Java - Programming Language

→ Desktop Application

→ Web Application

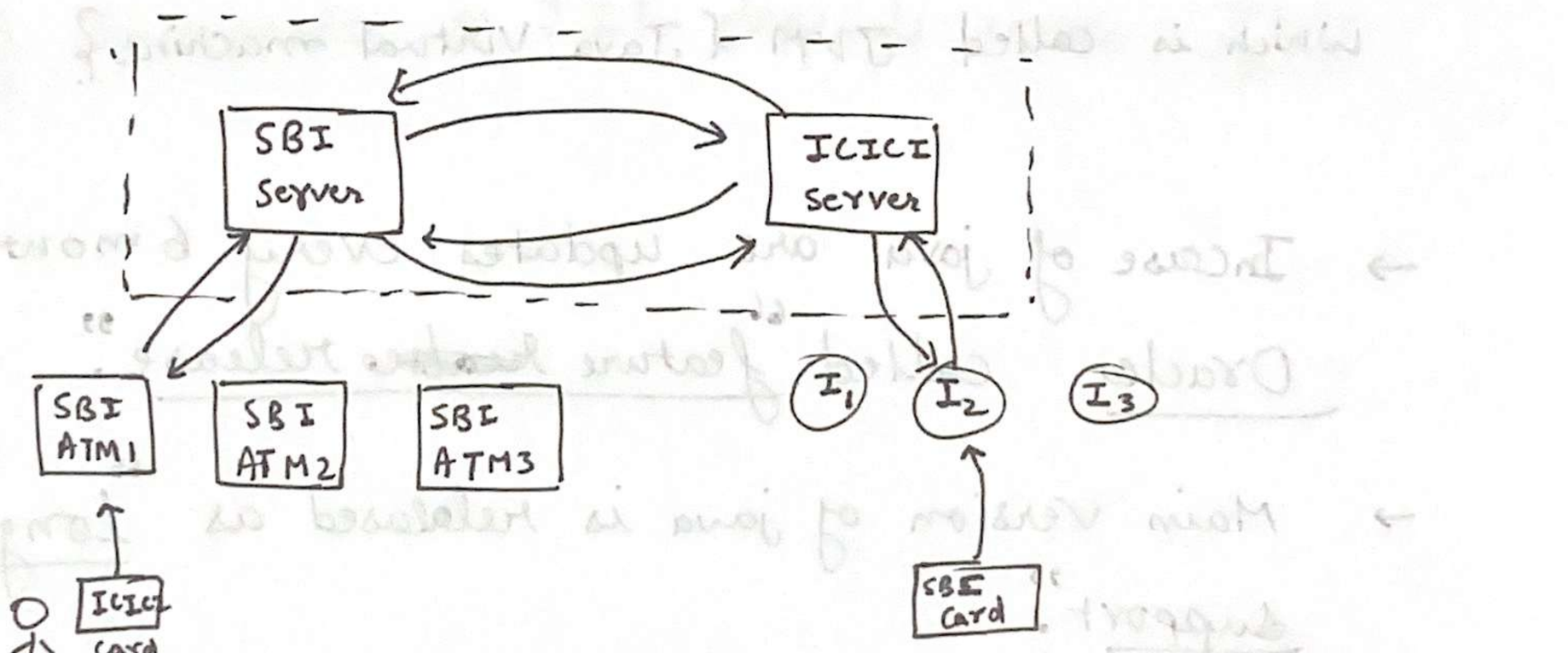
→ Applets {gets loaded to the client side (also runs on the client side) from server side}

- → JDK-9 {from this version applets are phased out.
- → JDK-11 {from} applets are no longer supported.
- → from JDK-9 Jlink was added.
- → from JDK-16 jPackage was added.

→ [In place of Applets]

→ Distributed applications.

– Programs distributed over multiple machines.

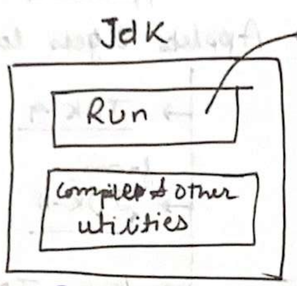


RMI {Remote method Invocation} - (Part of distributed application)

JDK, JRE & JVM

JDK → Java development kit → {Latest version - 21}

To install Java in a machine we need to install jdk.



