FUNDAMENTAL OF JAVA PROGRAMMING LANGAUGE

The Object Oriented programming language supports all the features of normal programming languages. In addition it supports some important concepts and terminology which has made it popular among programming methodology.

The important features of Object Oriented programming are:

- **Inheritance**
- **Polymorphism**
- **Data Hiding**
- **Encapsulation**
- **Data Abstraction**
- **Reusability**

Inheritance: it is the process by which object of one class acquire the properties or features of objects of another class. The concept of inheritance provide the idea of reusability means we can add additional features to an existing class without modifying it

POLYMORPHISM: A Greek term means ability to take more than one form. An operation may exhibited different behaviors in different instances. The behavior depends upon the types of data used in the operation.

Example: Operator Overloading Function Overloading.

Data Encapsulation: Combining data and functions into a single unit called class and the process is known as Encapsulation. Data encapsulation is important feature of a class

Data hiding: Through encapsulation data is not accessible to another class but only those function which are rapped in the class. This insulation of data by direct access is known as data hiding

Data abstraction: Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

Object: Objects are important runtime entities in object oriented method. They may characterize a location, a bank account, and a table of data or any entry that the program must handle.

Each object holds data and code to operate the data. Object can interact without having to identify the details of each other's data or code. It is sufficient to identify the type of message received and the type of reply returned by the objects.

Classes: A class is a set of objects with similar properties (attributes), common behavior (operations), and common link to other objects. The complete set of data and code of an object can be made a user defined data type with the help of class.

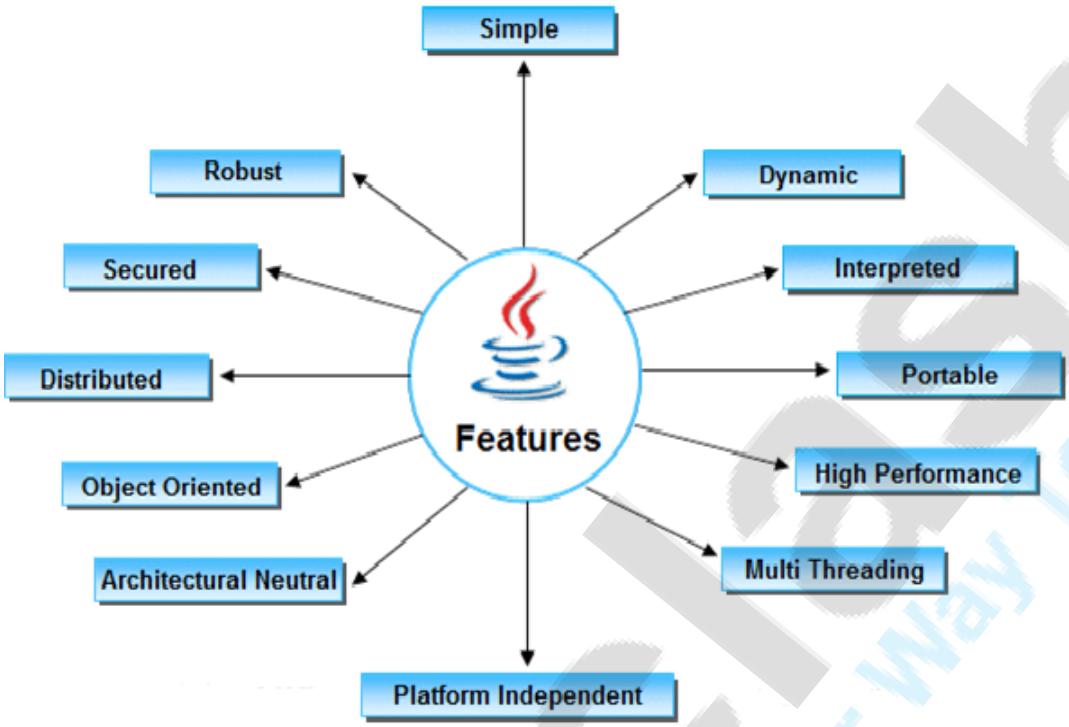
The objects are variable of type class. A class is a collection of objects of similar type. Classes are user defined data types and work like the build in type of the programming language. Once the class has been defined, we can make any number of objects belonging to that class. Each object is related with the data of type class with which they are formed.

