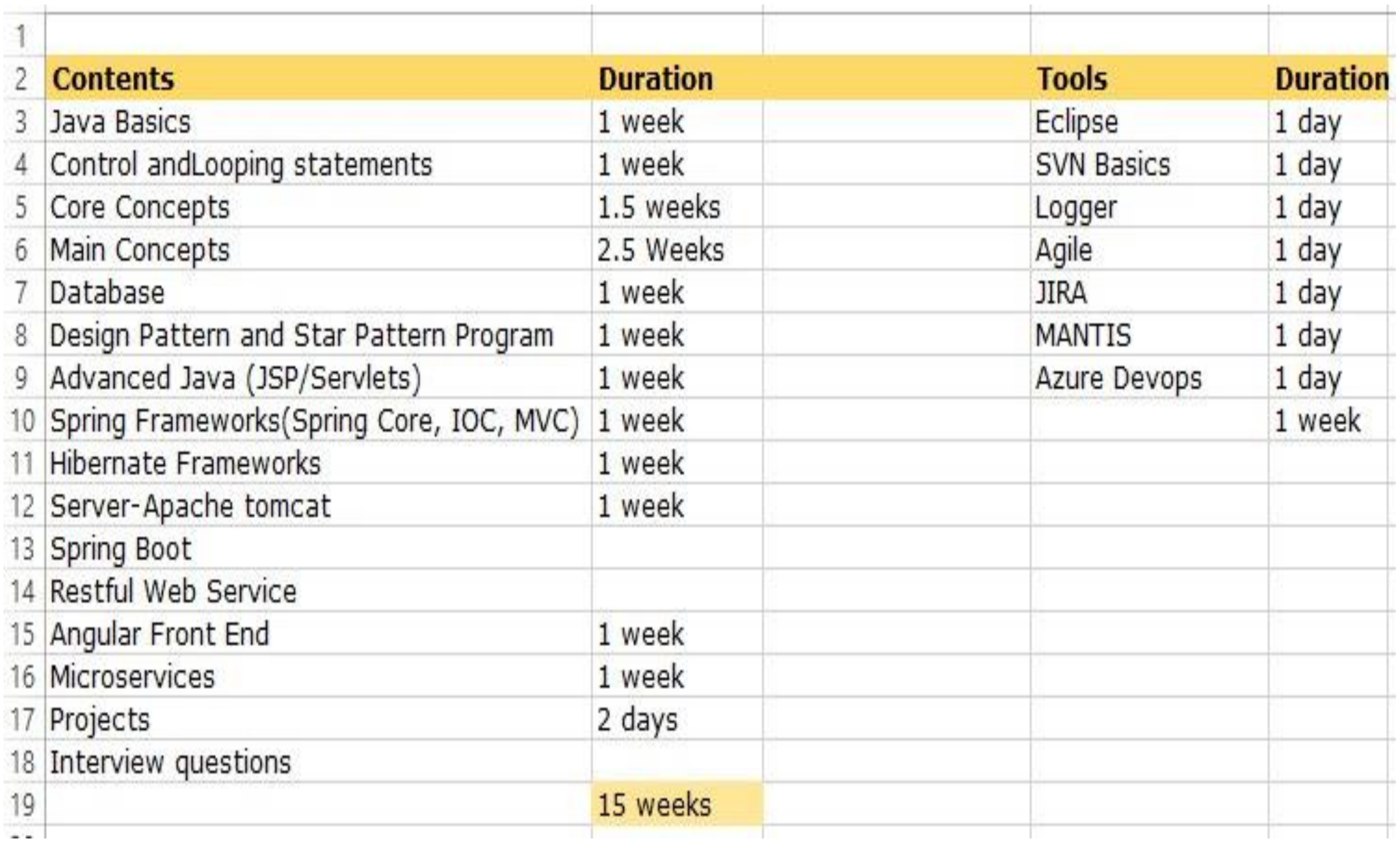
# SYLLABUS OF JAVA

- Core java
- Advanced java
  (JDBC, JSP, Servlets )
- Design Pattern
- Spring IOC
- Spring MVC
- Spring Boot
- Hibernate
- Restful web services
- Micro services
- Angular
- Database

