JAVA [Telusko]

Lecture 1:

Introduction

- Java is developed by Sun Microsystems in 1995

- Platform independent

- Statical typed language

- Java is WORA – write once run anywhere

Input in Java

Scanner :

- Scanner simplifies console input for common use cases, providing easy-to-use methods for various data types.

BufferedReader :

- Offers greater control and efficiency.

- suitable for complex input scenarios.

Lecture 2:

Install

- JDK (Java Development Kit)

- Vs code Or IntelliJ Idea

Check if Installed

Terminal:

java –version: if java installed or not

javac –version: java compiler

Lecture 3:

JShell

- JShell is a Read-Evaluate-Print Loop (REPL) tool introduced in Java 9.

- It allows developers to quickly test Java code snippets, expressions, and declarations without the need to write a full Java program or create separate source files

REPL : It is a type of interactive programming environment that allows you to execute code line by line and see the results immediately.

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anura>jshell
|  Welcome to JShell -- Version 22.0.1
|  For an introduction type: /help intro

jshell> 6+5
$1 ==> 11

jshell> 6%6
$2 ==> 0

jshell> System.out.println("hello")
hello

jshell>
```

This is the use of Jshell in CMD

Compile

In terminal: javac file_name

e.g.: javac .\Telusko\L3_basics.java

Lecture 4:

JVM

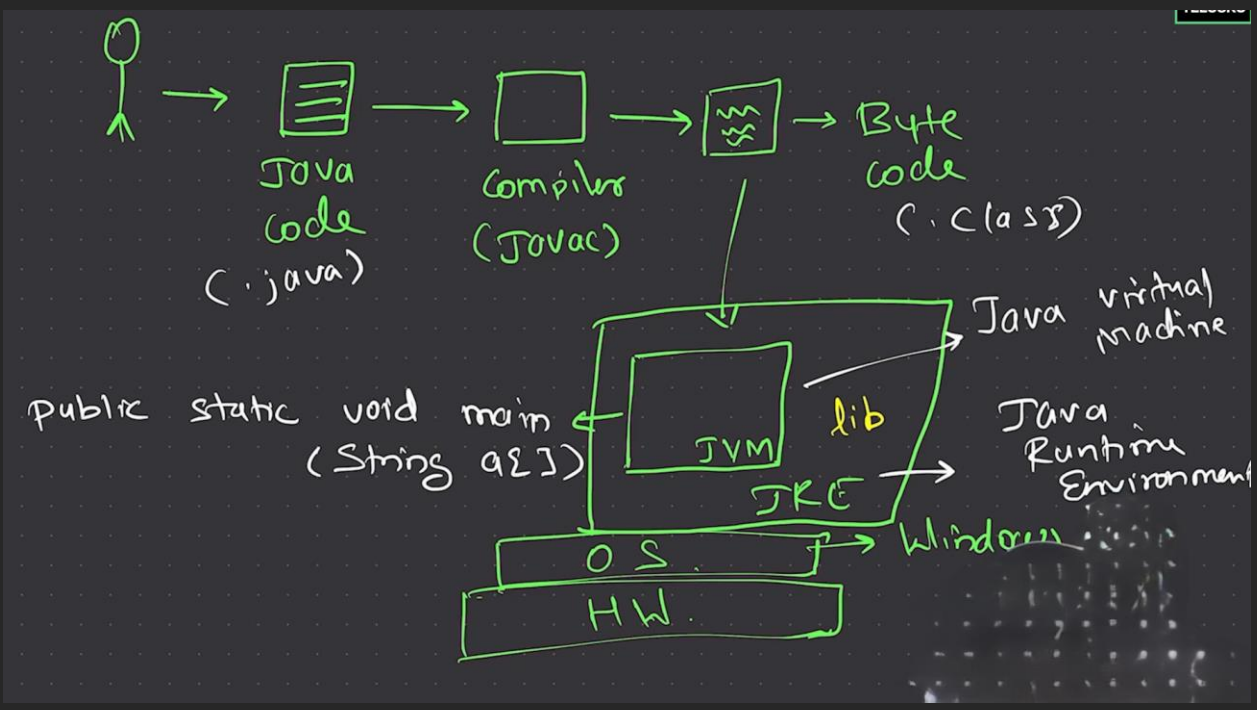
- Java Virtual Machine.

- In your machine JVM is installed irrespective of which OS you use.

- Without JVM Java applications can't run.

- JVM make Java platform independent, but IOS does not have JVM.

How Java Code Executed ?



- JVM always starts execution from a specified first file which has main method.

- JDK --> JRE --->JVM

Extensions:

Java code - .java

Byte code - .class

Lecture 5:

Variable

- By programming We want to solve the Real-world problem using virtual world solution. E.g. for shopping we have different applications, Banking s/w

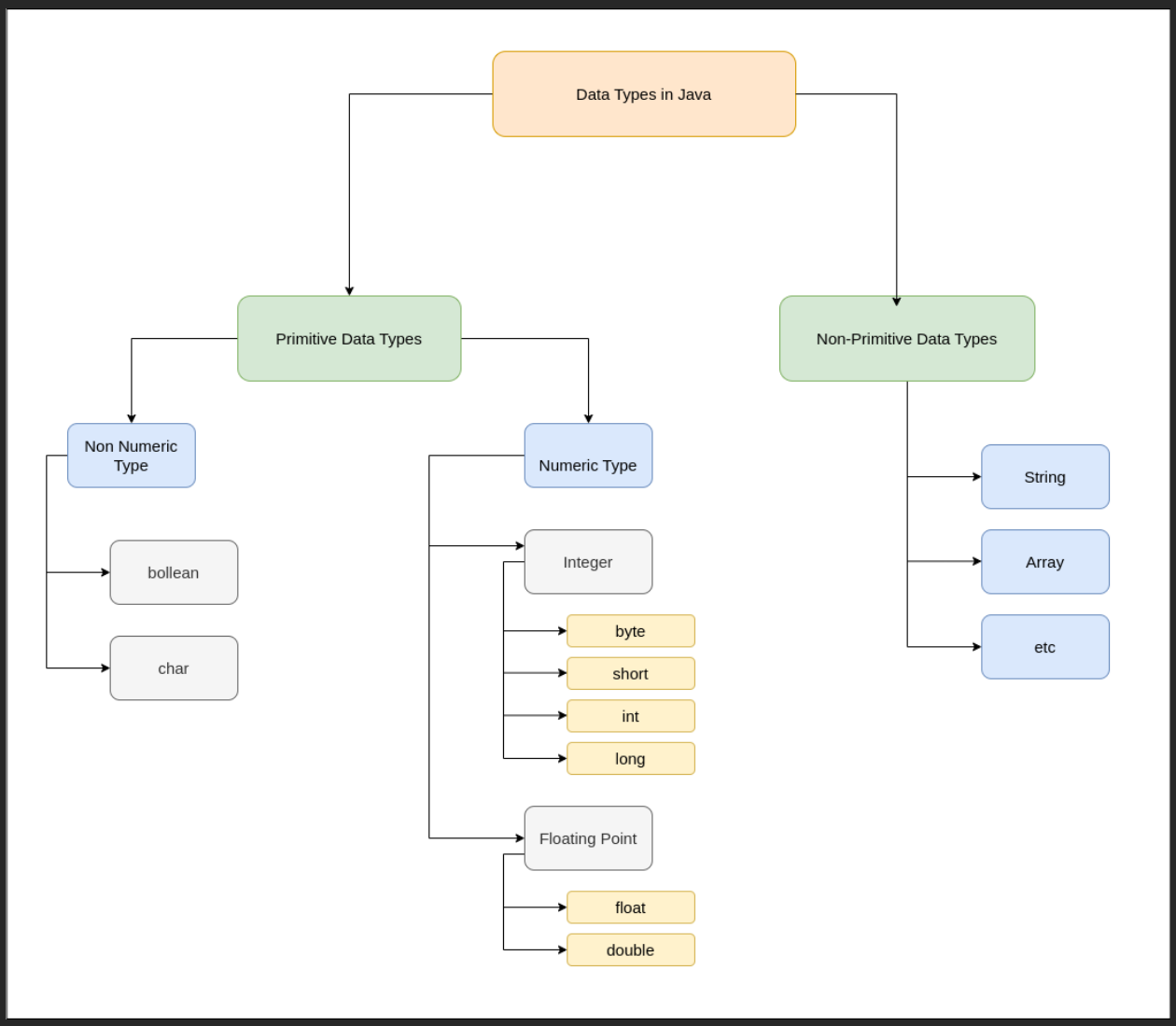
- Data -> processing

- During processing the place where data is stored temporarily is called Variable.

- Variable in Programming is a named storage location that holds a value or data.

Lecture 6:

Data Types



Primitive Data type

- Primitive data types are the fundamental data types provided by Java.

- These data types are predefined by Java and cannot be changed.

- Hold a single value.

- Don't have any built-in methods to manipulate that data.

There are eight primitive data types in Java:

- byte: 8-bit signed integer

- short: 2-byte signed integer

- int: 4-byte signed integer

- long: 8-byte signed integer (have to put 'l' at the end)

- float: 4-byte floating-point number (explicitly specify with 'f' at end)

- double: 8-byte floating-point number (default in java)

- char: 2-byte Unicode character (store single char in ' ')

- boolean: true or false value (boolean keyword to declare)

Non-Primitive Data Types

- Non-primitive data types are used to represent more complex data structures.

- Not predefined, they are user defined data types.

- **Reference-based:** Non-primitive data types of store references to the actual data, which is located in the heap memory. The variable itself holds the memory address of the object.

Here are some common examples of non-primitive data types in Java:

- `String` (pre-defined)
- `Array` (pre-defined).
- `Class`: A blueprint for creating objects (user-defined data types).
- `Interface` (user-defined data types).

Lecture 7:

Literals

- Any constant value which can be assigned to the variable is called literal/constant.

- int x = 100; // Here 100 is a constant/literal.

- int num = 0b1010; // for binary

- int num = 0x9B; // for hex value

- int num = 10_00_00_000; // For easy zero count

  System.out.println(num); // 100000000

Lecture 8:

Type Conversion and Casting

- Explicitly: Casting

e.g :