HISTORY OF JAVA

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June **1991**. The small team of sun engineers called **Green Team.** Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called **"Greentalk"** by James Gosling and file extension was **.gt.**
- After that, it was called **Oak** and was developed as a part of the Green project.
- Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.
- **In 1995, Oak was renamed as "Java"** because it was already a trademark by Oak Technologies.
- The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
- According to James Gosling "Java was one of the top choices along with Silk". Since java was so unique, most of the team members preferred java.
- **Java is an island of Indonesia** where first coffee was produced **(called java coffee).**
- Notice that Java is just a name not an acronym.
- Originally developed by James Gosling 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.
- There are many java versions that has been released. Current stable release of Java is Java SE 8.
  - JDK Alpha and Beta (1995)
  - JDK 1.0 (23rd Jan, 1996)
  - JDK 1.1 (19th Feb, 1997)
  - J2SE 1.2 (8th Dec, 1998)
  - J2SE 1.3 (8th May, 2000)
  - J2SE 1.4 (6th Feb, 2002)
  - J2SE 5.0 (30th Sep, 2004)
  - Java SE 6 (11th Dec, 2006)
  - Java SE 7 (28th July, 2011)
  - Java SE 8 (18th March, 2014)

FEATURES OF JAVA



1. **Simple**

- It is simple because of the following factors:
- It is free from pointer due to this execution time of application is improved. [Whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
- It has Rich set of API (application protocol interface).
- It has Garbage Collector which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
- It contains user friendly syntax for developing any applications.

2. **Platform Independent**

   A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S).

3. **Architectural Neutral**

   Architecture represents processor. A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering their development and compilation.

4. **Portable**

   Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

5. **Multithreading**

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along

5. **Robust**

Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

6. **Object Oriented**

In java everything is Object which has some data and behavior. Java can be easily extended as it is based on Object Model.

7. **Distributed**

Using this language we can create distributed applications. RMI and EJB are used for creating distributed applications. In distributed application multiple client system depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.

8. **High performance**

It have high performance because of following reasons;

- This language uses Bytecode which is faster than ordinary pointer code so Performance of this language is high.
- Garbage collector, collect the unused memory space and improve the performance of the application.
- It has no pointers so that using this language we can develop an application very easily.
- It support multithreading, because of this time consuming process can be reduced to executing the program.

9. **Secure**

It is a more secure language compared to other language; in this language, all code is covered in byte code after compilation which is not readable by human.

10. **Networked**

It is mainly designed for web based applications, J2EE is used for developing network based applications.

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.

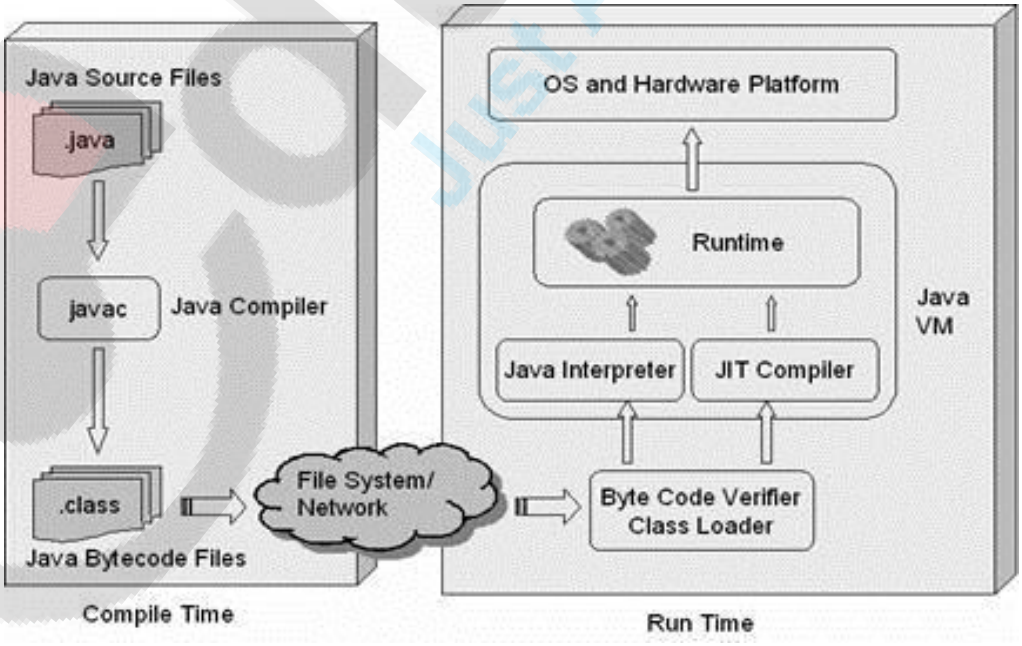
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

BASIC OF JAVA

Java is a platform independent, more powerful, secure, high performance, multithreaded programming language.

