in java.lang package.

A class must implement the Cloneable interface if we want to create the clone of the class object. The clone() method of the Object class is used to create the clone of the object.

However, if the class doesn't support the cloneable interface, then the clone() method generates the CloneNotSupportedException.

The syntax of the clone() method is given below.

protected Object clone() throws CloneNotSupportedException

class Student implements Cloneable

{

int rollno;

String name;

Student(int rollno,String name)

{

this.rollno=rollno;

this.name=name;

}

public Object clone() throws CloneNotSupportedException

{

return super.clone();

}

public static void main(String str[])

{

try

{

Student adarsh=new Student(125,"Adarsh");

Student raja=(Student)adarsh.clone();

raja.rollno=99;

raja.name="Raja Kumar";

System.out.println(adarsh.rollno+" "+adarsh.name);

System.out.println(raja.rollno+" "+raja.name);

}

catch(CloneNotSupportedException e)

{

}

}

}

Constructor Overloading in Java

Examples of valid overloaded constructors for class `Account` are

Constructor overloading means multiple constructors in a class. When you have multiple constructors with different parameters listed, then it will be known as constructor overloading.

```
Account(int a);
```

```
Account (int a,int b);
```

```
Account (String a,int b);
```

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- Widening Casting (automatically) - converting a smaller type to a larger type size

byte -> short -> char -> int -> long -> float -> double

- Narrowing Casting (manually) - converting a larger type to a smaller size type

double -> float -> long -> int -> char -> short -> byte

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

```
public class Main
{
    public static void main(String[] args)
    {
        int myInt = 9;
        double myDouble = myInt;
        // Automatic casting: int to double
        System.out.println(myInt); // Outputs 9
        System.out.println(myDouble); // Outputs 9.0
    }
}
```



Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

```
public class Main {  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble;  
        // Manual casting: double to int  
  
        System.out.println(myDouble); // Outputs 9.78  
        System.out.println(myInt);    // Outputs 9  
    }  
}
```

Exercise 1

Predict the output of the following code:



```
class JumbledGame {  
    final int choice;  
    public static void main(String[] args) {  
        JumbledGame obj1 = new JumbledGame ();  
        System.out.println(obj1.choice);  
    }  
}
```

Exercise 2

Write a program to create a class named `SampleClass`. The `SampleClass` class should contain a variable named `counter` and assign it the value, 1. Declare the `counter` variable as `public`. Create a method named `Display()` in the class. The `Display()` method should print the value of the `counter` variable. Create an object and access the `Display()` method using the object.

Exercise 3

Create a `Reservation` class with a member variable, `TicketID`, and a method, `ShowTicket()`. A constructor should initialize the `TicketID` variable, and the `ShowTicket()` method should display its value. In addition, create an object of the `Reservation` class to call these methods.

Exercise 4

ServeYourMoney bank offers various services to its customers. The bank has branches all over the country, and therefore, each branch has a unique id. The bank stores the details of its customers, such as customer name, account number, address, phone number, and email address. In addition, the bank stores the various details of its employees, such as employee name, employee id, address, phone number, and email id. The bank offers its customers a choice of accounts, such as savings account and current account. Ask the students to identify the various classes, their member variables, and data types.

Exercise 5

Create a Grocery class with a member variable, weight, and two methods, weightNow() and checkWeight(). The weightNow() method should initialize the weight variable and the checkWeight() method should display its value. In addition, create an object of the Grocery class to call the preceding methods.





