Linkdin : sachinshetkar

Language Fundamentals :

1.Identifiers

A name in java program is called "Identifier"

The name can be classNameinterfaceNameenumNameVariableNamemethodName and LabelName

```
Class Test{
    public static void main(String[] args){

    }
}
```

Identifier : Test , main , String , args , x

Rules for Identifiers:

Rule 1: The allowed characters in java identifiers are

a to z , A to Z , 0 to 9 , _ , $

Rule 2 : If we use any other character, then the program would result in "ComplieTime Error"

ABC123 - Valid

123ABC - Not Valid

Rule 3 : Identifiers should not starts with digits.

```
int number = 10;
int NUMBER = 10;
int nuMbeR = 10;
```

JVM will execute this . And JVM will Treat data in Case Sensitive Manner.

Rule 4 : Java Identifiers are case sensitive ,as such Java Lang is case sensitive(JVM).

Rule 5 : There is no constraint of length of the java identifiers, but it is not recommended to take the length more than 15

Eg : int sachinsubryarushiarun = 10;

```
Class Test{
    public static void main(String… args){
        int int = 10;//CE
        int if = 10;//CE
        int while = 100;//CE
    }
}
```

Rule 6 : Reserve words or built in words can't be used as " java identifiers"

If you try to use then it would result in "CompileError".

Ex : int ,if ,float, else,while,for , do ,… => reserve words/built in words/keywords

```
Class Test{
```

```
        public static void main(String… args){
                int String = 10;//No Error }
}
```

Inbuilt className,interfaceName,enumName can be used as “identifier”, but we don’t recommend it will create confusion.

Practice :

1. _$_ => Valid
2. ca$h =>valid
3. java2share =>valid
4. all@hands =>invalid
5. 123abc =>invalid
6. Total# => invalid
7. Int =>valid(not recommended)
8. Integer =>valid(not recommended)
9. int => invalid(reserve word)
10. tot123 => valid.

Basic points about MLL,ALL,HLL:

Microprocessor == CPU

Any programs/Instructions has to executed it will take help of Microprocessor.

Microprocessor will understand only 0’s and 1’s.

MP will take input from RAM (primary unit of memory) do some processing and give data to RAM.

