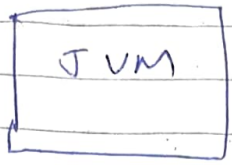
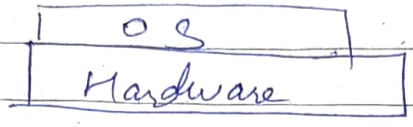
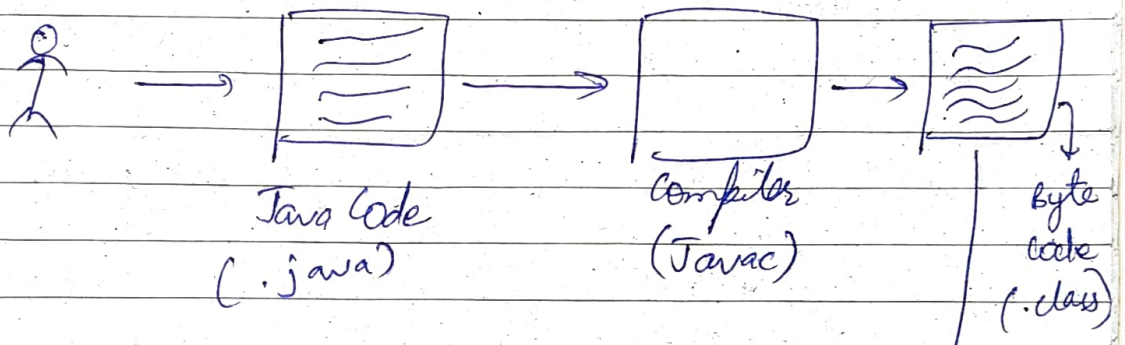




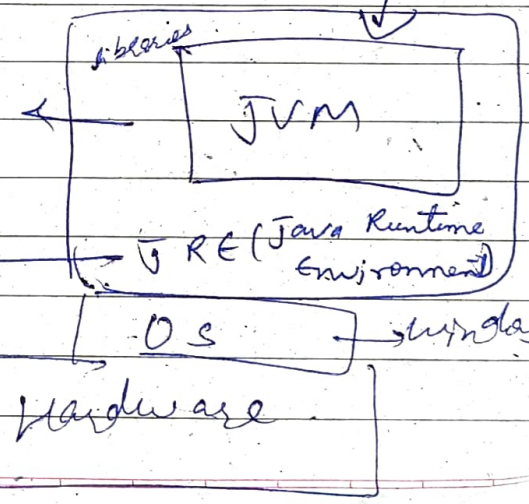
Platform independent means we can run our code on any machine.

JVM is platform dependent like it doesn't run on OS.



public static void main
(String a[])

This contains various libraries



Data Type

Primitive
- Integer → byte, short, int, long
- float → double, float
- Character →
- Boolean

Integer
- int — 4 bytes
- long — 8 bytes
- short — 2 bytes
- byte — 1 byte → -2^7 to $2^7 - 1$

→ −128 to 127

float ↔ 4 bytes
double — 8 bytes

float has limited precision level

By default Java use double because it has higher precision level.

```
double b = 6.5;
float c = 6.5f;
```

Character :- 2 Bytes

```
char c = 'k';
```

Boolean – True or False.

```
boolean b= true;
```

we donot use 0 & 1 in boolean in java.

