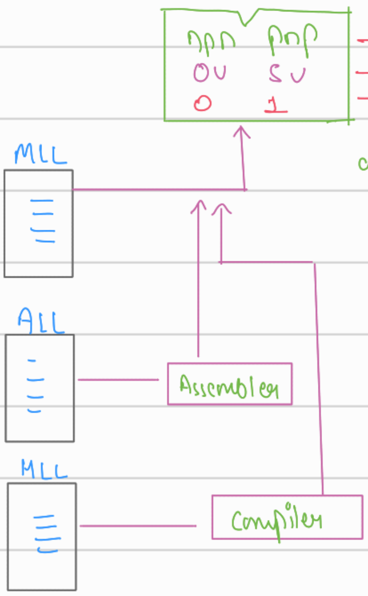
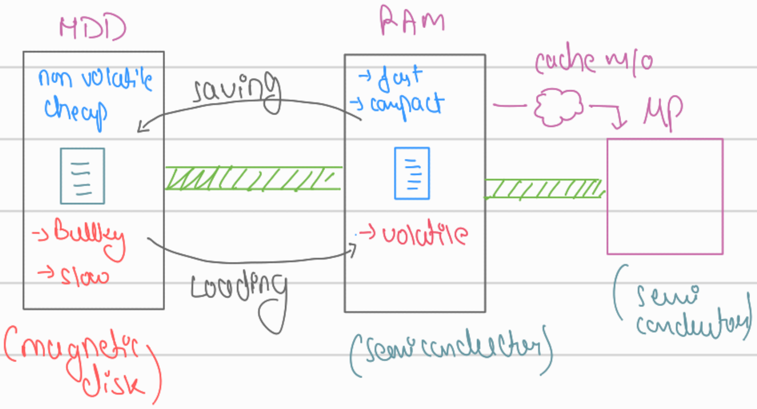
Fundamentals : font size - 14, Ink Draft, codesize : Arial, 10

CPU/mp (semiconductor)
↓
consist of transistor which understands only high voltage & low voltage.



MLL	ALL	HLL
01010	ADD	+
10010	SUB	−
010010	DIV	/
⋮	⋮	print scan

Computer Architecture :



SDD : uses flash m/o (semiconductor)
non-volatile

MLL
.obj vs .exe
incomplete file — complete file
i.e. before importing libraries..

linker	vs	Loader
.obj → .exe (linker)		Slow, which loads .exe file to RAM for execution.
link library files to .obj file & convert it to .exe		

file → HDD
byte → RAM
register → CPU/mp

→ lib files refer to instructions in MLL, these files are linked during execution.

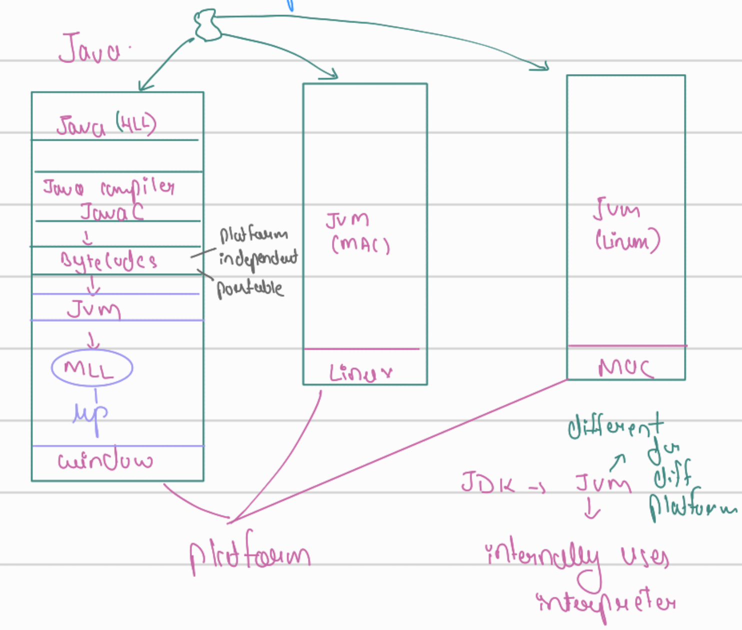
Java : 1995

Java 1.0 : 1996

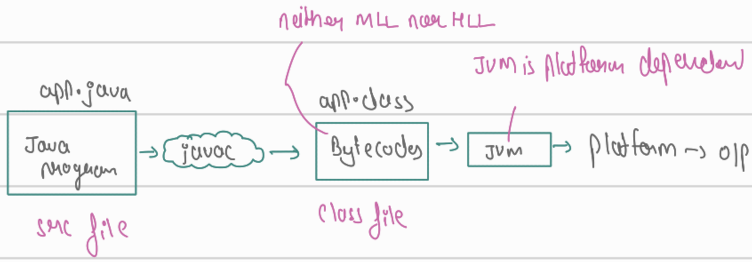
Sun micro system :→ James Gosling
└ opt by ORACLE in 2011