## ----Real time tools-------

- Agile Methodology
- Apache Tomcat Server
- Eclipse
- Git Version Control
- JIRA
- Junit
- Logger
- Maven
- Node JS
- Visual Studio code
- STS
- Postman
- Industrial projects-2
- ***Banking Domain***- Capgemini, Cognizant, HSBC, Barclays.....
- ***Healthcare Domain***- AMDOCS, Accenture, MediAssist, L& T



| Contents | Duration | Tools | Duration |
|---|---|---|---|
| Java Basics | 1 week | Eclipse | 1 day |
| Control andLooping statements | 1 week | SVN Basics | 1 day |
| Core Concepts | 1.5 weeks | Logger | 1 day |
| Main Concepts | 2.5 Weeks | Agile | 1 day |
| Database | 1 week | JIRA | 1 day |
| Design Pattern and Star Pattern Program | 1 week | MANTIS | 1 day |
| Advanced Java (JSP/Servlets) | 1 week | Azure Devops | 1 day |
| Spring Frameworks(Spring Core, IOC, MVC) | 1 week | | 1 week |
| Hibernate Frameworks | 1 week | | |
| Server-Apache tomcat | 1 week | | |
| Spring Boot | | | |
| Restful Web Service | | | |
| Angular Front End | 1 week | | |
| Microservices | 1 week | | |
| Projects | 2 days | | |
| Interview questions | | | |
| | 15 weeks | | |

**-----------------------Java notes by Prashant sir --------------------------------------**

## What is Java :

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

### History of Java :

| | |
|---|---|
| Founded by | : SUN Mc Systems (Oracle Corporation) |
| Author | : James Gosling |
| Objective | : To prepare simple electronic consumer goods. |
| Project | : Green |
| First Version | : JDK 1.0 (1996, Jan-23rd) |
| Strong Features | : Object-oriented, Platform Independent, Robust, Portable, Dynamic, Secure....... |

## Features of Java :

Java language has the following features :

- *Simple*
- *Object oriented*
- *Platform independent*
- *Architecture neutral*
- *Portable*
- *Robust*
- *Secure*
- *Dynamic*
- *Distributed*
- *Multi-threaded*
- *High performance*

**1) Simple:**

Java is simple programming language, because,

a) Java applications will take less memory and less execution time.

b) Java has removed all most all the confusion oriented features like pointers, multiple inheritance,.....

c) Java is using all the simplified syntaxes from C and C++.

**2) Object Oriented:**

Java is an object oriented programming language, because, JAVA is able to store data in the form of Objects only.

**3) Platform Independent:**

Java is platform independent programming Language, because, Java allows its applications to compile on one operating system and to execute on another operating system.

Java is platform independent because of its byte code (.class file)

**4) Architecture Neutral:**

It means that even if you write and compile a program on one hardware configuration and try to run on some other hardware configuration still it will execute.

**5) Portable:**

Java is a portable programming language, because, JAVA is able to run its applications under all the operating systems and under all the hardware Systems.

**6) Robust:**

Java is Robust programming language, because,

1) Java is having very good memory management system in the form of heap memory Management System, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.

2) JAVA is having very good Exception Handling mechanisms, because, Java has provided very good predefined library to represent and handle almost all the frequently generated exceptions in java applications.

**7) Secure:**

Java is very good Secure programming language, because,

1) JAVA has provided an implicit component inside JVM in the form of "Security Manager" to provide implicit security.

2) This makes Java popular for banking applications.

**8) Dynamic:**

If any programming language allows memory allocation for primitive data types at RUNTIME then that programming language is called as Dynamic Programming Language.

JAVA is a dynamic programming language, because, JAVA allows memory allocation for primitive data types at RUNTIME.

**9) Distributed:**

By using JAVA we are able to prepare two types of applications

1) Standalone Applications
2) Distributed Applications

## 1) Standalone Applications:

If we design any java application without using client-Server arch then that java application is called as Standalone application.

## 2) Distributed Applications:

If we design any java application on the basis of client-server arch then that java application is called as Distributed application.

To prepare Distributed applications, JAVA has provided a seperate module that is "J2EE/JAVA EE".

**10) Multi Threaded**:

Thread is a flow of execution to perform a particular task.

Multithreaded means ability to allow more than one thread to execute application, It follows parallel execution, it will reduce execution time, it will improve application performance.

JAVA is following Multi Thread Model, JAVA is able to provide very good environment to create and execute more than one thread at a time, due to this reason, JAVA is Multi threaded Programming Language.