Logical Operators

And → &

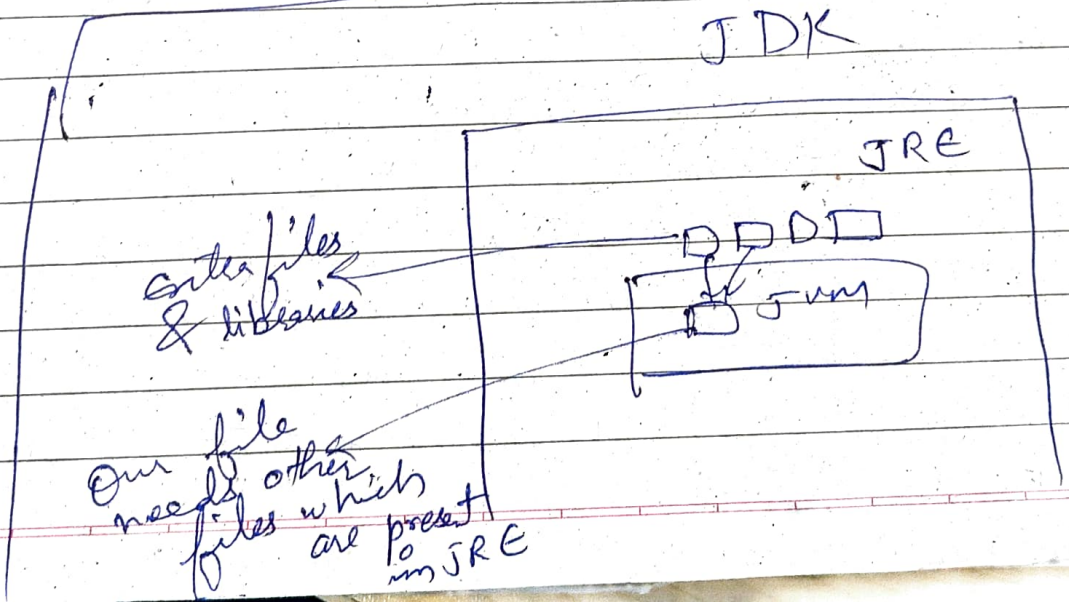
OR → |

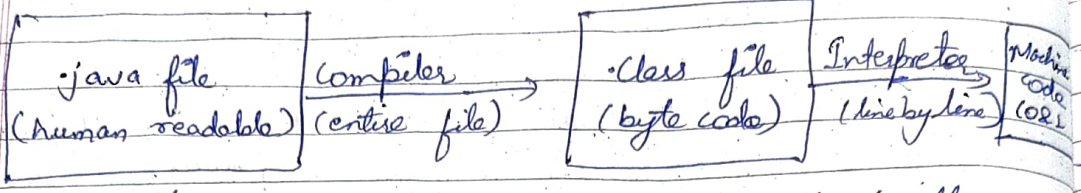
NOT → !

JDK → Java Development Kit

JRE → Java Runtime Environment

JVM → Java Virtual Machine





This is the source code.

- this code will not directly run on a system
- we need JVM to run this
- Reason why JAVA is platform independent

Platform Independence

→ It means that byte code can run on all operating systems.
→ We need to convert source code to machine code so computer can understand.
→ Compiler helps in doing this by turning it into executable code.
→ After compiling C/C++ code we get .exe file which is platform dependent.
→ In Java we get Bytecode, JVM converts this to machine code.
→ Java is platform independent but JVM is platform dependent.

JDK = JRE + Development Tools

JRE = JVM + Library Classes
(Java Runtime Environment)

Java Virtual Machine (JVM)

JIT
(Just in time)

JDK

→ Provides environment to develop and run the Java Program.

→ It is a package that includes:

1. development tools – to provide an environment to develop your program
2. JRE – to execute your program
3. a compiler – javac
4. archiver – jar
5. docs generator – javadoc
6. interpreter / loader.

JRE (Java Runtime Environment)

