Java

Java is a compiler and jvm based pure object Oriented programming language.

It is a part of the Java programming language that one can use for developing or creating a general-purpose app. Its main focus is to build such general applications. The J2SE (Java Standard Edition) is known as Core Java. It mainly covers concepts of object-oriented programming (OOP).

History

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.

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.

Features of Java/advantage of Java/characteristics of Java

Following are the notable features of Java:

Object Oriented

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent

Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable

Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust

Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Interpreted

Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

High Performance

With the use of Just-In-Time compilers, Java enables high performance.

Distributed

Java is designed for the distributed environment of the internet using several jvm at the time by load balancing concept and remote method invocation (RMI), CORBA(Common Object Request Broker Architecture), EJB(***enterprise java bean)***.

Dynamic

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

Structure of java Program

Let's see which elements are included in the structure of a Java program. A typical structure of a Java program contains the following elements:

Documentation Section

Package Declaration

Import Statements

Interface Section

Class Definition

Class Variables and Variables

Main Method Class

Methods and Behaviors

Syntax of Java Program.

```
Class Class_name
{
Variables ;
Public static void main(String args[])
{
Statements ;
}
}
```

Compile and Run :

For compilation of java code: javac file name.java
For execute/Run the java code: Java class_name

Token

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. Remember that the delimiters are not part of the Java tokens.

Types of Tokens

Java token includes the following:

- Keywords
- Identifiers
- Operators
- Separators
- Comments

Java Statements

Statements are roughly equivalent to sentences in natural languages. In general, statements are just like English sentences that make valid sense.

In Java, a **statement** is an executable instruction that tells the compiler what to perform. It forms a complete command to be executed and can include one or more expressions.

Types of Statements

Java statements can be broadly classified into the following categories:

Expression Statements

Declaration Statements

Control Statements

Constant and Variables

A **constant** is a value that cannot be altered by the program during normal execution, i.e., the value is constant. When associated with an identifier, a constant is said to be “named,” although the terms “constant” and “named constant” are often used interchangeably. This is contrasted with a **variable**, which is an identifier with a value that can be changed during normal execution, i.e., the value is variable.

A constant is a data item whose value cannot change during the program’s execution. Thus, as its name implies – the value is constant.

A variable is a data item whose value can change during the program’s execution. Thus, as its name implies – the value can vary.

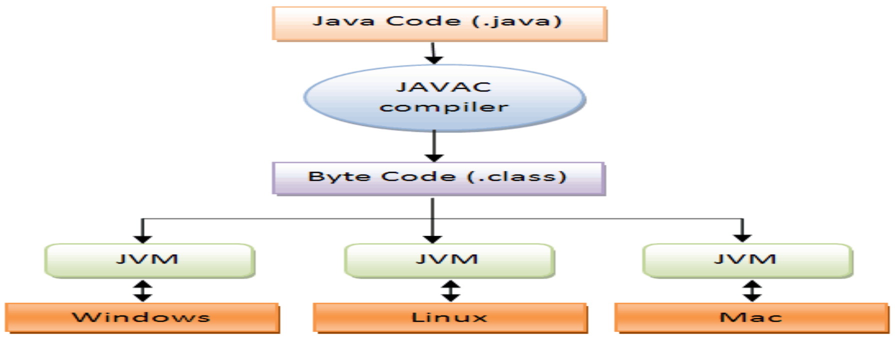
```
class test{
  public static void main(String args[]){
    final int a=10;
     a=20; //Error final variable cannot allow to change value
     System.out.println(a);
  }
```

}

Java Virtual Machine

Software from Oracle that converts a program in Java bytecode (intermediate language) into machine language and executes it. The Java Virtual Machine (JVM) is the runtime engine of the Java Platform, which allows any program written in Java or other language compiled into Java bytecode to run on any computer that has a native JVM. JVMs run in both clients and servers, and the Web browser can activate the JVM when it encounters a Java applet.

The JVM includes a just-in-time (JIT) compiler that converts the bytecode into machine language so that it runs as fast as a native executable. The compiled program can be cached in the computer for reuse.



Scope of a Variable

In programming, a variable can be declared and defined inside a class, method, or block. It defines the scope of the variable i.e. the visibility or accessibility of a variable. Variable declared inside a block or method are not visible to outside. If we try to do so, we will get a compilation error. Note that the scope of a variable can be nested.

- We can declare variables anywhere in the program but it has limited scope.
- A variable can be a parameter of a method or constructor.
- A variable can be defined and declared inside the body of a method and constructor.
- It can also be defined inside blocks and loops.
- Variable declared inside main() function cannot be accessed outside the main() function