On compilation of java program The Java compiler generates hardware-independent bytecodes from the source code. These bytecodes can be executed only by a Java Virtual machine (JVM).

Java virtual Machine (JVM) is a virtual Machine that provides runtime environment to execute java byte code



Define byte

- Byte code is the set of optimized instructions generated during compilation phase and it is more powerful than ordinary pointer code.

Define JRE

- The Java Runtime Environment (JRE) is part of the Java Development Kit (JDK). It contains a set of libraries and tools for developing Java application. The Java Runtime Environment provides the minimum requirements for executing a Java application.

Define JVM

- JVM is set of programs developed by sun Micro System and supplied as a part of the JDK for reading line by line of byte code and it converts into a native understanding form of operating system. The Java language is one of the compiled and interpreted programming language.

Garbage Collector

- The Garbage Collector is the system Java program which runs in the background along with a regular Java program to collect un-Referenced (unused) memory space for improving the performance of our applications.

Define an API

- An API (Application Programming Interface) is a collection of packages, a package is the collection of classes, interfaces and sub-packages. A sub-package is a collection of classes, Interfaces and sub packages etc.
- Java programming contains user friendly syntax so that we can develop effective applications. In other words if any language is providing user friendly syntax, we can develop error free applications.

Definition of JIT

- JIT is the set of programs developed by SUN Micro System and added as a part of JVM, to speed up the interpretation phase.

