

- ```
/**
```
- The `javap` command is a utility provided by the Java Development Kit (JDK) that displays information about the members of a compiled Java class.
  - In this case, the `javap` command is used to display information about the `Integer` class from the `java.lang` package.
  - 
  - The `Integer` class is a wrapper class for the primitive type `int` in Java. It provides various methods and constants for working with integer values.
  - By using the `javap` command with the `java.lang.Integer` class, you can see the details of its members, such as fields, constructors, and methods.
  - 
  - To use the `javap` command, you need to have the JDK installed on your system and set up the Java environment variables properly.
  - Once you have the JDK installed, you can open a command prompt or terminal and run the `javap` command followed by the fully qualified name of the class you want to inspect.
  - In this case, the command would be `javap java.lang.Integer`.
  - 
  - Running the `javap` command will display the details of the `Integer` class, including its fields, constructors, and methods, along with their signatures and modifiers.
  - This information can be useful for understanding how the `Integer` class is implemented and for exploring its available functionality. //\*
  - This code snippet demonstrates the usage of the `javap` command to inspect the methods of the `java.lang.Integer` class.
  - The `javap` command is a command-line utility that displays information about the members of compiled Java classes.
  - In this case, it is used to display the methods available in the `Integer` class. \*/ Java is class based high level, object oriented programming language which is statically typed, both compiled and interpreted language and it follows the write once run anywhere principle.

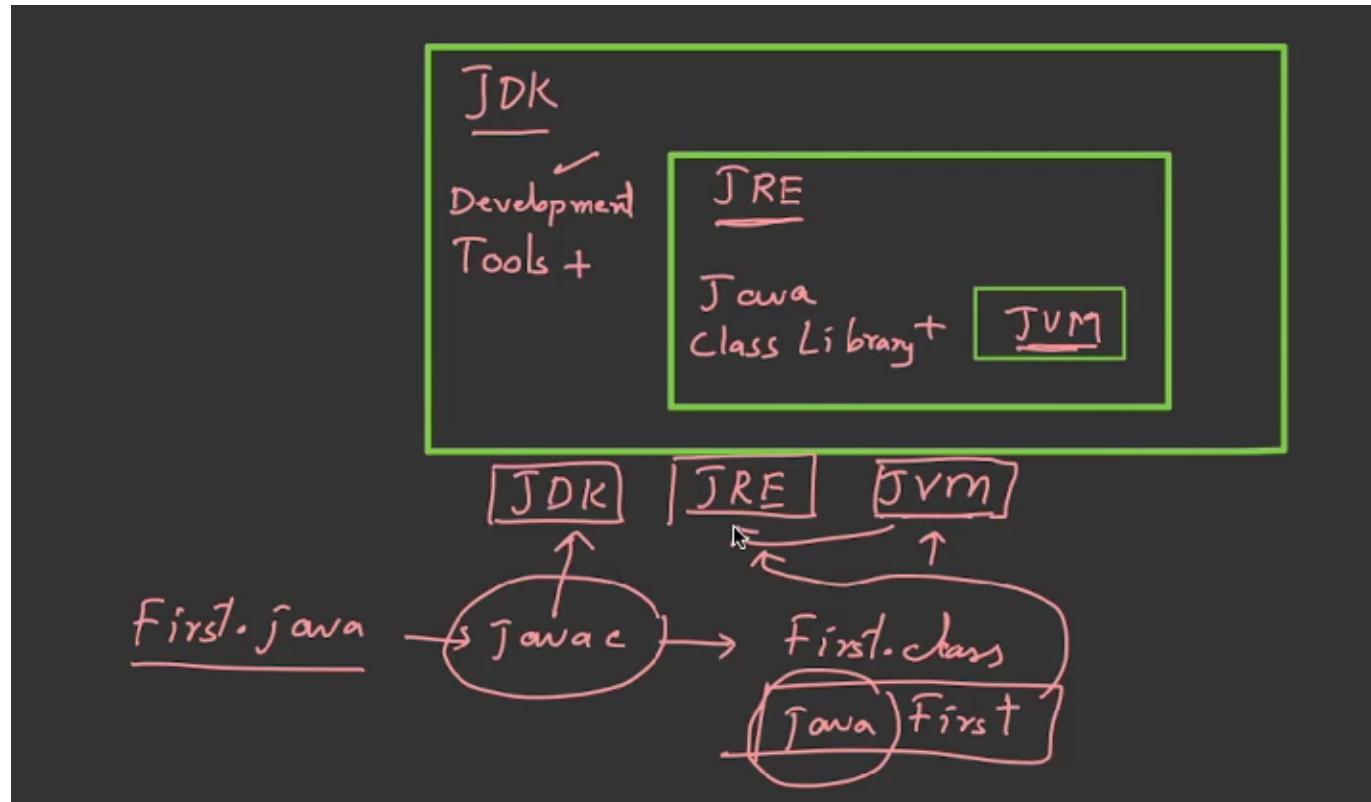
(java's byte code makes it platform independent and secure. java is the most popular programming language and it is used in the development of web applications, mobile applications, enterprise applications, and embedded systems.)

Java is high level, platform independent, object oriented, secure, robust, distributed, multi-threaded, portable, and dynamic language

## Introduction

---

1. Instruction Note
2. Download and Install JDK



For writing and developing java programs, we require JDK

JDK will have a compiler and runtime environment, which will help us write Java programs and execute them.

Along with the installation of JDK, we will get the JRE and JVM.

JDK means Java Development Kit which contains debugging tools, development tools(Javac, Java), and JRE

for executing java programs we require jvm which is the part of JRE

we say java programs are executed in JRE, but actually, they are executed in JVM

## Installation

Google search results for "download jdk":

- Oracle** <https://www.oracle.com/java/technologies/javase-downloads.html>
- Java Downloads | Oracle India** <https://www.oracle.com/java/technologies/javase-downloads.html>

About 62,90,00,000 results (0.30 seconds)

Download the Java including the latest version 17 LTS on the Java SE Platform. These downloads can be used for any purpose, at no cost, under the Java SE ...

1. Go to the Oracle website and download the JDK
2. Install the JDK
3. Set the environment variables preferred to set at the system variables
4. Verify the installation cmd> java -version cmd> javac -version

preferred version is LTS(Long Term Support) version JAVA 8 is popular and widely used (in the course java 13 or 14 is used)

File explorer> C:\Program Files\Java\jdk-14.0.1\bin with java dir all the java versions will be installed

```
C:\Users\Abdul Bari>javac -version
javac 17.0.7

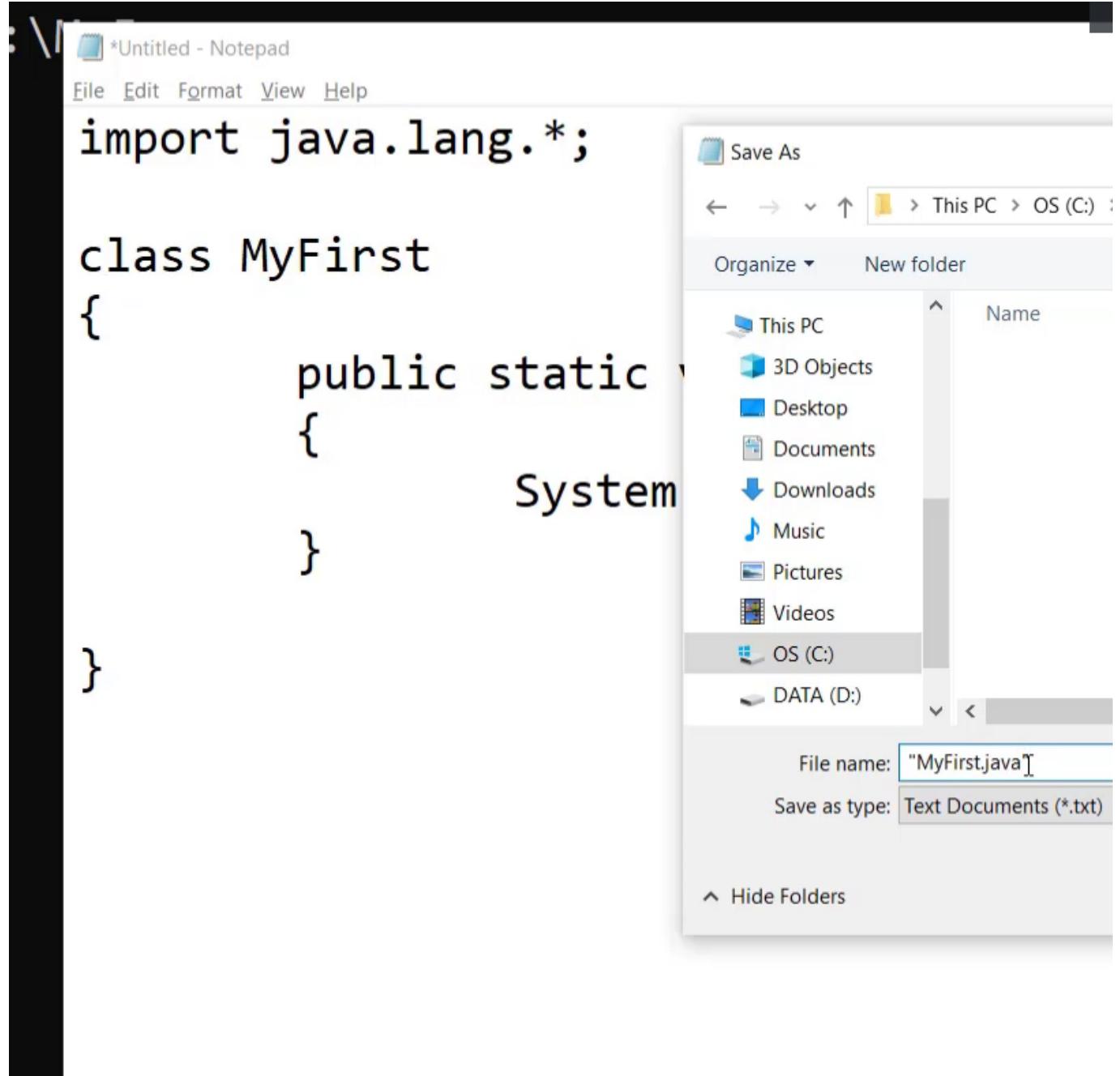
C:\Users\Abdul Bari>java -version
java version "17.0.7" 2023-04-18 LTS
Java(TM) SE Runtime Environment (build 17.0.7+8-LTS-224)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.7+8-LTS-224, mixed mode)

C:\Users\Abdul Bari>ja
```

dir //to see the list of files in the directory md java //to create a directory

```
import java.lang.*;//importing the lang package happens by default
class Hello{
    public static void main(String[] args){
        System.out.println("Hello World");
    }
}
```

when you are writing the first code in notepad then recommended to save the file with .java extension



//wrap around double quotes

```
C:\MyJava>type MyFirst.java
import java.lang.*;

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

C:\MyJava>type MyFirst.class
javac MyFirst.java
```

- to compile java program

```
javac Hello.java
```

- to run java program

```
java Hello
```

## 5. Skeleton of Java Program

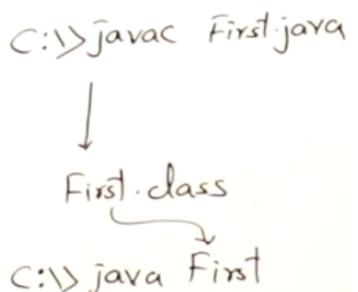
Skeleton of JAVA Program

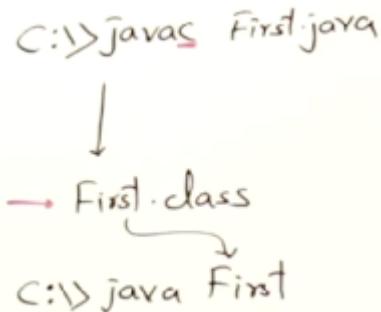
First.java

```
import java.lang.*;
class First
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

lang is basic package and get's imported by default

class name should be same as the file name inside the class, we can have multiple methods main method is the entry point of the program





if you are compiling and running the program from cmd

if you want to use anything from the class, then you need to create the object of the class. if it is static then you can directly use it without creating the object of the class

JVM initially calls the main method of the class First.main()

in java there is nothing outside the class and object System.out.println() is the method of the out object of the System class

## 6. Exploring the First Skeleton Program

System class is present in the lang package out is the object of the PrintStream class println is the method of the PrintStream class

when class is public then the file name should be same as the class name when class is not public then the file name can be anything

```

C:\MyJava>java First
Error: Main method not found in class First, please define the main method as:
  public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
  
```

```

public static void main(String arg[])
{
    System.out.println("Hello World");
    System.out.println(arg[0]);
}
  
```

```

1 C:\MyJava>java First
Hello World
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds
for length 0
        at First.main(First.java:8)

C:\MyJava>java First Bye
Hello World
Bye
  
```

## 7. Reading from Keyboard

java provides a class called scanner that is used for reading the input from different sources like keyboard, etc.

util is the built-in package in java, which contains the scanner class

```
import java.util.Scanner;
class Hello{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number");
        int a = sc.nextInt();
        System.out.println("The number is "+a);
    }
}
```

class Scanner

- nextInt() : reads the integer
- nextFloat() : reads the float
- nextDouble() : reads the double
- next() : reads the string
- nextLine() : reads the line
- nextBoolean() : reads the boolean
- nextByte() : reads the byte
- nextShort() : reads the short
- nextLong() : reads the long

//before reading the input , you need to verify the data

- hasNextInt() : checks if the next token is an integer
- hasNextFloat() : checks if the next token is a float

//close the scanner resource

- close() : closes the scanner

## 8. Reading the scanner

System.out is the object attached to the console System.in is the object attached to the keyboard

Scanner class is used to read the input from the keyboard

exception are the runtime errors which abruptly terminate the program need to be handled

```
import java.util.*;

public class Main{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int a,b;
        System.out.println("Enter two number : ");
```

```
a = sc.nextInt();
b = sc.nextInt();
System.out.println("sum is "+(a+b));
sc.close();
}
}
```

```
javap java.util.Scanner
```

//is used to see the methods of the class

## Section 2: Data Types and Operators

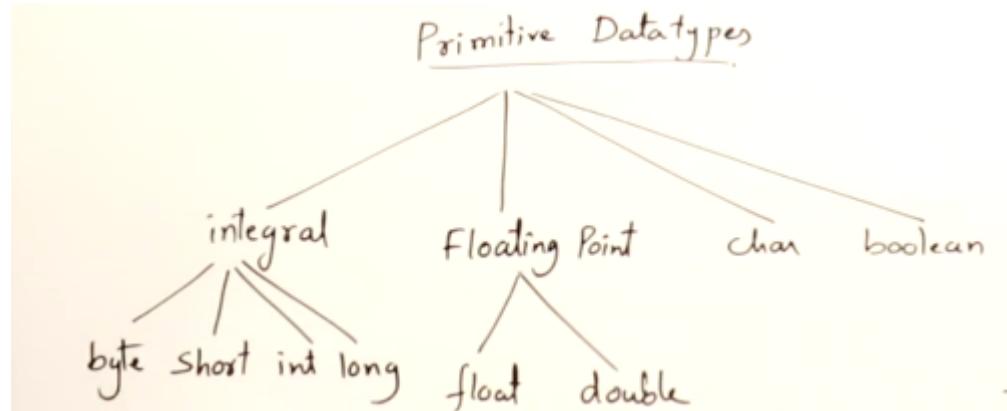
### 10. Introduction to Data Types

Data types are used to define the type of data a variable can store Date is the most important part of the program and ingredients of the program on which we perform the operations and do the processing

when program is running in the memory then the data of the program is hold by the variables

variables are meant for storing the data variables has the type of data and the name of the variable

primitive data types are the basic data types and inbuilt data types of the java program



| Type    | Size | Range                                                    | Default  |
|---------|------|----------------------------------------------------------|----------|
| byte    | 1    | -128 to 127                                              | 0        |
| short   | 2    | -32768 to 32767                                          | 0        |
| int     | 4    | -2147483648 to -2147483647                               | 0        |
| long    | 8    | —                                                        | 0        |
| float   | 4    | $\pm 1.4 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$    | 0.0f     |
| double  | 8    | $\pm 4.39 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ | 0.0d     |
| char    | 2    | 0 to 65535                                               | '\u0000' |
| boolean | ?    | true/false                                               | false    |

depending the size of data that specific data type can be used.

java supports the unicode character set means it supports the international languages char is 2 bytes and unicode format

float follows the IEEE 754 standard double follows the IEEE 754 standard

float is able to store the long

## 11. check size and range of the data types

for primitive data types there are equivalent wrapper classes are available. In those classes there are some constants are available which are used to check the size and range of the data types

```
class Main{
    public static void main(String args[]){
        System.out.println("Size of int : "+Integer.SIZE);
        System.out.println("Size of int : "+Integer.BYTES);
        System.out.println("Min value of int : "+Integer.MIN_VALUE);
        System.out.println("Max value of int : "+Integer.MAX_VALUE);
    }
}
```

javap java.lang.Integer

during the execution of the program, the data is stored in the variables and the variables are stored in the memory

local variables must be initialized before using them instances variables are initialized by default values

```
class Main{
    int a; //instance variable
    public static void main(String args[]){
        int b; //local variable
        System.out.println(a);
        System.out.println(b);
    }
}
```

if float is given to int then it will give the error

```
class Main{
    public static void main(String args[]){
        float a = 10.2;
        int b = a;
        System.out.println(b);
    }
}
```

## 13. Rules for variable names

while developing the big programs, we need to follow the naming conventions and the rules for the variable names

1. variables are case sensitive
2. variables can start with the alphabet, underscore, and dollar sign eg. int room\_number, cabin151, \_room, \$room
3. after the first character, we can use the digits
4. we can't use the reserved keywords as the variable names
5. we can't use the special characters except the underscore and dollar sign
6. we can't use the spaces in the variable names
7. we can't use the length more than 255 characters
8. we can't use the same name for the class and the variable
9. we can't use the same name for the method and the variable

int String; (above is allowed but not recommended)

camelCase is the recommended naming convention for the variables and the methods eg. roomNumber, averageMarks, studentName, employeeSalary PascalCase is the recommended naming convention for the

classes UPPERCASE is the recommended naming convention for the constants eg. PI, MAX\_VALUE, MIN\_VALUE

sample variable names: int rollNumber\$Student; //allowed int rollNumber\_Student; //allowed int rollNumber-Student; //not allowed int rollNumber Student; //not allowed int rollNumberStudent; //allowed (camelCase) //recommended class Student{} //allowed (PascalCase) //recommended final int PI = 3.14; //allowed (UPPERCASE) //recommended

## 15. Integra Data Types in Detail

olden days we have 16 and 32 bit systems

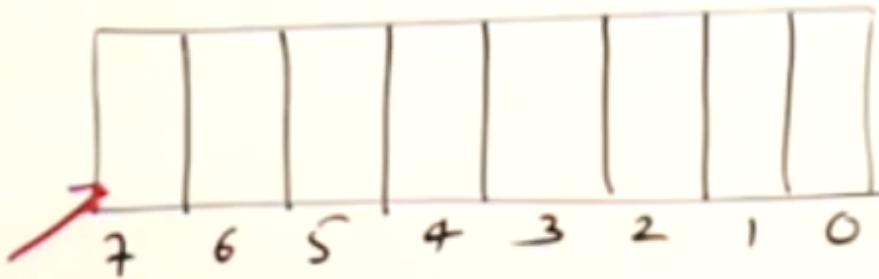
now modern days we have 64 bit systems

in a single shot how many bits are processed by the CPU is called the word size of the CPU cycle = 1 bit processing by the CPU in a single cycle, the CPU can process the data of the word size word size is the number of bits processed by the CPU in a single cycle

java has short and byte to support the backward compatibility with olden languages like C , cobol, etc

byte takes 8 bits in 7 bits what is the least value can be stored is -128 in 7 bits what is the max value can be stored is 127 7th bit is used for the sign

byte.



ign 0 0 0 0 0 0 0 — 0

o tve 0 1 1 1 1 1 1 — 127  
-ve  $2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1$

-5

$$\begin{array}{r} -5 \\ \hline 11111010 \\ +1 \\ \hline 11111011 \end{array}$$

1's comp 1 1 1 1 1 0 1 0  
2's comp 1 1 1 1 1 0 1 1  
 $2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1$

## 16. check binary bits of an integer

to see the binary bits of the integer we can use the wrapper class method `toBinaryString()`

```
class Main{
    public static void main(String args[]){
        System.out.println(Integer.toBinaryString(10));
    }
}
```

other methods

- toHexString()
- toOctalString()

## 17. Floating Point and Character in Detail

float and double are the floating-point data types decimal point is not stored in the memory then how the

$$\begin{aligned} &\text{Float} \\ &163.52 \\ &163.52 \times \frac{1}{100} \\ &16352 \times \frac{1}{100} \\ &16352 \times 10^{-2} \quad \text{exponent} \\ &\text{mantissa} \end{aligned}$$

$$16352 E -2$$

decimal point numbers are stored in the memory ? represented in the scientific notation it is represented in the mantissa and the exponent upto 6-7 decimal places float is suitable upto 15 decimal places double is suitable

ASCII is the 7 bit character set A,B,C...Z (65,66,67...90) a,b,c...z (97,98,99...122) 0,1,2,3,4,5,6,7,8,9 (48,49,50,51,52,53,54,55,56,57) +, -, \*, /, %, &, |, ^, ~, <<, >>, >>> (43,45,42,47,37,38,124,94,126,60,62,62,62)  
Unicode is the 16 bit character set

boolean size is not defined in the java, it is JVM dependent

visiting [unicode.org](http://unicode.org)

<https://unicode.org/charts/> click the language you need that are hexadecimal codes

```
//devangri // class Main{ // public static void main(String arg[]){ // // char c = 0x03C8;
// //greek language // for(char c =0x0900;c<=0x0970;c++) // System.out.print(c+ " "); // } // }
```

float f=35.6; it is invalid. 35.6 is a double literal. it should be float f=35.6f;

## Section 3: Setup Java Environment

---

19. Notepad++ Installation

20. Eclipse Installation

21. NetBeans Installation

22. IntelliJ IDEA Installation

## Section 4: Features and Architecture of Java

---

23. Compiler vs Interpreter

interpreter and compiler are used for translating our programs into the machine language and executing them

compiler translates the entire program at once and then executes the program

interpreter translates the program line by line and then executes the program

interpreter language are easy to debug and learn compiler languages are fast and efficient and difficult to debug java is both compiled and interpreted language

24 How java is platform independent

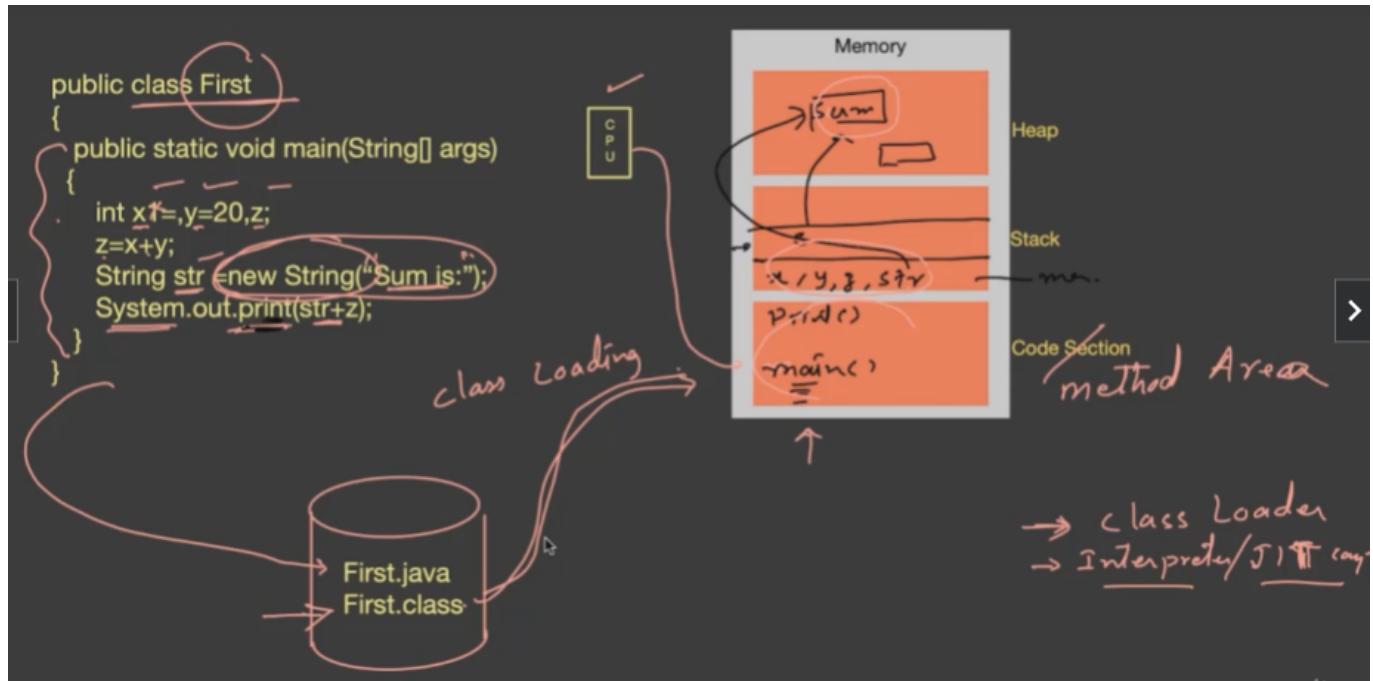
java is platform independent because of the JVM java is platform independent because of the byte code

program makes the system calls to the OS. in java the system calls are made by the JVM

JVM and byte code makes the java platform independent.

26. JVM Architecture

=> Class Loader Subsystem => Memory Area => Execution Engine (interpreter + JIT compiler) => Native Method Interface



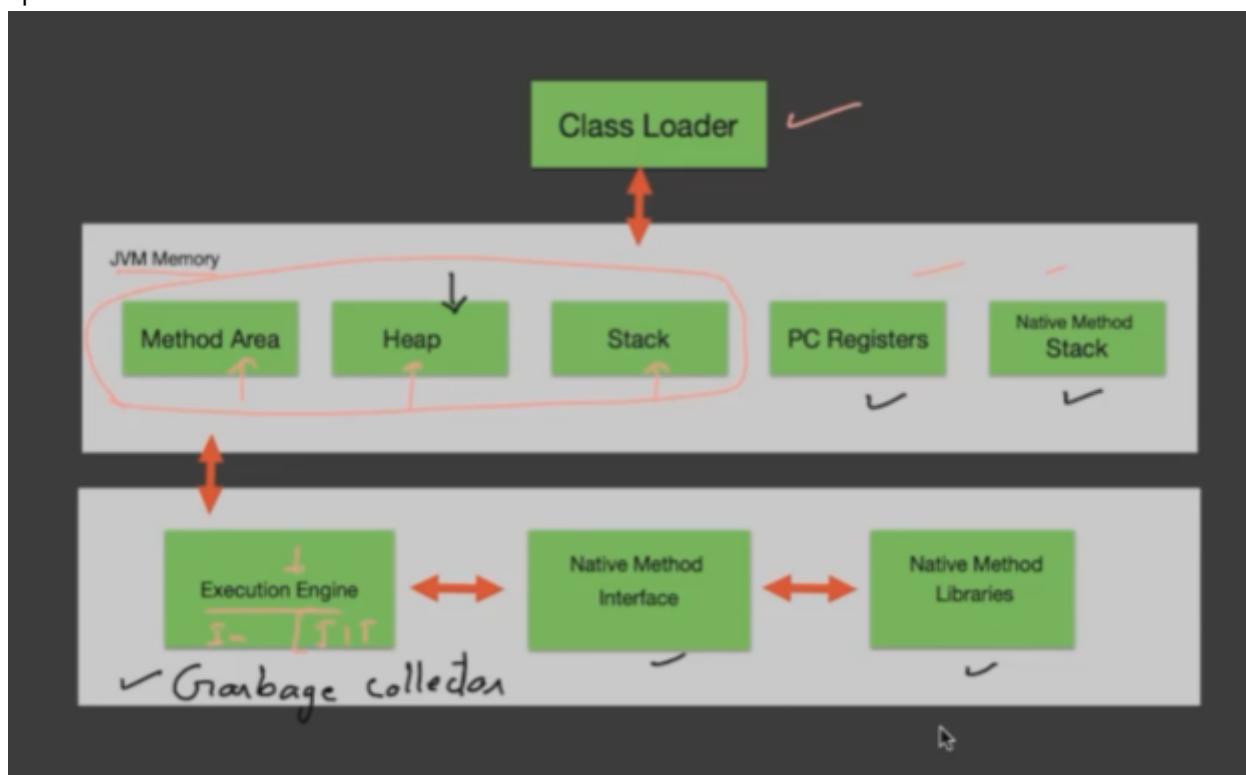
literals are kept in the string pool

```

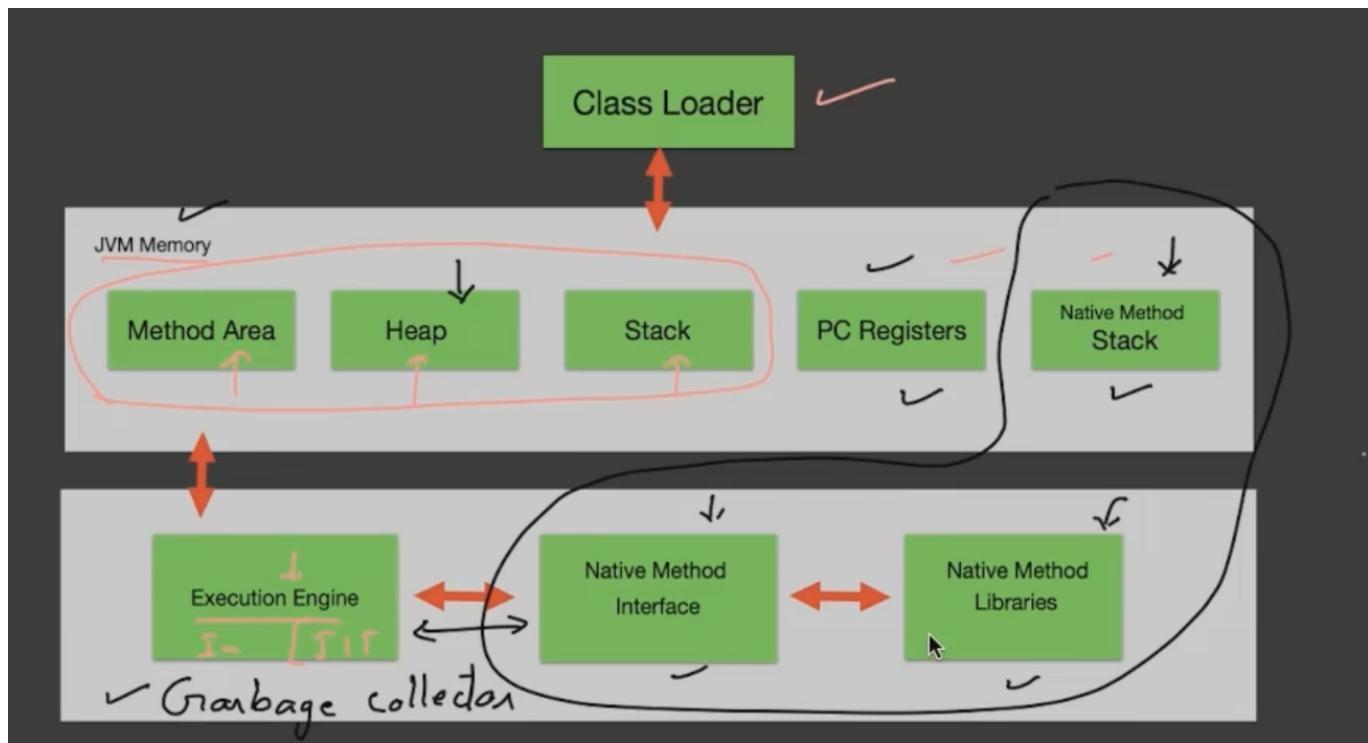
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");

```

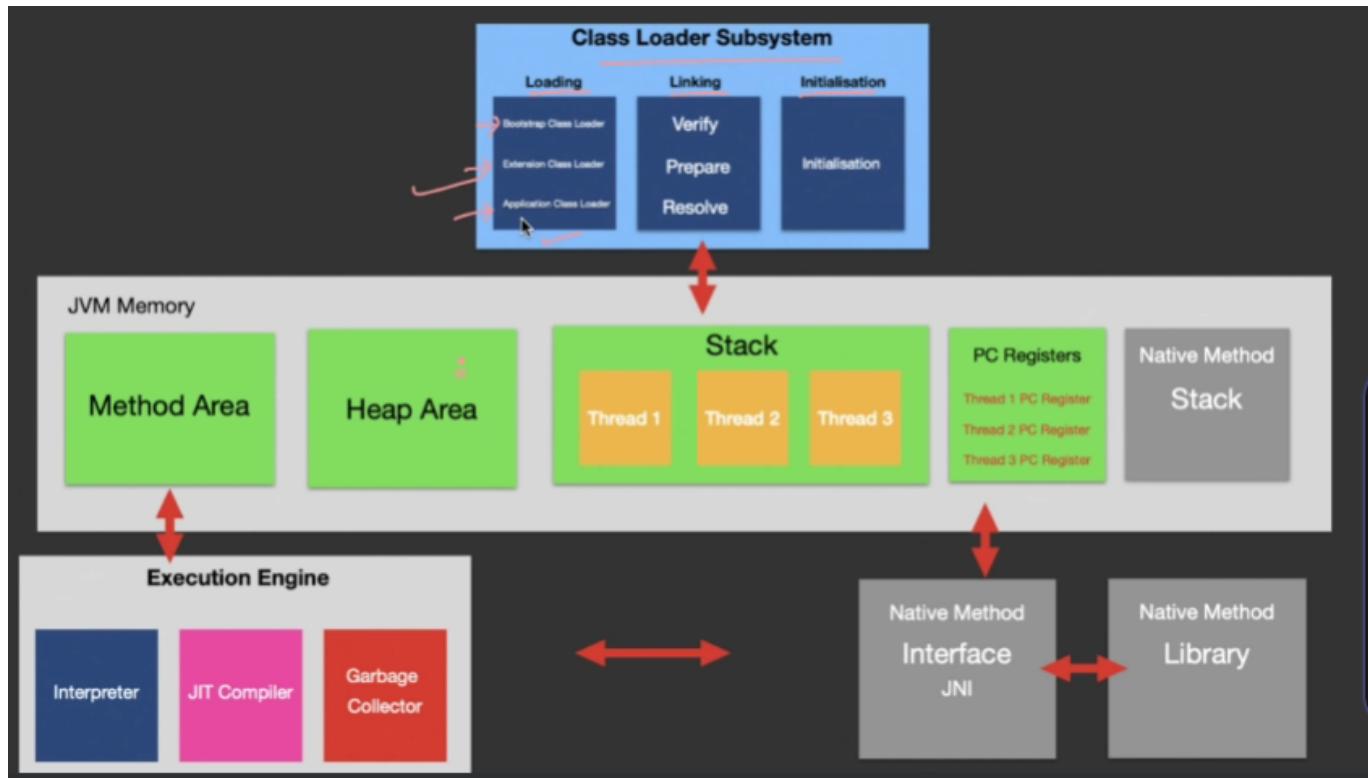
cpu store the addresses of instruction for next instruction to execute



external



## 27 JVM architecture in Detail



{ Application class loader loads the classes (our classes) whatever we are using in our program }

bootstrap class loader loads the classes which are present in the rt.jar file

extension class loader loads the classes which are present in the ext folder }

{ Linking is the process of combining the code of the class with the code of the other classes verify checks the byte code is valid or not (secure) prepare allocates the memory for the static variables and initializes the }

default values resolve replaces the symbolic references with the direct references (actual linking of the code happens here) }

{ Initialization is the process of executing the static blocks and initializing the static variables }

stack has multiple threads and each thread has its own stack and pc register has the address of the next instruction to execute

Interpreter reads the byte code and executes the instructions JIT compiler compiles the byte code into the native code and then executes the native code Garbage collector (execution engine) is the part of the memory area which is used to destroy the objects which are not in use

## 28. Java Features

1. Simple (syntax is similar to C and C++ and easy to learn and write the programs)
2. secure (java is secure because of the byte code and the JVM) (viruses can't be created in java)
3. Portable and platform independent (java is portable because of the byte code and the JVM)
4. Object Oriented (java is object oriented because of the classes and objects) (encapsulation, inheritance, polymorphism, abstraction)
5. Robust (java is robust because of the exception handling and the garbage collector)
6. Multithreaded (java is multithreaded because of the threads)
7. Architecture Neutral (Hardware and OS neutral)(java follows von neumann architecture)
8. Interpreted (java is interpreted because of the JVM)
9. High Performance (java is high performance because of the JIT compiler)
10. Distributed (java is distributed because of the RMI and EJB) (RMI is used for the remote method invocation and EJB is used for the enterprise java beans) (RMI and EJB are outdated) (spring framework is used in modern days and spring boot is used for the microservices)
11. Dynamic (java is dynamic because of the reflection and the class loader)

## Section 5: operators and expressions

---

### 29. Arithmetic operators and expressions

Arithmetic operations can't be performed on the boolean data type in java mod operator can be used for the float and double data types

```
class Main{  
    public static void main(String args[]){  
        int a = 10;  
        int b = 20;  
        System.out.println(a+b);  
        System.out.println(a-b);  
        System.out.println(a*b);  
        System.out.println(a/b);  
        System.out.println(a%b);  
    }  
}
```

**coercion** is the process of converting the data from one type to another type

```
byte b=10;
short s=15;
int i=7;
long l=50L;
float f=12.5f;
double d=17.5d;
char c='A';
```

(int)

byte + byte  
short + short  
byte + short  
int + int  
short + int

Coercion

|              |             |               |              |
|--------------|-------------|---------------|--------------|
| <b>int</b>   | $X = b + s$ | <b>int</b>    | $X = c + s;$ |
| <b>int</b>   | $X = s + i$ | <b>int</b>    | $X = c + i;$ |
| <b>float</b> | $X = i + f$ | <b>double</b> | $X = f + d;$ |
| <b>float</b> | $X = l + f$ | <b>double</b> | $X = l + d;$ |

### Increment / Decrement

post ++, post --  
++ pre, -- pre

### Arithametic

\* / %  
+, -

### Bitwise

&, |, ~, ^, <<, >>, >>>

### Relational

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

### Logical

&&, ||, !

(5\*10/2) is the expression

calculate the area of triangle

quadratic equation

### Roots of Quadratic Equation

$$\rightarrow (ax^2 + bx + c = 0)$$

$$(x + r_1)(x + r_2) = 0$$

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

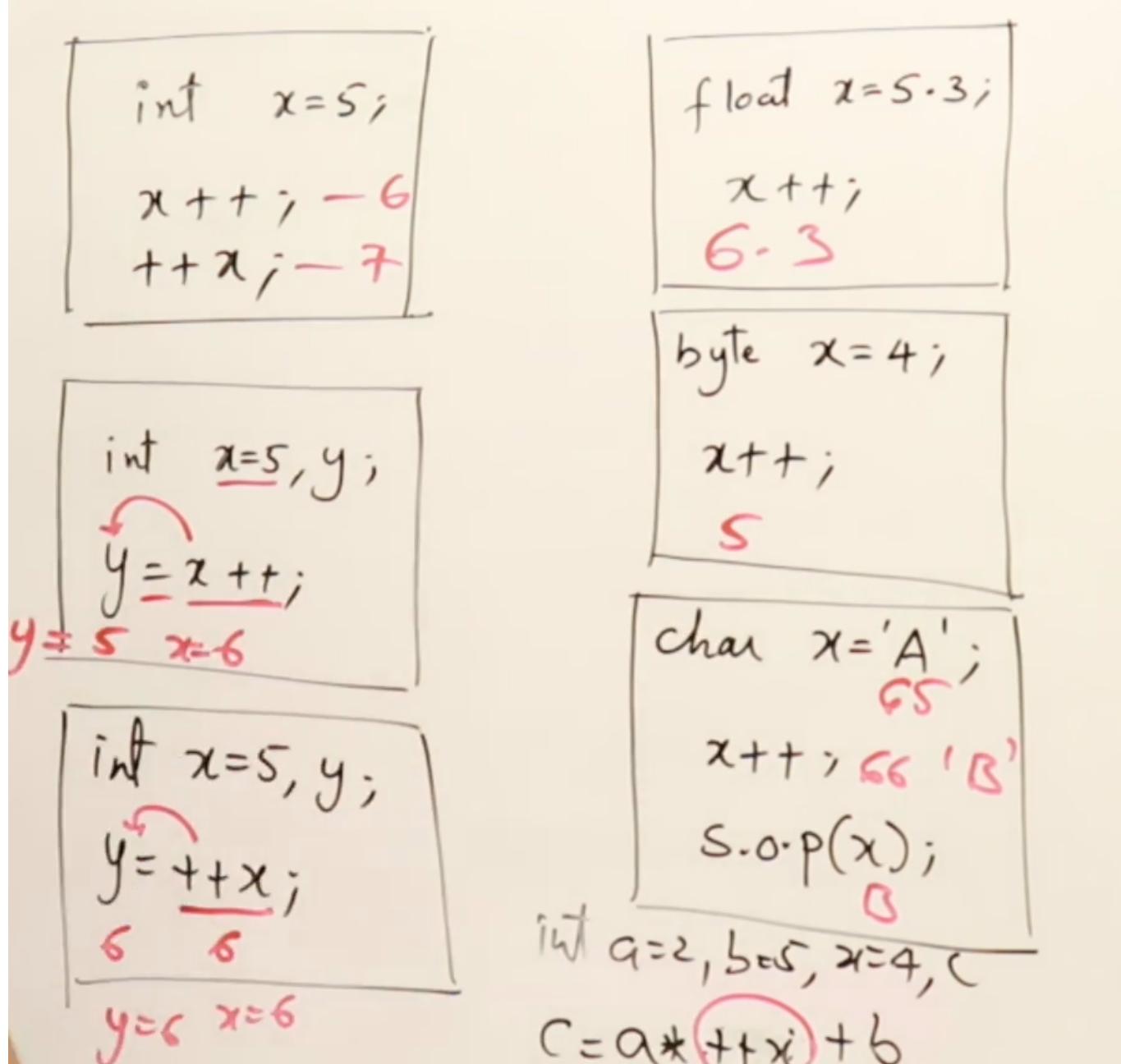
int a, b, c;  
double r1, r2;  
read

Math.sqrt

r1  
r2

a, b, c

## INCREMENT AND DECREMENT OPERATORS



increment and decrement can be performed on characters

```
class Main{
    public static void main(String args[]){
        char c = 'A';
        System.out.println(c);
        System.out.println(++c);
        System.out.println(c);
    }
}
```

when arithmetic operations performed on byte the result will be int when arithmetic operations performed on short the result will be int

```

public static void main(String[] args)
{
    byte b=5;
    incompatible types: possible lossy conversion from int to byte
    (Alt-Enter shows hints)
    b=b+1;

    System.out.println(b);
}

```

## 36. Bitwise Operator - AND, OR and XOR

| Bitwise Operators       |          |                |
|-------------------------|----------|----------------|
| AND                     | $\wedge$ | $\&$           |
| OR                      | $\vee$   | $\mid$         |
| NOT                     | $\neg$   | $\sim$         |
| XOR                     | $\oplus$ | $\wedge\wedge$ |
| RIGHT SHIFT             | $\gg$    |                |
| UNSIGNED<br>RIGHT SHIFT | $\gg\gg$ |                |
| LEFT SHIFT              | $\ll$    |                |

$\&$

| A | B | $A \& B$ |
|---|---|----------|
| 0 | 0 | 0        |
| 0 | 1 | 0        |
| 1 | 0 | 0        |
| 1 | 1 | 1        |

$\mid$

| A | B | $A \mid B$ |
|---|---|------------|
| 0 | 0 | 0          |
| 0 | 1 | 1          |
| 1 | 0 | 1          |
| 1 | 1 | 1          |

$\wedge\wedge$

| A | B | $A \wedge\wedge B$ |
|---|---|--------------------|
| 0 | 0 | 0                  |
| 0 | 1 | 0                  |
| 1 | 0 | 0                  |
| 1 | 1 | 0                  |

## Bitwise AND

## Bitwise Operators

```
int x=10, y=6, z;
```

$$x \rightarrow 00001010$$

$y \rightarrow 00000110$

$$z = x \& y \quad 0\ 0\ 0\ 0\ 0\ 0\ 1\ \underline{0}$$

```
1 package bitwisedemo;  
2  
3  
4 public class BitwiseDemo  
5 {  
6  
7     public static void main(String[] args)  
8     {  
9         int x=0b1010;  
10        int y=0b0110;  
11        int z;  
12  
13        z=x&y;  
14  
15        System.out.println(z);  
16  
17    }  
18}
```

```
Output - BitwiseDemo (run)

deps-jar:
Updating property file: /Users/abdulbari/NetBe
Compiling 1 source file to /Users/abdulbari/Ne
compile:
run:
2
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Bitwise OR

$$\begin{array}{r}
 x \rightarrow 00001010 \\
 y \rightarrow 00000110 \\
 \hline
 z = x \setminus y \quad 00001110 \\
 \qquad\qquad\qquad \underline{8421}
 \end{array}$$

```
package bitwisedemo;

public class BitwiseDemo
{
    public static void main(String[] args)
    {
        int x=0b1010;
        int y=0b0110;
        int z;

        z=x|y;

        System.out.println(z);
    }
}
```

Output - BitwiseDemo (run)

```
deps-jar:
Updating property file: /Users/abdulbari/Ne...
Compiling 1 source file to /Users/abdulbari/
compile:
run:
14
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Bitwise XOR

$$\begin{array}{r} x \rightarrow 00001010 \\ y \rightarrow 00000110 \\ \hline = x \wedge y & 00001100 \\ & \frac{8\ 4\ 2}{\phantom{8}\phantom{4}\phantom{2}\phantom{1}} \end{array}$$

$$z = 12$$

```
package bitwisedemo;

public class BitwiseDemo
{
    public static void main(String[] args)
    {
        int x=0b1010;
        int y=0b0110;
        int z;

        z=x^y;

        System.out.println(z);
    }
}
```

Output – BitwiseDemo (run)

```
deps-jar:
Updating property file: /Users/abdulbari/Ne
Compiling 1 source file to /Users/abdulbari
compile:
run:
12
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Bitwise Left Shift

int  $x=10, y=6, z;$

$x \rightarrow$    
 $y = x \ll 1$      $0\ 0\ 0\ 1\ 0\ 1\ 0$   
                         $\overline{16\ 8\ 4\ 2\ 1}$

$z = 20$

$n \times 2^k$

```
package bitwisedemo;

public class BitwiseDemo
{
    public static void main(String[] args)
    {
        int x=0b1;
        int y;
        y=x<<1;
        System.out.println(y);
    }
}
```

Output - BitwiseDemo (run)

deps-jar:

Updating property file: /Users/abdulbari/N

Compiling 1 source file to /Users/abdulbar

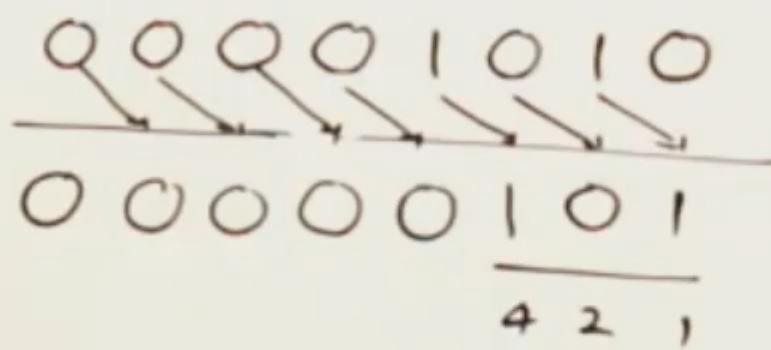
compile:

run:

2

BUILD SUCCESSFUL (total time: 0 seconds)

## Bitwise Right Shift

$x \rightarrow$    
 $y = x \gg 1$

$x \gg k$        $y = s$

$$\frac{n}{2^k}$$

```
1 package bitwisedemo;
2
3 public class BitwiseDemo
4 {
5
6     public static void main(String[] args)
7     {
8         int x=0b1000;
9
10        int y;
11
12        y=x>>2;
13
14        System.out.println(y);
15    }
16}
17
18
19 Output - BitwiseDemo (run)
20
21 deps-jar:
22 Updating property file: /Users/abdulbari/NetBeansProjects/BitwiseDemo/nbproject/ant-build.xml
23 Compiling 1 source file to /Users/abdulbari/NetBeansProjects/BitwiseDemo/bin
24 compile:
25 run:
26 2
27 BUILD SUCCESSFUL (total time: 0 seconds)
```

```
package bitwisedemo;

public class BitwiseDemo
{

    public static void main(String[] args)
    {
        int x=-0b1010;

        int y;

        y=x>>1;

        System.out.println(y);

    }
}
```

Output - BitwiseDemo (run)

```
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/BitwiseDemo/nbproject/properties.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/BitwiseDemo/bin
compile:
run:
-5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bitwise unsigned Right Shift

int  $x=10, y=6, z;$

int  $x=-10;$

$10 \rightarrow 00001010$

1's comp    11110101

2's comp     $\frac{+1}{11110110}$   
 $-10 \rightarrow 11110110$

$x \rightarrow 11110110$

$x \gg 1 \rightarrow 10111011$

$x \ggg 1 \rightarrow 01111011 \quad 123$   
64 32 16 8 4 2 1

NOTE : minor mistake in the above image

```
1 package bitwisedemo;  
2  
3  
4 public class BitwiseDemo  
5 {  
6  
7     public static void main(String[] args)  
8     {  
9         int x=-0b1010;  
10  
11         int y;  
12  
13         y=x>>>1;  
14  
15         System.out.println(y);  
16     }  
17 }
```

Output - BitwiseDemo (run)

```
deps-jar:  
Updating property file: /Users/abdulbari/NetBe...  
Compiling 1 source file to /Users/abdulbari/Net...  
compile:  
run:  
2147483643  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1 package bitwisedemo;
2
3 public class BitwiseDemo
4 {
5
6     public static void main(String[] args)
7     {
8         int x=-0b1010;
9
10        int y;
11
12        y=x>>1;
13
14        System.out.println(String.format("%s",Integer.toBinaryString(y)));
15    }
16
17
18 Output - BitwiseDemo (run)
19
20 deps-jar:
21 Updating property file: /Users/abdulbari/NetBeansProjects/BitwiseDemo/compile/
22 Compiling 1 source file to /Users/abdulbari/NetBeansProjects/BitwiseDemo/compile/
23 compile:
24 run:
25 11111111111111111111111111111111011
26 BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1 package bitwisedemo;
2
3 public class BitwiseDemo
4 {
5
6     public static void main(String[] args)
7     {
8         int x=-0b1010;
9
10        int y;
11
12        y=x>>>1;
13
14        System.out.println(String.format("%s",Integer.toBinaryString(x)));
15        System.out.println(String.format("%32s",Integer.toBinaryString(y)));
16
17
18 Output - BitwiseDemo (run)
19
20 Updating property file: /Users/abdulbari/NetBeansProjects/BitwiseDemo/compile/
21 Compiling 1 source file to /Users/abdulbari/NetBeansProjects/BitwiseDemo/compile/
22 compile:
23 run:
24 111111111111111111111111111111110110
25 111111111111111111111111111111110111
```

## Bitwise NOT

int x=10, y=6, z;

$x \rightarrow 00001010$   
 $y = \sim x \rightarrow \begin{array}{r} 11110101 \\ \hline -11 \end{array}$

$z \rightarrow +1110101$   
 $1's \rightarrow 00001010 \quad //$   
 $2's \rightarrow \begin{array}{r} +1 \\ \hline 00001011 \\ z + 2^1 \end{array}$

```
package bitwisedemo;

public class BitwiseDemo
{

    public static void main(String[] args)
    {
        int x=0b1010;
        int y;
        y=~x;
        System.out.println(String.format("%s",Integer.toBinaryString(x)));
        System.out.println(String.format("%32s",Integer.toBinaryString(y)));
    }
}
```

Output - BitwiseDemo (run)

```
Updating property file: /Users/abdulbari/NetBeansProjects/BitwiseDemo/project.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/BitwiseDemo/bin
compile:
run:
1010
111111111111111111111111111111110101
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Merging and Masking

Merging means combining the bits of two numbers  
Masking means extracting the bits of the number  
MASKING |

## Merging and Masking

byte a=0;

$$\begin{array}{r}
 \text{a} \rightarrow \begin{array}{ccccccccc} & \checkmark & & & & & & & \\ & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \text{b}=8 \rightarrow & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \\
 \hline
 \text{a} = a \mid b \quad \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}
 \end{array}$$

|        |         |
|--------|---------|
| 0      | 1       |
| ↑      | ↑       |
| no     | yes     |
| false  | true    |
| absent | present |

## Merging and Masking

byte a=0;

$$\begin{array}{r}
 \text{a}=8 \rightarrow \begin{array}{ccccccccc} & & 5 & 4 & 3 & 2 & 1 & 0 \\ & & 0 & \otimes & 0 & 0 & 1 & 0 & 0 \end{array} \\
 \text{b}=64 \rightarrow \begin{array}{ccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\
 \hline
 \text{a} = a \mid b \quad \begin{array}{ccccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}
 \end{array}$$

|        |         |
|--------|---------|
| 0      | 1       |
| ↑      | ↑       |
| no     | yes     |
| false  | true    |
| absent | present |

MERGING &

## Merging and Masking

byte a=0;

$$\begin{array}{r}
 \text{a}=8 \rightarrow \begin{array}{ccccccccc} & & & 3 & & & & & \\ & & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \text{b}=8^{16} \rightarrow & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \\
 \hline
 \text{a} \& b \quad \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & a & 0 & 0 & 0 \end{array} \\
 \hline
 > 0 \quad = 0
 \end{array}$$

|        |         |
|--------|---------|
| 0      | 1       |
| ↑      | ↑       |
| no     | yes     |
| false  | true    |
| absent | present |

```

class Main{
    public static void main(String args[]){
        int a = 0x0F;
        int b = 0x0A;
        System.out.println(a&b);
        System.out.println(a|b);
        System.out.println(a^b);
        System.out.println(~a);
        System.out.println(a<<2);
        System.out.println(a>>2);
        System.out.println(a>>>2);
    }
}

```

EG : SUPPOSE TO STORE THE 0 TO 10 NUMBER byte data type is enough can we able to store in less space (through masking and merging) 0 to 10 to store 4 bits are enough storing 5 and 9 in 1 byte

Merging and Masking

byte  $a=0$ ;  $0, 1, 2, \dots, 10$

$a \rightarrow$        $b=5$        $a = a' b$

|  |                                                                                                               |                                               |
|--|---------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
|  | $\begin{array}{cccccccccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$ |                                               |
|  |                                                                                                               | $\frac{1010}{\text{4-bits}}$<br><u>nibble</u> |

like the below we can store

### Merging and Masking

byte a=0;       $0, 1, 2, \dots, 10$

$a \rightarrow$        $b = 9 \rightarrow$        $b = b \ll 4$        $a = a \mid b$

|                                                                                                                                                                                                                              |                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| $\begin{array}{cccccccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$ | $\frac{1010}{4\text{-bits}}$<br><b>nibble</b> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|

how to retrieve the 5 and 9 from the 1 byte by using &

### Swapping

$a = 9$        $b = 12$

$a = a \wedge b$   
 $b = a \wedge b$   
 $\underline{a = a \wedge b}$

$a \wedge b$        $9 \quad 12$

$a = 00000100 +$   
 $b = 00001001$   
 $\hline 00001100$

storing 2 numbers in single byte

```
class Bitwise
{
    public static void main(String args[])
    {
        byte a=9,b=12;

        byte c;

        c=(byte)(a<<4);
        c=(byte)(c|b);

        System.out.println((c&0b11110000)>>4);
        System.out.println((c&0b00001111));

    }

}

class Bitwise
{
    public static void main(String args[])
    {

        byte c;

        c=(byte)(9<<4);
        c=(byte)(c|12);

        System.out.println((c&0b11110000)>>4);
        System.out.println((c&0b00001111));

    }

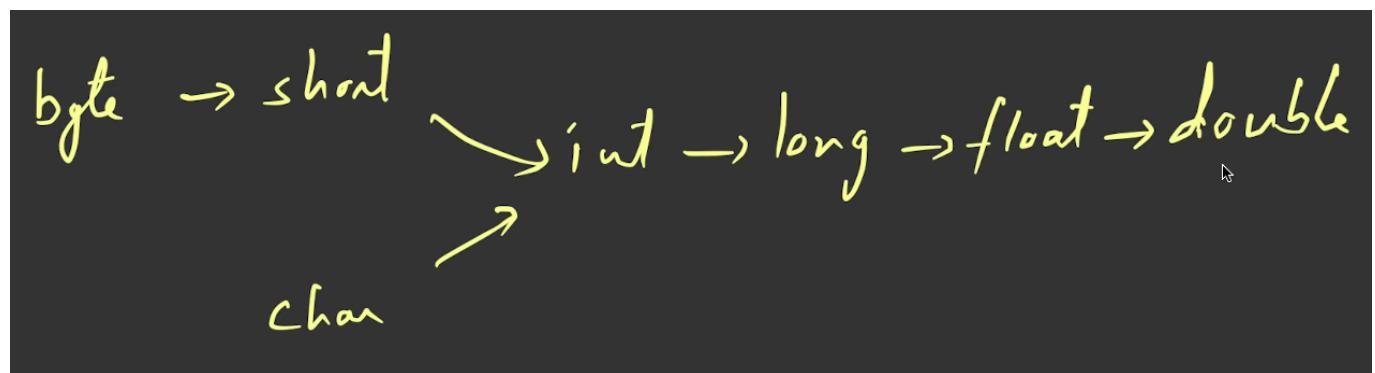
}
```

## 41. Widening and Narrowing

Widening is the process of converting the data from the lower data type to the higher data type  
Narrowing is the process of converting the data from the higher data type to the lower data type

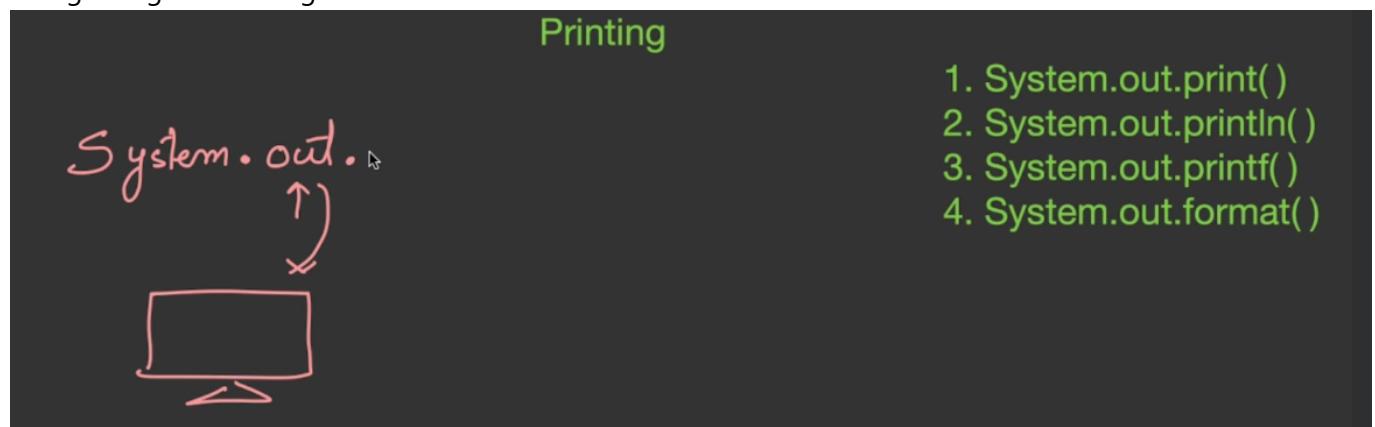
character we can assign char and int

|                                |         | left side ↗ = |       |     |      |       |        |      |         |
|--------------------------------|---------|---------------|-------|-----|------|-------|--------|------|---------|
|                                |         | byte          | short | int | long | float | double | char | boolean |
| Source<br>right side<br>↓<br>= | byte    | ✓             | ✓     | ✓   | ✓    | ✓     | ✓      | ✗    | ✗       |
|                                | short   | ✗             | ✓     | ✓   | ✓    | ✓     | ✓      | ✗    | ?       |
|                                | int     | ✗             | ✗     | ✓   | ✓    | ✓     | ✓      | ✓    | ✗       |
|                                | long    | ✗             | ✗     | ✗   | ✓    | ✓     | ✓      | ✗    | ✗       |
|                                | float   | ✗             | ✗     | ✗   | ✗    | ✓     | ✓      | ✗    | ✗       |
|                                | double  | ✗             | ✗     | ✗   | ✗    | ✗     | ✓      | ✗    | ✗       |
|                                | char    | ✗             | ✗     | ✓   | ✓    | ✓     | ✓      | ✓    | ✗       |
|                                | boolean | ✗             | ✗     | ✗   | ✗    | ✗     | ✗      | ✗    | ✓       |



## Section 6: Strings

String StringBuffer StringBuilder



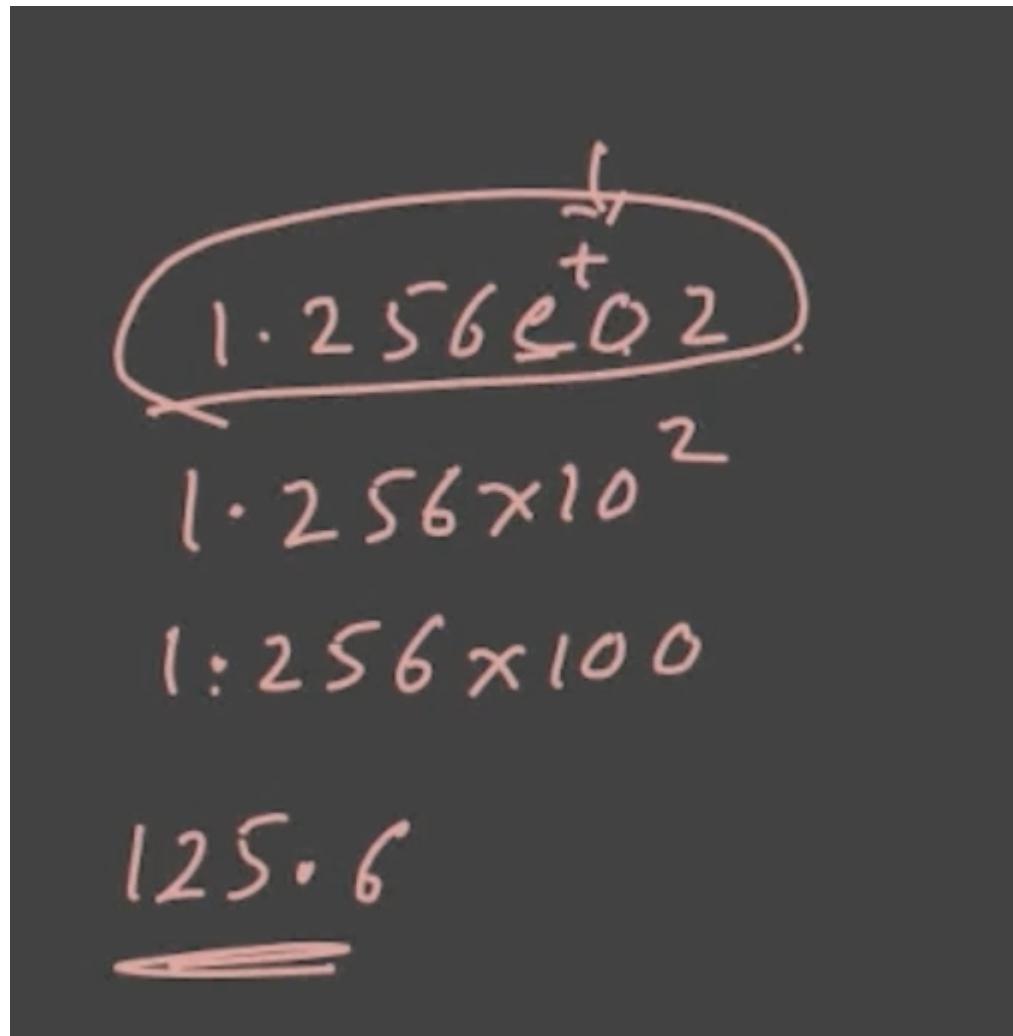
out is the static object of the PrintStream class Method with same name and different parameter list =

method overloading print() is overloaded method

The screenshot shows an IDE interface with a Java file named `Printing.java` open. The code defines a class `Print` with a static method `System.out.p`. A tooltip for the `print` method of `java.io.PrintStream` is displayed, explaining its behavior for printing boolean values. Below the tooltip, a list of `print` methods is shown, each with its signature and return type:

- `print(Object obj)` void
- `print(String s)` void
- `print(boolean b)` void
- `print(char c)` void
- `print(char[] s)` void
- `print(double d)` void
- `print(float f)` void
- `print(int i)` void
- `print(long l)` void
- `printf(String format, Object... args)` PrintStream
- `printf(Locale l, String format, Object... args)` PrintStream
- `println()` void

At the bottom right of the tooltip, there is a note: "Instance Members; Press 'Ctrl+SPACE' Again for All Items".



```
int x=10;
float y=0.0012f;
char z='A';

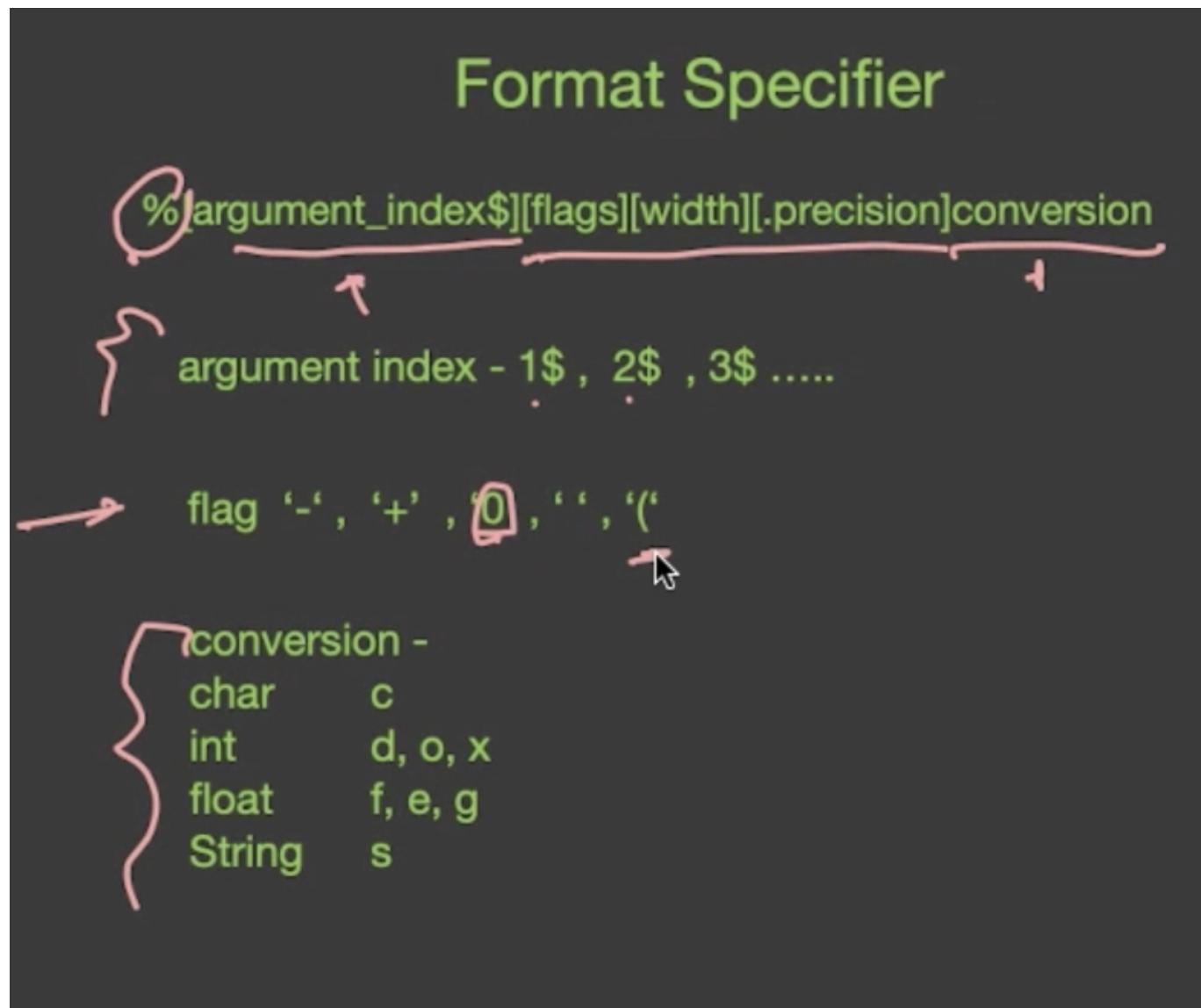
ant -f /Users/abdulbari/NetBeansProject
init:
Deleting: /Users/abdulbari/NetBeansProj
deps-jar:
Updating property file: /Users/abdulbar
Compiling 1 source file to /Users/abdul
compile:
run:
Hello 1.200000e-03
BUILD SUCCESSFUL Total time: 0 seconds

1.2e-3
1.2 × 10-3
1.2 ×  $\frac{1}{1000}$ 
.012
```

order of taking arguments

```
System.out.printf("%1$d %1$d %1$d",x);
```

Format Specifier



float width and precesion

```
float a=123.45f;
```

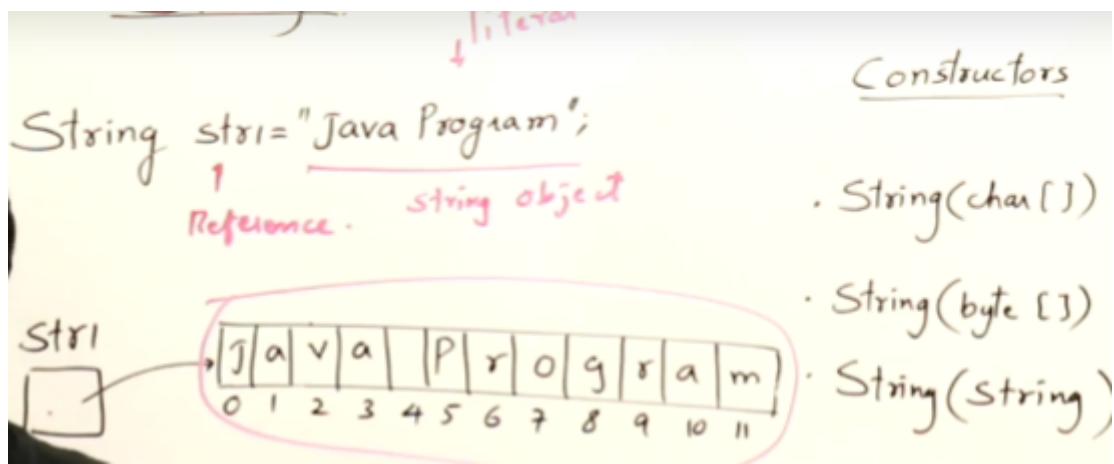
```
System.out.printf("%6.2f", a);
```

## 45. String Object

String is a builtin class available in the `java.lang` package and it is immutable. used like a data type.