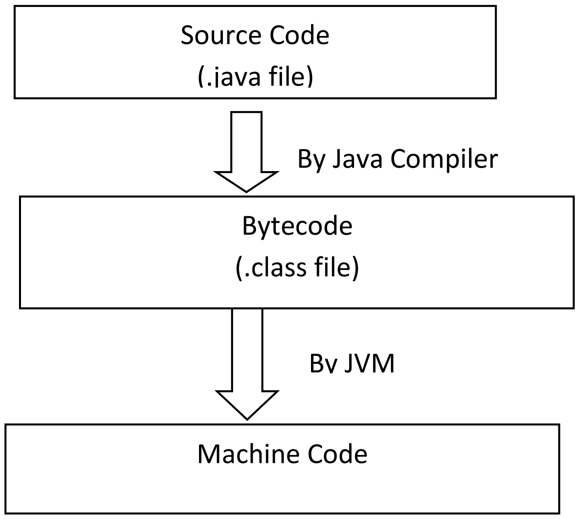
## 11) High Performance:

JAVA is high performance programming language due to its rich set of features like Platform independent, Arch Neutral, Portable, Robust, Dynamic.

## ***Java Program Execution:***

Any java program execution has the following set of steps:

1) Whatever code we write using IDE, Notepad, etc, it is called as the source code(.java file).
2) This source code is converted to bytecode(.class file) using compiler. This bytecode is machine-independent and hence makes java language as platform independent.
3) Further this bytecode is converted to machine code by the JVM which is custom-built for every operating system. . This machine code is machine/OS specific.



**4)** Due to the two-step execution process described above, a java program is independent of the target operating system. However, because of the same, the execution time is way more than a similar program written in a compiled platform-dependent program.

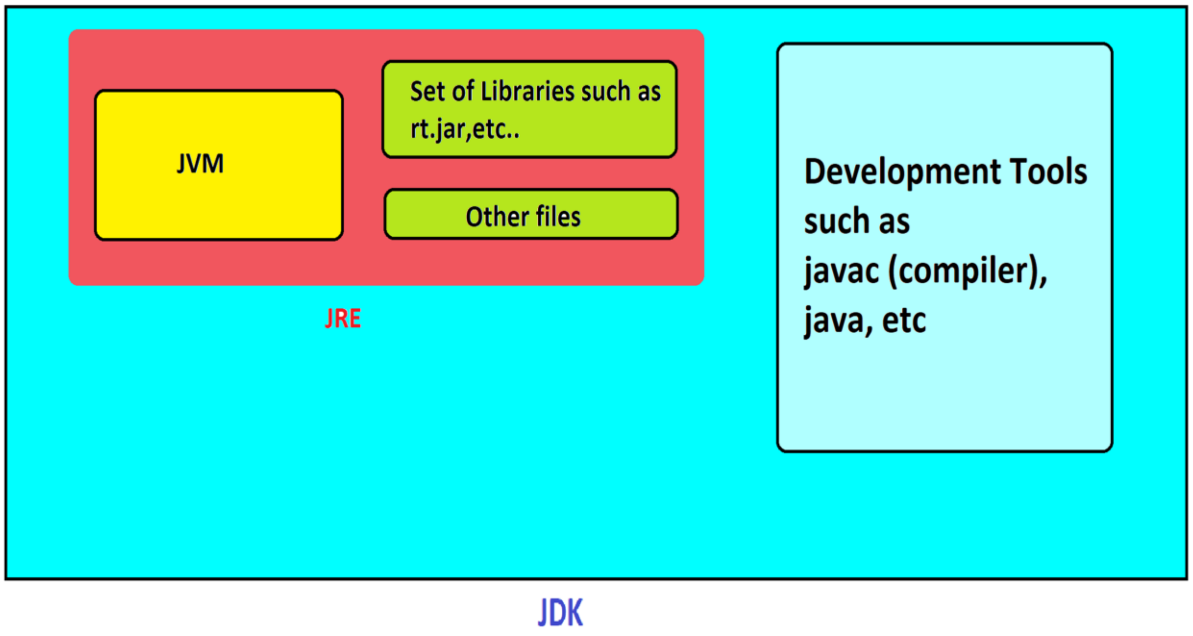
## ***JVM JRE and JDK :***

**JDK:**

**Java Development Kit (JDK**) is a software development environment used for developing Java applications. It includes the 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:**

JRE stands for **Java Runtime Environment**. The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files like rt.jar files.


JVM
Set of Libraries such as rt.jar,etc..
Other files
JRE
Development Tools such as javac (compiler), java, etc
JDK

**JVM:**JVM stands for **Java Virtual Machine**. It is a part of JRE. JVM is responsible to load and run the java program.

