11.12.19

WEB TECHNOLOGY

SUBJECT CODE - IT 7402

Unit 1, 2 → JAVA, Threads (OS), OOPS concepts

Unit 3 → HTML, XML, Angular JS

Unit 4 → Server side concept of JAVA, JDBC connectivity, Front-end Back-end connectivity

Unit 5 → Python basics, Classes and Objects, OOPS concepts, Web. App., Connectivity, Sample Working Code

Books -	JAVA Complete Reference by Herbert Smith (Programs and Solutions in Net)	} Unit 1, 2
	Web Technologies by Uttam K. Roy	Unit 3, 4
	Any Python Book (or) TutorialsPoint	Unit 5

1st Assessment Portions → Unit 1, Unit 2 (Some), Unit 3 (HTML, JavaScript part)

Order of teaching → Unit 1, 2, 3, 5, 4

⊛ Marks for Notes in both Assessments
No assignments
If boost is needed, Q.Paper programs' execution in lab will be taken.

11.12.19 UNIT I - JAVA BASICS

⊗

⇒ Java is completely Platform independent.
C++ Platform dependent

⇒ Java doesn't support multiple inheritance. It supports interface.
C++ supports multiple inheritance.

⇒ Java doesn't support operator overloading.
C++ supports operator overloading.

⇒ In Java, no pointer support. Only restricted pointer usage.
C++ supports pointers.

⇒ In Java, no structures and unions.
C++ has structures and unions.

⇒ In Java, no virtual keyword. Here static and non-static methods. All non-static methods can be overridden by default.

$\text{static} \downarrow$ no objects needed

$\text{non-static} \downarrow$ accessed using objects

C++ has virtual keyword.

⇒ Java programs used for application programming
C++ for system programming.

Differences b/w Java and C++: ↑

Scanned by CamScanner

Editions in Java :

→ Standard → Simple application programming

→ Enterprise → Developing n/w applications. App. distributed in different places (eg.) Banking application
EJB (Enterprise Java Beans)

→ Micro → Developing mobile applications (or) embedded systems

JavaFX (Advanced concept of Swings)
Internet applications

Family of JAVA :

Derived from C and C++
C ↓ Syntax
C++ ↓ OOPS concepts

History of Java :

Started by Green team. Initially it was supposed to be used in Cable TV. But it was advanced for it. So, they switched to Internet programming.

Initially named as OAK. Later JAVA (Not an acronym)
JAVA ↓ Its an island's name in Indonesia

Initial extension .gt. Now java

Why coffee? Coz in Java island, coffee production is famous.

In 1996, JDK 1.0
Now upto JDK 10.0 (or) 11.1.0
In lab JDK 8. 9 can also be used

Features of Java :

→ Very simple (Syntax of C++) (Confusing features of C++ are removed)

→ Automatic garbage collection (No destructor like in C++)

→ All basic concepts of OOPS as in C++ (Objects, Class, Inheritance, Polymorphism, Encapsulation, Abstraction)

→ Platform independent

→ Class file (bytecode) can be seen whenever a java code is compiled. This bytecode will be converted to machine language and executed. Since for execution, we use only class file, it is platform independent.

Rev. JVM Java Virtual Machine

JDK Java Development Kit

JRE Java Runtime Environment

→ Java is very secure coz it has no explicit pointers in it.

→ Java is very robust coz it has good memory management and it has automatic garbage control

→ Effective exception handling

→ Interpreter in Java → interpretting the byte code.

→ Since many processes are dynamic, Java is considered slower. But there is JIT → reduce the time utilized dynamically

→ Multi-threading concept in Java (Multiple processes)

Basic snippet of Java :

```
class <class_name>
{
    public static void main (String args[])
    {
        System.out.println("...");
    }
}
```

public → access specifier
void → return type
main → method

System.out ↓ instance of printstream class available in java.io

Why main is void ?
Since once main is executed, we will be exiting out of entire program and thus can't return anything.