String is a collection of characters.

```
String str1 = "Java Programming"; //String literal //string object
```



```
char c[] = {'a','b','c','d'}; String s = new String(c); //converted to string
```

```
byte b[] = {65,66,67,68}; String s1 = new String(b); //converted to string0
```

```
String s2 = new String("Java Programming"); // 2 objects are created
```

```
String s3 = "Java Programming"; //string literal is created in the string pool
```

java mains the string pool for the string literals

```
| javap java.lang.String
```

```
compare the references String str1 = "Java"; String str2 = "Java"; String str3 = new String("Java"); String str4 = new String("Java"); System.out.println(str1==str2); //true System.out.println(str1==str3); //false System.out.println(str3==str4); //false
```

## 47. String Methods#1

- int length()

- String toLowerCase()

- String toUpperCase()

- String trim()

- String substring(int begin)

- String substring(int begin, int end)

String replace(char old, char New)

boolean startsWith(String s)

boolean endsWith(String s)

char charAt(int index)

int indexOf(String s)

int lastIndexOf(String s)

boolean equals(String s)

boolean equalsIgnoreCase(String s)

int compareTo(String s)

String valueOf(int i)



.toUpperCase() won't changes the original string new string object is created and reference is changed.

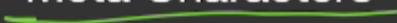
### 53. Regular Expressions

## Matching Symbols



| Regular Expression | Description                |
|--------------------|----------------------------|
| .                  | Any character              |
| [abc]              | Exactly given letters      |
| [abc][vz]          | Either first or second set |
| [^abc]             | Except abc                 |
| [a-zA-Z]           | a-z or A-Z                 |
| A B                | A or B                     |
| XZ                 | Exactly XZ                 |

## Meta Characters



| Regular Expression | Description               |
|--------------------|---------------------------|
| \d                 | Digits                    |
| \D                 | Not digits                |
| \s                 | Space                     |
| \S                 | Not space                 |
| \w                 | Alphabets or digit        |
| \W                 | Neither alphabet or digit |

**Quantifiers**

| Regular Expression | Description          |
|--------------------|----------------------|
| *                  | 0 or more time       |
| +                  | One or more          |
| ?                  | 0 or 1 time          |
| {X}                | X times              |
| {X,Y}              | Between X and Y time |

$a^*$       abde  
 $a^+$       a-7P<sup>nd</sup>  
 $a?$       abc aabbcc accbba  
 $\{a\}$       abm  
 $\{a,b\}$       X  
 $\{a,b\}^*$       apple  
 $\{a,b\}^{\{5\}}$       apmbc KPKPP<sup>r</sup>

Udemy

Find if the email id is on gmail

Find username and domain name form email

String str = "programmer@gmail.com";  
i = indexOf("@")  
uname = substring(0, i)  
domain = " " (i+1, str.length())  
compareTo  
equals

```
package scstring;

public class SCString
{
    public static void main(String[] args)
    {
        String str="programmer@gmail.com";

        int i=str.indexOf("@");
        String uname=str.substring(0,i);
        String domain=str.substring(i+1, str.length());

        System.out.println("Username : "+uname);
        System.out.println("Domain :" +domain);

        System.out.println(domain.startsWith("gmail"));
    }
}
```

```
public class SCString
{
    public static void main(String[] args)
    {
        String str="programmer@hotmail.com";
        int i=str.indexOf("@");
        String uname=str.substring(0,i);
        String domain=str.substring(i+1, str.length());
        System.out.println("Username : "+uname);
        System.out.println("Domain :" +domain);

        int j=domain.indexOf(".");
        String name=domain.substring(0,j);
        System.out.println(name.equals("gmail")));
    }
}
```

Find if a Number is Binary or not  
Find is a Number is Hexa-Decimal or not  
Find is the data in Date format (dd/mm/yyyy)

```
int b = 10110001 ;  
String str = b + ""; valueOf(b)  
System.out.println(str.matches("[01]*"));  
System.out.println(str.matches("[0-9A-F]+"));
```

Find if a Number is Binary or not  
Find is a Number is Hexa-Decimal or not  
Find is the data in Date format (dd/mm/yyyy)

$[0-9A-F]^+$

Find if a Number is Binary or not

Find is a Number is Hexa-Decimal or not

Find is the data in Date format (dd/mm/yyyy)

String d = "01/12/2000";

"[0-9][0-9]/[0-9][0-9]/[0-9]{4}"

- ✓ Remove Special characters from a string
  - ✓ Remove extra spaces from string
- Find number of words in a String

String str="abc de fgh ijk";

str.replaceAll("\s+", " ");

```
package scstring;

public class SCString
{
    public static void main(String[] args)
    {
        String str=" abc def gh ijk ";
        System.out.println(str.replaceAll("\s+", " ").trim());
    }
}
```

sp<sup>l/t</sup> ("ws") ;

find number of words

```

public class SCString
{
    public static void main(String[] args)
    {
        String str="      abc      def      gh   ijk      ";
        str=str.replaceAll("\\s+", " ").trim();
        String words[]=str.split("\\s");
        System.out.println(words.length);
    }
}

```

## Section 7 Control Statements

---

### 58.Relational and logical operators

Relational Operators

<  
≤  
>  
≥  
==  
!=

Logical operators

&&  
||  
!



| A | B | A&&B | A  B | !A |
|---|---|------|------|----|
| T | T | T    | T    | F  |
| T | F | F    | T    | F  |
| F | T | F    | T    | T  |
| F | F | F    | F    | T  |

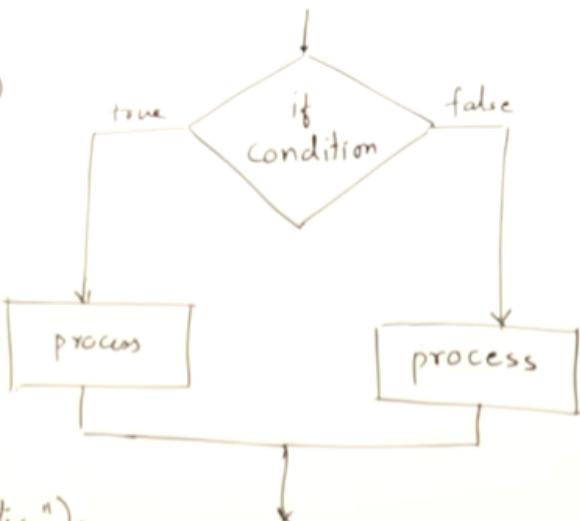
all this returns boolean value

### 59. if else statement

## Conditional Statements

## class Test

```
{ public static void main(String arg[])
{
    int x=-5;
    if( x >= 0 )
    {
        System.out.println("Positive");
    }
    else
    {
        System.out.println("Negative");
    }
}
```



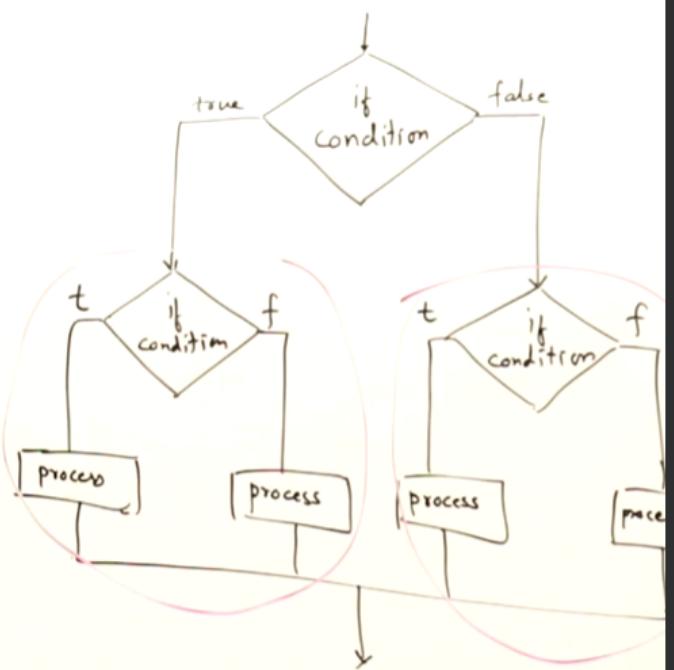
## Conditional Statements

## Nested if

```
if (condition)  
{
```

```
if (condition)
{
    =
}
else
{
    =
}
```

{  
  | if (condition)  
  | =  
  | {  
  | | dx  
  | | =  
  | }  
  | }  
}|

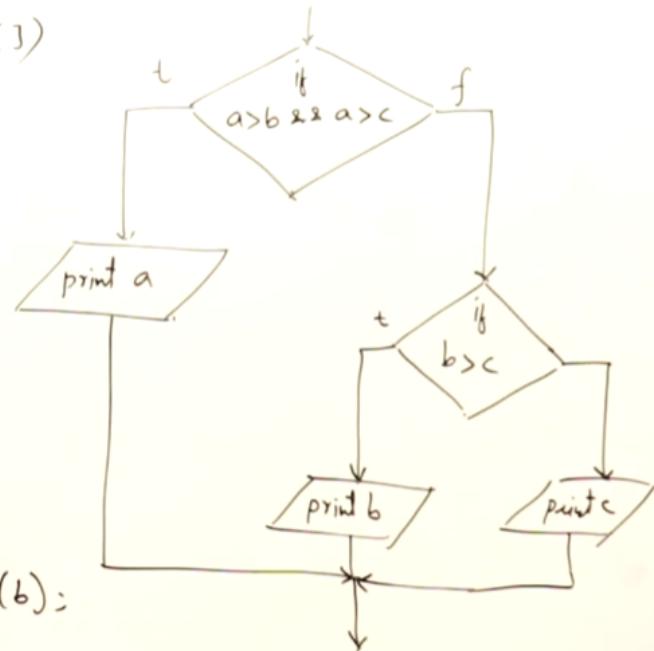


## Conditional Statements

```

class Test
{
    public static void main(String args[])
    {
        int a=5, b=6, c=10;
        if(a>b && a>c)
            System.out.println(a);
        else
            if(b>c)
                System.out.println(b);
            else
                System.out.println(c);
    }
}

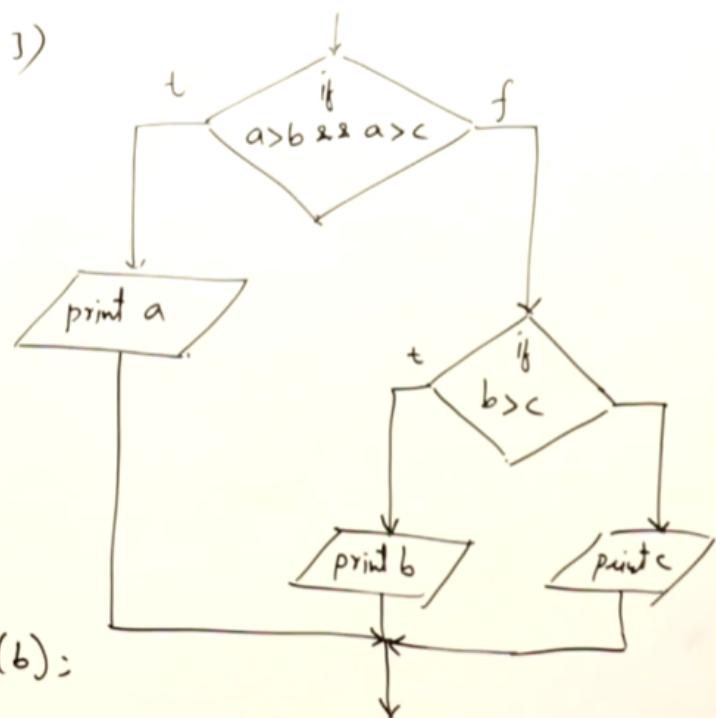
```



```

public static void main(String args[])
{
    int a=5, b=6, c=10;
    if(a>b && a>c)
        System.out.println(a);
    else if(b>c)
        System.out.println(b);
    else
        System.out.println(c);
}

```



## else Ladder

```
if (condition)
{
    =
}
else if (condition)
{
    =
}
else if (condition)
{
    =
}
else
{
    =
}
```

- ✓ Find a number is odd or even
- Find person is young or not young.
- Find grades for given marks

*int n;*

*n=5*    *n=13*    *n=16*

*n=5 ←*

2)  $\frac{16}{8}$     2)  $\frac{13}{12}$

0 ←      1

- ## Find radix of a number given in a string
- ## Find a given year is a leap year

<https://www.codingninjas.com/studio/library/leap-year-program-in-java>

<https://www.geeksforgeeks.org/java-program-to-find-if-a-given-year-is-a-leap-year/>

```
if(num.matches("[01]+"))
{
    System.out.println("Binary Radix=2");
}
else if(num.matches("[0-7]+"))
{
    System.out.println("Octal Radix=8");
}
else if(num.matches("[0-9]+"))
{
    System.out.println("Decimal| Radix=8");
}
```

✓ Display name of a day based on number

Find type of website and the protocol used

day=1  
Monday

switch case

### Switch.. case

```
int day=
switch(day)
{
    case 1: s.o.p("MON");
    break;
    case 2: s.o.p("TUE");
    break;
    case 3: s.o.p("WED");
    break;
    :
    default : s.o.p("Invalid Day");
}
```

```
int day=
if(day==1)
    s.o.p("MON");
else if(day==2)
    s.o.p("TUE");
else if(day==3)
    s.o.p("WED");
:
:
```

- ✓ 1. Display name of a day based on number  
2. Display name of a month based on number  
3. Display type of website

### Make a Menu Driven Program for Arithmetic Operations

Menu

1. ADD
2. SUB
3. MUL
4. DIV

Enter 2 no: 10 5.

Enter option in words ADD

option = nextLine()

```
switch(option)
{
    case "ADD": _____
        break;
    case "SUB": _____
        break;
    _____
    _____
}
```

#### CAREFULL STUDY

```
System.out.println("Enter 2 Numbers");
int x=sc.nextInt();
int y=sc.nextInt();
sc.nextLine();
System.out.println("Enter Option in Words ");
String option=sc.nextLine();

switch(option)
{
    case "ADD":
```

```
System.out.println("Enter Option in Words ");
String option=sc.nextLine();
option=option.toUpperCase();
switch(option)
{
    case "ADD":
```

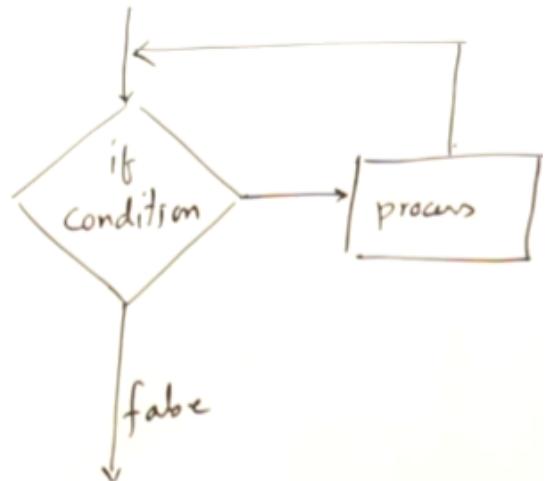
The "switch" selection structure must end with the default case.

Which section is mandatory in for loop ?

quiz

## Loops

1. while Loop
2. do..while loop
3. for Loop
4. for each loop

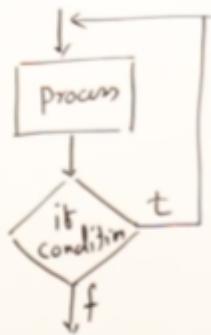
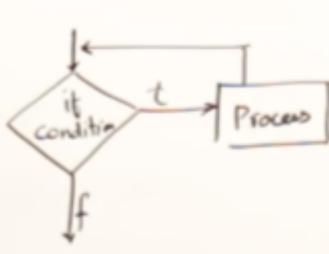


## Section 8: Loops

---

GCD, HCF and LCM

## Loops



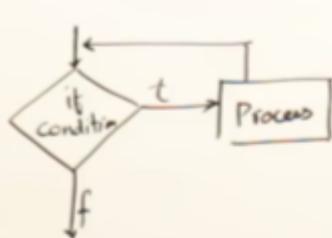
```
while (condition)  
{  
    }  
    }  
    }  
}
```

```
{ do  
        
        
  } while(condition);
```

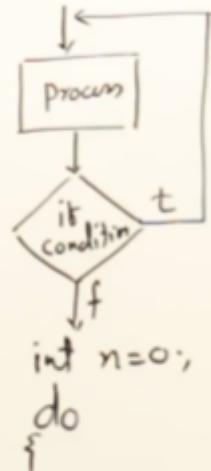
## (pre test loop) while loop

executes as long as the condition is true

do while is (post test loop) do while loop executes atleast once and then checks the condition



```
int n=0;  
{ while( n!=0 )  
    == process
```



PROCED



## Loops.

```
int i=1, n=100;
while(i < n)
{
    s.o.p(i);
    i=i*2;
}
```

```
int i=1, n=100;
do
{
    s.o.p(i);
    i=i*2;
} while(i < n);
```

| s.o.p(i) | i = i * 2 |
|----------|-----------|
| 1        | 2         |
| 2        | 4         |
| 4        | 8         |
| 8        | 16        |
| 16       | 32        |
| 32       | 64        |
| 64       | 128       |

| s.o.p(i) | i = i * 2 |
|----------|-----------|
| 1        | 2         |
| 2        | 4         |
| 4        | 8         |
| 8        | 16        |
| 16       | 32        |
| 32       | 64        |
| 64       | 128       |

while(true){ //infinite loop } System.out.println("Hello"); //unreachable code //compilation error

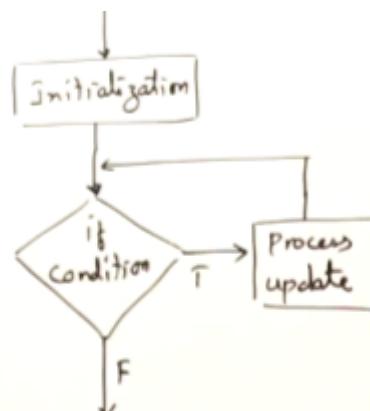
if(true){ System.out.println("Hello"); } else{ System.out.println("Bye"); } // unreachable code //compilation warning.

for loop is the counter controlled loop

```
for(initialization; condition; updation)
```

≡

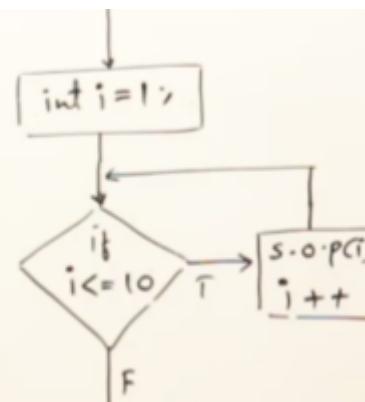
}



```

for(int i=1 ; i<=10 ; i++)
{
    s.o.p(i);
}

```



print in reverse order

## 72.sc : Factorial

- ✓ 1. Display Multiplication Table
- 2. Find Sum of n Numbers
- 3. Factorial of a Number

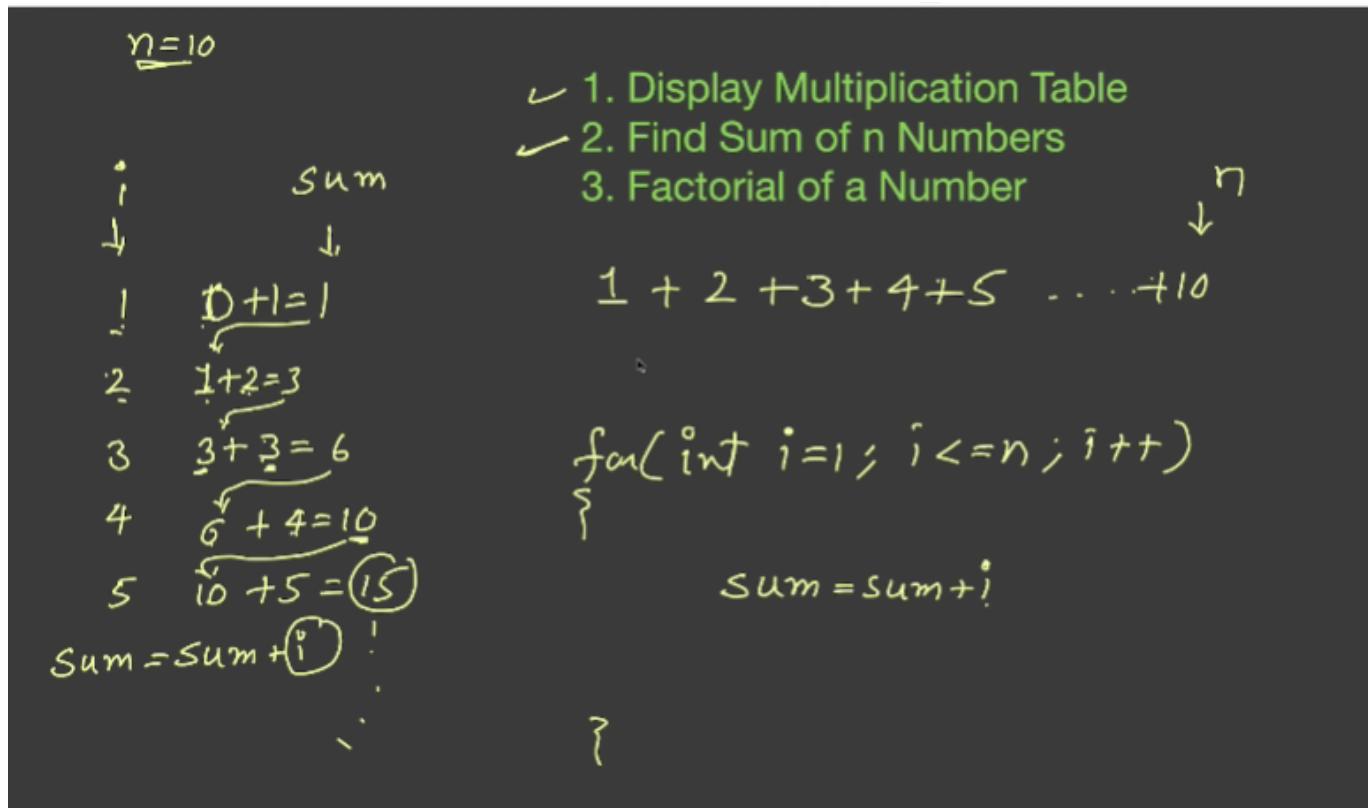
1.

$$\begin{array}{c} \textcircled{n=5} \\ \cdot \\ \begin{array}{ccc} n & i & n*i \\ \downarrow & \downarrow & \\ \textcircled{5 \times 1 = 5} \\ 5 \times 2 = 10 \\ 5 \times 3 = 15 \\ 5 \times 4 = 20 \\ 5 \times 5 = 25 \\ \vdots \\ 5 \times 10 = 50 \end{array} \end{array}$$

- ✓ 1. Display Multiplication Table
- 2. Find Sum of n Numbers
- 3. Factorial of a Number

$\text{for}(i=1; i \leq 10; i++)$   
 $\{$   
 $\text{System.out.println}(n + " \times " + i + "=" + n*i);$   
 $\}$

2.



3.

- ✓ 1. Display Multiplication Table
- ✓ 2. Find Sum of n Numbers
- ✓ 3. Factorial of a Number

$$n=5!$$

1  ~~$fact \leftarrow 1$~~   $1 * 2 * 3 * 4 * 5 = 120$   
 2  ~~$fact \leftarrow fact * i$~~   $= 120$   
 3  ~~$fact \leftarrow fact * 2$~~   $1 * 2 = 2$   
 4  ~~$fact \leftarrow fact * 3$~~   $2 * 3 = 6$   
 5  ~~$fact \leftarrow fact * 4$~~   $6 * 4 = 24$   
 6  ~~$fact \leftarrow fact * 5$~~   $24 * 5 = 120$

```

for(int i=1; i<=n; i++)
{
    fact=fact*i;
}
    
```

```

public class SCLoop1
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter a Number");
        int n=sc.nextInt();

        long fact=1;

        for(int i=1;i<=n;i++)
        {
            fact=fact*i;
        }

        System.out.println("Factorial is "+fact);
    }
}
    
```

---

## 73.Armstrong

== sum of the cubes of the digits of the number is equal to the number itself

$$\begin{aligned}
 n &= 153 \\
 3^3 + 5^3 + 1^3 \\
 27 + 125 + 1 &= 153
 \end{aligned}$$

- ✓ 1. Display Digits of No.
- 2. Count Digits of a Number
- 3. Finding a number is Armstrong or not
- 4. Reverse a Number
- 5. Check a number is palindrome

1.

$$\begin{array}{ccc}
 n & r = n \% 10 & n/10 \\
 257 & 7 & 257/10 = 25 \\
 25 & 5 & 25/10 = 2 \\
 2 & 2 & 2/10 = 0 \\
 0 & &
 \end{array}$$

```

public static void main(String[] args)
{
    Scanner scan=new Scanner(System.in);

    System.out.println("Enter a Number");
    int n=scan.nextInt();

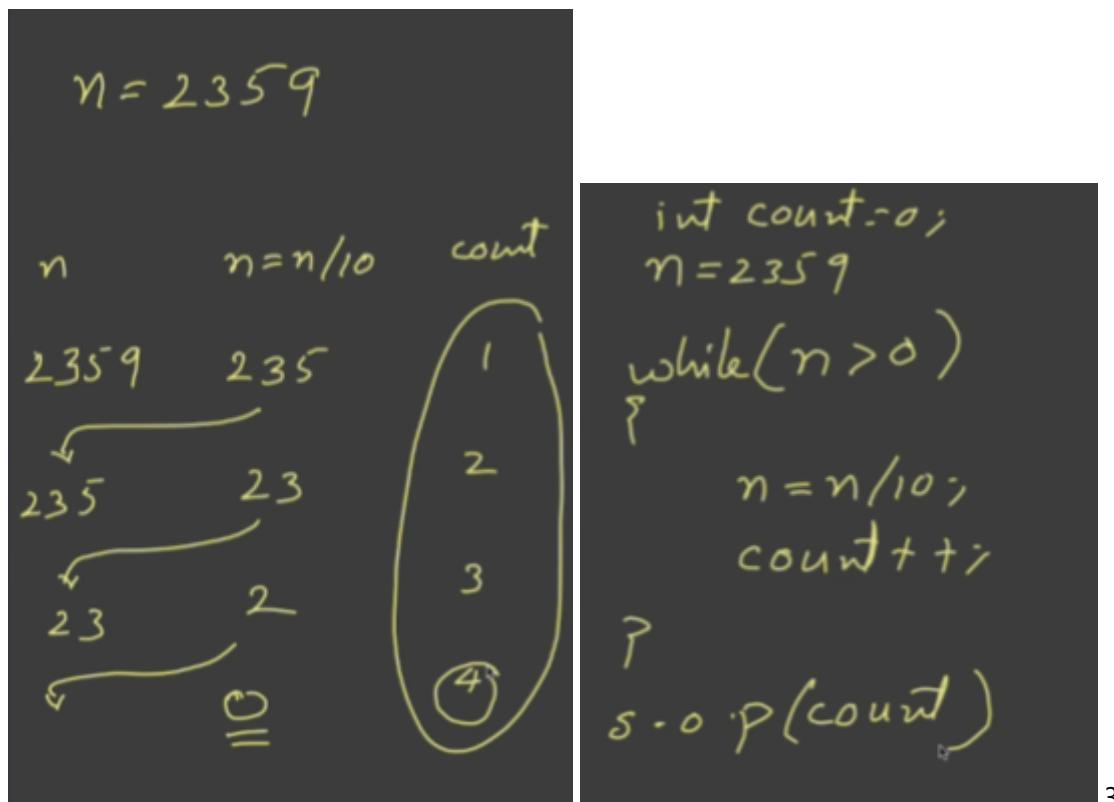
    int r;

    while(n>0)
    {
        r=n%10;
        n=n/10;

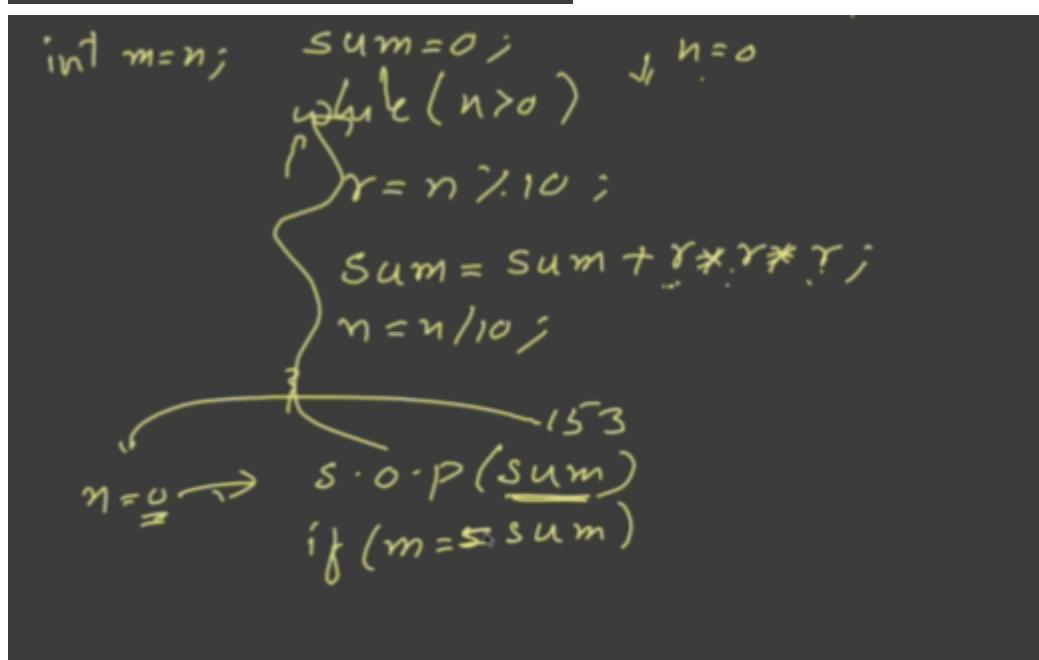
        System.out.println(r);
    }
}

```

2.



$$\begin{aligned}
 n &= 153 \\
 3^3 + 5^3 + 1^3 \\
 27 + 125 + 1 &= 153
 \end{aligned}$$



74. sc : palindrome number

4.

$$n = 237$$

$$\text{rev} = 732$$

- ✓ 1. Display Digits of No.
- ✓ 2. Count Digits of a Number
- ✓ 3. Finding a number is Armstrong or not
- ✓ 4. Reverse a Number
- ✓ 5. Check a number is palindrome

$$\begin{array}{llll}
 n & r = n \% 10 & \text{rev} & n = n / 10 \\
 237 & 7 & 7 & 23 \\
 23 & 3 & 7 \times 10 + 3 = 73 & 2 \\
 2 & 2 & \underline{73 \times 10 + 2 = 732} & 0
 \end{array}$$

$$n = 237$$

$$\text{rev} = 732$$

- ✓ 1. Display Digits of No.
- ✓ 2. Count Digits of a Number
- ✓ 3. Finding a number is Armstrong or not
- ✓ 4. Reverse a Number
- ✓ 5. Check a number is palindrome

$$\begin{array}{llll}
 n & r = n \% 10 & \text{rev} & n = n / 10 \\
 237 & 7 & \left\{ \begin{array}{l} 0 \times 10 + 7 = 7 \\ 7 \times 10 + 3 = 73 \\ \hline 73 \times 10 + 2 = 732 \end{array} \right. & 23 \\
 23 & 3 & & 2 \\
 2 & 2 & & 0
 \end{array}$$

```
rev=0

}

    rev=n%10;
    rev=rev*10+r;
    n=n/10;

    s.o.p(rev)
```

```
Scanner scan=new Scanner(System.in);

System.out.println("Enter a Number");
int n=scan.nextInt();

int rev=0,r;

while(n>0)
{
    r=n%10;
    rev=rev*10+r;
    n/=10;
}

System.out.println("Reverse Number "+rev);
```

5.

```

int m=n;
rev=0
while(n>0)
{
    r=n%10;
    rev=rev*10+r;
    n=n/10;
}
if(rev==m)
    s.o.p(✓)
else
    ...

```

75.

Display a number in words even with tailing 0

$n = 2 \underline{3} \underline{7}$        $n = 1 \underline{7} \underline{0} \underline{0}$

Two      Three      Seven      one      seven      three      zero      zero

$n = 2 \underline{3} \underline{7}$

~~rev = 7 3 2~~

$n = 1 \underline{7} \underline{0} \underline{0}$

$rev = (7 1)$

str  $\underline{\underline{str = 0 0 7 1}}$

$n = 1 \underline{7} \underline{0} \underline{0}$

$\left\{ \begin{array}{l} r = n \% 10; \\ n = n / 10; \\ str = str + r; \end{array} \right.$

```

    } -->
    r=n%10;
    n=n/10;
    str= str+r;
    for(i=str.length()-1;i-->)
    {
        c= str.charAt(i)
    }

```

use switch case

```

Scanner scan=new Scanner(System.in);

System.out.println("Enter a Number");
int n=scan.nextInt();

int r;
String str="";

while(n>0)
{
    r=n%10;
    n=n/10;
    str=str+r;

}
System.out.println(str);

```

CODE

```

char c;
for(int i=str.length()-1;i>=0;i--)
{
    c=str.charAt(i);
    switch(c)
    {
        case '0':System.out.print("Zero ");
                    break;
    }
}

```

(usage : above code is use full in certificates)

- ✓ 1. Display AP Series
- ✓ 2. Display GP Series
- ▷ 3. Display fibonacci series

1. ap = starting term and common difference

## Arithmetic Progression Series

$$\textcircled{5}, 10, 15, 20, \dots a=5 \quad d=15-10=5$$

$$3, 8, 13, 18, 23 \quad a=3 \quad d=13-8=5$$

$$7, 9, 11, 13, 15, \dots a=7 \quad d=2$$

↳

```
Scanner sc=new Scanner(System.in);
```

```
System.out.println("Program to print AP Series");
```

```
System.out.println("Enter a, d and n");
```

```
int a=sc.nextInt();
```

```
int d=sc.nextInt();
```

```
int n=sc.nextInt();
```

```
int term=a;
```

```
for(int i=0;i<n;i++)
```

```
{
```

```
    System.out.print(term+",");
```

```
    term=term+d;
```

```
}
```

2.

## Geometric Progression

### 3. Display fibonacci series

$$2, 6, 18, 54, \dots a=2 \quad r=\frac{18}{6}=3 \quad Ap: a + ad + a^2d + a^3d + \dots$$

$$5, 10, 20, 40, \dots a=5 \quad r=\frac{20}{10}=2$$

$$a + ar + ar^2 + ar^3 + \dots$$

↳

## Fibonacci Series

$\underbrace{0, 1}_{0^{\text{th}}}, \underbrace{1, 2}_{1^{\text{st}}}, 3, 5, 8, 13, 21$   
 $f_{\text{ib}}(0) = 0$   
 $f_{\text{ib}}(1) = 1$

$$f_{\text{ib}}(n) = f_{\text{ib}}(n-2) + f_{\text{ib}}(n-1)$$

$$n = 10$$

$$a = 0 \quad b = 1$$

$$c = a + b$$

$a$      $b$      $c = a + b$   
 $\underbrace{0, 1}_{a}, \underbrace{1, 2}_{b}, 3, 5, 8, 13, 21$

$$c = a + b$$

$$\begin{aligned} c &= a + b \\ a &= b \\ b &= c \end{aligned}$$

```

Scanner sc=new Scanner(System.in);

System.out.println("Program to Fibonacci Series");
System.out.println("Enter number of Terms");
int n=sc.nextInt();

int a=0,b=1,c;

System.out.print(a+" "+b+" ");
for(int i=0;i<n-2;i++)
{
    c=a+b;
    System.out.print(c+" ");
    a=b;
    b=c;
}
    
```

## 77. Nested Loops

## Nested Loops

```
for(int i=0; i<10; i++)
```

{

$i = 0, 1, 2, 3, 4, 5, \dots, 10$

$\xrightarrow{\text{linear}}$

linear

```
for(int i=0; i<5; i++)
```

```
    for(int j=0; j<5; j++)
```

$\equiv$

{

{

```
while(i<n)
```

$\equiv$

{

```
do
```

$\equiv$

```
    while( )
```

$\equiv$

{

```

Nested Loops
for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
        System.out.print(i+" "+j)
}

```

|     |  | j → |     |     |     |     |
|-----|--|-----|-----|-----|-----|-----|
|     |  | 1   | 2   | 3   | 4   | 5   |
| i ↓ |  | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 |
| 2   |  | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 |
| 3   |  | 3,1 | 3,2 | 3,3 | 3,4 | 3,5 |
| 4   |  | 4,1 | 4,2 | 4,3 | 4,4 | 4,5 |
| 5   |  | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 |

```

public static void main(String[] args)
{
    for(int i=1;i<=5;i++)
    {
        for(int j=1;j<=7;j++)
        {
            System.out.print("( "+i+" , "+j+" ) ");
        }
        System.out.println("");
    }
}

```

|     |  | j → |   |   |   |   |
|-----|--|-----|---|---|---|---|
|     |  | 1   | 2 | 3 | 4 | 5 |
| i ↓ |  | 1   | 2 | 3 | 4 | 5 |
| 1   |  | 1   | 2 | 3 | 4 | 5 |
| 2   |  | 1   | 2 | 3 | 4 | 5 |
| 3   |  | 1   | 2 | 3 | 4 | 5 |
| 4   |  | 1   | 2 | 3 | 4 | 5 |
| 5   |  | 1   | 2 | 3 | 4 | 5 |

Patterns

```

for(int i=1; i<=5; i++)
{
    for(int j=1;j<=5;j++)
        System.out.print(j+" ");
    System.out.println();
}

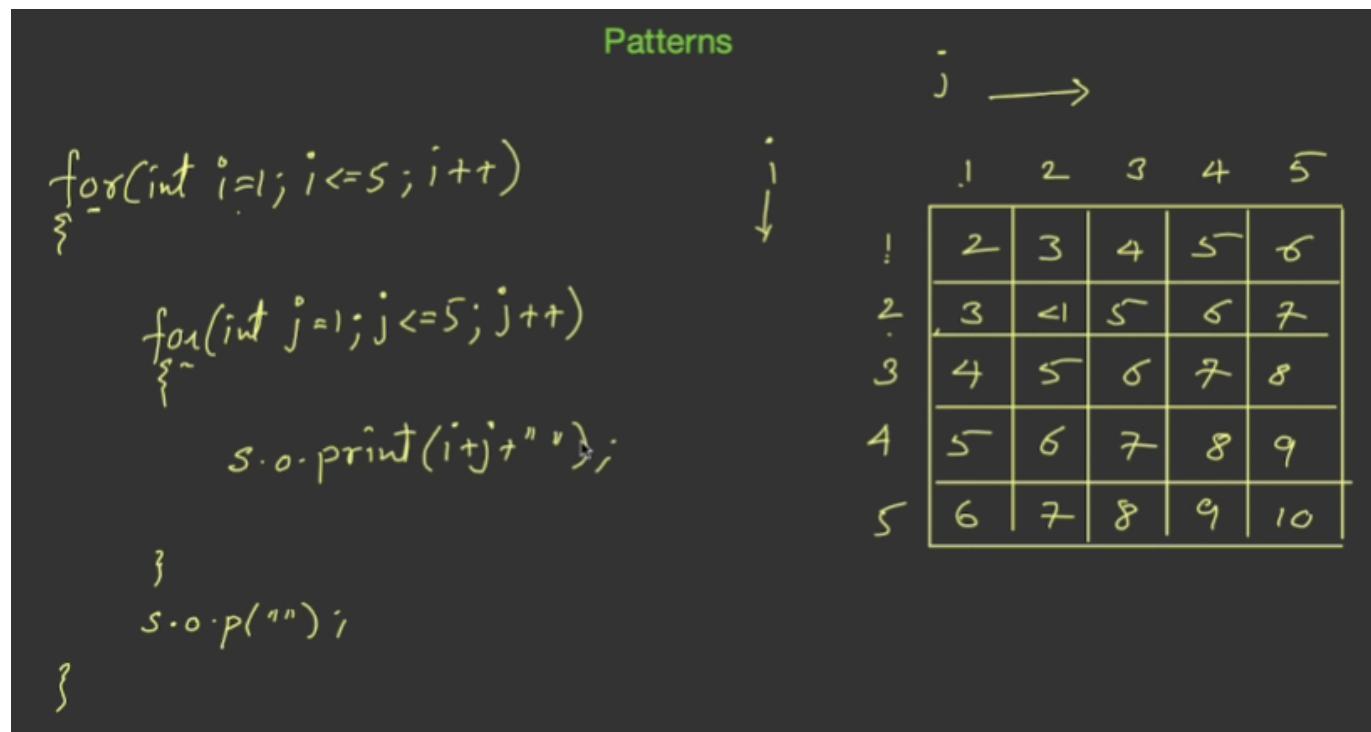
```

```
for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
        System.out.print(i+" ");
    System.out.println();
}
```

j →

| j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| i | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 2 | 2 | 2 |
| 2 | 3 | 3 | 3 | 3 | 3 |
| 3 | 4 | 4 | 4 | 4 | 4 |
| 4 | 5 | 5 | 5 | 5 | 5 |
| 5 |   |   |   |   |   |

```
public static void main(String[] args)
{
    for(int i=1;i<=5;i++)
    {
        for(int j=1;j<=5;j++)
        {
            System.out.print(i+" ");
        }
        System.out.println("");
    }
}
```



```

public static void main(String[] args)
{
    for(int i=1;i<=5;i++)
    {
        for(int j=1;j<=5;j++)
        {
            System.out.print(i+j+" ");
        }
        System.out.println("");
    }
}

```

```

int count=0;
for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
    {
        count++;
        System.out.print( count + " ");
    }
    System.out.println();
}

```

Patterns

j →

|   |    |    |    |    |    |
|---|----|----|----|----|----|
|   | 1  | 2  | 3  | 4  | 5  |
| 1 | 1  | 2  | 3  | 4  | 5  |
| 2 | 6  | 7  | 8  | 9  | 10 |
| 3 | 11 | 12 | 13 | 14 | 15 |
| 4 | 16 | 17 | 18 | 19 | 20 |
| 5 | 21 | 22 | 23 | 24 | 25 |

```

public static void main(String[] args)
{
    int count=0;

    for(int i=1;i<=5;i++)
    {
        for(int j=1;j<=5;j++)
        {
            count++;
            System.out.print(count+" ");
        }
        System.out.println("");
    }
    System.out.format("%2d ",count);
}

```

79.

Patterns

```

for(int i=1; i<=5; i++)
{
    for(int j=1; j<=i; j++)
        s.o.print(j + " ");
    s.o.p();
}

```

|   |   |   |   |   |   |
|---|---|---|---|---|---|
|   | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 |   |   |   |   |
| 2 | 1 | 2 |   |   |   |
| 3 | 1 | 2 | 3 |   |   |
| 4 | 1 | 2 | 3 | 4 |   |
| 5 | 1 | 2 | 3 | 4 | 5 |

```

int count=0;
for(int i=1;i<=5;i++)
{
    for(int j=1;j<=i;j++)
    {
        count++;
        System.out.print(count+" ");
    }
    System.out.println("");
}

```

Patterns

```

for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5-i+1; j++)
        s.o.print(j + " ");
    s.o.p();
}

```

|   |   |   |   |   |   |
|---|---|---|---|---|---|
|   | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 1 | 2 | 3 | 4 |   |
| 3 | 1 | 2 | 3 |   |   |
| 4 | 1 | 2 |   |   |   |
| 5 | 1 |   |   |   |   |

Patterns

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | * | * | * | * | * |
| 2 | - | * | * | * | * |
| 3 | - | - | * | * | * |
| 4 | - | - | - | * | * |
| 5 | - | - | - | - | * |

```

for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
    {
        if(j>=i)
            s.o.p(" * ");
        else
            s.o.p("   ");
    }
    s.o.p("\n");
}

```

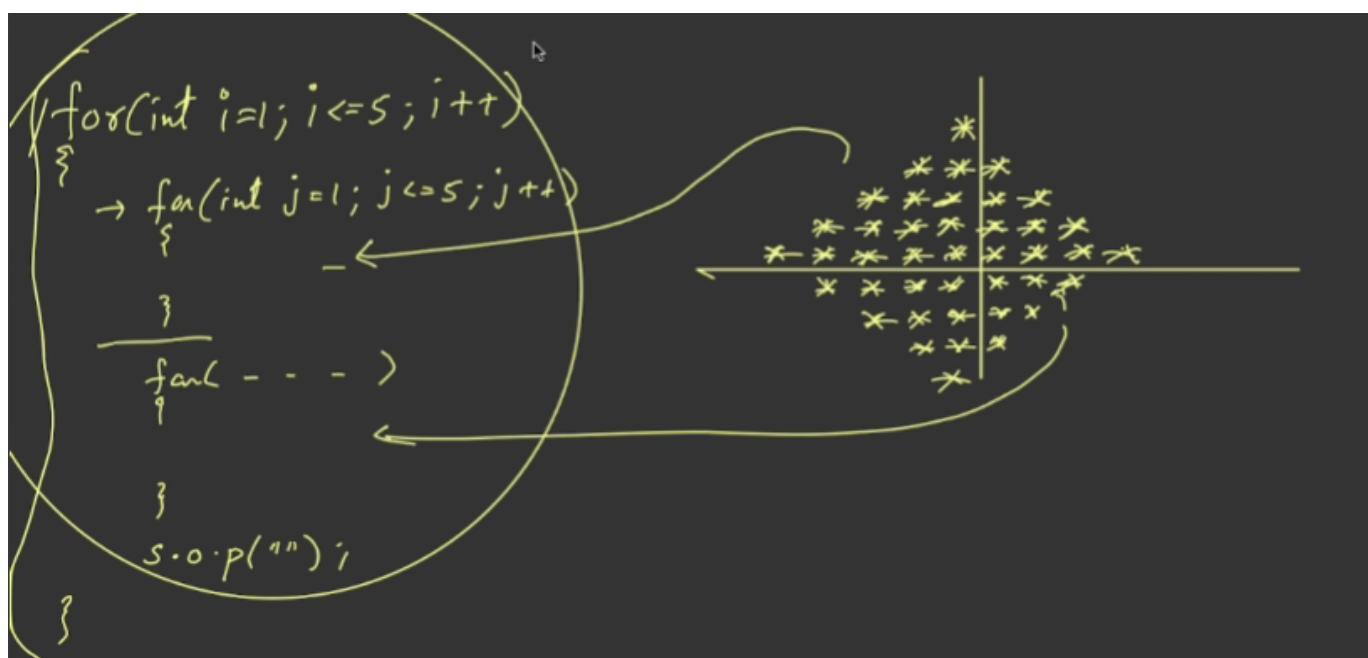
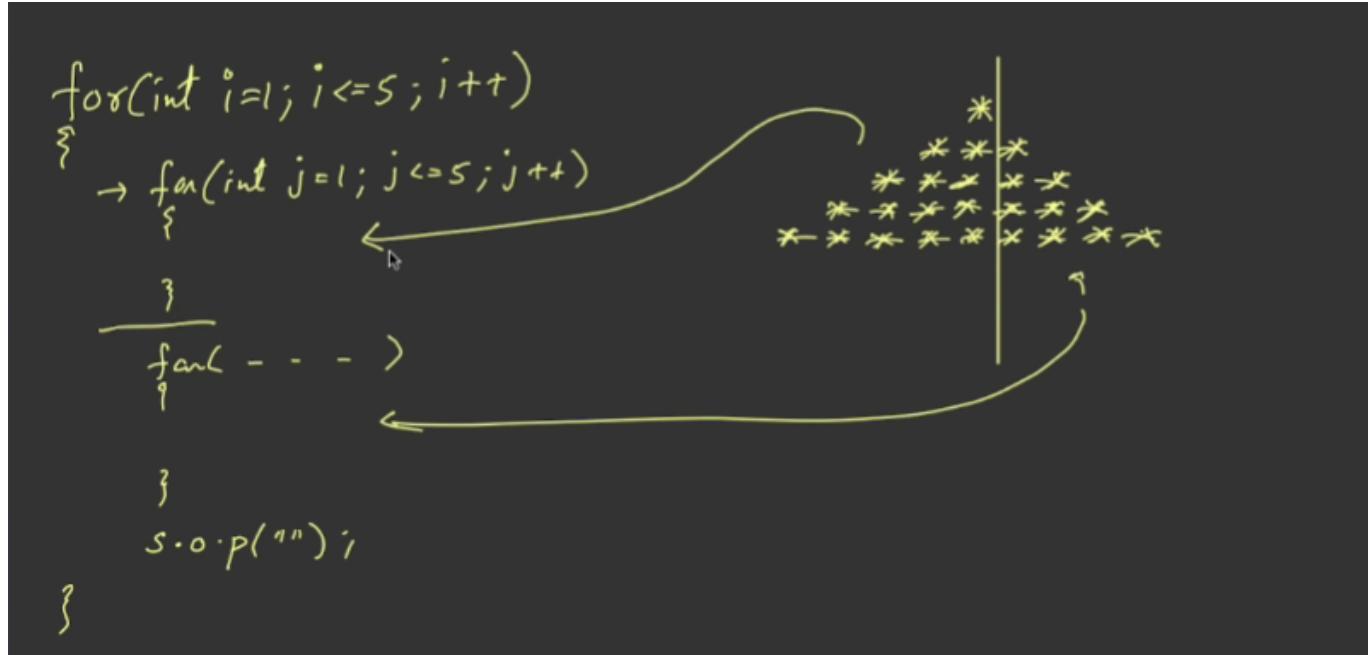
Patterns

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | - | - | - | * |
| 2 | - | - | - | * | * |
| 3 | - | - | * | * | * |
| 4 | - | * | * | * | * |
| 5 | * | * | * | * | * |

```

for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
    {
        if(i+j>5)
            s.o.p(" * ");
        else
            s.o.p("   ");
    }
    s.o.p("\n");
}

```



quiz

Which loop type would you use to repeat the set of instructions for specific number of times?

The loop that is guaranteed to be executed once is \_\_\_\_\_.

Variables declared inside a for loop are limited in scope to the loop.

What loop will display each of the numbers in this array on a separate line: float [ ] nums= {1.1f, 2.2f, 3....}

What is essential in making sure that your loop is not infinite ?

## Section 9: Arrays

### 81. 1D array

1. what is an Array
2. How to create and Access
3. for each Loop

== collection of similar data elements

//eg. x = [5,2,1,4,3]

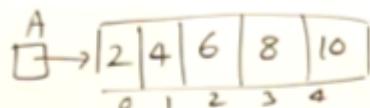
```
int a[] = new int[5]; a[] = {5,2,1,4,3}
```

- every array in java is an object and every array object is created in the heap memory a[] = is the reference int[5] = is the array object
- size of the array is fixed property System.out.println(a.length);

```
arr[5] = {5,2,1,4,3};
```

- array index starts from 0
- array index ends at n-1
- array index can be positive and negative

int A[] = {2, 4, 6, 8, 10}



```
for(int i=0; i < A.length; i++)
    System.out.println(A[i]);
```

```
for(int i=A.length-1; i >= 0; i--)
    System.out.println(A[i]);
```

- array index can be int and char

### for each loop

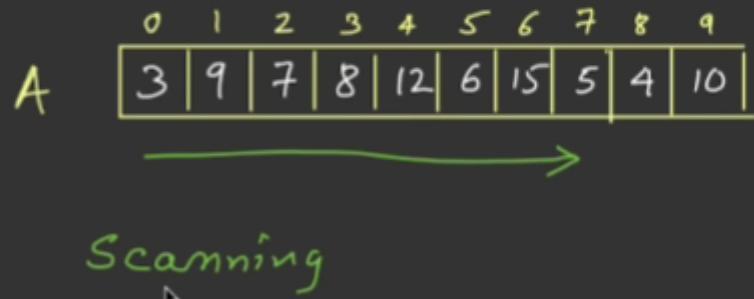
```
for(int x:arr){ System.out.println(x); }
```

we can only access in the forward direction only.

for each loop doesn't have index

array is mutable values can be incremented using the counter controlled loop

- ✓ 1. Finding Sum of all elements
- 2. Searching an Element
- 3. Finding Maximum Element
- 4. Finding Second Largest Element



Scanning  
Traversing  
Visiting

- ✓ 1. Finding Sum of all elements
- 2. Searching an Element
- 3. Finding Maximum Element
- 4. Finding Second Largest Element

length = 10

```
Sum = 0;
for(int i=0; i < A.length; i++)
{
    sum = sum + A[i];
}
```

A 

1.

```

public static void main(String[] args)
{
    int A[]={3,9,7,8,12,6,15,5,4,10};

    int sum=0;

    for(int i=0;i<A.length;i++)
    {
        sum=sum+A[i];
    }

    System.out.println("Sum is "+sum);
}

```

2.

↗ 1. Finding Sum of all elements  
 ↗ 2. Searching an Element  
 3. Finding Maximum Element  
 4. Finding Second Largest Element

Key = ?      length = 10

for(int i=0;i<A.length;i++)  
 {  
 if(A[i] == key)  
 {  
 System.out.println(i);  
 System.exit(0);  
 }  
 System.out.println("Not found");

A     
 

|   |   |   |   |    |   |    |   |   |    |
|---|---|---|---|----|---|----|---|---|----|
| 0 | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8 | 9  |
| 3 | 9 | 7 | 8 | 12 | 6 | 15 | 5 | 4 | 10 |

  
 ↓  
 ↗ [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

Key = 6      Key = 10

↗ 1. Finding Sum of all elements  
 ↗ 2. Searching an Element  
 ↗ 3. Finding Maximum Element  
 ↗ 4. Finding Second Largest Element

$\text{max} = A[0];$        $\text{length} = 10$

```

for(int i=0; i < A.length(); i++)
{
  if(A[i] > max)
    max = A[i];
}
System.out.println(max);
  
```

$A$ 

|   |   |   |   |    |   |    |   |   |    |
|---|---|---|---|----|---|----|---|---|----|
| 0 | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8 | 9  |
| 3 | 9 | 7 | 8 | 12 | 6 | 15 | 5 | 4 | 10 |

$\text{max} = A[0];$   
 ~~$\text{max} = 3 + 9 + 12 + 15 =$~~

4.

$A$ 

|   |   |   |   |    |   |    |   |   |    |
|---|---|---|---|----|---|----|---|---|----|
| 0 | 1 | 2 | 3 | 4  | 5 | 6  | 7 | 8 | 9  |
| 3 | 9 | 7 | 8 | 12 | 6 | 15 | 5 | 4 | 10 |

$\text{max1} = \cancel{3} \cancel{9} \cancel{12} \cancel{15}$        $\text{max2} = \cancel{3} \cancel{9} \cancel{12} \cancel{8} \cancel{15}$

```

max1=max2=A[0];

for(int i=0;i<A.length;i++)
{
    if(A[i]>max1)
    {
        max2=max1;
        max1=A[i];
    }
    else if(A[i]>max2)
    {
        max2=A[i];
    }
}
System.out.println("Second Largest is "+max2);

```

84.

✓ 1. Rotating an Array      left      Right
   
 2. Inserting an Element
   
 3. Deleting an Element

```

int A[]={---};
int temp=A[0];
for(int i=1; i<A.length; i++) temp=A[i];
A[A.length-1]=temp;
  
```

1. left rotation

```

int A[]={3,9,7,8,12,6,15,5,4,10};

for(int x:A)
    System.out.print(x+",");
System.out.println("");

int temp=A[0];

for(int i=1;i<A.length;i++)
{
    A[i-1]=A[i];
}
A[A.length-1]=temp;

for(int x:A)
    System.out.print(x+",");
System.out.println("");

```

try right rotation

|                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\text{int } A[] = \text{new int}[10];$<br>$A[0] = 5;$<br>$A[1] = 9;$<br>$A[2] = 6;$<br>$A[3] = 10;$<br>$A[4] = 12;$<br>$A[5] = 7;$<br><br>$\text{for } (\text{int } i=n; i>\text{index}; i--)$<br>$\quad A[i] = A[i-1];$<br>$A[\text{index}] = x;$ | <p style="text-align: center;"> <u>2. Inserting an Element</u><br/> <u>3. Deleting an Element</u> </p> <p style="text-align: right;"> <math>\text{length} = 10</math><br/> <math>n = 6</math> </p> <p style="text-align: center;"> <math>x = 15</math><br/> <math>\text{index} = 2</math> </p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

inserting an element in middle or given index

```

int A[] = new int[10];
A[0]=3;A[1]=9;A[2]=7;A[3]=8;A[4]=12;A[5]=6;

int n=6;

for(int i=0;i<n;i++)
    System.out.print(A[i]+",");
System.out.println("");

int x=20;
int index=2;

for(int i=n;i>index;i--)
    A[i]=A[i-1];
A[index]=x;

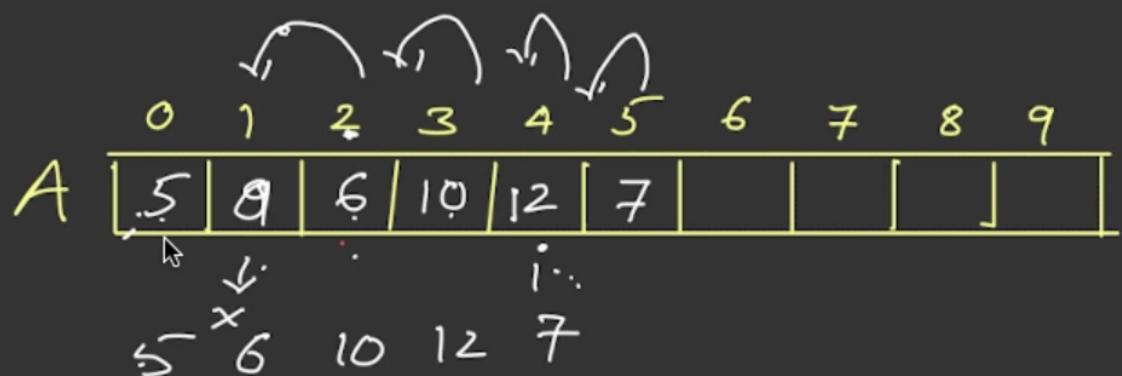
for(int i=0;i<n;i++)
    System.out.print(A[i]+",");
System.out.println("");

```

code for the below diagrams

### 3. Deleting an Element

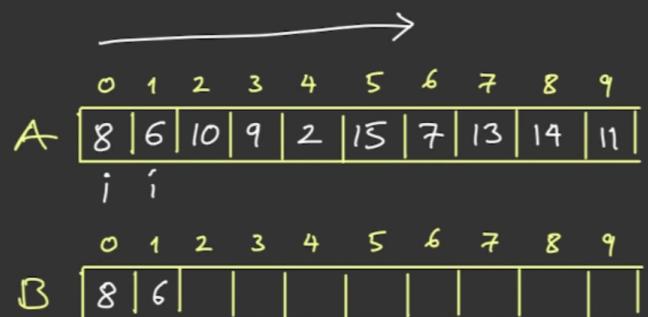
$$\begin{aligned} \text{length} &= 10 \checkmark \\ n &= \cancel{6} \quad 5 \end{aligned}$$



$$\text{index} = 1$$

- 1. Copying an Array
- 2. Reverse Copying an Array
- 3. Increasing Size of Array

```
int A[] = { - - - - };
int B[] = new int[10];
for(int i=0; i<A.length; i++)
{
}
```



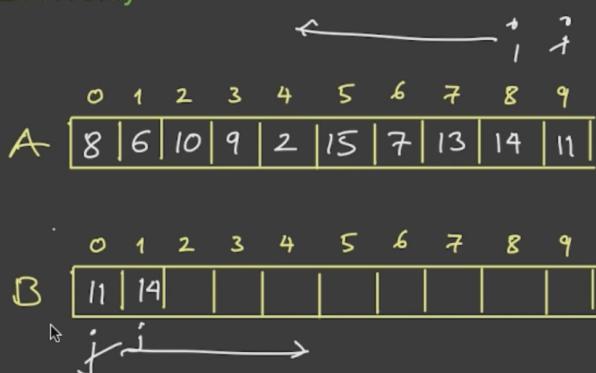
— — . . .

}

— — . . .