→ obj oriented
→ platform independent or WORA (write once Run anywhere)
→ Internet PL (programming language)

Platform Independent :



→ Because of the architecture of Java it is platform independent.
↘ JVM



Java Hierarchy : LTS version (Long Term support)

- Java 7 } ✗
- Java 8 }
- Java 11 } ✓
- Java 17 } ✓

.c → C compiler → .obj → windows (✓)
Linux (✗)
mac (✗)

.java → javac → .class → window (✓)
Linux (✓)
mac (✓)

JDK = Compiler + JRE
└→ JVM + library tools

Developer (JDK) : write, Test & run code.

Endusers (JRE) : just run the code.

Java versions :

J2SE 1.2 → 1998
- collection framework
- Swing
- JIT Compiler

J2SE 1.3 → 2000
1.4 → 2002
5.0 → 2004
- Annotation
- Autoboxing
- Enumeration
- Enhanced for loop

Java 6 → 2006
Java 7 → 2011
- string in switch
- try with resources
- <> diamond operator

Java SE 8 → 2014
- Lamda expression
- support for JS code
- date & time API
- stream API

Java 9 → 2017 (6 months)
- Modularity
- Reactive streams
- Jshell

Java 10 → 2018 → local variable Type inference

Java 11 → Run src file
└ var for lamda

Java 12 →

Java 13 → - switch expression
- multiple strings

Java 14 → records
→ packaging tool

Java 15 –

Java 16 –

Java 17 → Sealed classes

Java 18 →

Java 19 → virtual threads
→ Vector API

12th oct 2022

C O R E J A V A

- Java is a verbose long. (define structure)

To run java code

Javac <File-name>.java

Java <File_name>

for every class Javac (compiler) creates .class files

Eg. 100 .java 600 classes

then 600 .class --> 1 .jar executable
jar files → package of .class files.

jar → java Archive

Java 9 : jshell → RepL (Read evaluate print loop)

- main

to receive command line argument data

public static void main (String args[])
- public ↓ accessible to All
- static → access w/o object
- void → return type
- main → name known to JVM
- args → name can be anything

OS → JVM → Java Hello First → main method (Hello.java, code) ↓ in command line

args	First	code
	0	1

other Syntax :

static public vm (String [] args)
or
static public vm (String ... args)

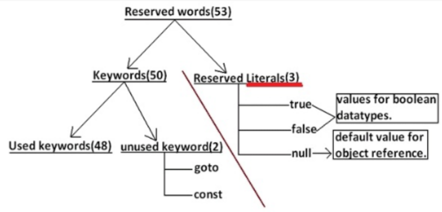
Statically Typed PL (program lang)	Dynamically Typed PL
1. int a = 10;	2. a=10;
Data type is being checked during compile time.	PL where Type of data is not being checked during compile time, it is automatically add during runtime.
E.g. Java, C++, C;	E.g. Python, JS ...

web appl. archive

- .class file : src code converted by compiler to .class file
- .war file : collection of many .class + .html files
- .jar file : collection of .class files

java archive

13th Oct 2022



Reserved words for data types: (8)
1) byte
2) short
3) int
4) long
5) float
6) double
7) char
8) boolean

Reserved words for flow control:(11)
1) if
2) else
3) switch
4) case
5) default
6) for
7) do
8) while
9) break
10) continue
11) return

OOP's in Java

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

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

Exception Handling

Keywords for exception handling:(6)
1) try
2) catch
3) finally
4) throw
5) throws
6) assert(1.4 version)

void -> keyword associated with method
unused keywords: goto,constant

Identifier : It is a name in java program.

- class name, - variable name, method name & label name.

```
Class Test {
    Public static void main (String [] args) {
        S.O.P. ("Hello");
    }
}
```

identifiers = 7

Rules (Syntax for Compiler + JVM) for writing an Identifier.

Rule 1 : The only allowed characters in java identifiers

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

Rule 2 : Identifiers are not allowed to start with digits

int Value01 = 100 ✓
int 1Value = 10 × (invalid)

Rule 3 : Java identifiers are Case Sensitive.

int num = 10;
int Num = 20; } different

Rule 4 : There is no length limit on Java identifiers, but it is a good practise to keep the length of the identifier not more than 15 characters.