```
int b = 123;
byte a =(byte)b;
System.out.println(a); // 123
```

Here we convert int to byte, big to small. So, explicitly mentioned.

- Implicitly: Conversion

e.g :

```
byte b = 123;
int a = b;
System.out.println(a); // 123
```

Here we convert byte to int, small to big.

Converting out of range Value

- e.g : 257 to

```
int b = 257;
byte a =(byte)b;
System.out.println(a);// 1
```

Read more in DSA sharyians Note

Type Promotion

```
byte a = 30;
byte b = 100;
```

```
int result = a*b;// this value is outof range of byte and promoted to int
System.out.println(result);// 3000
```

Lecture 9:

Operators

- Same as other language

Lecture 10:

Relational operator

- <, >, ==, !=, <=, >=

Lecture 11:

Logical operator

- &, |
- && (AND) and || (OR) are called short-circuit

Lecture 12:

Conditional Statement

If-else

Lecture 13: if else if

Lecture 14:

Ternary Operator

- ?: concise way to write conditional statement

- e.g. int result = 4%2==0? 20 : 10;

Lecture 15:

Switch

- Based on particular value it will execute a particular case.

Lecture 16:

Loop

- To repeat same thing until condition become false.

Lecture 17:

while loop

A `while` loop is used when the number of iterations is not known, and the loop should continue until a certain condition is met

Lecture 18:

Do while loop

If we want to run the code at least one time, then repeat until the condition becomes false.

Lecture 19:

For loop

A for loop is generally used when the number of iterations is known beforehand.

Lecture 20:

Which loop to use ?

The above theory are good explanation of it.

Lecture 21:

Object Oriented Programming

Object

- have Properties and Behaviours
- Object – Instance of class

Class

- Act as blueprint to create Objects.
- JVM creates objects.

Lecture 22:

Creating Class and Object

```java
// Define a class named 'calculate'
class calculate {
    public int add(int n1, int n2) {
        return n1 + n2;
    }
}

class hello {

    public static void main(String [] args) {
        // Declare and initialize two integer variables
        int a = 50;
        int b = 45;

        // Create an instance (object) of the 'calculate' class
        calculate cal = new calculate();

