⇒ Object Oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming.

⇒ The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function

* A class is a blue print that defines properties and behaviours, while an object is an instance of a class representing real-world entities.

JAVA

* Java is a high-level, object-oriented programming language used to build applications across platforms from web and mobile apps to enterprise software.
* It is known for its Write Once, Run Anywhere capability
* It means - code written in Java can run on any device that supports the Java Virtual Machine (JVM)

Uses :-

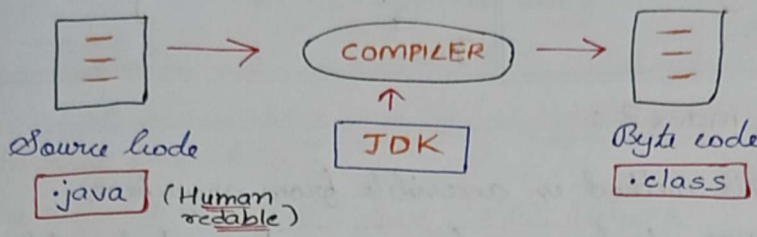
* Used to develop mobile apps, desktop apps, web apps, web servers, games etc.
* Java's syntax is similar to C / C++
* Popular platforms like Java used in LinkedIn, Amazon and Netflix for their back-end architecture.

* Java was invented by James Gosling, created in 1995.
* Currently Oracle owns it.
* JDK 23 is the latest version of Java

RULES :-

* The file name must match with public class name.
* Java is case Sensitive
* If a file has no "public class", the file name can be anything but it should be matched with primary class name

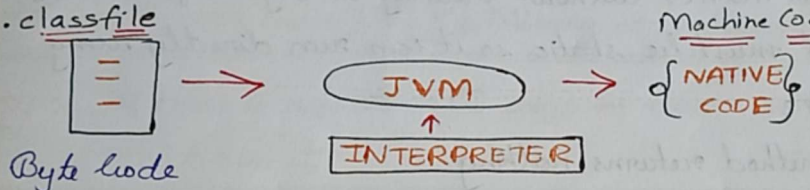
COMPILATION :-



JDK :- Java development Kit

* It is a software development environment used for creating & building Java applications
* Without JDK / JRE we cannot run java code.
* JRE means Java Runtime Environment. It provides the necessary runtime libraries and the JVM to Execute code.
* JRE is a crucial component for running Java applications and is part of the broader Java development Kit (JDK).

EXECUTION :-



* JVM stands for Java Virtual Machine.
* It is a virtual machine that enables the execution of byte code
* It is important that it provides a platform - independent environment meaning you can write Java code once and run it on any device that has a JVM

JAVA :- Compiler & Interpreter

SAMPLE CODE :-

index2.java

```
public class index2 {
    public static void main (String[] args) {
        System.out.println ("Hello World");
    }
}
```

void main() { } Function	class main05 { void main() { } } Class

public class index2 :-

public :- This method is accessible from anywhere

* When the JVM starts your program, it must be able to access and access the main() method by making it public

class :- The class which we will use, name groups of properties

index2 :- The file name must match with public class name. So file name = index2.java, class name = index2

public static void main :-

public :- This method is accessible from anywhere

static :- Belongs to class not to an object

* The JVM calls main() without creating an object of the class.
* So the method must be static so it can run directly using the class name

void :- The method returns nothing

* The main() method just starts the program, it doesnot return a value to the JVM.

* public → Allows JVM to access main() from anywhere.
* static → Allows JVM to run main() without creating an object
* void → No return value

What does public mean before a class ?

* Public means the class is accessible from any other class or package

*** Only one public class is allowed for .java file, and that file must match the class name.

```
public class A {
}
public class B {
}
```

IT WILL NOT WORK

How to solve this problem?