{Both JDK & JRE are products (can be downloaded) JVM is not}

{To run Java application we just need JRE}

→ JRE creates a virtual environment over our O/S which is called JVM {Java Virtual machine}.

→ Incase of java are updates every 6 months by Oracle called "feature ~~release~~ release".

→ Main version of java is released as "long term support".

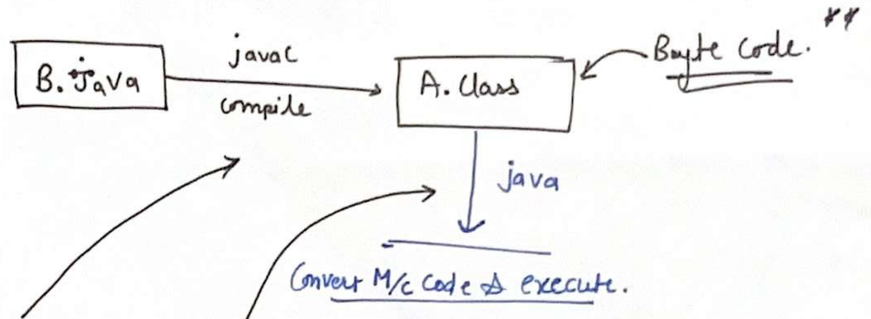
→ To run java programs all the necessory resources like memory, storage etc are managaged by JVM.

B.Java

```
class A{
    Public Static Void Main (String [] args){
        System.out.println(" Hello");
    }
}
```

:~ javac B.java {Compilation}

:~ java A {Execution}



Compiler & Interpreter

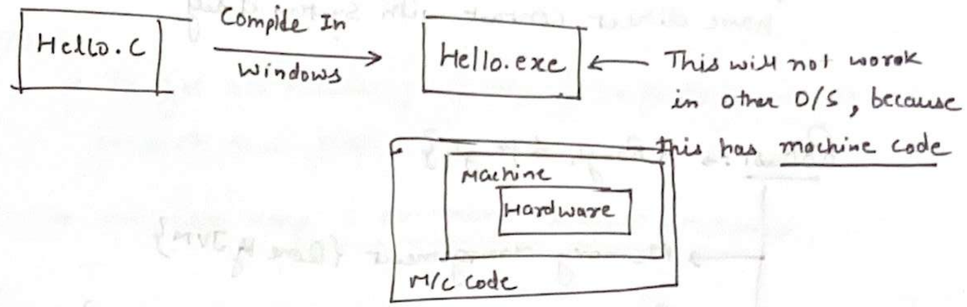
Interpreter slows down the processing due to line wise conversion which is where the new concept JIT {Just-in Time} compiler.

Use of Interpreter or JIT depends upon JVM

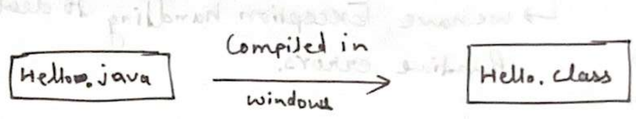
Platform Independency

Hardware Operating System.

In C ⇒

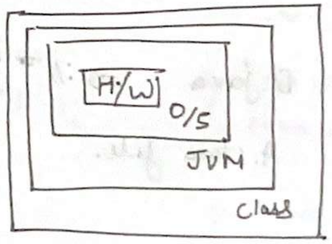


In Java



{ Java application are platform independent, Java itself is Platform dependent }

Eg⇒ If machine is 32bit then it will have different JDK than when machine is 64bit or when O/S is different.



Secure ⇒

We mainly talk about applets here, applets are always run on the client side. {They get transfered to the client from the server when client access the service}. In case of java applets, applets are run by JVM completely. Meaning it will not have direct contact with system itself

Robust :- {Rough & tough}

→ Memory Management {Done by JVM}

→ Runtime error. → {depends upon developer}

↳ we have Exception handling to deal with Runtime errors.

Object Oriented :-

B.java.

```
class A
{
    Public static void main (String [] arg){
        System.out.println (" Hello.");
    }
}
```

⇒ :// javac B.java ⇒ :// java A ⇒ Output ⇒ Hello.

we will get A.class file.

Source file Structure ⇒

Java file can have three type of statement

→ Package Statement

→ Import Statement

→ class / Interface Statement.

① Package → group of related classes.

— Packages are

=:- first non-commented statement of your java file.

=:→ 0 or 1 package statement {can't be more than 1}

② Import →

→ To get all necessory classes for a package we need to import that first.