Why main is static ?
No need for objects to access it since it belongs to class.

main is the keyword denoting entry point to the program in JVM.

(String args[ ]) ↳ Any name can be given

Access specifiers in Java :

→ default → default access specifier
→ private
→ protected
→ public

Command Prompt :

Compile : javac filename.java

Execute : java filename

filename and classname in program should be same.
Class file : filename.class

System :

Java has no header files. It only has packages that can be imported.

system is a class in java.lang

println : method of printstream class in java.io

12.12.19 Compile : javac filename.java
Execute : java classname

Compile Time :

Java code → byte code
sample.java → sample.class

JIT (Just In Time) compile

Run Time :

$$\begin{tikzcd} \boxed{\text{Class file}} \arrow[d] \\ \boxed{\text{Class loader}} \arrow[d] \\ \boxed{\text{Bytecode verifier}} \arrow[d] \\ \boxed{\text{Interpreter}} \end{tikzcd}$$

Class loader → 3 types
In general, we use application / standard class loader

Bytecode verifier → Checks byte code for run time error

Interpreter → Converts byte code to machine language suitable to given os

JVM (Java Virtual Machine)

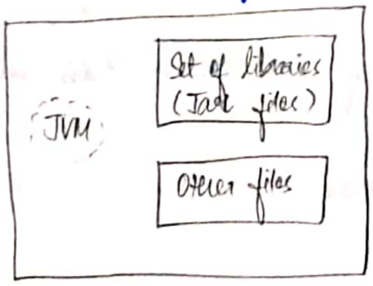
→ Provides run time environment for Java byte code to be executed.

→ Can also run those programs written in other languages and compiled to Java bytecode. (Eg) C code converted to java byte code.

JRE (Java Runtime Environment)

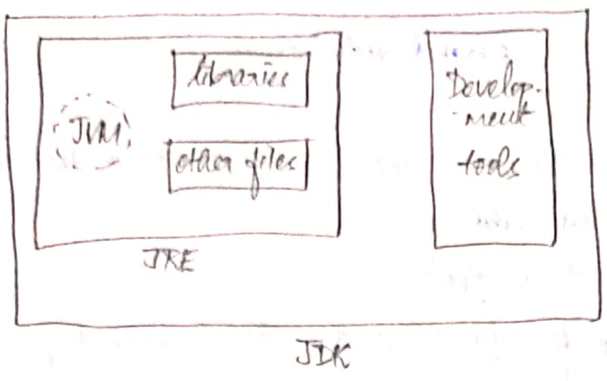
→ Provides run time environment

→ Set of libraries + Other files that Java uses at run time



Jar files
.jar extension
(supporting libraries)
present in JRE folder

JDK (Java Development Kit)