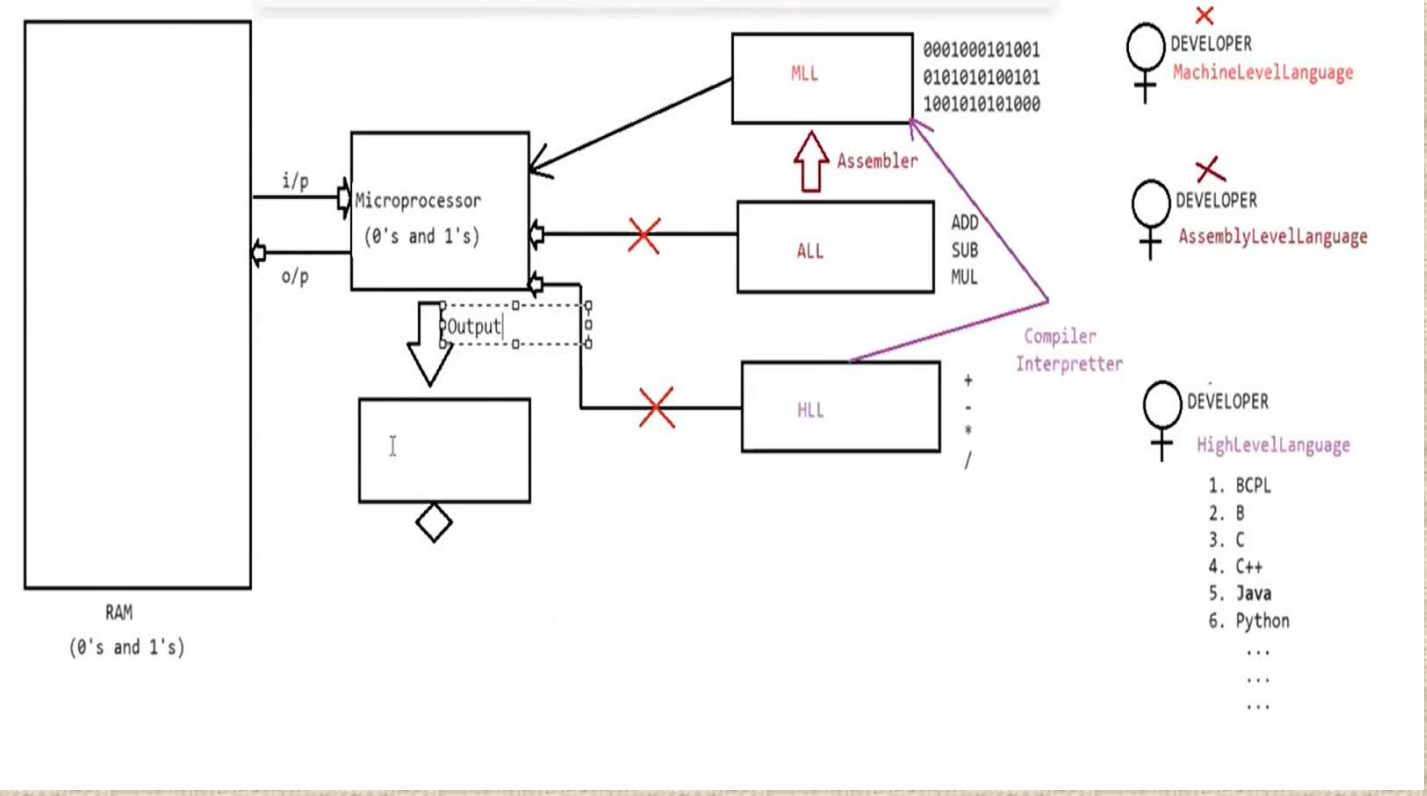
Any program will execute on RAM only.

MP reads instructions from RAM.

If you give 0101 program directly to MP it will execute . But as developer it will be hectic we can't do in 0101(Machine level language) .

Instead of 0101 we write ADD,SUB,MUL it is far better than MLL . we can't give this directly to CPU it will not understand. We have to Assembly code to

Assembler it will generate 0101 code. But writing Assembly lang program for bigger program it will problem for developer.



Developer Perspective.

So Developer will write a code or instructions in symbolic style (+ - * /) i.e in

High Level Language (it means easy human readable) we can't this as i/p to CPU it will not execute so we need compiler/interpreter it will convert HLL to MLL .

Conclusion: As a developer we write code in HLL .