JVM Runs Java Byte Code by creating 5 Identical Runtime Areas to execute Class

• Class Loader Sub System
• Memory Management System
• PC-Registers
• Execution Engine
• Native Methods Stack

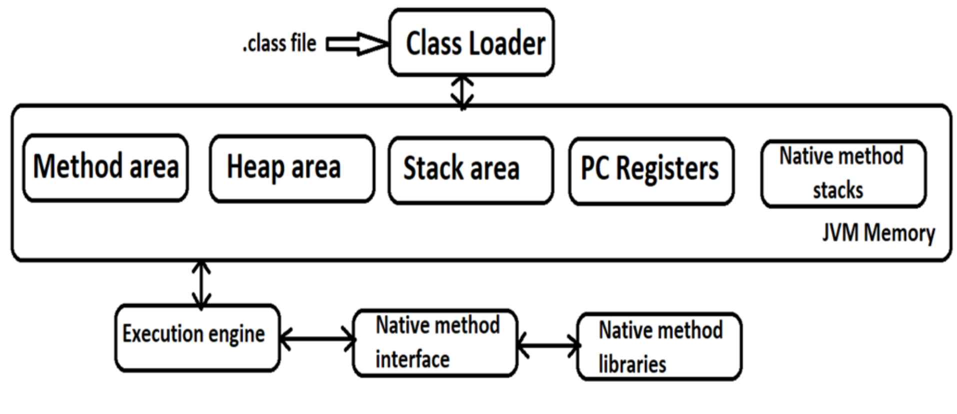


Fig. JVM Arch

## Class Loader:

Class Loader is used to load the .class file in the memory.

### a) Memory Management System:

While Loading and Running a Java Program JVM required Memory to Store Several Things Like Byte Code, Objects, Variables, Etc.
Total JVM Memory organized in the following 5 Categories:
• Method Area
• Heap Area OR Heap Memory
• Java Stack Area
• PC Registers Area
• Native Method Stacks Area

### Method Area:

• Total Class Level Binary Information including Static Variables Stored in Method Area.

**Heap Area:**

• All Objects and corresponding Instance Variables will be stored in the Heap Area.

**Java Stack Area:**

• All Method Calls and corresponding Local Variables, Intermediate Results will be stored in the Stack.

**b) PC Registers Area**

• For Every Thread a Separate PC Register will be created at the Time of Thread Creation.

• PC Registers contains Address of Current executing Instruction.

• Once Instruction Execution Completes Automatically PC Register will be incremented to Hold Address of Next Instruction.

**c) Execution Engine**

This is the Central Component of JVM.

• Execution Engine is Responsible to Execute Java Class Files.

• It contain information about JIT (Just in time) compiler and interpreter

**d) Native Methods Stack**

• For Every Thread JVM will Create a Separate Native Method Stack.

• All Native Method Calls invoked by the Thread will be stored in the corresponding Native Method Stack

**e) Java Native Interface (JNI):**

JNI Acts as Bridge (Mediator) between Java Method Calls and corresponding Native Libraries.

**f) Java Native Library:**

Java Native Library is the collection of Native methods which are required in java.

Native method is a method declared in java, but, implemented in non java programming languages like C , C++,...

## Identifiers:

A name in java program is called identifier. It may be class name, method name,

Variable name and label name.

*Rules to define java identifiers:*

*Rule 1:* The only allowed characters in java identifiers are:

1) a to z
2) A to Z
3) 0 to 9
4) _ (underscore)
5) $

*Rule 2:* If we are using any other character we will get compile time error.

Example: Total#------------------invalid

*Rule 3:* identifiers are not allowed to starts with digit.

Example: 123ABC---------invalid

*Rule 4:* java identifiers are case sensitive of course java language itself treated as case sensitive language.

*Rule 5:* There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.