- 2. Reverse Copying an Array
- 3. Increasing Size of Array

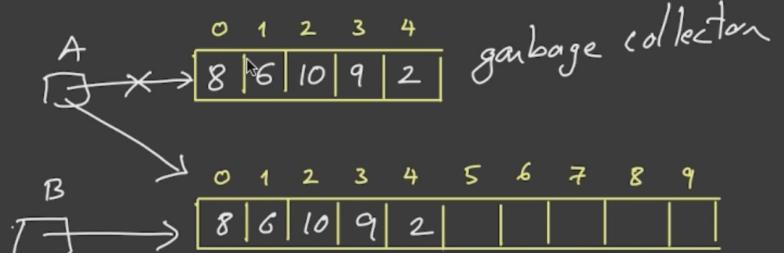
```
for(int i=A.length-1, j=0; i>=0; i--, j++)
{}
```

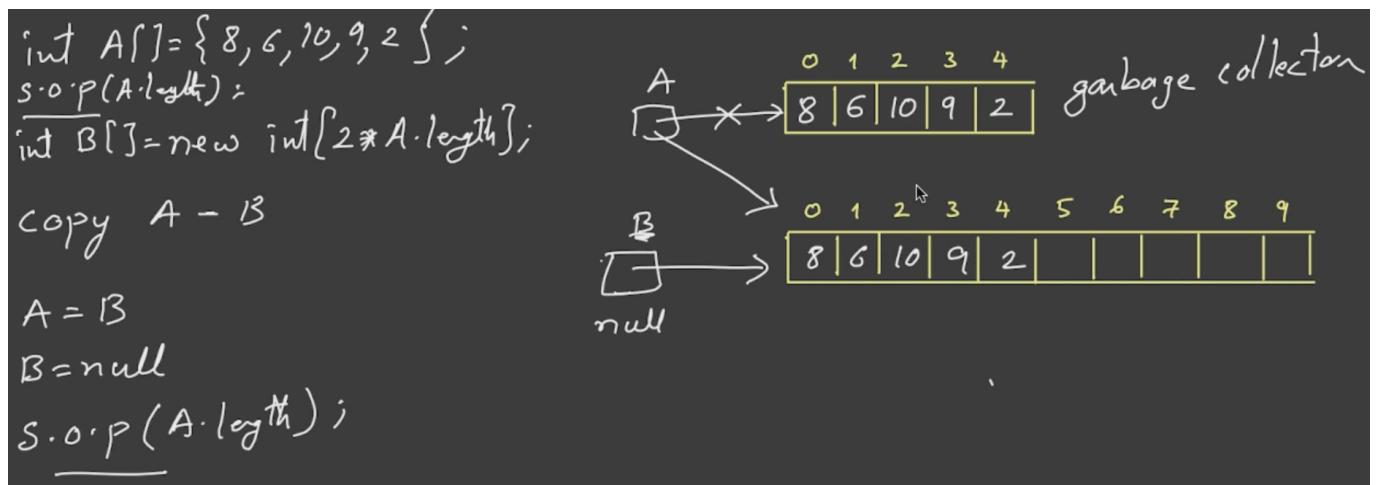


Display B

- 3. Increasing Size of Array

```
int A[] = { 8, 6, 10, 9, 2 };
int B[] = new int[2*A.length];
```

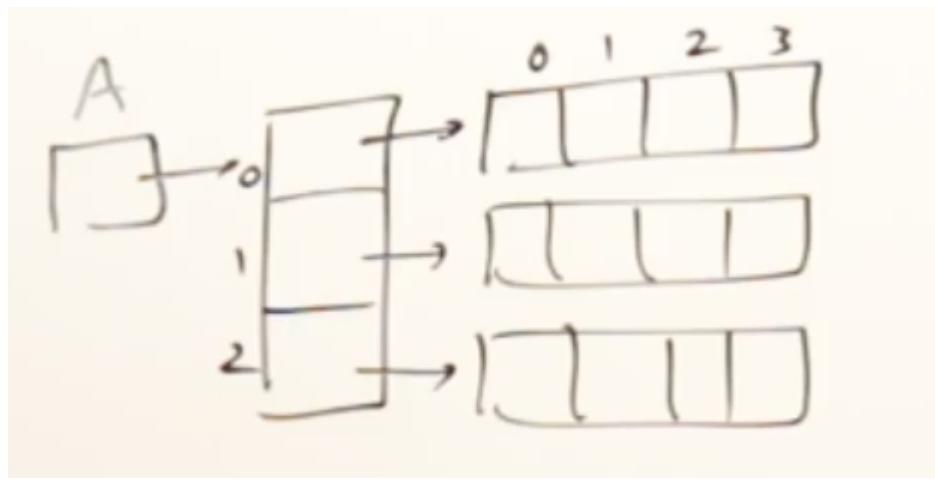




## 2D Array

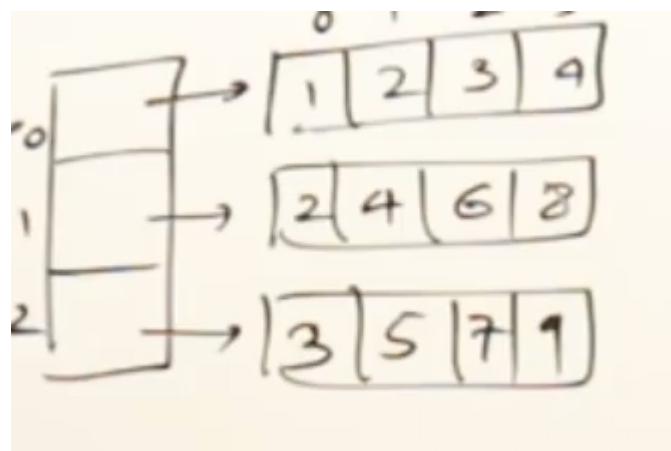


`int A[][] = new int[3][4];`



array of ref array of elems

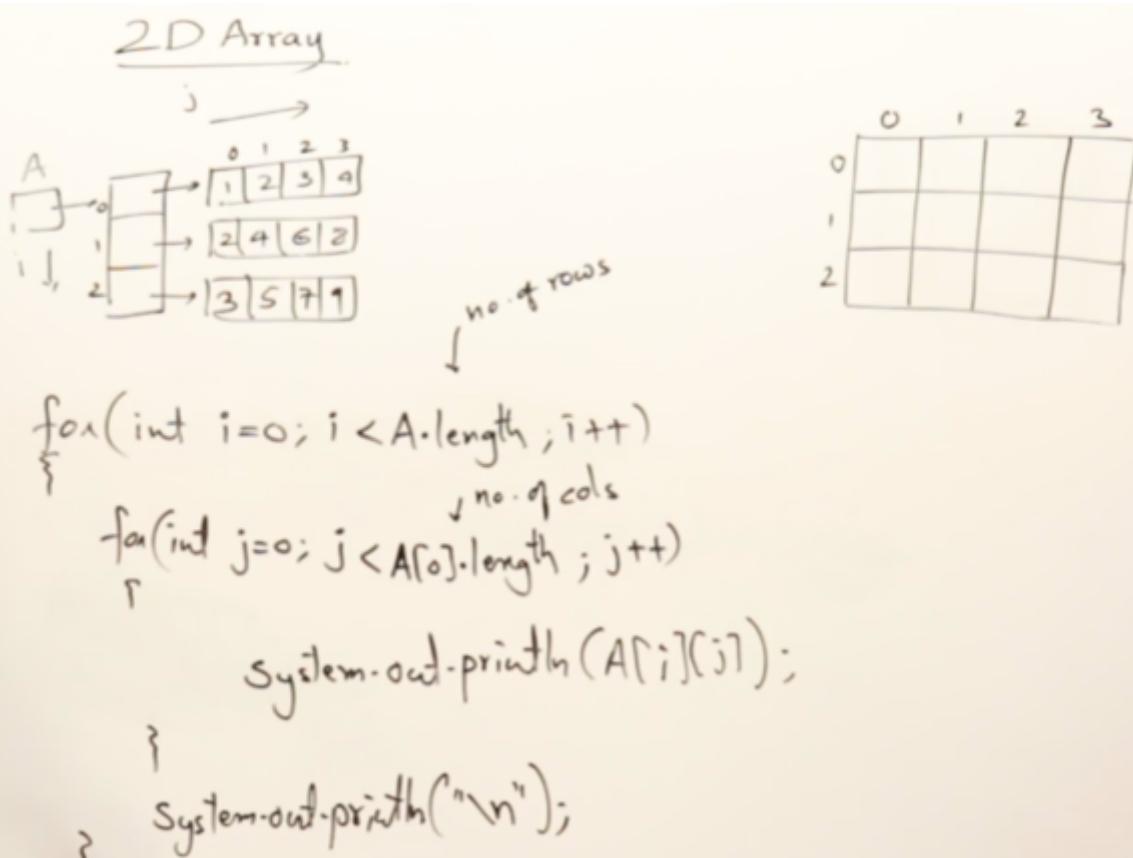
`int A[][] = {{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 5, 7, 9}};`

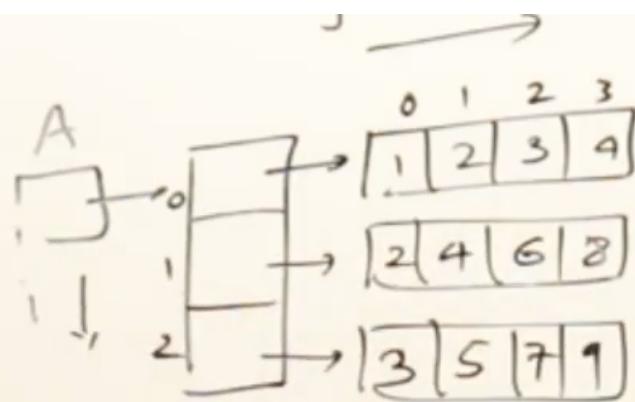


```
int A[ ][ ];
A=new int[3][4];
```

ref's of 2d and

and declaration



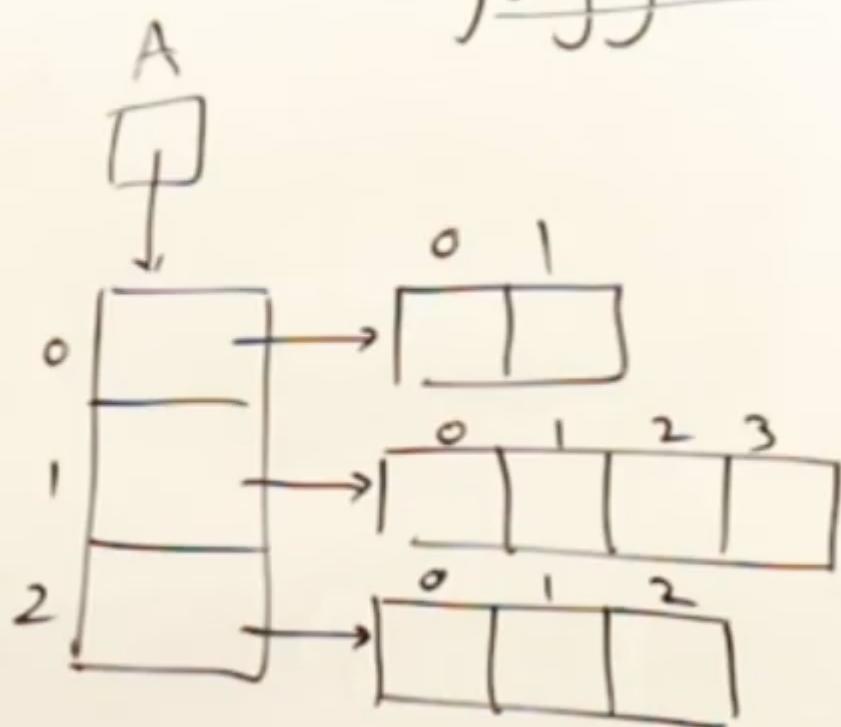


```
for(int x[]: A )  
{  
    for(int y : x )  
        System.out.println(y);  
    System.out.println("\n");  
}
```



## 2D Array

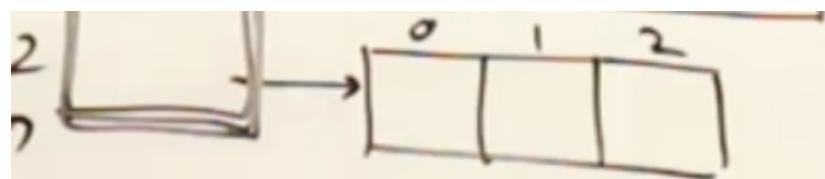
## Jagged Array



## 2D Array

## Jagged Array





```
int A[][];  
A = new int[3][];  
A[0] = new int[2];  
A[1] = new int[4];  
A[2] = new int[3];
```

88. sc : matrix mul

- ✓ 1. Adding 2 Matrices
- 2. Multiplying 2 Matrices
- 3. Sorting Array of Strings

 $3 \times 3$  $3 \times 3$ 

A

| 0 | 1 | 2 |   |
|---|---|---|---|
| 0 | 3 | 5 | 9 |
| 1 | 7 | 6 | 2 |
| 2 | 4 | 3 | 5 |

B

| 0 | 1 | 2 |   |
|---|---|---|---|
| 0 | 1 | 5 | 2 |
| 1 | 6 | 8 | 4 |
| 2 | 3 | 9 | 7 |

C

| 0 | 1 | 2 |  |
|---|---|---|--|
| 0 |   |   |  |
| 1 |   |   |  |
| 2 |   |   |  |

A + B

```

for(int i=0; i<3; i++)
{
    for(int j=0; j<3; j++)
    {
        c[i][j] = A[i][j] + B[i][j];
    }
}

```

A

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 3 | 5 | 9 |
| 1 | 7 | 6 | 2 |
| 2 | 4 | 3 | 5 |

B

|   | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 5 | 2 |
| 1 | 6 | 8 | 4 |
| 2 | 3 | 9 | 7 |

C

A+B

|   | 0   | 1   | 2   |
|---|-----|-----|-----|
| 0 | 3+1 | 5+5 | 9+2 |
| 1 |     |     |     |
| 2 |     |     |     |

```

public static void main(String[] args)
{
    int A[][]={{3,5,9},{7,6,2},{4,3,5}};
    int B[][]={{1,5,2},{6,8,4},{3,9,7}};

    int C[][]=new int[3][3];

    for(int i=0;i<A.length;i++)
    {
        for(int j=0;j<A[0].length;j++)
        {
            C[i][j]=A[i][j]+B[i][j];
        }
    }
}

```

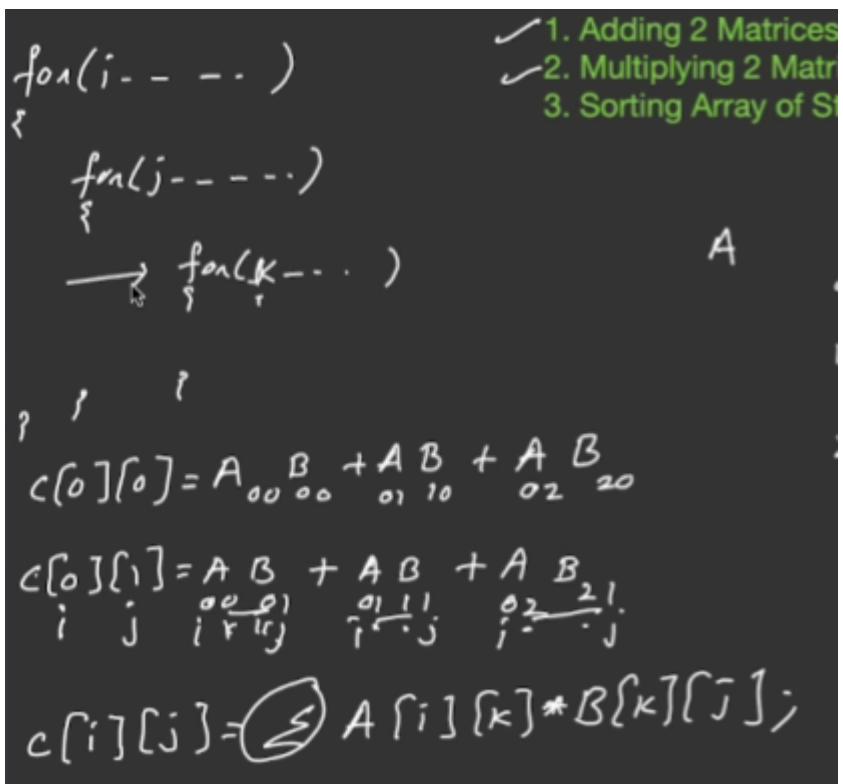
|                                            |                                                     |
|--------------------------------------------|-----------------------------------------------------|
| $A$<br>$2 \times 3$<br>$3 \times 3$<br>mul | $B$<br>$3 \times 5$<br>$2 \times 3$<br>$AB$<br>$BA$ |
|--------------------------------------------|-----------------------------------------------------|

|                                                                                                                                    |                                                                                                                                    |                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| $A$<br>$\begin{array}{ c c c } \hline 0 & 1 & 2 \\ \hline 3 & 5 & 9 \\ \hline 7 & 6 & 2 \\ \hline 4 & 3 & 5 \\ \hline \end{array}$ | $B$<br>$\begin{array}{ c c c } \hline 0 & 1 & 2 \\ \hline 1 & 5 & 2 \\ \hline 6 & 8 & 4 \\ \hline 3 & 9 & 7 \\ \hline \end{array}$ | $C$<br>$\begin{array}{ c c c } \hline 0 & 1 & 2 \\ \hline - & - & - \\ \hline - & - & - \\ \hline - & - & - \\ \hline \end{array}$ |
|------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|

$$C[0][0] = A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$$

$$C[0][1] = A_{00}B_{01} + A_{01}B_{11} + A_{02}B_{21}$$



CODE

```

int A[][]={{3,5,9},{7,6,2},{4,3,5}};
int B[][]={{1,0,0},{0,1,0},{0,0,1}};

int C[][]=new int[3][3];

for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        C[i][j]=0;
        for(int k=0;k<3;k++)
        {
            C[i][j]=C[i][j]+A[i][k]*B[k][j];
        }
    }
}

```

first check with identity matrix

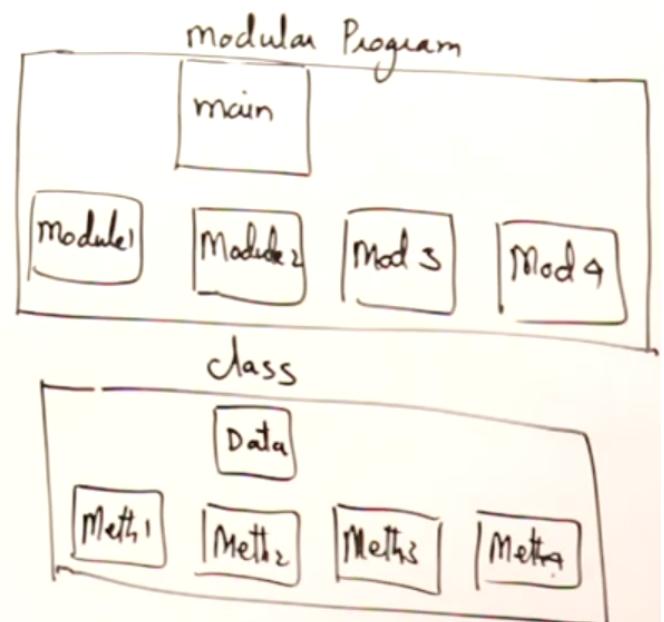
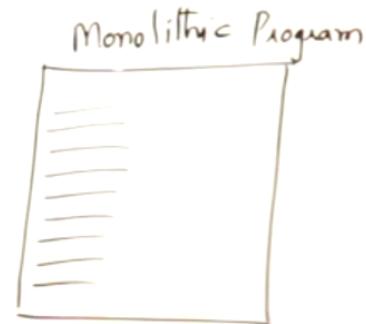
inbuilt sort

```
public static void main(String args[])
{
    String arr[]={ "java", "python", "pascal", "smalltalk", "ada", "basic" };
    java.util.Arrays.sort(arr);
    for(String x:arr)
        System.out.println(x);
}
```

## Section 10: Methods

### Methods

1. What are Methods
2. How to write a Method
3. Parameter Passing
4. Method Return type
5. Method Overloading
6. varargs



=methods are the members of the classes, which provides the functionality for the class.

methods/subroutines/functions

- skeleton of method

returnType    methodName( parameter List )  
{  
      
      
      
}

≡

}

int max(int x, int y){ if(x>y){ return x; } else{ return y; } }

class Test  
{

    static int max(int x,int y)

{

        if(x>y)  
            return x;  
        else  
            return y;

}

    public static void main(String args[])

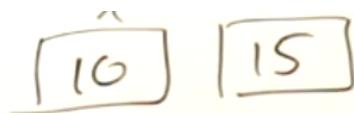
        int a=10,b=15,c;

        c=max(a,b);

}

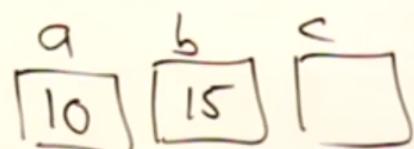
respective parameter are copied

```
static int max(int x,int y)
```



```
{
    if(x>y)
        return x;
    else
        return y;
}
```

```
public static void main(String args[])
{
    int a=10,b=15,c;
```

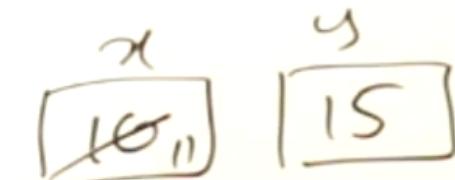


```
c=max(a,b);
```

```

}
System.out.println(c);
}
```

```
int max(int x,int y)
```



```
x++;
```

```
if(x>y)
```

```
    return x;
```

```
else
```

```
    return y;
```

actual parameter and formal parameters

(pass by value and pass by reference)

static methods can call only static method it can't call non static methods

CALL BY VALUE value of actual parameter will not be modified by the formal parameter

```
static void inc(int x)
{
    x++;
    System.out.println(x);
}

public static void main(String[] args)
{
    int a=10,b=15;

    inc(a);

    System.out.println(b);
}
```

91. Passing Obj as the parameter.

```
static void update(int A[])
{
    A[0]=25;
}
```

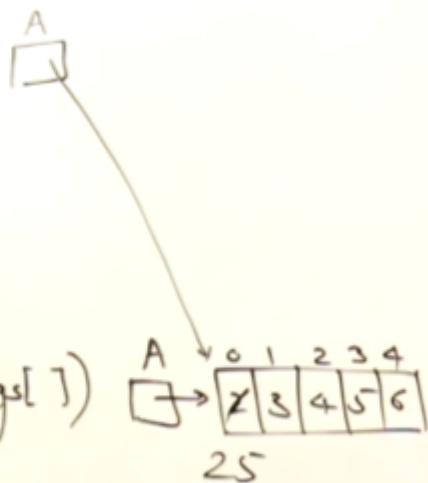
public static void main(String args[])
{
 int A[]={2,3,4,5,6};
 update(A);
 S.O.P(A[0]);
}

A → 

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 2 | 3 | 4 | 5 | 6 |

```
static void update(int A[])
A[0]=25;
```

```
public static void main(String args[])
int A[]={2,3,4,5,6};
update(A);
```



```
void
int max(int x,int y)
{
    return x>y?x:y;
}
```



```
String userName(String email)
{
    int a=email.indexOf('@');
    String name=email.substring(0,a);
    return name;
}
```

eg.

93.

## Parameter Passing

```
int add(int x,int y)
{
    int z;
    z=x+y;
    return z;
}
```

```
void welcome(String n)
{
    System.out.println("Welcome Mr./Miss "+n);
}
```

```
P.S.V.main(---)
{
    int a=10,b=5,c;
    c=add(a,b);
    System.out.println(c);
}
```

```
P.S.V.main(..)
{
    String name="Victor";
    welcome(name);
}
```

Parameter Pa

outpt      inpt

```
int add(int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```

x      y      z  
10     5     15

Detailed description: This diagram illustrates parameter passing by value. It shows a function definition for 'add' with two formal parameters, 'x' and 'y'. Below it, a call to 'main' initializes variables 'a', 'b', and 'c' with values 10, 5, and 10 respectively. Arrows point from the actual parameters 'a' and 'b' to their respective formal parameters 'x' and 'y'. Another arrow points from 'c' to the return value '15', which is enclosed in a box labeled 'z'.

P.S.V.main(---)

```
{}
int a=10, b=5, c;
```

method call → c=add(a,b);  
S.O.P(c); *actual Param*

```
void welcome(String n)  
{
```

```
    s.o.p("Welcome Mr./Miss "+n)
```

```
}
```

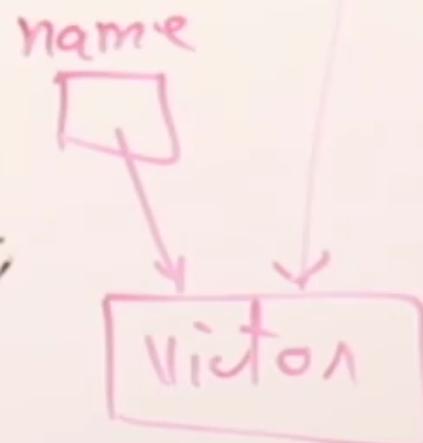


```
P.S.V.main(..)  
{
```

```
    String name="Victor";
```

```
    Welcome(name);
```

```
}
```



#### 94. sc : find prime number

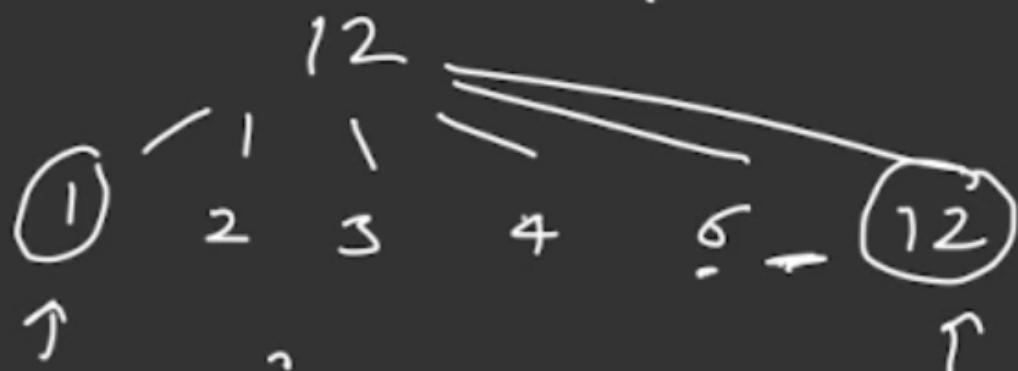
- 1. Find a Number is Prime
- 2. Find GCD of 2 numbers
- 3. Find Max Element in an Array

$$n = \underline{1} \underline{2}$$

$$\begin{array}{r} 3 ) 12 ( 4 \\ \underline{12} \\ \underline{0} \end{array}$$

$$\begin{array}{r} 6 ) 12 ( 2 \\ \underline{12} \\ \underline{0} \end{array}$$

non-prime.



prime

- 1. Find a Number is Prime
- 2. Find GCD of 2 numbers
- 3. Find Max Element in an Array

non-prime

~~x1, 3~~

~~n=91~~ ~~13~~

static boolean isPrime(int n)

prime.

$$n=19$$

|   |   |
|---|---|
| 1 |   |
| 2 | x |
| 3 | x |
| 4 | x |
| 5 | x |
| 6 | x |
| 7 | x |
| 8 | x |
| 9 | x |

$$\frac{19}{2} = 9.5$$

|   |   |
|---|---|
| 1 |   |
| 2 | x |
| 3 | x |
| 4 | x |
| 5 | x |
| 6 | x |
| 7 | x |

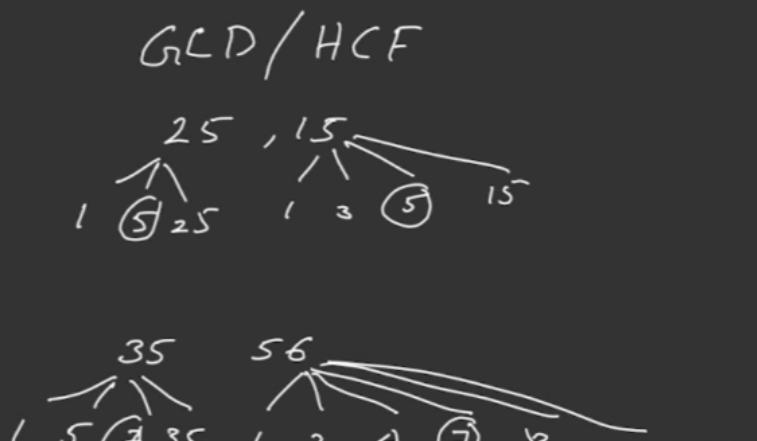
That's it.

```

public class SCMethod1
{
    static boolean isPrime(int n)
    {
        for(int i=2;i<n/2;i++)
        {
            if(n%i==0)
                return false;
        }
        return true;
    }
    public static void main(String[] args)
    {
        System.out.println(isPrime(19));
    }
}

```

2. Find GCD of 2 numbers  
 3. Find Max Element in an Array



Algorithm for gcd

|                    |                   |           |
|--------------------|-------------------|-----------|
| $m = 25$           | $n = 15$          |           |
| $m = 25 - 15 = 10$ | $n = 15$          |           |
| $m = 10$           | $n = 15 - 10 = 5$ |           |
| $m = 10 - 5 = 5$   | $n = 5$           |           |
|                    |                   | GCD = 5 ↴ |

|                    |                    |  |
|--------------------|--------------------|--|
| $m = 35$           | $n = 56$           |  |
| $m = 35$           | $n = 56 - 35 = 21$ |  |
| $m = 35 - 21 = 14$ | $n = 21$           |  |
| $m = 14$           | $n = 21 - 14 = 7$  |  |
| $m = 7$            | $n = 7$            |  |

```
static int gcd(int m,int n)
{
    while(m!=n)
    {
        if(m>n) m=m-n;
        else n=n-m;
    }
    return m;
}
```

## Method Overloading

```
int max(int x, int y)
{
    return x > y ? x : y;
}
```

```
float max(float x, float y)
{
    return x > y ? x : y;
}
```

```
int max(int x, int y, int z)
{
    return x > y && x > z ? x : (y > z ? y : z);
}
```

From this example I'll explain

methods having same name and different parameter list (Order of the parameters, type of the parameters, number of parameters)

```
int max(int x, int y)
{
    return x > y ? x : y;
}

max(10, 15);
```

→

```
float max(float x, float y)
{
    return x > y ? x : y;
}

max(5.5f, 7.7f);
```

incompatible type possible loss of conversion

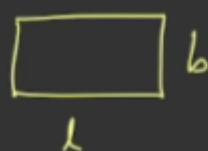
```
static int max(int x,int y)
{
    return x>y?x:y;
}

public static void main(String[] args)
{
    System.out.println(max(10.5f,5.4f));
}
```

name is same, but method call is different this is known as false polymorphism (behaviour is different, based on the parameters) return type is not considered for the method overloading

## 97. SC : overload validate method

- 1. overloaded method to calculate areas
- 2. overloaded method to reverse a int or array
- 3. overloaded method to validate name and age



$$\cdot a = l * b \quad a = \text{Math.PI} * r * r$$

double area(double l, double b)

double area(double radius)



$$\cdot a = l * b$$



$$a = \text{Math.PI} * r * r$$



$$\text{Area} = \frac{1}{2} (a+b) h$$

```
double area(double l, double b)
```

```
double area(double radius)
```

↓

$n = 237$

732

```
int reverse(int n)
```

{

=  
=  
=  
=

}

```
void reverse(int A[])
```

{

=  
=  
=

}

```
int reverse(int n)
{
    int rev=0;

    while(n != 0)
    {
        rev=rev*10+n%10;
        n=n/10;
    }
    return rev;
}
```

```
int [] reverse(int A[])
{
    int B[] = new int[A.length];

    for(int i=A.length-1, j=0; i>=0; i--, j++)
        B[j] = A[i];
    return B;
}
```

3.

```
boolean validate(String name)
{
    return name.matches("[a-zA-Z\\s]+");
}

boolean validate(int age)
{
    return age>=3 && age<=15;
}
```

### variable arguments

98.

```
void show(int... x) { int [3]x
{
    for(int a: x)
    {
        s.o.p(a);
    }
}
```

```
show();
show(10);
show(10, 20, 30);
show(10, 20, 30, 40);
show(10, 20, 30, 40, 50, ...);
• Show(new int[7]{10, 20, 30, 40});
    void show(int * , int... y)
```

```
show();
show(10);
show(10, 20, 30);
show(10, 20, 30, 40);
show(10, 20, 30, 40, 50, ...);
show(new int[7]{10, 20, 30, 40});
```

### Variable Arguments varargs

int a=5, b=6, c=7, d=8. .

printf(" .d .d .d .d", a, b, c, d);

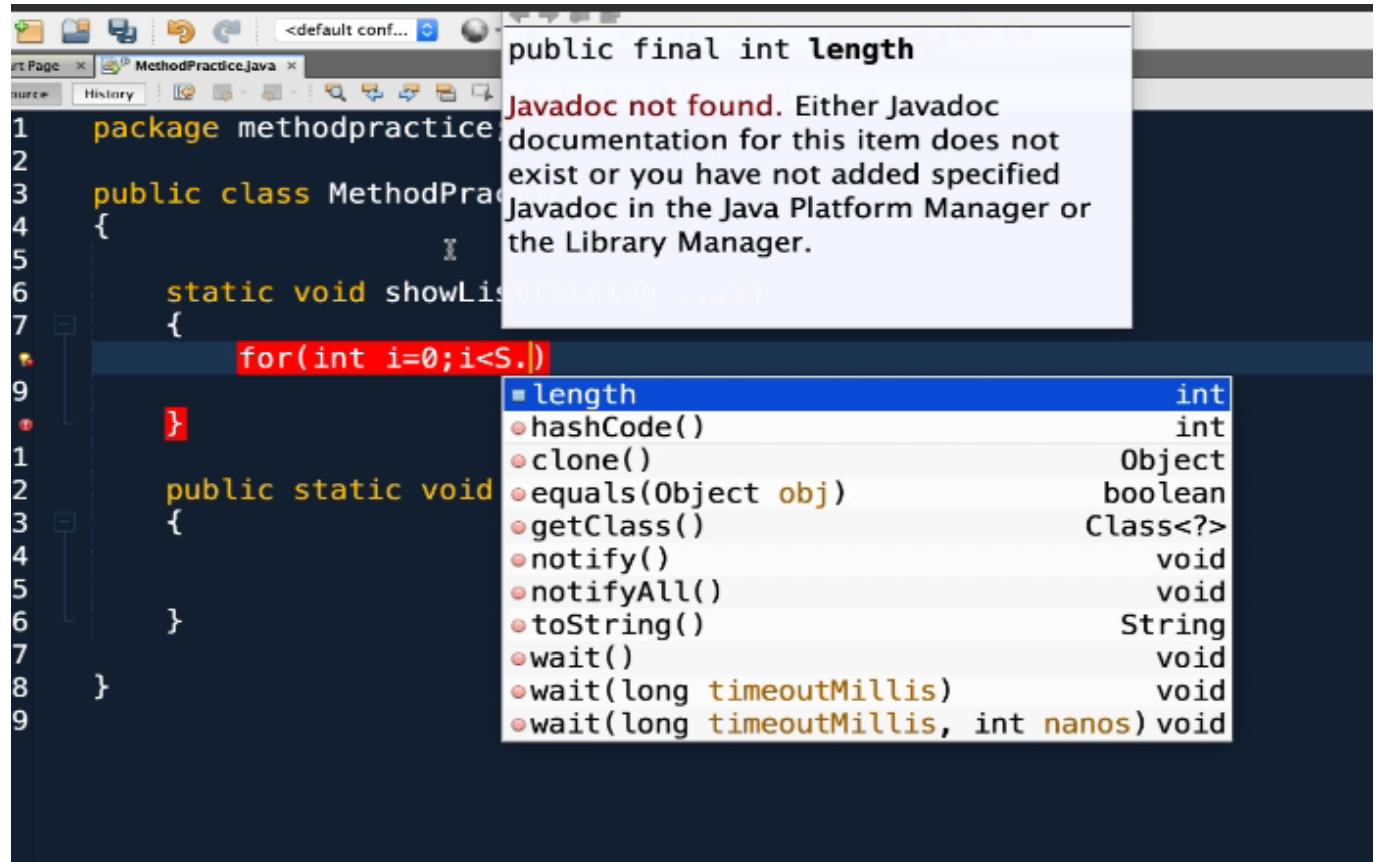
printf uses the ellipse in c

var args is more powerfull than array

```
public class MethodPractice
{
    static void show(int ...A)
    {
        for(int x:A)
        {
            System.out.println(x);
        }
    }
    public static void main(String[] args)
    {

        show();
        show(10,20,30);
        show(new int[]{3,5,7,9,11,13,15});
    }
}
```

inbuilt methods of the var args



.length

try it

```
public class MethodPractice
{
    static void showList(String ...s)
    {
        for(int i=0;i<s.length;i++)
        {
            System.out.println(i+1+". "+s[i]);
        }
    }

    public static void main(String[] args)
    {
        showList("John","Smith","Ajay","Ahmed");
    }
}
```

variable arguments must be the last parameter in the method 100.

1. Maximum of numbers using varargs
2. Sum of all elements using varargs
3. Calculate Discount using varargs

1.

```

int max(int ...A)
{
    if(A.length == 0) return Integer.MIN_VALUE;
    int m = A[0];
    for(int i=1; i < A.length; i++)
    {
        →
        m = A[i] > m ? A[i] : m;
    }
    return m;
}

```

```

static int max(int ...A)
{
    if(A.length==0) return Integer.MIN_VALUE;
    int max=A[0];
    for(int i=1;i<A.length;i++)
        if(A[i]>max)max=A[i];

    return max;
}

public static void main(String[] args)
{
    System.out.println(max());
    System.out.println(max(10));
    System.out.println(max(10,20));
    System.out.println(max(10,20,30));
}

```

2.3.

int sum(int ...A[])

→ { int s=0;  
     for(int i=0; i < A.length; i++)  
         s += A[i];  
     return s;
}

discount  
 $\downarrow$

prices  
 $\downarrow$

double discount(double ...P)

sum --

if (sum < 500)  
 $\quad \quad \quad$  if ( $500 - 1000$ )  
 $\quad \quad \quad$   $\frac{10\%}{45\%}$   
 $\quad \quad \quad$  if ( $1000 - 2000$ )  
 $\quad \quad \quad$   $\frac{20\%}{50\%}$

**command line args**

```
import java.lang.*;  
  
class CommandTest  
{  
    public static void main(String args[])  
    {  
        for(String s:args)  
        {  
            System.out.println(s);  
        }  
    }  
}
```

```
C:\MyJava>java CommandTest hello all how are you  
hello  
all  
how  
are  
you
```

## Adding Numbers using Command Line

C:\>java SUM 12 15 16.92 8.55 3.7 ↵

```
main(String args[])
{
    s=0;
    s = s + Double.parseDouble(" ");
    System.out.println(s)
```

```
public class MySum
{
    public static void main(String[] args)
    {
        double s=0;

        for(String x:args)
        {
            //if(x.matches("[0-9|.|]+"))
            s=s+Double.parseDouble(x);
        }

        System.out.println("Sum is "+s);
    }
}
```

103. Recursion

## Recursion

```
void fun(int n)
{
    if(n>0)
    {
        s.o.p(n);
        fun(n-1);
    }
}
```

---

$\downarrow$

---

fun(3);

```
void fun(int n)
{
    if(n>0)
    {
        fun(n-1);
        s.o.p(n);
    }
}
```

---

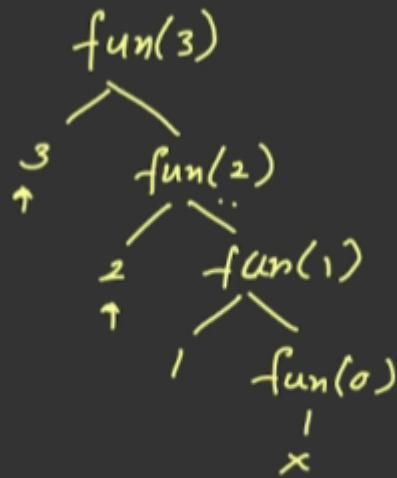
fun(3);

= calling the method itself java is not optimized for the recursion java supports the recursion eg.rubber band

```
void fun(int n)
{
    if(n>0)
    {
        s.o.p(n);
        fun(n-1);
    }
}
```

---

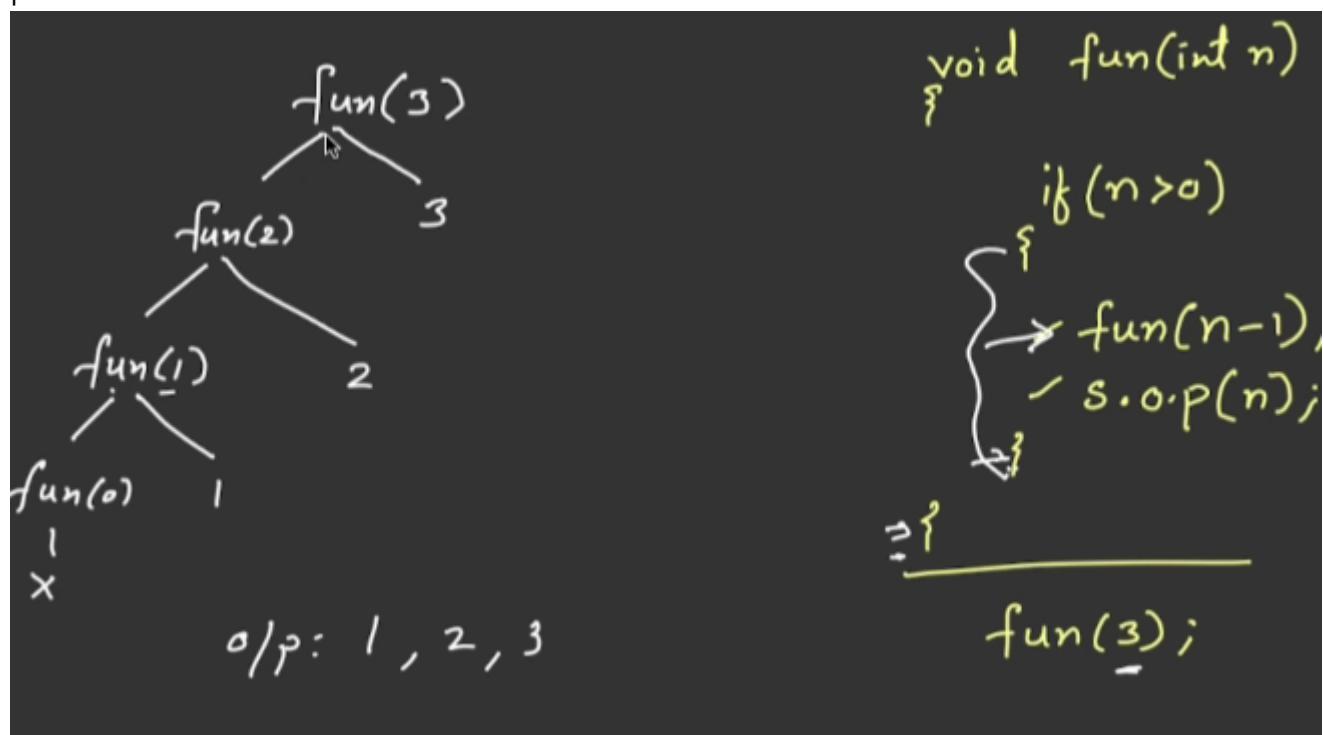
fun(3);



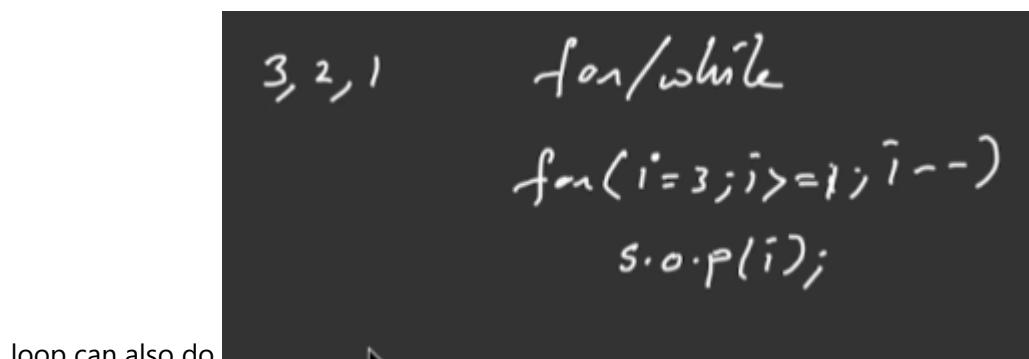
o/p: 3, 2, 1

first value is

printed then the recursive call is made RECURSION



first recursive call is made then the value is printed



loop can also do

then why recursion ? see recursion, when it is calling itself again and again.Then it'll return back. so while returning also it can do something while returning it can do something and it can be used in the tree traversal (this can't be done in the loop. easily)

recursion is not used much as it takes extra memory and time. in problem solving, mathematics -> recursion is used (maths doesn't have loops, so recursion is used)

## Section 11: OOPs

### 104 : principles of oops

learning oops and java is different (oops is the concept and general)

- principles of oops
  1. Abstraction
  2. Encapsulation
  3. Inheritance
  4. Polymorphism

this concepts are related to the real world things **Abstraction** means hiding the internal details and showing the only required things eg. TV = just on and off button is shown, but internally there are many things (like circuits, etc) eg. car = just steering wheel, accelerator, brake, etc are shown, but internally there are many things (like engine, etc) **Encapsulation** means wrapping the data and the methods into a single unit eg. TV = wire and circuits are wrapped in the plastic (capuled as a single unit) eg. car = engine, gear, etc are wrapped in the metal body (capuled as a single unit) **Inheritance** means acquiring the properties of the parent class (reusability, code reusability, borrowing the features) eg. TV = CRT, LCD, LED, etc are the child classes and the parent class is the TV eg. car = car is the parent class and the child class is the sports car, luxury car, etc

**Polymorphism** smartphone = mi, samsung, pixel, motrolla, car = toyota, honda, maruthi, etc

1.generalization (inheritance) 2.specialization (overloading and overriding)

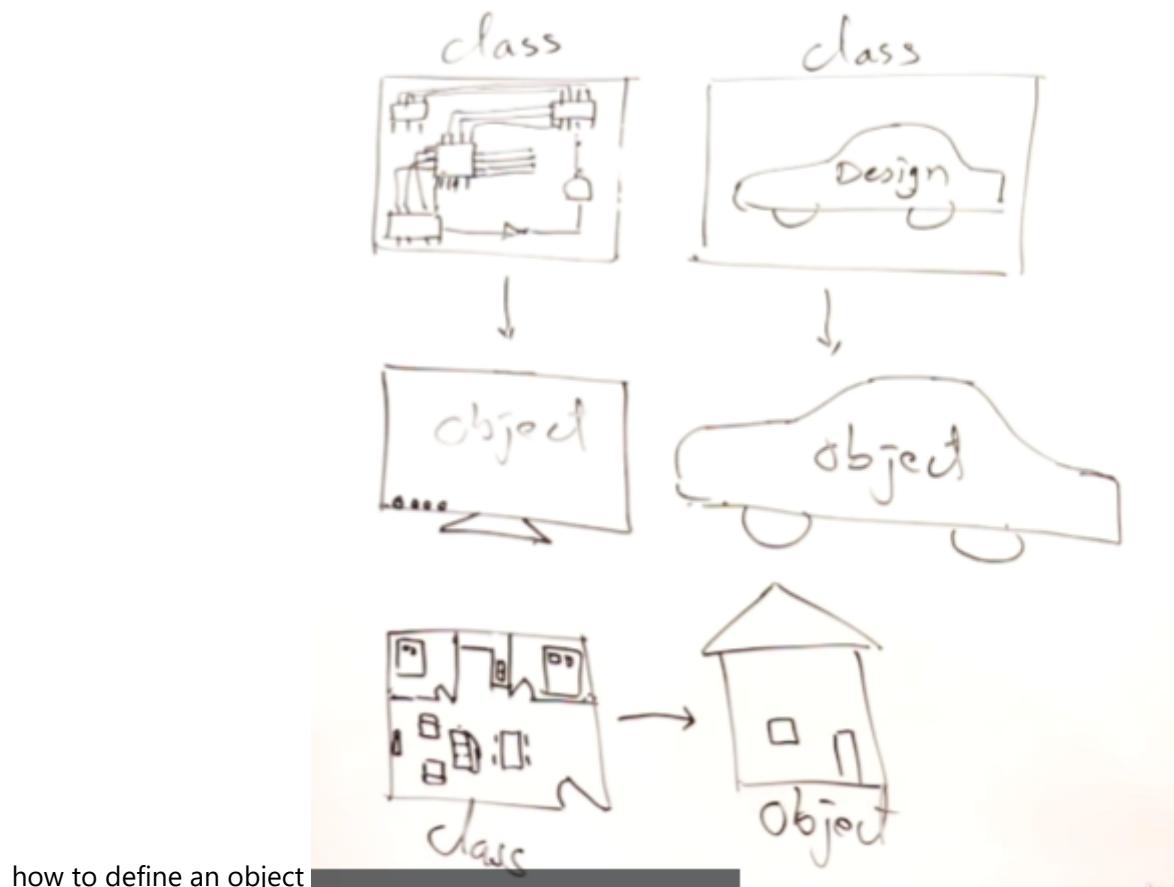
abstraction = hiding the complexity and showing the essential things encapsulation = wrapping the data and the methods into a single unit inheritance = acquiring the properties of the parent class polymorphism = one name and multiple forms

## 105. class vs object

class is the blueprint/template/collection of the object object is the instance of the class

any thing in the world is defined in terms of properties and behaviours properties = variables / fields / data members (nouns and adjectives) behaviours = methods / functions / operations (verbs)

class is the collection of the properties and the behaviours



how to define an object

```
class Television
{
    private int channel;
    private int volume;
    public void changeChannel()
    {
        public void changeVolume()
    }
}

class Test
{
    P.S.V. main()
    {
        Television t=new Television();
        :
    }
}
```

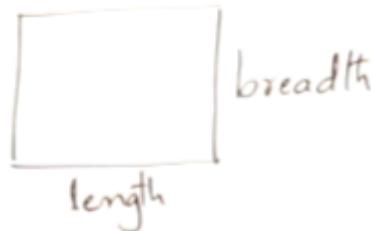


## 106. How to write the class

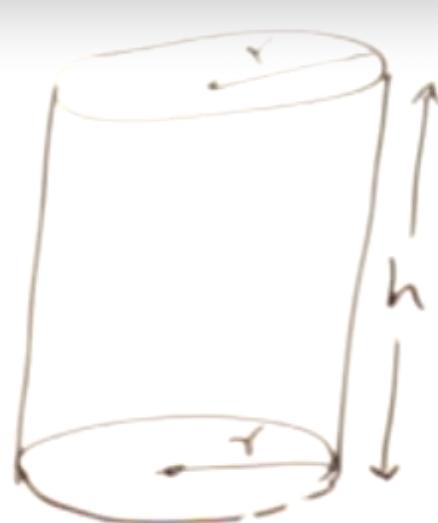
1. Circle
2. Rectangle
3. Cylinder
4. Student
5. Account
6. Car
7. Television

```
class Circle
{
    public double radius;
    public double area()
    {
        return 3.14 * radius * radius;
    }
    public double perimeter()
    {
        return 2 * 3.14 * radius;
    }
}
```

computations are made as the methods



```
class Rectangle  
{  
    public int length;  
    public int breadth;  
  
    public int area()  
    {  
        return length * breadth;  
    }  
  
    public int perimeter()  
    {  
        return 2 * (length + breadth);  
    }  
}
```



```
class Cylinder  
{
```

```
    public double radius;  
    public double height;
```

```
public double lidArea()
{
    public double surfaceArea()
    {
        public double volume()
    }
}
```



```
class Student
{
    public int rollno;
    public String name;
    public String course;
    public int m1, m2, m3;

    public int total()
    {
        return m1 + m2 + m3;
    }

    public float average()
    {
        return (m1 + m2 + m3) / 3;
    }
}
```

```
class Account
{
    public long accNo;
```

```
public String name;  
public double balance;  
  
public void deposit(int amt)  
{  
    —  
}  
  
public void withdraw(int amt)  
{  
    —  
}  
}
```

1. public double deposit(double amt)

—

}

2. public double withdraw(double amt)

—

}

class Car

{

    public String name;

    public String regNo;

    public Color col;

    public double fuelQty;

    public void start(){}..}

    public void stop(){}..}

    public void changeGear(){}..}

    public void accelerate(){}..}

    :  
}

class Television

{

    public String name;

    public int currNo;

    public int volume;

    public void switchON(){}..}

    public void switchOff(){}..}

    public void changeChannel(){}..}

    public void increaseVolume(){}..}

    :  
}

```
package java1;

class Circle
{
    public double radius;
```

```
public double area()
{
    return Math.PI*radius*radius;
}
public double perimeter()
{
    return 2*Math.PI*radius;
}
public double circumference()
{
    return perimeter();
}

}

public class Java1 {

    public static void main(String[] args) {
        Circle c1=new Circle();
        c1.radius=7;
        System.out.println("Area:"+c1.area());
        System.out.println("Perimeter:"+c1.perimeter());
        System.out.println("Circumference:"+c1.circumference());
    }
}
```

for every class java has to create a separate file

```
package rectangletest;

class Rectangle
{
    public double length;
    public double breadth;

    public double area()
    {
        return length*breadth;
    }

    public double perimeter()
    {
        return 2*(length+breadth);
    }

    public boolean isSquare()
    {
        if(length==breadth)
            return true;
        else
            return false;
    }
}
```

```
    }

}

public class RectangleTest {

    public static void main(String[] args) {
        Rectangle r=new Rectangle();
        r.length=10.5;
        r.breadth=5.5;

        System.out.println("Area"+r.area());
        System.out.println("perimeter"+r.perimeter());

        System.out.println("Is it a Square"+r.isSquare());
    }

}
```

```
package cylindertest;

class Cylinder
{
    private int radius;
    private int height;

    public Cylinder()
    {
        radius=height=1;
    }
    public Cylinder(int r,int h)
    {
        radius=r;
        height=h;
    }
    public int getHeight()
    {
        return height;
    }
    public int getRadius()
    {
        return radius;
    }

    public void setHeight(int h)
    {
        if(h>=0)
            height=h;
        else
            height=0;
    }
}
```

```
public void setRadius(int r)
{
    if(r>=0)
        radius=r;
    else
        radius=0;
}
public void setDimensions(int h,int r)
{
    height=h;
    radius=r;
}

public double lidArea()
{
    return Math.PI*radius*radius;
}
public double perimeter()
{
    return 2*Math.PI*radius;
}
public double drumArea()
{
    return 2*lidArea()+perimeter()*height;
}
public double volume()
{
    return lidArea()*height;
}
}

public class CylinderTest {

    public static void main(String[] args) {
        Cylinder c=new Cylinder();
        c.setHeight(10);
        c.setRadius(7);
        c.setDimensions(10, 7);

        System.out.println("LidArea "+c.lidArea());
        System.out.println("Circumference "+c.perimeter());
        System.out.println("totalSurfaceArea "+c.drumArea());
        System.out.println("Volume "+c.volume());
        System.out.println("Height"+c.getHeight());
        System.out.println("Radius"+c.getRadius());

    }
}
```

```
package scoops3;

class Subject
{
    private String subID;
    private String name;
    private int maxMarks;
    private int marksObtain;

    public Subject(String subID, String name, int maxMarks)
    {
        this.subID=subID;
        this.name=name;
        this.maxMarks=maxMarks;
    }
    public String getSubID(){return subID;}
    public String getName(){return name;}
    public int getMaxMarks(){return maxMarks;}
    public int getMarksObtain(){return marksObtain;}

    public void setMaxMarks(int mm)
    {
        maxMarks=mm;
    }
    public void setMarksObtain(int m)
    {
        marksObtain=m;
    }
    boolean isQualified()
    {
        return marksObtain>=maxMarks/10*4;
    }
    public String toString()
    {
        return "\nSubject ID:"+subID+"\nName :" +name+ "\nMarks Obtained:"+
        "+marksObtain;
    }
}

class Student
{
    private String rollNo;
    private String name;
    private String dept;
    private int numOfSub;
    private Subject sub[];

    public Student(String roll, String name)
    {
        this.rollNo=roll;
        this.name=name;
```

```
    }
    public Student(String roll, String name, int ns)
    {
        this.rollNo=roll;
        this.name=name;
        this.numOfSub=ns;
    }

    public String getRollNo(){return rollNo;}
    public String getName(){return name;}
    public String getDept(){return dept;}
    public int getNoOfSubjects(){return numOfSub;}
    public Subject[] getSubjects(){return sub;}

    public void setDept(String dept)
    {
        this.dept=dept;
    }
    public void setSubjects(Subject ...subs)
    {
        for(int i=0;i<subs.length;i++)
            sub[i]=subs[i];
    }

    public String toString()
    {
        return "Roll :" + rollNo + "\nName :" + name + "\nDept :" + dept;
    }
}

public class SCoops3
{

    public static void main(String[] args)
    {
        Subject subs[] = new Subject[3];
        subs[0] = new Subject("s101", "DS", 100);
        subs[1] = new Subject("s102", "Algorithms", 100);
        subs[2] = new Subject("s103", "Operating Systems", 100);

        for(Subject s:subs)
            System.out.println(s);
    }
}
```

```
package studenttest;
```

```
class Student
{
    public int roll;
    public String name;
    public String course;
    public int m1,m2,m3;

    public int total()
    {
        return m1+m2+m3;
    }
    public float average()
    {
        return (float)total()/3;
    }
    public char grade()
    {
        if(average()>=60)
            return 'A';
        else
            return 'B';
    }
    public String toString()
    {
        return "Roll No:"+roll+"\n"+ "Name:" +name+ "\n" + "Course:" +course+ "\n";
    }
}

public class StudentTest
{
    public static void main(String[] args)
    {
        Student s=new Student();

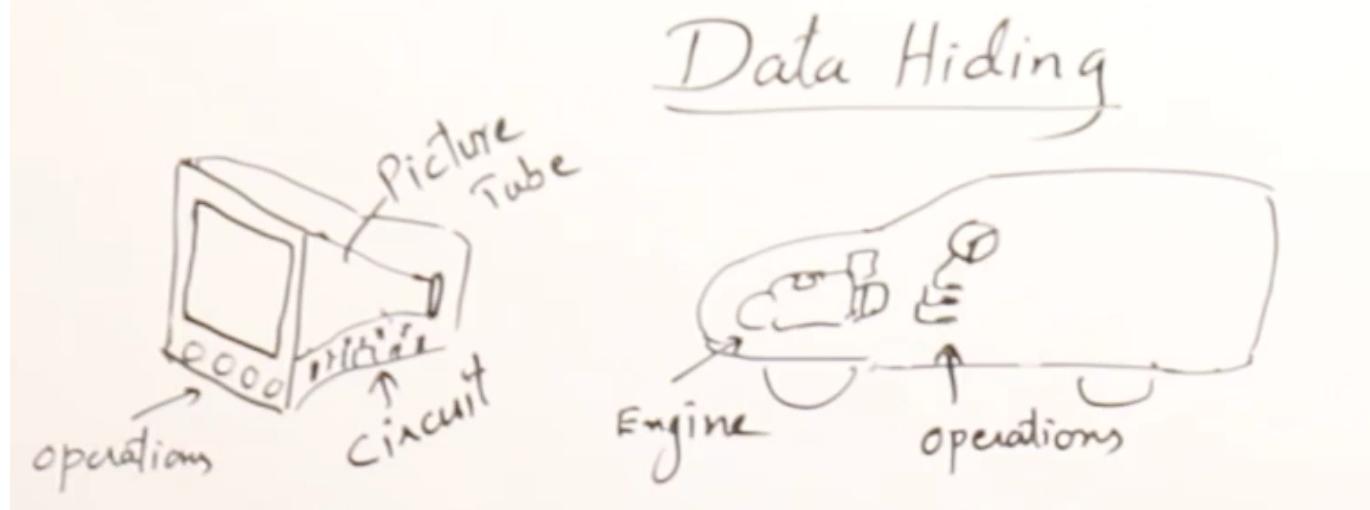
        s.roll=1;
        s.name="John";
        s.course="CS";
        s.m1=70;
        s.m2=80;
        s.m3=65;

        System.out.println("Total :" +s.total());
        System.out.println("Average :" +s.average());

        System.out.println("Details:\n "+ s );
    }
}
```

## Data Hiding

Encapsulation = data hiding + abstraction  
Encapsulation = wrapping the data and the methods into a single unit  
Data hiding = hiding the data from the outside world  
abstraction = hiding the complexity and showing the essential things



class Rectangle

```
private int length;  
private int breadth;
```

```
public int area()  
{  
    return length*breadth;  
}
```

```
public int perimeter()  
{  
    return 2*(length+breadth);  
}
```

Rectangle r=new Rectangle();  
X r.length = 10;  
X r.breadth = 5;

class Rectangle  
length  
breadth  
d  
); read { int getLength()  
; return length;  
write { void setLength(int l)  
; length=l;  
getXXX()  
setXXX()

class Rectangle  
private int length;  
private int breadth;  
int getLength()  
; return length;  
void setLength(int l)  
if(l>0)  
length=l;  
else length=0;

```
package rectangletest1;

class Rectangle
{
    private double length;
    private double breadth;

    public double getLength()
    {
        return length;
    }

    public double getBreadth()
    {
        return breadth;
    }

    public void setLength(double l)
    {
        if(l>=0)
            length=l;
        else
            length=0;
    }

    public void setBreadth(double b)
    {
        if(b>=0)
            breadth=b;
        else
            breadth=0;
    }

    public double area()
    {
        //return length*breadth;
        return getLength()*getBreadth();
    }

    public double perimeter()
    {
        return 2*(length+breadth);
    }

    public boolean isSquare()
    {
        if(length==breadth)
            return true;
        else
            return false;
    }
}
```

```
public class RectangleTest1 {  
  
    public static void main(String[] args) {  
        Rectangle r=new Rectangle();  
        r.setLength(10.5); //check with negative values.  
        r.setBreadth(5.5);  
  
        System.out.println("Area "+r.area());  
        System.out.println("Perimeter "+r.perimeter());  
        System.out.println("Is Square "+r.isSquare());  
  
        System.out.println("Length "+r.getLength());  
        System.out.println("Breadth "+r.getBreadth());  
    }  
  
}
```

```
public double area()  
{  
    return getLength()*getBreadth();  
}
```

113.



```
class Rectangle
{
    private double length;
    → getLength()
    ← setLength(l)
```

1. (90% of properties)

```
class Student
{
    private int m1;
    → getM1()
    → setM1(m)
```

```
class Student
{
    private int roll;
    → int getRoll()
    ← setRoll(r)
```

class Account
{
 private int accno;
 → getAccNumber()
 ← set

2. Read only

```
class Producer
{
    private int sharedData;

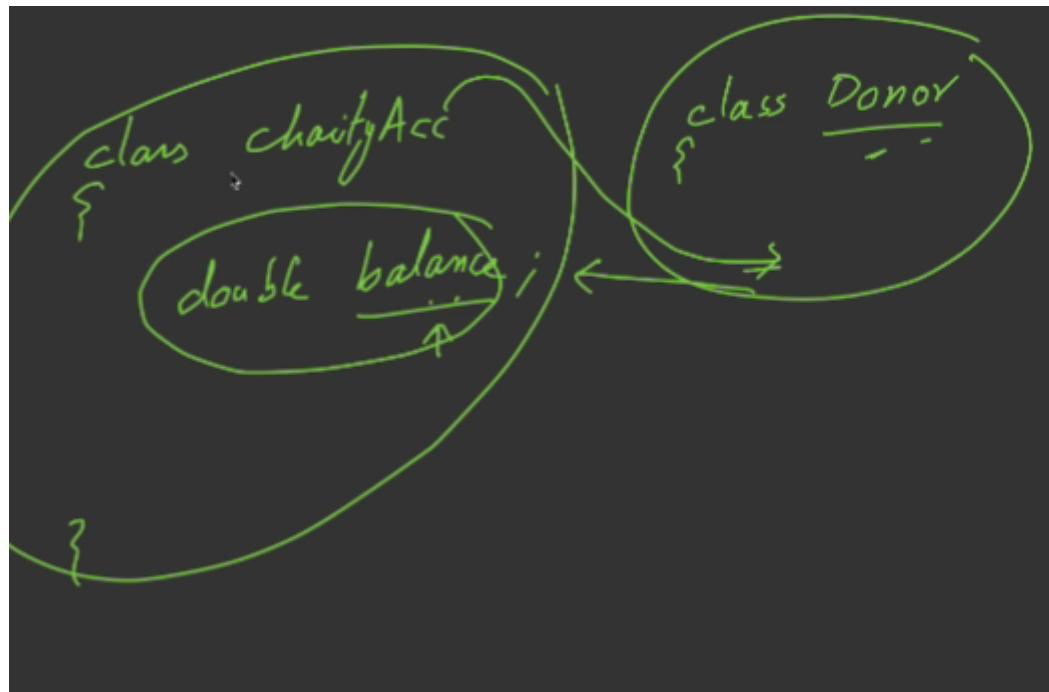
    public void setData(int d)
    {
        sharedData=d;
    }
}
```

3. Write only

thread or two classes are working on the same data producer and consumer (set and get) eg. Charity

rare two

### Acc and Donor ACC



(internally inside

the software write only property is used.)

## 114. Constructor

= special type of method which is used to initialize the object

- name of the constructor is same as the class name
- constructor doesn't have any return type
- constructor is called automatically when the object is created
- constructor is used to initialize the object
- constructor is used to allocate the memory for the object
- constructor is used to initialize the properties of the object
- constructor is used to initialize the default values for the properties
- default constructor is provided by the java
- default constructor is used to initialize the default values for the properties

## 115.