Rule 5 : We can use Reserve words as identifier.

e.g. int if = 10; (CE) compiler error

Rule 6 : Predefined class names can be used as identifiers like String, Runnable

E.g. String Runnable = "Sachin";
S.O.P. (Runnable) ; // sachin

int Integer = 10;
SOP(Integer) ; ✓

note : Even though class names can be used as an identifiers, it is not a good practice to keep.

Reserve words : built in words | keywords which has already a predefined meaning to it.

Literal : Any constant value which can be assigned to a variable is called Literal.

int data = 10; → literal
(int = DT; data = identifier / var name)

Naming Convention :

Class / Interface → PascalCase (start with capital)

reserve words → lower case.

method – CamelCase

variables – CamelCase

Data Types : all types are strictly typed / define in java as java is strictly Type / statically typed language.

Compiler Role : compiler will check the value stored can be handled by DT or not.

This checking is done by compiler is called "Typechecking / strictly Type checking".

Primitive DT	Non-primitive
- already defined in Java.	- created by developer.
- Byte	- String
- short	- Arrays
- int	- classes
- long	- Interface

(Byte, short, int, long → whole no.)

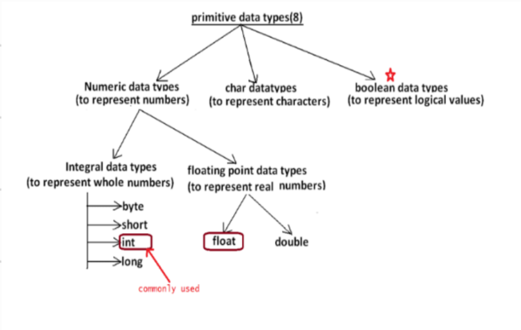
- float
- double

(float, double → Real no.)

Note : sop(Byte.SIZE); // 8 bit

sop (Byte.MIN_VALUE); // -128

sop(Byte.MAX_VALUE); // 127



Byte :

size : 8 bits

min val : -128

max val : 127

E.g. byte marks = 35 // valid

= 135 // CE : possible loss of precision

= -1 // valid

byte a = true // incompatible Type

→ when to use byte?

It is commonly used when we handle the data which is coming from stream, network.

Stream → java.io. package

Short : size : 16 bits (2B)

min val : -32768 max val : +32767

E.g. short data = 137 // valid

Note : This DT is not at all used in java & this data type is used only in old processor like 8086.

int : size = 32 bits (4B)

min val : -2147483648 max : 2147483647

note : The most commonly used DT for storing whole number is "int" only & by default if we specify any literal of number type compiler will try to keep it as "int" only.

long: size = 64 bits (8B)

min : -9223372036854775808

max : 9223372036854775807

Note:

If data goes beyond the range of int, then to keep the data inside long DT we need to explicitly suffix the data with 'L' or 'l' otherwise it would give "CE".

Eg. long data = 9223372036854775807
// CE

long data = 9223372036854775807l
// ✓

Float: size : 4B (32 bits)

min val : 1.4E-5

max val : 3.402823 5E38

Note: By default if you specify any real no./decimal no. compiler will treat it as "Double",

to specify, compiler to treat it as float, we need to suffix 'f' or 'F'

Eg: float a = 10.5 // CE possibly loss of precision

float a = 10.5f;

" " = 25.5F;

float a = 1.05l;

Note: long can be type casted to float instead float is 4B & Long is 8B, coz, float follows IEEE, i.e. internal architecture of float is much bigger.

Double: size : 8B

min value = 4.9E-324

max value : 1.7976931348623157E308

Note: Datatypes are actually represented to compiler & jvm using reserve words.

normally in lower case.

To map primitive DT as object in java from JDK 1.5 concept of "Wrapper class" was introduced

byte →	Byte
short →	Short
int →	Integer
long →	Long
float →	Float
double →	Double
char →	Character

Char: size : 2B as Java follow "Unicode"

ASCII : American standard code for information interchange
└ 128 characters (only some languages)

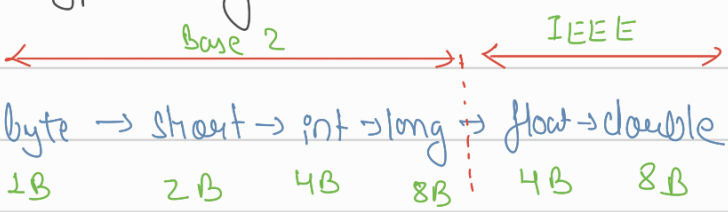
IEEE:
↳ 65536 → characters/symbol = 2^{16}

Eng, hindi, Tamil, Russian ----- 16 bits = 2B

Java → UTF → 2B (Char)

Boolean: size : 1B true & false

Type Casting :