        // Call the 'add' method of the 'calculate' class using the object 'cal'
        // Print the result of adding 'a' and 'b'
        System.out.println(cal.add(a, b));
    }
}
```

Lecture 23:

JDK, JRE and JVM

JVM

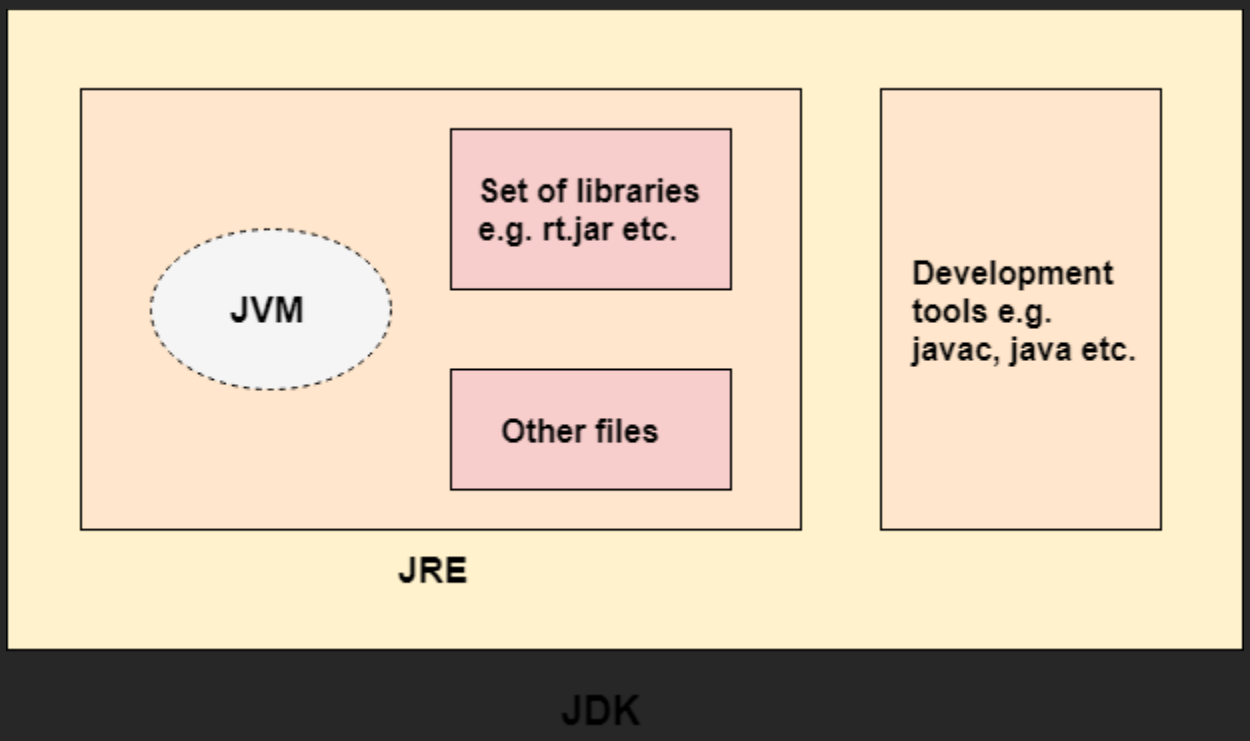
- The **Java Virtual Machine** (JVM) runs the Java bytecodes.
- The JVM doesn't understand Java source code; that's why you need compile your `*.java` files to obtain `*.class` files that contain the bytecodes understood by the JVM.
- It's also the entity that allows Java to be a "portable language or platform independent" (*write once, run anywhere*)

JRE / Java RTE

- It contains a set of libraries + other files that JVM uses at runtime.

JDK

- Complete development kit for Java, includes JRE and development tools.



Lecture 24:

Methods

- Note: In Java, all paths of a non-void method must return a value.

```java
public String getPen(int cost) {
    if (cost > 1) {
        return "pen";
    }
    return null; // Return null if cost is not greater than 1
}
```

Lecture 25:

Method Overloading

- Method overloading in Java is a feature that allows a class to have more than one method with the same name.

- Rules for Method Overloading
- **Different Parameter Lists**: The methods must differ in the number or type of parameters.
- **Return Type**: Method overloading is not determined by the return type of the method.

```java
class methodOverloading{
    // Single interger
    public void method(int a){
        System.out.println("Integer value is : "+a);
    }
    // Two integers
    public void method(int a, int b){
        System.out.println("Two Integer value is : "+ a +" and " + b);
    }
    // Double value
    public void method(double a){
        System.out.println("Double value is : "+a);
    }
    // String value
    public void method(String a){
        System.out.println("String value is : "+a);
    }
}

class hello{
    Run main | Debug main
    public static void main(String[] args) {
        methodOverloading obj = new methodOverloading();
        obj.method(5);
        obj.method(4, 6);
        obj.method(7.5);
        obj.method("hello");
    }
}
```

-

Lecture 26:

Local and Instance variable