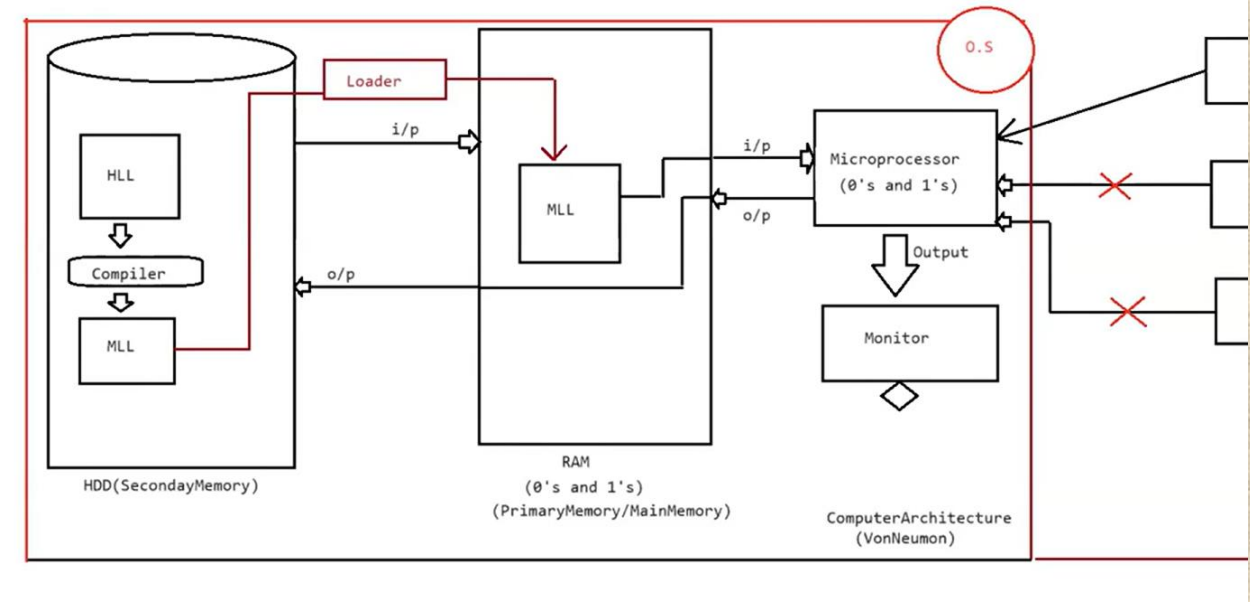
Ex : 1.BCPL 2.B 3.C 4.C++ 5.Java 6.Python ……

We will learn Java.

We can't write code in directly to RAM(Primary/Main Memory) . we have one more memory HDD(Secondary Memory). All our programs(HLL) will stored in HDD.

We can't put this HLL Code directly into RAM for that we require compiler so it also stored in HDD.

HLL=>Compiler=>MLL



Computer Architecture

All this are H/W Devices are Dumped devices without software it can't be operational . For this we require software that is Operating System.

Ex : We will take i/p from HDD or give to RAM . i.e Loader worker of OS.

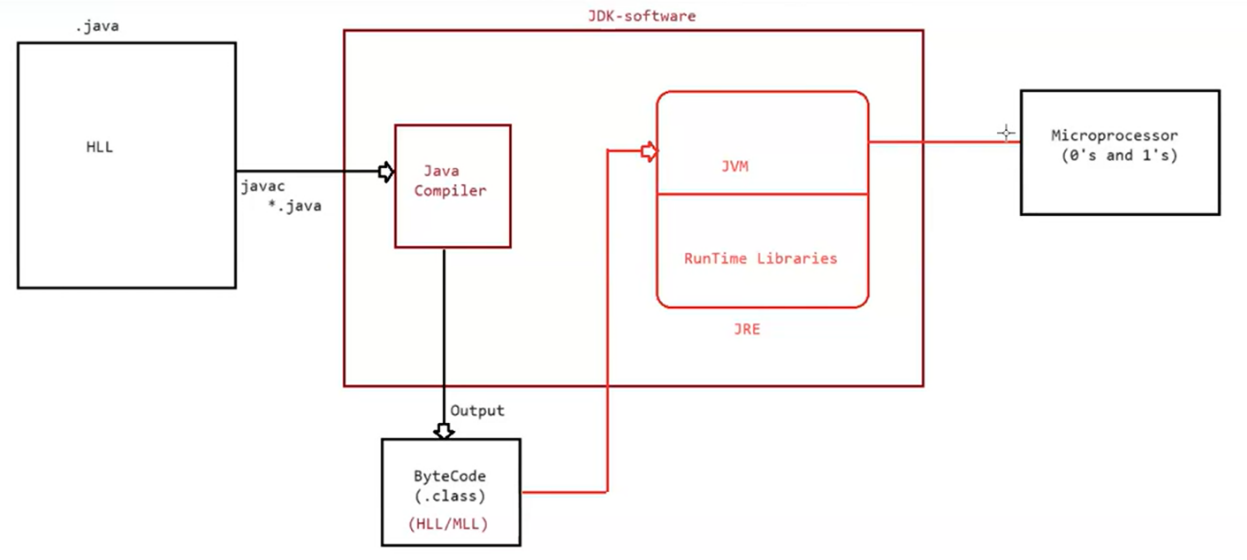
We can send output to Monitor and HDD .