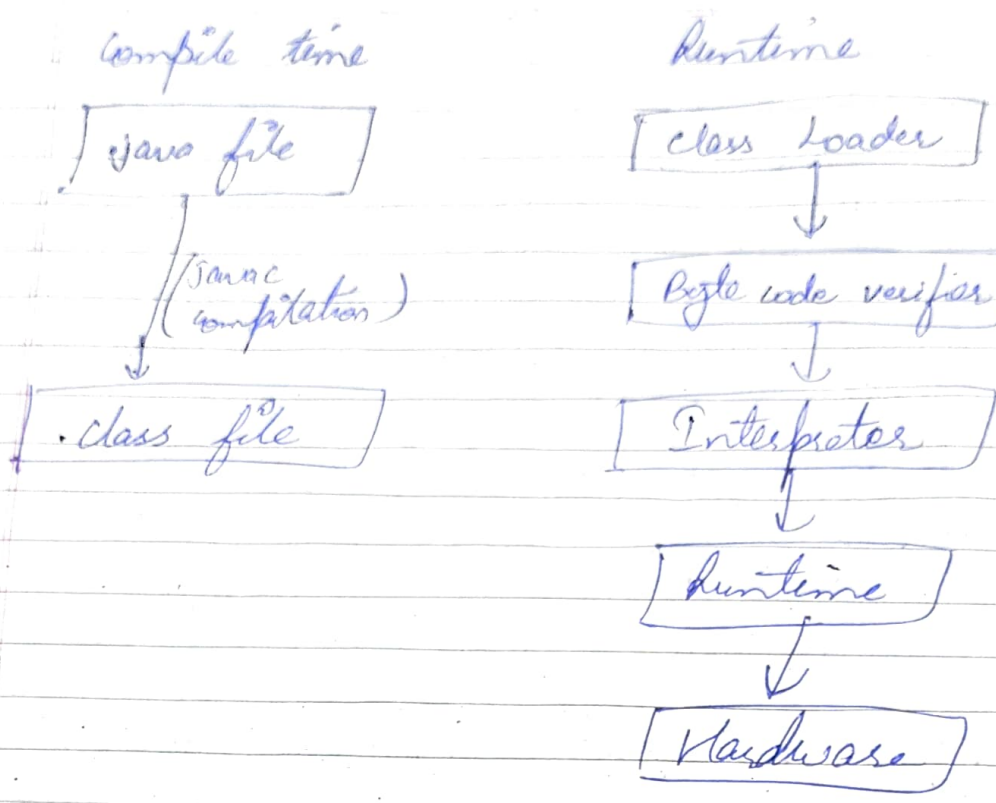
→ It is an installation package that provides environment to only run the program

– It consists of:

1. Deployment Technologies
2. user interface toolkits
3. Integration libraries
4. Base libraries
5. JVM

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

1. class loader loads all classes needed to execute the program.
2. JVM sends code to Byte code verifier to check the format of code.



JVM contains the stack & heap memory allocations.

(How JVM works) Class loader

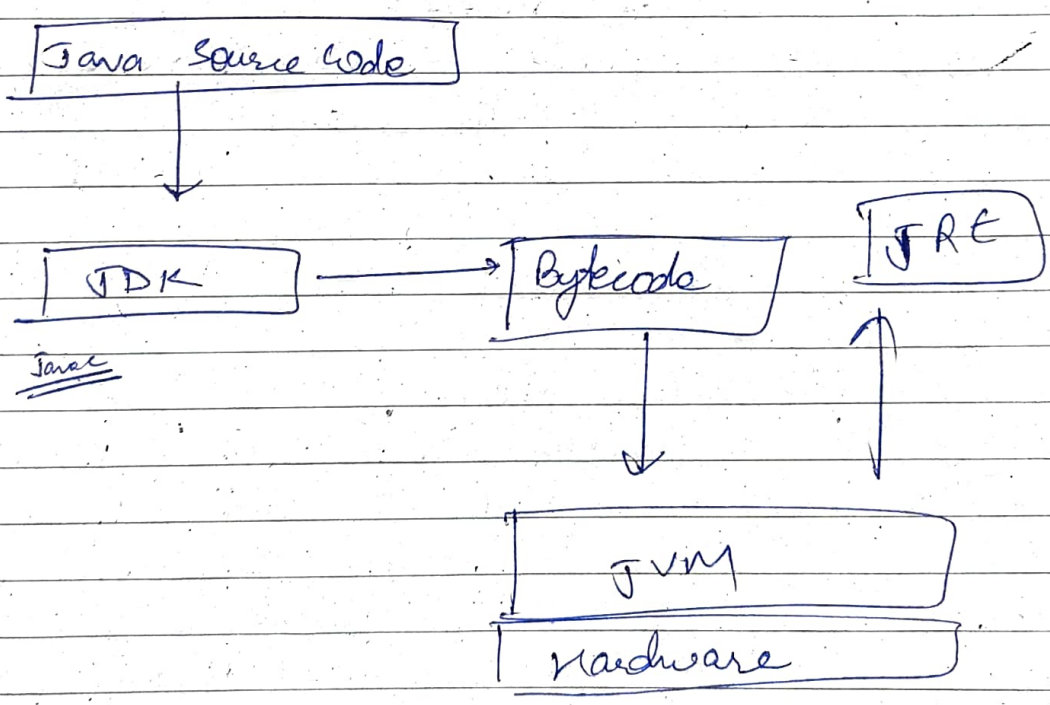
- loading
  - reads .class file & generate binary data
  - an object of this class is created on heap
- linking
  - JVM verifies the .class file
  - allocates memory for class variables & default values.
  - replace symbolic references from the type with direct references.
- Initialization
  - All static variables are assigned with their values defined in the code & static block.