```java
class hello {
    int value; // instance variable
    public int add(int a, int b) {
        return a + b;// local variable
    }
}
```

Stack and Heap (imp)

Re-watch the video again : 12min with 2x

When You cleared Write some Notes

Lecture 27:

Need of Array

- To store multiple values in single variable.
- It makes the code easy to maintain.
- In java Array store data of same type.

Lecture 28:

Creation of Array

- int arr[ ] = new int[5];
  Or
- int arr[ ] = {4,5,6…};
- By default, the integer array values will be 0.

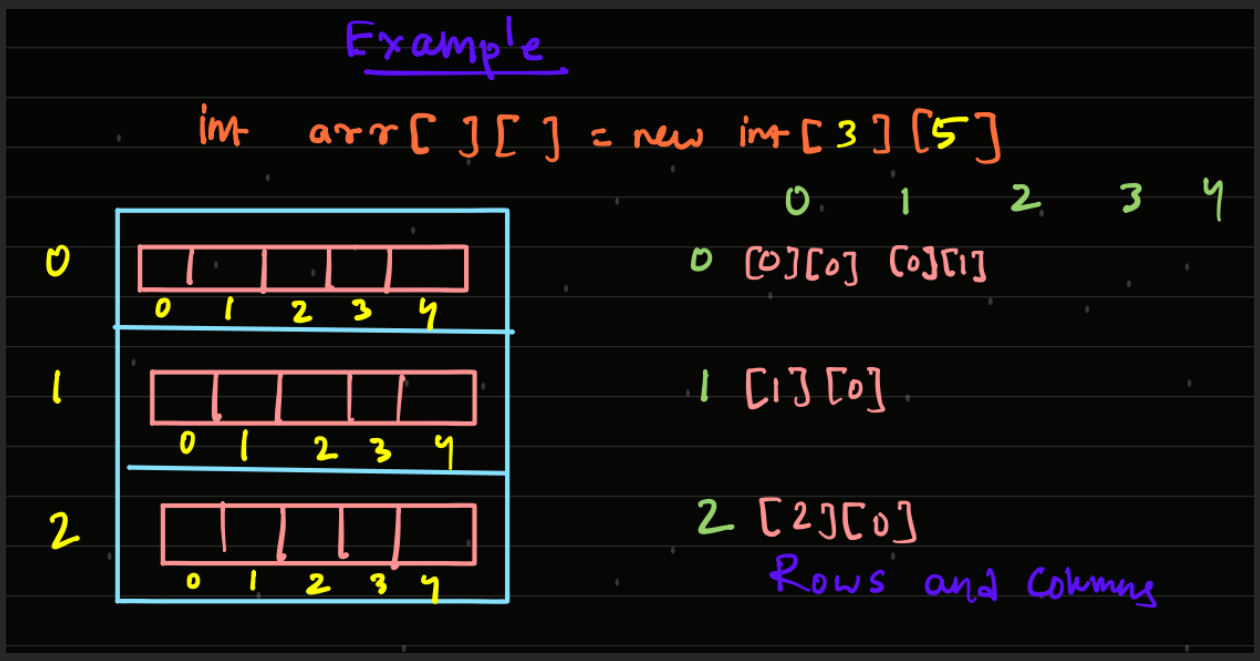
Lecture 29:

Multi-Dimensional Array

- A multidimensional array is an array of arrays.

2D Array

- `int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };`
- `int arr[][] = new int[3][5];`

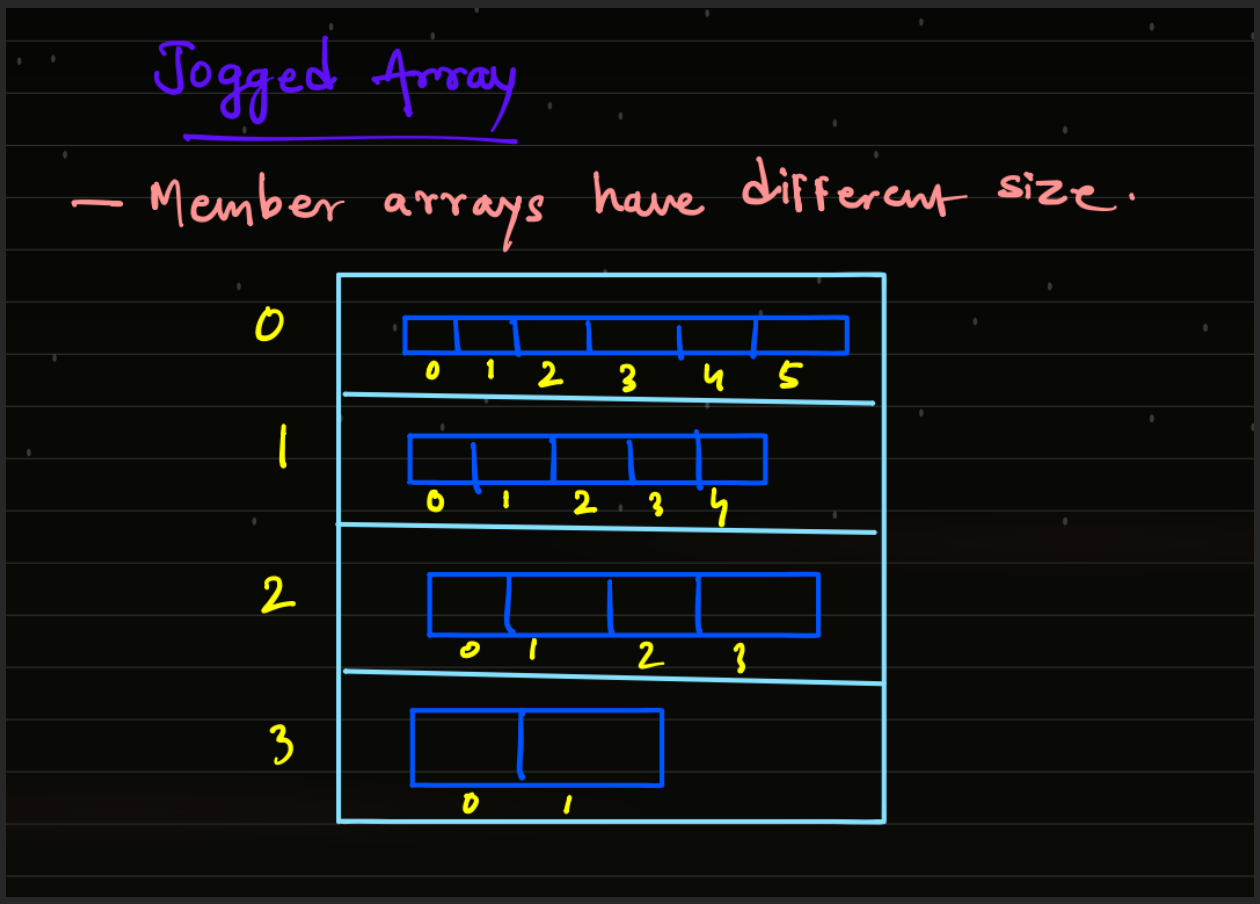


Enhanced For loop

- ```
  int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
  for (int[] row : myNumbers) {
    for (int i : row) {
      System.out.println(i);
    }
  }
  ```

Lecture 30:
  ```