```
package rectangletest;

class Rectangle
{
    public double length;
    public double breadth;
```

```
public double area()
{
    return length*breadth;
}

public double perimeter()
{
    return 2*(length+breadth);
}

public boolean isSquare()
{
    if(length==breadth)
        return true;
    else
        return false;
}

}

public class RectangleTest {

    public static void main(String[] args) {
        Rectangle r=new Rectangle();
        r.length=10.5;
        r.breadth=5.5;

        System.out.println("Area"+r.area());
        System.out.println("perimeter"+r.perimeter());

        System.out.println("Is it a Square"+r.isSquare());
    }

}
//write a constructor for the above code
```

```
package rectangletest;

class Rectangle
{
    public double length;
    public double breadth;

    public Rectangle()
    {
        length=1;
        breadth=1;
    }

    public double area()
    {
```

```
        return length*breadth;
    }

    public double perimeter()
    {
        return 2*(length+breadth);
    }

    public boolean isSquare()
    {
        if(length==breadth)
            return true;
        else
            return false;
    }
}

public class RectangleTest {

    public static void main(String[] args) {
        Rectangle r=new Rectangle();
        r.length=10.5;
        r.breadth=5.5;

        System.out.println("Area"+r.area());
        System.out.println("perimeter"+r.perimeter());

        System.out.println("Is it a Square"+r.isSquare());
    }
}
```

## parameterized constructor

```
package rectangletest;

class Rectangle
{
    public double length;
    public double breadth;

    public Rectangle()
    {
        length=1;
        breadth=1;
    }

    public Rectangle(double l,double b)
    {
        length=l;
        breadth=b;
    }
}
```

```
}

//instance methods / facilitators
public double area()
{
    return length*breadth;
}

public double perimeter()
{
    return 2*(length+breadth);
}

//validator / inspector methods
public boolean isSquare()
{
    if(length==breadth)
        return true;
    else
        return false;
}

}

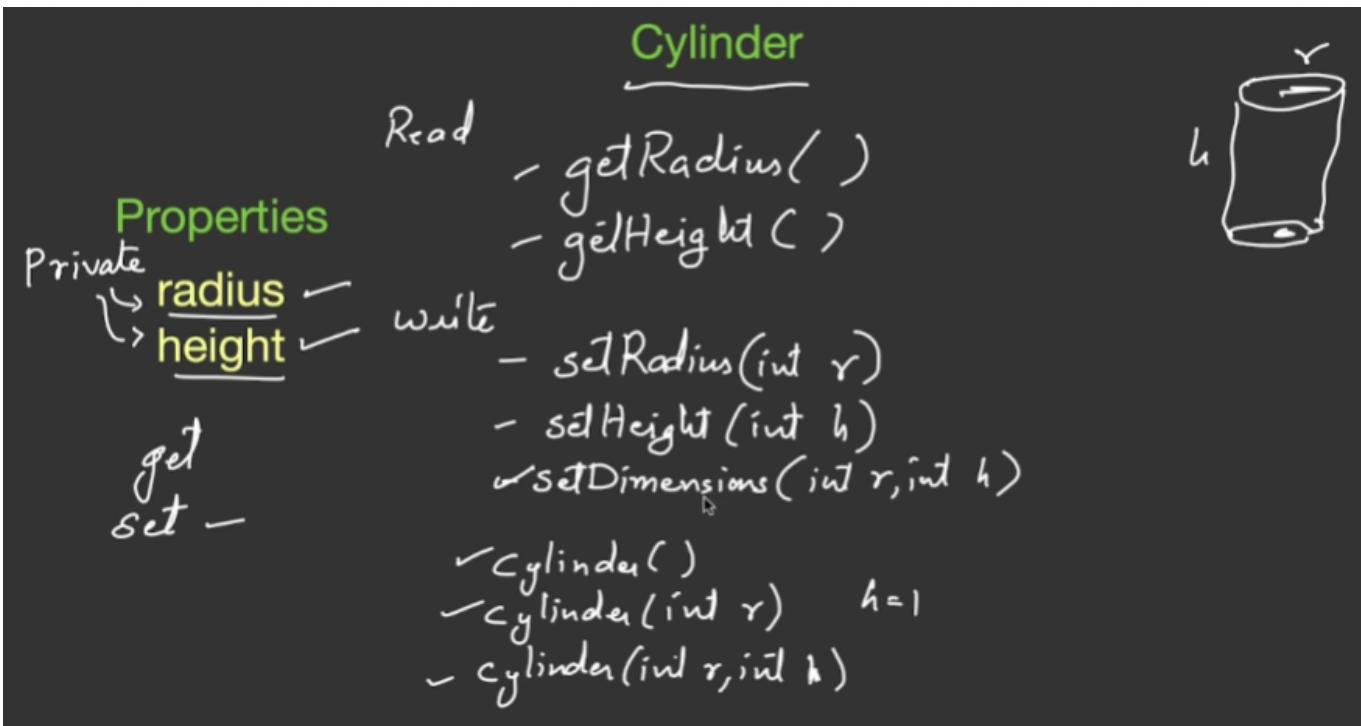
public class RectangleTest {

    public static void main(String[] args) {
        Rectangle r=new Rectangle(10.5,5.5);

        System.out.println("Area"+r.area());
        System.out.println("perimeter"+r.perimeter());

        System.out.println("Is it a Square"+r.isSquare());
    }

}
```



```

package cylindertest;

class Cylinder
{
    private int radius;
    private int height;

    public Cylinder()
    {
        radius=height=1;
    }
    public Cylinder(int r,int h)
    {
        radius=r;
        height=h;
    }
    public int getHeight()
    {
        return height;
    }
    public int getRadius()
    {
        return radius;
    }

    public void setHeight(int h)
    {
        if(h>=0)
            height=h;
        else
            height=0;
    }
}

```

```
public void setRadius(int r)
{
    if(r>=0)
        radius=r;
    else
        radius=0;
}
public void setDimensions(int h,int r)
{
    height=h;
    radius=r;
}

public double lidArea()
{
    return Math.PI*radius*radius;
}
public double perimeter()
{
    return 2*Math.PI*radius;
}
public double drumArea()
{
    return 2*lidArea()+perimeter()*height;
}
public double volume()
{
    return lidArea()*height;
}
}

public class CylinderTest {

    public static void main(String[] args) {
        Cylinder c=new Cylinder();
        c.setHeight(10);
        c.setRadius(7);
        c.setDimensions(10, 7);

        System.out.println("LidArea "+c.lidArea());
        System.out.println("Circumference "+c.perimeter());
        System.out.println("totalSurfaceArea "+c.drumArea());
        System.out.println("Volume "+c.volume());
        System.out.println("Height"+c.getHeight());
        System.out.println("Radius"+c.getRadius());

    }
}
```

Product

string itemno  
 string name  
 double price  
 → short qty

(A25-76)

property Methods

string getItemNo()  
 string getName()

~~setItemNo(string itn)~~  
 setPrice(---)

constructor

Product(string itemno),  
 Product(string itemno, string name)

Customer

String custId  
 " name  
 " address  
 " phno

+91

0597 - ..

property Methods

get

=

void setAddress(---)

void setPhno(---)

construction

Customer(custId, name)

Customer(custId, name, address, phone)

```
package scoops2;

class Product
{
    private String itemNo;
    private String name;
    private double price;
    private short qty;

    public Product(String itemno)
    {
        itemNo=itemno;
    }
    public Product(String itemno,String name)
    {
        itemNo=itemno;
        this.name=name;
    }
    public Product(String itemno,String name,double price,short qty)
    {
        itemNo=itemno;
        this.name=name;
        setPrice(price);
        setQuantity(qty);
    }

    public String getItemNo(){return itemNo;}
    public String getName(){return name;}
    public double getPrice(){return price;}
    public short getQuantity(){return qty;}

    public void setPrice(double price)
    {
        this.price=price;
    }
    public void setQuantity(short qty)
    {
        this.qty=qty;
    }
}

class Customer
{
    private String custId;
    private String name;
    private String address;
    private String phno;

    public Customer(String custId,String name)
    {
        this.custId=custId;
        this.address=name;
    }
}
```

```
        }
        public Customer(String custId, String name, String address, String phno)
        {
            this.custId=custId;
            this.address=name;
            setAddress(address);
            setPhno(phno);
        }

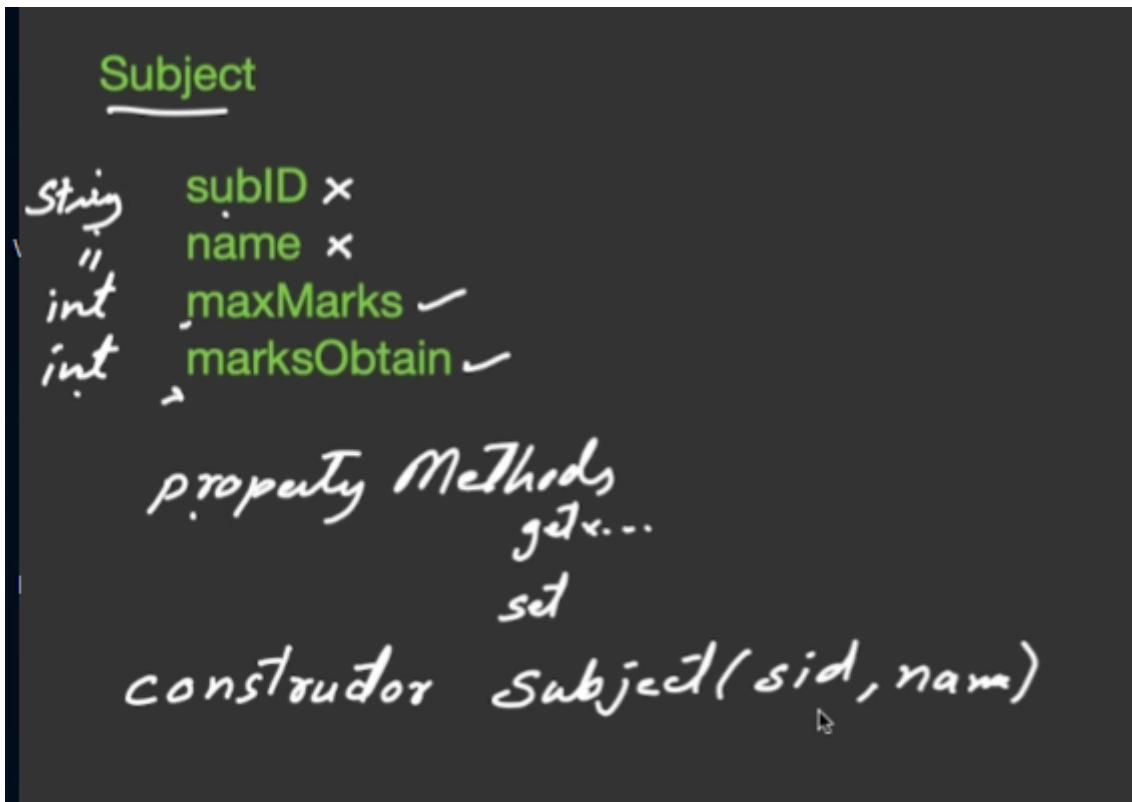
        public String getCustId(){return custId;}
        public String getName(){return name;}
        public String getAddress(){return address;}
        public String getPhno(){return phno;}

        public void setAddress(String address)
        {
            this.address=address;
        }
        public void setPhno(String phno)
        {
            this.phno=phno;
        }
    }

    public class SCoops2
    {
        public static void main(String[] args)
        {
        }
    }
}
```

## 118. Array of objects





this = reference

to the current object

```
package scloops;

class Subject
{
    private String subID;
    private String name;
    private int maxMarks;
    private int marksObtains;

    public Subject(String subID, String name, int maxMarks)
    {
        this.subID=subID;
        this.name=name;
        this.maxMarks=maxMarks;
    }

    public String getSubID(){return subID;}
    public String getName(){return name;}
    public int getMaxMarks(){return maxMarks;}
    public int getMarksObtains(){return marksObtains;}

    public void setMaxMarks(int mm)
    {
        maxMarks=mm;
    }

    public void setMarksObtain(int m)
    {
        marksObtains=m;
    }
}
```

```
}

boolean isQualified()
{
    return marksObtains>=maxMarks/10*4;
}

public String toString()
{
    return"\n SubjectID: "+subID+"\n Name "+name+"\n MarksObtained
"+marksObtains;
}

}

public class SCLoops {

    public static void main(String[] args)
    {
        Subject subs[]=new Subject[3];
        subs[0]=new Subject("s101","DS",100);
        subs[1]=new Subject("s102","Algorithms",100);
        subs[2]=new Subject("s103","Operating Systems",100);

        for(Subject s:subs)
            System.out.println(s);
    }
}
```

```
class Subject
{
    private String subID;
    private String name;
    private int maxMarks;
    private int marksObtains;

    public Subject(String subID,String name,int maxMarks)
    {
        this.subID=subID;
        this.name=name;
        this.maxMarks=maxMarks;
    }

    public String getSubID(){return subID;}
    public String getName(){return name;}
    public int getMaxMarks(){return maxMarks;}
    public int getMarksObtains(){return marksObtains;}

    public void setMaxMarks(int mm)
```

```
{  
    maxMarks=mm;  
}  
  
public void setMarksObtain(int m)  
{  
    marksObtains=m;  
}  
  
boolean isQualified()  
{  
    return marksObtains>=maxMarks/10*4;  
}  
  
public String toString()  
{  
    return"\n SubjectID: "+subID+"\n Name "+name+"\n MarksObtained  
"+marksObtains;  
}  
  
}  
  
public class Main {  
  
    public static void main(String[] args)  
    {  
        Subject subs[]=new Subject[3];  
        subs[0]=new Subject("s101","DS",100);  
        subs[1]=new Subject("s102","Algorithms",100);  
        subs[2]=new Subject("s103","Operating Systems",100);  
  
        // Create student object  
        Student student = new Student("19TU1A0508", "SidduGansh", "CSE");  
  
        // Enroll student in subjects (assuming student can take all three)  
        for (Subject subject : subs) {  
            student.enroll(subject);  
        }  
  
        // Set marks for some subjects (assuming you have information)  
        student.setMarks("s101", 90); // Set marks for DS  
        student.setMarks("s103", 85); // Set marks for Operating Systems  
        student.setMarks("s102", 85);  
  
        // Print student information  
        System.out.println(student);  
  
        // Additional calculations (optional)  
        System.out.println("Total Marks: " + student.getTotalMarks());  
        System.out.println("Overall Percentage: " + student.getOverallPercentage() +  
        "%");  
        System.out.println("Passed: " + student.isPassed());  
    }  
}
```

```
        }
    }

    class Student {

        private String studentID;
        private String name;
        private String department;
        private Subject[] enrolledSubjects;

        public Student(String studentID, String name, String department) {
            this.studentID = studentID;
            this.name = name;
            this.department = department;
            this.enrolledSubjects = new Subject[0]; // Initialize empty array
        }

        public String getStudentID() {
            return studentID;
        }

        public String getName() {
            return name;
        }

        public String getDepartment() {
            return department;
        }

        public Subject[] getEnrolledSubjects() {
            return enrolledSubjects; // Consider returning a copy if mutability is a concern
        }

        // Method to enroll student in a subject (assuming no duplicates)
        public void enroll(Subject subject) {
            Subject[] updatedSubjects = new Subject[enrolledSubjects.length + 1];
            System.arraycopy(enrolledSubjects, 0, updatedSubjects, 0,
            enrolledSubjects.length);
            updatedSubjects[updatedSubjects.length - 1] = subject;
            enrolledSubjects = updatedSubjects;
        }

        // Method to set marks for a specific subject enrolled by the student
        public void setMarks(String subjectID, int marks) {
            for (Subject subject : enrolledSubjects) {
                if (subject.getSubID().equals(subjectID)) {
                    subject.setMarksObtain(marks);
                    return; // Marks set, exit the loop
                }
            }
            System.out.println("Subject with ID " + subjectID + " not found for student " +
+ studentID);
        }
    }
}
```

```
// Calculate total marks obtained by the student across all subjects
public int getTotalMarks() {
    int totalMarks = 0;
    for (Subject subject : enrolledSubjects) {
        totalMarks += subject.getMarksObtains();
    }
    return totalMarks;
}

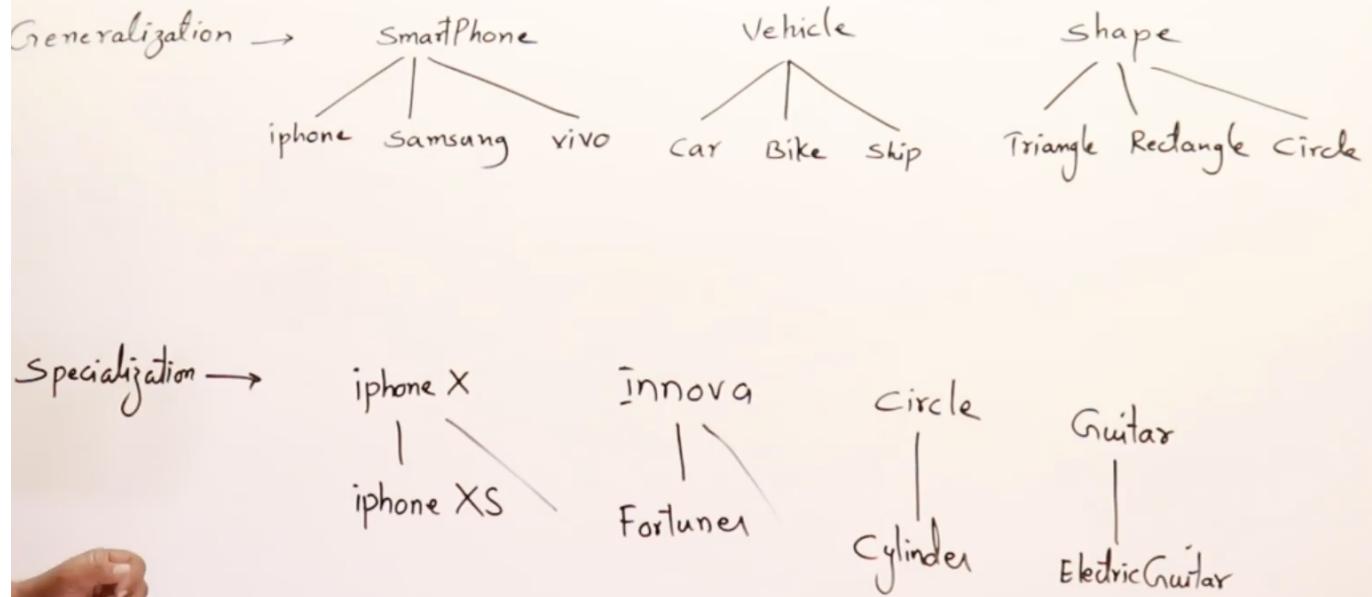
// Calculate overall percentage (assuming all subjects have same weightage)
public double getOverallPercentage() {
    int totalMarks = getTotalMarks();
    int totalMaxMarks = 0;
    for (Subject subject : enrolledSubjects) {
        totalMaxMarks += subject.getMaxMarks();
    }
    if (totalMaxMarks == 0) {
        return 0.0; // Avoid division by zero
    }
    return (double) totalMarks / totalMaxMarks * 100;
}

// Check if student is passed (based on your definition in Subject.isQualified)
public boolean isPassed() {
    for (Subject subject : enrolledSubjects) {
        if (!subject.isQualified()) {
            return false;
        }
    }
    return true;
}

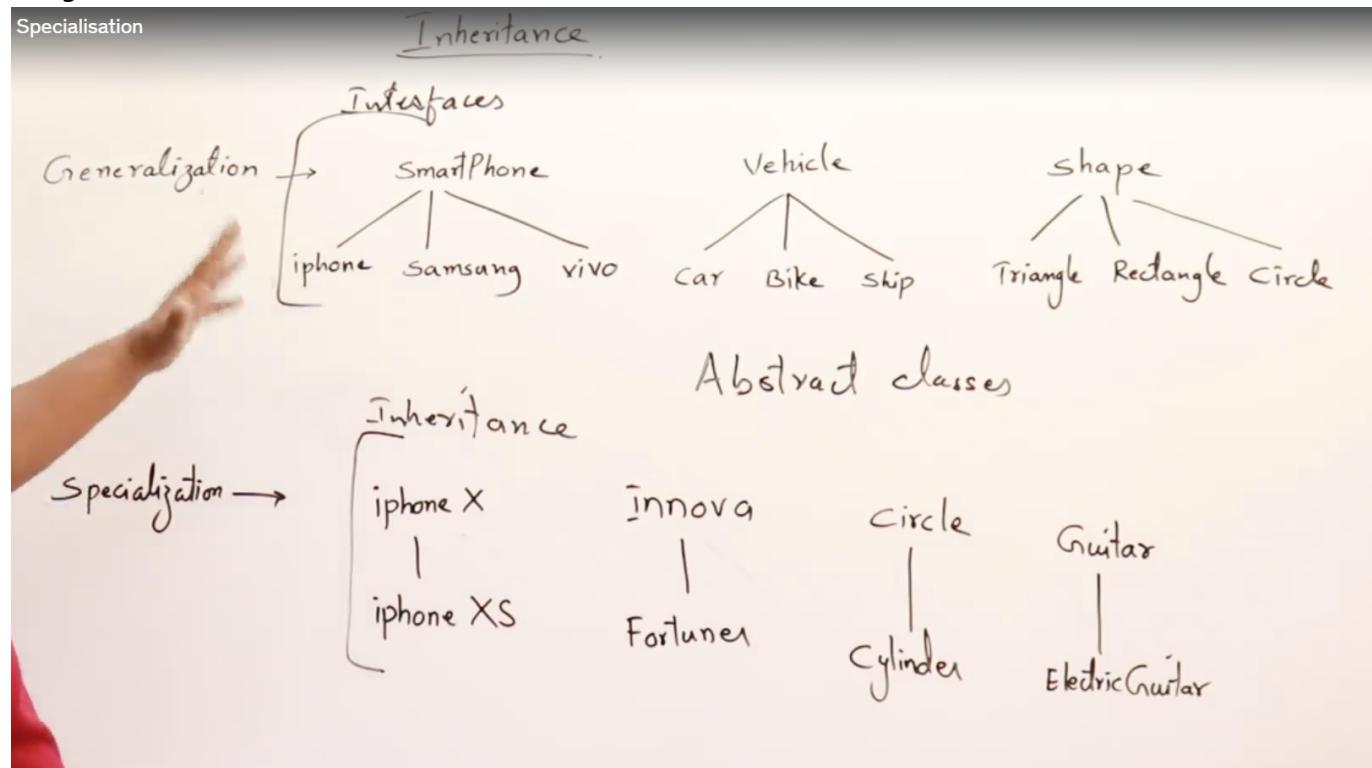
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("\nStudent ID: ").append(studentID).append("\n");
    sb.append("Name: ").append(name).append("\n");
    sb.append("Department: ").append(department).append("\n");
    sb.append("Enrolled Subjects:\n");
    for (Subject subject : enrolledSubjects) {
        sb.append(subject).append("\n");
    }
    return sb.toString();
}
```

## Section 12: Inheritance

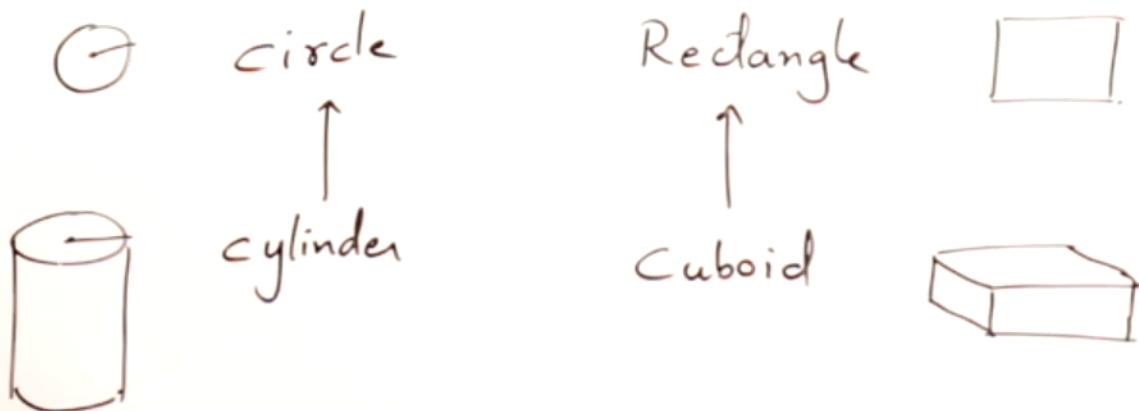
---



generalization means bottom up is the hierarchy designed. specialization means top down is the hierarchy designed.



inheritance = acquiring the properties of the parent class



- properties
- Methods

class Circle

private double radius;

```

public Circle()
{
    radius=0.0;
}
  
```

```

public double area();
public double perimeter();
  
```

}

class Cylinder extends Circle

private double height;

```

public Cylinder()
{
    height=0.0;
}
  
```

```

public double Volume();
  
```

}

class Test

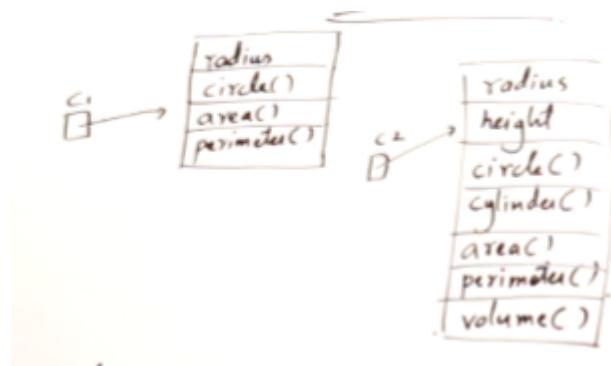
```

p.s.v.main()
{
  
```

Circle c1=new Circle();

Cylinder c2=new Cylinder();

}



```
package circle1;

class Circle
{
    public double radius;

    public double area()
    {
        return Math.PI * radius * radius;
    }

    public double perimeter()
    {
        return 2*Math.PI*radius;
    }
    public double circumference()
    {
        return perimeter();
    }
}

class Cylinder extends Circle
{
    public double height;

    public double volume()
    {
        return area()*height;
    }
}

public class Circle1
{
```

```

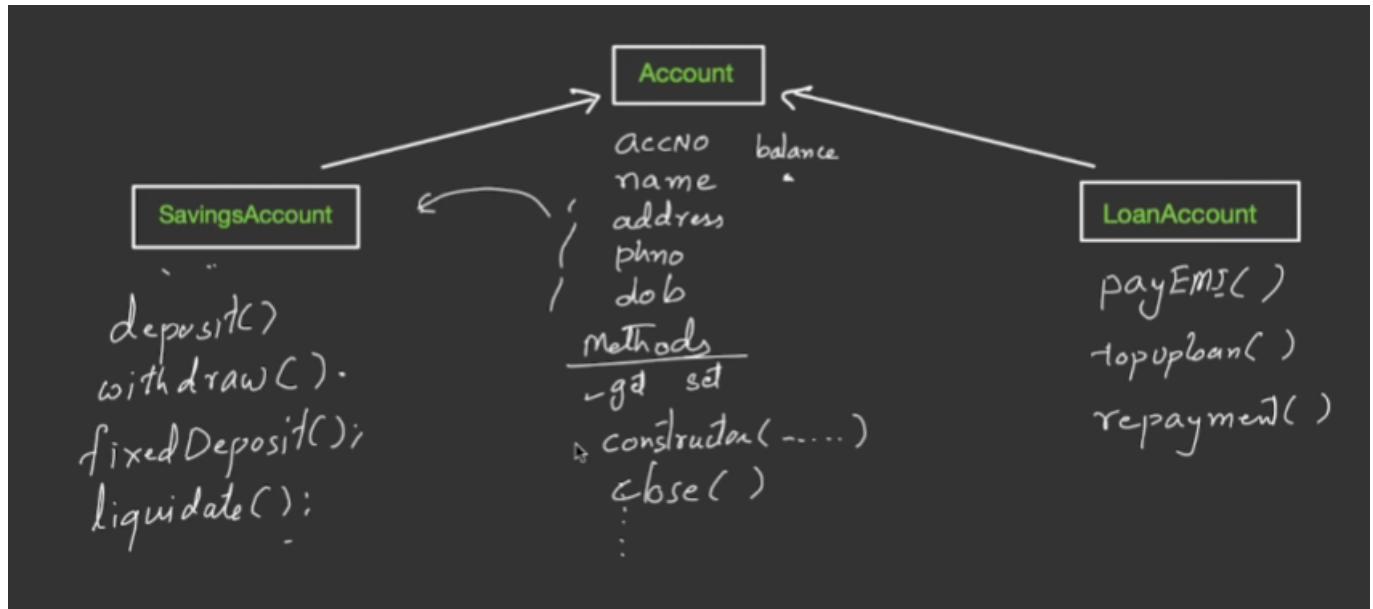
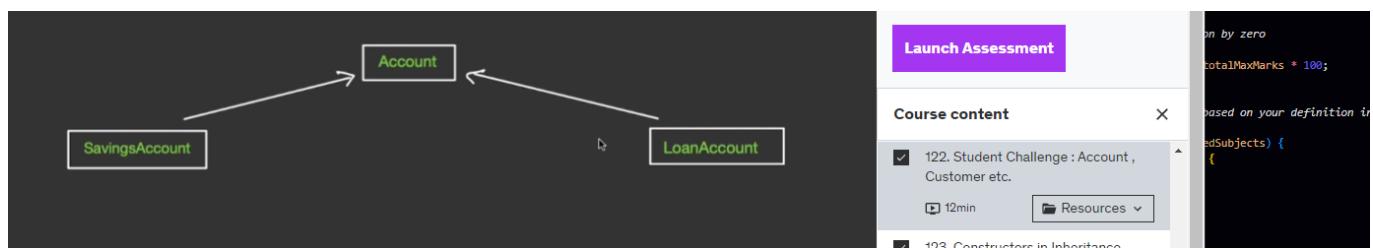
public static void main(String[] args)
{
    Cylinder c=new Cylinder();

    c.radius=7;
    c.height=10;

    System.out.println("Volume "+c.volume());
    System.out.println("Area "+c.area());

}

```



```

package scinherit;

class Account
{//closeAccount() , openAccount()
    private String accNo;
    private String name;
    private String address;
    private String phno;
    private String dob;
    protected long balance;

    public Account(String acc, String n, String add, String phno, String dob)
    {

```

```
accNo=acc;
name=n;
address=add;
this.phno=phno;
this.dob=dob;
balance=0;
}
public String getAccNo(){return accNo;}
public String getName(){return name;}
public String getAddress(){return address;};
public String getPhno(){return phno;}
public String getDOB(){return dob;}
public long getBalance(){return balance;}

public void setAddress(String add)
{
    address=add;
}
public void setPhno(String phno)
{
    this.phno=phno;
}

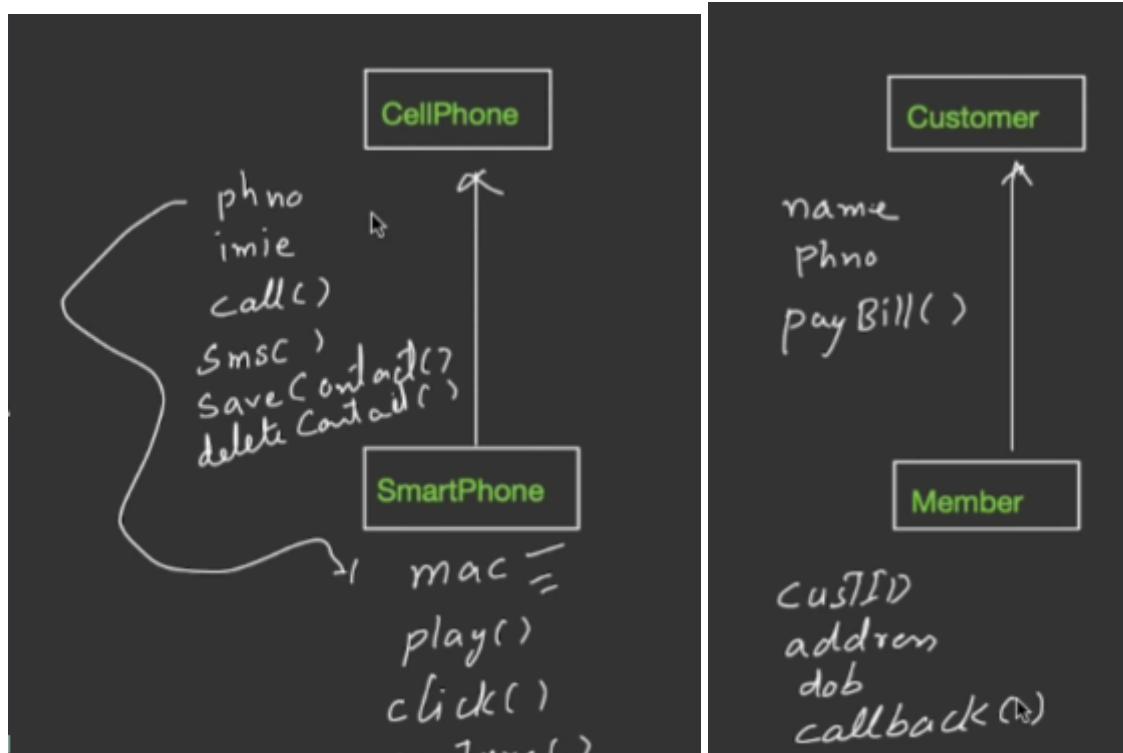
}

class SavingsAccount extends Account
{//fixedDeposit() , liquidate();
    public void deposit(long amt)
    {
        balance+=amt;
    }
    public void withdraw(long amt)
    {
        balance-=amt;
    }
}

class LoanAccount extends Account
{//topUpLoan()
    public void payEMI(long amt)
    {
        balance-=amt;
    }
    public void repay(long amt)
    {
        if(balance==amt)
            balance=0;
    }
}

public class SCIInherit
{
    public static void main(String[] args)
    {
```

}



```
package inheritconst;

class Parent
{
    public Parent()
    {
        System.out.println("Parent Constrcutor");
    }
}

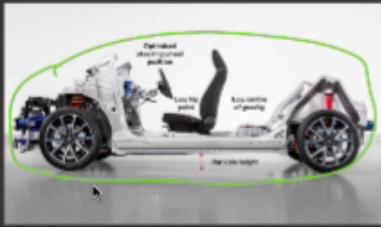
class Child extends Parent
{
    public Child()
    {
        System.out.println("Child Constructor");
    }
}

public class InheritConst
{
    public static void main(String[] args)
    {
        Child c=new Child();
    }
}
```

parent class constructor is called first and

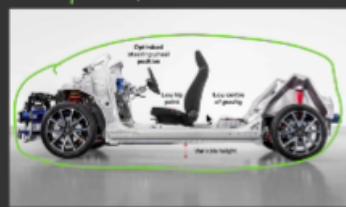
executed first then the child class constructor is called and executed.

platform



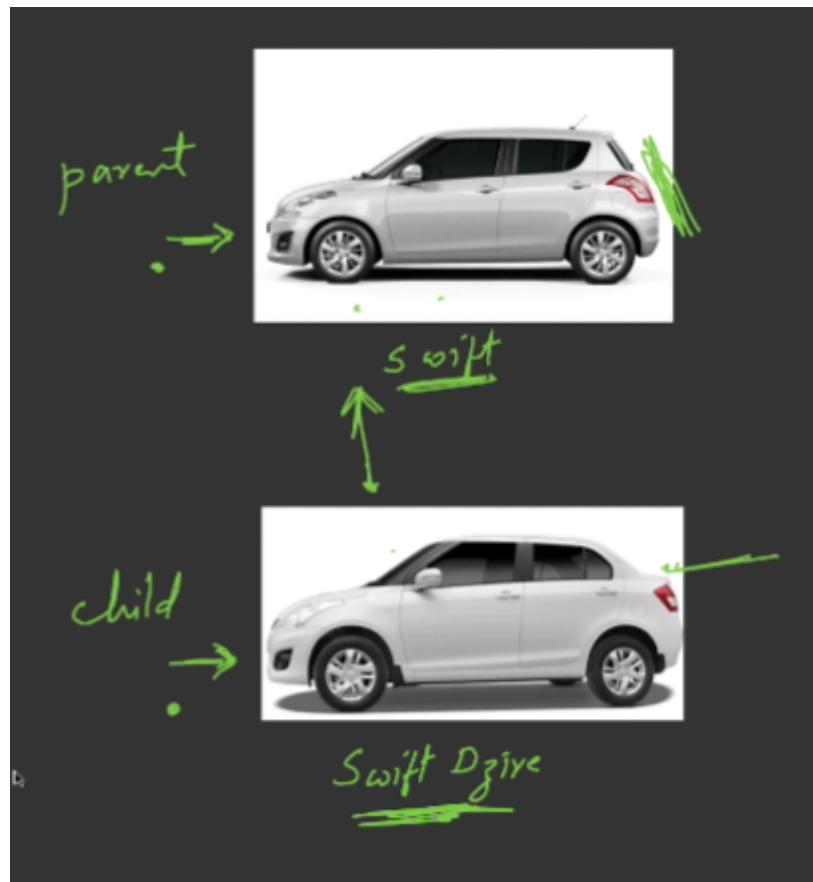
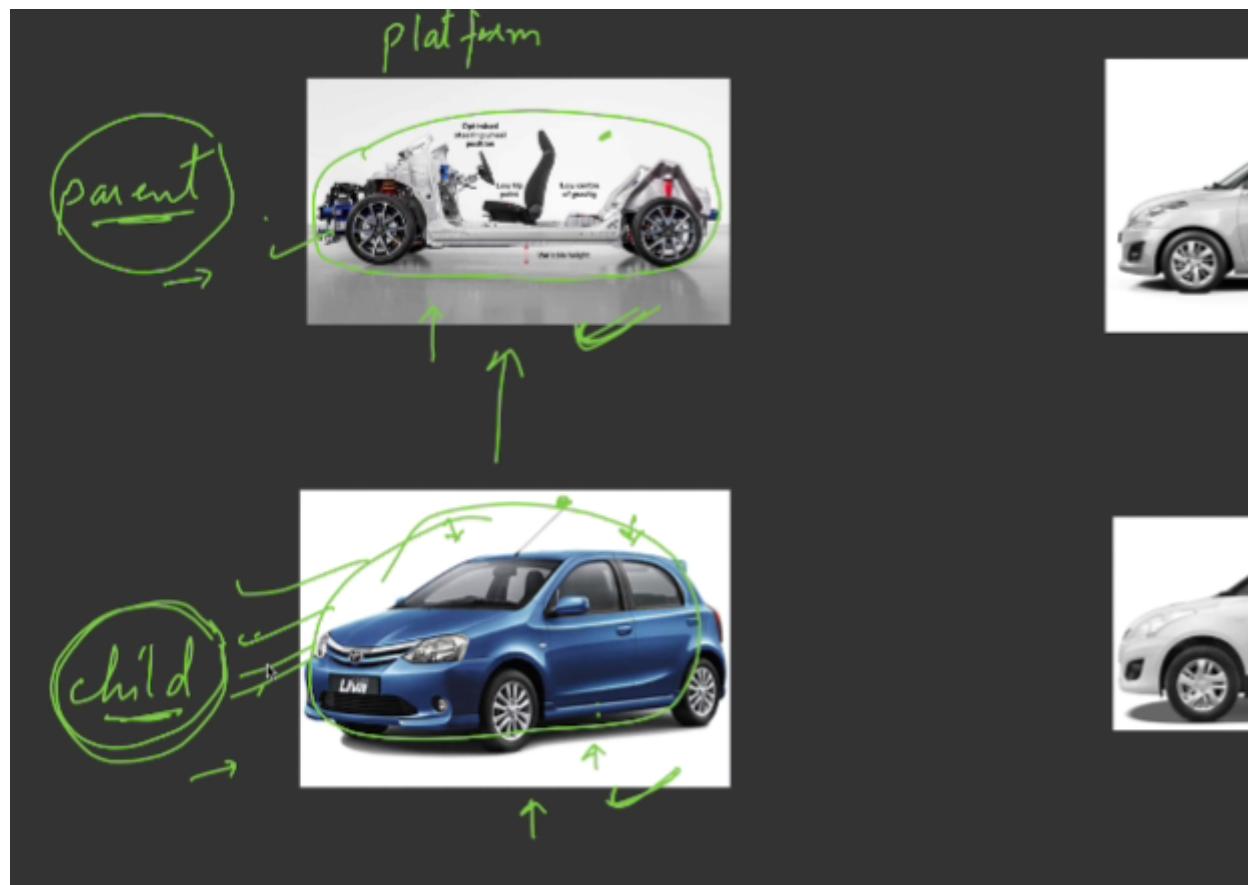
platform

parent



child





## 124. parameterized constructor

how to call the parameterized constructor of the parent class with child class using super() method.

```
package superconstr;

class Parent
{
    Parent()
    {
        System.out.println("Non-Param of parent");
    }
    Parent(int x)
    {
        System.out.println("Param of parent "+x);
    }
}

class Child extends Parent
{
    Child()
    {
        System.out.println("Non-Param of child");
    }
    Child(int y)
    {
        System.out.println("Param of child");
    }
    Child(int x,int y)
    {
        super(x);
        System.out.println("2 param of child "+y);
    }
}

public class SuperConstr {

    public static void main(String[] args) {
        //Child c=new Child();
        //Child c=new Child(20);
        Child c=new Child(10,20);
    }
}
```

```
class Cuboid extends Rectangle
{
    int height;

    Cuboid()
    {
        height=1;
    }

    Cuboid(int h)
    {
        height=h;
    }
    Cuboid(int l,int b,int h)
    {
        super(l,b);
        height=h;
    }

    int volume()
    {
        return length*breadth*height;
    }
}
```

## this and super

```
class Rectangle
{
    int length;
    int breadth;

    Rectangle(int l,int b)
    {
        this.length=l;
        this.breadth=b;
    }

    void display()
    {
        System.out.println("Length :" +this.length);

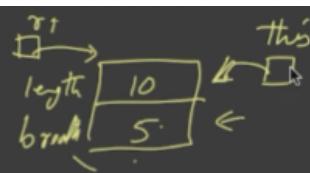
        System.out.println("Breadth :" +this.breadth);
    }
}
```

this is used to refer the current object

```
class Rectangle
{
    int length; ✓
    int breadth; ✓
    Rectangle(int l,int b)
    {
        this.length=l;
        this.breadth=b;
    }

    void display()
    {
        System.out.println("Length :" +this.length);
        System.out.println("Breadth :" +this.breadth);
    }
}
```

## this and super



Rectangle r1=new Rectangle(10,5);  
r1.display()

```
int length;
int breadth;

Rectangle(int length,int breadth)
{
    this.length=length;
    this.breadth=breadth;
}
```

this is used to resolve the ambiguity and name conflict

## this and super

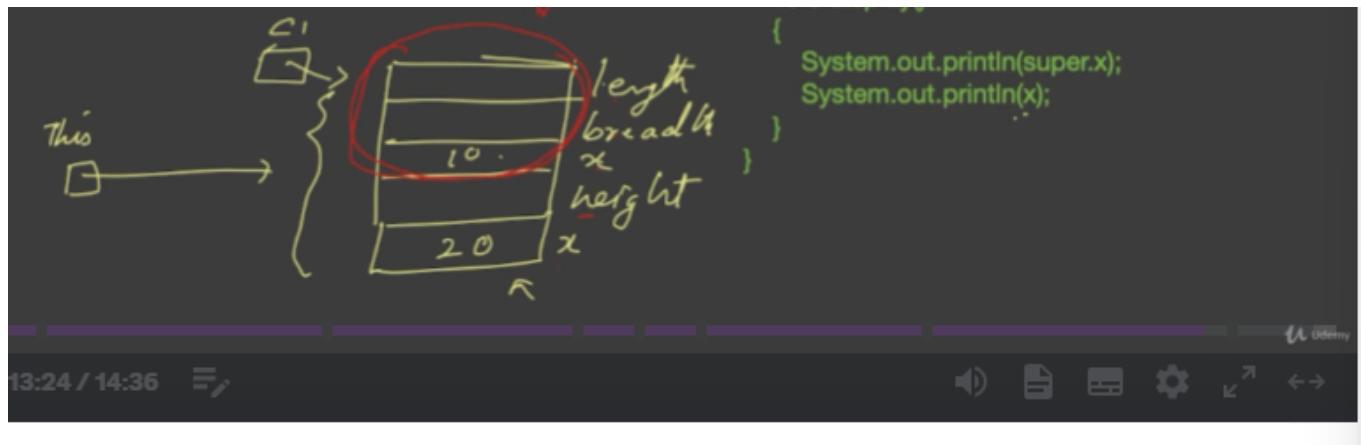
```
class Rectangle
{
    int length; ✓
    int breadth; ✓
    int x=10; ✓

    Rectangle(int length,int breadth) ✓
    {
        this.length=length;
        this.breadth=breadth;
    }
}
```

```
class Cuboid extends Rectangle
{
    → int height; ✓
    int x=20; ✓

    Cuboid(int l,int b,int h)
    {
        super(l,b);
        height=h; ✓
    }

    void display()
    {
        System.out.println(super.x);
        System.out.println(x);
    }
}
```



```
package inheritconst;

class Parent
{
    public Parent()
    {
        System.out.println("Parent Constrcutor");
    }
}

class Child extends Parent
{
    public Child()
    {
        System.out.println("Child Constructor");
    }
}

class GrandChild extends Child
{
    public GrandChild()
    {
        System.out.println("Grand Child Constructor");
    }
}

public class InheritConst
{

    public static void main(String[] args)
    {
        GrandChild c=new GrandChild();
    }
}
```

## 126. Method Overriding

= redefining the method in the child class

## Method Overriding

1. What is Method Overriding
2. Why Method Overriding
3. Dynamic Method Dispatch
4. Rules
5. Overloading vs Overriding

method is redefined

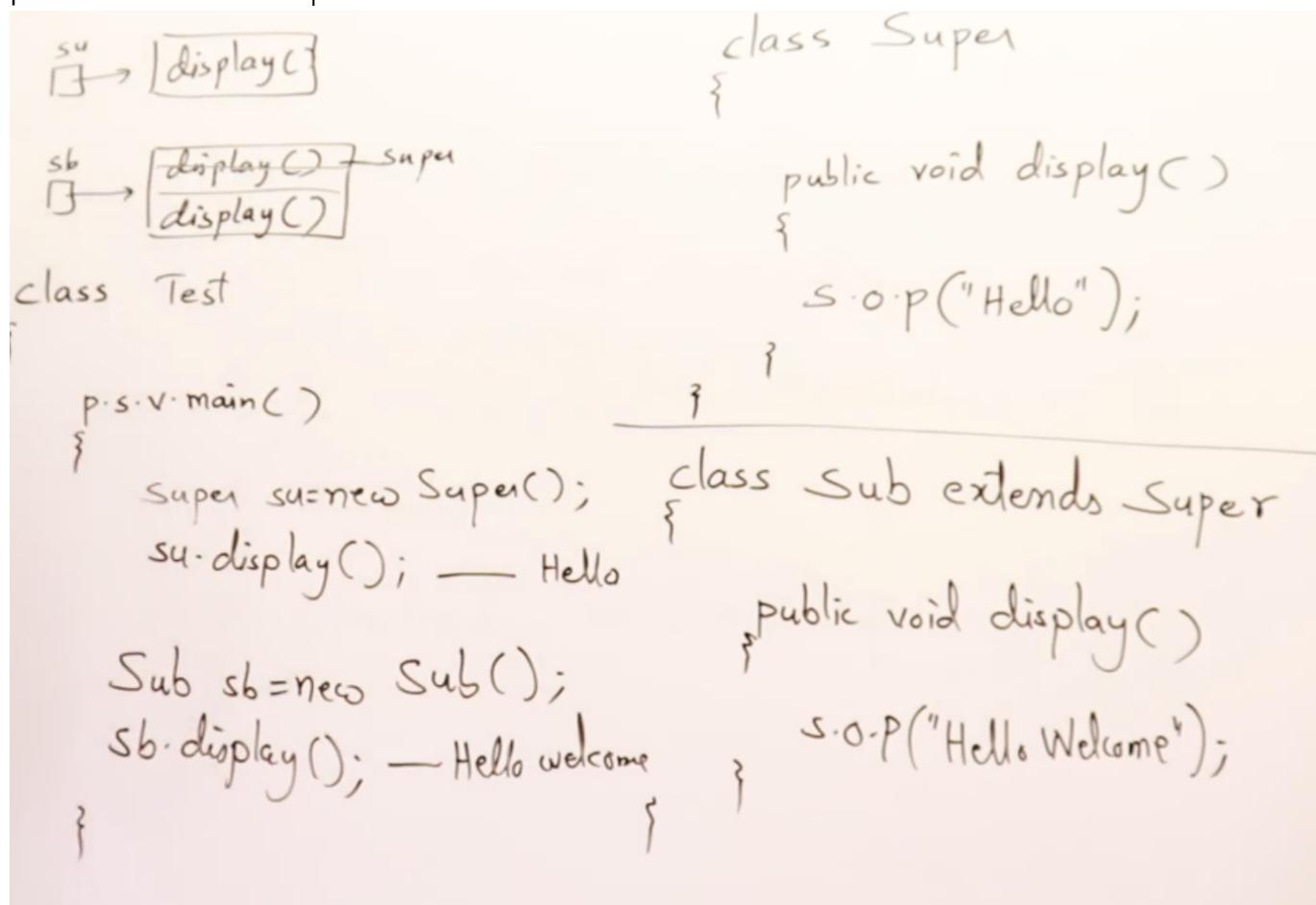
```
class Super
{
    public void display()
    {
        s.o.p("Hello");
    }
}
```

---

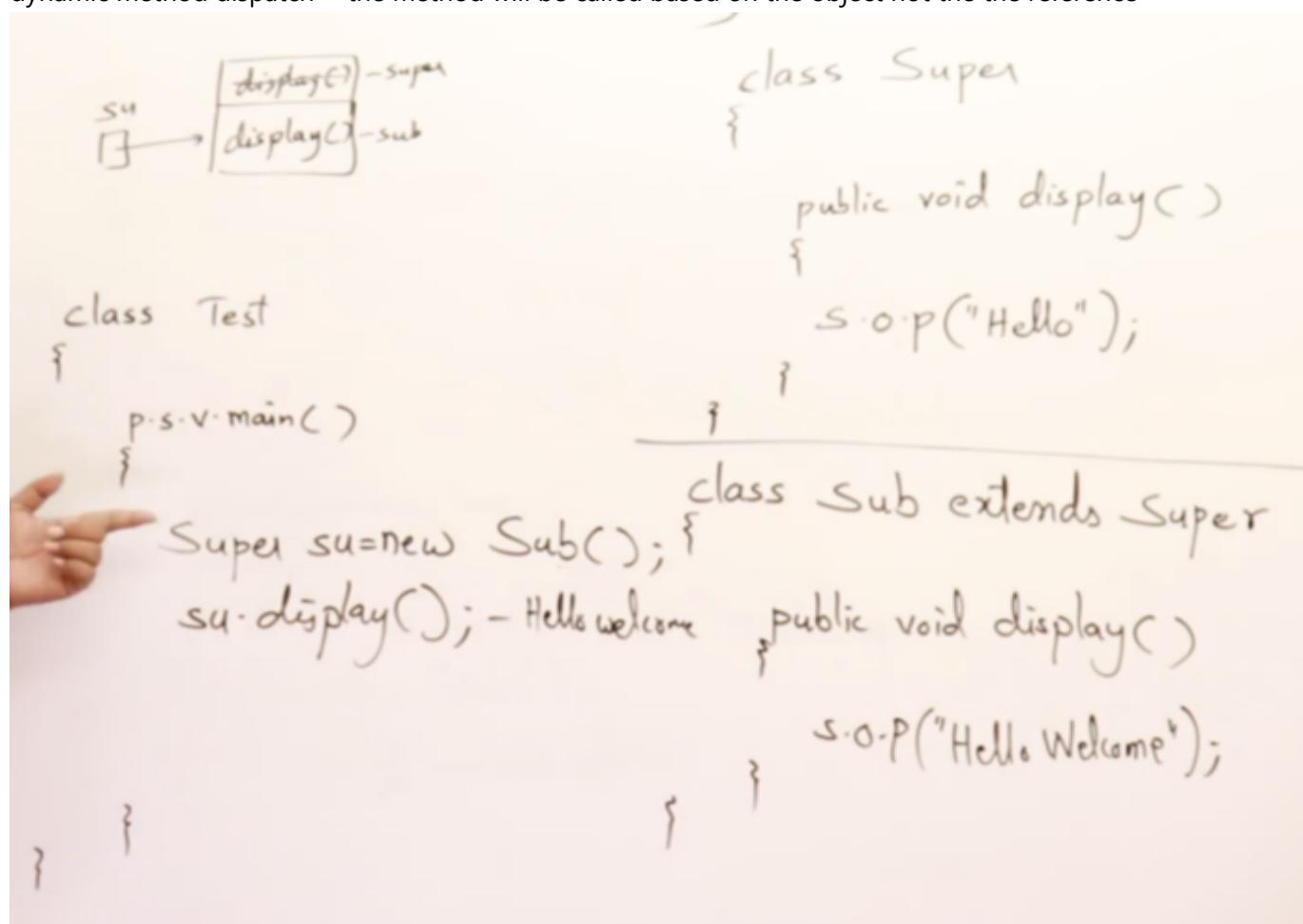
```
class Sub extends Super
{
}
```

```
public void display()
{
    s.o.p("Hello, Welcome");
}
```

parent method is hidden | shadowed



dynamic method dispatch = the method will be called based on the object not the reference



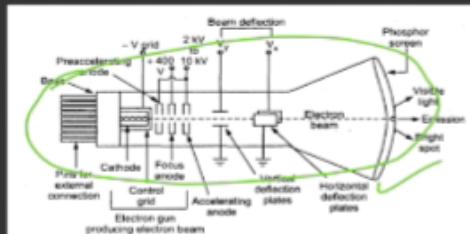
= A superclass reference holding an object of subclasss, and overrided(redefined) method is called, then the method of an object will be called, not the method of reference.

```
package override;

class Super
{
    public void display()
    {
        System.out.println("Super Display");
    }
}

class Sub extends Super
{
    @Override
    public void display()
    {
        System.out.println("Sub Display");
    }
}

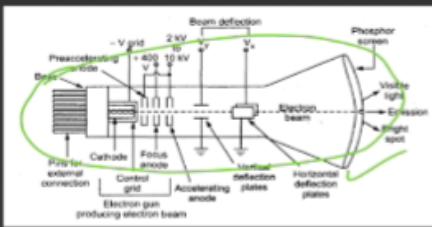
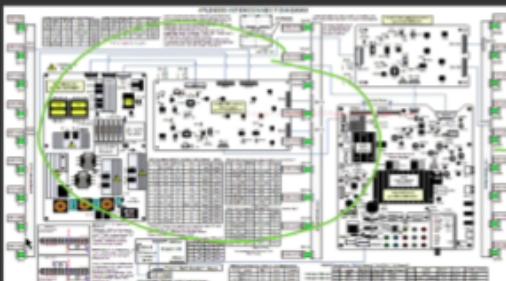
public class Override
{
    public static void main(String[] args)
    {
        Super s=new Sub();
        s.display(); //Sub Display
    }
}
```



Class TV



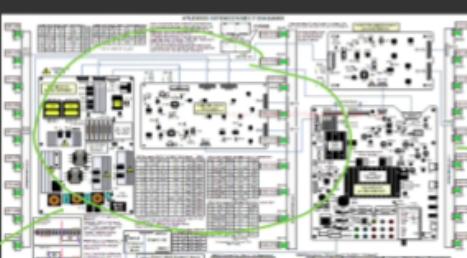
object.



Class TV



object.

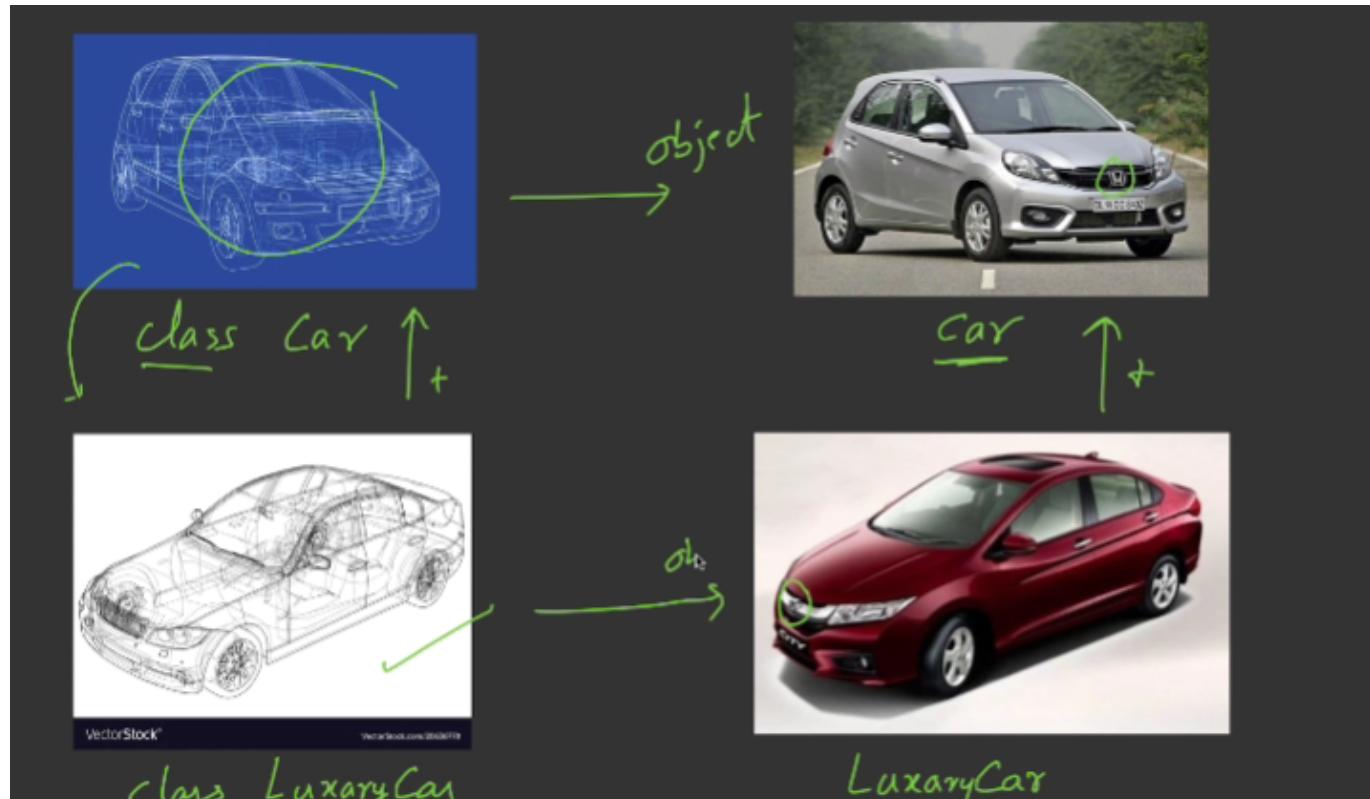
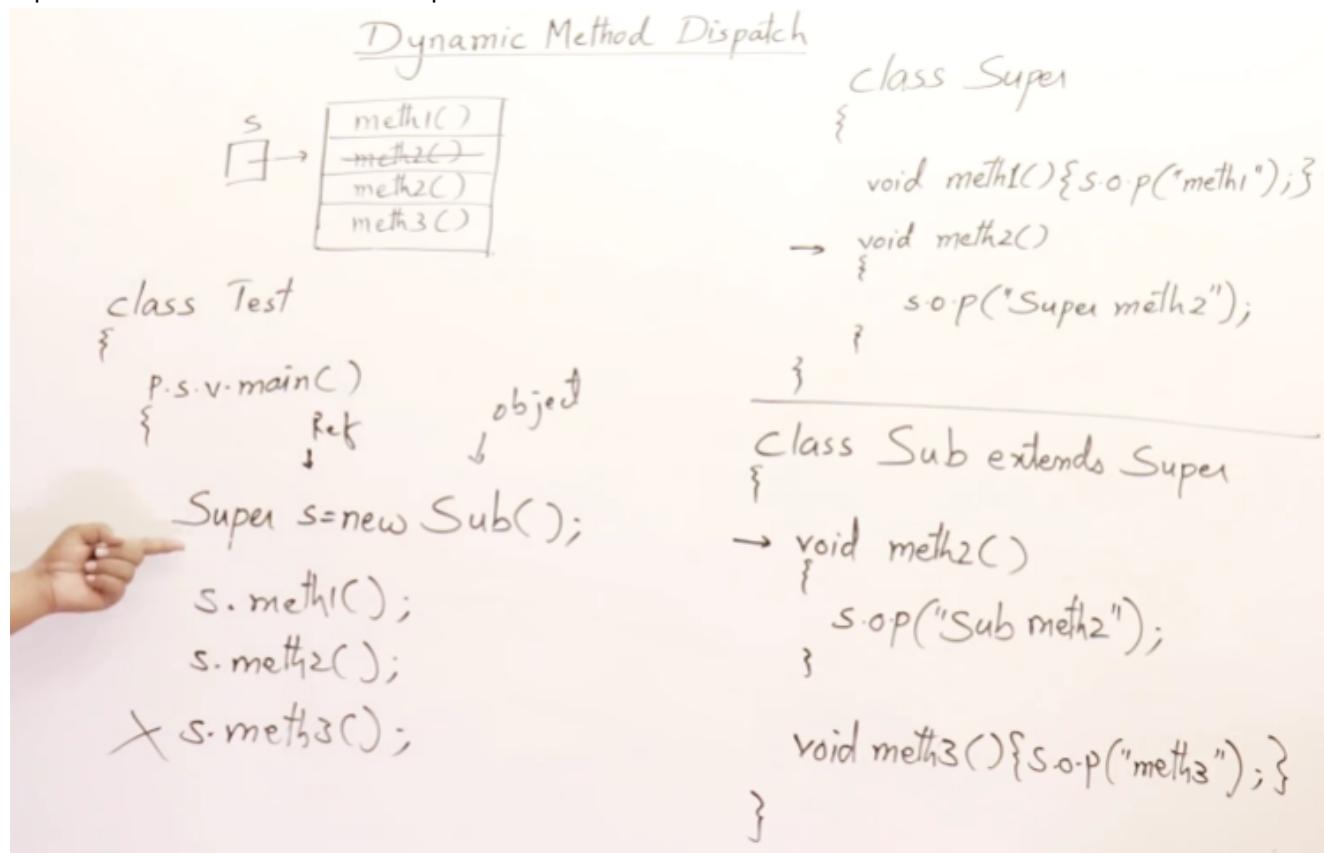


class SmartTV

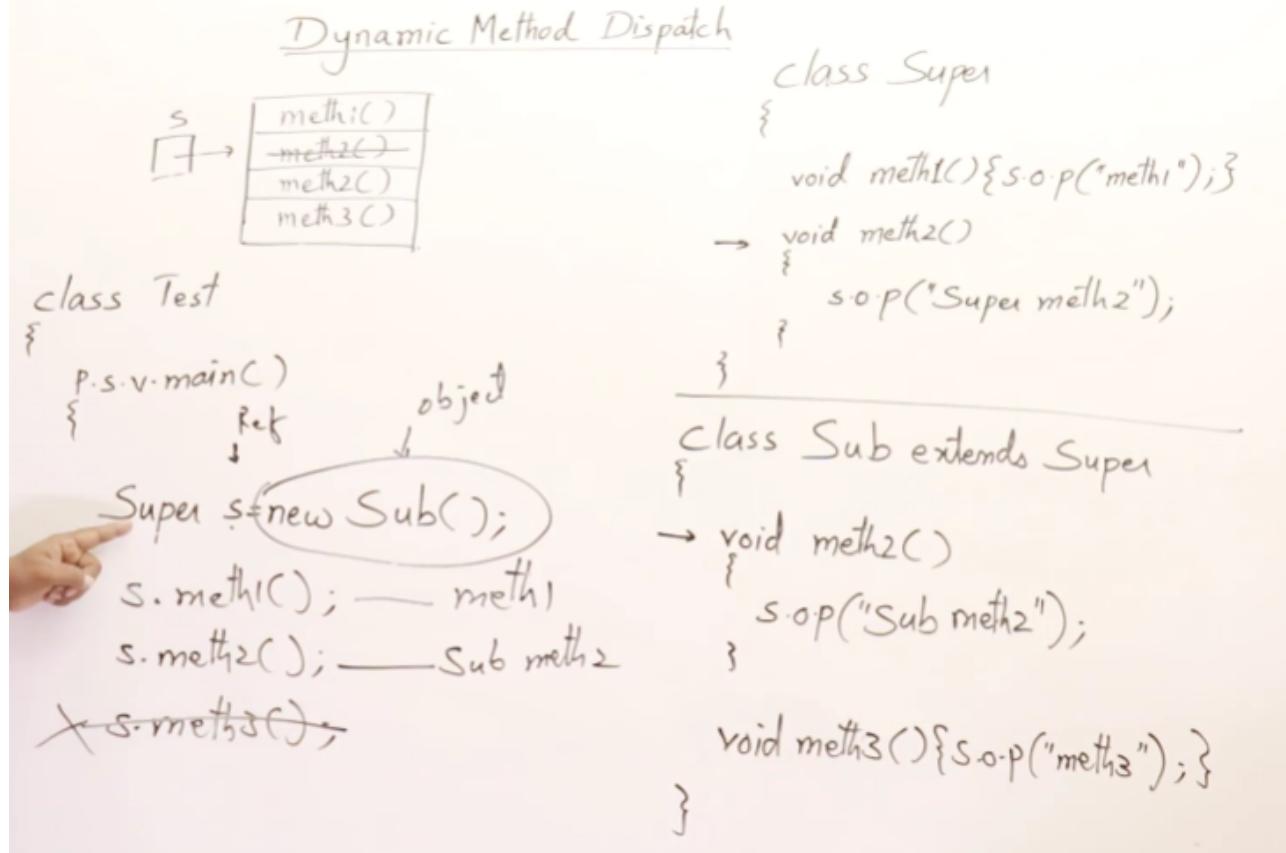


object

if parent is used as reference the specialized method of child class can't be called.



130. Dynamic method dispatch it is called as the runtime polymorphism method will be called based on the object not the reference



return type must match in the method overriding.

only covariant return type can be difference

```
class A{}  
class B extends A{}  
  
class Super  
{  
  
    public A display()  
    {  
        System.out.println("Super Display");  
        return new A();  
    }  
}  
  
class Sub extends Super  
{  
    @Override  
    public A display()  
    {  
        System.out.println("Sub Display");  
        return new A();  
    }  
}
```

```
class A{}  
class B extends A{}  
  
class Super  
{  
  
    public A display()  
    {  
        System.out.println("Super Display");  
        return new A();  
    }  
}  
  
class Sub extends Super  
{  
    @Override  
    public B display()  
    {  
        System.out.println("Sub Display");  
        return new B();  
    }  
}
```

we can't override the static and final methods parent access modifier can be any thing. but child can't be restrictive. or lower than the parent class.(either equal modifier or greater)

```
package overriderules;  
  
  
class Super  
{  
    private void display()//or even protected may not give any error.  
    {  
        System.out.println("Super display.");  
    }  
}
```

```

class Sub extends Super
{
    //Override if public
    public void display()
    {
        System.out.println("Sub display.");
    }
}

public class OverrideRules {

    public static void main(String[] args) {

        Sub s=new Sub();

    }
}

```

## 133 POLYMORPHISM

package overloading;

```

class Test
{
    public int max(int a,int b)
    {
        return a>b?a:b;
    }

    public int max(int a,int b,int c)
    {
        if(a>b && a>c) return a;
        else if(b>c) return b;
        return c;
    }
}

public class Overloading
{
    public static void main(String[] args)
    {
    }
}

```

---

package override;

```

class Super
{
    protected void display()
    {
        System.out.println("Super Display");
    }
}

class Sub extends Super
{
    public void display()
    {
        System.out.println("Sub Display");
    }
}

public class Override
{
    public static void main(String[] args)
    {
    }
}

```

In Java, polymorphism is achieved using the method overloading and overriding.

Method overloading compiler will decides the which method to call "compile time polymorphism"

```
package overloading;

class Test
{
    public int max(int a,int b)
    {
        return a>b?a:b;
    }

    public int max(int a,int b,int c)
    {
        if(a>b && a>c) return a;
        else if(b>c) return b;
        return c;
    }
}

public class Overloading
{
    public static void main(String[] args)
    {
        Test t=new Test();
        t.max(10,5);
        t.max(10,15,5);

    }
}
```

Method overriding = redefining the method of parent in the child class is known as the method overriding or