JVM Execution:

Interpreter:
- Line by line execution
- When one method is called many times it will interpret again & again.

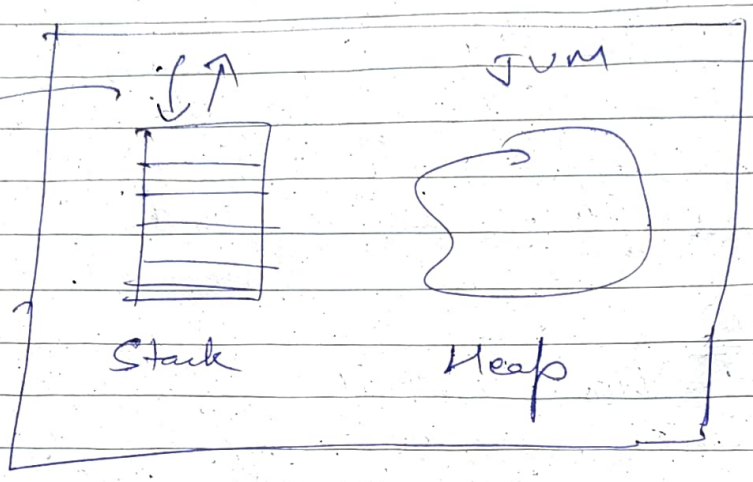
JIT
- those methods that are repeated, JIT provides direct machine code so, re-interpretation is not required
- makes execution faster.

Garbage Collector.



Stack & Heap in Java

First in Last out



→ Every Method has its own stack.

```
class Calculator {
    int num = 5;
    public int add(int n1, int n2)
    {
        return n1 + n2;
    }
}

public class Demo {
    public static void main(String a[])
    {
        int data = 10;
        Calculator obj = new Calculator();
        Calculator obj1 = new Calculator();
        int r1 = obj.add(3, 4);
        //System.out.println(r1);
        obj.num = 8;
        System.out.println(obj.num);
        System.out.println(obj1.num);
    }
}
```

This will only change the value of num for object obj

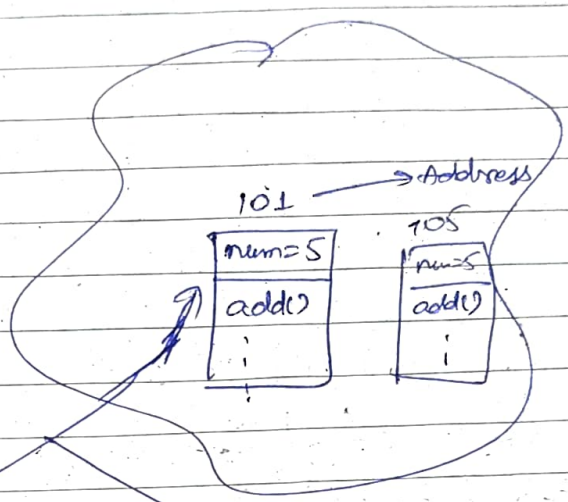
String

add

n_2	4
n_1	3

main

obj	101
r_1	7
data	10

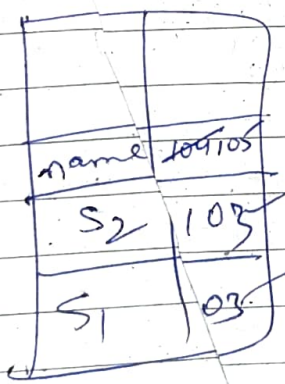


Link between Stack & Heap.