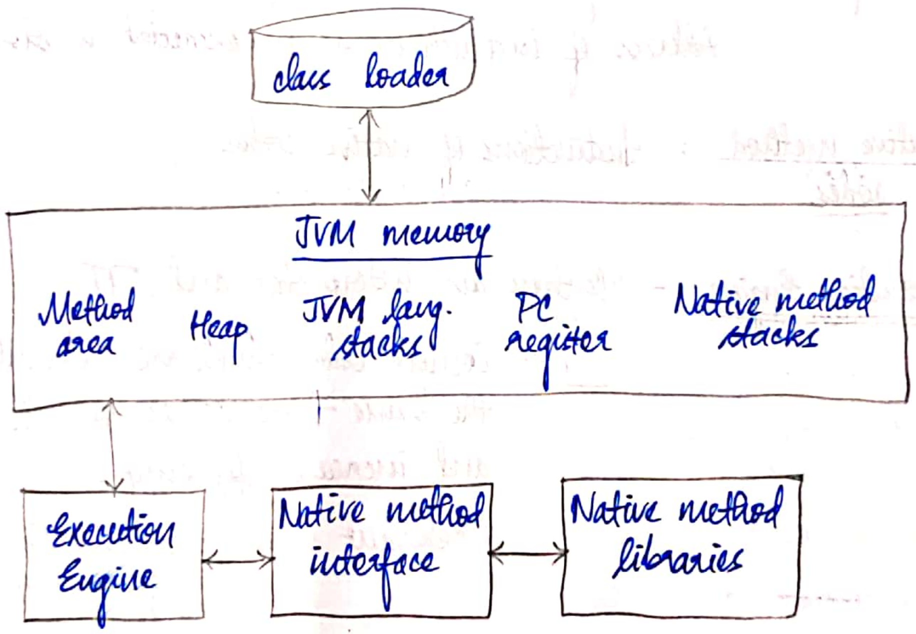


All JDK, JRE and JVM are platform dependent (ie configuration depends on OS)

Only Java code is platform independent

Includes → interpreter / loader
→ compiler
→ archiver
→ document generator

JVM architecture :



Class loader - loads the

JVM Memory - separated into areas

Method area : Class structure stored as metadata inside the method area.
Run time tools
Code for functions are stored separately

Heap : All object related instance variables and arrays are stored here

JVM language stacks : All local variables are stored in this stack
Partial results

PC register : PC (Program counter)
Address of instructions to be executed is stored here

Native method stacks : Instructions of native codes

Execution Engine - Contains an interpreter and JIT
JIT : Compile codes which are relevant at the same time so as to reduce time and increase efficiency.
(Relevant codes → Codes with same functionality)

Native method interface - Libraries to run the codes (native) and supporting libraries are manipulated here

Native method libraries - Libraries and supporting libraries information are stored here

Object Oriented Concepts

→ Objects (Instance of a class)
  Attributes / Behaviour

→ Methods and Classes
  ↓ Perform tasks   ↓ Program units

→ Instantiation (The term to create instance)

→ Reuse

→ Encapsulation and Information hiding

→ Inheritance (deriving properties from parent class)

→ Interfaces (saying what is going to be done, doesn't explain how it is done)

→ Polymorphism

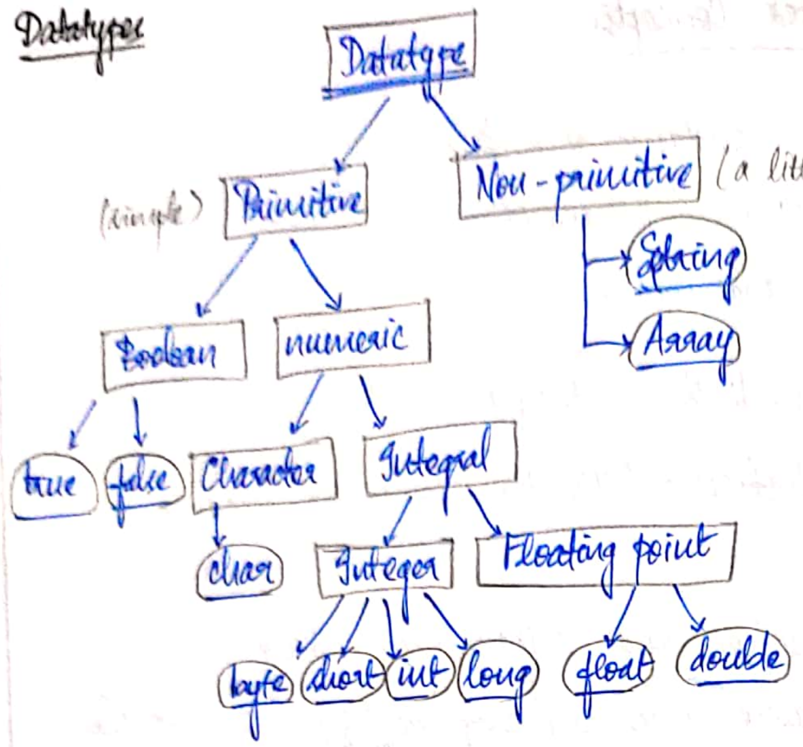
Variable Declaration

datatype var_name ;

→ local variable
→ instance variable
→ static variable

```
class a {
  int data = 50 ;            // instance variable (outside method, inside class) → need objects to access them
  static int m = 100 ;       // static variable → no need of objects to access them
  void m() {
     int n = 90 ;            // local variable ( inside method )
  }
}
```

Datatypes



In Java, no garbage value for variables.