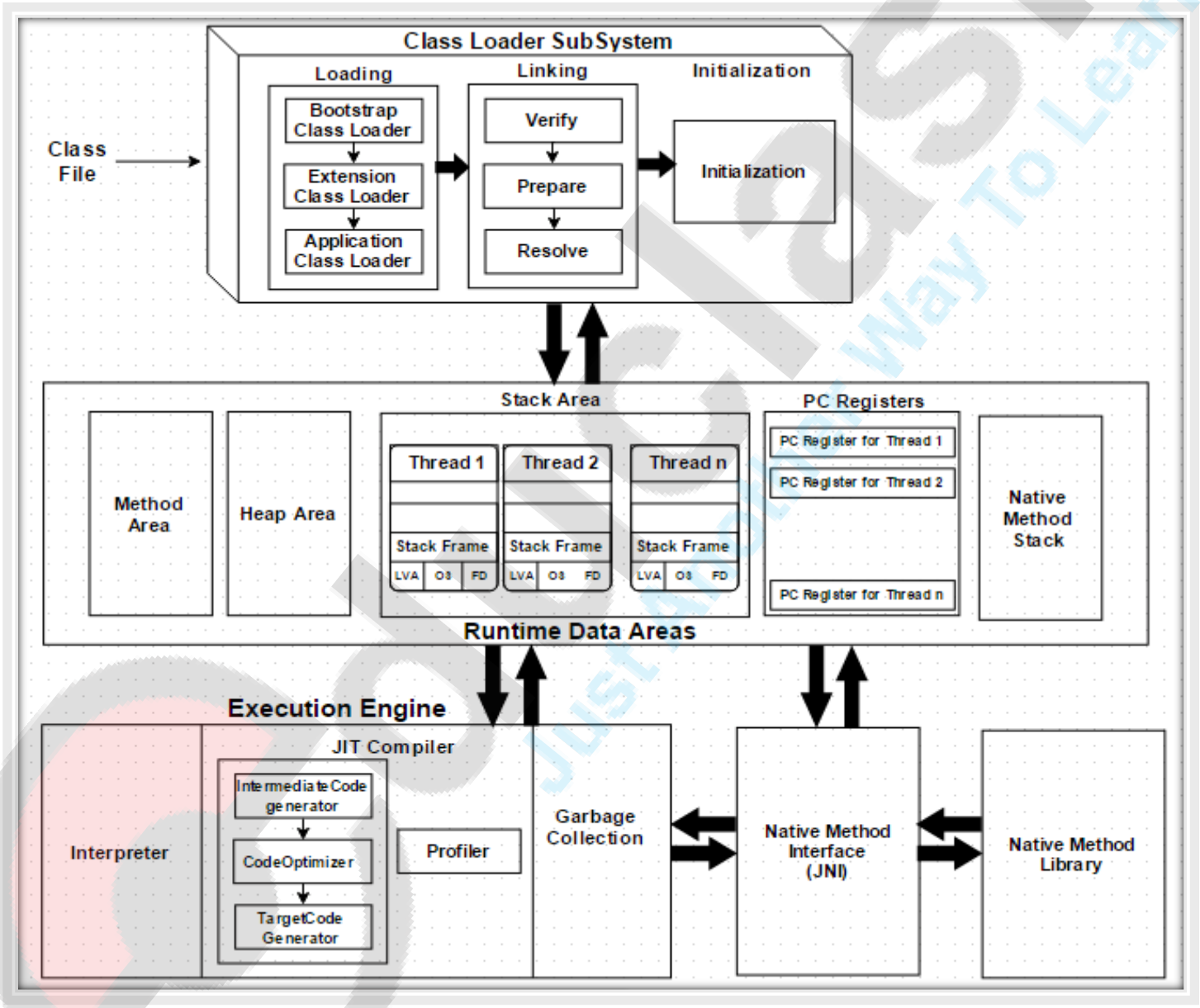
Network based application

- Network based application are mainly classified into two types.
- Centralized Applications
- Distributed Applications

JVM ARCHITECTURE

What is the JVM?

A Virtual Machine is a software implementation of a physical machine. Java was developed with the concept of WORA (Write Once Run Anywhere), which runs on a VM. The compiler compiles the Java file into a Java .class file, then that .class file is input into the JVM, which Loads and executes the class file. Below is a diagram of the Architecture of the JVM.



How Does the JVM Work?

As shown in the above architecture diagram, the JVM is divided into three main subsystems:

- **Class Loader Subsystem**
- **Runtime Data Area**
- **Execution Engine**

Class Loader Subsystem

Java's dynamic class loading functionality is handled by the class loader subsystem. It loads, links. And initializes the class file when it refers to a class for the first time at runtime, not compile time.

1) Loading

Classes will be loaded by this component. Boot Strap class Loader, Extension class Loader, and Application class Loader are the three class loader which will help in achieving it.

- Boot Strap Class Loader – Responsible for loading classes from the bootstrap class path, nothing but rt.jar. Highest priority will be given to this loader.
- Extension Class Loader – Responsible for loading classes which are inside ext folder (jre\lib).
- Application Class Loader –Responsible for loading Application Level Class path, path mentioned Environment Variable etc.

The above Class Loaders will follow Delegation Hierarchy Algorithm while loading the class files.

2) Linking

Verify – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.

Prepare – For all static variables memory will be allocated and assigned with default values.

Resolve – All symbolic memory references are replaced with the original references from Method Area.

3) Initialization

This is the final phase of Class Loading, here all static variables will be assigned with the original values, and the static block will be executed.

2) Runtime Data Area

The Runtime Data Area is divided into 5 major components:

Method Area – All the class level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.

Heap Area – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread safe.

Stack Area – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread safe since it is not a shared resource. The Stack Frame is divided into three sub entities:

Local Variable Array – Related to the method how many local variables are involved and the corresponding values will be stored here.

Operand stack – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.

Frame data – All symbols corresponding to the method is stored here. In the case of any exception, the catch block information will be maintained in the frame data.

PC Registers – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.

Native Method stacks – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

3. Execution Engine

The bytecode which is assigned to the Runtime Data Area will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

Interpreter – The interpreter interprets the bytecode faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.

JIT Compiler – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.

- Intermediate Code generator – Produces intermediate code
- Code Optimizer – Responsible for optimizing the intermediate code generated above
- Target Code Generator – Responsible for Generating Machine Code or Native Code
- Profiler – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.
- Garbage Collector: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling "System.gc ()", but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.
- Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.
- Native Method Libraries: It is a collection of the Native Libraries which is required for the Execution Engine.