→ we can have 0 or more .Import statement.

{Even if we donot import any package, Java.lang package is imported automatically.

{In B.Java we had String class which is part of Java.lang which gets automatically imported}

↳ Static Binding {Code binded to the file at compile time}

Java → Dynamic Binding {Code binded to the file at run time.

{Code here refers to imports.}

Test.Java

```
class A{
    P.S.V.M (__)
    {
        S.O.P("A");
    }
}
class B{
    P.S.V.M (__)
    {
        S.O.P("B");
    }
}
class C
{
}
```

:~ javac Test.java

{This will create three .class files even with C not having any main.}

:~ java A

↳ main() of A will be executed.

:~ java B

↳ main() of B

:~ java C

↳ error.

If file has any public class then file name has to be that class name otherwise it can anything.

You cannot more than 1 public classes in a java file

05|2|2024

:~ javac x.java.
↳ A.class.

:~ java A

Output⇒ String arr

x.java

```
Class A
{
 PSVM(int[] args)
  {s.o.P("int args");
  }
 PSVM (int a){
   s.o.P("one int");
 }
 PSVM (String [] args){
  s.o.P("String arr");
 }
}
```

** We can have multiple main(), with condition that type of arguments OR .no. of arguments are different

** By default String array will be called

Comments

There are three ways we can comment in Java.

// → Single line comment.

/** */ ⇒ Multiline comment

/** */ → Java doc comments.

Java doc utility will convert all these comments in java docs & Java help files }.

for executing javadoc utility

:~ javadoc B.java

→ It will create html help files that we have written in our comments (/** */).

** To run javadoc must be public

```
class A
 {
   PSVM( String [] args){
     s.o.P("");
   }
 }
```

Every individual unit of a program is called "Token".

Token → Can be categorised in 5 categories.

- Keywords
- Identifiers
- Literals
- Seperators
- Operators.

① Keywords ⇒

→ There are total 67 Keywords in java.

→ 16 Keywords are context sensitive.
(works as keyword at a specific place)

→ ~~strictfp~~ ~~strictfp~~ is obsolete from jdk-17

→ from jdk9 "_" is a keyword.

→ Keyword which are not having reserve meaning const, goto.

→ Reserve words which are not keywords
{ true, false, null }

06/02/24

② Identifiers → Names (Variable, method, class name etc)

{ → Identifier in java must start with "_, $, or letter" }

→ Subsequent character can be digits.
eg→ _AB, AB_, AB123,

→ No length restriction.

→ Unicode Characters are also allowed.
↳ { have 65536 character ~~spt~~ support, 1 character is represented by 2 bytes. }

③ Literals → {Constants} → eg 5, 10, -5, -10

(0101)$_2$ = (5)$_{10}$ Binary

① Integer Literals → eg→ 5, 10, -5, 10_34_12, 0b0101

0x123, 0123

(123)$_{16}$ = (291)$_{20}$ Hexadecimal

0123 ↳ Octal → (123)$_8$ = (83)$_{10}$

10_34_12 = 103412 { _ in Integer just works as seperators }

{ _123
123_ } Not allowed - can only be used in between numbers

Q1 0123 + 0b10 ⇒ 83 + 2 ⇒ (85)$_{10}$ ✓

Q2 0879 + 56 ⇒ error
↳ Octal can only have (0-7)

② Floating point literals →

eg→ 2.5, 3.7, 2.7f, 2.5e-1, 2.5e4
(2.5×10^{-1}) (2.5×10^4)

double, float

2.5d

{ By default datatype of all floating point literals are double }

Java is a strictly typed language

In C ⇒ float f1 = 2.5 { allowed as C converts 2.5 which is double to float. }

In Java float f1 = 2.5 { will give us error, because 2.5 is a double value }

③ Character literals →

'a' , 'A' , '\101' , '\u0041'

'\101' → Character which is having ASCII code Octal 101 in Octal

'\u0041' → Character which have unicode 0041 or 41 in Hexadecimal.