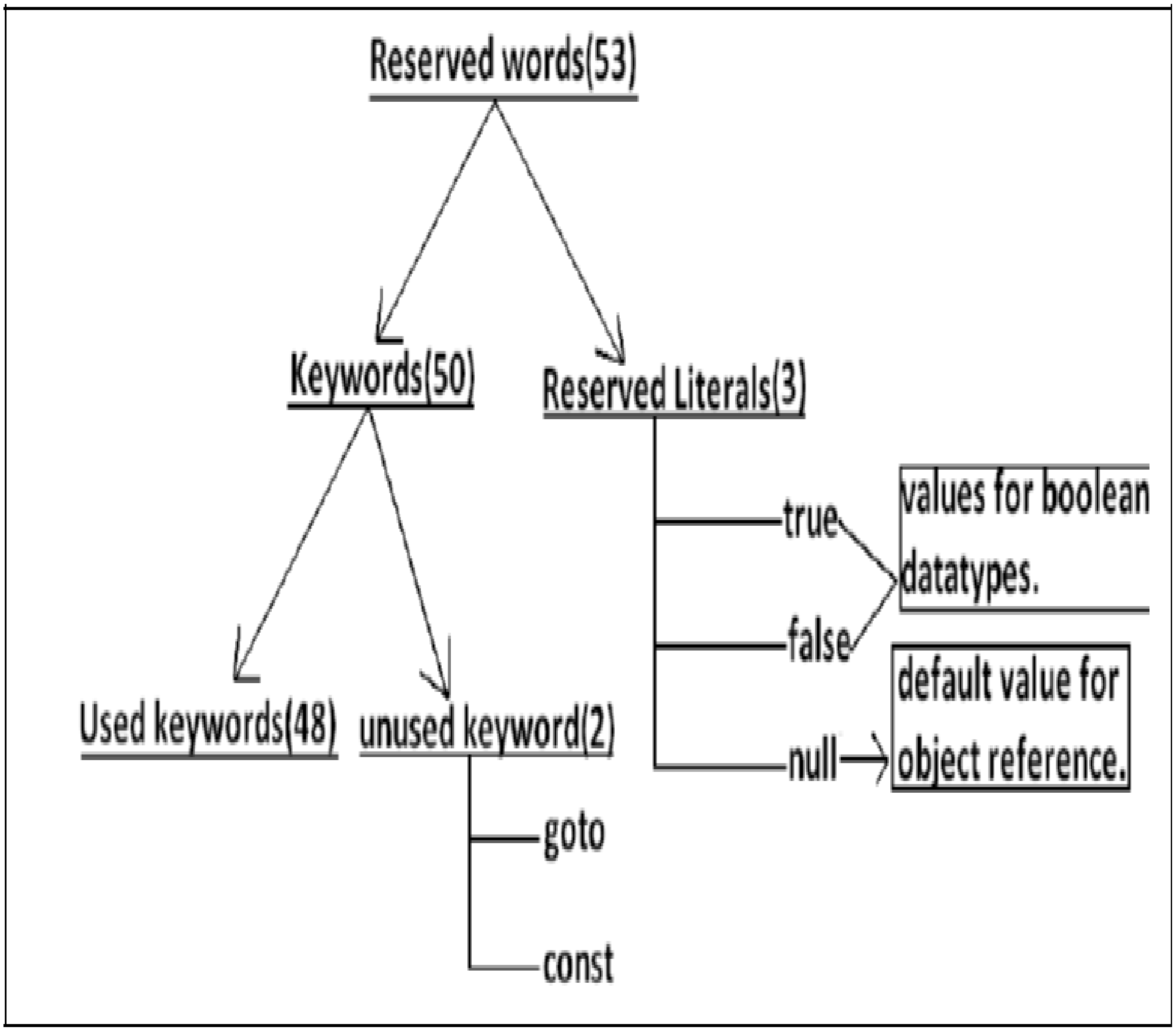
*Rule 6:* We can't use reserved words as identifiers.

Example: int if=10; --------------invalid

*Rule 7:* All predefined java class names and interface names we use as identifiers. Even though it is legal to use class names and interface names as identifiers but it is not a good programming practice.

## ***Reserved word and keywords:***

In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.



**Reserved words for data types: (8)**

| | |
|---|---|
| 1) byte | 5) float |
| 2) short | 6) double |
| 3) int | 7) char |
| 4) long | 8) Boolean |

**Reserved words for flow control:(11)**

| | | |
|---|---|---|
| 1) if | 6) for | 11) return |
| 2) else | 7) do | |
| 3) switch | 8) while | |
| 4) case | 9) break | |
| 5) default | 10) continue | |

**Keywords for modifiers:(11)**

| | | |
|---|---|---|
| 1) public | 6) abstract | 11) volatile |
| 2) private | 7) synchronized | |
| 3) protected | 8) native | |
| 4) static | 9) strictfp(1.2 version) | |
| 5) final | 10) transient | |

**Keywords for exception handling:(6)**

| | |
|---|---|
| 1) try | 4) throw |
| 2) catch | 5) throws |
| 3) finally | 6) assert(1.4 version) |

**Class related keywords:(6)**

1) class
2) package
3) import
4) extends
5) implements
6) interface

**Object related keywords:(4)**

1) new
2) instanceof
3) super
4) this

**Void return type keyword:**

1) void

## *Data types:*

Java every variable has a type, every expression has a type and all types are strictly define more over every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed programming language.