There are default values for each data type

Operators

→ Unary (Increment, Decrement, Not)

→ Arithmetic (+, -, /, *, %)

→ Shift (>>, <<)

→ Relational (>, <, >=, <=, instanceof) → Checks whether object is instance of class or not

→ Bitwise (&, |, ~)

→ Logical (AND, OR, NOT → &&, ||)

→ Ternary

→ Assignment (=, +=, -=, *=, /=, >>=)

Control statements

→ if (condition) {...}

→ if..else

```
if (condition)
{...}
else
{....}
```

Scanned by CamScanner

→ if...else if

```
if (condition)
{... }
else if (condition)
{... }
else
{... }
```

→ switch

```
switch( )
{
   case 0 : ....
              break ;
   default : ....
              code ;
}
```

Loop statements

→ while

```
while (condition)
{... }
```

→ do..while

```
do
{... } while (condition) ;
```

→ for

```
for ( initialization; condition ; inc./dec. )
{... }
```

→ for...each

```
for (datatype var_name : array)
{... }
```

⇒ Extended version of for
⇒ No conditions
⇒ Automatic increment only (no decrement)
⇒ Used for arrays (or) collection
⇒ Similar to foreach in PHP

Getting input from user

```
import java.util.* ;
class addition {
    public static void main(String args[])
    {
        Scanner i = new Scanner(System.in) ;
        int x1 ;
        int x2;
        int sum ;
        System.out.println("Enter 1st integer : ") ;
        x1 = i.nextInt() ;
        System.out.println("Enter 2nd integer : ") ;
        x2 = i.nextInt() ;
        sum = x1 + x2 ;
        System.out.println("Sum is "+sum) ;
    }                                  Concatenation operator
}
```

```
import java.util.* ;
class temp {
    public static void main(String args[])
    {
        Scanner s = new Scanner(System.in) ;
        System.out.println("Enter String for C : ");
        String c =s.next~~Line~~() ;
        System.out.println("C is "+c) ;
        System.out.println("Enter String for D : ");
        String d = s.next~~Line~~() ;
    }   System.out.println("D is "+d) ;
}
```

→ nextLine() ;
→ next().CharAt(0) ;
→ nextDouble() ;

} → Methods (Functions) of Scanner class

Object declaration : classname obj = new classname() ;

To use a class, we need to import its package.
For Scanner, package is util.
* indicates all classes.

$$\begin{tikzcd}\boxed{\text{import java . util . * ;}} \arrow[d] \\ \text{Package} \quad \text{Sub-package} \quad \text{All classes}\end{tikzcd}$$

```
import java . util . * ;
```
java → Package
util → Sub-package
* → All classes

You can specify class name separately also.

Scanner class is used to get input from user.

```
Scanner i = new Scanner ( System . in ) ;
```
System → Class available in lang package (default package)

<u>in, out</u>

Objects of printstream class (or) static instances of System class

```
x1 = i. nextInt() ;
```
nextInt() → To get an integer only. If input is not an integer, then it throws an error (or) exception

(*) Java is Case sensitive

next() } → Accepts string only
nextLine() }

next().CharAt(0) → Accepts 1st character

nextDouble() → Accepts Double only

Difference b/w next() and nextLine().

(Eg:) Welcome to IT dept

next() will accept Welcome only and cursor remains there itself. So again when it is uses, it will accept to only and so on.

nextLine() accepts the entire line and cursor will be placed in the next new line.

For accessing files, we have separate classes.

System.in → Accepts keyboard input
filename.in → Accepts input from file

System.out.println (sum) ;   // Prints sum without any text