Jagged Array



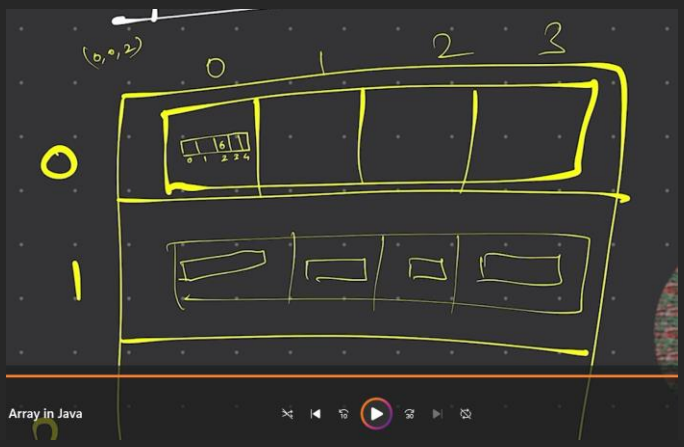
```java
// Jogged Array
int arr[][] = new int[3][];
    arr[0] = new int [1];
    arr[1] = new int [2];
    arr[2] = new int [4];

for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr[i].length; j++) {
        int ran = (int)(Math.random() * 101);
        arr[i][j] = ran;
    }
}
System.out.println();
System.out.println();

for(int n[] : arr){
    for(int a : n){
        System.out.print(a + " ");
    }
    System.out.println();
}
```

3D array

Int arr[ ][ ][ ] = new int [3][4][5]



Lecture 31:

Draws Backs of Array

- We can't increase or decrease the size of array in future.
- Searching and sorting we have to traverse through all element.
- Multiple data types can't be stored in a single array.

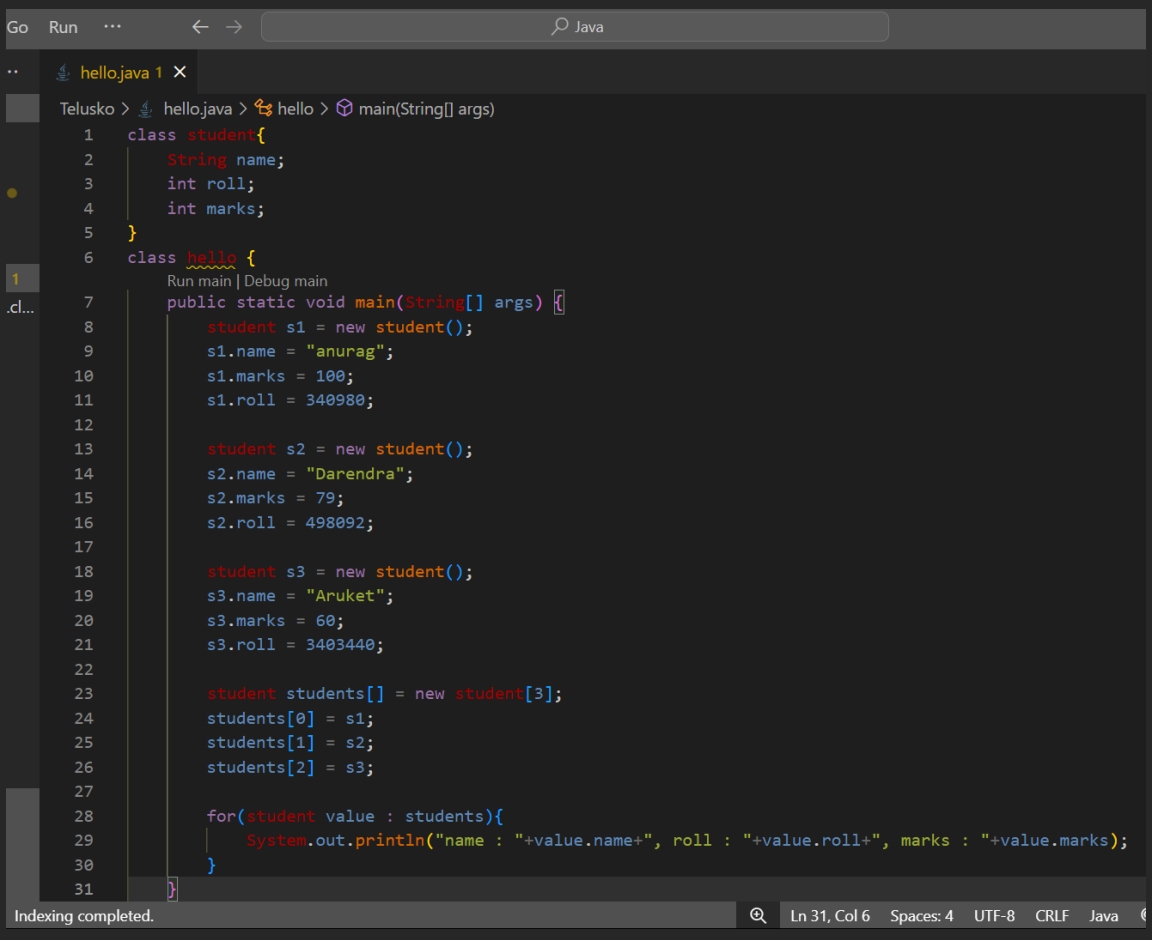
These are not drawbacks but that's how array works.

Upgraded version of array – Collections

Lecture 32:

- Exceptions are runtime error

Array of Objects




```java
class student{
    String name;
    int roll;
    int marks;
}
class hello {
    public static void main(String[] args) {
        student s1 = new student();
        s1.name = "anurag";
        s1.marks = 100;
        s1.roll = 340980;

        student s2 = new student();
        s2.name = "Darendra";
        s2.marks = 79;
        s2.roll = 498092;

        student s3 = new student();
        s3.name = "Aruket";
        s3.marks = 60;
        s3.roll = 3403440;

        student students[] = new student[3];
        students[0] = s1;
        students[1] = s2;
        students[2] = s3;

        for(student value : students){
            System.out.println("name : "+value.name+", roll : "+value.roll+", marks : "+value.marks);
        }
    }
```

Lecture 33:

Enhanced for loop

For(data_type value : array){ }

- This will iterate through each element of array

Lecture 34:

String

- String – class
- String str = “anurag “;
- String str = new String (“ anurag “);

Lecture 35:

Mutable and Immutable String

Immutable :

- String

This explains how String stored in stack and heap.

How same value string have one object reference.

Mutable:

- String buffer
- String builder

Lecture 36:

String Buffer

- StringBuffer name = new StringBuffer("anurag");

- name.append(" tarai");
- String buffer changes the anurag to anurag tarai in heap instead of creating another location for anurag tarai

Note:

- StringBuffer: Thread Safe
- StringBuilder: Not Thread Safe

Make Clear Note

Use Word + Ipad for note :

word – code and imp points

Ipad – diagram representation

Lecture 37:

Static Keyword

- When a variable is declared static then only one instance of this variable is created for all objects.
- Static variables can be used in non-static methods.
- Non static variable can't be referenced from static context.
- The static keyword belongs to the class than an instance of the class

`Static` keyword is known as a non-access modifier. This means that it does not affect the access level (visibility) of the variable, method, or block of code it is applied to, but rather it modifies other aspects of its behavior.