DATA TYPES AND VARIABLES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are 3 types of variables

1) **Local variable**
2) **Instance variable**
3) **Class/static variable**

Local variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

```
public class Test {
   public void pupAge() {
      int age = 0;
      age = age + 7;
      System.out.println("Puppy age is : " + age);
   }
   public static void main(String args[]) {
      Test test = new Test();
      test.pupAge();
   }
}//here age is the local variable
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.

- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.

- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.

```java
import java.io.*;
public class Employee {

   // this instance variable is visible for any child class.
   public String name;

   // salary  variable is visible in Employee class only.
   private double salary;

   // The name variable is assigned in the constructor.
   public Employee (String empName) {
      name = empName;
   }

   // The salary variable is assigned a value.
   public void setSalary(double empSal) {
      salary = empSal;
   }

   // This method prints the employee details.
   public void printEmp() {
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]) {
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

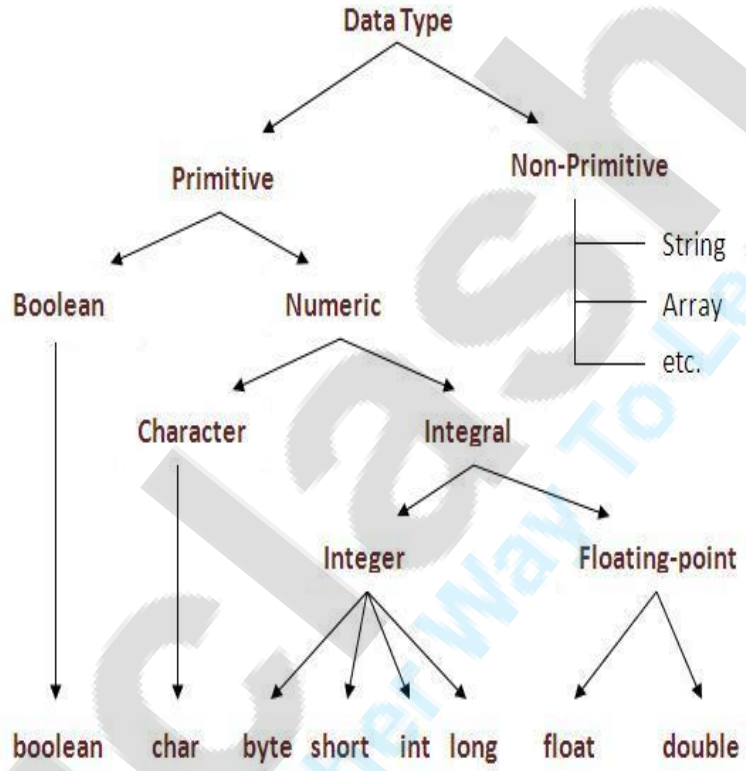
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

```
import java.io.*;
public class Employee {

   // salary  variable is a private static variable
   private static double salary;

   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";

   public static void main(String args[]) {
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + sala
   }
}
```

In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



Primitive datatype

- **Byte:** Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7) and Maximum value is 127 (inclusive) (2^7 -1) Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

Short data type is a 16-bit signed two's complement integer

- Minimum value is -32,768 (-2^15) and Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

Int data type is a 32-bit signed two's complement integer.

- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

Long data type is a 64-bit signed two's complement integer

- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

Float data type is a single-precision 32-bit IEEE 754 floating point

- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double data type is a double-precision 64-bit IEEE 754 floating point

- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean data type represents one bit of information

- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char data type is a single 16-bit Unicode character

- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character

```
class SizePrimitiveTypes
{
  public static void main (String[] args)
  {
    System.out.println("Size of byte:   " + (Byte.SIZE/8)      + " bytes.");
    System.out.println("Size of short:  " + (Short.SIZE/8)     + " bytes.");
    System.out.println("Size of int:    " + (Integer.SIZE/8)   + " bytes.");
    System.out.println("Size of long:   " + (Long.SIZE/8)      + " bytes.");
    System.out.println("Size of char:   " + (Character.SIZE/8)+ " bytes.");
    System.out.println("Size of float:  " + (Float.SIZE/8)     + " bytes.");
    System.out.println("Size of double: " + (Double.SIZE/8)    + " bytes.");
  }
}
```

```
C:\1Practical>javac SizePrimitiveTypes.java

C:\1Practical>java SizePrimitiveTypes
Size of byte:   1 bytes.
Size of short:  2 bytes.
Size of int:    4 bytes.
Size of long:   8 bytes.
Size of char:   2 bytes.
Size of float:  4 bytes.
Size of double: 8 bytes.

C:\1Practical>_
```

JAVA EXPRESSION

Expressions are **essential building blocks** of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable.