7/2/24

④ Boolean Literals →

true, false are boolean literals in java

---

④ Seperators → which seperates tokens

{ , ; () {} [] <> }

(Skipped for now Operators next part)

Data Types →

① Primitive Datatypes / Primary datatypes →

ⓐ Integer datatypes

- Byte (1 byte) -128 to 127
- short (2 bytes) -32768 to 32767
- int (4 bytes)
- long (8 bytes)

ⓑ Character datatype →

- char (2 bytes)

ⓒ floating-point datatype →

- float (4 byte)
- double (8 byte)

ⓓ Boolean datatypes →

- boolean {Size not defined in java}

Variables ← {Named memory Location}

There are three types of variables :→

- Local Variables
- Instance Variables
- Class / Static Variables

} defined in class

① Local Variables →

→ which are defined within a method or a block.

eg →

```
Class A {
    PS VM (String [] arg){
        int a, b, c;   //Local Variables
        a=10; b=20;
        c=a+b;
        S.o.P(c);
    }
}
```

Output ⇒ 30

- Local variables must be initialised before being used.
- There is no default values of Local variables in java

```
Class B
{
    PSVM(_){
        int a, b, c, d;
        c=a+b;
        SoP(c);
    }
}
```

Output ⇒ error

- Variable a might not have been initialised
- Variable b might not have been initialised.

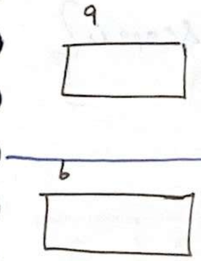
```
Class A
{
    PSVM(_){
        int a, b;
        a=10;
        if(a>5){
            b=20;
        }
        S.o.P(b);
    }
```

error = Variable b might not have been initialised

At compile time 5 is compared with 'a' variable

value a 10 is assigned to 'a' at Runtime. That's why we are getting error at Compile time.

```
Class A
{
    PSVM(_){
        int a, b;
```



8/2/24

Scope & Lifetime of Local Variable

→ Lifetime of Local Variable is within the block/method where it is defined.

Example →

```
Class A
{
    PSVM(_)
    {
        int i=1;
        {
            int i=2;
            SoP(i);
        }
        SoP(i);
    }
}
```

'a' is already defined in Main method.

Output ⇒ error

Scope of local variables cannot overlap

```
Class A{
  PSVMC_){
      for(int i=1 ; i<10;i++){
          S.o.P( i);
      }
      for (int i=1 ; i<10 ; i++){
          S.O.P(i);
      }
  }
```

will work perfectly fine in java as int i. is local variable to the block of loop.

```
Void main()
{
  for (int i=1; i<10 ; i++)
  {
    cout<<i;
  }
  for (int i=1; i<10;i++){
    cout<<i;
  }
}
```

error because in CPP int i is declared before for loop, which will give re-declaration error.

{ No redeclaration in same block or scope is allowed for local variables }

Operators

(1) Arithmetic operators →

① ⇒ +
② −
③ /
④ *
⑤ %

In C/C++ this was only allowed for integer values in but in java it is allowed with floating point values as well.

6.4 % 2.1 ⇒ 0.1

(2) Comparison Operator ⇒

① <, ② <=, ③ >, ④ >=, ⑤ ==, ⑥ !=, ⑦ instanceof.

(3) Boolean Logical Operators ⇒

& = AND
| = OR
! = Not
^ = XOR

① AND (&&)

A	B	Result
T	T	T
T	F	F
F	T	F
F	F	F

```
int a, b, C;
boolean d;
a=10 ; b=15; C=20;
d = a<b && b<c;
SOP(d);
d = a<b && b>C;
SoP (d);
d = a>b && b>C;
SoPC d);
```

② OR (||)

A	B	Result
T	T	T
T	F	T
F	T	T
F	F	F

a = 10, b = 15, c = 20;

boolean d;

d = a < b || b < c; (T)

d = a > b || b < c; (T)

d = a < b || b > c; (T)

d = a > b || b > c; (F)

③ XOR (^)

A	B	^
T	T	F
T	F	T
F	T	T
F	F	F

a = 10, b = 15, c = 20;

d = a < b ^ b < c (F)

d = a < b ^ b > c; (T)

d = a > b ^ b > c (F)

④ Not (!)

A	!
T	F
F	T

④ - Short circuit Operator ⇒

&& = AND

|| = OR

```
y = 10 ; x = 0 ;
if (x != 0 && y/x > 5)
{

}
```

⇒ divide by zero exception

& In case of '&' will evaluate all the condition & then move forward

```
y = 10 ; x = 0 ;
if (x != 0 && y/x > 5)
{

}
```

In case of '&&' as if x != 0 is false it will skip forward as our final result will be false either next is true or false

In C/C++, all boolean logical operators are by default short caircuit.