- Blocks
- Variables
- Methods
- Classes

Static Variable

hello.java 1 | methodOverloading.class

hello.java > hello > main(String[] args)

```java
class staticKey{
    static int value = 0; // declaring value as static

    public void Update(){
        value++;
    };
    public void show(){
        System.out.println(value);
    };
}
class hello {
    Run main | Debug main
    public static void main(String[] args) {

    // Creating two objects
    staticKey obj1 = new staticKey();
    staticKey obj2 =  new staticKey();

    // increamenting value using update() for each class
    // print the value using show()
    obj1.Update();
    obj1.show(); // 1

    obj2.Update();
    obj2.show(); // 2
    // accessing static variable using class name
    System.out.println(staticKey.value); // 2
    // this shows that value belongs to class not instance of object
    }
}

```

Lecture 38:

Static Method

- Can use static variable in static method.
- Can't use non static variable in static method.

```
class staticKey{
    int value = 0; // non static variable
    static  public void Update(){ // static method
        value++;
    };
}
```

- It creates confusion of which Object the value belong to.

Use non static variable in Static method

```
class staticKey {

    int value = 0; // non static variable

    static public void Update(staticKey object) { // static method
        object.value++;
    }
;

}
class hello {

    Run main | Debug main
    public static void main(String[] args) {

        // Creating two objects
        staticKey obj1 = new staticKey();
        staticKey obj2 = new staticKey();

        // we have to pass reference of object if we want to
        // use static method for non static variable
        staticKey.Update(obj1);
    }
}
```

Why Main method is Static ?

- If main is not static, we must create object of hello to invoke the main.
- But main is the starting point of execution, then how you can create object while the execution is not started.
- Thats why main is static, so we don't have to create class hello object to invoke main.

Lecture 39:

Static Block

- Every time we create objects there is two step : 1. class is loaded and 2. Object get instantiated
- Static block is called only once when class is loaded.
- Static {

  }
- We can load the class without creating object of it by using a class "Class" : Class.forName("class_name");

Lecture 40:

Encapsulation

- Encapsulating data with methods is called encapsulation.

Lecture 41:

Getters and Setters

- These are methods used to get value and set value for private variable in a class.

Lecture 42:

This

- Represents current object.
-

Lecture 43:

Constructor

- It is used to assign value to instance variable during creation of object.
- It is a method itself and has the same name as class.
- It never returns anything.
- Every time we create a object, the constructor will be called.

Lecture 44:

Constructor Overloading

```
class student {

    private String name;
    private int roll;

    public student() { // dfault constructor
       name = "anurag";
       roll = 230615107;
    }

    public student(String name, int roll) { // parameterized constructor
        this.name = name;
        this.roll = roll;
    }

    public String getName() {
        return this.name;
    }

    public int getRoll() {
        return this.roll;
    }
}

class hello {

    Run main | Debug main
    public static void main(String[] args) {
        student obj = new student();
        student obj1 = new student("tanbir", 230615102);
        System.out.println(obj.getName()+ " : " +obj.getRoll());
```

Lecture 45:

Naming conventions

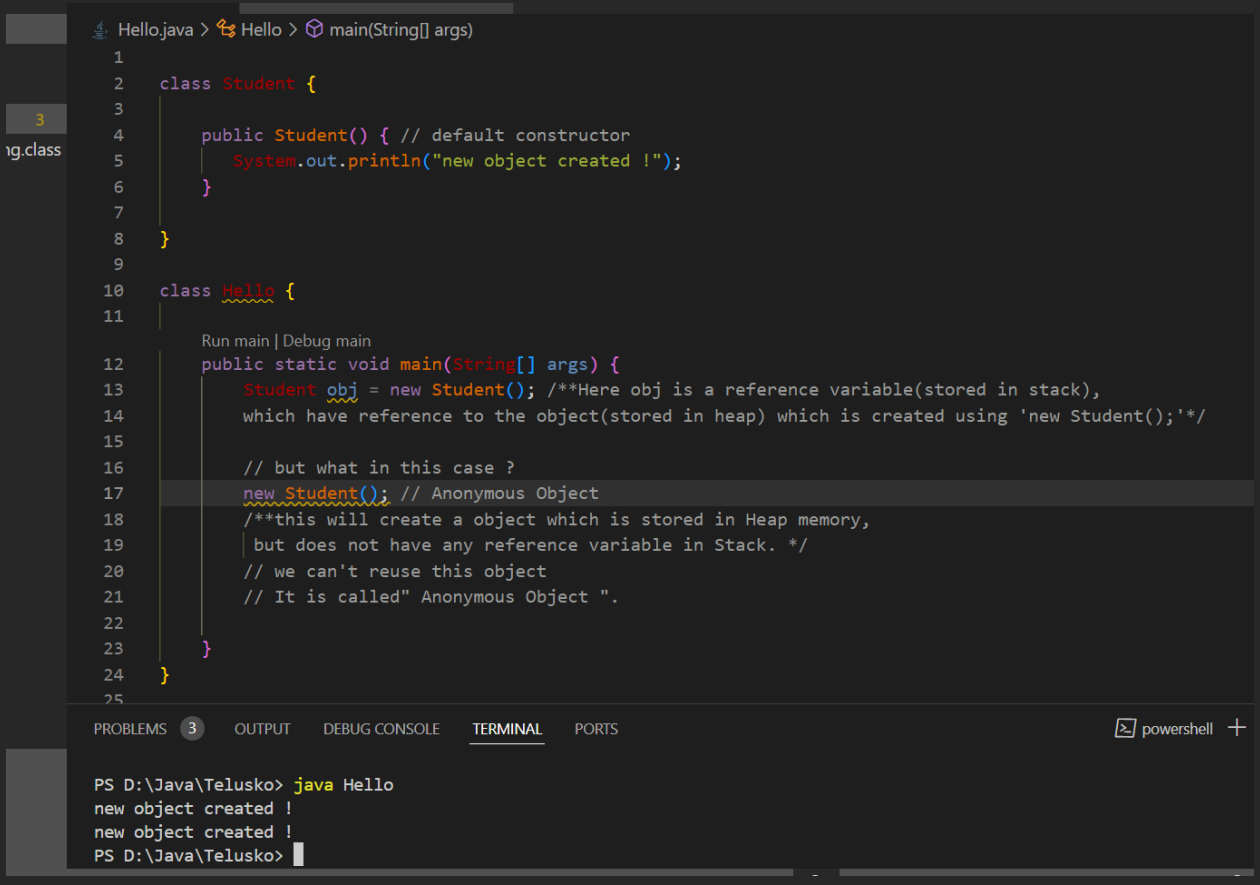
- PascalCase (classes and interfaces)