To solve we will use class Helper

```
class Helper {
}
```

public class ⇒ Accessible from anywhere

class ⇒ Accessible only in same package

public static void main (String[] args) { ✓

public void main (String[] args) { ✗ If no static, JVM cannot call it directly

static void main (String[] args) { ✗ If no public, JVM cannot access it

public static void main () ✗ Missing args parameters

NOTE :- If there is no String[] args in main, then JVM won't run it

OUTPUT :- (System)

* A built-in class in the Java 'java.lang' package
* It provides access to system-level resources
* It contains useful fields and methods like System.in, System.out, System.err

```
System.out.print("Hello World"); ✓
System.out.println("Hello World"); ✓
System.out.print('Hello World'); ✗
System.out.println('Hello World'); ✗
```

Example code :-

```
public class index3 {
    public static void main (String[] args) {
        System.out.print("Hello");
        System.out.print("World!");
        // Output : HelloWorld!
    }
}
```

```
System.out.println(358);
```
Output :- 358

```
System.out.println("Hello");
System.out.println("World!");
```
// Output : Hello
World!

C → Compiler
C++ → Compiler
JS → Interpreter

PYTHON → Compiler & Interpreter
JAVA → Compiler & Interpreter
PHP → Interpreter

PLATFORM INDEPENDENCE

Why Java is Platform - Independent ?

* Java code is compiled into byte code by Java Compiler
* This bytecode is the same across all platforms and can be executed on any system that has a JVM
* So, Java's "Write once, run anywhere" capability comes from the fact that bytecode can be run on any JVM

Why JVM is Platform - Independent ?

* The Java Virtual Machine (JVM) is a program that interprets or compiles Java bytecode to machine code specific to the host operating system and hardware.
* Each OS (Windows, Linux, macOS, etc) has its own implementation of the JVM.
* For Example, the JVM for Windows is different from the one for Linux because it translates bytecode into OS-specific machine instructions.

Component	Platform Independent	Why ?
JAVA	✓ YES	Same bytecode runs anywhere with a JVM.
JVM	✗ NO	Needs to be implemented for each OS / platform

* After compiling C/C++ code we get, .exe file which is platform dependent
* But in Java we get byte code, JVM converts this into machine code.
* Hence, the Java is platform independent and the JVM is dependent.

Converting .java to .class file / byte code :-

* We have to install JDK (Java Development Kit) which will have JAVAC compiler with it
* It will compile source code to byte code
* Use command ['javac Main.java] in your terminal
* This will create a Main.class file in file explorer & can be accessed in the form of byte code

In Terminal :-

javac Main.java → First compile it
java Main 30 "vijju" → command line argument

Output :- vijju [as arrays start from [0] index]

System.out.println(args[1]);

System.out.println(args[0]);

Output :- 30

* Here, the command line arguments were given in the form of strings but stored in the form of arrays starting from [0] to [n-1]

* If we want to change the location of byte code :-

javac -d location Main.java
javac -d. Main.java → In current directory
javac -d.. Main.java → In previous directory

Commands :-

* Where javac

Output :- C:\Program Files\Eclipse Adoptium\jdk-21.0.7.6-hotspot\bin\javac.exe

* ls location

Output :- We will get all the list of files in this particular location.

* ls location | grep javac

Output :- |grep → is filter, which filters all the files present in that particular location and finds out the given file. (javac) Output

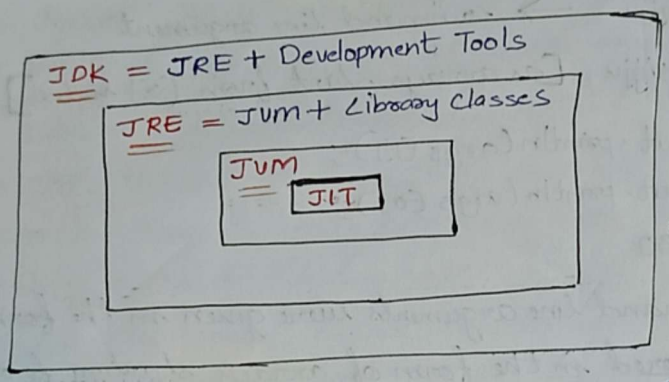
* open location

  Output :- Opens the location

* echo $PATH

  Output :- Gives all the paths in the computer / laptop and when we use (where) where command it will only give the path from the above output

JAVA ARCHITECTURE



CODE EDITOR

* VS CODE
* INTELLIJ
* ECLIPSE

JDK :- Java Development Kit

JRE :- Java Runtime Environment

JVM :- Java Virtual Machine

JIT :- Just-in-time

JIT :- Those methods that are repeated, JIT provides direct machine code so re-interpretation is not required.