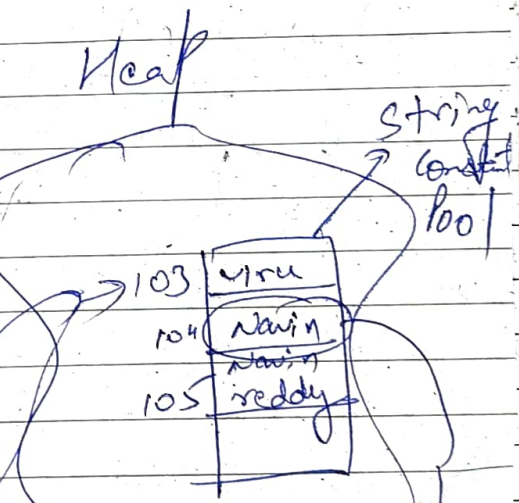
String

String S1 = "Viru";
String S2 = "Viru";
name = "Navin";
name = ~~"Navin"~~ name + "reddy"

This will change the address of name

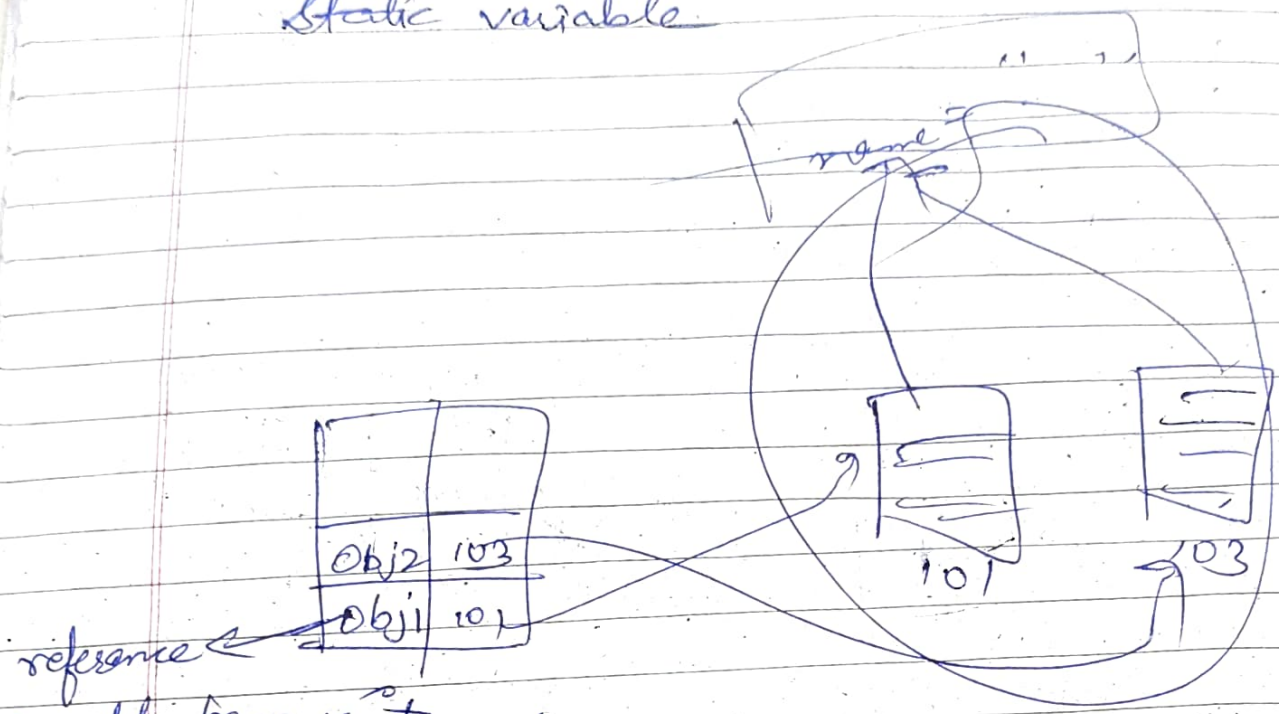


Stack



Since address is changed for name at 104 the value is removed in garbage collection.

Static variable



reference variable because it is refering to the object in heap memory.

Types of Interface

Normal

Interface having two or more methods.

