Module 2 - What is Java?

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

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

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

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

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 best suited for internet programming. Later, Java technology was incorporated by Netscape.

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

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.

Features of Java

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

Java buzzwords.

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

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

1. 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 it is 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.

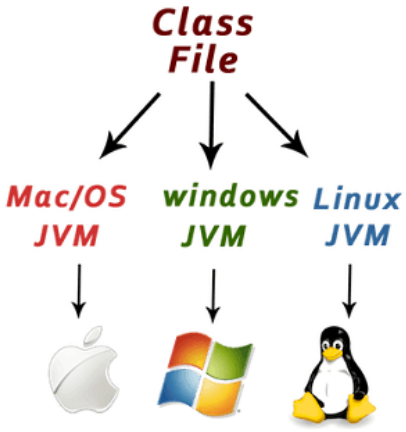
2. Object-oriented
Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means the software or application developed is organized as a combination of different types of objects that incorporate both data and behavior.

3. 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).

4. Secured

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

1. No explicit pointer
2. 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 provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

5. 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.

6. 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.

7. Portable

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

8. 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.

9. 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.

10. 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 multimedia, Web applications, etc.

11. 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).

First Java Program | Hello World Example

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

            System.out.println("Hello                          World!");
      }
}
```

Save the above file as **`Hello.java.`**

To compile the program:

```
javac Hello.java
```

To execute the program:

```
java Hello
```

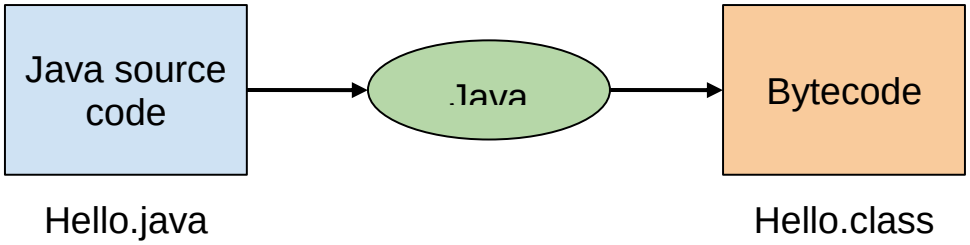
Components of the First Java Program (Hello World Program)

This section discusses the meaning of class, public, static, void, main, String[], System.out.println().

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

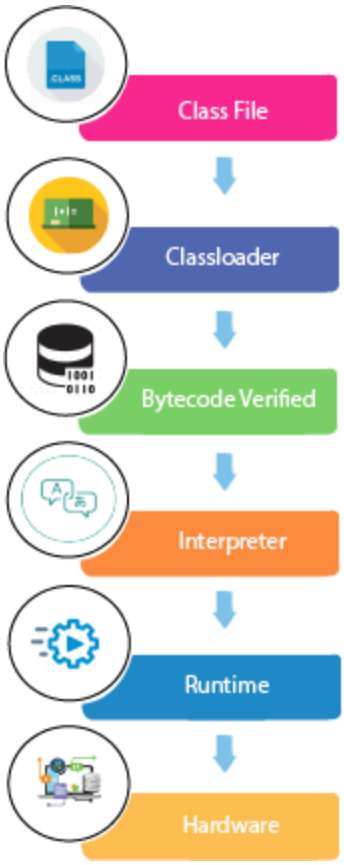
What happens at compile time?

At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into **bytecode**.



What happens at runtime?

At runtime, the following steps are performed:



- **Classloader:** It is the subsystem of JVM that is used to load class files.
- **Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.
- **Interpreter:** Read bytecode stream then execute the instructions.

Java Tokens

The Java compiler breaks the line of code into text (words) called Java tokens. These are the smallest elements of the Java program.

Types of Tokens

Java token includes the following:

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Strings
6. Separators

1. Keywords – Java keywords are also known as reserved words. Keywords are particular words

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

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

2. Identifier – Identifiers are used to name a variable, constant, function, class, and array. It is usually defined by the user. It uses letters, underscores, or a dollar sign as the first character. The label is also known as a special kind of identifier that is used in the goto statement. Remember that the identifier name must be different from the reserved keywords. There are some rules to declare identifiers are:

- The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
- The whitespace cannot be included in the identifier.
- Keywords cannot be used as identifiers
- Identifiers are case sensitive.

3. Literals – In programming literal is a notation that represents a fixed value (constant) in the source code. It can be categorized as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Once it has been defined cannot be changed. Java provides five types of literals are as follows:

- Integer
- Floating Point
- Character
- String
- Boolean

4. Operators – In programming, operators are the special symbol that tells the compiler to perform a special operation. Java provides different types of operators that can be classified according to the functionality they provide. There are eight types of operators in Java, are as follows:

- Arithmetic Operators
- Assignment Operators
- Relational Operators

- Unary Operators
- Logical Operators
- Ternary Operators
- Bitwise Operators
- Shift Operators

Operator	Symbols
Arithmetic	+ , - , / , * , %
Unary	++ , - - , !
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=
Logical	&& , ||
Ternary	(Condition) ? (Statement1) : (Statement2);
Bitwise	& , | , ^ , ~
Shift	<<, >>, >>>

5. Strings – Strings will be discussed later in detail.

6. Separators – The separators in Java are also known as punctuators. There are nine separators in Java, are as follows:

- **Square Brackets []:** It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.
- **Parentheses ():** It is used to call the functions and parsing the parameters.
- **Curly Braces {}:** The curly braces denote the starting and ending of a code block.
- **Comma (,):** It is used to separate two values, statements, and parameters.
- **Assignment Operator (=):** It is used to assign a variable and constant.
- **Semicolon (;):** It is the symbol that can be found at end of the statements. It separates the two statements.
- **Period (.):** It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

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 the name of the memory location. The term “variable” itself indicates its value can be changed in the course of program execution.

Types of Variables

There are three types of variables in Java:

1. local variable
2. instance variable
3. static variable

1. Local Variable

A variable declared inside the body of the method is called a 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 the "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.

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

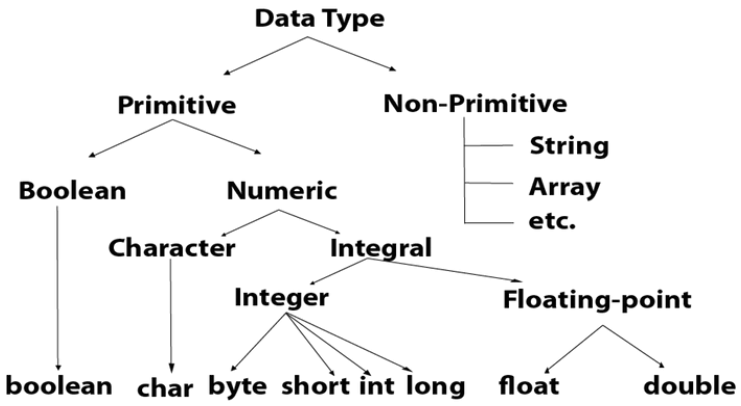
In Java language, primitive data types are the building blocks of data manipulation. These are the

most basic data types available in Java language.

There are 8 types of primitive data types:

1. boolean
2. byte
3. char
4. short
5. int
6. long
7. float
8. double

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



Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java

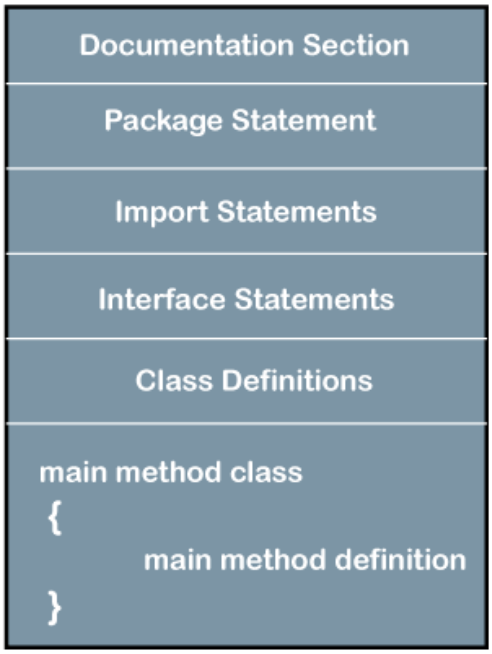
provides statements that can be used to control the flow of Java code. Such statements are called

control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - **for-each loop (Enhanced for loop)**
3. Jump statements
    - break statement
    - continue statement

Structure of Java Program



Structure of Java Program

A typical structure of a Java program contains the following elements:

1. Documentation Section
2. Package Declaration
3. Import Statements
4. Interface Statements
5. Class Definition
6. Class Variables and Variables
7. Main Method Class

1. Documentation Section

The documentation section is an important section but optional for a Java program. It includes basic information about a Java program. The information includes the **author's name, date of creation, version, program name, company name, and description of the program**. It improves the readability of the program and it is suggested to write documentation. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, The **comments** are used. The comments may be **single-line, multi-line, and documentation comments**.

- **Single-line Comment:** It starts with a pair of **forwarding slash (*//*)**. For example:

```
//First Java Program
```

- **Multi-line Comment:** It starts with a **/* and ends with */.** We write between these two symbols. For example:

```
/*It is an example of
```

```
multiline comment*/
```

- **Documentation Comment:** It starts with the delimiter **(/**) and ends with */.** For example:

```
/**It is an example of documentation comment*/
```

2. Package Declaration

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the package name in which the class is placed. Note that there can be only one package statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes the package belongs to a single parent directory. The keyword **package** is used to declare the package name.

```
package sucet; //where sucet is the package name

package com.sucet; //where com is the root directory and sucet is the
subdirectory
```

3. Import Statements

The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. The **import** keyword is used to import the class. It is written before the class declaration and after the package statement. The import statement can be written in two ways, either to import a specific class (or an interface) or to import all classes & interfaces of a particular package. In a Java program, we can use multiple import statements. For example:

```
import java.util.Scanner; //imports the Scanner class from the
//java.util package, this statement imports only the Scanner class

import java.util.*; //it imports all the classes and interfaces of the
//java.util package, however it will not import sub-packages that exist
in a package.
```

4. Interface Statements

It is an optional section. We can create an interface in this section if required. The **interface** keyword is used to create or define an interface. An interface is slightly different from the class. It contains only constants and method declarations. Another difference is that it cannot be instantiated. We can use interfaces in classes by using the implements keyword. An interface can also be used with other interfaces by using the extends keyword. For example:

```
interface                                                          car
{
      void                                                      start();

      void                                                       stop();
}
```

5. Class Definition

In this section, we define the class. It is a vital part of a Java program. Without the class, we

cannot create any Java program. A Java program may contain more than one class definition. The **class** keyword is used to define the class. In OOP, the class is a blueprint or a template of an object. It contains information about user-defined methods, variables, and constants. **Every Java program has at least one class that contains the main() method**. For example:

```
class               Student               //class               definition
{

}
```

6. Class Variables and Constants

In this section, we define variables and constants that are to be used later in the program. In a Java program, the variables and constants are defined just after the class definition. The variables and constants store values of the parameters. It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

```
class               Student               //class               definition
{
      String                sname;                                  //variable
      int                                                                id;
      double                                                       percentage;
}
```

7. Main Method Class

In this section, we define the main() method. It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

```
public       class       Student       {       //class       definition
  public          static          void          main(String          args[])
      {  //statements       }
}
```