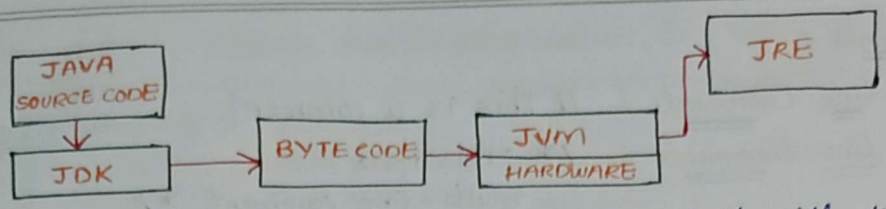
* It makes execution faster, garbage collector

JDK :- Provides environment to develop and run the Java program.

JRE :- It is an installation package that provides environment to only run the program

JVM :- Line by Line Execution.

* When one method is called many times, it will interpret again and again.
* Whatever JVM require JRE will provide
* JRE contains extra files, libraries, which is provided to the JVM to work



*** JIT will have the machine code and it will directly provide the machine code to the interpreter whenever it requires the same block of code to interpret.

* It makes Execution faster

CODE SNIPETS :- (OUTPUT, BASIC SYNTAX)

```
System.out.println(3);
```
Output :- 3

```
System.out.println(3+3);
```
Output :- 6

```
System.out.println(2*5);
```
Output :- 10

```
System.out.println(true);
```
Output :- true

```
System.out.println(System.out.println("Koushik"));
```
Output :- Error

```
System.out.print(System.out.print("Koushik"));
```
Output :- Error

```
public class Koushik {
    public static void main(String[] args){
        System.out.println("Koushik");
    }
    public static void main(String args){
        System.out.println("Teja");
    }
}
```
Output :- Koushik

```
public static void main(String[] args){
    // \u000d System.out.println("hello");
}
```
Output :- hello

```
public static void main(String[] args){
    System.out.println(2+3*4);
}
```
Output :- 14

```
public static void main(String[] args){
    System.out.println("Line1\nLine2\tTabbed");
}
```
Output :- Line1
Line2 Tabbed

```
System.out.println("Number:"+5+10);
```
Output :- Number:510

```
System.out.println("She said, \"Hello!\"");
```
Output :- She said, "Hello!"

```
System.out.println("This is a backslash: \\");
```
Output :- This is a backslash: \

Comments :-

Single-line Comments :- // This is a comment

Multi-line Comments :- /* This is a
multi-line comment */

VARIABLES

* Variables are containers for storing data values.
* In Java, there are different types of variables, for example
  String, int, float, char, boolean
* The equal sign is used to assign values to the variable

Example :-
```
int myNum = 15;
System.out.println(myNum);
```

$, _, letter — starting
letter, digit, _, $ — word

DATATYPE

PRIMITIVE

1. INT
2. FLOAT
3. CHAR
4. DOUBLE
5. SHORT INT
6. LONG INT
7. BOOLEAN
8. BYTE

NON-PRIMITIVE

1. STRING
2. ARRAY
3. CLASS
4. OBJECT
5. INTERFACE

INTEGER TYPES :- byte, short, int, long

FLOATING TYPES :- float, double

FLOAT :- Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits 3.14f (f is necessary to write)

DOUBLE :- Stores fractional numbers. Sufficient for storing 15 to 16 decimal digits

CHAR :- Stores a single character / letter or ASCII values

BYTE :- Stores whole numbers from -128 to 127

SHORT :- Stores whole numbers from -32,768 to 32,767

INT :- Stores whole numbers from -2,147,483,648 to 2,147,483,647

LONG :- Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

100456789L

BOOLEAN :- Stores true or false values

ASCII VALUES :- A = 65 , a = 97
B = 66 , b = 98
C = 67 , c = 99

* Primitive types in Java are predefined and built into the language, while non-primitive are created by the programmer.
* Non-primitive types can be used to call methods to perform certain operations; whereas primitive types cannot
* Primitive types always holds a value, whereas non-primitive types can be null.