```
package override;

class Super
{
    public void display()
    {
        System.out.println("Super Display");
    }
}

class Sub extends Super
{
    public void display()
    {
        System.out.println("Sub Display");
    }
}

public class Override
{
    public static void main(String[] args)
    {
    }
}
```

"run time polymorphism" execution is decided at runtime

new means objects are created at runtime.

## all class

Question 2:

Which of the following statements are true for inheritance in java?

- Private members of superclass are accessible to subclass**
- All members of a Super class are available in Sub class, but all are not accessible.**
- A class can inherit from multiple classes.**



**Good job!**

Dynamic Binding means, deciding the methods call at runtime

Question 6:

Dynamic Methods Dispatch means

- Static binding

- Dynamic Binding**

- Early Binding

## Section 13 Abstract Class

---

132 what is abstract class ?

# Abstract class

1. What is an Abstract class
2. Example of Abstract class
3. Why Abstract classes
4. Key points

abstract class vs concrete class we can't create the obj of the abstract class but you can create the reference of abstract class we can create the obj of the concrete class.

Abstract method is method with no body and method signature is marked with abstract keyword. if a class having atleast one abstract method then the class is declared as the abstract method.

Abstract class can have zero or more abstract methods in it

if any class is inheriting the abstract class then that sub class also becomes the abstract class if abstract methods of extending class, defination is not given in the sub class.

## ABSTRACT CLASSES



### What is an Abstract Class?

- There are two types of classes **Abstract class and Concrete class**
- If **abstract** keyword is used before the class then it is an **Abstract Class** if nothing is written before class then it is a **Concrete class**
- Object of an Abstract class cannot be created but object of Concrete class can be created
- reference of abstract class is allowed

```
//asuper bstract class  
abstract class Super  
{  
    Super()  
}
```

```
{  
    System.out.println("Super");  
}  
void meth1()  
{  
    System.out.println("meth1");  
}  
abstract void meeth2();  
}  
//concrete class  
class sub extends Super  
{  
    Void meth2()  
    {  
        System.out.println("meth2");  
    }  
}  
class test  
{  
    public static void main()  
    {  
        Super s1; // reference of abstract is allowed  
        sub s2 =new sub();  
    }  
}
```

- Method which is not having a body is known as Abstract method, the method must be declared as abstract • The abstract method is undefined method • A class is Abstract class if at least one of the methods is abstract
- If any other class inherits abstract class then that class also becomes abstract class but to become a concrete class the subclass must override the undefined method • A class becomes useful if it overrides all the methods of abstract class • Abstract classes are used for imposing standards and sharing methods • Sub classes are meant for following standards

➤ Do's and Don'ts of Abstract Class • An Abstract class cannot be final because if it is made final then it cannot be extended whereas abstract class is meant for inheritance • An Abstract method cannot be final because if it made final then it cannot be overridden whereas Abstract method is meant for overriding • Abstract Class and method can neither be final nor static • A Sub class must override an abstract method or else it will become abstract class

```

abstract class Super
{
    public Super() { System.out.println("Super Constructor"); }

    public void meth1()
    {
        System.out.println("Meth1 of Super");
    }
}

public class AbstractExample
{
    public static void main()
    {
        Super s=new Super();
        s.meth1();
    }
}

```

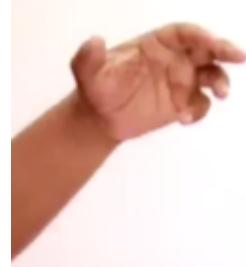
use of abstract class = abstract methods are meant for inheritance.

## 136 EXAMPLE 1

Abstract class (are meant to provide the guidelines)

Abstract class

|                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> abstract class Hospital {     abstract void emergency();     abstract void appointment();     abstract void admit();     abstract void billing(); } </pre> | <pre> class MyHospital extends Hospital {     MyHospital()     {         void emergency()         {             ;         }          void appointment()         {             ;         }          void admit()         {             ;         }          void billing()         {             ;         } } </pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



## 137 Example 2 : KFC

outlet

## Abstract class

abstract class KFC

KFC()

{  
}=

void makeItem()

{  
}=

abstract void billing();

abstract void offer();

{

class MyKFC extends KFC

{

MyKFC()

{  
}=

void billing()

{  
}=

void offer()

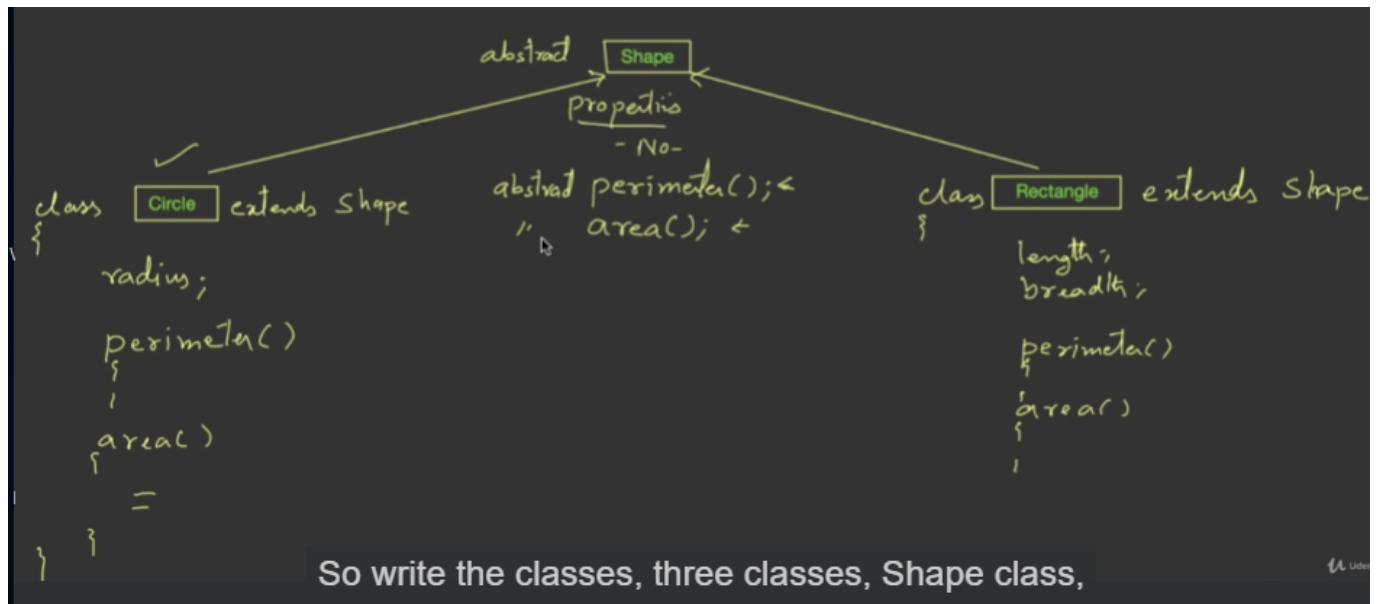
{  
}=

void festiveOffer()

{  
}=

KFC K=new MyKFC();

138 : sc : shapes



```

package scabstract1;

abstract class Shape
{
    abstract public double perimeter();
    abstract public double area();
}

class Circle extends Shape
{
    double radius;

    public double perimeter()
    {
        return 2*Math.PI*radius;
    }
    public double area()
    {
        return Math.PI*radius*radius;
    }
}

class Rectangle extends Shape
{
    double length;
    double breadth;

    public double perimeter()
    {
        return 2*(length+breadth);
    }
    public double area()
    {
        return length*breadth;
    }
}

```

```
}

public class SCAbstract1
{
    public static void main(String[] args)
    {
        Rectangle r=new Rectangle();
        r.length=10;
        r.breadth=5;

        Shape s=r;

        System.out.println(s.area());
    }
}
```

final classes can't be extended if a class is abstract then we can't make it final

```
abstract final class Super
{
    abstract public void meth1();
}

public class AbstractRules
{
    public static void main(String[] args)
    {
        Super s;
        s=new Super();
    }
}
```

```
abstract class Super
{
    abstract final public void meth1();
}

public class AbstractRules
{
    public static void main(String[] args)
    {
        Super s;
    }
}
```

final means we can't override method. but abstract methods are meant for overriding

```
package abs
abstract static class Super
{
    abstract public void meth1();
}

public class AbstractRules
{
    public static void main(String[] args)
    {
        Super s;
    }
}
```

```
abstract class Super
{
    abstract static public void meth1();
}

public class AbstractRules
{
    public static void main(String[] args)
    {
        Super s;
    }
}
```

```

abstract class Super
{
    abstract public void meth1();
}

abstract class Sub extends Super
{
}

public class AbstractRules
{
    public static void main(String[] args)
    {
        Super s;
    }
}

```

## Abstract class

### Do and Dont

1. ~~abstract class Test  
{  
=}~~
2. ~~abstract final class Test  
{  
=}~~
3. ~~abstract static class Test  
{  
=}~~
4. ~~abstract class Test  
{  
final abstract void display();  
=}~~
5. ~~abstract class Test1  
abstract void display();  
{  
class Test2 extends Test1  
{  
void display()  
=}}~~

usually static classes are the nested classes.

Question 2:

Which of the following statements regarding abstract classes are true ?

- A subclass of a non-abstract superclass can be abstract.
- A subclass overriding only concrete method of a superclass will remain abstract.
- An abstract class can be extended.
- All of the above

Question 4:

Which of these is not a correct statement?

- Every class containing abstract method must be declared abstract
- reference of an abstract class can be declared.
- Abstract class can be initiated by new operator
- abstract classes are useful for achieving generalisation.

with abstract class inheritance and polymorphism is achieved but interface is purly meant to achieve th polymorphism

```
abstract class Test1  
{  
    abstract public void meth1();  
    abstract public void meth2();  
}
```

```
interface Test1  
{  
    void meth1();  
    void meth2();  
}
```

```
class Test2 extends Test1
```

```
public void meth1()  
{  
}  
public void meth2()  
{  
}  
=
```

```
class Test2 implements Test1  
{
```

```
public void meth1()  
{  
}  
public void meth2()  
{  
}  
=
```

```
Test1 t=new Test2(); }
```

if a class has all abstract methods then it is called as interface. better to use interface instead of abstract class  
interface reference can be created but object can't be created

```
package interfacepractice;

interface Test
{
    void meth1();
    void meth2();
}

class My extends Test
{

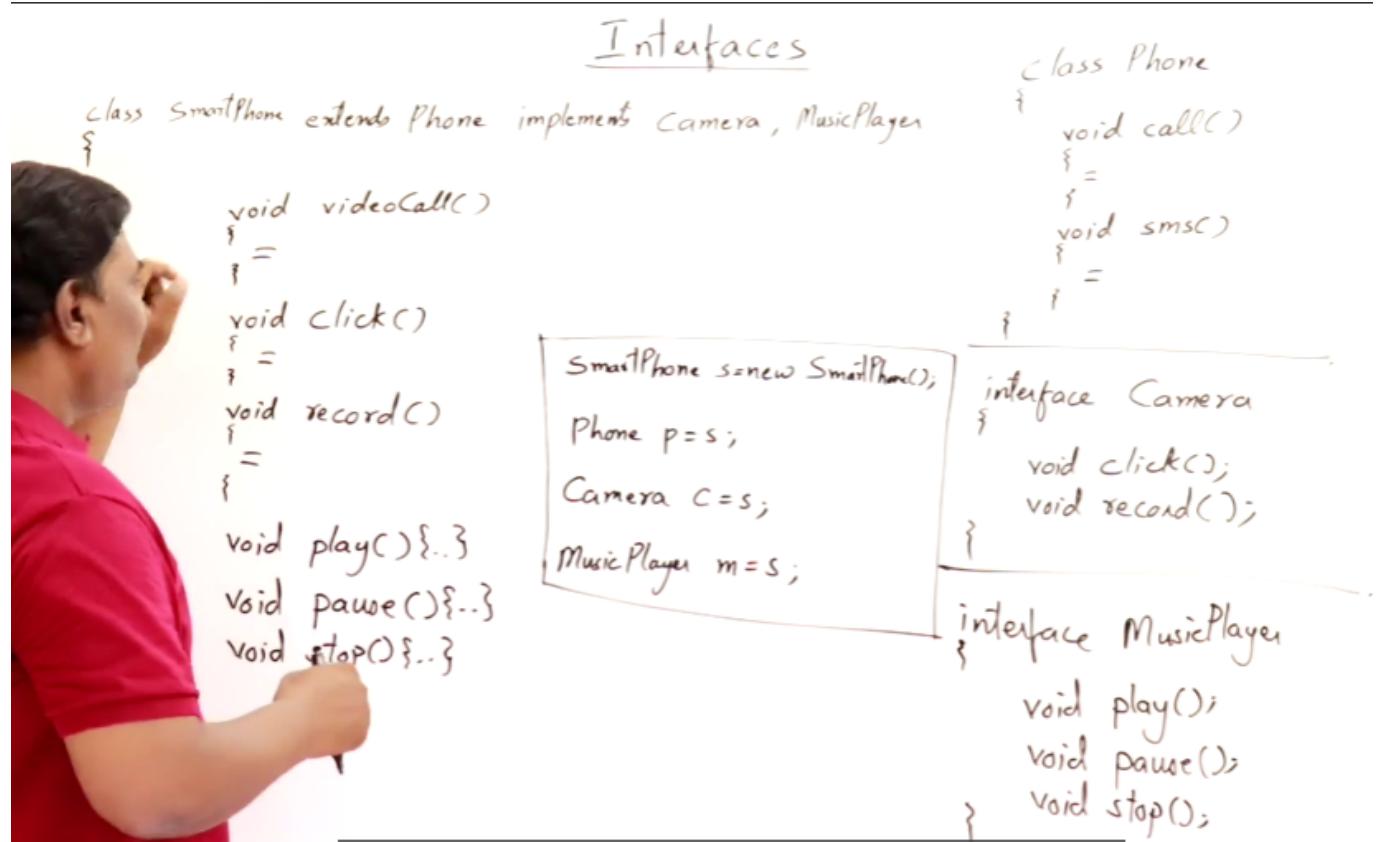
}

public class InterfacePractice
{
    public static void main(String[] args)
    {
        Test t;
```

```
interface Test
{
    void meth1();
    void meth2();
}

class My implements Test
{
    public void meth1()
    {
        System.out.println("Meth1 of class My");
    }
    public void meth2()
    {
        System.out.println("Meth2 of class My");
    }
    public void meth3()
    {
        System.out.println("Meth3 of class My");
    }
}
```

interfaces are meant for achieving the runtime polymorphism or dynamic dispatch mechanism.



by default methods inside the interface are public and abstract

```

4   interface Test
5   {
6       int X=10;
7       public abstract void meth1();
8       public abstract void meth2();
9   }
  
```

variables / identifiers inside the interface

write

the uppcases for variables inside the interfaces by default variables are public static final

interface can have static methods with the body.

```

interface Test
{
    final static int X=10;
    public abstract void meth1();
    public abstract void meth2();
    public static void meth3()
    {
        System.out.println("Meth3 of Test");
    }
}

public class InterfacePractice
{
    public static void main(String[] args)
    {
        System.out.println(Test.X);  X
    }
}
  
```

```

interface Test
{
    final static int X=10;
    public abstract void meth1();
    public abstract void meth2();
    public static void meth3()
    {
        System.out.println("Meth3 of Test");
    }
}

interface Test2 extends Test
{
    void meth4();
}

public class InterfacePractice
{
    public static void main(String[] args)
    {
        System.out.println(Test.X);
        Test.meth3();
    }
}

```

interfaces can be extended.

from JAVA 8 onward default methods are also available inside the interfaces

```

{
    final static int X=10;
    public abstract void meth1();
    public abstract void meth2();
    default void meth3()
    {
        System.out.println("Meth3 of Test");
    }
}

interface Test2 extends Test
{
    void meth4();
}

class My implements Test2
{
    public void meth1(){}
    public void meth2(){}
    public void meth4(){}
}

public class InterfacePractice
{
    public static void main(String[] args)
    {
        System.out.println(Test.X);
        Test.meth3();
    }
}

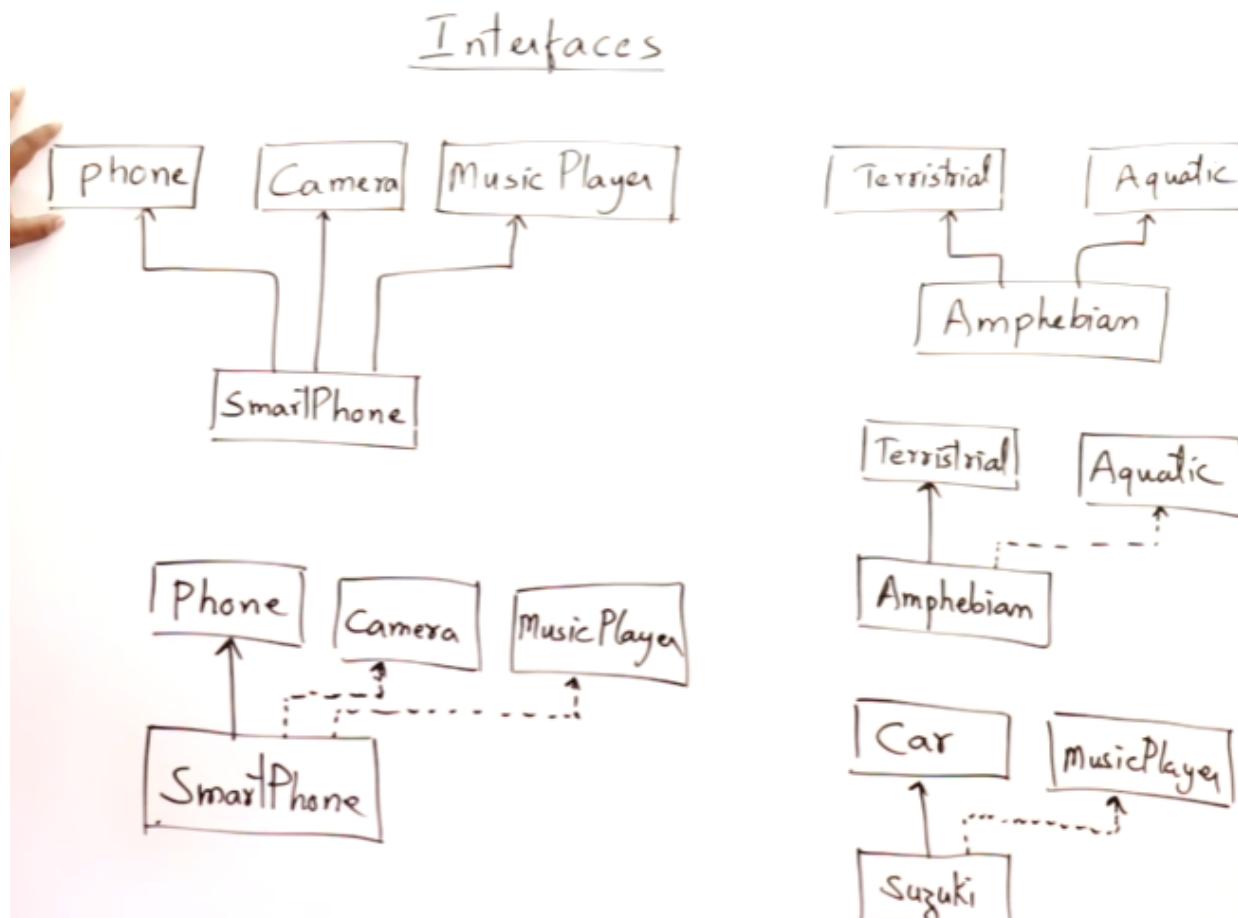
```

what is the use of default methods in the interfaces if to modify the interface then all the classes which are implementing the interface should be modified. to avoid this default methods are used. then other classes are not disturbed.

FROM JAVA 9 ONWARDS PRIVATE METHODS ARE ALSO ALLOWED INSIDE THE INTERFACE but not abstract

```
private void meth3()
{
    System.out.println("Meth3 of Test");
}
```

private methods are allowed. it is used for the internal use of the default methods



java perspective interface is the best way than the c++ interface

Question 2:

Instead of multiple inheritance, Java uses following concept:

abstraction

polymorphism

encapsulation

interface

## Section 15: Inner Classes

Need of inner class = if a class is becoming big and complex then it is better to divide the class into the smaller classes.

### Inner class

1. Nested Inner class
2. Local inner class
3. Anonymous inner class
4. Static inner class

interface inside an interface class inside an class

```
class Outer
{
    int x=10;

    class Inner
    {
        int y=20;
        void innerDisplay()
        {
            System.out.println(x);
        }
    }
}
```

```
    } } S.O.P(y);  
② void outerDisplay()  
{  
    Inner i=new Inner();  
    i.innerDisplay();  
    S.O.P(i.y);  
}  
er class, a class itself.  
}
```

class Test

{

    p.s.v.main()

}

    Outer o=new Outer();

    o.outerDisplay();

    Outer.Inner i=new Outer().new Inner();

}

}

```
package nestedinner;

class Outer
{
    static int x=10;
    Inner i=new Inner();

    class Inner
    {
        int y=20;
        public void innerDisplay()
        {
            System.out.println(x+" "+y);
        }
    }

    public void outerDisplay()
    {
        i.innerDisplay();
        System.out.println(i.y);
    }
}
```

```
public class NestedInner {  
  
    public static void main(String[] args)  
    {  
        Outer.Inner oi=new Outer().new Inner();  
        oi.innerDisplay();  
  
    }  
  
}
```

c++ has above limitations but java has no limitations.

## Local and Anonymous Inner class

we can define the class inside the method that class is called as the local inner class (only visible and accessible inside the method)

```
class Outer
{
    void Display()
    {
        class Inner
        {
            void innerDisplay()
            {
                System.out.println("Hello");
            }
        }
        Inner i = new Inner();
        i.innerDisplay();
    }
}
```

Anonymous inner class

```
abstract class My
{
    abstract void display();
}
```

```
class Outer
{
```

```
    public void meth()
    {
```

```
        My m=new My();
    }
```

```
        { public void display()
```

```
            S.O.P("Hello");
        }
```

```
    };
```

```
    m.display();
}
```

object of that class

even it is

interface we can do it same.

```
class Outer
{
    public void display()
    {
        My m=new My(){ public void show(){ System.out.println("Hello"); }};
    }
}
```

## Static inner class

Inner class ✗

class Outer

{

    static int x=10;

    int y=20;

}

class Test

{

    P.S.V.m()

}

Outer.Inner i=new Outer.Inner();

i.display();

}

void display()

{

    S.O.P(x);

    ✗ S.O.P(y);

}

anonymours object

```
class Outer
{
    public void display()
    {
        class Inner
        {
            public void show()
            {
                System.out.println("Hello");
            }
        }

        new Inner().show();
    }
}
```



**Good job!**

if you want to use any keyword, then only abstract is allowed.

Question 2:

Which is true about a method-local inner class?

It must be marked final.

It can be marked abstract.

It can be marked public.

It can be marked static.

## Section 16: Static and Final

---

## Static

- Static Variables
- Static Methods
- Static Nested class

## Static Blocks

static keyword is used for representing metadata. Metadata means data about data. basically they are useful for representing the information of a class not the objects.

Though all the objects can share the information.

static members are useful for representing information or data related to a class. so if you have only data related to the class then it is better to make it as static variable.

If that data need some computation or processing then it is better to make it as static method.

and if you have lots of data and that can be grouped together and made as a nested class.

```

class HondaCity
{
    static long price=10;
    int a, b;
}

static double onRoadPrice(String city)
{
    switch(city)
    {
        case "delhi":
            return price+price*0.1;
        case "mumbai":
            return price+price*0.09;
    }
}

```

class Test

```

p.s. v. main()
{
    HondaCity h1=new HondaCity();
    HondaCity h2=new HondaCity();
}

```

static members are shared by all the objects **any one of the objects modifies it then it is modified for all the objects.** can be used as the shared data.

//static members of the class are created inside the method area. static variables can be accessible without obj creation.

```
class HondaCity  
{
```

```
    static long price=10;
```

```
    int a, b;
```

```
    static double OnRoadPrice(String city)
```

```
    { switch(city)
```

```
        case "delhi":
```

```
            return price+price*0.1;
```

```
        case "Mumbai":
```

```
            return price+price*0.09;
```

```
    }
```

if multiple static methods are available then makes it as the static class and keep the methods inside the class.

static classes can't use this or super keyword because they refer to the instance of the class.

outer class can't be static but inner class can be static.

```
package staticpractice;

class Test
{
    static int x=10;
    int y=20;

    void show()
    {
        System.out.println(x+" "+y);
    }

    static void display()
    {
        System.out.println(x);
    }
}

public class StaticPractice {

    public static void main(String[] args) {

        Test t1=new Test();
        t1.show();
        t1.x=30;
        t1.y=50;

        Test t2=new Test();
        t2.show();

    }
}
```

## 154. static block

we can write set of stmts inside the block. make it as static block. those statements will be executed only once when the class is loaded into the memory.

inside the static block u can access only static member or methods.

```
class My
{
    static int s;

    static
    {
        s.o.p("Block1");
        s=10;
    }

    static
    {
        s.o.p("Block2");
    }
}
```

there can be more than one static block in the class. but they can be executed in the order. this feature is introduced in 1.7 version. but not much used.

## 157. final keyword

1. final variable
2. final Method
3. final class

final variables are just like constant , means values are fixed



```
class My
{
    final int MIN=1;
    final int NORMAL;
    final int MAX;

    static
    {
        NORMAL=5;
    }

    My()
    {
        MAX=10;
    }
}
```

directly we can assign the value to the final variable static block can be used to initialize the final variables.  
inside constructor also we can initialize the final variables

Final Method

```
class Super
```

```
final void meth1()
{
    s.o.p("Hello");
}
```

```
class Sub extends Super
```

```
X void meth1()
{
    s.o.p("Hi");
}
```

```
void meth2()
{
    s.o.p("Bye");
}
```

Final method. final method can't be overridden. (or redefined)

final class

```
final class Super
```

{

==

}

X class sub extends Super

==

}

final class can't be extended. but you can create the object and use it.

you can make policy and freeze of class and method so that other class can't borrow features of it.

## 158 Demo of Final

if final is the member of class, then it must be initialized

```
public class Fin {
    final float PI;
}

public static void main(String[] ar
```

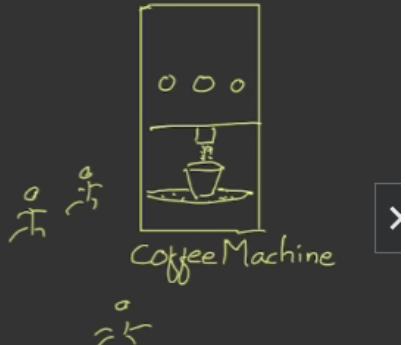
instance block

## 159. Singleton class

.getInstance()

a class which can create only one object more than once obj of a class are not allowed

**Singleton Class**



```
class CoffeeMachine {
    private float coffeeQty;
    private float milkQty;
    static private CoffeeMachine our=null;
    private CoffeeMachine()
        coffeeQty=1;
        milkQty=1;
    }
    static public CoffeeMachine getInstance()
    {
        if(our==null)
            our=new CoffeeMachine();
        return our;
    }
}
```

~~CoffeeMachine c=new CoffeeMachine();  
CoffeeMachine s=CoffeeMachine.getInstance();~~

```
package singleton;

class CoffeeMachine
{
    private float coffeeQty;
    private float milkQty;
    private float waterQty;
    private float sugarQty;

    static private CoffeeMachine my=null;

    private CoffeeMachine()
    {
        coffeeQty=1;
        milkQty=1;
        waterQty=1;
    }
}
```

```
sugarQty=1;
}

public void fillWater(float qty)
{
    waterQty=qty;
}
public void fillSugar(float qty)
{
    sugarQty=qty;
}
public float getCoffee()
{
    return 0.15f;
}

static CoffeeMachine getInstance()
{
    if(my==null)
        my=new CoffeeMachine();
    return my;
}

}

public class Singleton
{
    public static void main(String[] args)
    {
        CoffeeMachine m1=CoffeeMachine.getInstance();
        CoffeeMachine m2=CoffeeMachine.getInstance();
        CoffeeMachine m3=CoffeeMachine.getInstance();

        System.out.println(m1+" "+m2+" "+m3);
        if(m1==m2 && m1==m3)
            System.out.println("Same");
    }
}
```

```
package scstatic1;
import java.util.Date;

class Student
{
    private String rollNo;

    private static int count=1;

    private String assignRollNo()
    {
```

```
Date d=new Date();

String rno="Univ-"+(d.getYear()+1900)+"-"+count;
count++;
return rno;
}
Student()
{
    rollNo=assignRollNo();
}
public String getRollNo()
{
    return rollNo;
}

}

public class SCStatic1
{
    public static void main(String[] args)
    {
        Student s1=new Student();
        Student s2=new Student();
        Student s3=new Student();

        System.out.println(s1.getRollNo());
        System.out.println(s2.getRollNo());
        System.out.println(s3.getRollNo());

    }
}
```



Good job!

Question 5:

We can override static methods

True

False



Good job!

Question 4:

We can overload static methods

True

False

## Section 17. Packages

---

what are the packages ?

= a package is a the collection of classes, interfaces and other packages. file explorer program files > jdk > lib > src (it contains all packages) = java , lang math , again more packages for exception there are java files

import java.lang.\*; //importing a package. so that use class. import java.lang.String; //it saves the compiler time

fully qualified importing

162. how to create own package and use it.

dir > demo class for disp method > compiled the u get Demo.class how to include a class inside a package

```
javac -d . Demo.java
```

compiler creates the package

```
javac -d C:  
//you can mention directory.
```

```
del *.class //to delete the class
```

```
javac -d . Demo2.java //adds to the current package available
```

```
package mypack1.inner;

public class Demo3
{
    public void display()
    {
        System.out.println("Welcome to Demo3");
    }
}
```

Ln 1, Col 20      100%      Windows (CRLF)      UTF-8

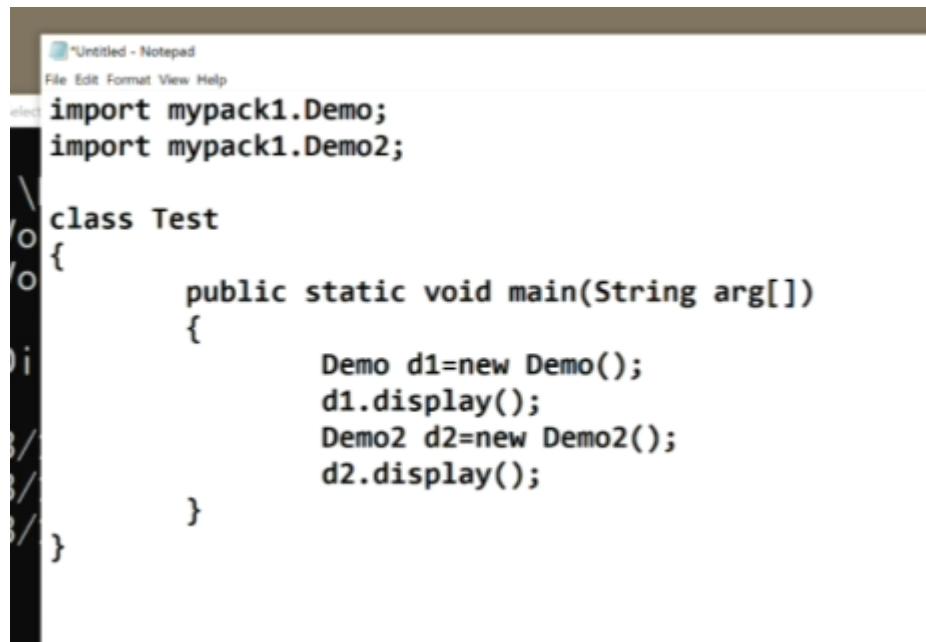
6:10 PM 3/23/2020

Udemy

package and sub package,

3117 ## what  
3118 = a pack  
3119 file exp  
3120 program  
3121 packages  
3122 import j  
3123 import j  
3124 fully qu  
3125 ## 162.  
3126 dir > de  
3127 compiled  
3128 how to i  
3129 >javac -  
3130 compiler  
3131 > javac  
3132 //you ca  
3133 3134  
3135 3136  
3137 3138  
3139 3140  
3141 >del \*.c  
3142 //to del  
3143 >javac -  
3144 //addr +

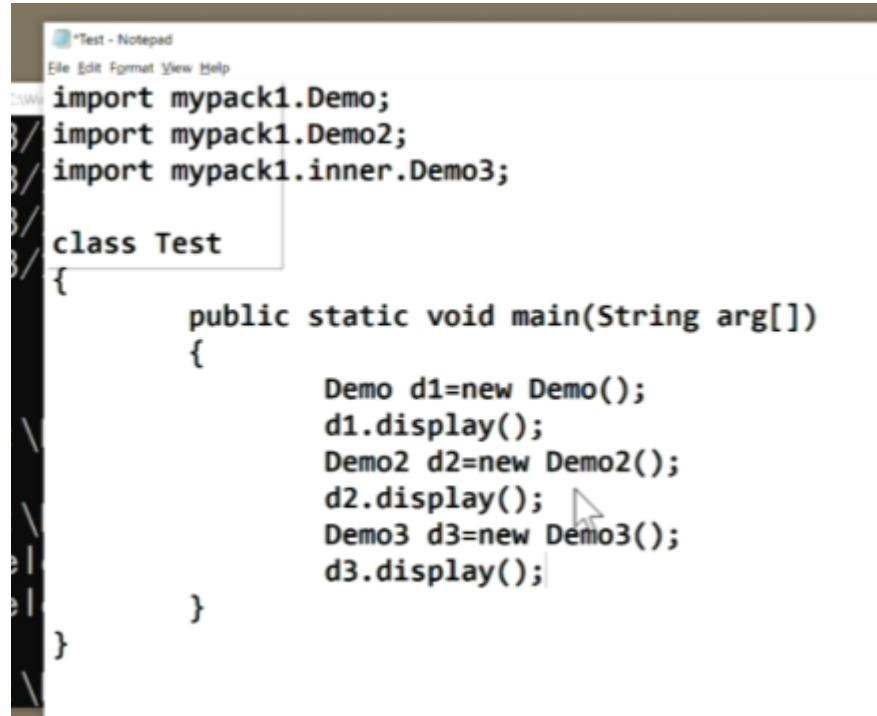
```
javac -d . Demo3.java
```



```
*Untitled - Notepad
File Edit Format View Help
import mypack1.Demo;
import mypack1.Demo2;

class Test
{
    public static void main(String arg[])
    {
        Demo d1=new Demo();
        d1.display();
        Demo2 d2=new Demo2();
        d2.display();
    }
}
```

accessing



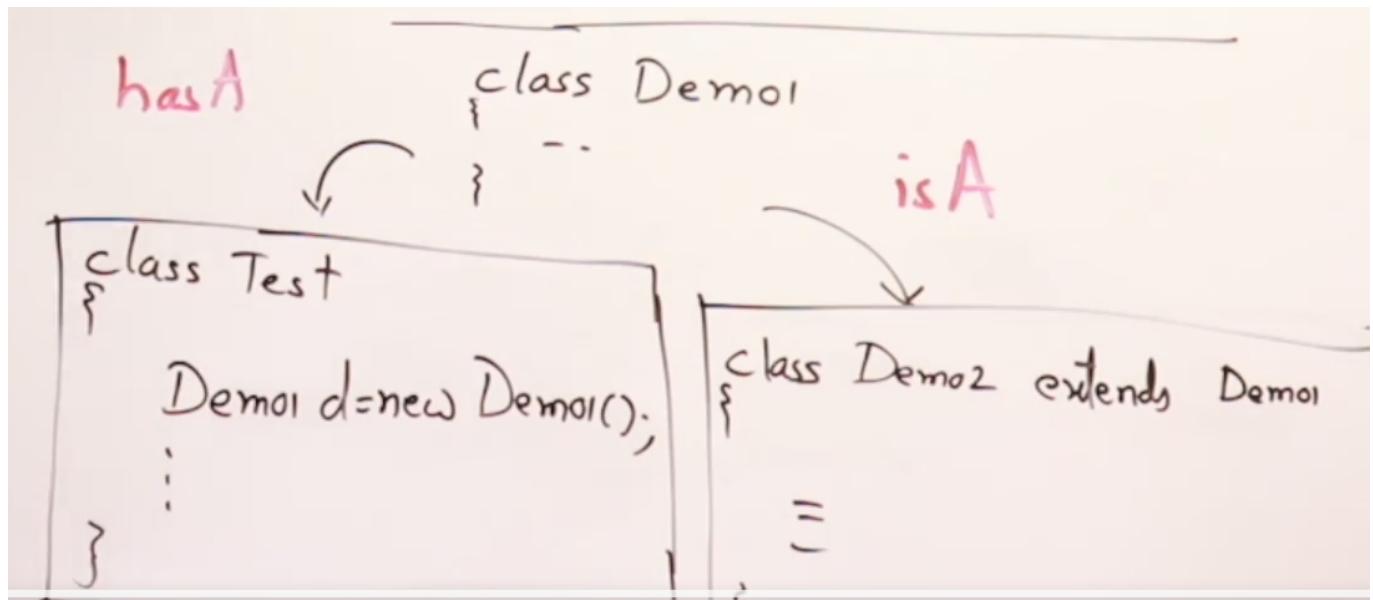
```
*Test - Notepad
File Edit Format View Help
import mypack1.Demo;
import mypack1.Demo2;
import mypack1.inner.Demo3;

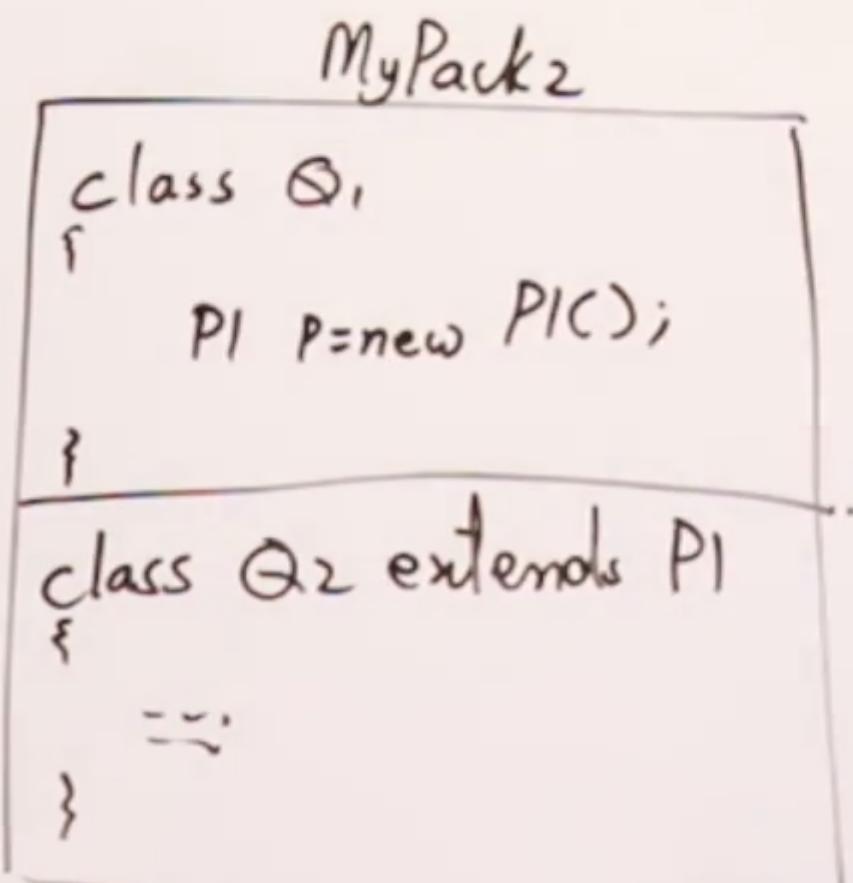
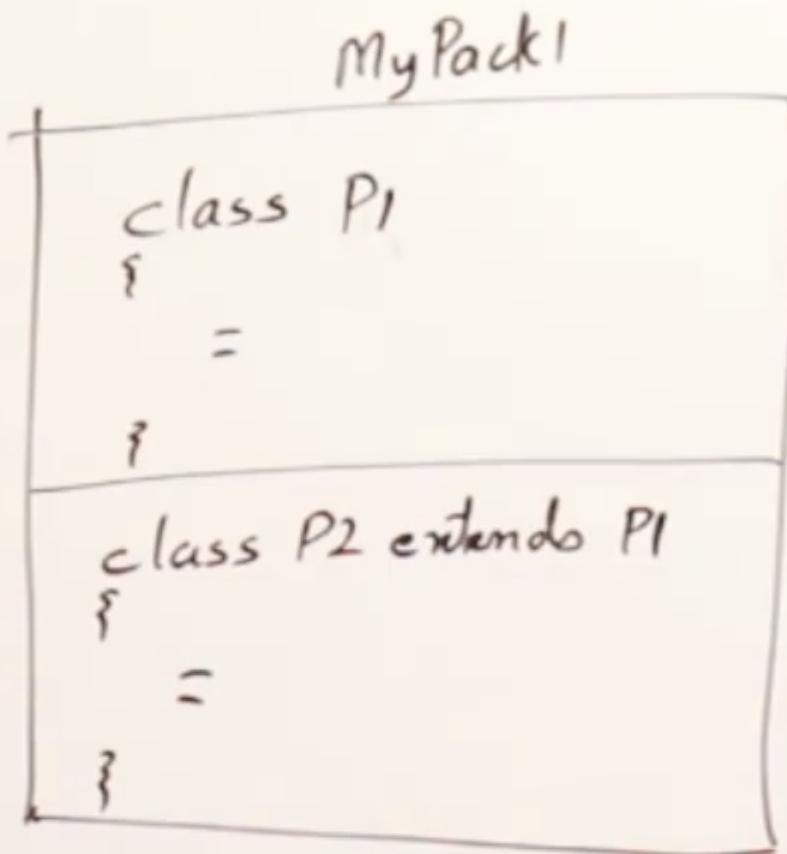
class Test
{
    public static void main(String arg[])
    {
        Demo d1=new Demo();
        d1.display();
        Demo2 d2=new Demo2();
        d2.display(); d3.display();
        Demo3 d3=new Demo3();
        d3.display();
    }
}
```

## 163. Accesss Modifier

default private protected public

outer class can be only public or default





|                          | default | private | protected | Public |
|--------------------------|---------|---------|-----------|--------|
| Same class               | ✓       | ✓       | ✓         | ✓      |
| Same Pack sub class      | ✓       | ✗       | ✓         | ✓      |
| Same Pack non-sub class  | ✓       | ✗       | ✓         | ✓      |
| Difft Pack subclass      | ✗       | ✗       | ✓         | ✓      |
| Difft Pack non-sub class | ✗       | ✗       | ✗         | ✓      |

```

package mypack1;

class Demo1
{
    int a=10;
    //private.
    public int b=20;
    protected int c=30;
    public int d=40;

    public void display()
    {
        System.out.println(a+b+c+d);
    }
}
/*Public class Demo2()
{
    Demo1 d=new Demo1();
    public void show()
    {
        System.out.println(d.a+d.b+d.c+d.d);
    }
}*/
class Demo3 extends Demo1

```

```
{  
    Demo1 d=new Demo1();  
    public void show()  
    {  
        System.out.println(d.a+d.b+d.c+d.d);  
    }  
}  
  
class Demo4 extends Demo1  
{  
    public void show()  
    {  
        System.out.println(a+b+c+d);  
    }  
}  
  
public class MyPack1 {  
  
    public static void main(String[] args)  
    {  
        Demo1 d1=new Demo1();  
        d1.display();  
        System.out.println(d1.a+d1.b+d1.c+d1.d);  
    }  
}
```

*Demo1.java*

|                                   | PRIVATE | DEFAULT | PROTECTED | PUBLIC |
|-----------------------------------|---------|---------|-----------|--------|
| Same class                        | Yes     | Yes     | Yes       | Yes    |
| Same package<br>Subclass          | No      | Yes     | Yes       | Yes    |
| Same package<br>Non-subclass      | No      | Yes     | Yes       | Yes    |
| Different package<br>Subclass     | No      | No      | Yes       | Yes    |
| Different package<br>Non-subclass | No      | No      | No        | Yes    |

*Test.java*

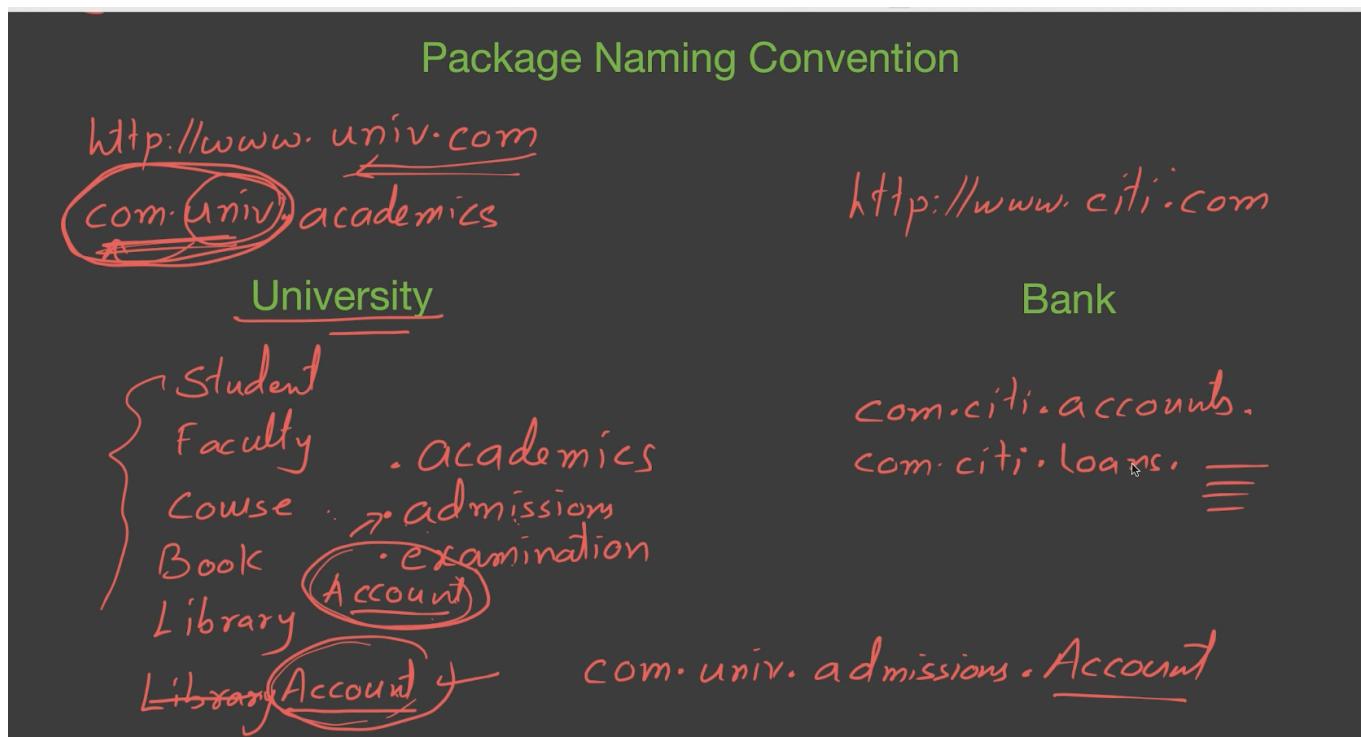
```

File Edit Format View Help
package mypack1;

public class Demo3 extends Demo1
{
    public void show()
    {
        System.out.println(a+b+c+d);
    }
}

```

## Package Naming Conventions

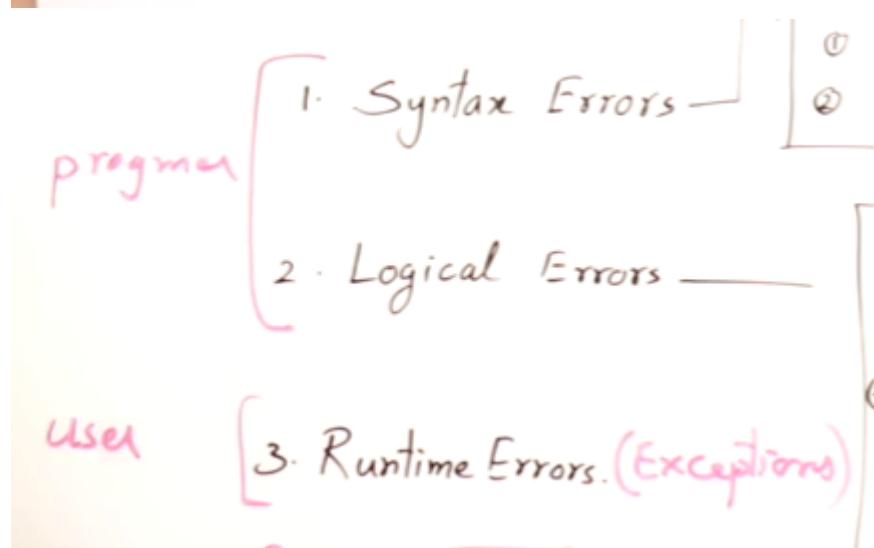
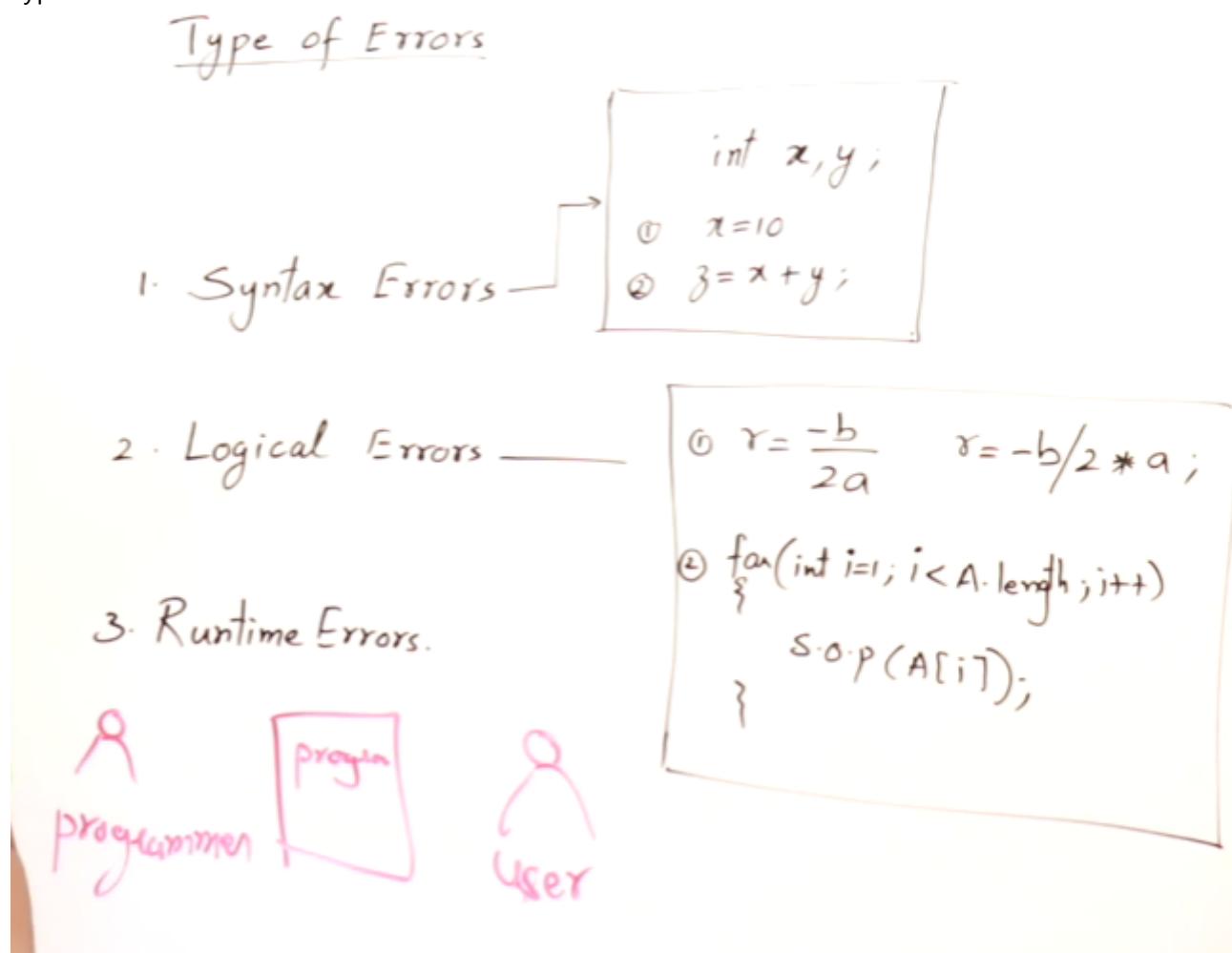


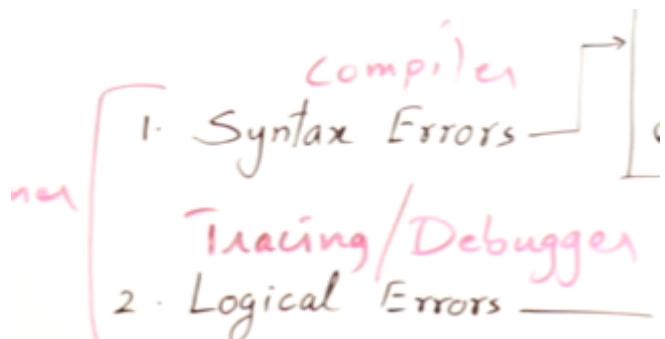
# Section 18 . Exception Handling

## 166 what are exception

= exceptions are the runtime errors

Types of Errors





invalid inputs required resources are not available user faces the issues because of mishandling of the application. 1.Bad inputs 2. Network issues 3. File not found 4. Database not available 5. Memory issues 6. Arithmetic issues 7. Null pointer exception 8. Array index out of bound exception 9. Class not found exception etc...

but programmer can give the message to the user.

## 167. How to handle the exceptions

### Exception Handling Construct

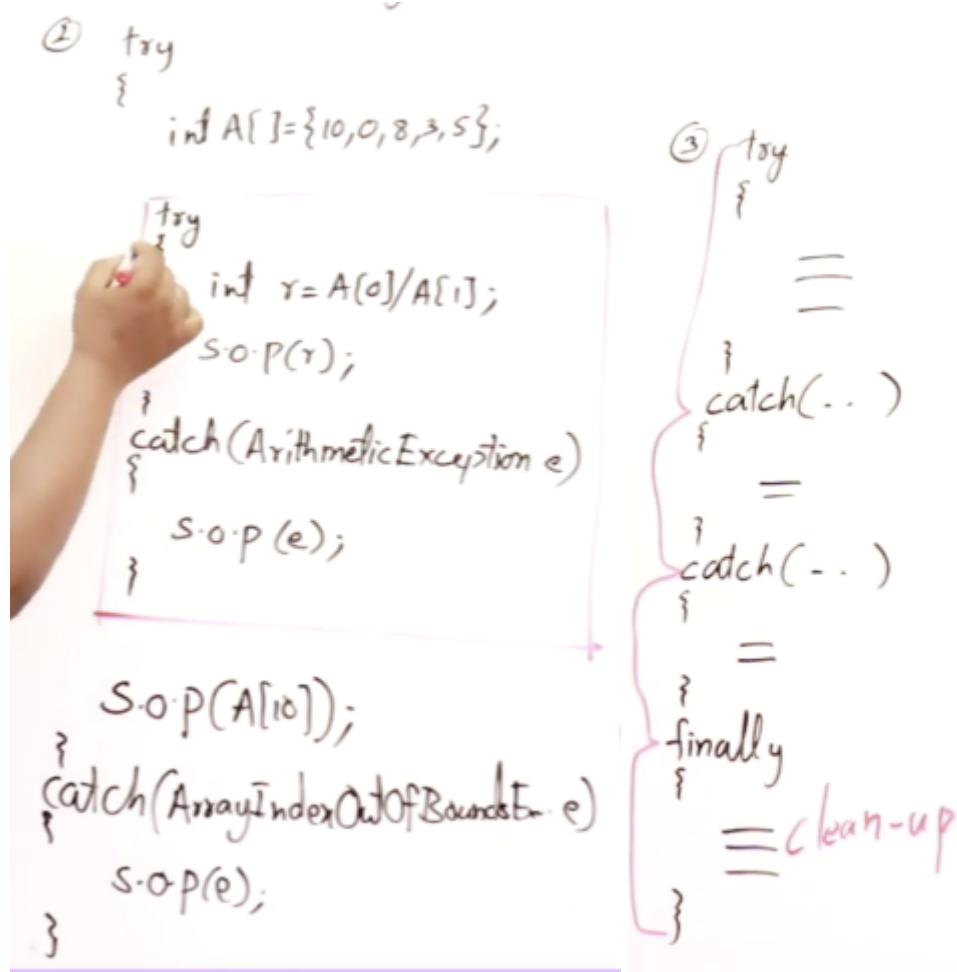
```
class Test
{
    p.s.v.main(- -)
    {
        int a,b,c;
        a=5;
        b=0;
        c=a/b;
        s.o.p(c);
        s.o.p("End of program");
    }
}
```

terminate abnormally.

dividing by 0 exception : program will be

```
class Test
{
    p.s.v main(..)
    {
        int a,b,c;
        try
        {
            a=10;
            b=2;
            → c=a/b;
            ✓ s.o.p("Result is "+c);
        }
        catch (ArithmaticException e)
        {
            s.o.p("Division by zero"+e);
        }
    }
}
```

① try {  
    int A[] = {10, 0, 8, 3, 5};  
    int r;     <sup>1</sup> 0 1  
    → r =  $\frac{A[0]}{A[1]}$ ;  
    s.o.p(r);  
    s.o.p(A[10]);  
}  
catch (ArithmetiException e)  
{  
    s.o.p(e);  
}  
catch (ArrayIndexOutOfBoundsException e)  
{  
    s.o.p(e);  
}



## 168. Try and catch block

## 169. Multiple and Nested Try and Catch block

```

int A[]={30,20,10,40,0};

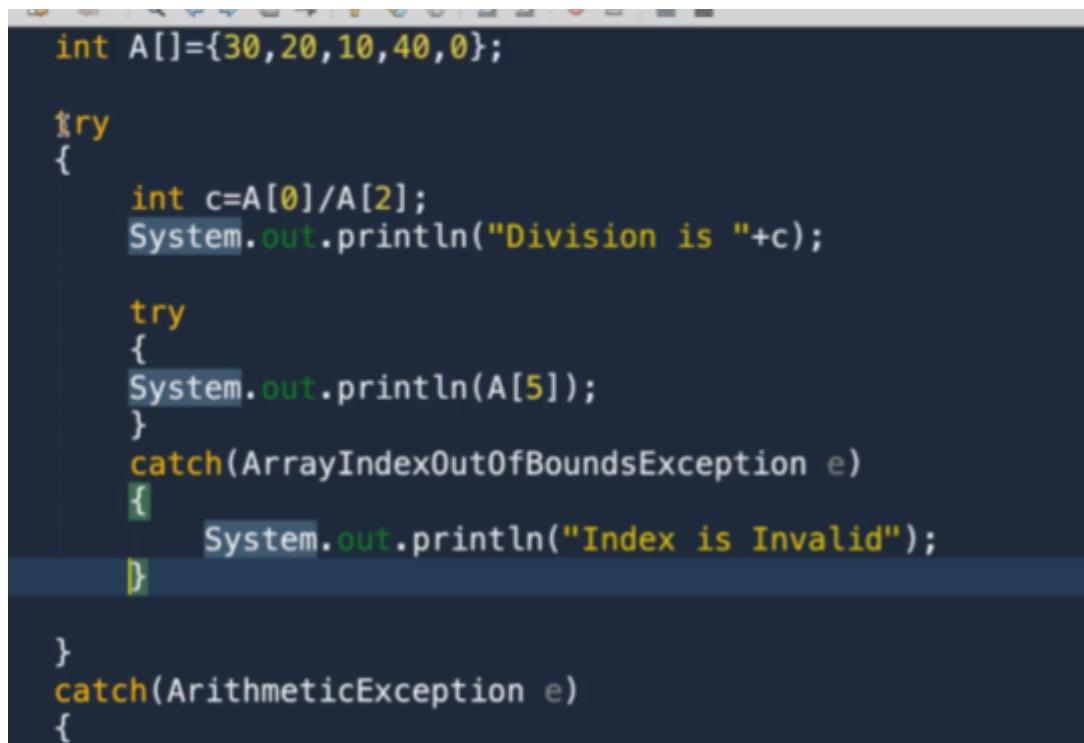
try
{
    int c=A[0]/A[2];
    System.out.println("Division is "+c);
}

System.out.println(A[5]);
}

catch(ArithmetricException e)
{
    System.out.println("Denominator should not be 0");
}

catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Index is Invalid");
}

System.out.println("Bye");
  
```

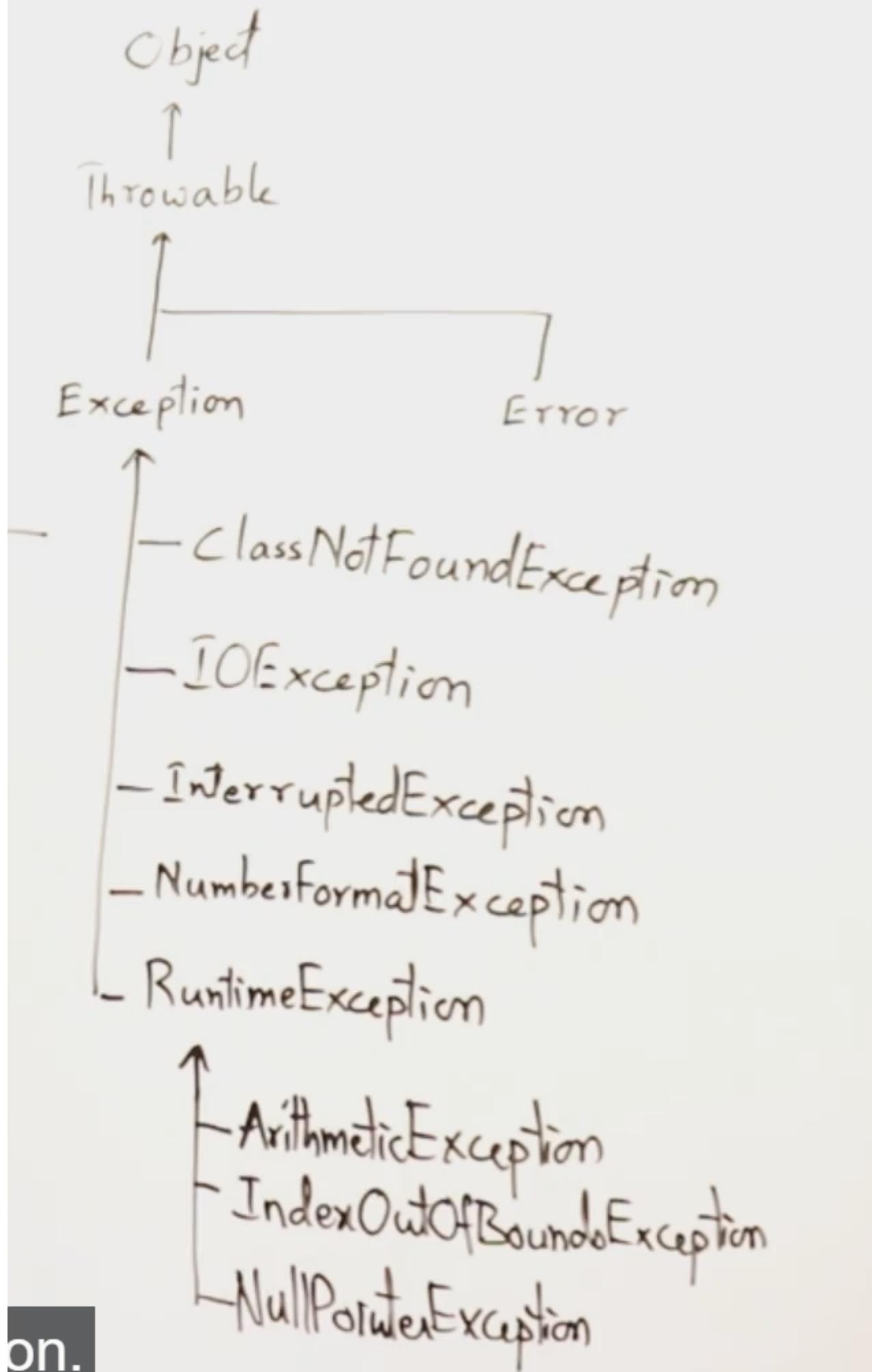


A screenshot of a Java IDE showing a code editor with the following Java code:

```
int A[]={30,20,10,40,0};  
  
try  
{  
    int c=A[0]/A[2];  
    System.out.println("Division is "+c);  
  
    try  
    {  
        System.out.println(A[5]);  
    }  
    catch(ArrayIndexOutOfBoundsException e)  
    {  
        System.out.println("Index is Invalid");  
    }  
}  
catch(ArithmeticException e)  
{
```

## 170. Class Exception

Java having a lot of built in classes for handling the exceptions.

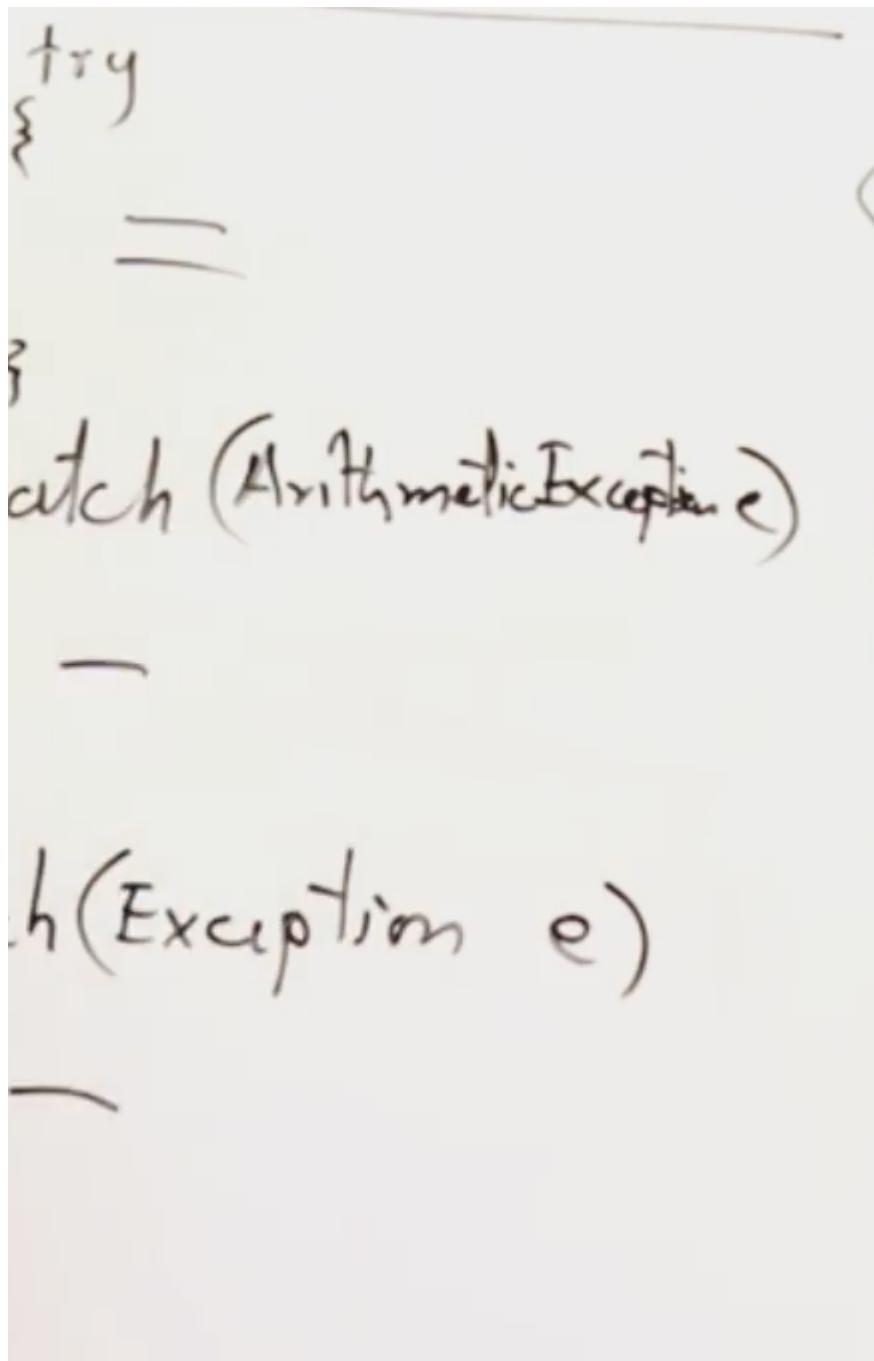


- checked

- unchecked

class Exception

- string getMessage()
- String toString()
- void printStackTrace()



```
try  
{  
}  
  
catch (ArithmaticException e)  
  
}
```

```
class Exception  
{  
    String getMessage()  
    String toString()  
    void printStackTrace()  
}
```

creating own exceptions

## 171. Checked and Unchecked Exceptions

```
package checkedunchecked;

public class CheckedUnchecked
{
    static void fun1()
    {

    }

    static void fun2()
    {
        fun1();
    }

    static void fun3()
    {
        fun2();
    }

    public static void main(String[] args)
    {
        exception
    }
}
```

comes as sequence of the methods it raised. //printStackTrace() method is used to print the sequence of the methods it raised. // .getMessage() method is used to print the message of the exception.