- camelCase (variable and method names)

- Constants – PIE, BRAND

Lecture 46:

Anonymous Object in java



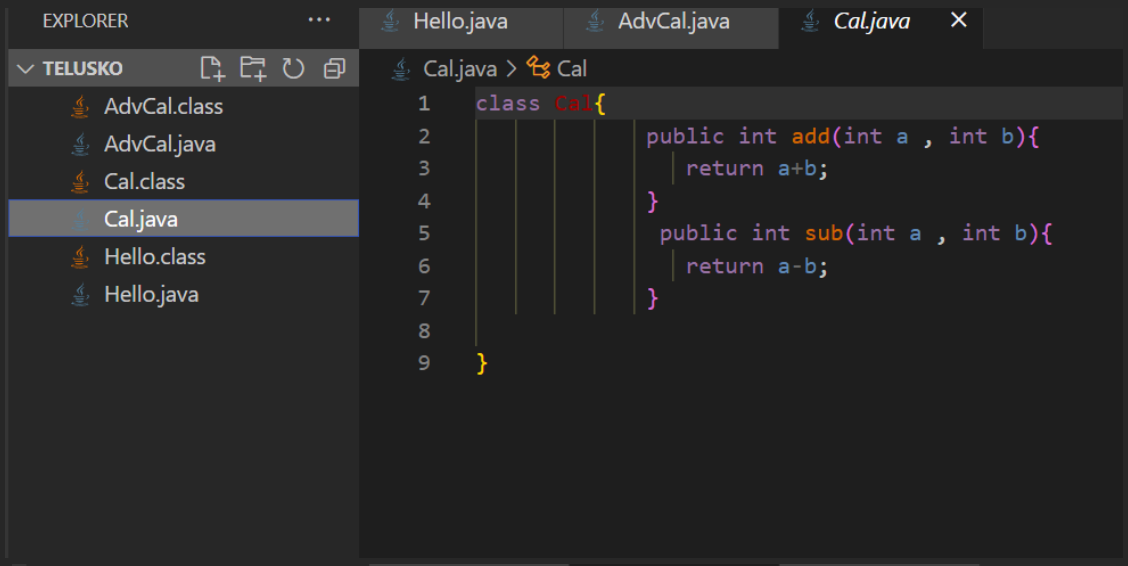
Lecture 47:

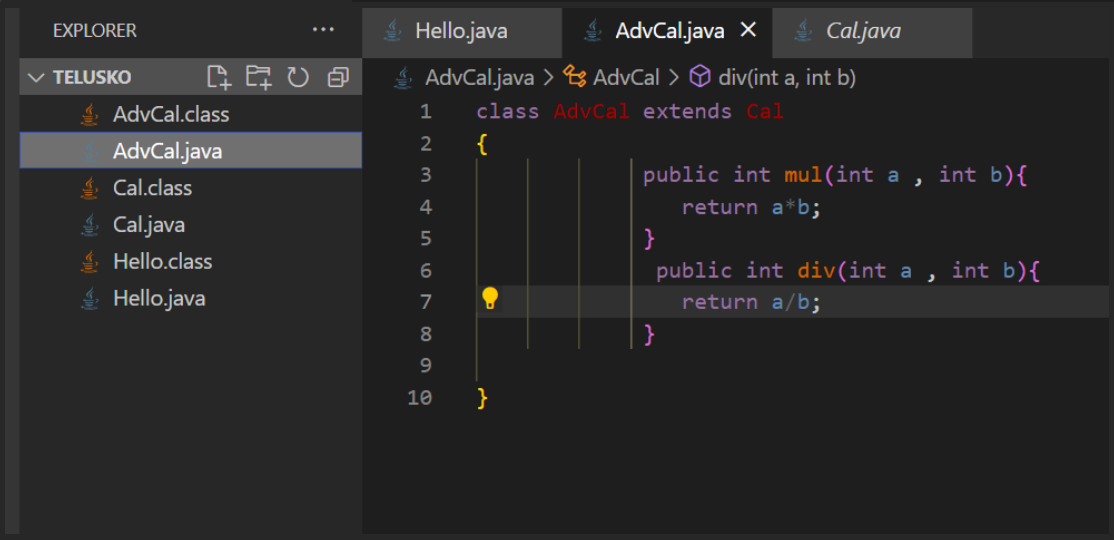
Need of Inheritance in Java

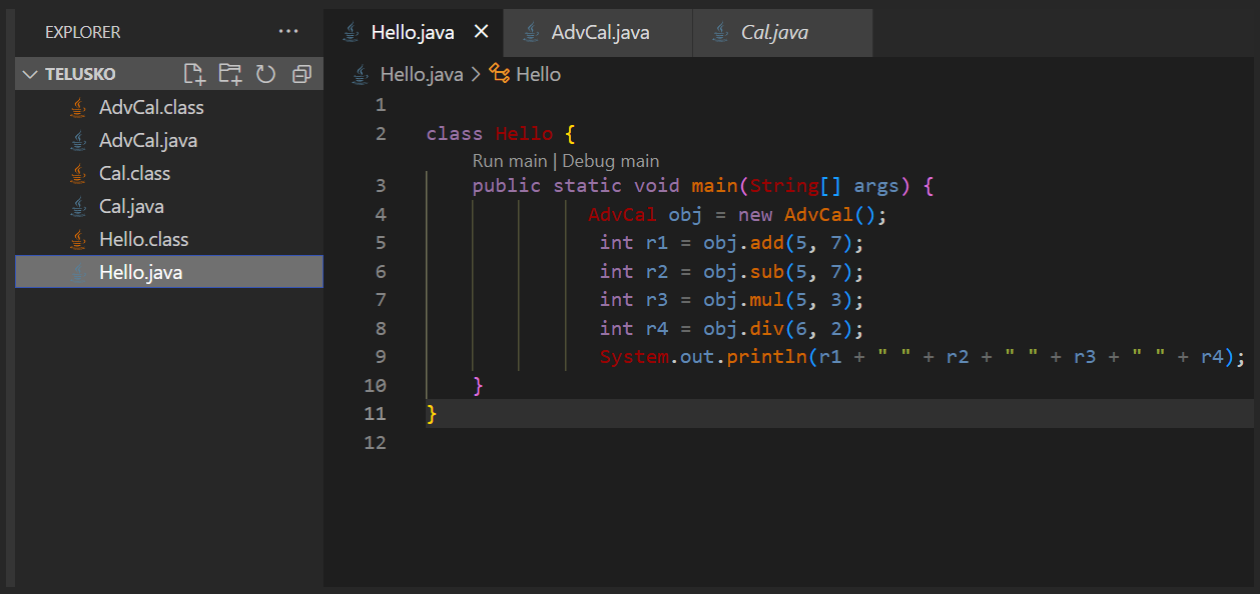
Lecture 48:

Example

- AdvCal inheriting the properties of Cal.



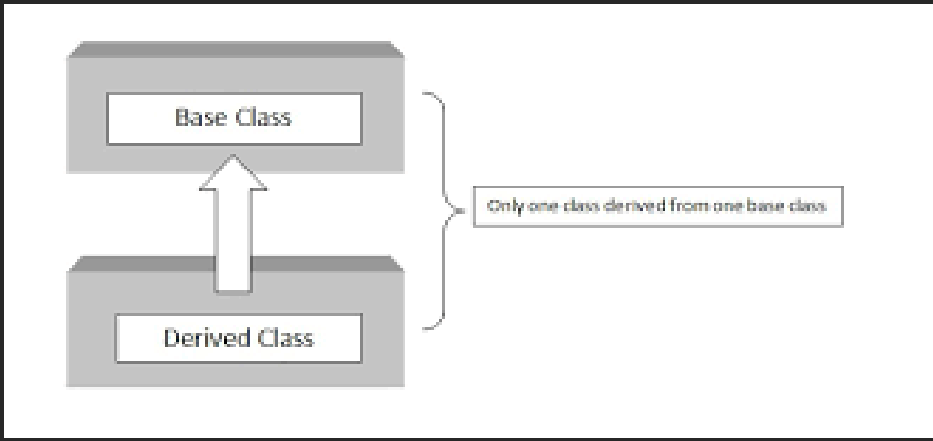




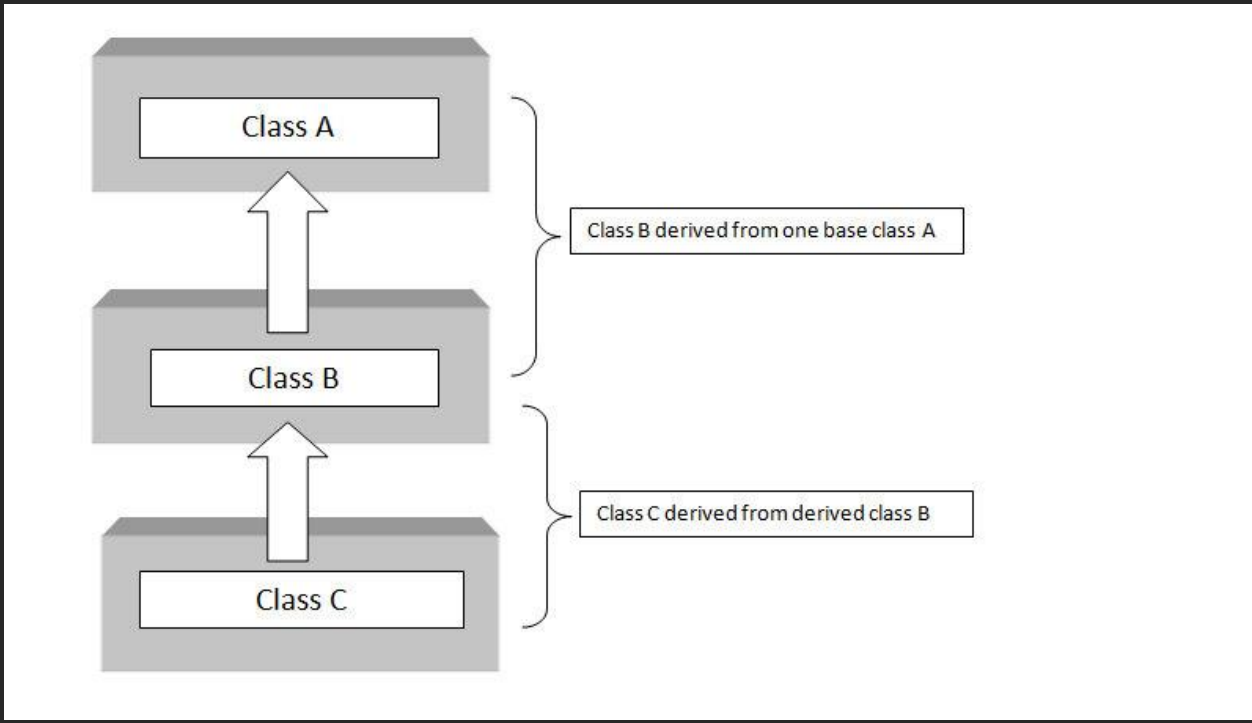
Lecture 49:

Single and Multiple Inheritance

- Single

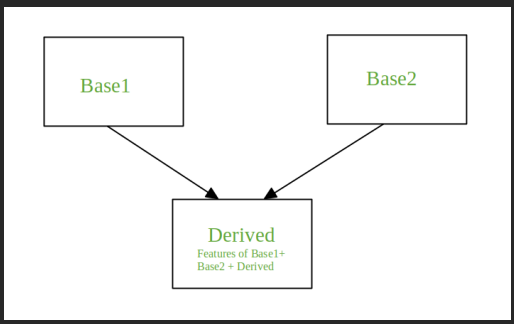


- Multilevel



Lecture 50:

- Multiple (Not support in java)
- NOTE : Java does not support multiple inheritance
  - What if both the parent or base class have method with same name.
  - Which one will be called when called using object of the child class.
  - This creates Ambiguity.



However, Java does support multiple inheritance with interfaces. An interface is a blueprint of a class that contains abstract methods.