PRIMITIVE DATA TYPE MEMORY SIZES :-

DATA TYPE	64 BIT		32 BIT		16 BIT		8 BIT	
	BITS	BYTES	BITS	BYTES	BITS	BYTES	BITS	BYTES
BYTE	8	1	8	1	8	1	8	1
SHORT	16	2	16	2	16	2	16	2
INT	32	4	32	4	32	4	32	4
LONG	64	8	64	8	64	8	64	8
FLOAT	32	4	32	4	32	4	32	4
DOUBLE	64	8	64	8	64	8	64	8
CHAR	16	2	16	2	16	2	16	2

*** Java primitive data type sizes remain fixed, they do not depend on the underlying architecture (8, 16, 32, 64 bit) This is one of the Java's key feature : Platform Independence

* Boolean is JVM dependent
* Boolean logically requires only one bit

DATATYPE	SUFFIX
FLOAT	f or F
DOUBLE	d or D
LONG	l or L

*** Why do we add f & L at last ?
* By default the decimal values are stored in double
* To make them store in float we add f at the last
* Long provides us to store larger values, by default the values with no decimal point are actually Integers.
* So, to store them as long data type - we add L at the last

For Output → We have System class

For Input → We have Scanner class

Scanner → It can pass primitive types & strings using regular expressions.

```
* import java.util.Scanner
public class Main {
        public static void main(String[] args){
            scanner input = new Scanner(System.in);
        }
}
```

`System.in` → standard input stream, which is our keyword

`Scanner input` → new Scanner(file), If I want some file input / read the file

```
scanner input = new Scanner(System.in);
System.out.println(input.nextInt());
```

Output :- Asks for a value, If I give 400 & Enter it will print 400 in next line.

`System.out.println();` → This is actually inherited

* This already exists in an inbuilt package of json
* This is used to display things, like output

```
package com.apnacollege;
import java.util.*;
public class Main {
        public static void main(String[] args){
            Scanner sc = new Scanner(System.in);
            int a = sc.nextInt();
            int b = sc.nextInt();
            int sum = a+b;
            System.out.println(sum);
        }
}
// Sum of Two Numbers
// 1+2 = 3
```

```
import java.util.scanner;
public class Inputs {
        public static void main (String[] args){
        Scanner input = new Scanner (System.in);
        System.out.print (" Please enter some input");
        int rollno = input.nextInt();
        System.out.println ("Your roll number is" + rollno);
    }
}
// Ask for input of roll no & print " Your roll no is ___"
// Your roll number is 2406090
```

```
public static void main (String[] args){
        Scanner sc = new Scanner (System.in);
        String name = sc.next();
        System.out.println (name);
    }
}
```

* The next() method reads the next complete token from the input. A token is completed sequence of characters seperated by white space.
* When you use next(), it reads input until it encounters a whitespace character (such as space, tab, or newline).
* This means it reads one word at a time.

```
String name = sc.nextLine();
```

* The nextLine(); method of the Scanner class is used to read an entire line of input, including spaces, until a line seperator is encountered

```
String name = sc.nextInt();
String name = sc.nextFloat();
```

```
import java.util.*
```

* This line imports the **`java.util`** package, which contains utility classes such as `Scanner`
* The '*' means import **all classes** in the `java.util` package.
* Required for using the `Scanner` class in this program

PACKAGE :- It is the folder in which our java file will lie

* Some times we need to provide some rules that, this particular file should be accessed by only certain type of files. So in this case, packages are very helpful.

```
package koushik;
```

This is folder name in which my java file lies

To create package :-

```
package com.koushik;
```

* Then koushik is the folder in "com" named folder.
* We can create a new package by right clicking on src & new & package.

IF / ELSE :-

```
import java.util.*;
public class Conditions {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int age = sc.nextInt();
        if (age>18) {
            System.out.println("Adult");
        }else{
            System.out.println("Not Adult");
        }
    }
}
```

```
// To check the entered age is Adult or not
import java.util.*;
public class Conditions {
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        if (x%2==0) {
            System.out.println("Even");
        }else{
            System.out.println("odd");
        }
    }
}
// To check if a Number is even or odd
```