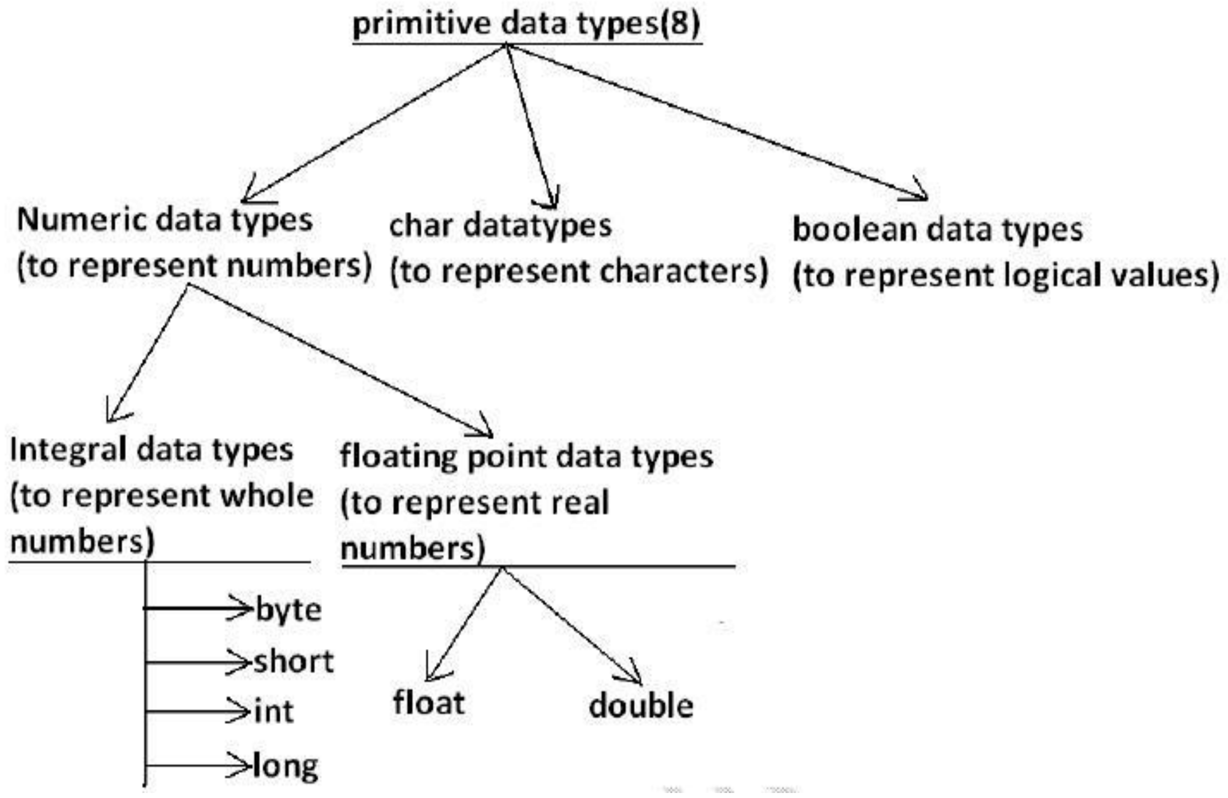
Datatypes are of two type:

1) Primative Data type (Already defined in Java)

This are the data type which are predefined in Java
For ex. Boolean, int, long, etc

2) Non Primative Data type(programmer defined)

Programmer defines it. Not already defined in Java, except String
For ex. Student, Employee,etc.



| Data type | Size | Range | Corresponding Wrapper class | Default value |
|---|---|---|---|---|
| Byte | 1 byte | $-2^{7}$ to $2^{7}-1$ (-128 to 127) | Byte | 0 |
| Short | 2 bytes | $-2^{15}$ to $2^{15}-1$ (-32768 to 32767) | Short | 0 |
| Int | 4 bytes | $-2^{31}$ to $2^{31}-1$ (-2147483648 to 2147483647) | Integer | 0 |
| Long | 8 bytes | $-2^{63}$ to $2^{63}-1$ | Long | 0 |
| Float | 4 bytes | -3.4e38 to 3.4e38 | Float | 0.0 |
| Double | 8 bytes | -1.7e308 to 1.7e308 | Double | 0.0 |
| Boolean | NA | Not applicable(but allowed values true\|false) | Boolean | false |
| Char | 2 bytes | 0 to 65535 | Character | 0(represents blank space) |

# *Class and Objects:*

## ***Class:***

- Class is a blueprint or template from which objects are created.
- Object is an instance (i.e. example) of class.

