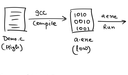
Java fundamentals

Programming → ?

1940 → low level (Binary)
1950 → Symbolic
1960 → High level (English)
1970 → Algol
1972 → BCPL
1972 → C lang (Procedural)
1982 → C++ → (OOPs + Procedural)
1991 → python → (OOPs + procedure)
1995 → Java → (OOPs)

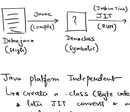
High → low

Compiler	Interpreter
* Scan the whole code.	* Line by line.
* Only Translate	* Translate but also they execute or RUN.



↳ Mingw

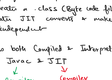
Java → ① Javac → High → Bytecode
② JIT → Bytecode → low
and also execute the code.



Java platform Independent
↳ Creates a .class (Byte code file)
* Intel JIT converts it makes it independent

Java is both Compiled & Interpreted
↳ Javac & JIT
↓ Compiler (High → byte)
↓ Compiler (not as interpreter) = RUNTIME that is why called Just in Time. (Bytecode → low + Run)

JDK vs JRE vs JVM



OOPs

↳ * Reusability
* Security

class Student
↳ 10 lines

~~struct student st = malloc(sizeof(~~
Student s1 = new Student();
class obj constructor

int i;
datatype variable.

Primitive	Non-Primitive
byte	Structures
int	Unions
short	Interfaces
long	Array
float	Classes
double	
char	

Java Packages
↳ Packages are collection of similar classes.

System
↳ Com.Company.project.module. _ java
↳ com.accenture.hdfc.login.....java

OOPs

* Static keyword allows us to access a function directly without creating the class object.

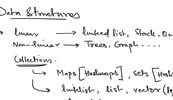
Steps
✓ ① Create class
✓ ② Attributes → Property of class → Access by all obj
↳ eg you name it
✓ ③ Method → constructor with same.
✓ ④ object

Constructor
↓ Create obj
↓ Initializing obj Attributes (Instance Var)

```
public class Student {
    String name;
    int usn;

    Student(String name, int usn){
        this.name = name;
        this.usn = usn;
    }

    public static void main(String[] args){
        Student s1 = new Student("Ishmeet", usn:1213);
    }
```



Data Structures

↳ Linear → Linked list, Stack, Bin...
Non-linear → Trees, Graph....

Collections.
↳ Maps [Hashmap], sets [Hashset]
↳ Linklist, list, vector (legacy)
, Arraylist, Treeset, etc, etc.

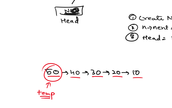
Linked list → Singly, Doubly, circular
, ordered, unordered etc...

Singly

insert → Head, tail, mid
delete → head, tail, mid.
Traverse

Insert-At Head

① Create Node
② n.next = Head
③ Head = n



{loop → while.
System.out(temp.data)
temp = temp.next

while (temp != null)
while (temp.next != null)

Feedback code → 16275