```
public class CheckedUnchecked
{
    static void fun1()
    {
        try
        {
            System.out.println(10/0);
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
            e.printStackTrace();    I
        }
    }

    static void fun2()
    {
        fun1();
    }

    static void fun3()
    {
        fun2();
    }
}
```

```
class LowBalanceException extends Exception
{
    public String toString()
    {
        return "Balance Should not be less than 5000";
    }
}

public class CheckedUnchecked
{
    static void fun1()
    {
        try
        {
            throw new LowBalanceException();
        }
        catch(LowBalanceException e)
        {
            System.out.println(e);
        }
    }
}
```

```
package checkedunchecked;

class LowBalanceException extends Exception
{
    public String toString()
    {
        return "Balance Should not be less than 5000";
    }
}

public class CheckedUnchecked
{
    static void fun1()
    {
        try
        {
            throw new LowBalanceException();
        }
        catch(LowBalanceException e)
        {
            System.out.println(e);
        }
    }

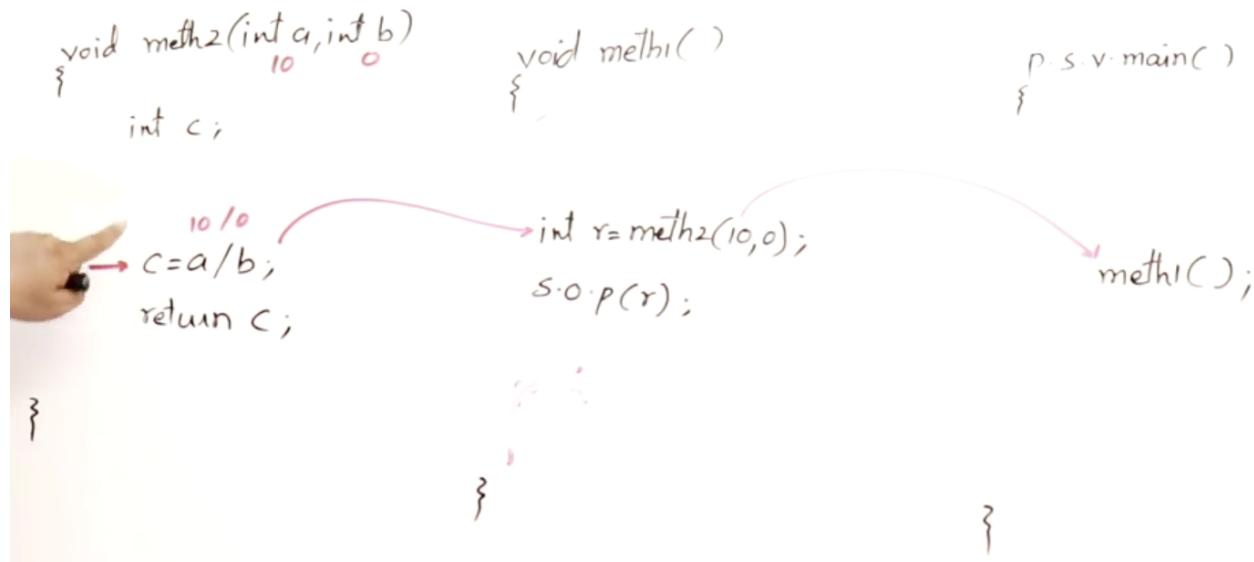
    static void fun2()
    {
        fun1();
    }
}
```

```
static void fun3()
{
    fun2();
}

public static void main(String[] args)
{
    fun3();
}
}
```

## 172. Throws and Throw

how an exception is propagated from one method to another method.



program crashes if the exception is not handled.

```

int area(int l, int b)
{
    if(l<0 || b<0)           int meth1()
        throw new Exception("—");
    int a=l*b;
    return a;
}

int a=area(-10,5);
s.o.p(a);
meth1();
}
}

```

int area(int l, int b) throws Exception

```

{
    if(l<0 || b<0)           int meth1()
        throw new Exception("—");
}

```

| Throw ✘ Throws                                 |
|------------------------------------------------|
| <u>int area(int l, int b) throws Exception</u> |
| {                                              |
| if(l<0    b<0)           int meth1()           |
| throw new Exception("—");                      |
| int a=l*b;                                     |
| return a;                                      |
| }                                              |
| try                                            |
| int a=area(-10,5);                             |
| s.o.p(a);                                      |
| catch(Exception e)                             |
| s.o.p(e);                                      |
| }                                              |
| meth1();                                       |
| }                                              |

int area(int l, int b) throws Exception

```

if(l<0 || b<0)      { int meth1() throws Exception
    throw new Exception("—"); } p.s.v. main() throws Exception
                        { int a=area(-10,5);
    int a=l*b;           meth1();
    return a;           s.o.p(a);
                        { catch(Exception e)
                            { s.o.p(e); }
}
}

```

## Throw vs Throws

```

class NegativeDimensionException extends Exception
{
    public void toString()
    {
        return "Dimension cannot be Negative";
    }
}

```

int area(int l, int b) throws NegativeDimensionException

```

if(l<0 || b<0)
    throw new NegativeDimensionException();
}

```

```

int a=l*b;
return a;
}

```

## Throw vs Throws

```
class NegativeDimensionException extends Exception
{
    public void String toString()
    {
        return "Dimension cannot be Negative";
    }
}
```

```
int area(int l,int b) throws NegativeDimensionException
{
    if(l<0 || b<0)
        throw new NegativeDimensionException();
    int a=l*b;
    return a;
}
```

propagation of exception

```
package throwthrowsdemo;

public class ThrowThrowsDemo
{
    static int meth1()
    {
        return 10/2;
    }
    static void meth2()
    {
        meth1();
    }
    static void meth3()
    {
        meth2();
    }
    public static void main(String[] args)
    {
        meth3();
    }
}
```

below is unchecked

exception (Arithmetic exception)

```
package throwthrowsdemo;

public class ThrowThrowsDemo
{
    static int meth1()
    {
        return 10/0;
    }
    static void meth2()
    {
        meth1();
    }
    static void meth3()
    {
        meth2();
    }
    public static void main(String[] args)
    {
        meth3();
    }
}
```

```
public static void main(String[] args)
{
    try
    {
        meth3();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
```

handled in

```
public class ThrowThrowsDemo
{
    static int area(int l,int b) throws Exception
    {
        if(l<0 || b<0)
            throw new Exception();
        return l*b;
    }
    static void meth1() throws Exception
    {
        System.out.println("Area is "+area(10,5));
    }

    public static void main(String[] args) throws Exception
    {
        meth1();
    }
}
```

```
package throwthrowsdemo;

class NegativeDimensionException extends Exception
{
    public String toString()
    {
        return "Dimensions of a Rectangle cannot be Negative";
    }
}

public class ThrowThrowsDemo
{
    static int area(int l,int b) throws NegativeDimensionException
    {
        if(l<0 || b<0)
            throw new NegativeDimensionException();
        return l*b;
    }
    static void meth1() throws NegativeDimensionException
    {
        System.out.println("Area is "+area(10,5));
    }

    public static void main(String[] args)
    {
        try
        {
            meth1();
        }
        catch(NegativeDimensionException e)
        {

```

```
        System.out.println(e);
    }
    System.err.println("hi");
}
}
```

## 173. Finally block

```
public class FinallyDemo
{
    public static void main(String[] args) throws Exception
    {
        try
        {
            System.out.println(10/0);
        } finally
        {
            System.out.println("Final Message");
        }
    }
}
```

```
public static void main(String[] args) throws Exception
{
    try
    {
        System.out.println(10/0);
    } catch(ArithmeticException e)
    {
        System.out.println(e);
    } finally
    {
        System.out.println("Final Message");
    }
}
```

```
package finallydemo;

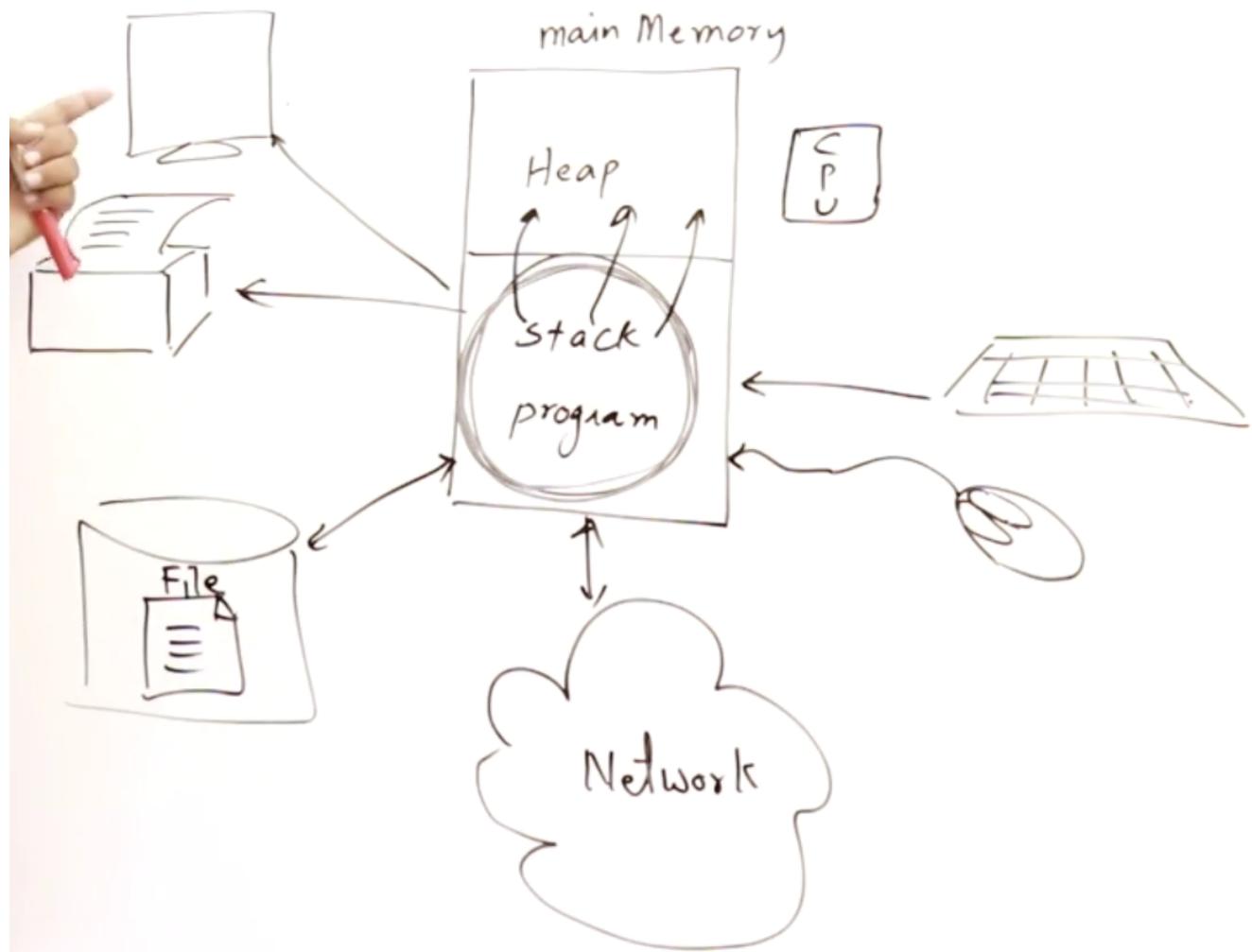
public class FinallyDemo
{
    static void meth1()throws Exception
    {
        try
        {
            throw new Exception();
        }
        finally
        {
            System.out.println("Final Message");
        }
    }

    public static void main(String[] args) throws Exception
    {
        meth1();
    }
}
```

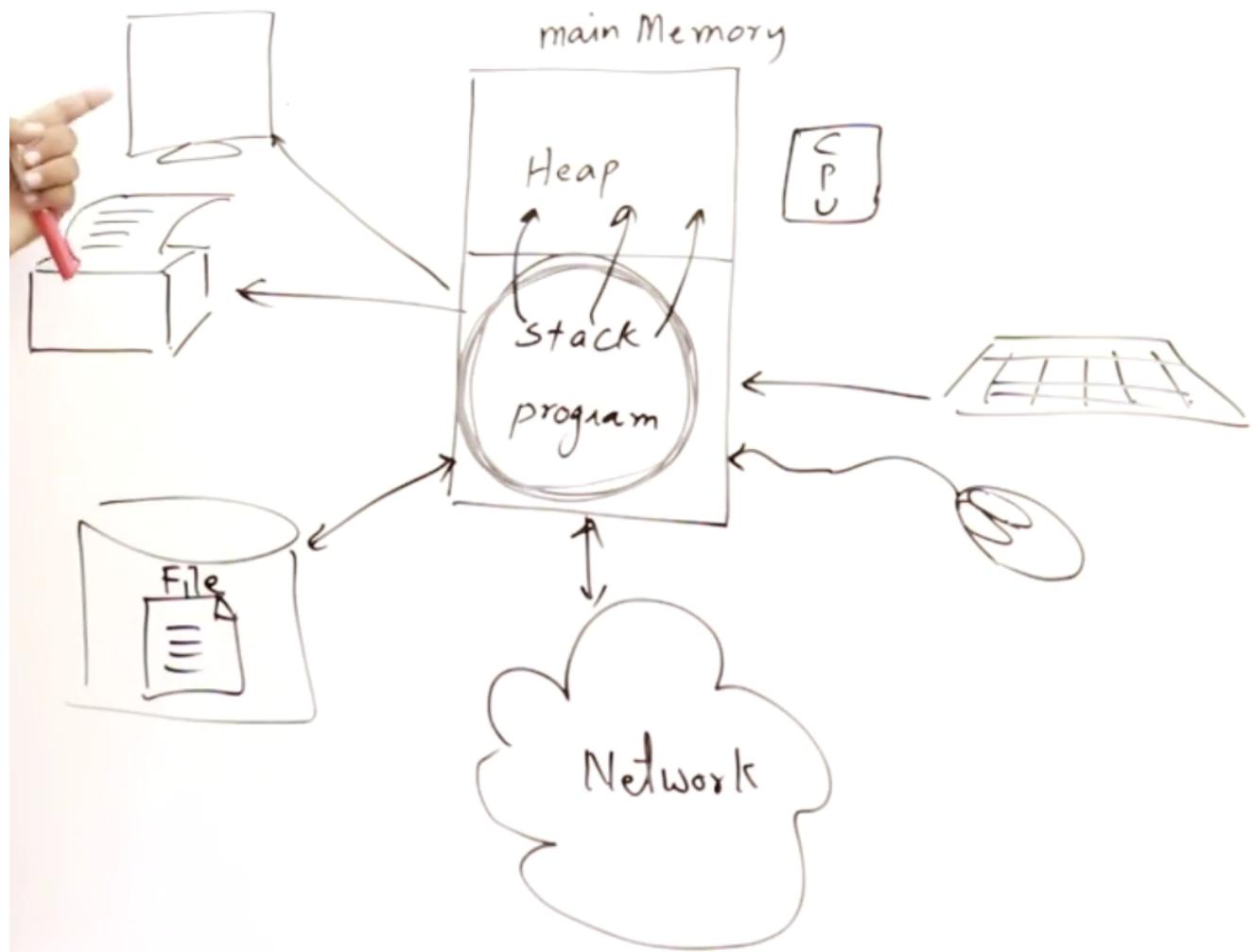
## Try with resource

1.7 version introduced

## Try with Resources



## Try with Resources



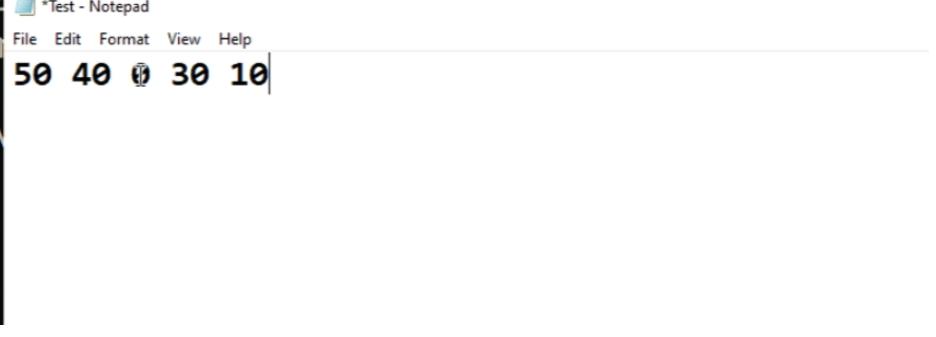
```
int meth1() throws Exception  
{  
    FileReader fi;  
  
    try {  
        ✓ → f = new FileReader("my.txt");  
        ✓ → // use file  
        ✗ f.close();  
        ✗ return result;  
    } finally {  
        f.close();  
    }  
}
```

```
int method() throws Exception  
{  
    try (FileReader f = new FileReader("my.txt"))  
    {  
        // use file  
        return result;  
    }  
}
```

to avoid the finally block for closing the resources then use the try with resource

if a resource has the close method then it is called as the auto closable resource. then it is used in the try with resource block.

## File handling



```
Z:\NetBeansProjects\NestedInner\build\classes\nestedinner
Volume in drive Z: is NTFS
Volume Serial Number is 50 40 0 30 10

Directory of Z:\NetBeansProjects\NestedInner\build\classes\nestedinner

02/27/2020  03:06
02/27/2020  03:06
02/27/2020  03:06

package resourcesdemo;
import java.io.*;
import java.util.*;

public class ResourcesDemo
{
    static FileInputStream fi;

    static void Divide() throws FileNotFoundException
    {
        fi=new FileInputStream("/Users/abdulbari/Desktop/Test.txt");

        Scanner sc=new Scanner(fi);
        int a=sc.nextInt();
        int b=sc.nextInt();
        int c=sc.nextInt();
        System.out.println(a/b);
    }

    public static void main(String[] args) throws Exception
    {
        Divide();
    }
}
```

```
public class ResourcesDemo
{
    static FileInputStream fi;

    static void Divide() throws Exception
    {
        fi=new FileInputStream("/Users/abdulbari/Desktop/Test.txt");

        Scanner sc=new Scanner(fi);
        int a=sc.nextInt();
        int b=sc.nextInt();
        int c=sc.nextInt();
        System.out.println(a/b);

        fi.close();
    }

    public static void main(String[] args) throws Exception
    {
        Divide(); 
    }
}
```

```
package resourcesdemo;
import java.io.*;
import java.util.*;

public class ResourcesDemo
{

    static void Divide() throws Exception
    {

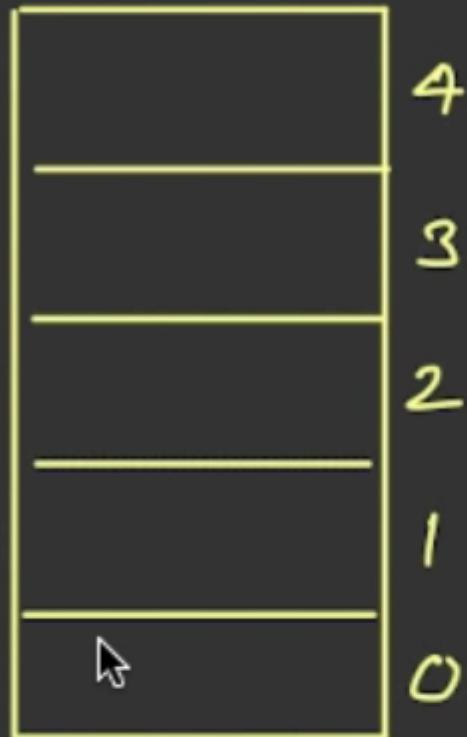
        try(FileInputStream fi=new
FileInputStream("/Users/abdulbari/Desktop/Test.txt");Scanner sc=new Scanner(fi) )
        {
            int a=sc.nextInt();
            int b=sc.nextInt();
            int c=sc.nextInt();
            System.out.println(a/c);
        }

    }

    public static void main(String[] args) throws Exception
    {
        try
        {
            Divide();
        }
        catch(Exception e)
    }
}
```

```
{  
    System.out.println(e);  
}  
  
//int x=sc.nextInt();  
  
// System.out.println(x);  
  
}
```

$size = 5$



$\leftarrow top = -1$

Stack  
LIFO

user defined exception

## User Defined Exception

$\text{size} = 5 -$

|    |   |                         |
|----|---|-------------------------|
| 12 | 4 | $\leftarrow \text{Top}$ |
| 4  | 3 |                         |
| 17 | 2 |                         |
| 20 | 1 |                         |
| 15 | 0 |                         |

if ( $\text{Top} == \text{size} - 1$ )  
StackOverflow

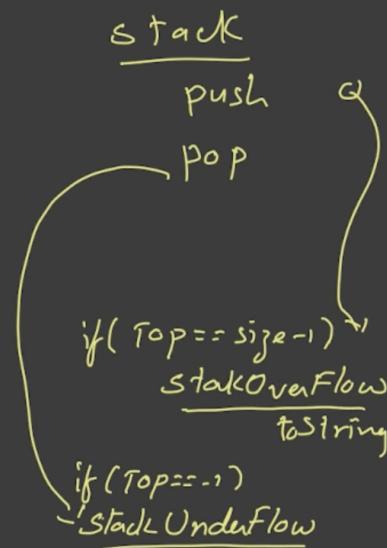
if ( $\text{Top} == -1$ )  
StackUnderflow

Stack  
LIFO

## User Defined Exception

$\text{size} = 5 -$

- ① StackOverflow
- ② StackUnderflow
- ③ Stack



|    |   |                         |
|----|---|-------------------------|
| 4  | 3 | $\leftarrow \text{Top}$ |
| 4  | 3 |                         |
| 17 | 2 |                         |
| 20 | 1 |                         |
| 15 | 0 |                         |

$\rightarrow$  Stack  
LIFO

```

class StackOverflow extends Exception
{
    public String toString()
    {
        return "Stack is Full";
    }
}
  
```

```
}

class StackUnderFlow extends Exception
{
    public String toString()
    {
        return "Stack is Empty";
    }
}

class Stack
{
    private int size;
    private int top=-1;
    private int S[];

    public Stack(int sz)
    {
        size=sz;
        S=new int[sz];
    }

    public void push(int x) throws StackOverFlow
    {
        if(top==size-1)
            throw new StackOverFlow();
        top++;
        S[top]=x;
    }

    public int pop() throws StackUnderFlow
    {
        int x=-1;

        if(top==-1)
            throw new StackUnderFlow();
        x=S[top];
        top--;
        return x;
    }
}

public class SCException1
{
    public static void main(String[] args)
    {
        Stack st=new Stack(5);
        try
        {
            st.push(10);
        }
```

```
    st.push(15);
    st.push(10);
    st.push(15);
    st.push(10);
    st.push(15);

}
catch(StackOverFlow s)
{
    System.out.println(s);
}

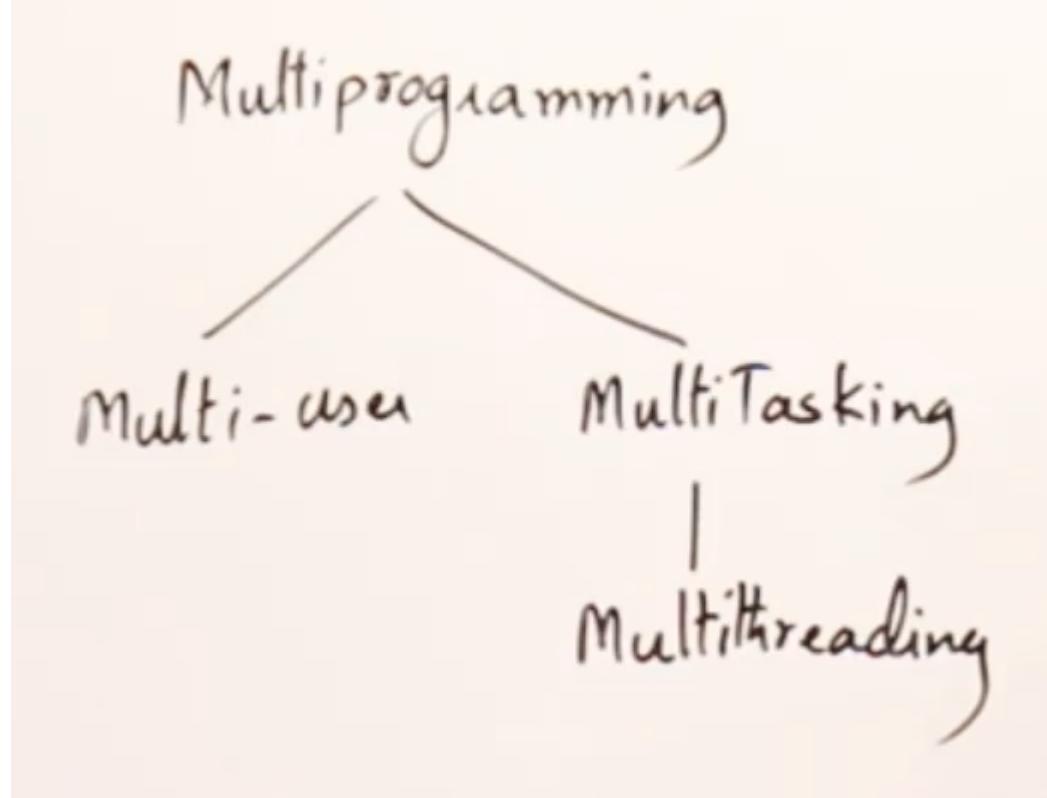
}
}
```

## Quiz

## Section 19. Multithreading

---

1. Multi Programming
2. Why Multithreading?
3. Control Flow of a Program
4. Thread in Java

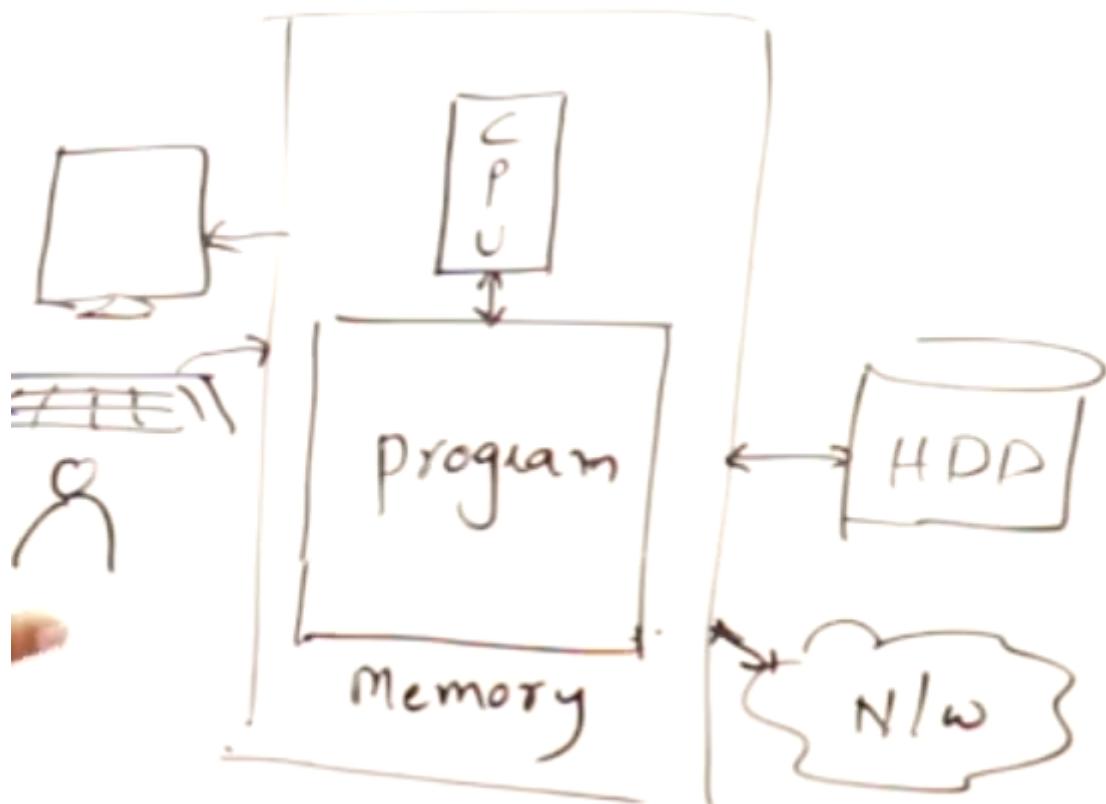


multiprogramming =

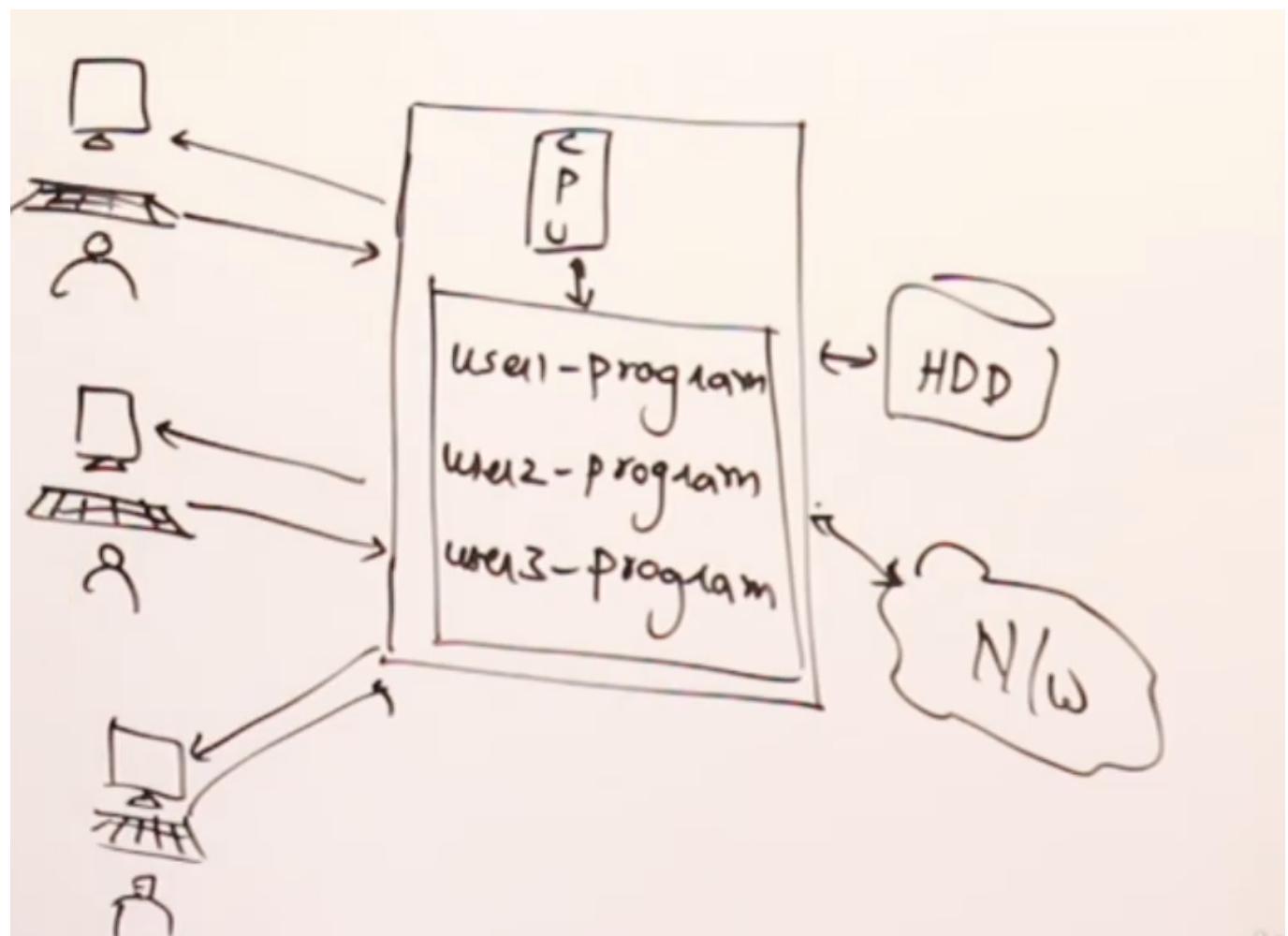
running many programs in a single machine there are multiple forms of multiprogramming

1. multi-user = more than one user using the same machine simultaneously.
2. multitasking = more than one task is running in the same machine simultaneously by single user.

multithreading is a type of multitasking. multiple threads are running



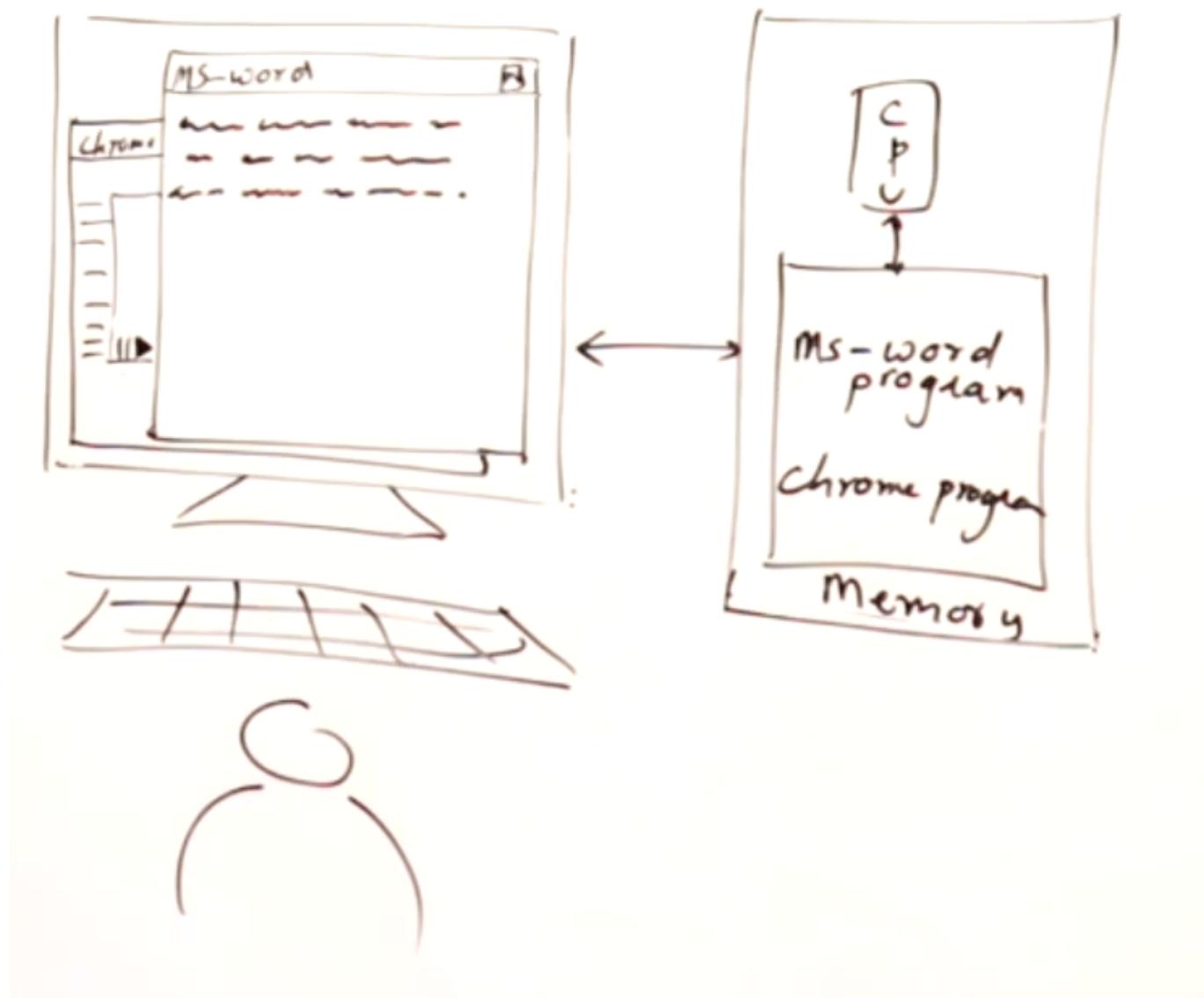
like a round robin fashion



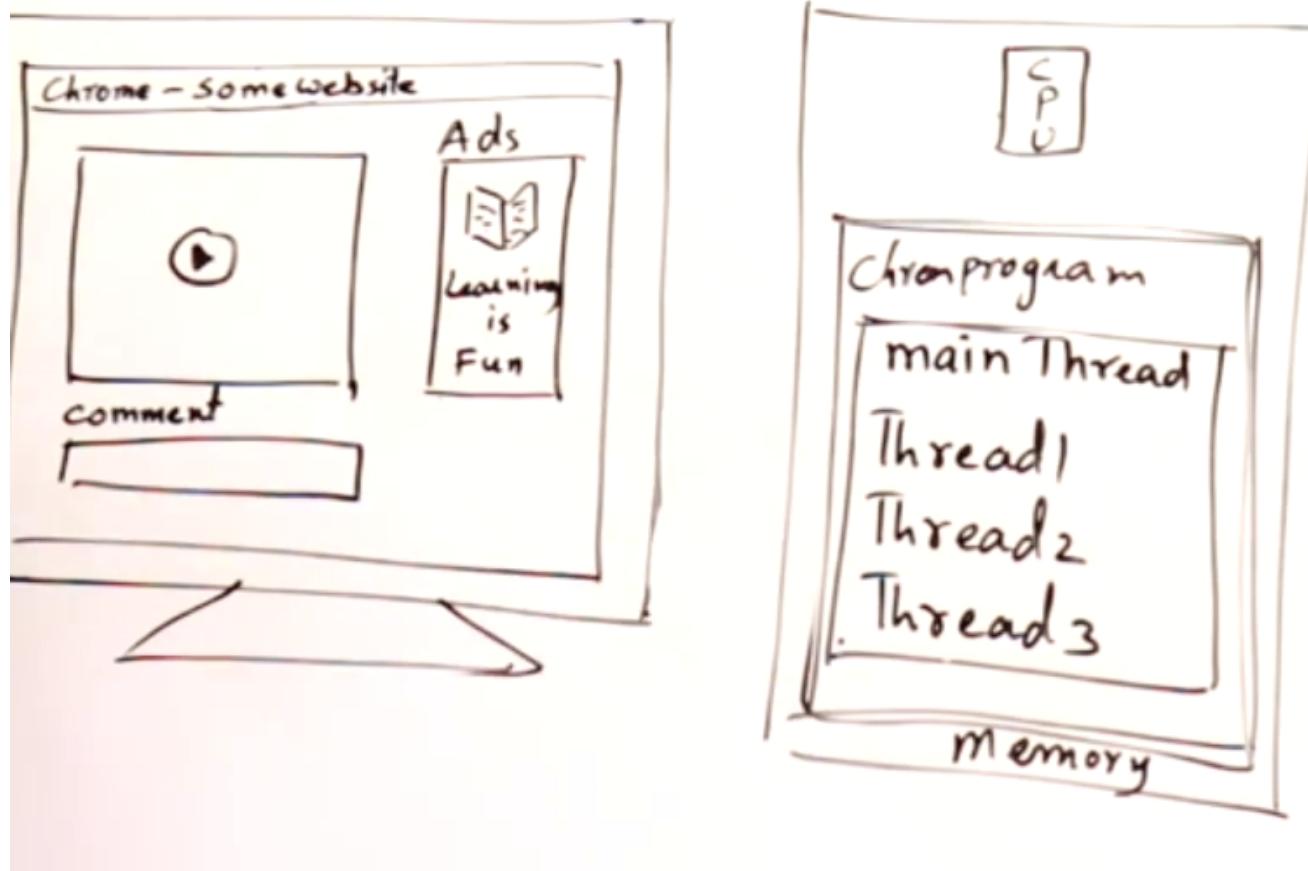
multi user days are gone

today we are using multitasking and multithreading

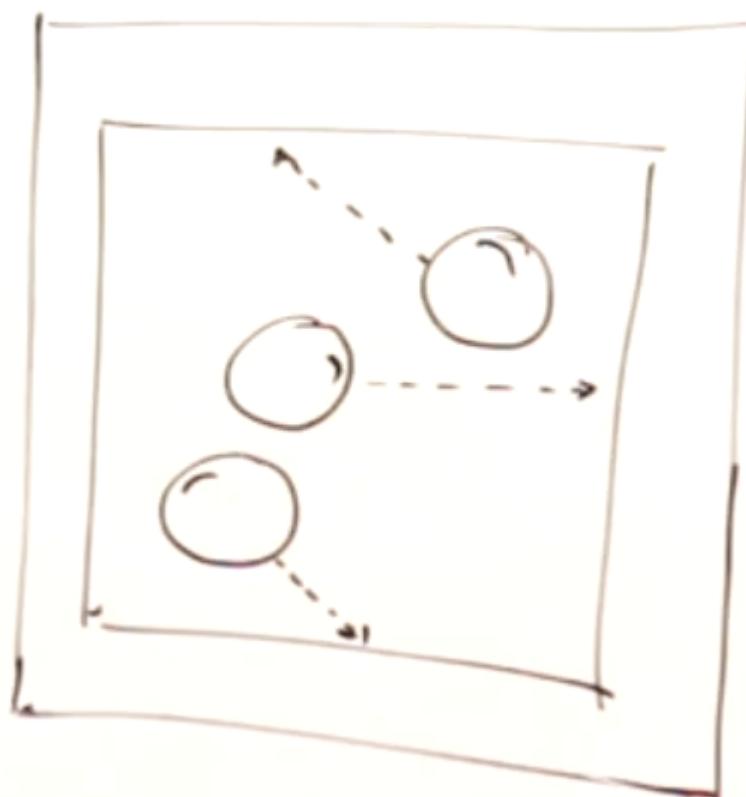
## 179. Multi tasking



## MULTI THREADING



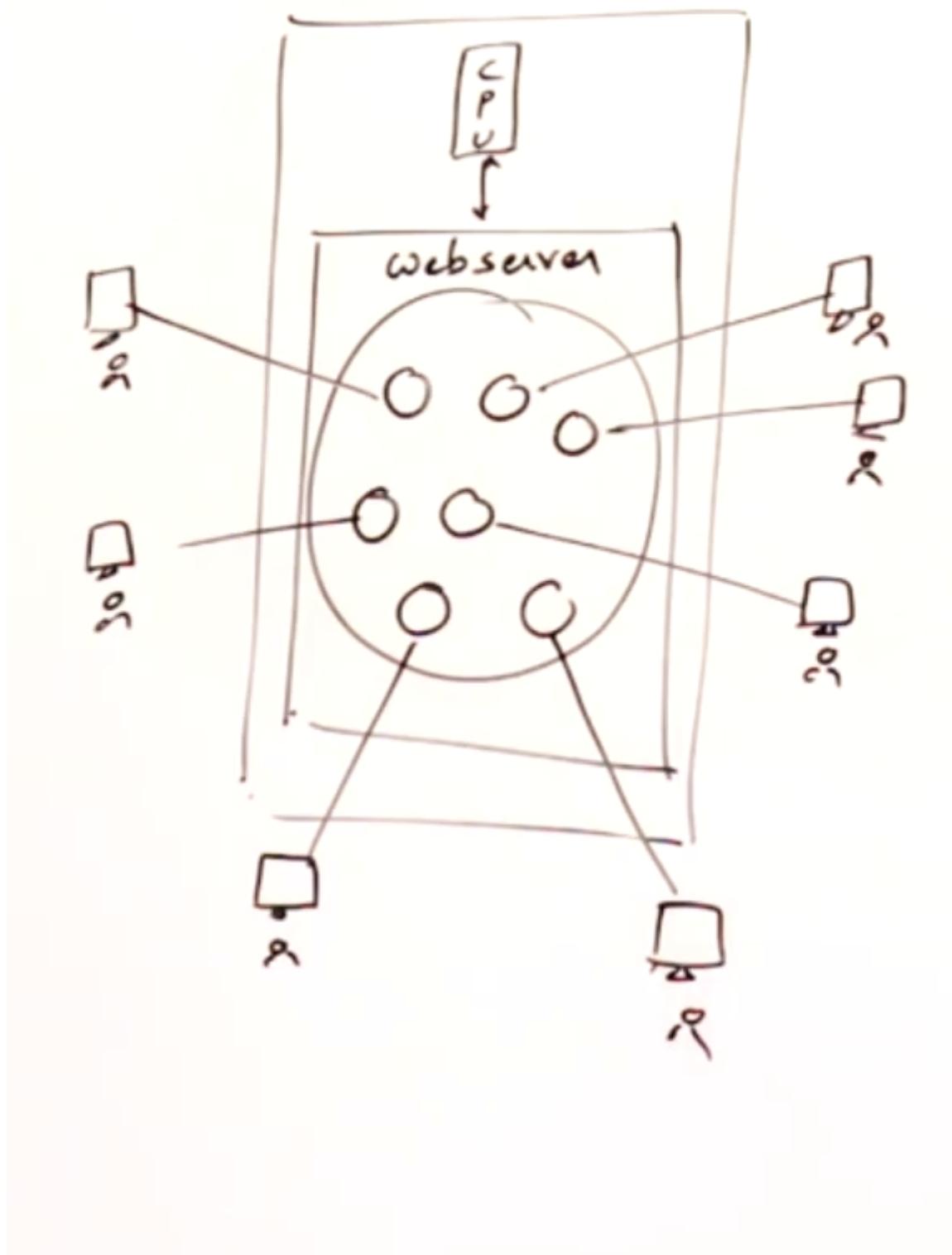
## Animation



EACH ball uses one thread

Game





control flow of java program

```
class Test
{
    static void display()
    {
        S.o.p("Hello");
    }

    P.s.v.main(..)
    {
        display();
        S.o.p("world");
    }
}
```

single control flow (single program )(single thread)

```
class Test
{
    static void display()
    {
        System.out.println("Hello");
    }

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

```
class Test
{
    static void display()
    {
        int i=1;
        while(true)
        {
            S-O-P(i+"Hello");
            i++;
        }
    }

    P-S-V main()
    {
        display();
        int i=1;
        while(true)
        {
            S-O-P(i+"world");
            i++;
        }
    }
}
```

require simultaneously run

```
- class Test
{
    static void display()
    {
        int i=1;
        while(true)
        {
            s.o.p(i+"Hello");
            i++;
        }
    }

    public static void main()
    {
        display();
        int i=1;
        while(true)
        {
            s.o.p(i+"world");
            i++;
        }
    }
}
```

## 181. multithreading

to achieve multithreading

1. Thread class

2. Runnable Interface

actual mechanism of multithreading is present inside the Thread class

even runnable interface is used then there is a thread class inside the runnable interface.

```
class MyThread extends Thread  
{  
    public void run()  
    {  
        int i=1;  
        while(true)  
        {  
            System.out.println(i+" Hello");  
            i++;  
        }  
    }  
}
```

```
class Test  
{  
    public static void main()  
    {  
        MyThread t=new MyThread();  
        t.start();  
        int i=1;  
        while(true)  
        {  
            System.out.println(i+" world");  
            i++;  
        }  
    }  
}
```

```
class Test extends Thread  
{  
    public void run()  
    {  
        int i=1;  
        while(true)  
        {  
            System.out.println(i+" Hello");  
            i++;  
        }  
    }  
}
```

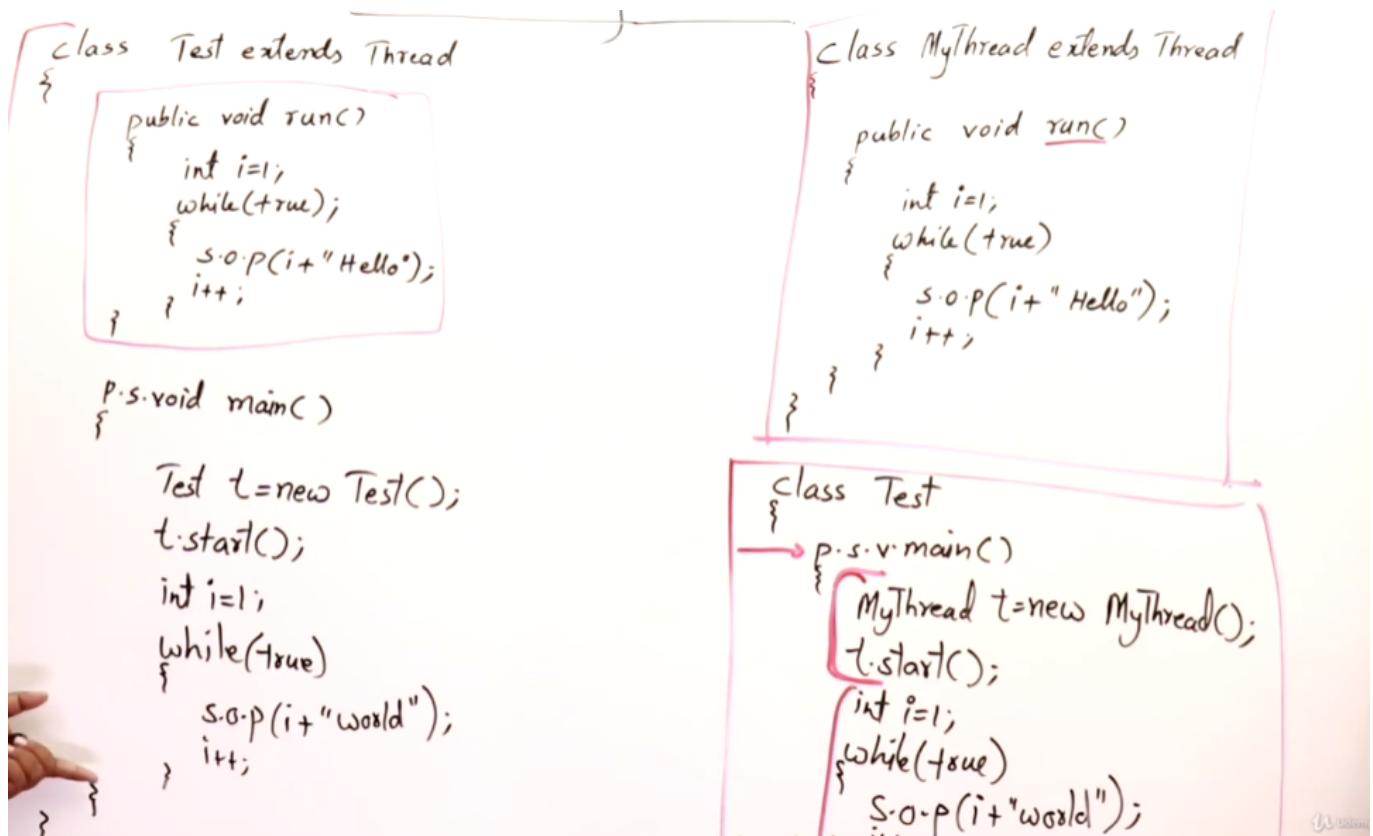
```
P.S. void main()  
{
```

```
Test t=new Test();  
t.start();
```

class has to extend Thread then override the run() method start() method is used to start the thread

```
class Main extends Thread{  
  
    public void run(){  
        int i=0;  
        while(true){  
            System.out.println(i+" WORLD");  
            i++;  
        }  
    }  
  
    public static void main(String args[]){  
  
        Main t = new Main();  
        t.start();  
        int i=0;
```

```
while(true){  
    System.out.println(i+" Hello");  
    i++;  
}  
}  
}
```



```
class My implements Runnable  
{  
    public void run()  
    {  
        int i=1;  
        while(true)  
        {  
            System.out.println(i + "Hello");  
            i++;  
        }  
    }  
}  
  
class Test  
{  
    public static void main()  
    {  
        My m=new My();  
        Thread t=new Thread(m);  
        t.start();  
  
        int i=1;  
        while(true){System.out.println(i + "World"); i++;}  
    }  
}
```

RUNNABLE

class Test implements Runnable

```
{
    public void run() {
        int i=1;
        while(true) {
            System.out.println(i + "Hello");
            i++;
        }
    }
}
```

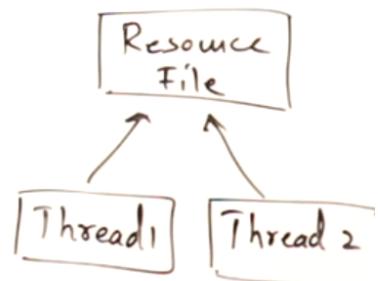
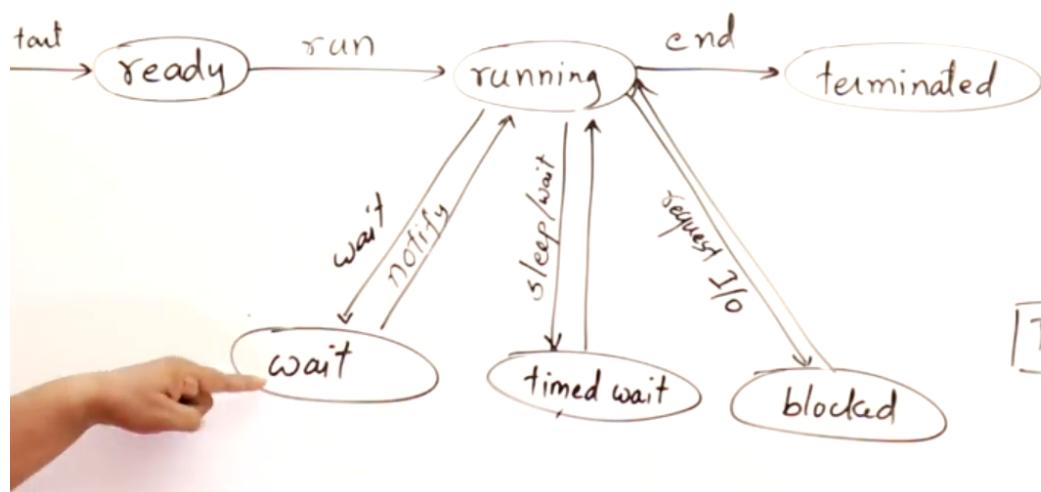
p.s.v.main()

```
{
    Test m=new Test();
    Thread t=new Thread(m);
    t.start();

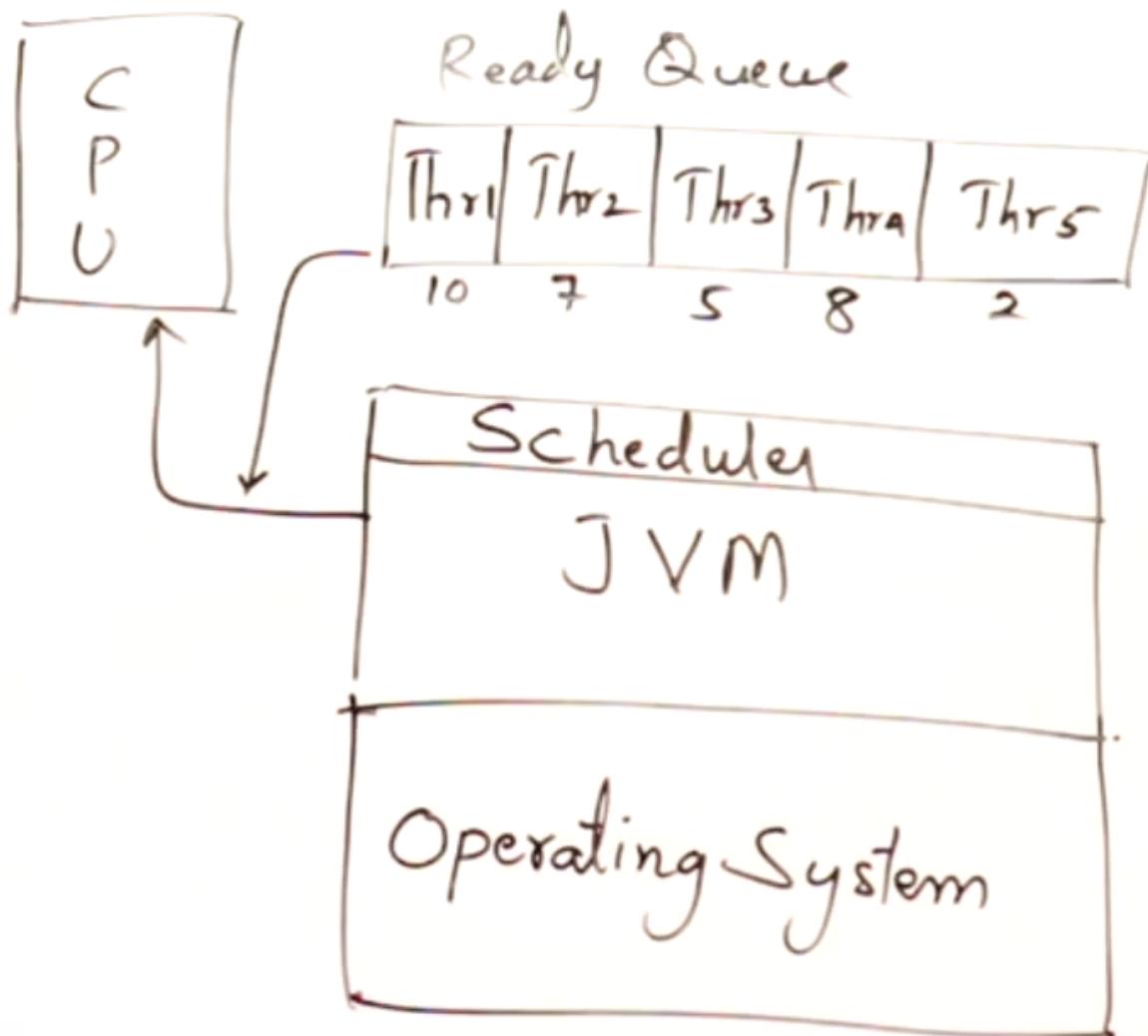
    int i=1;
    while(true){System.out.println(i + "world"); i++;}
}
```

}

## States of Thread



## Thread Properties



Thread.MIN\_PRIORITY = 1 Thread.NORM\_PRIORITY = 5 Thread.MAX\_PRIORITY = 10

Reg. sequence MS Word Thread : taking input from the keyboard Thread : spell checker & grammar checker  
Thread : Auto save

Chrome Thread : Pulling the data from the internet Thread : Rendering the data from the internet

## Thread Class

## Constructors

Thread( )

Thread(Runnable r)

Thread(Runnable r, String name)

Thread(ThreadGroup g, String name)

whenever thread is created jvm or jre will

give an id. //like ThreadGroup is used for stopping the animations

## Constructors

✓ Thread( )

✓ Thread(Runnable r)

✓ Thread(Runnable r, String name)

→ Thread(ThreadGroup g, String name)

Thread(String name)

Thread classgetXXX() / setXXX()Enquiry

long getId()

boolean isAlive()

String getName()

boolean isDaemon()

int getPriority()

boolean isInterrupted()

Thread.State getState()

ThreadGroup getThreadGroup()

void setName(String name)

void setPriority(int p)

void setDaemon(boolean d)

Instance Methods

void interrupt()

static Methods

int activeCount()

void join()

Thread currentThread()

void join(long millis)

void yield()

void run()

void dumpStack()

void start()

wait sleep

## static Methods.

- ✓ `int activeCount()`
- ✓ `Thread currentThread()`
- `void yield()`
- `void dumpStack()`

Thr1 Thr2  
10 4

`yield()` = lets give chance to lower priority thread to execute

dumpStack() => you can know the depth of the method classes and it's sequence calls.

```
package threadtest;

class MyRun implements Runnable
{
    public void run(){}
}
public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {

        Thread t=new Thread(new MyRun());
    }
}
```

```
class MyThread extends Thread
{
    public MyThread(String name)
    {
        super(name);
    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread("My Thread 1");

        System.out.println("ID "+t.getId());
        System.out.println("Name "+t.getName());
        System.out.println("Priority "+t.getPriority());
        System.out.println("State "+t.getState()); []
        System.out.println("Alive "+t.isAlive());

    }
}
```

```
class MyThread extends Thread
{
    public MyThread(String name)
    {
        super(name);
        setPriority(Thread.MIN_PRIORITY+2);
    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread("My Thread 1");

        System.out.println("ID "+t.getId());
        System.out.println("Name "+t.getName());
        System.out.println("Priority "+t.getPriority());
        t.start();
        System.out.println("State "+t.getState());
        System.out.println("Alive "+t.isAlive());

    }
}
```

```
class MyThread extends Thread
{
    public MyThread(String name)
    {
        super(name);
    }

    public void run()
    {
        int count=1;
        while(true)
        {
            System.out.println(count++);
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
        }
    }
}
```

```
while(true)
{
    System.out.println(count++);
    try
    {
        Thread.sleep(1000);
    }
    catch(InterruptedException e)
    {
        System.out.println(e);
    }
}

public class ThreadTest
{
    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread("My Thread 1");

        t.start();
        t.interrupt();
    }
}
```

## UNDERSTAND THE THREADS

```
package threadtest;

class MyThread extends Thread
{
    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);
        }
    }
}

public class ThreadTest
{
    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread();
        t.start();
    }
}
```

Daemon

```
package threadtest;

class MyThread extends Thread
{
    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);
        }
    }
}

public class ThreadTest
{
    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread();
        t.setDaemon(true);
        t.start();
    }
}
```

check

The screenshot shows a Java IDE interface with the title bar "Start Page" and "ThreadTest.java". The main window displays the following Java code:

```
10         {
11             System.out.println(count++);
12         }
13     }
14 }
15 }
16
17 public class ThreadTest
18 {
19
20     public static void main(String[] args) throws Exception
21     {
22         MyThread t=new MyThread();
23         t.setDaemon(true);
24         t.start();
25
26         try{Thread.sleep(100);}catch(Exception e){}
27     }
28 }
```