Expressions are built using values, variables, operators and method calls.

With the right punctuation, it can sometimes stand on its own, although it can also be a part of a sentence. Some expressions equate to statements by themselves (by adding a semicolon at the end) but more commonly, they comprise part of a statement.

For example, (a * 2) **is an expression**. b + (a * 2); **is a statement**. You could say that the expression is a clause, and the statement is the complete sentence since it forms the complete unit of execution.

A statement doesn't have to include multiple expressions, however. You can turn a simple expression into a statement by adding a semi-colon: (a * 2);

TYPES OF EXPRESSIONS

While an expression frequently produces a result, it doesn't always. There are three types of expressions in Java:

1) Those that produce a value, i.e. the result of (1 + 1)
   - 3/2
   - 5% 3
   - pi + (10 * 2)

2) Those that assign a variable, for example (v = 10)

3) Those that have no result but might have a "side effect" because an expression can include a wide range of elements such as method invocations or increment operators that modify the state (i.e. memory) of a program.

   While some expressions produce no result, they can have a side effect which occurs when an expression changes the value of any of its operands.
   For example, certain operators are considered to always produce a side effect, such as the assignment, increment and decrement operators. Consider this int product = a * b;
   The only variable changed in this expression is product; a and b are not changed. This is called a side effect.

The Java programming language allows you to construct compound expressions from various smaller expressions as long as the data type required by one part of the expression matches the data type of the other.

Here's an example of a compound expression: **1 * 2 * 3**

In this particular example, the order in which the expression is evaluated is unimportant because the result of multiplication is independent of order; the outcome is always the same

However, this is not true of all expressions. For example, the following expression gives different results, depending on whether you perform the addition or the division operation first:

x + y / 100 // ambiguous

You can specify exactly how an expression will be evaluated using balanced parenthesis: (and). For example, to make the previous expression unambiguous, you could write the following:

(x + y) / 100 // unambiguous, recommended

JAVA OPERATORS

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

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

JAVA UNARY OPERATOR EXAMPLE: ++ AND --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10  12  12   10
```

JAVA UNARY OPERATOR EXAMPLE 2: ++ AND --

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21
 }}
```

Output:

```
22   21
```

JAVA UNARY OPERATOR EXAMPLE: ~ AND !

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
```

```
System.out.println(~a);//-11 (minus of total positive value which starts from 0)

System.out.println(~b);//9 (positive of total minus, positive starts from 0)

System.out.println(!c);//false (opposite of boolean value)

System.out.println(!d);//true

}}
```

Output:

```
-11     9       false         true
```

JAVA ARITHMETIC OPERATOR EXAMPLE

```
class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

System.out.println(a+b);//15

System.out.println(a-b);//5

System.out.println(a*b);//50

System.out.println(a/b);//2

System.out.println(a%b);//0

}}
```

Output:

```
15     5     50     2     0
```

JAVA ARITHMETIC OPERATOR EXAMPLE: EXPRESSION

```
class OperatorExample{

public static void main(String args[]){

System.out.println(10*10/5+3-1*4/2);

}}
```

Output:

```
21
```

JAVA SHIFT OPERATOR EXAMPLE: LEFT SHIFT

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40    80    80    240
```

JAVA SHIFT OPERATOR EXAMPLE: RIGHT SHIFT

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2      5      2
```

JAVA SHIFT OPERATOR EXAMPLE: >> VS >>>

```
class OperatorExample{
public static void main(String args[]){                //For positive number, >> and >>> works same
  System.out.println(20>>2);
  System.out.println(20>>>2);                          //For nagative number, >>> changes parity bit (MSB) to 0
  System.out.println(-20>>2);
  System.out.println(-20>>>2);  }}
```

Output:

```
5    5      -5      1073741819

20
```

JAVA AND OPERATOR EXAMPLE: LOGICAL && AND BITWISE &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a<b&&a<c);//false && true = false

System.out.println(a<b&a<c);//false & true = false

}}
```