Data Types

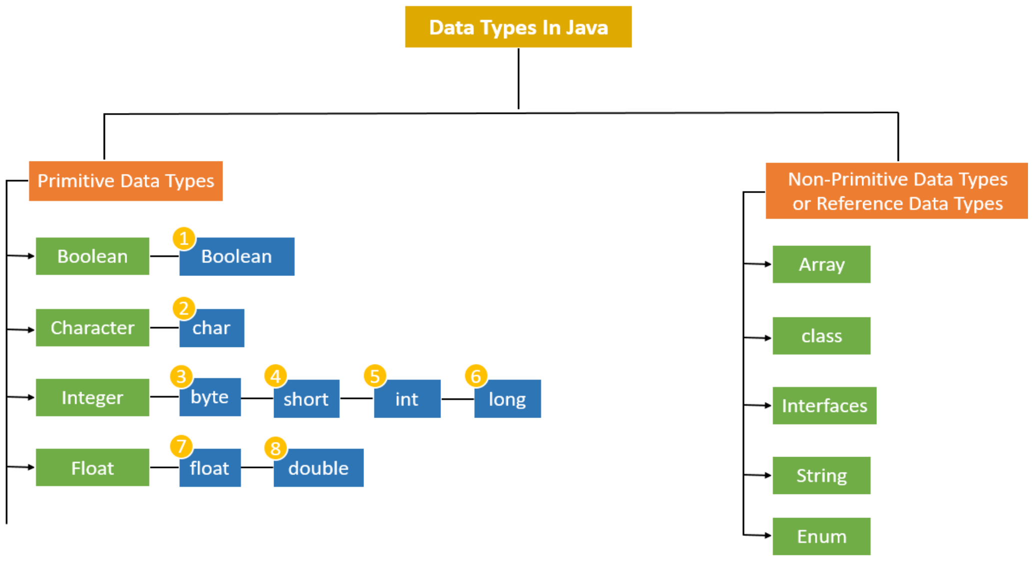
Data types are different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. Also, let us cover up other important ailments that there are majorly two types of languages that are as follows:

1. First, one is a Statically typed language where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types. For example C, C++, Java.

2. The other is Dynamically typed languages. These languages can receive different data types over time. For example Ruby, Python

Java is statically typed and also a strongly typed language because, in Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

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



Symbolic Constant.

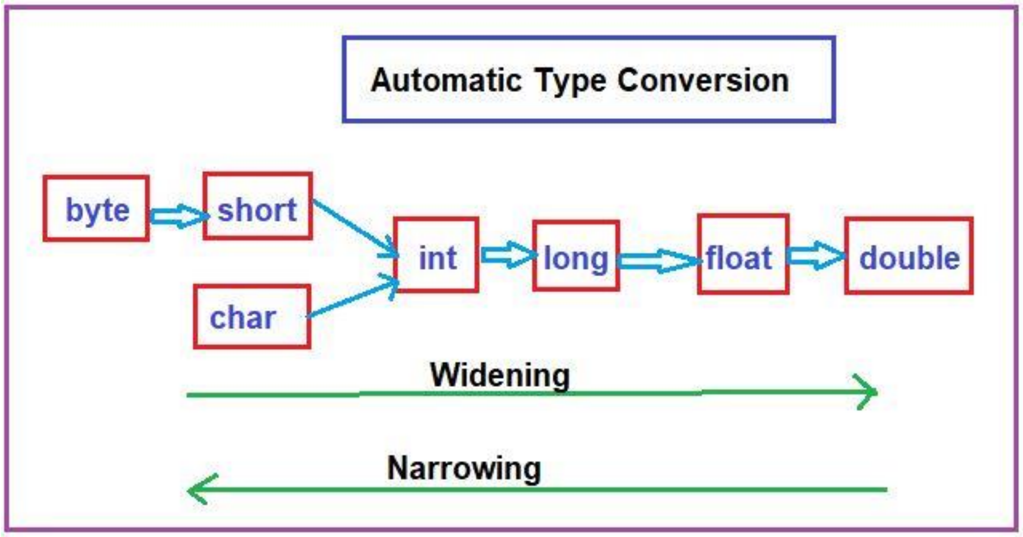
Symbolic constants are what most people think of when someone refers to "constants" in any programming language. Symbolic constants are nothing more than a label or name that is used to represent a fixed value that never changes throughout a program. For example, one might define **PI** as a constant to represent the value 3.14159.

Example

```
final  double  PI = 3.14159;
```

Type casting in java

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



Types of Type Casting

There are two types of type casting:

Widening Type Casting

Narrowing Type Casting

Widening Type Casting

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

Exam:

```
class test{
   public static void main(String args[]){
      int a=100;
      double b=a;
      System.out.println("type casting from int to double= "+b);
   }
}
```

Narrowing Type Casting

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

Exam:

```
class test{
  public static void main(String args[]){
    double a=100.234;
    int b=a;  //cast.java:4: error: incompatible types: possible lossy conversion from double to int
   // int b=a;
    System.out.println("type casting from double to int = "+b);
  }
}
```

Forcefully Converting

```
class test{
  public static void main(String args[]){
    double a=10.98;
    int b=(int)a;
    System.out.println("type casting from int to double = "+b);
  }
}
```

Command Line Arguments

Java command-line argument is an argument i.e. passed at the time of running the Java program. In the command line, the arguments passed from the console can be received in the java program and they can be used as input. The users can pass the arguments during the execution bypassing 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. These arguments convert into a string array and are provided to the main() function as a string array argument.

When command-line arguments are supplied to JVM, JVM wraps these and supplies them to args[]. It can be confirmed that they are wrapped up in an args array by checking the length of args using args.length.

Internally, JVM wraps up these command-line arguments into the args[ ] array that we pass into the main() function. We can check these arguments using args.length method. JVM stores the first command-line argument at args[0], the second at args[1], the third at args[2], and so on.

The main purpose of command line arguments is to customized the main method.

Example:

```
class test{
    public static void main(String args[]){
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
         System.out.println(args.length);
    }
 }
```

Example:

```
class test{
    public static void main(String args[]){
        int a = Integer.parseInt(args[0]);
        System.out.println("the square of number is  "+a*a);
    }
 }
```