INTRODUCTION TO PROGRAMMING

Date.....
Page.....

* Types of language →

① Procedural →

- Specifies a series of well structured steps and procedures to compose a program
- Contains a systematic order of statements, functions and commands to complete a task.

② Functional →

- Writing a program only in pure functions i.e. never modify variables, but only create new ones as an output.
- Used in situations where we have to perform lots of different operations on the same set of data, like ML.

③ Object Oriented →

- Revolves around object
- code + Data = object
- Developed to make it easier to develop, debug, reuse, and maintain software.

A language can be OO or PO or Function and all three

Static v/s Dynamic languages

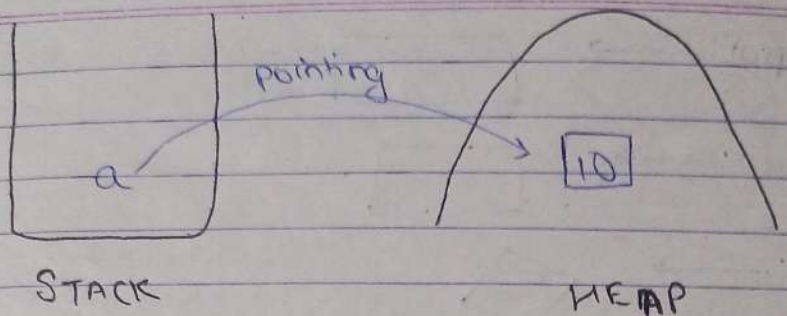
Compilation → JAVA to binary (time)

Static →	Dynamic →
• Performs type checking at compile time	• Perform type checking at run-time.
• Errors will show at compile time Ex→ int a = 10; a = "jaideep" // error	• Error might not show till program is run
• Declare data type before you use it Ex → int a = 10	• No need to declare datatype of variable. Ex→ a = 10 //no error a = "Jaideep"
• More control	• Save time but might give runtime error

* Memory management

{ variables get stored in stack and objects get stored in heap and the variable points to the object }

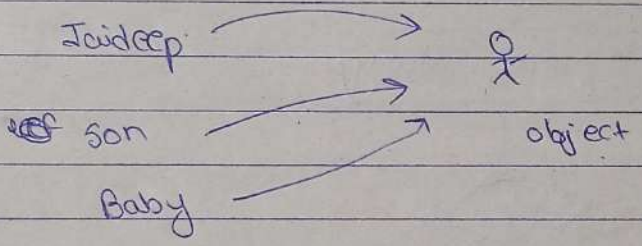
Date



String a = "10"
↓ reference variable
↓ object

{ JAVA has only pass by value method }

Ex →



Primitives are stored in stack memory and Objects are stored in heap memory

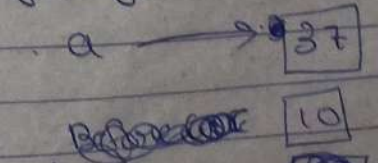
① More than one ref. variables can point to a same object (for non primitive) ② If any one of the reference variable changes the object then the change will be for all ref. variables.

Ex → a = [10, 20, 30]
b = a
a[0] = 99
output (b) → [99, 20, 30]

So, JAVA only has pass by value method

When objects do not have a ref variable then these objects gets destroyed during garbage collection process

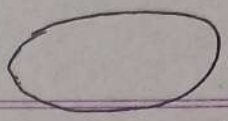
Ex → a = 10
a = 37

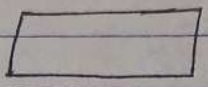


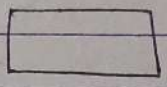
10 will be destroyed from memory

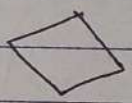
* Flow charts →

Date.....
Page.....