Output:

```
False      false
```

JAVA AND OPERATOR EXAMPLE: LOGICAL && VS BITWISE &

```
class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a<b&&a++<c);//false && true = false

System.out.println(a);//10 because second condition is not checked

System.out.println(a<b&a++<c);//false && true = false

System.out.println(a);//11 because second condition is checked

}}
```

Output:

```
False        10       false        11
```

JAVA OR OPERATOR EXAMPLE: LOGICAL || AND BITWISE |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a>b||a<c);//true || true = true

System.out.println(a>b|a<c);//true | true = true

//|| vs |

System.out.println(a>b||a++<c);//true || true = true

System.out.println(a);//10 because second condition is not checked

System.out.println(a>b|a++<c);//true | true = true

System.out.println(a);//11 because second condition is checked

}}
```

Output:

```
True    true    true    10    true    11
```

JAVA TERNARY OPERATOR EXAMPLE

```
class OperatorExample{

public static void main(String args[]){

int a=2;

int b=5;

int min=(a<b)?a:b;

System.out.println(min);

}}
```

Output:

```
2
22
```

ANOTHER EXAMPLE:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

5

JAVA ASSIGNMENT OPERATOR EXAMPLE

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output: 14 16

JAVA ASSIGNMENT OPERATOR EXAMPLE

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
```

```
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:

13 9 18 9

JAVA ASSIGNMENT OPERATOR EXAMPLE: ADDING SHORT

```
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
//a+=b;//a=a+b internally so fine
a=a+b;//Compile time error because 10+10=20 now int
System.out.println(a);
}}
```

Output:

Compile time error

AFTER TYPE CAST:

```
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
a=(short)(a+b);//20 which is int now converted to short
System.out.println(a);
}}
```

Output: 20

CONTROL STRUCTURE

Control Statements

1) Java If-else
    a. Java IF Statement
    b. Java IF-else Statement
    c. Java IF-else-if ladder Statement
2) Java Switch
3) Java For Loop
    a. Simple For Loop
    b. For-each or Enhanced For Loop
    c. Labeled For Loop
4) Java While Loop
5) Java Do While Loop
6) Java Break
7) Java Continue
8) Java Comments

THE JAVA IF STATEMENT

It is used to test the condition. It checks boolean condition: true or false. There are various types of if statement in java.

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

Output is : Age is greater than 18

```
public class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");break;
    case 20: System.out.println("20");break;
    case 30: System.out.println("30");break;
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

JAVA SWITCH STATEMENT

The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

Output : 20

```
public class IfExample {
public static void main(String[] args) {
    int age=20;
    if(age>18){
        System.out.print("Age is greater than 18");
    }
}
}
```

FOR LOOP

Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

For-each loop

It is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation. It works on elements basis not index. It returns element one by one in the defined variable.

```
public class ForEachExample {
public static void main(String[] args) {
    int arr[]={12,23,44,56,78};
    for(int i:arr){
        System.out.println(i);
    }
}
}
```

Output:

```
12
23
44
56
78
```

Java Labeled For Loop

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

```
public class LabeledForExample {
public static void main(String[] args) {
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        break aa;
                    }
                    System.out.println(i+" "+j);
                }
        }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
```

JAVA WHILE LOOP

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

JAVA DO-WHILE LOOP

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java do-while loop is executed at least once because condition is checked after loop body.

```
public class DoWhileExample {
public static void main(String[] args) {
  int i=1;
  do{
    System.out.println(i);
  i++;
  }while(i<=10);
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

ARRAYS IN JAVA

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Advantage of Java Array

Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.

Random access: We can get any data located at any index position.

Disadvantage of Java Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

- Single Dimensional Array
- Multidimensional Array

SYNTAX TO DECLARE AN ARRAY IN JAVA

dataType[] arr;

dataType []arr;

dataType arr[];

INSTANTIATION OF AN ARRAY IN JAVA

arr = new datatype[size];

```
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
```

DECLARATION, INSTANTIATION AND INITIALIZATION OF JAVA ARRAY

int a[]={33,3,4,5};

```
int a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
```

PASSING ARRAY TO METHOD IN JAVA

```
class Testarray2{
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];

System.out.println(min);
}

public static void main(String args[]){

int a[]={33,3,4,5};
min(a);//passing array to method

}}
```

Output:3

SYNTAX TO DECLARE MULTIDIMENSIONAL ARRAY IN JAVA

dataType[][] arr;

dataType [][]arr;

dataType arr[][];

dataType []arr[];