Functional / SAM (Single Abstract Method)

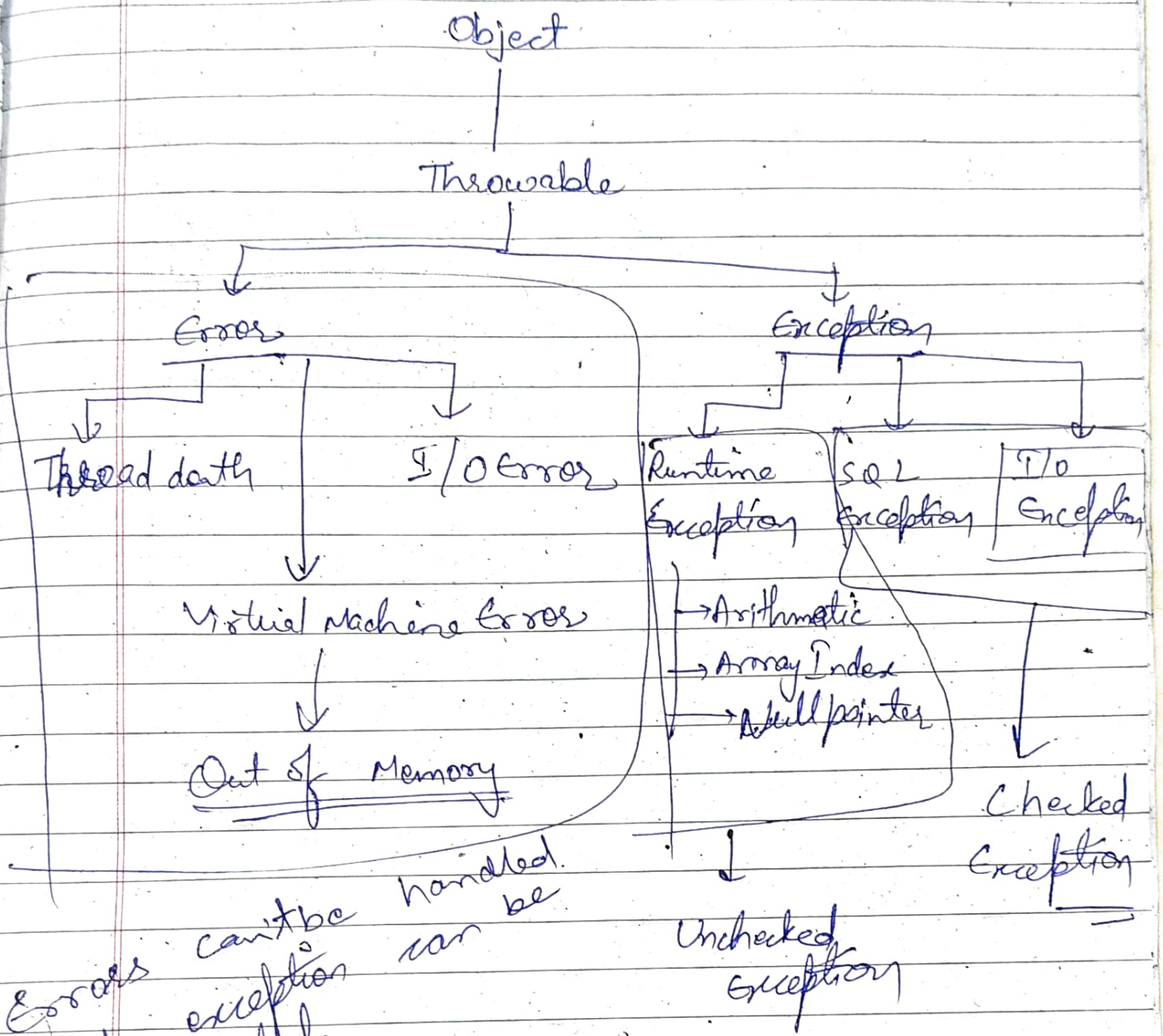
Interface having only one method

Marker

Interface having zero methods.

- Used to update some values in compiler

Exception Hierarchy in Java.



Errors can't be handled.

only exception can be handled.

Thread States in Java