Start / stop ⟶ 

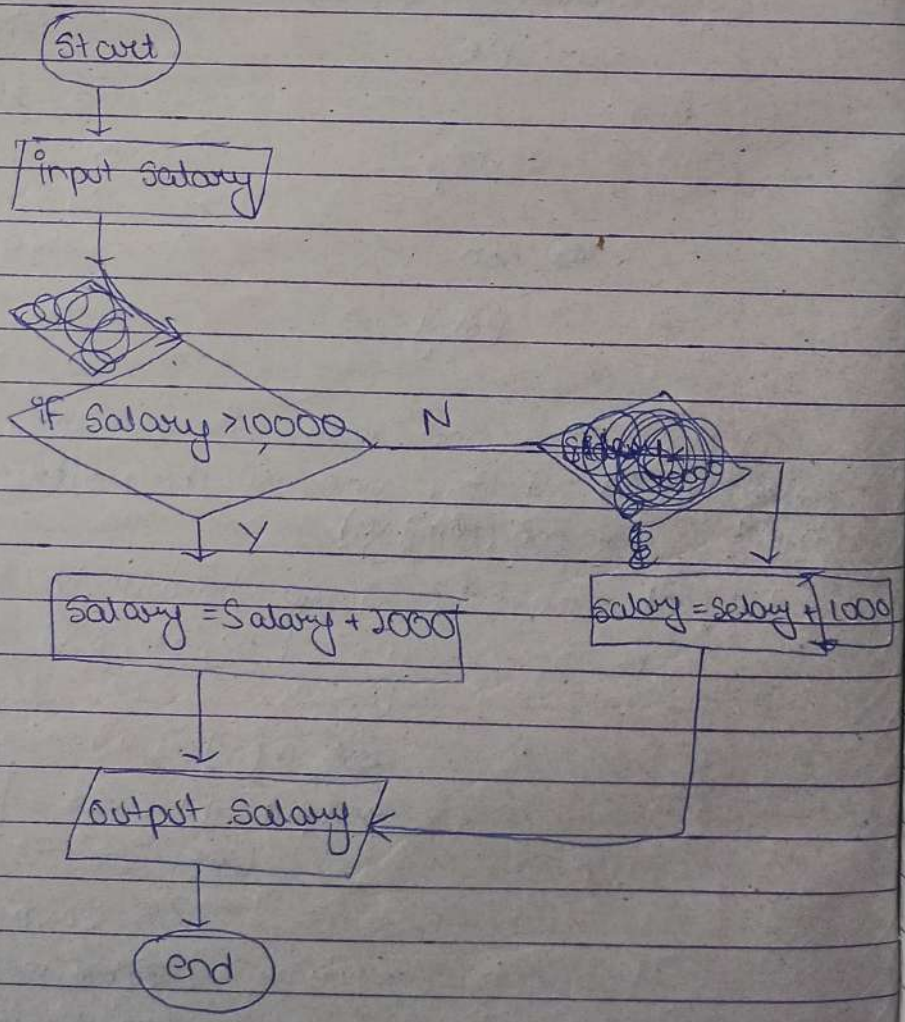
Input / output ⟶ 

Processing ⟶ 

Condition ⟶ 

⟶ Flow of program

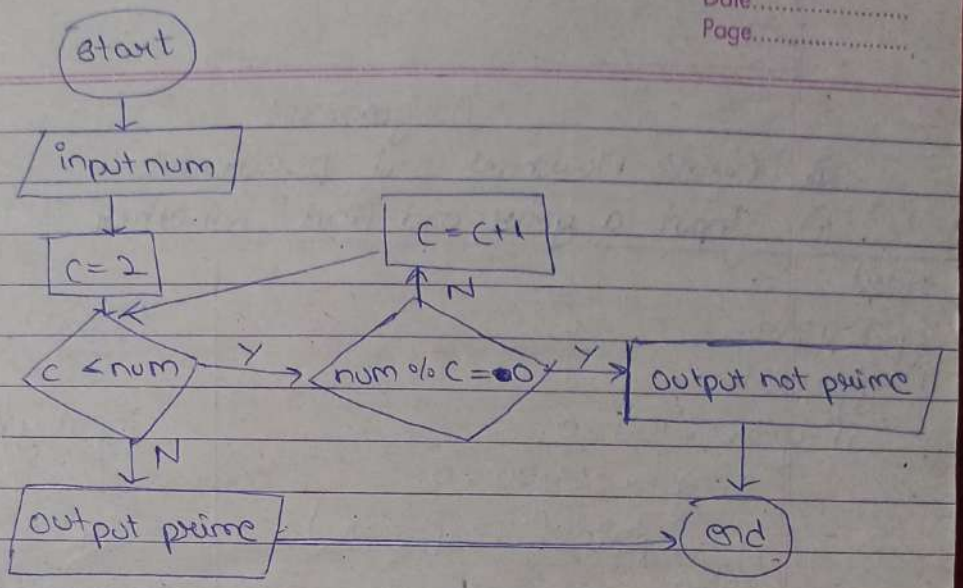
Ques → Take input of a salary. If the salary is greater than 10,000 add bonus 2000, otherwise add bonus as 1000.




```
start
input salary
if salary > 10000:
    salary = salary + 2000
else:
    salary = salary + 1000
output salary
end
```