Implicit Typecasting / Numeric Type Promotion :

```
int d = 5;
double a = d; // 5 ✓
```

Explicitly Typecasting :

```
double a = 45.5;
byte b = (byte) a ; // loss of precision
```

note : if operations is performed using some operators in that case the result will be ; casted to "int". automatically

e.g.
```
byte b = 10;
byte c = 20;
byte d = (byte)(b x c);
```
int

Pattern :

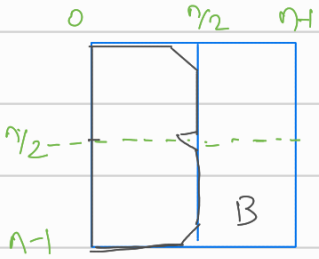
$(i==0 \;\&\&\; j<(n-1)/2)$.

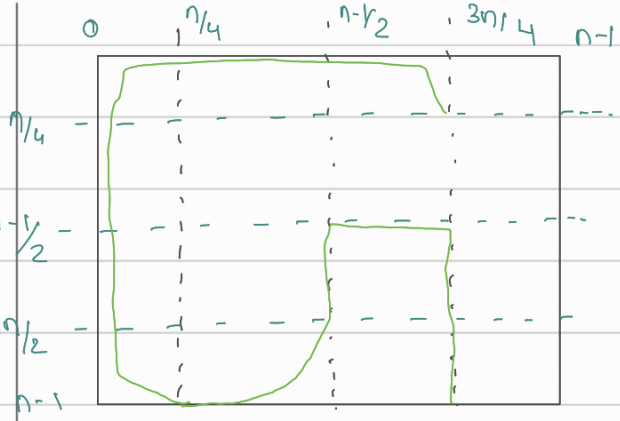
|| $(i==(n-1)/2 \;\&\&\; j<(n-1)/2)$

||

||$(j==n-1/2 \;\&\&\; i>0 \;\&\&)$

$(i!=n-1/2 \;\&\& j<n-1)$



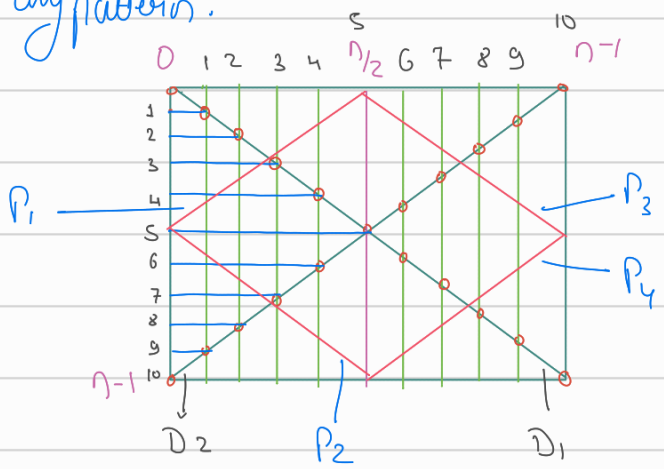



```
for(i:0 to n)
{ for(j=0 to n)
{if((i==0 && j>0 && j<(3n)/4) ||
 (j==0 && i>0 && i<n-1)||
 (j==3n/4) && i>=(n-1)/2) ||
 (i==n-1 && j<(n-1)/2)||
 (j==(n-1)/2 && i>=(n-1)/2))

  Print("*");
else
  Print(" ");
```

Approach : K X Y Z N M Q W V

any pattern :



Juice : very complex problem

D1

i	j
0	0
1	1
2	2
:	:
n-1	n-1

$D_1 : i == j$

D2

i	j	
0	10	=10
1	9	=10
2	8	=10
:	:	
n-1	0	=n-1

$D_2 : i + j = n-1$

```
for(i=0 to n)
{
   for(j=0 to n)
   { if()
      sop("*");
     else()
      sop(" ");
   }
for(int j=0 to n)
   { if();
     else();
   }

for(j=0 to n)
   { if();
     else();
   }
sop();
}
```

P1

i	j
5 +	0
4 +	1
3 +	2
:	:

$P_1 : i+j = \frac{n-1}{2}$

P2

i		j
5	-	0 = 5
6	-	1 = 5
7	-	2 = 5
:		:

$P_2 : i-j = \frac{n-1}{2}$

P3

i		j
0	-	5 = 5
1	-	6 = 5
2	-	7 = 5

$P_3 = j-i == \frac{n-1}{2}$

P4 (n = 11)

i		j
10	+	5 = 15
9	+	6 = 15
8	+	7 = 15
:		:

$P_4 = i+j = n+\frac{(n-1)-1}{2}$