The code defines a class `ThreadTest` with a `main` method. Inside `main`, it creates a `MyThread` object, sets it as a daemon thread, and starts it. A `try`-`catch` block is present at the bottom of the `main` method, which is currently highlighted in blue.

```
join()
```

```
public class ThreadTest
{
    public static void main(String[] args) throws Exception
    {
        MyThread t=new MyThread();
        t.setDaemon(true);
        t.start();
        Thread.mainThread=Thread.currentThread();
    }
}

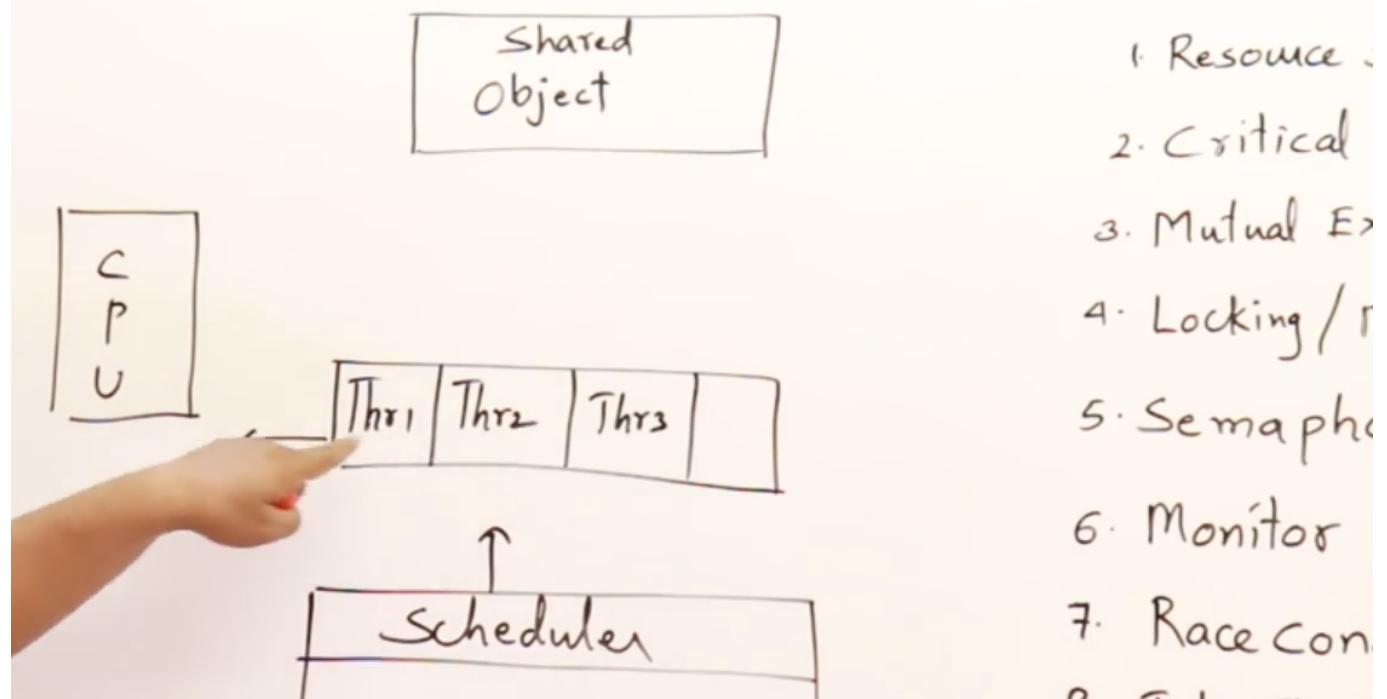
public static void main(String[] args) throws Exception
{
    MyThread t=new MyThread();
    t.start();

    int count=1;

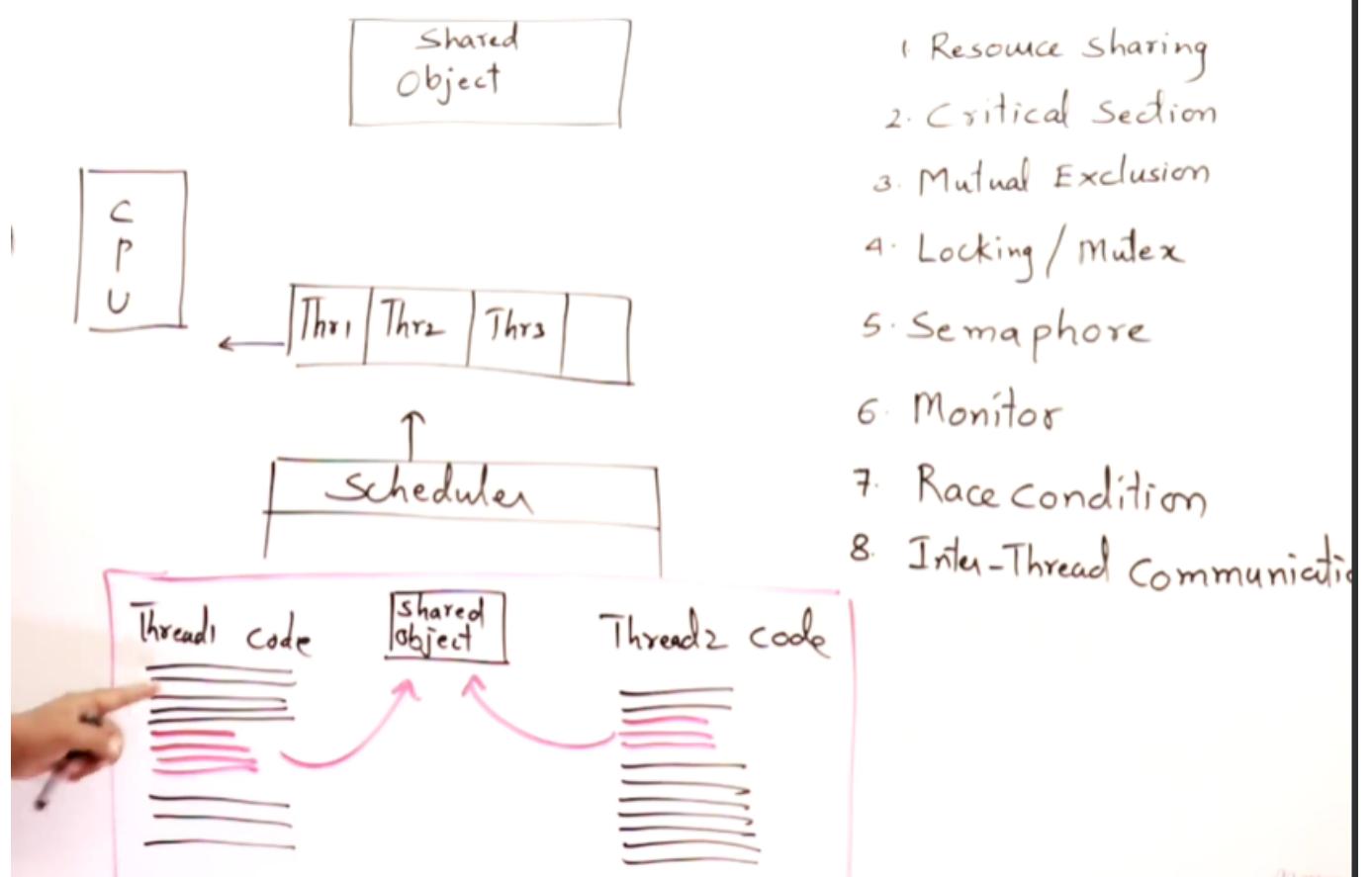
    while(true)
    {
        System.out.println(count++ +" Main");
        Thread.yield();
    }
}
```

189. Sync [concept]

it is the coordination between the threads

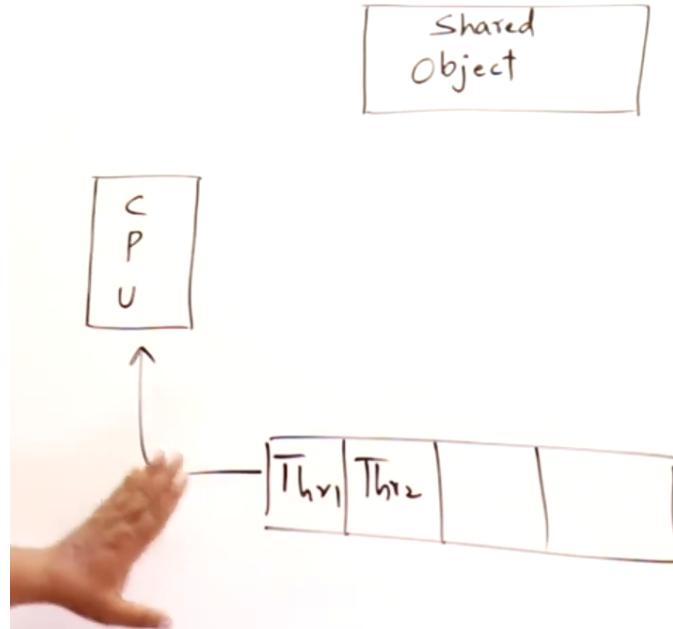


## Synchronization

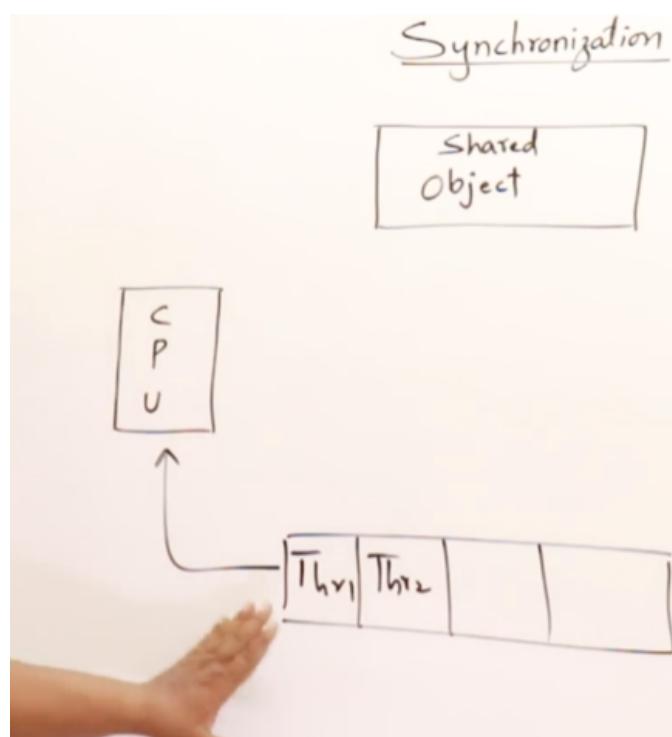


= happening of one thread prevents the other

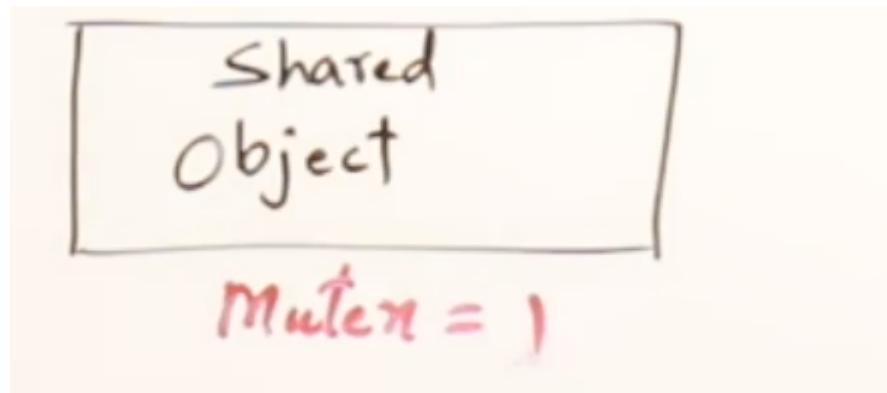
## Synchronization



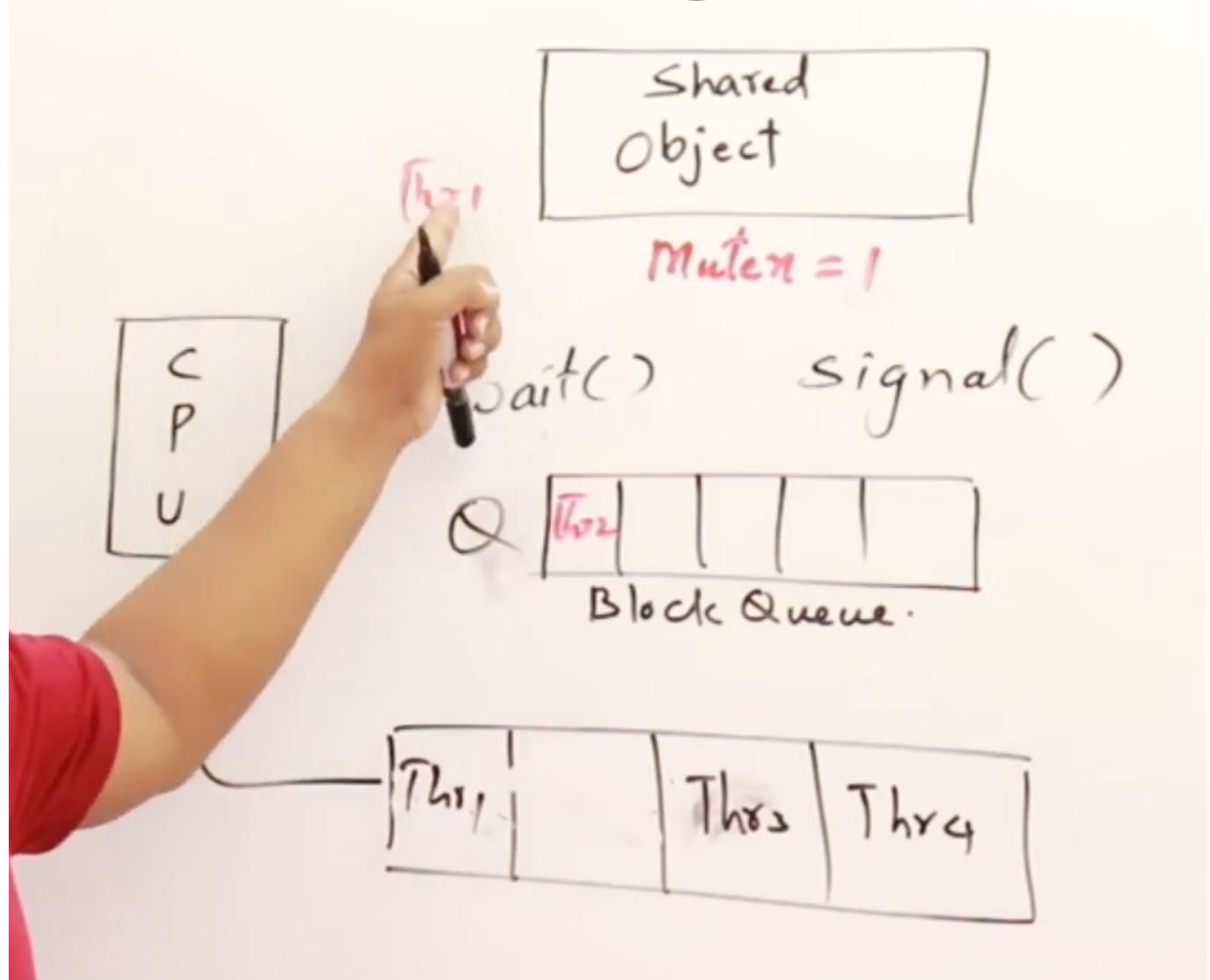
1. Resource sharing
2. Critical Section
3. Mutual Exclusion
4. Locking / Mutex
5. Semaphore
6. Monitor
7. Race Condition
8. Inter-Thread Communication

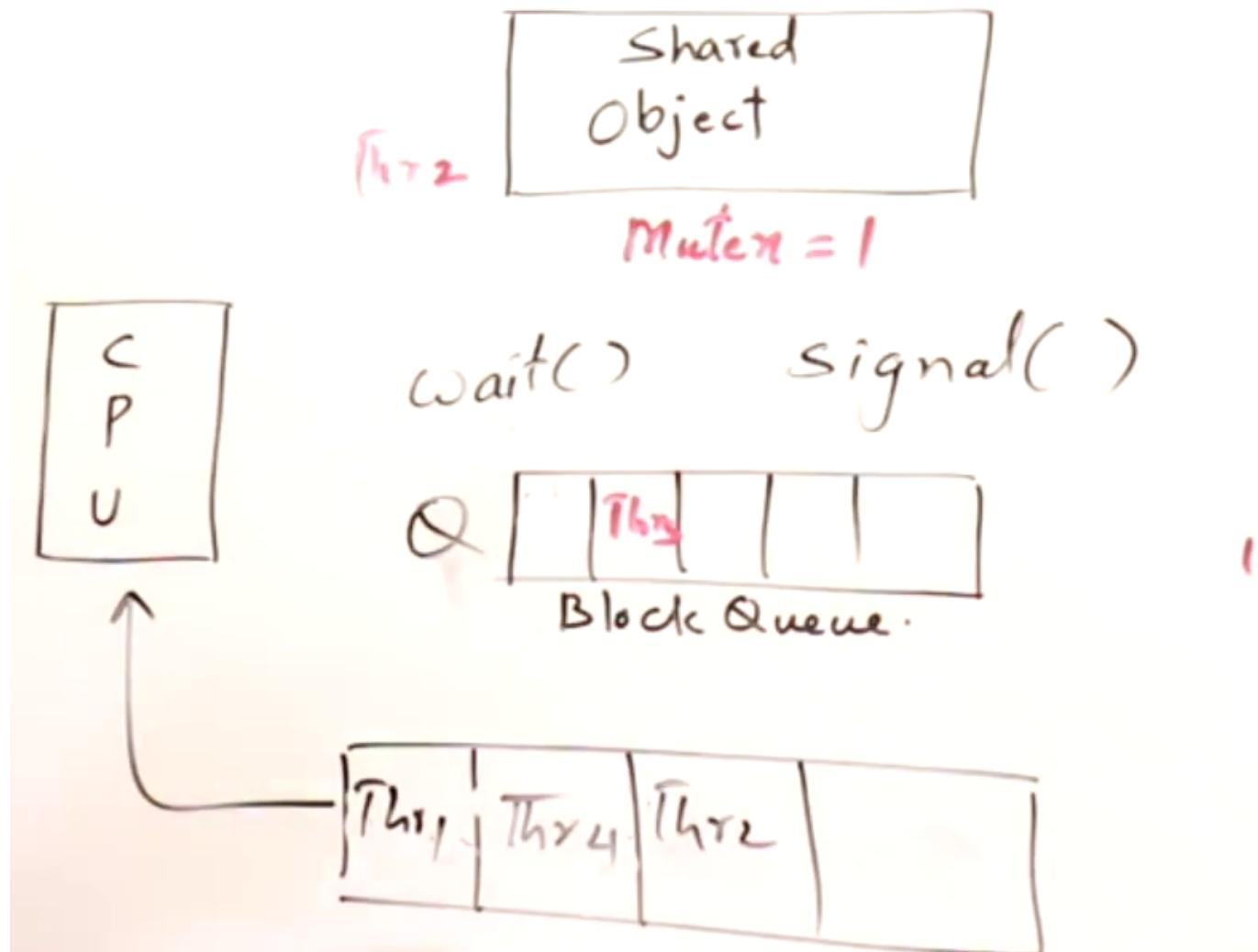


1. Resource sharing
2. Critical Section
3. Mutual Exclusion
4. Locking / Mutex
5. Semaphore
6. Monitor
7. Race Condition
8. Inter-Thread Communication

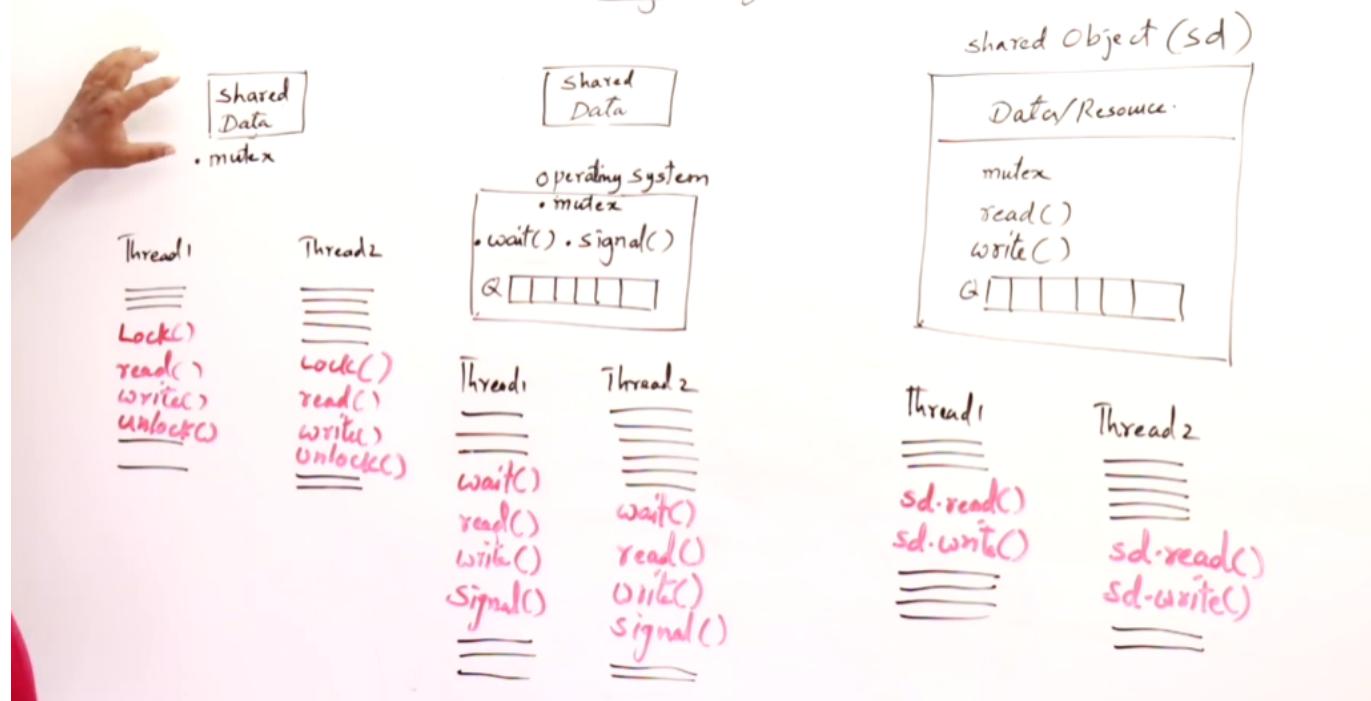


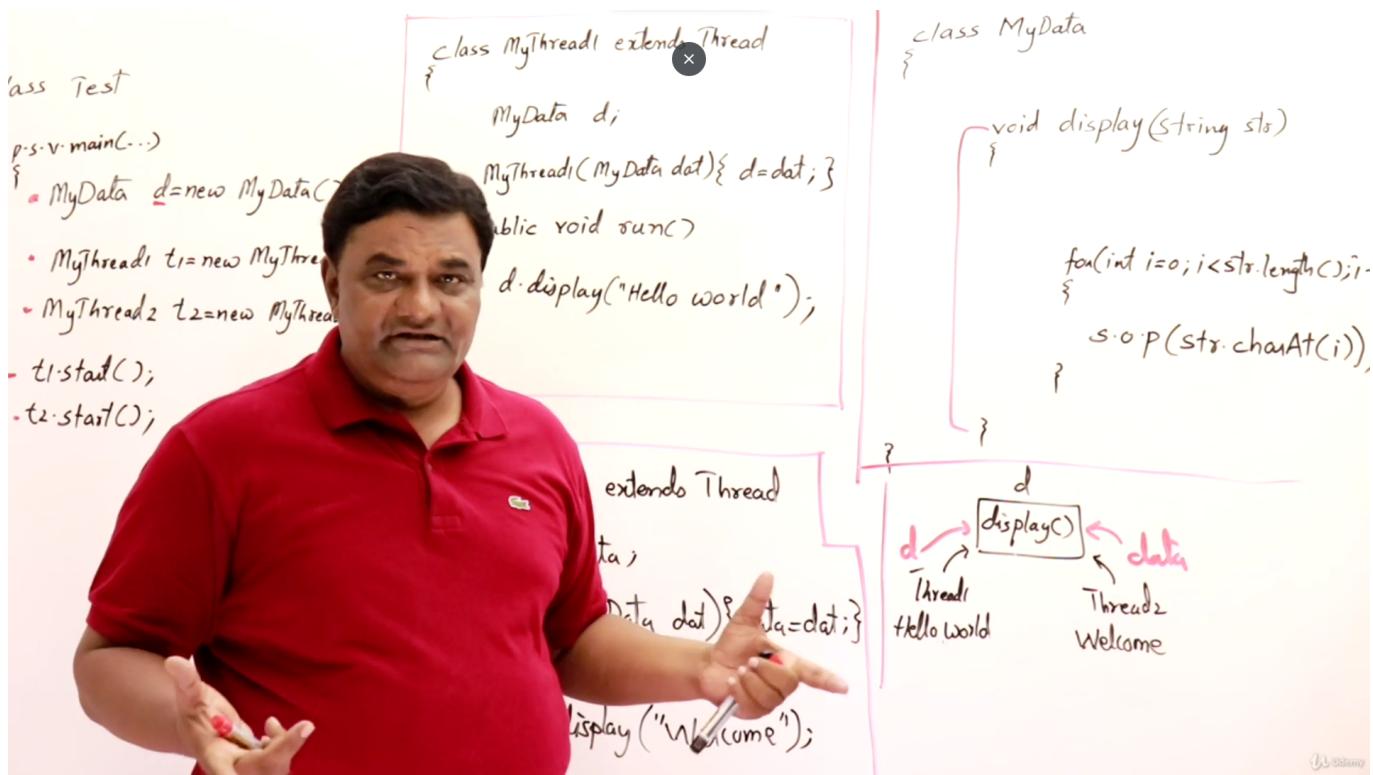
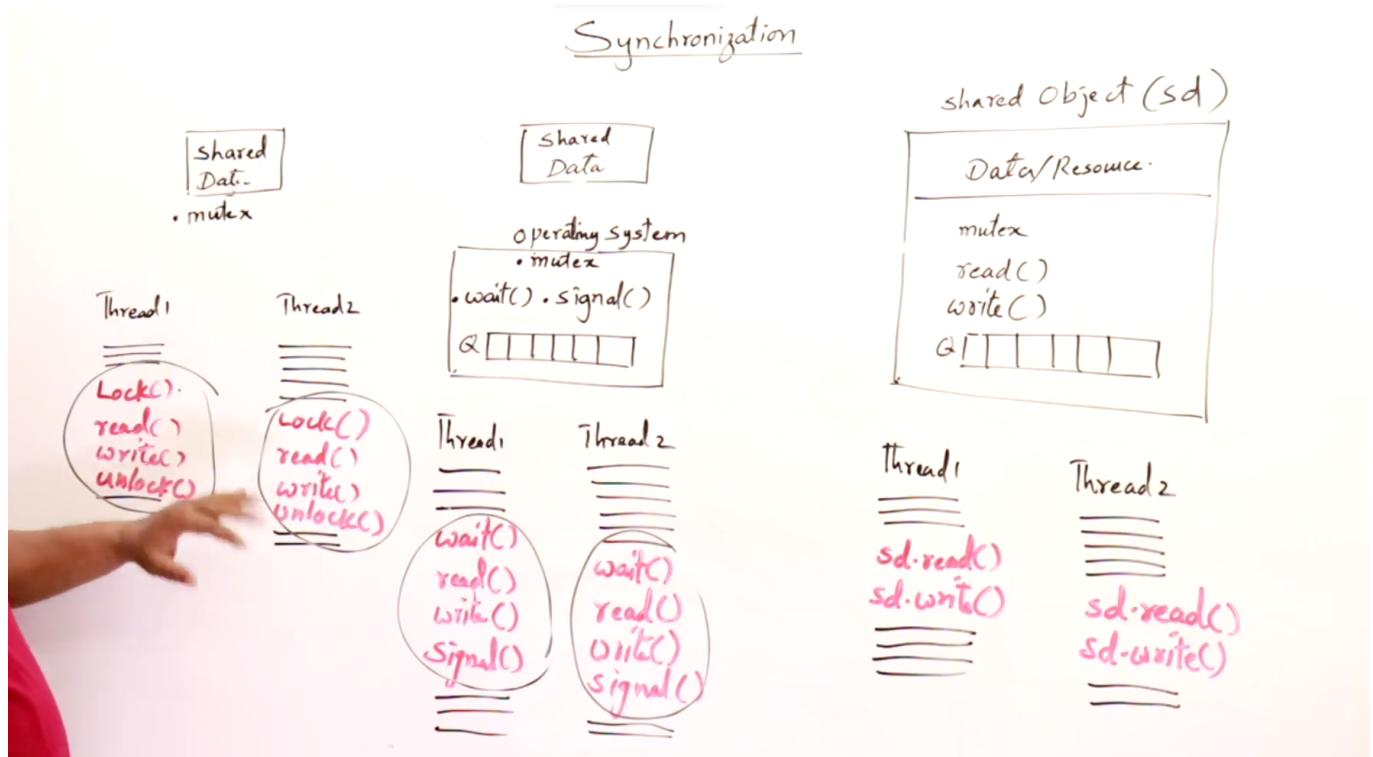
semaphore maintains the separate queue





### Synchronization





```
void display(String str)
{
    synchronized(this)
    {
        for(int i=0; i<str.length(); i++)
        {
            System.out.print(str.charAt(i));
        }
    }
}
```

class MyData

{

    synchronized void display(String str)

    {

        for(int i=0; i<str.length(); i++)

        {

            System.out.print(str.charAt(i));

        }

    }

}

    d

```
package syncdemo;

class MyData
{
    synchronized public void display(String str)
    {
        for(int i=0;i<str.length();i++)
        {
            System.out.print(str.charAt(i));
            try{Thread.sleep(100);}catch(Exception e){}
        }
    }

    class MyThread1 extends Thread
    {
        MyData d;
```

```
public MyThread1(MyData d)
{
    this.d=d;
}

public void run()
{
    d.display("Hello World");
}

}

class MyThread2 extends Thread
{
    MyData d;
    public MyThread2(MyData d)
    {
        this.d=d;
    }

    public void run()
    {
        d.display("Welcome All");
    }
}

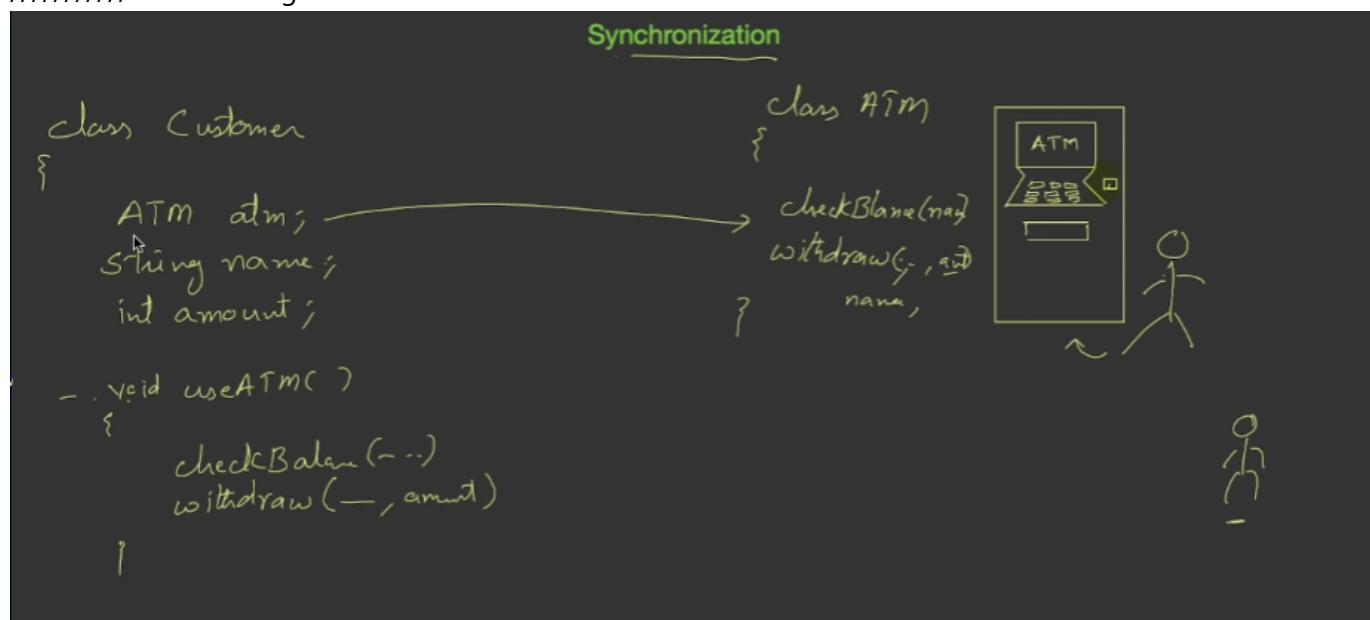
}

public class SyncDemo
{
    public static void main(String[] args)
    {
        MyData data=new MyData();

        MyThread1 t1=new MyThread1(data);
        MyThread2 t2=new MyThread2(data);

        t1.start();
        t2.start();
    }
}
```

## //////////ATM challenge



```

package scthread1;

class ATM
{

    synchronized public void checkBalance(String name)
    {
        System.out.print(name + " Checking ");

        try{Thread.sleep(1000);}catch(Exception e){}

        System.out.println("Balance");
    }

    synchronized public void withdraw(String name,int amount)
    {
        System.out.print(name + " withdrawing ");

        try{Thread.sleep(1000);}catch(Exception e){}

        System.out.println(amount);
    }
}

class Customer extends Thread
{
    String name;
    int amount;
    ATM atm;

    Customer(String n,ATM a,int amt)
    {

```

```

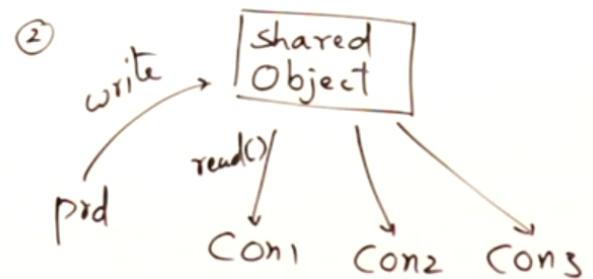
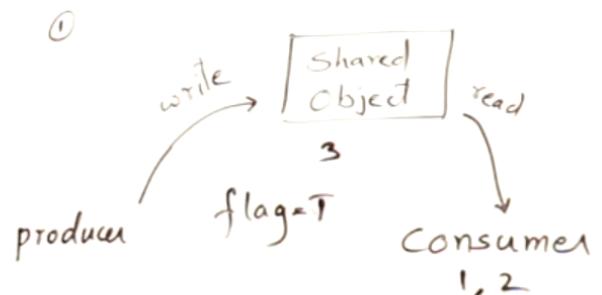
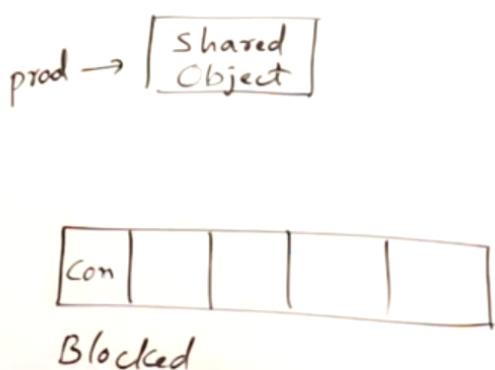
        name=n;
        atm=a;
        amount=amt;
    }
    public void useATM()
    {
        atm.checkBalance(name);
        atm.withdraw(name, amount);
    }
    public void run()
    {
        useATM();
    }
}

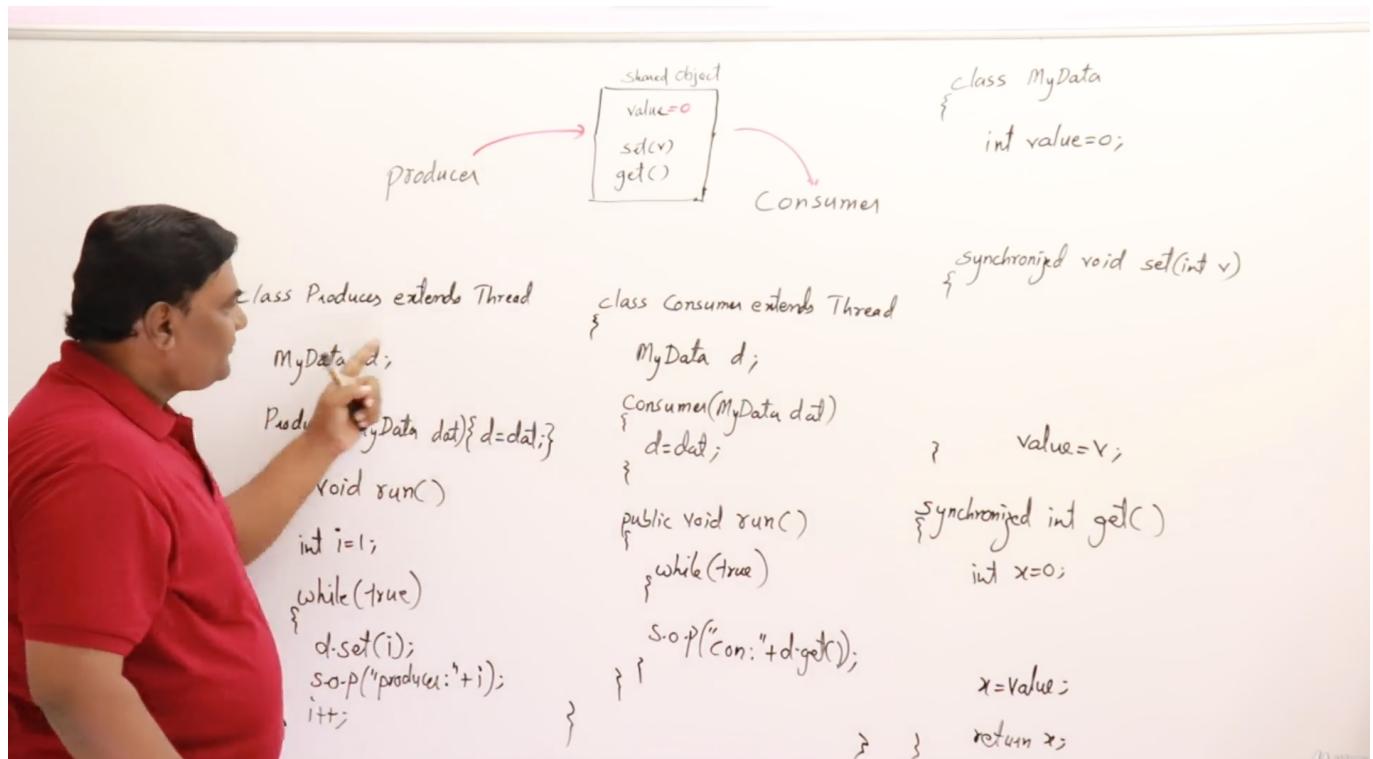
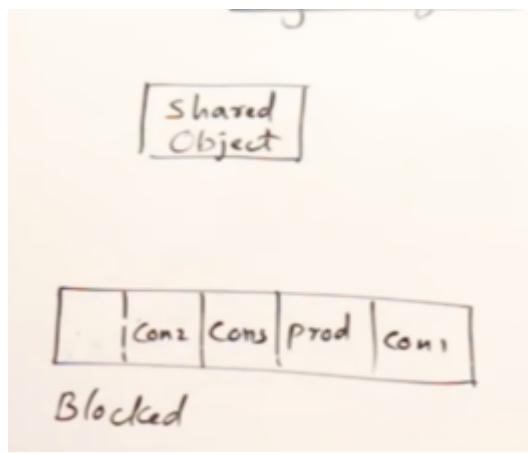
public class SCThread1
{
    public static void main(String[] args)
    {
        ATM atm=new ATM();
        Customer c1=new Customer("Smith",atm,100);
        Customer c2=new Customer("John",atm,200);
        c1.start();
        c2.start();

    }
}

```

## Synchronization





```
class MyData
{
    int value=0;
    boolean flag=true;

    synchronized void set(int v)
    {
        while(flag!=true)
            wait();
        value=v;
    }

    synchronized int get()
    {
        int x=0;
    }
}
```

```
class MyData  
{  
    int value=0;  
    boolean flag=true;
```

```
{ synchronized void set(int v)  
{  
    while(flag!=true)  
        wait();  
    value=v;  
    flag=false;  
    notify();  
}
```

```
synchronized int get()  
{
```

```
    int x=0;  
    while(flag!=false) wait();  
    x=value;  
    flag=true;  
    notify();  
    return x;
```

```
package interprocess;

class MyData
{
    int value;
    boolean flag=true;

    synchronized public void set(int v)
    {
        while(flag!=true)
            try {wait();}catch(Exception e){}

        value=v;
        flag=false;
        notify();
    }

    synchronized public int get()
    {
        int x=0;
        while(flag!=false)
            try {wait();}catch(Exception e){}

        x=value;
        flag=true;
        notify();

        return x;
    }
}

class Producer extends Thread
{
    MyData data;

    public Producer(MyData d)
    {
        data=d;
    }
    public void run()
    {
        int count=1;
        while(true)
        {
            data.set(count);
            System.out.println("Producer "+count);
            count++;
        }
    }
}
```

```
class Consumer extends Thread
{
    MyData data;

    public Consumer(MyData d)
    {
        data=d;
    }
    public void run()
    {
        int value;
        while(true)
        {
            value=data.get();
            System.out.println("Consumer "+value);
        }
    }
}

public class InterProcess
{
    public static void main(String[] args)
    {
        MyData data=new MyData();

        Producer p=new Producer(data);
        Consumer c=new Consumer(data);

        p.start();
        c.start();
    }
}
```

```
package scthread2;

class WhiteBoard
{
    String text;
    int numberOfStudents=0;
    int count=0;
    public void attendance()
    {
        numberOfStudents++;
    }

    synchronized public void write(String t)
    {
        System.out.println("Teacher is Writing " +t);
    }
}
```

```
        while(count!=0)
            try{wait();}catch(Exception e){}
        text=t;
        count=numberOfStudents;
        notifyAll();

    }
synchronized public String read()
{
    while(count==0)
        try{wait();}catch(Exception e){}

    String t=text;
    count--;
    if(count==0)
        notify();
    return t;
}

}
class Teacher extends Thread
{
    WhiteBoard wb;

    String notes[]={ "Java is language", "It is OOPs", "It is Platform Independent", "It supports Thread", "end" };

    public Teacher(WhiteBoard w)
    {
        wb=w;
    }

    public void run()
    {
        for(int i=0;i<notes.length;i++)
            wb.write(notes[i]);
    }
}

class Student extends Thread
{
    String name;
    WhiteBoard wb;
    public Student(String n,WhiteBoard w)
    {
        name=n;
        wb=w;
    }

    public void run()
    {
        String text;
```

```
wb.attendance();

do
{
    text=wb.read();
    System.out.println(name + " Reading " + text);
    System.out.flush();
}while(!text.equals("end"));
}

}

public class SCThread2
{
    public static void main(String[] args)
    {
        WhiteBoard wb=new WhiteBoard();
        Teacher t=new Teacher(wb);

        Student s1=new Student("1. John",wb);
        Student s2=new Student("2. Ajay",wb);
        Student s3=new Student("3. Kloob",wb);
        Student s4=new Student("4. Smith",wb);

        t.start();

        s1.start();
        s2.start();
        s3.start();
        s4.start();

    }
}
```

## Section 20 java.lang.Object

---

Object class is the Parent class of all the classes

```
public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();
        Object o2=o1;

        System.out.println(o1.equals(o2));
    }
}
```

Output - LangDemo (run)

```
Compiling 1 source file to /Users/abdulbari/Ne
compile:
run:
true
```

<https://docs.oracle.com/javase>

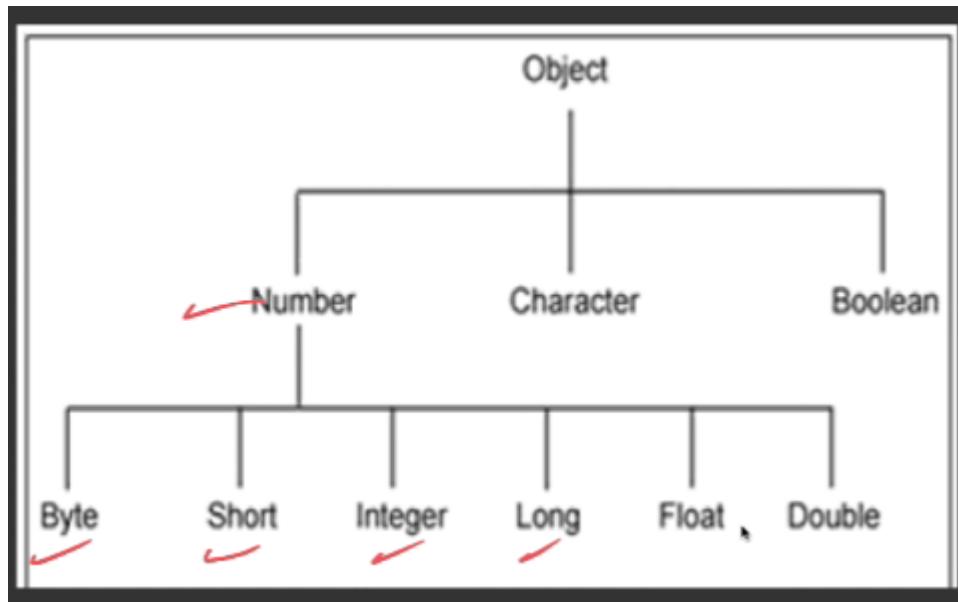
```
import java.lang.*;  
  
class MyObject  
{  
    public String toString()  
    {  
        return "My Object";  
    }  
  
    public int hashCode()  
    {  
        return 100;  
    }  
}  
  
public class LangDemo  
{  
    public static void main(String[] args)  
    {  
        MyObject o1=new MyObject();  
  
        System.out.println(o1);  
    }  
}  
  
public int hashCode()  
{  
    return 100;  
}  
public boolean equals(Object o)  
{  
    return this.hashCode()==o.hashCode();  
}  
  
public class LangDemo  
{  
    public static void main(String[] args)  
    {  
        MyObject o1=new MyObject();  
        MyObject o2=new MyObject();  
  
        System.out.println(o1.equals(o2));  
    }  
}
```

.toString() .equals(Object o) .hashCode()

## 199. Wrapper Classes

# Wrapper Classes

- ✓ Character
- ✓ Byte
- ✓ Short



```
package wrapperdemo2;

public class WrapperDemo2 {

    public static void main(String[] args) {

        Integer i=new Integer(10);
        Integer a=Integer.valueOf(10);
        Integer b=10;

        Byte c=15;
        Byte d[Byte.valueOf("15")];

        Short f=Short.valueOf("123");

        Float g=12.3f;

        Double j=Double.valueOf(123.456);

        Character k=Character.valueOf('A');

    }
}
```

```
Boolean l=Boolean.valueOf("true");

byte bb=15;
Byte e=Byte.valueOf(bb);

Float h=Float.valueOf("123.5");
float x=h.floatValue();
float y=h;

int m=10;
//Integer n=Integer.valueOf(m);
Integer n=m;
//int p=n.intValue();
int p=n;
}

}
```

to get inbuilt methods of the wrapper classes

```
int m1 = 10; Integer m2 = m1; Integer m3 = m2; System.out.println(m2.equals(m1));//true //compares content
Integer m4 = Integer.valueOf("1010",2); //binary and
```

```
//INTEGER CLASS
package wrapperdemo;

public class WrapperDemo {

    public static void main(String[] args) {

        int m1=15;

        //Integer m2=m1;
        //Integer m3=15;

        Integer m2=Integer.valueOf("123");
        //Integer m3=Integer.valueOf("11111111", 2); //255
        Integer m3=Integer.valueOf("A7", 16); //167 //16*10+7=167
        Integer m4=Integer.decode("0xA7"); //true //hexa to decimal

        //System.out.println(m2.equals(m1));
        //System.out.println(m2.equals(m3));
        //System.out.println(m3);

        //System.out.println(Integer.parseInt("123")); //converts into a number.
        //System.out.println(Integer.parseInt("123a")); //NumberFormatException

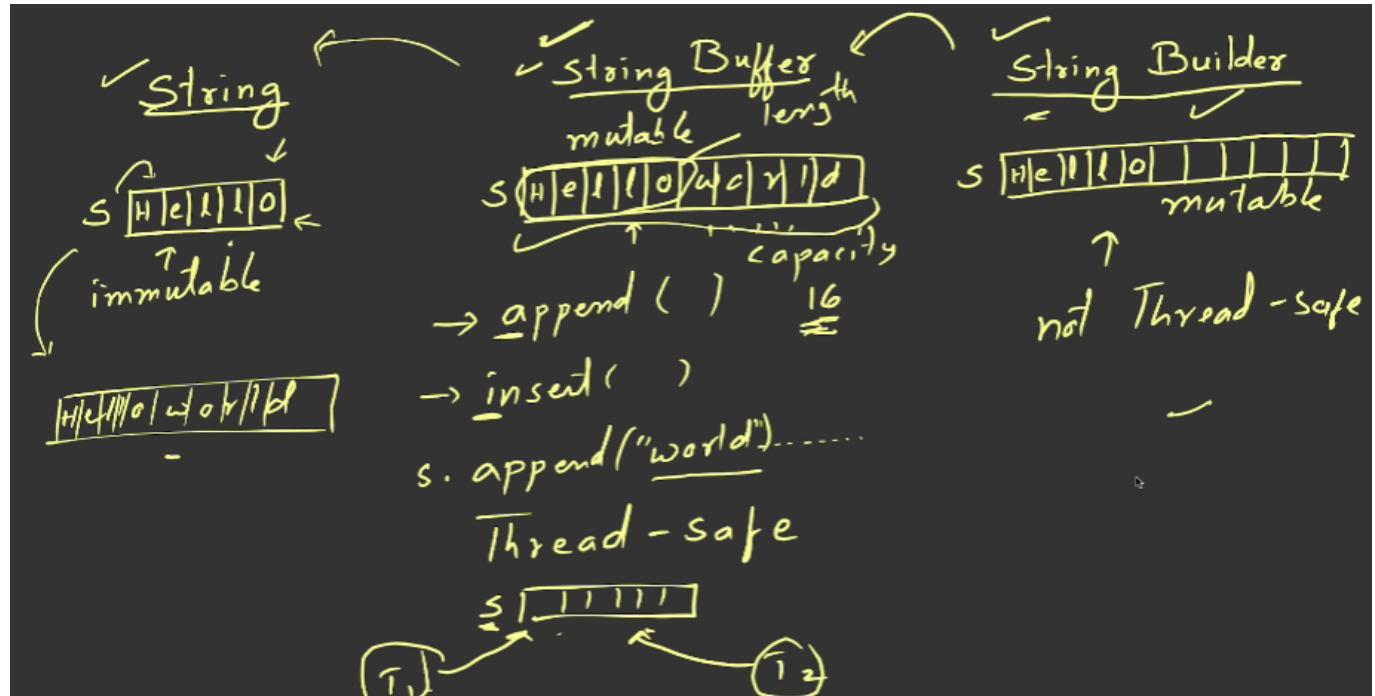
        //System.out.println(Integer.reverseBytes(128)==Integer.MIN_VALUE);
        System.out.println(Integer.reverseBytes(128));
    }
}
```

```

        System.out.println(Integer.toBinaryString(40));
    }
}

```

//String string in java are immutable StringBuffer is thread safe StringBuilder is not a thread safe present in the lang package



```

package stringbufferbuilder;

public class StringBufferBuilder
{
    public static void main(String[] args)
    {
        String s1=new String("Hello");

        StringBuffer s2=new StringBuffer("Hello");

        StringBuilder s3=new StringBuilder("Hello");

        s1.concat(" World");
        s2.append(" World");
        s3.append(" World");

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);

    }
}

```

```
//MATH Class .abs() //gives the positive value  
StrictMath.abs(-123); //at runtime it will be checked and it will be executed.  
  
Math.sqrt() Math.cbrt() Math.decrementExact() //prevents the value of underflow Math.getExponent(123.45);  
//6 //you Math.floorDiv(50,9)  
  
Math.exp(1) StrictMath.exp(1)
```

```
package mathdemo;  
  
public class MathDemo  
{  
    public static void main(String[] args)  
    {  
        System.out.print("Absolute :");  
        System.out.println(Math.abs(-123));  
  
        System.out.print("Absolute :");  
        System.out.println(StrictMath.abs(-123));  
  
        System.out.print("Cube Root :");  
        System.out.println(Math.cbrt(27));  
  
        System.out.print("Exact Decrement :");  
        // System.out.println(Math.decrementExact(Integer.MIN_VALUE));  
  
        int i=Integer.MIN_VALUE;  
        i--;  
        System.out.println(i);  
  
        System.out.print("Exponent value in Floating Point Rep. :");  
        System.out.println(Math.getExponent(123.45));  
  
        System.out.print("Round Division :");  
        System.out.println(Math.floorDiv(50, 9));  
  
        System.out.print("e power x :");  
        System.out.println(Math.exp(1));
```

```
System.out.print("e power x :");
System.out.println(Math.exp(1));

System.out.print("Log base 10 :");
System.out.println(Math.log10(100));

System.out.print("Maximum :");
System.out.println(Math.max(100, 50));

System.out.print("Tan :");
System.out.println(Math.tan(45*Math.PI/180));

System.out.print("Convert to Radians :");
System.out.println(Math.toRadians(90));

System.out.print("Convert to Degree :");
System.out.println(Math.toDegrees(Math.atan(1)));

System.out.print("Convert To Degree :");
System.out.println(Math.toDegrees(Math.tanh(1)));

System.out.print("Random :");
System.out.println(Math.random()*1000);

System.out.print("Power :");
System.out.println(Math.pow(2, 3));

System.out.print("Exact Product :");
System.out.println(Math.multiplyExact(100, 200));

System.out.print("Next Float Value :");
System.out.println(Math.nextAfter(12.5, 11)); //if greater number then
higher value or if lower number then lower value

}

}
```

## enum

inside enum identifier is the constant public static final variables

```
enum Dept
{
    CS,IT,CIVIL,ECE
}

public class EnumDemo
{
    public static void main(String[] args)
    {
        Dept d=Dept.CIVIL;

        System.out.println(Dept.valueOf("IT"));
    }
}
```

output is : IT

```
enum Dept
{
    CS,IT,CIVIL,ECE
}

public class EnumDemo
{
    public static void main(String[] args)
    {
        Dept d=Dept.CIVIL;

        Dept list[]=Dept.values();

        for(Dept x:list)
            System.out.println(x);

    }
}
```

list will be printed

operations

```
enum Dept
{
    CS, IT, CIVIL, ECE
}

public class EnumDemo
{
    public static void main(String[] args)
    {
        Dept d=Dept.CIVIL;

        switch(d)
        {
            case CS: System.out.println("Head: John \nBlock: A");
            break;
            case IT: System.out.println("Head: Smith \nBlock: B");
            break;
            case CIVIL: System.out.println("Head: Srinivas \nBlock: C");
            break;
            case ECE: System.out.println("Head: Dave \nBlock: D");
            break;
        }
    }
}
```

enum can have other methods and constructor.

```
package enumdemo;

enum Dept
{
    CS("John", "Block A"), IT("Smith", "Block B"), CIVIL("Srinivas", "Block C"),
    ECE("Dave", "Block D");

    String head;
    String location;

    private Dept(String head, String loc)
    {
        this.head=head;
        this.location=loc;
    }
    public String getHeadName()
    {
        return head;
    }
    public String getLocation()
    {
        return location;
    }
}

public class EnumDemo
```

```
{  
    public static void main(String[] args)  
    {  
        Dept d=Dept.CS; //when enum is loaded then constructor is automatically  
        called and object is created.  
  
        System.out.println(d.getHeadName());  
        System.out.println(d.getLocation());  
  
    }  
}
```

```
package reflectdemo;  
  
import java.lang.reflect.*;  
  
class My  
{  
    private int a;  
    protected int b;  
    public int c;  
    int d;  
  
    public My() {}  
  
    public My(int x,int y) {}  
  
    public void display(String s1,String s2) {}  
    public int show(int x,int y) {return 0;}  
  
}  
  
public class ReflectDemo  
{  
    public static void main(String[] args)  
    {  
        Class c=My.class;  
  
        Field field[] = c.getDeclaredFields();  
  
        Method meth[] = c.getMethods();  
  
        for(Method m:meth)  
            System.out.println(m);  
  
        Parameter param[] = meth[0].getParameters();  
  
        for(Parameter p:param)  
        {
```

```
        System.out.println(p);
    }

}
}
```

enum is loaded then constructor is automatically called

```
enum Dept
{
    CS,IT,CIVIL,ECE;

    private Dept()
    {
        System.out.println(this.name());
    }
}
```

```
enum Dept
{
    CS,IT,CIVIL,ECE;

    private Dept()
    {
        System.out.println(this.name());
    }
    public void display()
    {
        System.out.println(this.name() + " " + this.ordinal());
    }
}
```

```
public class EnumDemo
{
    public static void main(String[] args)
    {
        Dept d=Dept.CS;
        d.display();
    }
}
```

## 205 Reflection Package

```
java.lang.reflect.*;
```

let's see how definition of class can be called

```
import java.lang.reflect.*;

class My
{
    private int a;
    protected int b;
    public int c;
    int d;

    public My() {}

    public My(int x,int y) {}

    public void display(String s1,String s2) {}
    public int show(int x,int y) {return 0;}
}

public class ReflectDemo
{
    public static void main(String[] args)
    {
        }
}
```

Class c = My.class; //gives the definition another way

```
public class ReflectDemo
{
    public static void main(String[] args)
    {
        Class c=My.class;

        Field field[]=c.getDeclaredFields();

        for(Field f:field)
            System.out.println(f);
    }
}
```

to see declared members or fields

```
Constructor con[]=c.getConstructors();

for(Constructor ct:con)
    System.out.println(ct);
```

to see constructors

```
Method meth[] = c.getMethods();

for(Method m:meth)
    System.out.println(m);
```

to see methods

to see arguments of the specific method

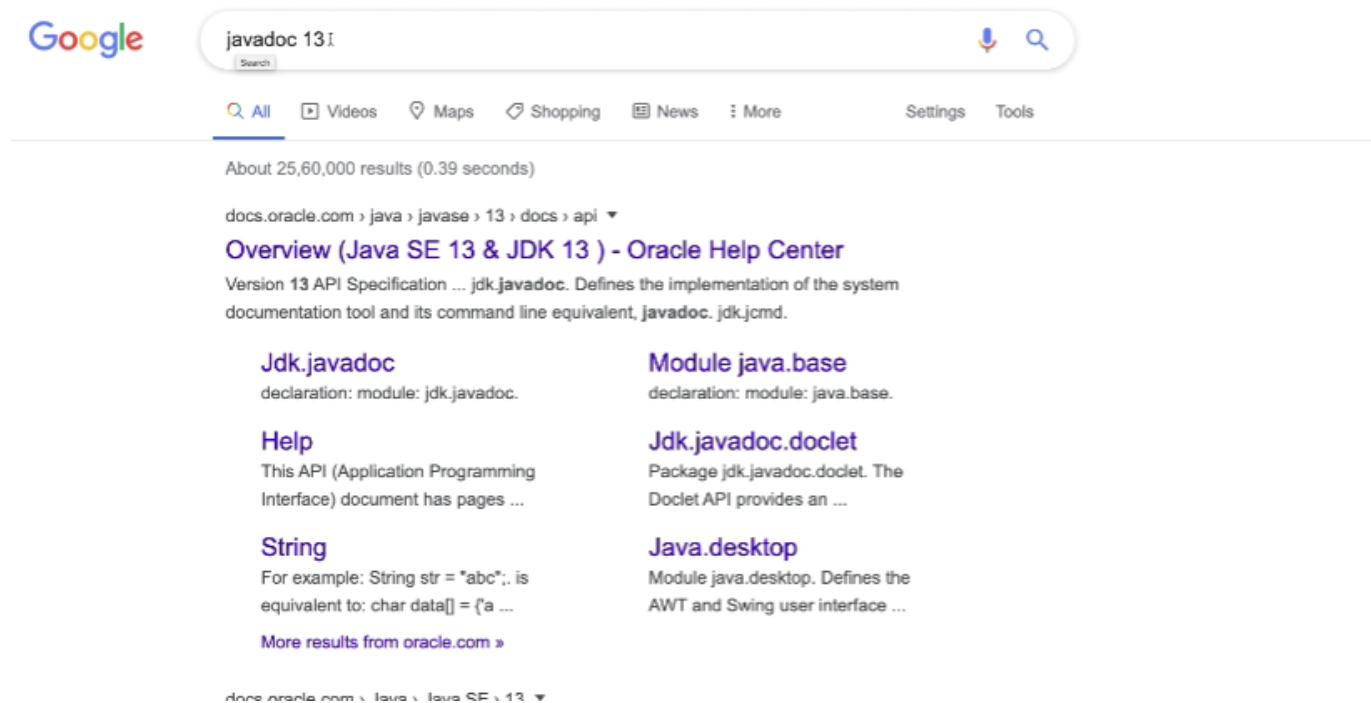
```
Parameter param[] = meth[0].getParameters();

for(Parameter p:param)
{
    System.out.println(p);
}
```

## Section 21: Annotations and JAVA Docs

---

search > javadoc 13



The screenshot shows a Google search results page for the query "javadoc 13". The search bar contains "javadoc 13". Below the search bar, the "All" tab is selected, along with other categories like Videos, Maps, Shopping, News, More, Settings, and Tools. The search results indicate approximately 25,60,000 results found in 0.39 seconds. The top result is a link to the "Overview (Java SE 13 & JDK 13) - Oracle Help Center" on docs.oracle.com. The page content includes sections for "Jdk.javadoc", "Help", "String", and "Module java.base". Each section provides a brief description and a link to "More results from oracle.com". The footer of the page shows navigation links for "docs.oracle.com", "java", "Java SE", and "13".

java.base packages..... interfaces classes : String, fields , constructor, methods, package and modules.

there is tool called javadoc to make documentation.

```
package javadocdemo;

public class Book
{
    static int val=10;

    public Book(String s){
    }

    public void issue(int roll) throws Exception{
    }

    public boolean available(String str){
        return true;
    }

    public String getName(int id){
        return "";
    }
}
```

## JavaDoc Tags

| <u>Class</u> | Method      | Others  |
|--------------|-------------|---------|
| @author      | @param      | @link   |
| @version     | @return     | @value  |
| @since       | @throws     | @serial |
| @see         | @exception  |         |
|              | @deprecated |         |
|              | @code       |         |

```
/** @author Abdul Bari
 *  @version 2.0
 *  @since 2015
 */
package javadocdemo;

/***
 *
 * @author abdulbari
 *
```

```
* Class for Library Book
*/



public class Book
{

    /**
     * @value 10 default value
     */
    static int val=10;

    /**
     * Parametrized Constructor
     * @param s Book Name
     */
    public Book(String s){

    }

    /**
     * Issue a Book to a Student
     * @param roll roll number of a Student
     * @throws Exception if book is not available, throws Exception
     */
    public void issue(int roll) throws Exception{

    }

    /**
     * Check if book is available
     * @param str Book Name
     * @return if book is available returns true else false
     */
    public boolean available(String str){
        return true;
    }

    /**
     * Get Book name
     * @param id book id
     * @return returns book name
     */
    public String getName(int id){
        return "";
    }
}
```

Run > Generate Javadoc (JavaDocDemo) > Next > Finish documentation will be created in html file

project dir > javadocs > dist > index.html

## Annotations

### 1. Applied to Code

### 2. Applied to Other Annotations

Annotations are used for giving attributes or defining the attributes of the class or method or interface. so annotations are useful for giving metadata to the class or method or interface.

## Annotations

### in-built Annotations

#### 1. @Override

#### 2. @Deprecated

#### 3. @FunctionalInterface

#### 4. @SuppressWarnings

```
@Deprecated
public void show()
{
    System.out.println("Hi");
}

public class AnnoDemo
{
    public static void main(String[] args)
    {
        OldClass oc=new OldClass();
        oc.show();
    }
}
```

output: deprecation

warning

@SuppressWarning("deprecation") upon method then it won't show's the warnings. attribute are deprecation, unchecked.

@SuppressWarning("unchecked") upon method then it won't show's the warnings.

```
public class AnnoDemo
{
    static List l;

    @SuppressWarnings("deprecation")
    public static void main(String[] args)
    {
        l.add(10);

        /* OldClass oc=new OldClass();
         * oc.show();*/
    }
}
```

method must be private or final for the annotation. safeVarargs

```
package annodemo;

class My<T>
{
    @SafeVarargs
    private void show(T...arg)
    {
        for(T x:arg)
            System.out.println(x);
    }
}

public class AnnoDemo
{
    public static void main(String[] args)
    {
    }
}
```

```
package annotationdemo;

class A
```

```
{  
    public void display()  
    {  
  
    }  
}  
  
class B extends A  
{  
    @Override  
    public void display()  
    {  
  
    }  
    @Deprecated  
    public void show()  
    {  
  
    }  
  
}  
  
public class AnnotationDemo  
{  
  
    public static void main(String[] args)  
    {  
        int i;  
        @SuppressWarnings("deprecation")  
  
        B b=new B();  
  
        b.show();  
    }  
  
}
```

Functional Interface = interface having only one abstract method is known as the functional interface.  
functional interfaces are used to write the lambda expressions.

dummy or empty annotation for parameters also we can give the annotations and for the local variables also we can give the annotations.

```
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    int data;

    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        @MyAnno
        int x;

    }
}
```

```
import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    int data;

    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        @MyAnno
        int x;

    }
}
```

They are called as elements of annotation.

DEFINING THE ANNOTATION when annotation has field then you must give the value to the field.

```
import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
    String project();
    String date();
    String version();
}

@MyAnno(name="Ajay")
public class AnnoDemo
{
    @MyAnno(name="Ajay")
    int data;

    @MyAnno(name="Ajay")
    public static void main(@MyAnno(name="Ajay") String[] args)
    {
        @MyAnno(name="Ajay")
```

So this is metadata.

above shows data or information about the class.

```
package annodemo;

//import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name(); //can't use throws Exception or parameters.
    String project();
    String date() default "today";
    String version() default "13";
}

@MyAnno(name="Ajay",project="Bank")
public class AnnoDemo
{
    //@MyAnno(name="Ajay")
    int data;

    //@MyAnno(name="Ajay")
    public static void main(String[] args)
    {
        //@MyAnno(name="Ajay")
        int x;
    }
}
```

```
}
```

```
import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
    String project();
    String date();
    String version();
}

@MyAnno(name="Ajay",project="Bank",date="1/1/2000",version="13")
public class AnnoDemo
{
    int data;

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

## in-built Annotations

1. Retention
2. Documented
3. Target
4. Inherited
5. Repeatable

@Retention(RetentionPolicy.RUNTIME) //class means available within the class //by using the reflection you can get the data.

@Documented. //provides the documentation.

```
package annodemo;

import java.lang.annotation.*;

@Target(ElementType.LOCAL_VARIABLE)
@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";
}

public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;
    }
}
```

```
@Target(ElementType.LOCAL_VARIABLE)
@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";
}

@MyAnno(name="Ajay",project="Bank")
public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;
    }
}
```

works for the local variable

```
@Target(ElementType.LOCAL_VARIABLE)
@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";
}

public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        @MyAnno(name="Ajay",project="Bank")
        int x;
    }
}
```

now available at the method level

```
@Target(value=[ElementType.LOCAL_VARIABLE, ElementType.METHOD])  
  
@interface MyAnno  
{  
    String name();  
    String project();  
    String date() default "today";  
    String version() default "13";  
}  
  
public class AnnoDemo  
{  
    int data;  
    @MyAnno(name="Ajay", project="Bank")  
    public static void main(String[] args)  
    {  
        int x;  
    }  
}
```

annotation can be used only once if Repeatable is used then multiple times can be used.

```

@Target(value={ElementType.LOCAL_VARIABLE, ElementType.METHOD})
@Repeatable(MyAnno.class)
@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";
}

public class AnnoDemo
{
    int data;
    @MyAnno(name="Ajay",project="Bank")

    public static void main(String[] args)
    {

        int x;
    }
}

```

has

```

@Target(value={ElementType.LOCAL_VARIABLE, ElementType.METHOD})
@Repeatable(MyAnno.class)
public interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";
}

public class AnnoDemo
{
    int data;
    @MyAnno(name="Ajay",project="Bank")

    public static void main(String[] args)
    {

        int x;
    }
}

```

to be public

## Section 22 : Lambda Expressions

### 210 Intro to lambda expression

=lambda expressions are used to define the anonymous functions or nameless methods. lambda expressions are written with help of functional interfaces. lambda expressions are defined using the -> operator.

//= if you have only one abstract method then it is called as the functional interface.