Ques → Input a number and print whether it is prime or not

Date.....
Page.....




```
start
input num
c = 2
while c < num :
    if num % c == 0 :
        output "not prime"
        end
    c = c+1
end while
output "prime"
end
```

We can optimize this code by just checking till square root of the num

c < sqrt(num)

1 × 36
2 × 18
3 × 12
4 × 9
6 × 6
9 × 4
12 × 3
18 × 2
36 × 1

Repeat

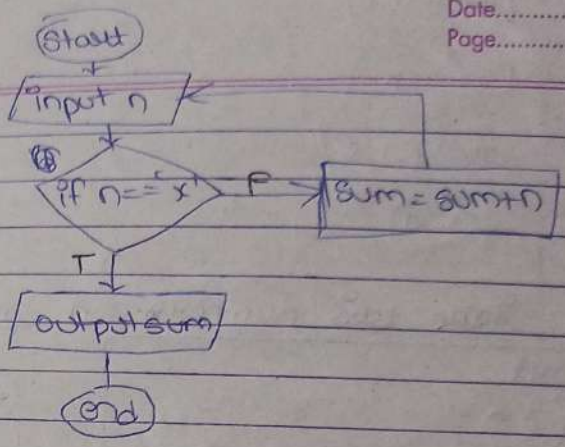
Optimized CODE →

```
start
input n
if n <= 1 :
    print ("neither prime nor composite")
c = 2
while c*c <= n :                    TC = sqrt(n)
    if n % c == 0 :
        output "not prime"
        exit
    c += 1
```

⑥ Taking number as inputs till the user enters 'x' after that print sum of all

Date.....
Page.....

```
start
input n, sum = 0
while n != 'x':
    sum = sum + n
    input n
while end
output sum
end
```



HOW JAVA CODE EXECUTES

.Java file	compiler →	.class file	interpreter →	Machine code
(human readable)	(entire file)	(byte code)	(line by line)	(0 and 1)

This is source code

- this code will not directly run on a system. We need JVM to run is
This the reason why Java is independent

In C/C++ the compiler converts source code directly to .exe file but in Java compiler converts it to a byte code and then JVM converts it into machine code

* Platform independence →

① It means that byte code can run on all Operating systems
② We need to convert source code to machine code so that computer can understand, a compiler helps us in doing this by turning it into executable code.
③ This executable code is a set of instructions for computer.
④ In C/C++ after compiling code we get a .exe file which is platform dependent
⑤ In Java we get byte code, JVM converts this to machine code
⑥ Java is platform-independent but JVM is platform dependent. i.e. JVM for windows, JVM for linux, JVM for mac.

x) after that

Date
Page

= sum+n)

Machine code
(0 and 1)

ndependent
.exe file
JVM converts

ns
t computer
ing it into

er.
is platform

ine code
pendent.

* Architecture of JAVA →

JDK = JRE + Development tools
(Java Development Kit)

JRE = JVM + Library classes
(Java runtime environment)

Java Virtual Machine (JVM)

JIT
(just in time)

* JAVA Development Kit →

① It is a package, which provide environment to develop and run the Java program.

② It includes →

- Development tools → to provide an environment to develop program
- JRE → to execute your program
- a compiler → Javac
- a archiver → jar
- docs generator → javadoc
- interpreter / loader

* Java Runtime environment →

It is an installation package that provides environment to only run the program.

It consists of → ① Deployment technologies
② User interface toolkit
③ Integration libraries
④ Base libraries
⑤ JVM

After we get .class file, the next things happen at runtime :→

1. class loader loads all classes needed to execute the program

1. JVM sends code to "Byte code verifier" to check the format of code

Date.....
Page.....

* Working →

Compile time

•java file

javac compilation

• class file

Runtime

class loader

Byte code verifier

Interpreter

Runtime

Hardware

(How JVM works)

• Class loader

→ Loading

* reads .class file and generate binary data

* an object of this class is created in heap

→ Linking

* JVM verifies the .class file

* allocates memory for class variable and default values

* replace symbolic ref. from the type with direct ref.

Ex → s = a+b

s = 10+20 (values of a, b are placed in eqn

→ Initialization

* all static variables are assigned with their values defined in the code and static block

JVM contains the stack and Heap memory allocations

JVM Execution →

• Interpreter

* line by line execution

* When one method or function is called many times, it will interpret it again and again. Ex → if Hello() is used again and again then it will be interpreted again and again which makes program slower. To solve this JIT is used

- JIT
  - those methods which are repeated, JIT provides fixed machine code of it so that re-interpretation does not take place hence, execution of code is faster.

Date.....

* Garbage collector

* Final Flowchart