### ***Why we write a class:***

  - Java is an object-oriented programming language.
  - To create an object first, we need to define the properties which that object should have.
  - These properties and behavior which we want to impart to the object are declared in class. Hence class is called as blueprint of objects.

## ***Coding standard for class name:***

- Usually class are nouns
- Class name starts with uppercase letter.
- If using multiple words, then each first letter of word should start in uppercase.
- For ex. class Student, class String, class StringBuffer, class StundentInformation etc.

## ***Type of classes:***

1) Build in classes
2) User defined classes

## 1) **Build in classes:**

- Java language has provided set of predefined classes within predefined package.

- This classes are required commonly in all type of projects hence they are provided by default in java.

- Majorly used building classes are:

    - java.lang.String
    - java.lang.Exception
    - java.lang.Object
    - java.lang.Class
    - java.util.Date
    - java.util.Scanner
    - java.util.HashMap
    - java.util.ArrayList ….etc…

**2)***Custom defined classes:*

This is user defined classes which are created by user as per their project requirement.

**How to declare a class**

Syntax:

<**Access Specifier**> **class** <**Class Name**>{

//class body here

}

For ex. public class Employee{}

## Components of class:

### 1)Fields:

1) Fields are used to define the properties of the object of class.
2) Fields are declared within class body.
3) Fields also means variables.

For ex.

```
public class Student{

 private double percent; private int rollNum;

}
```

### **2)Methods:**

Method is a collection of statement which defines a behavior of class object.

### **3)Constructor:**

Constructors are special type of method which are used for object creation.

## 4)Blocks:

1) Whenever we want to execute some code during class loading or object creation then we use blocks.
2) There are two blocks in java. Static blocks and instance blocks.

## 5)Nested / Inner class:

A class within a class in called as Inner class.

**Rules for class:**

1) A java class must have a class keyword.
2) Class name must start with uppercase and if using more than one word then each starting letter of word should be upper case.
3) There should not be any space or special character in class name. Only allowed special character are $ and _ (underscore)
4) Java class can only have public or default access specifier.

5) A class can extend only one parent class. By default all the class in java extends java.lang.Object class directly or indirectly.

6) Class can implement any number of interfaces separated by commas.

7) Class containing main method is known as main class and is the entry point of any java program.

## Objects:

1)Object is a real world entity which has its own property/state and behavior.

2)So object contains of the following things:

***a)State:***

This is represented by the attribute and properties of object

*b)Behaviour*:

This is defined by the methods of the object.
For ex.

Mobile is an object which has :

State/property such as colour, model number, ram,etc Behaviour such as calling, messaging, etc.

**How to create an Object:**

Syntax:

<Class Name> <object name> = new <Class Name>(); For ex.

If there is a student class then to create object of student class :

Student student = new Student();

Hello World Program:

Open IDE

Select File -> New -> Project -> Under Java select Java Project -> Under Project Name - >Enter any name you wish -> Finish

Right click on project name -> New -> Class -> Under Name enter the class name you wish -> Finish

Program:

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

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

Output :

```
Hello World
```

## ***Variable in Java:***

1) The variable is the basic unit of storage in a Java program.
2) A variable is a name given to a memory location. It is the basic unit of storage in a program.
3) Variable is a memory location name of the data.
4) A variable is defined by the combination of an identifier, a type, etc.
5) In addition, all variables have a scope, which defines their visibility, and a lifetime.

**Declaring a variable:**

3) In Java, all variables must be declared before they can be used.