Overview According To JAVA :

Software support for developing and running java is JDK-Software.

Compiler – Java Compiler (javac)

HLL => javac(Java Compiler) => ByteCode (.class) => JVM => 010101 => CPU => o/p (on Monitor or HDD or Cloud or DB)



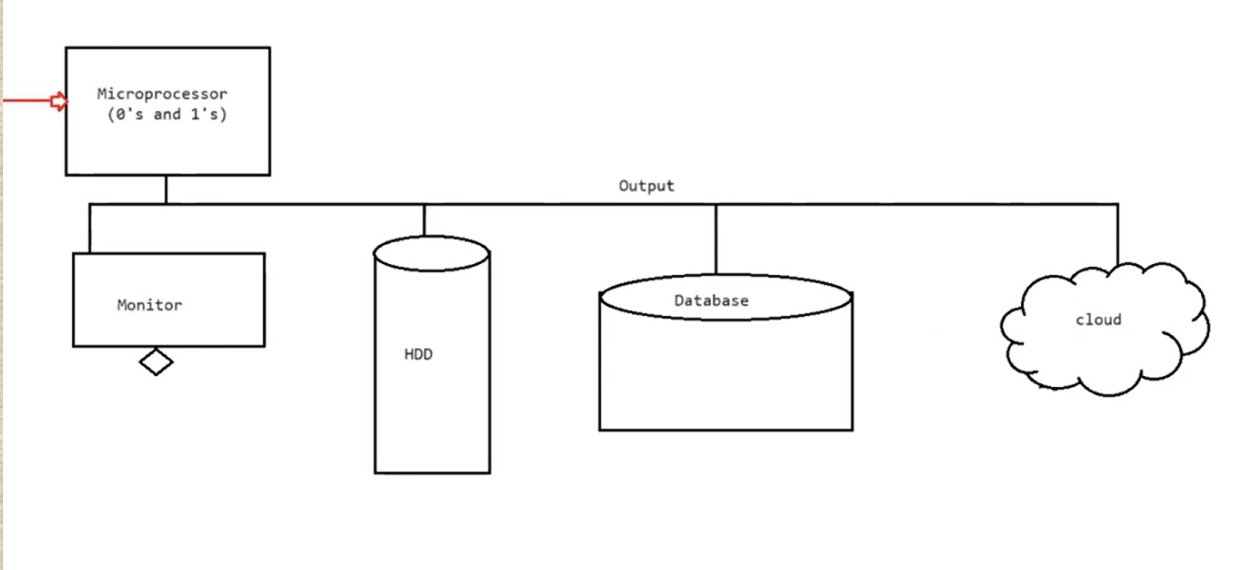
JDK = Java Complier + JRE (JVM + Runtime Libraries)

JDK => Java Development Kit (Loader ,Linker are also available in JDK but internally use)

JRE => Java Runtime Environment

JVM => Java Virtual Machine

Output from CPU we can take on :



-Compile Time Error: If compile find something wrong related to syntax, then it will give CTError.

-When we are executing bytecode if something problem comes, we will get Exception.

We are having software like Compiler and Interpreter.

1.Complier

It will compile all the lines in .java file and generate the bytecode (.class file).

2.Interpreter

It will take one line at a time and read generates one line instructions.

And it will go to second line again till last line.

3.JVM

JVM will read one by one line from bytecode and executes. It will executing interpreted mode.

Java program happens in two phase compilation and interpreted . i.e in Hybrid manner.