**Syntax:**

*<Date-type>  <variableName> = value,< variableName>= value ;*

## ***Rules / Coding standard for variables/object name:***

1) Usually variable names are nouns.
2) Should starts with lowercase alphabet symbol and if it contains multiple words every inner word should starts with upper case character.(camel case convention)

For ex. student, studentInformation, employee, employeeAddress, etc

## **Types of variables:**

Variables in java can be classified in two ways:

Division 1 : Based on the type of value represented by a variable all variables are divided into 2 types. They are:

### 1. Primitive variables:

Primitive variables can be used to represent primitive values.
Example: `int x=10;`

**2.** Reference variables
Reference variables can be used to refer objects.
Example: Student student=**new** Student();

Division 2 : Based on the behavior and position of declaration all variables are divided into the following 3 types.
1) Instance variables
2) Local variables
3) Static variables

## ***Instance variables:***

1) If the value of a variable is varied from object to object such type of variables are called instance variables.
2) For every object a separate copy of instance variables will be created.
3) Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
4) Instance variables will be stored on the heap as the part of object.
5) Instance variables should be declared with in the class directly but outside of any method or block or constructor.

6) Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area. But by using object reference we can access instance variables from static area.
7)Instance variables also known as object level variables or attributes.

Example:
```
public class Demo {
	int rollNum =10;
	public static void main(String[] args){
	//System.out.println(rollNum); //C.E:non-static variable i cannot be
	referenced from a static context(invalid)
	Test test=new Test();
	System.out.println(test. rollNum);//10(valid)
	test.methodOne();
}
public void methodOne(){
		System.out.println(i);//10(valid)
}}
```

**Note:**
**For the instance variables it is not required to perform initialization, JVM will always provide default values.**

Example:
```
class Test{
	boolean isValid;
		public static void main(String[] args){
			Test test=new Test();
			System.out.println(test.isValid);//false
		}
}
```

-> Assignment -> Once ping personally->
Anwser sheet -> Mock Discussion and Interview
->
-> Mock Interview ->Sat/Sun
-> 117 ->data on google
Your group -> on that ->time mutually
Interview -> Friday

-> Daily status sheet ->
-> Group Discussion -> Start today
-> Join seesion using Laptop
-> SOP

==========================================
Methods:
-> Variable
-> Method -> code reusability

-> Method
-> A block of statements
-> Declare a method / Syntax:

```
<Access Specifier> <return type> <method name>(argument){
     //method body
     return ;
}
```

-> Coding standard method :
-> Identifier allapplicable
-> verb -> some action, running, addtion
verb+noun -> getSalary,

studentinformation() -> wrong
stduentInformation() ->
studentDetailInformation() ->variable

void -> empty -> method that is not going to return anything

If your method is going to return/output anything (except void) then you need to write return keyword, which is the last statement of your method

10line -> 10 th line -> return->

rollNum -> int
Name -> String

```
Student ->class ->
Method return -> Student Object
```

```
Return ->Datatype + void

==============
Method Types
->Two types
-> Static Method
-> Whenever ->static keyword
-> static method
-> main()
public static int demo(String [] args){

}
->Call this static-> 3ways
-> Non static/ Instance
-> static -> not using-> non static
->

Instance ->special static variable
==========================
Program

public class MethodDemo3 {

public static void main(String [] prashant) {

     MethodDemo3 methodDemo = new MethodDemo3();

     double d=methodDemo.average(11, 22, 33);

    //System.out.println("The average of 11,22,33 is -->"+d);

     methodDemo.average1(11,22,33);
}

   public double average(int a, int b, int c) {

          double averageResult = (a+b+c)/3;

          //System.out.println(averageResult);

          return averageResult; //50
  }

public void average1(int a, int b, int c) {

     double averageResult = (a+b+c)/3;
     System.out.println("Average 1 method " + averageResult);
```

```
---------program   MethodDemo2 -----------------
public class MethodDemo2
{

    int rollNum=25;

     public static void main(String [] args) {

     //Creating Object of your class
    MethodDemo2 methodDemo = new MethodDemo2();

    methodDemo.demo();

    System.out.println(methodDemo.rollNum);

}

//non static method

public void demo() {

System.out.println("Non static method");
}

}

--------------------------------------------------------------

public class MethodDemo {
```

## //Method call ->

## //Static method 3 ways ->

```
//1st way -> Creating the object of the class which

contain static int rollNum=25; //instance varible

public static void main(String [] args) {
//1st way
     MethodDemo methodDemo = new MethodDemo();

methodDemo.demoOne(); //Calling our demoOne method using
Object reference
System.out.println(methodDemo.rollNum);
2 nd way is by using class name MethodDemo.demoOne();
```

```
System.out.println(MethodDemo.rollNum);


   //3rd way -> directly use methoname/variable -> method same
class

demoOne();

System.out.println(rollNum);
}

public static void demoOne() {

System.out.println("In Demo One Method using 3rd way");
```