```
@FunctionalInterface  
interface MyLambda  
{  
    public void display();  
}  
  
class My implements MyLambda  
{  
    public void display()  
    {  
        System.out.println("Hello World");  
    }  
}  
  
public class LamdaDemo  
{  
    public static void main(String[] args)  
    {  
        My m=new My();  
        m.display();  
    }  
}
```

instead of another class, we can write the

anonymous inner class

```
@FunctionalInterface  
interface MyLambda  
{  
    public void display();  
}  
  
public class LamdaDemo  
{  
    public static void main(String[] args)  
    {  
        MyLambda m=new MyLambda(){  
            public void display()  
            {  
                System.out.println("Hello World");  
            }  
        };  
  
        m.display();  
    }  
}
```

But I'm writing it anonymous inner class.

below is known as the lambda expression

```
@FunctionalInterface  
interface MyLambda  
{  
    public void display();  
}  
  
public class LamdaDemo  
{  
    public static void main(String[] args)  
    {  
        MyLambda m=  
            ()->  
            {  
                System.out.println("Hello World")  
            }  
        m.display();  
    }  
}
```

=it is anonymous method which is acting as the object. this is the defination of the display method.

```
MyLambda m=()>{System.out.println("Hello World");};
```

to make a functional interface easy for implementation, this lambda expressions are very useful, very handy and very easy to write.

lambda expressions makes the programmers like easy, when override the functional interface.

```
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display(String str);
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=(s)->{System.out.println(s);}

        m.display("Hello World");
    }
}
```

passing the arguments

```
@FunctionalInterface
interface MyLambda
{
    public int add(int x,int y);
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=(a,b)->{return a+b;};

        System.out.println(m.add(20,30));
    }
}
```

adding two nums

```
@FunctionalInterface  
interface MyLambda  
{  
    public int add(int x,int y);  
}  
  
public class LamdaDemo  
{  
    public static void main(String[] args)  
    {  
        MyLambda m=(a,b)->a+b;  
        |  
        int r=m.add(20,30);  
        System.out.println(r);  
    }  
}
```

capture in lamdba expression

```
interface MyLambda  
{  
    public void display();  
}  
  
class Demo  
{  
  
    public void method1()  
    {  
  
        MyLambda ml=()->System.out.println("Hi");  
    }  
}  
  
public class LamdaDemo  
{  
    public static void main(String[] args)  
    {  
        Demo d=new Demo();  
        d.method1();  
    }  
}
```

```
package lamdademo2;  
  
interface MyLambda  
{  
    public void display();  
}  
  
class UseLambda  
{
```

```
public void callLambda(MyLambda ml)
{
    ml.display();
}
}

class Demo
{
    public void method1(){
        UseLambda ul=new UseLambda();
        ul.callLambda(()->{System.out.println("Hello");});

        /*int temp=10;

        public void method1()
        {
            *int count=0;//inner class and lambda can't access the local variable

            MyLambda ml=()->
            {
                //int count=0;
                //count++;
                int x=0;
                System.out.println("Hi");
                System.out.println("Bye"+(++temp));
            };
        }

        ml.display();*/
    }
}

public class LamdaDemo2 {

    public static void main(String[] args) {
        Demo d=new Demo();
        d.method1();
    }
}
```

lambda expressions can be passed as the method as the object.

```
interface MyLambda
{
    public void display();
}

class UseLambda
{
    public void callLambda(MyLambda ml)
    {
        ml.display();
    }
}
```

we can pass lambda expression as the

parameter and the object.

```
class Demo
{

    public void method1()
    {
        UseLambda ul=new UseLambda();
        ul.callLambda(()->{System.out.println("Hello");});
    }
}
```

## Method Reference

```
package lamdademo;

interface MyLambda
{
    public void display(String str);
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda ml=System.out::println;
    }
}
```

scope resolution operator :: println is the static method.

```
MyLambda ml=System.out::println;
ml.display("Hello");
```

```
interface MyLambda
{
    public void display(String str);
}

public class LamdaDemo
{
    public static void reverse(String str)
    {
        StringBuffer sb=new StringBuffer(str);
        sb.reverse();
        System.out.println(sb);
    }

    public static void main(String[] args)
    {
        MyLambda ml=LamdaDemo::reverse;

        ml.display("Hello");
    }
}
```

```
public class LamdaDemo
{
    public void reverse(String str)
    {
        StringBuffer sb=new StringBuffer(str);
        sb.reverse();
        System.out.println(sb);
    }

    public static void main(String[] args)
    {
        LamdaDemo ld=new LamdaDemo();

        MyLambda ml=ld::reverse;

        ml.display("Hello");
    }
}
```

for non static method

can you assign the constructor as the method reference

```
interface MyLambda
{
    public void display(String str);
}

public class LamdaDemo
{
    public LamdaDemo(String s)
    {
        System.out.println(s.toUpperCase());
    }

    public static void main(String[] args)
    {

        MyLambda ml=LamdaDemo::new;

        ml.display("Hello");
    }
}
```

//HELLO

```
interface MyLambda
{
    public int display(String str1, String str2);
}

public class LamdaDemo
{
    public LamdaDemo(String s)
    {
        System.out.println(s.toUpperCase());
    }

    public static void main(String[] args)
    {

        MyLambda ml=String::compareTo;

        System.out.println(ml.display("hello", "hello"));
    }
}
```

returns the integer

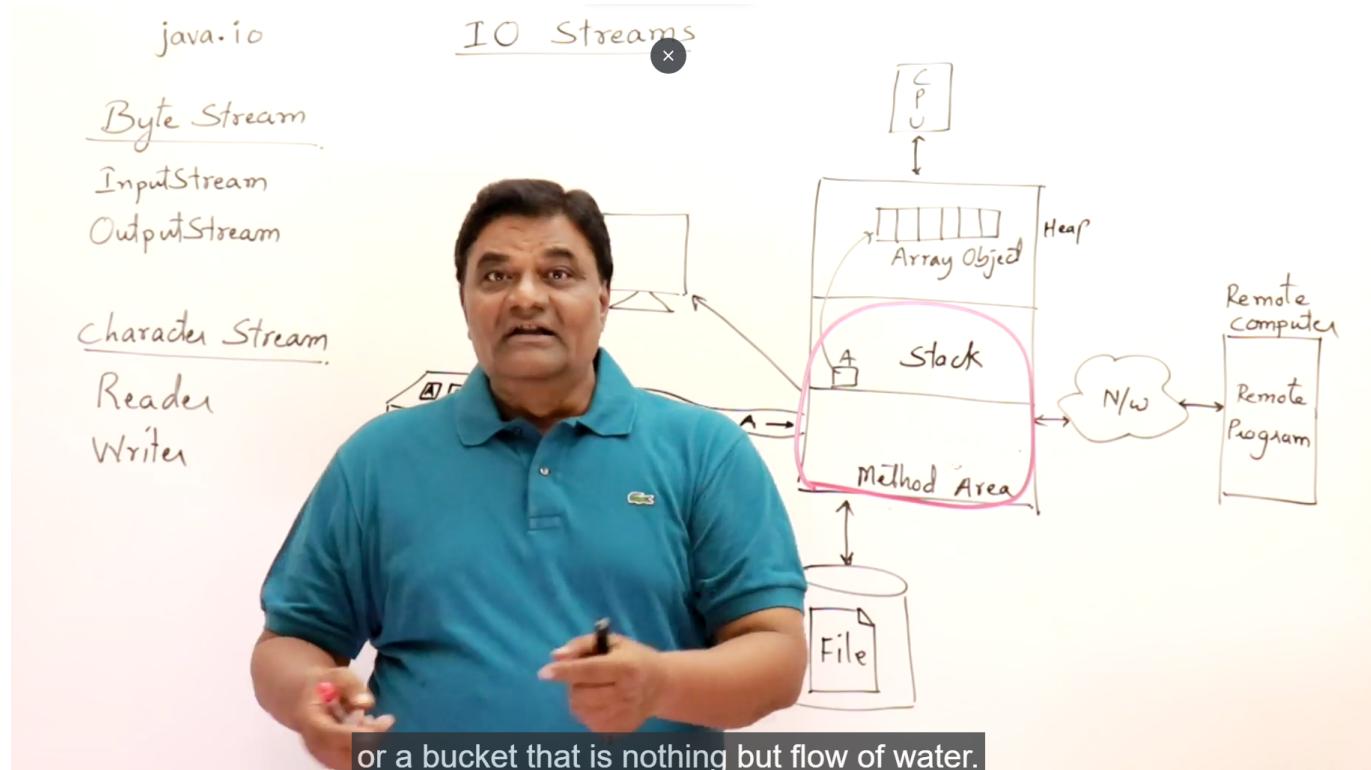
it's like the polymorphism, having the parent type reference and calling the child class object

## Section 23 : JAVA IO streams.

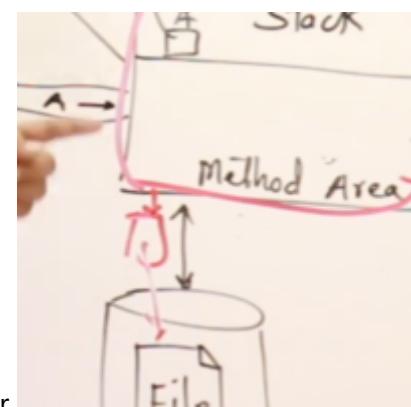
## What is streams ?

stream is flow of data.

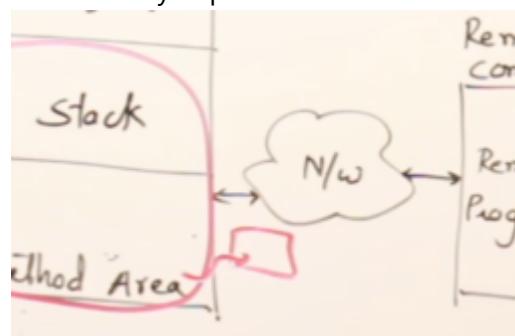
data may be flowing from program to a resource or resource to the program.

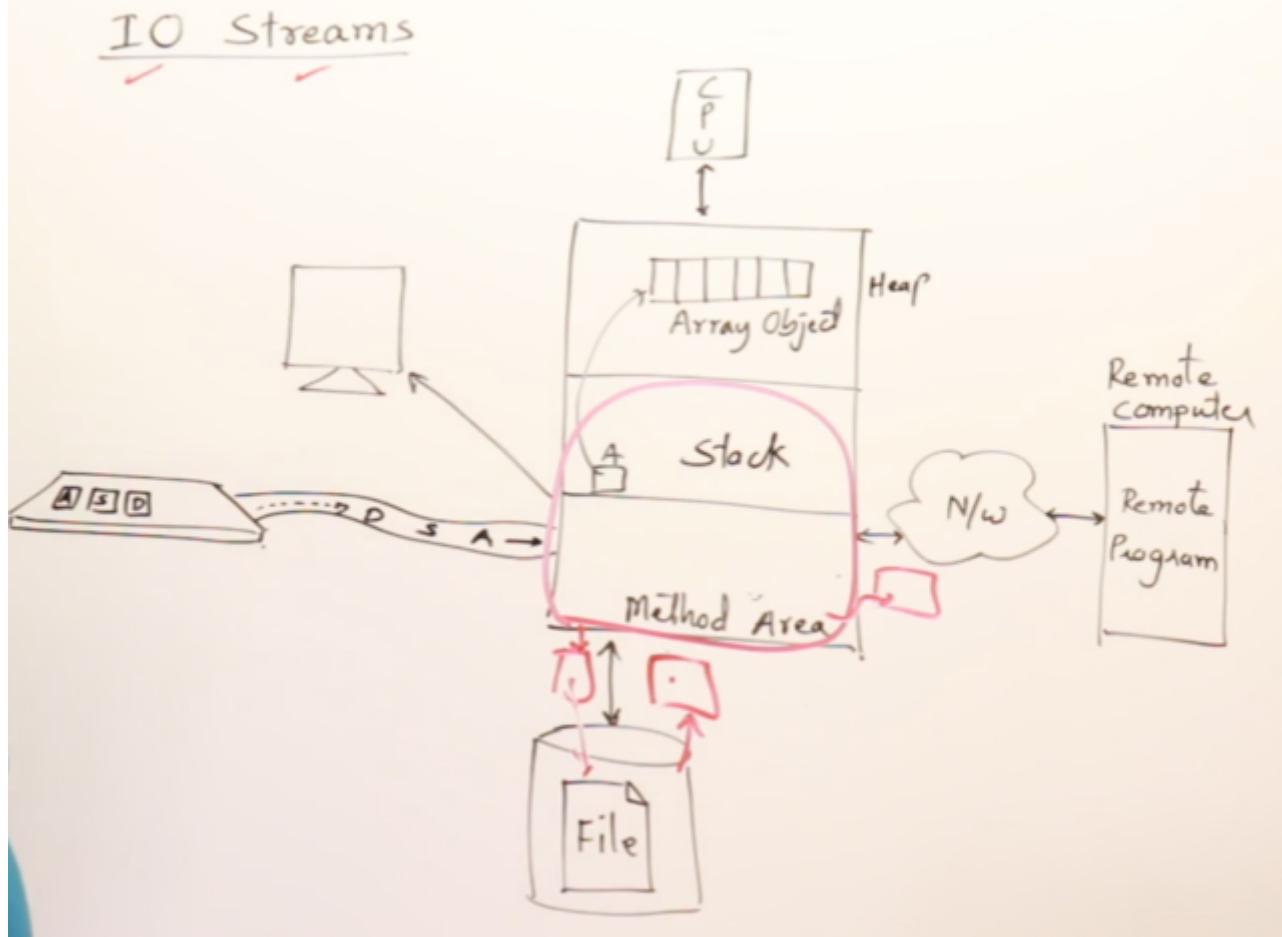


buffer is the memory object which holds the data for some time to bring compatibility between the fast and slow devices.



buffer is very important for the transfer of the data. buffer





java.io

Byte Stream

InputStream

OutputStream

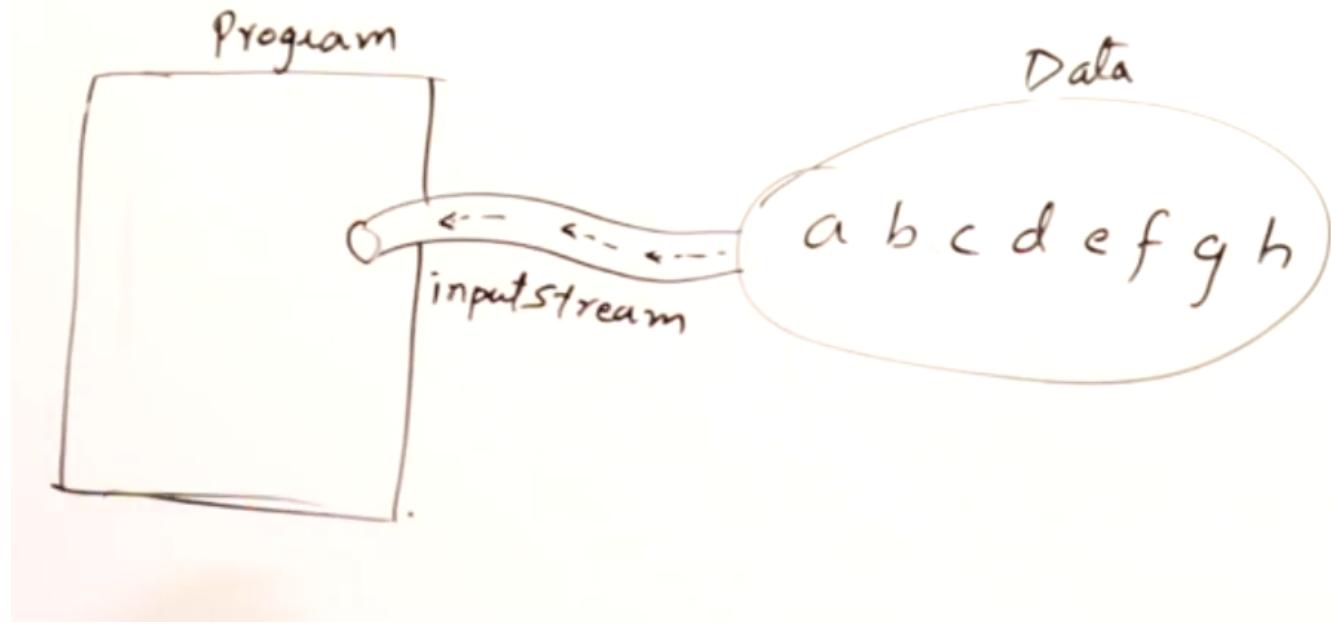
Character Stream

Reader

Writer

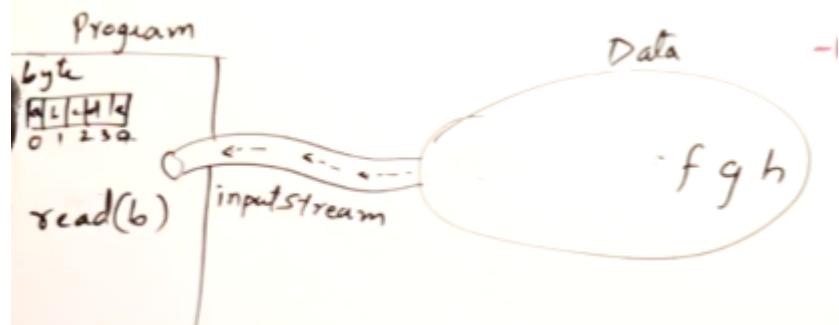
# class InputStream

- int read()
- int read(byte[] b)
- int read(byte[] b, int off, int len)
- int available()
- long skip(long n)
- void mark(int limit)
- void reset()
- boolean markSupported()
- void close()  
right?



`int read() //reads the one byte from the resource` `read()` means consuming the data is not there in the resource it is present in the program once consumed.

if nothing to return, then it returns the -1 `read(byte[] b)`



`int available()`

`int skip(long n) //skips the n bytes`

`void close() //closes the stream`

//below both works together `mark(int limit)` // `reset()` //above mark is possible only if it is buffered stream.  
buffer is a memory acts as the extra memory for holding the data.

- boolean markSupported()

- void close()

the mark.

ng tools

15min

Resources

821

822

823

824

825

826

827

828

829

830

831

832

833

218. FileInputStream & FileReader

8min

Resources

821

822

823

824

825

826

827

828

829

830

831

832

833

219. Student Challenge : Copy a File

8min

Resources

821

822

823

824

825

826

827

828

829

830

831

832

833

220. Byte Streams &  
CharArrayReader

13min

Resources

821

822

823

824

825

826

827

828

829

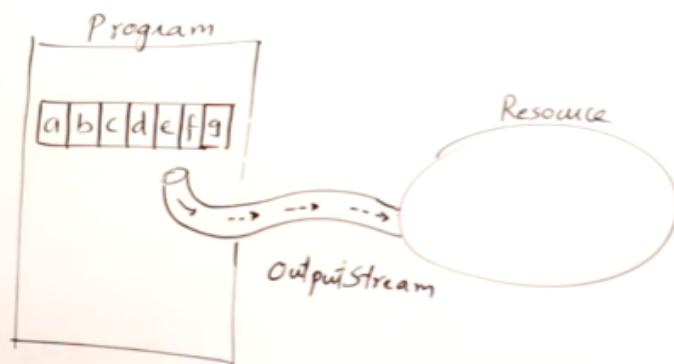
830

831

832

833

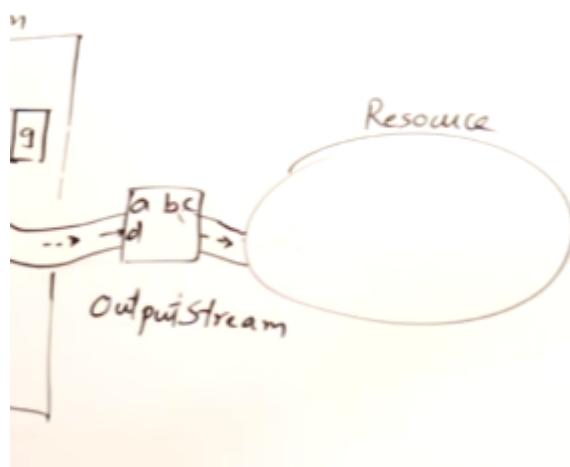
## InputStream & OutputStream



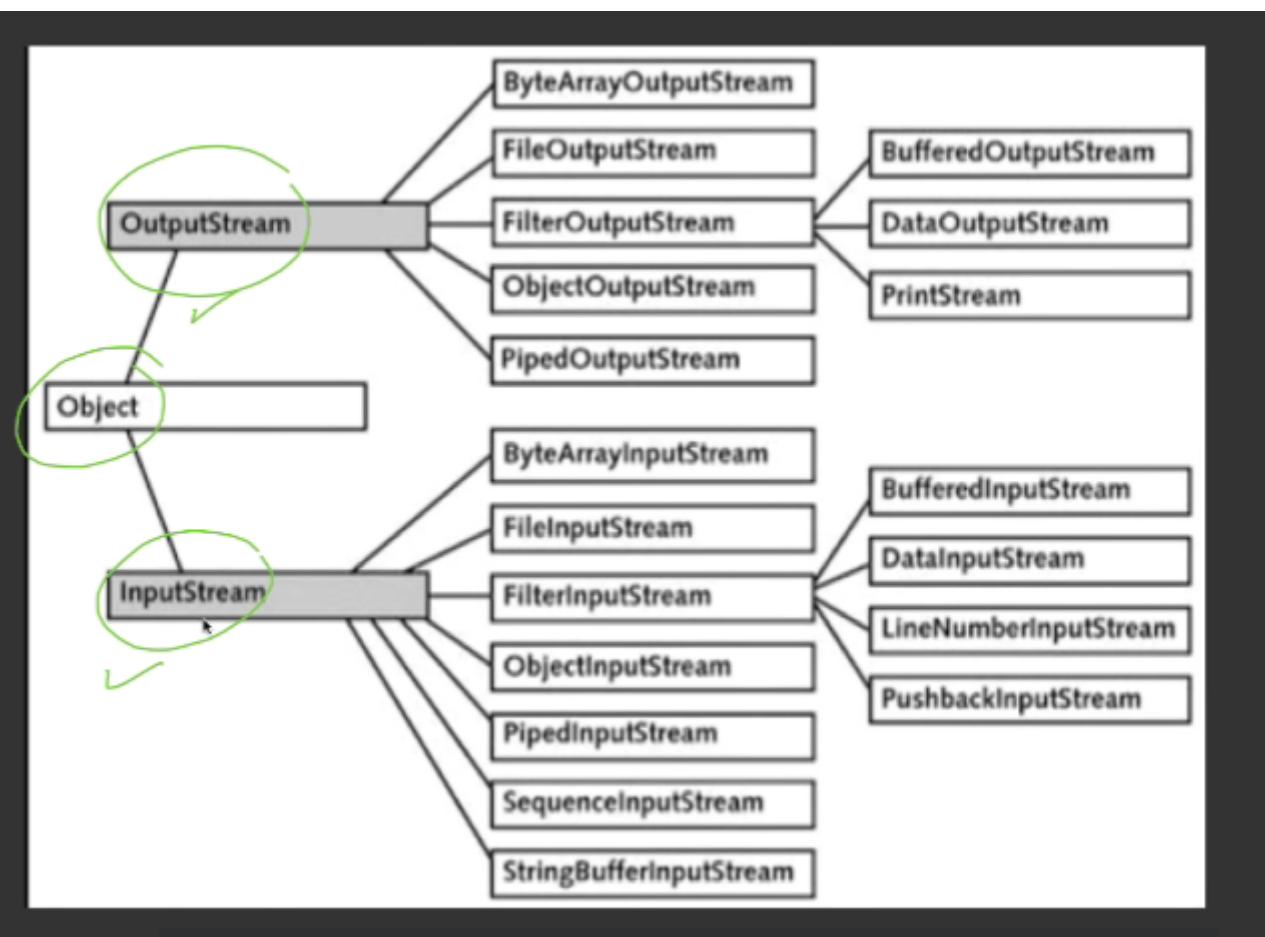
### class OutputStream

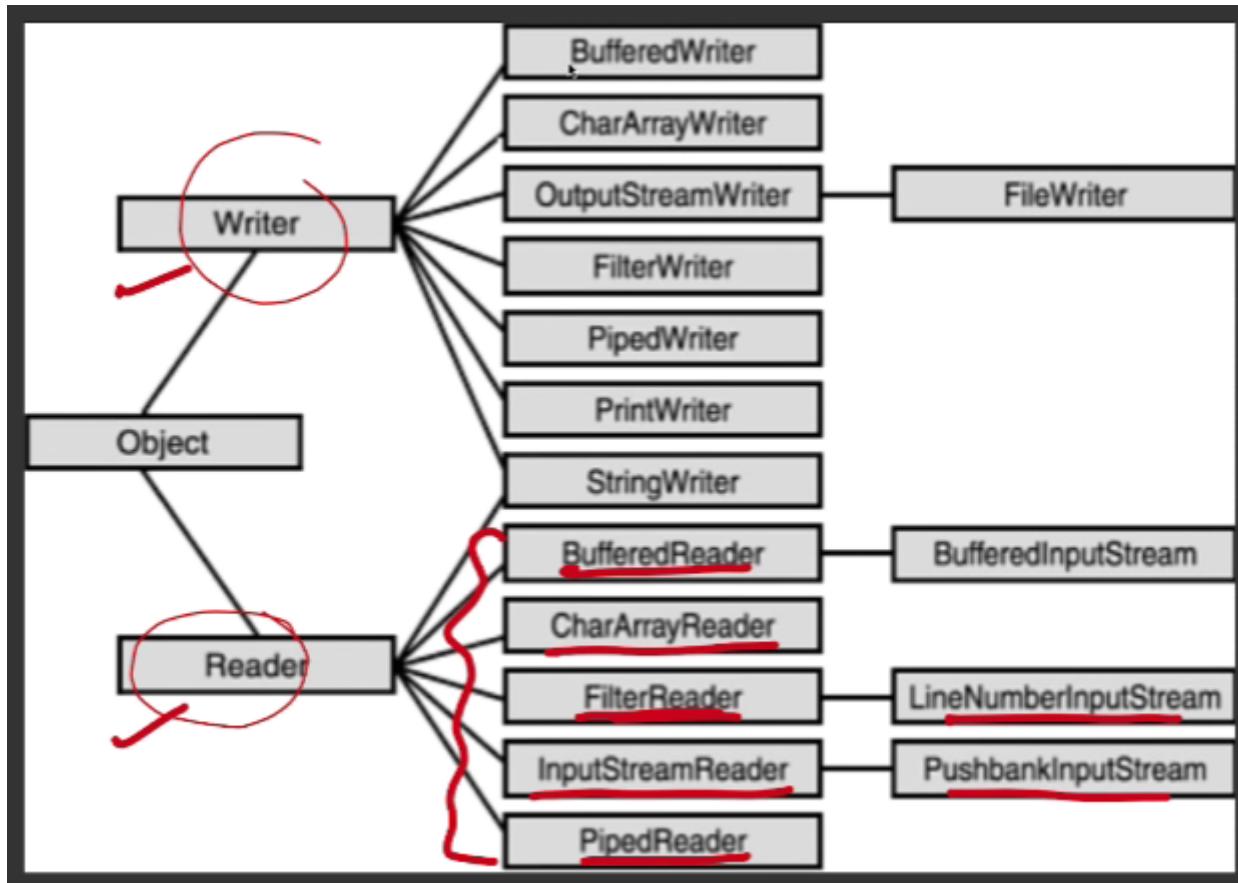
- void write(int b)
- void write(byte[] b)
- void write(byte[] b, int off, int len)
- void flush()
- void close()

flush() //flushes the data from the buffer to the resource. //works only in the buffered stream.



- java.lang.Object
  - java.io.InputStream
    - java.io.ByteArrayInputStream
    - java.io.FileInputStream
    - java.io.ObjectInputStream
    - java.io.PipedInputStream
    - java.io.SequenceInputStream
    - java.io.StringBufferInputStream
  - java.io.FilterInputStream
    - java.io.BufferedInputStream
    - java.io.DataInputStream
    - java.io.PushbackInputStream





PrintStream important

character streams

*character*

```

graph TD
    Object --> Writer
    Object --> Reader
    Writer --> BufferedWriter
    Writer --> CharArrayWriter
    Writer --> OutputStreamWriter
    Writer --> FilterWriter
    Writer --> PipedWriter
    Writer --> PrintWriter
    Writer --> StringWriter
    Reader --> BufferedReader
    Reader --> CharArrayReader
    Reader --> FilterReader
    Reader --> InputStreamReader
    Reader --> PushbackInputStream
    Reader --> PipedReader
    OutputStreamWriter --> FileWriter
  
```

Character takes two bytes and it also supports Unicode.

Made by Udemy, this generalized assessment is a great way to check in on your skills.

**Launch Assessment**

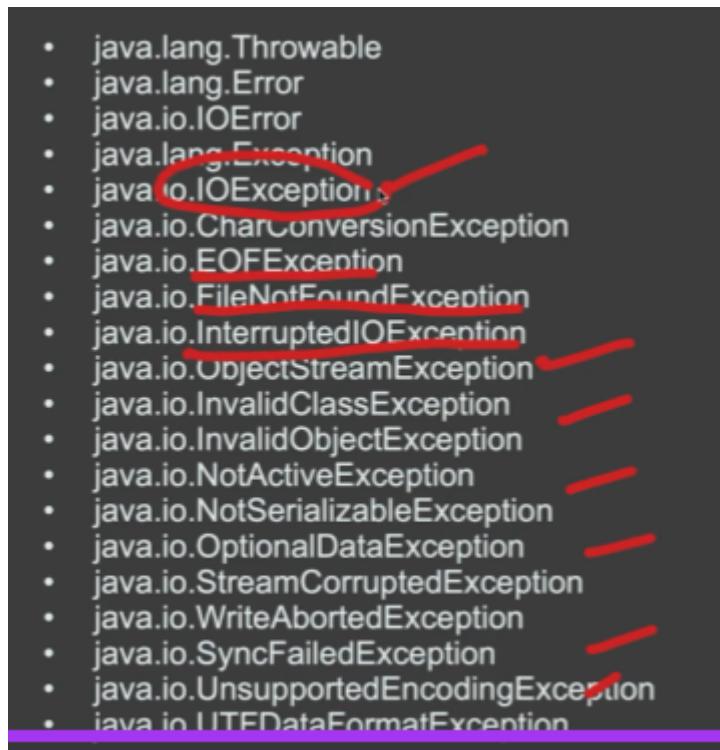
Course content

- Section 23: Java IO Streams
- Section 24: Java Generics
- Section 25: Collection Framework
- Section 26: Date and Time API
- Section 27: Network

- java.lang.Object
  - java.io.InputStream
    - java.io.ByteArrayInputStream
    - java.io.FileInputStream
    - java.io.ObjectInputStream
    - java.io.PipedInputStream
    - java.io.SequenceInputStream
    - java.io.StringBufferInputStream
  - java.io.FilterInputStream
    - java.io.BufferedInputStream
    - java.io.DataInputStream
    - java.io.PushbackInputStream
- ▷ java.io.OutputStream
  - java.io.ByteArrayOutputStream
  - java.io.FileOutputStream
  - java.io.ObjectOutputStream
  - java.io.PipedOutputStream
  - java.io.FilterOutputStream
    - java.io.BufferedOutputStream
    - java.io.DataOutputStream
    - java.io.PrintStream

- → java.io.RandomAccessFile
- java.io.StreamTokenizer
- File
- java.io.Reader
  - java.io.FileReader
  - java.io.BufferedReader
    - java.io.LineNumberReader
  - java.ioCharArrayReader
  - java.io.FilterReader
    - java.io.PushbackReader
  - java.io.InputStreamReader
    - java.io.FileReader
  - java.io.PipedReader
  - java.io.StringReader

- java.io.Writer
  - java.io.FileWriter
  - java.io.BufferedWriter
  - java.io.CharArrayWriter
  - java.io.FilterWriter
  - java.io.OutputStreamWriter
    - java.io.FileWriter
  - java.io.PipedWriter
  - java.io.PrintWriter
  - java.io.StringWriter



## File output stream

```
package fileexample;

import java.io.*;

public class FileExample {

    public static void main(String[] args) throws Exception
    {

        try(FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt"))
        {

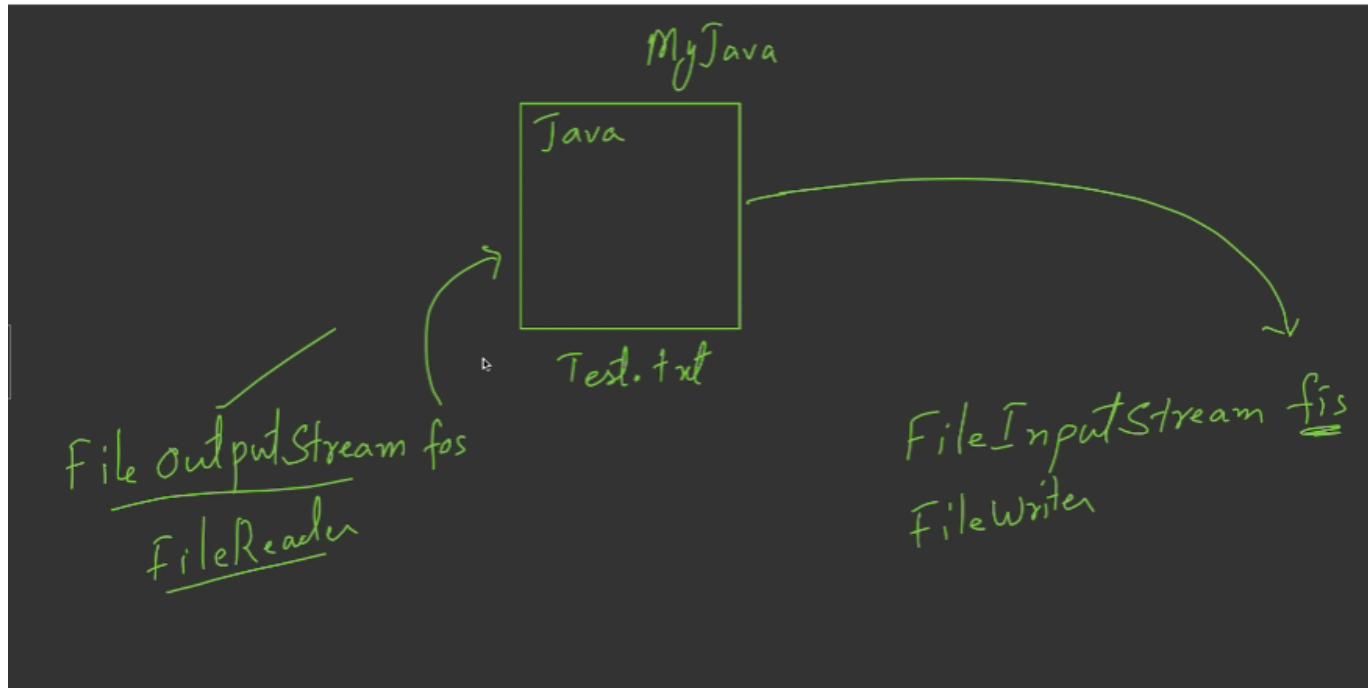
            String str="earn Java Programming.";
            byte b[]={str.getBytes();

            /*
            //fos.write(str.getBytes());
            for(byte x:b)
                fos.write(x);*/
            //fos.write(b, 6, str.length()-6);

            fos.write(b);
            //fos.close();

        }
        /*catch(FileNotFoundException e)
        {
```

```
        System.out.println(e);
    }
    catch(IOException e)
    {
        System.out.println(e);
    }/*
}
}
```



computer works mostly in bytes so we work with input and output streams.

## 218 FileInputStream & FileReader

```
public static void main(String[] args) throws Exception
{
    try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
    {
        byte b[] = new byte[fis.available()];
        fis.read(b);

    }
}
```

```
public static void main(String[] args) throws Exception
{
    try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
    {
        byte b[] = new byte[fis.available()];
        fis.read(b);
        String str = new String(b);
        System.out.println(str);
    }
}
```

```
package fileexample;

import java.io.*;

public class FileExample {

    public static void main(String[] args) throws Exception
    {
        try (FileOutputStream fos = new FileOutputStream("C:/MyJava/Test.txt"))
        {
            String str = "earn Java Programming.";
            byte b[] = str.getBytes();
            /*
            //fos.write(str.getBytes());
            for (byte x : b)
                fos.write(x); */
            //fos.write(b, 6, str.length() - 6);

            fos.write(b);
            //fos.close();
        }
        /*catch(FileNotFoundException e)
        {
            System.out.println(e);
        }
        catch(IOException e)
        {
            System.out.println(e);
        }*/
    }
}
```

```
}
```

```
}
```

```
try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
{
    int x;

    do
    {
        x=fis.read();
        if(x!=-1)
            System.out.print((char)x);
    }while(x!=-1);
```

commonly used

```
try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
{
    int x;

    while((x=fis.read()) != -1)
    {
        System.out.print((char)x);
    }
}
```

FileReader

```
try (FileReader fis = new FileReader("C:/MyJava/Test.txt"))
{
    int x;

    while((x=fis.read()) != -1)
    {
        System.out.print((char)x);
    }
}
```

219 sc: copy a file data to another.

```
package scio1;

import java.io.*;

public class SCIO1
{
    public static void main(String[] args) throws Exception
    {

        FileInputStream fis=new FileInputStream("Source1.txt");
        FileOutputStream fos=new FileOutputStream("Source2.txt");

        int b;
        while((b=fis.read())!=-1)
        {
            if(b>=65 && b<=90)fos.write(b+32);
            else fos.write(b);
        }

        fis.close();
        fos.close();

    }
}
```

```
package scio1;

import java.io.*;

public class SCIO1
{
    public static void main(String[] args) throws Exception
    {

        FileInputStream fis1=new FileInputStream("Source1.txt");
        FileInputStream fis2=new FileInputStream("Source2.txt");

        FileOutputStream fos=new FileOutputStream("Destination.txt");

        SequenceInputStream sis=new SequenceInputStream(fis1,fis2);

        int b;
        while((b=sis.read())!=-1)
        {

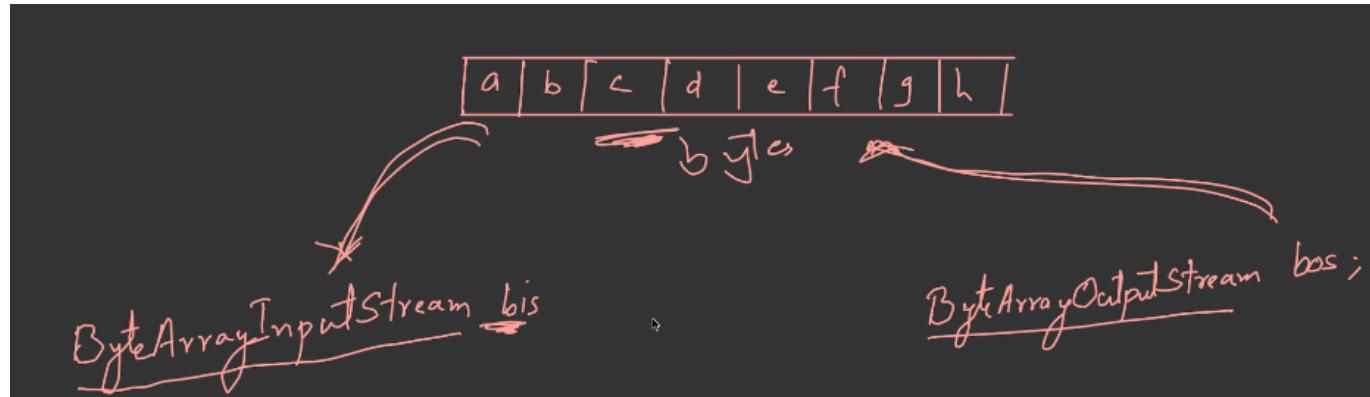
            fos.write(b);
        }

        sis.close();
        fis1.close();
        fis2.close();
        fos.close();

    }
}
```

```
    }  
}
```

## Byte Stream



```
package bytedemo;  
  
import java.io.*;  
  
public class ByteDemo  
{  
    public static void main(String[] args) throws Exception  
    {  
        byte b[]={‘a’,‘b’,‘c’,‘d’,‘e’,‘f’,‘g’,‘h’,‘i’,‘j’};  
        ByteArrayInputStream bis=new ByteArrayInputStream(b);  
        int x;  
        while((x=bis.read())!=-1)  
        {  
            System.out.print((char)x);  
        }  
        bis.close();  
    }  
}
```

if byte array stream is comming from stream

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={‘a’,‘b’,‘c’,‘d’,‘e’,‘f’,‘g’,‘h’,‘i’,‘j’};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

        String str=new String(bis.readAllBytes());

        System.out.println(str);

        bis.close();
    }
}
```

for files it is not supported mark is supported on byte. it means we can go forward and backward to read or

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={‘a’,‘b’,‘c’,‘d’,‘e’,‘f’,‘g’,‘h’,‘i’,‘j’};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

        String str=new String(bis.readAllBytes());

        System.out.println(bis.markSupported());           ↴

        bis.close();
    }
}
```

write data. o/p: true

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        ByteArrayOutputStream bos=new ByteArrayOutputStream(20);

        bos.write('a');
        bos.write('b');
        bos.write('c');
        bos.write('d');

        byte b[]=bos.toByteArray();

        for(byte x:b)
            System.out.println((char)x);

        bos.close();
    }
}
```

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        ByteArrayOutputStream bos=new ByteArrayOutputStream(20);

        bos.write('a');
        bos.write('b');
        bos.write('c');
        bos.write('d');

        byte b[]=bos.toByteArray();

        for(byte x:b)
            System.out.println((char)x);

        bos.close();
    }
}
```

stored in the form of bytes

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        ByteArrayOutputStream bos=new ByteArrayOutputStream(20);

        bos.write('a');
        bos.write('b');
        bos.write('c');
        bos.write('d');

        bos.writeTo(new FileOutputStream("/Users/abdulbari/Desktop/Test.txt"));

        bos.close();
    }
}
```

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        char c[]={ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};

        CharArrayReader cr=new CharArrayReader(c);

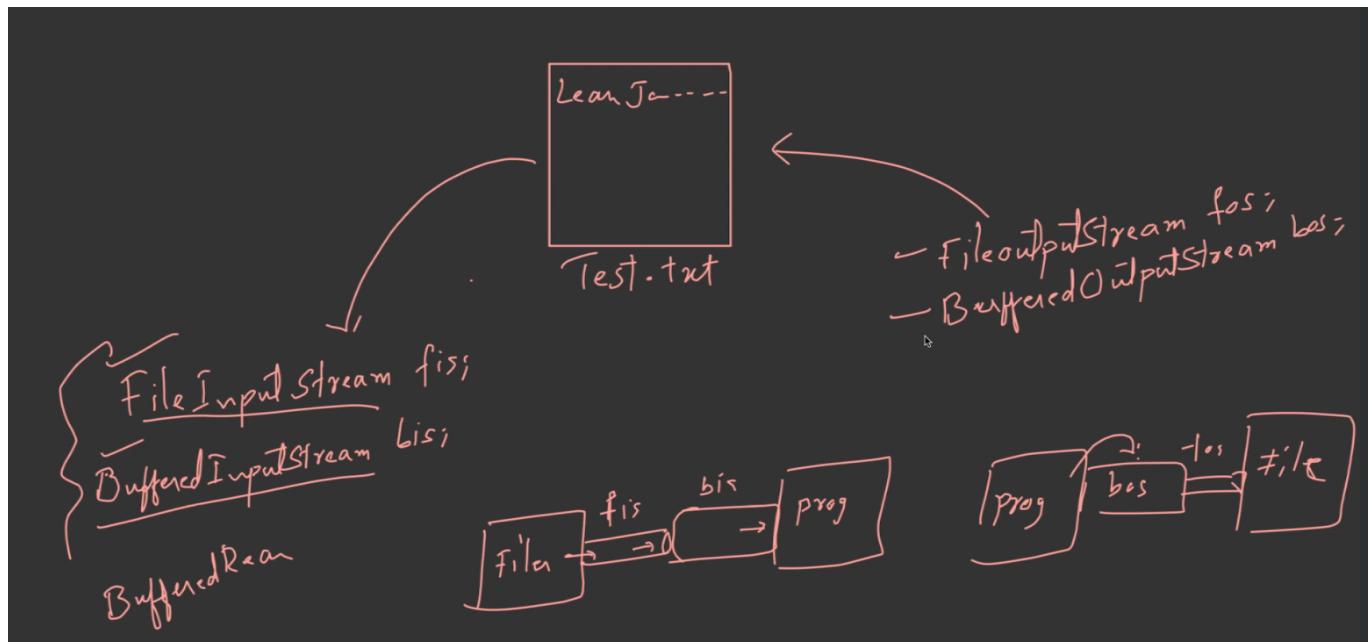
        int x;

        while((x=cr.read())!=-1)
        {
            System.out.println((char)x);
        }

        cr.close();
    }
}
```

char writer. practice char array

## 221 Buffered Stream & Buffered Reader



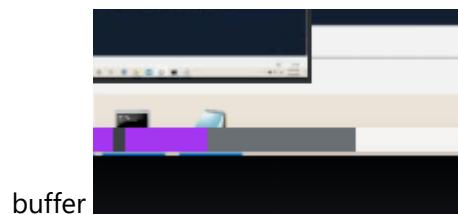
```
public class BufferedDemo
{
    public static void main(String[] args) throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        int x;
        while((x=bis.read())!=-1)
        {
            System.out.print((char)x);
        }
    }
}
```

benefits of using the buffered

```
public class BufferedDemo
{
    public static void main(String[] args) throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        System.out.println("File "+fis.markSupported());
        System.out.println("Buffer "+bis.markSupported());
    }
}
```



```
public static void main(String[] args) throws Exception
{
    FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
    BufferedInputStream bis=new BufferedInputStream(fis);

    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    bis.mark(10);
    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    bis.reset();

    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
```

```
public static void main(String[] args) throws Exception
{
    FileReader fis=new FileReader("C:/MyJava/Test.txt");
    BufferedReader bis=new BufferedReader(fis);

    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    bis.mark(10);
    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
    bis.reset();

    System.out.print((char)bis.read());
    System.out.print((char)bis.read());
```

Readers are used for characters InputStreams are used for bytes

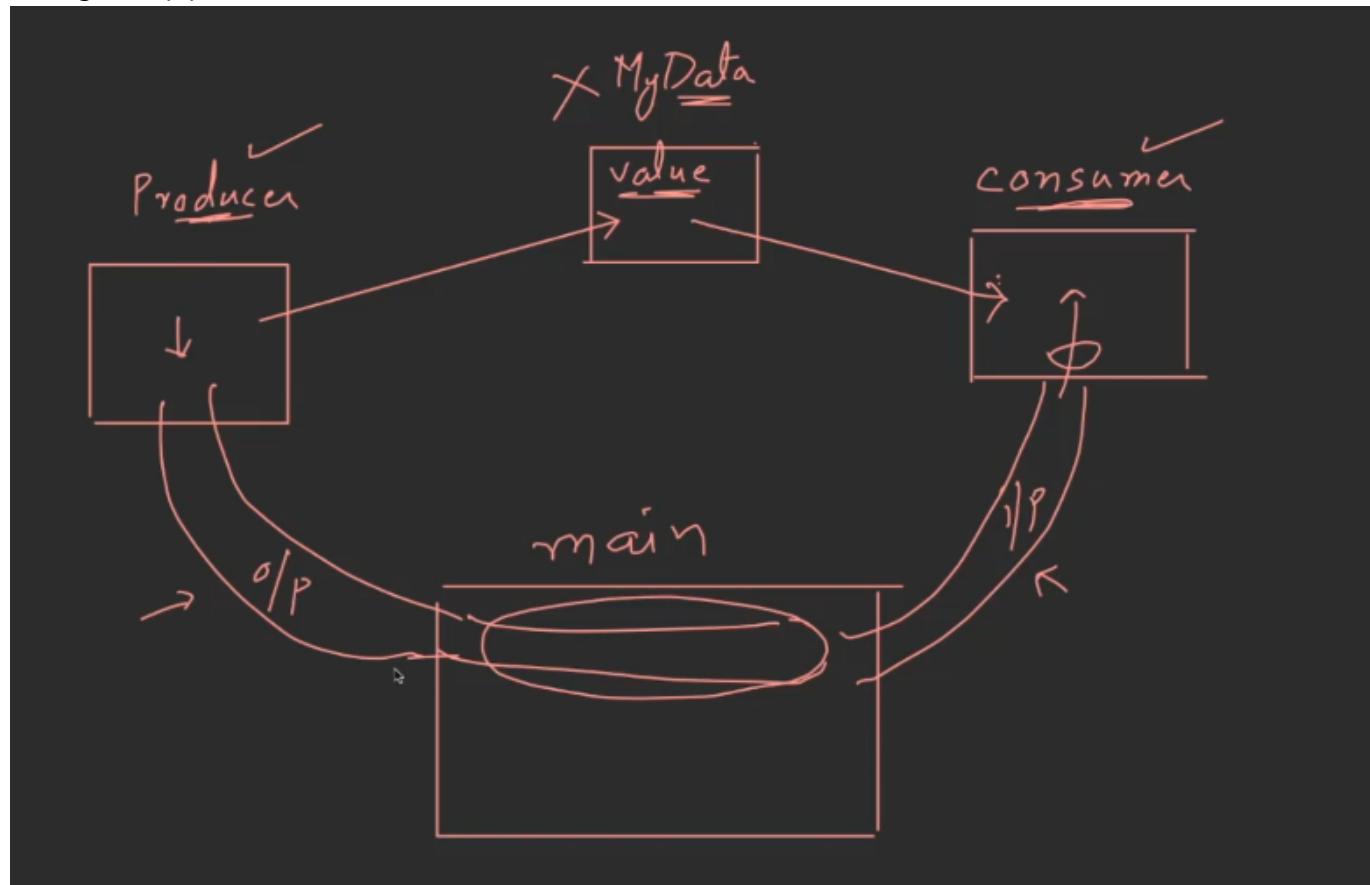
```
System.out.print((char)bis.read());
bis.reset();

System.out.print((char)bis.read());
System.out.print((char)bis.read());

System.out.println("String "+bis.readLine());
```

## Piped Streams

(eg.in multithreading > inter thread communication) Here, separate thread for P and C they share the data through the pipes.



```
package pipeddemo;

import java.io.*;

class Producer extends Thread
{
    OutputStream os;
    ...
```

```
public Producer(OutputStream o)
{
    os=o;
}

public void run()
{
    int count=1;

    while(true)
    {
        try{
            os.write(count);
            os.flush();

            System.out.println("Producer "+count);
            System.out.flush();

            Thread.sleep(10);
            count++;
        }catch(Exception e){}
    }
}

class Consumer extends Thread
{
    InputStream is;

    public Consumer(InputStream s)
    {
        is=s;
    }

    public void run()
    {
        int x;

        while(true)
        {
            try{
                x=is.read();

                System.out.println("Consumer "+x);
                System.out.flush();
                Thread.sleep(10);

            }catch(Exception e){}
        }
    }
}
```

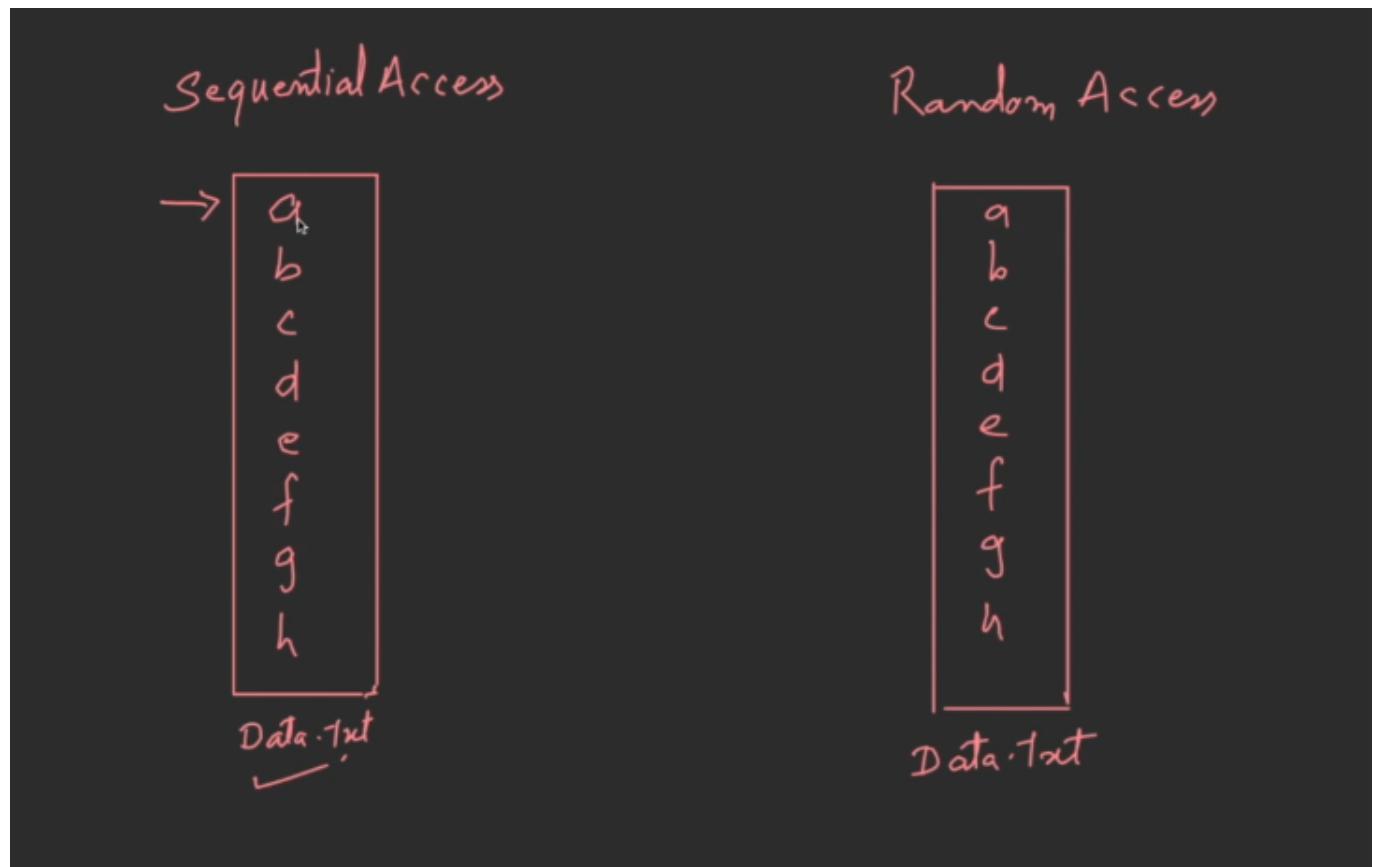
```
public class PipedDemo
{
    public static void main(String[] args) throws Exception
    {
        PipedInputStream pis=new PipedInputStream();
        PipedOutputStream pos=new PipedOutputStream();

        pos.connect(pis); //pis.connect(pos); both are same.

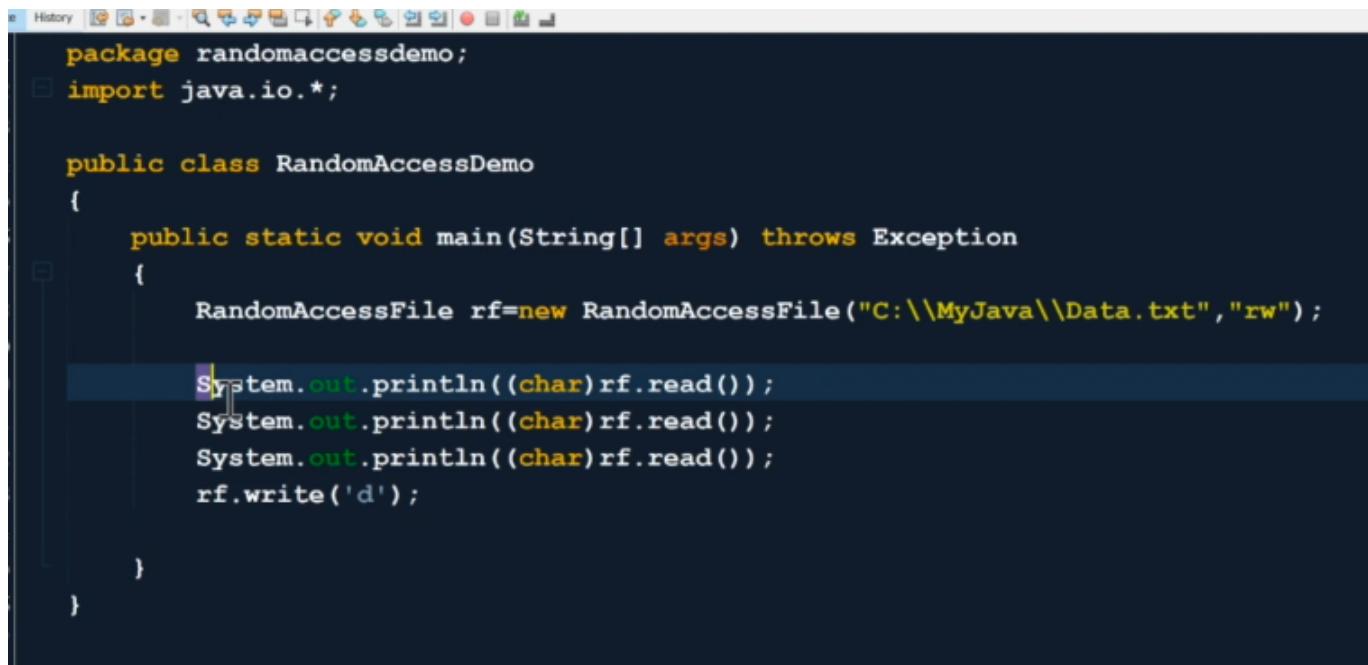
        Producer p=new Producer(pos);
        Consumer c=new Consumer(pis);

        p.start();
        c.start();
    }
}
```

## Random Access File



seek(3)

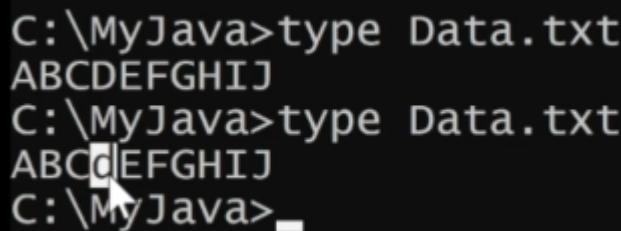


```
package randomaccessdemo;
import java.io.*;

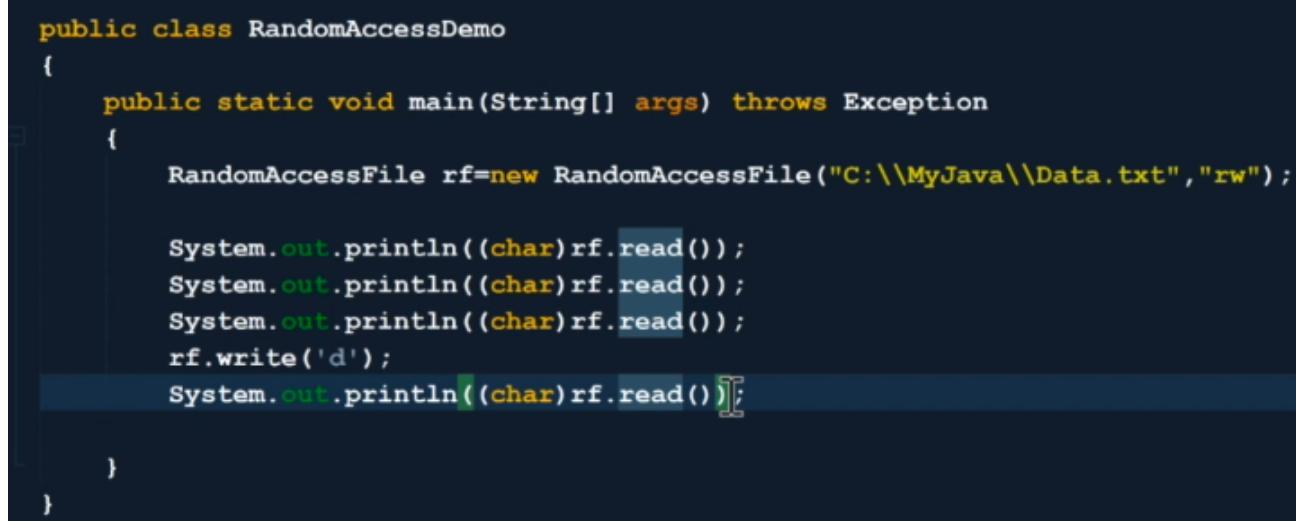
public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');

    }
}
```



```
C:\MyJava>type Data.txt
ABCDEFGHIJ
C:\MyJava>type Data.txt
ABCDEFGHIJ
C:\MyJava>
```



```
public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
    }
}
```

A B C E

```

package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
        //byte b[]={ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
        System.out.println((char)rf.read());
    }
}

```

```

package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
        //byte b[]={ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
        System.out.println((char)rf.read());
        rf.seek(3);
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println(rf.getFilePointer());
    }
}

```

get current file pointer position  
possible

+2 and ...

## 224 File Class

to access the properties of the files.

```
//check about the File class
package filehandling;
```

```

import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
        // select path of any directory on your computer
        File f=new File("C:\\MyJava");

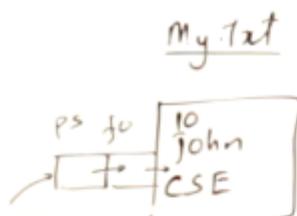
        System.out.println(f.isDirectory());
        File list[] = f.listFiles();

        for(File x:list)
        {
            System.out.println(x.getParent()+" "+x.getName());
        }
    }
}

```

## Serialization : storing data into file

=(to remember) pencil, scale , rubber, sharpner and box everything is stored in the form of strings. below



```

class MyWrite
{
    public static void main() throws Exception
    {
        FileOutputStream fo=new FileOutputStream('my.txt');

        PrintStream ps=new PrintStream(fo);
        Student s=new Student();
        s.rollno=10; s.name="John"; s.dept="CSE";

        ps.println(s.rollno);
        ps.println(s.name);
        ps.println(s.dept);
    }
}

```

## Serialization

```

class Student
{
    int rollno;
    String name;
    String dept;
}

class MyRead
{
    public void main() throws Exception
    {
        FileInputStream fis=new FileInputStream("My.txt");
        BufferedReader br=new BufferedReader(
            new InputStreamReader(fis));
        Student s=new Student();
        s.rollno=Integer.parseInt(br.readLine());
        s.name=br.readLine();
        s.dept=br.readLine();
        System.out.println(s.rollno+" "+s.name+" "+s.dept);
    }
}

```

```

public static void main(String[] args) throws Exception
{
    FileOutputStream fos=new FileOutputStream("C:\\MyJava\\Student1.txt");
    PrintStream ps=new PrintStream(fos);

    Student s=new Student();
    s.rollno=10;
    s.name="John";
    s.dept="CSE";

    ps.println(s.rollno);
    ps.println(s.name);
    ps.println(s.dept);

    ps.close();
    fos.close();
}

```

bridge class type casting won't works here. so parse int is used.

```
public class PrintStreamDemo
{
    public static void main(String[] args) throws Exception
    {
        FileInputStream fis=new FileInputStream("C:\\MyJava\\Student1.txt");
        BufferedReader br=new BufferedReader(new InputStreamReader(fis));

        Student s=new Student();
        s.rollno=Integer.parseInt(br.readLine());
        s.name=br.readLine();
        s.dept=br.readLine();                                }

        System.out.println("Roll No"+s.rollno);
        System.out.println("Name "+s.name);
        System.out.println("Dept "+s.dept);

    }
}
```

## 227. Serialization : using DataInput and DataOutput Stream

formatted storing as per their data types

My.txt

↓ (for reference)

E.g.

```
class MyWrite
{
    public static void main() throws Exception
    {
        FileOutputStream fo=new FileOutputStream("my.txt");
        DataOutputStream d=new DataOutputStream(fo);
        → Student s=new Student();
        s.rollno=10; s.name="John"; s.dept="CSE";
        → d.writeInt(s.rollno);
        → d.writeUTF(s.name);
        → d.writeUTF(s.dept);
    }
}
```

```
class Student  
{  
    int rollno;  
    String name;  
    String dept;  
}
```

```
class MyRead  
{
```

```
    public static void main() throws Exception  
{
```

```
        FileInputStream fis=new FileInputStream("My.txt");  
        DataInputStream d=new DataInputStream(fis);
```

```
        Student s=new Student();
```

```
        s.rollno=d.readInt();
```

```
        s.name=d.readUTF();
```

```
        s.dept=d.readUTF();
```

```
        System.out.println(s.rollno + " " + s.name + " " + s.dept);
```

```
}
```

## 228. Data Streams

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** The title bar displays "Start Page" and "DataStreamDemo.java".
- Toolbar:** A standard toolbar with icons for file operations like Open, Save, Print, and others.
- Status Bar:** A status bar at the bottom right corner says "Press Esc to exit full screen".
- Code Editor:** The main area contains the Java code for `DataStreamDemo.java`. The code defines a `Student` class with attributes `rollno`, `name`, and `dept`. It also defines a `DataStreamDemo` class with a `main` method that creates a `Student` object and writes its data to a file named `Student2.txt`.

```
1 package datastreamdemo;
2 import java.io.*;
3
4 class Student
5 {
6     int rollno;
7     String name;
8     String dept;
9 }
10 public class DataStreamDemo
11 {
12     public static void main(String[] args) throws Exception
13     {
14         FileOutputStream fos=new FileOutputStream("C:\\MyJava\\Student2.txt");
15         DataOutputStream dos=new DataOutputStream(fos);
16
17         Student s=new Student();
18         s.rollno=10;
19         s.name="John";
20         s.dept="CSE";|
21     }
22 }
```

```
package datastreamdemo;
import java.io.*;

class Student
{
    int rollno;
    String name;
    float avg;
    String dept;
}
public class DataStreamDemo
{
    public static void main(String[] args) throws Exception
    {
        //writing in a file
        FileOutputStream fos=new FileOutputStream("Student2.txt");
        DataOutputStream dos=new DataOutputStream(fos);

        Student s1=new Student();
        s1.rollno=10;
        s1.name="John";
        s1.dept="CSE";
        s1.avg=75.9f;
        dos.writeInt(s1.rollno);
        dos.writeUTF(s1.name);
        dos.writeUTF(s1.dept);
        dos.writeFloat(s1.avg);
        dos.close();
        fos.close();
    }
}
```

```
//reading from file
FileInputStream fis=new FileInputStream("Student2.txt");
DataInputStream dis=new DataInputStream(fis);

Student s=new Student();

s.rollno=dis.readInt();

s.name=dis.readUTF();

s.dept=dis.readUTF();
s.avg=dis.readFloat();

System.out.println("Roll no "+s.rollno);
System.out.println("Name "+s.name);
System.out.println("Average "+s.avg);
System.out.println("Dept "+s.dept);

dis.close();
fis.close();
}

}
```

boxes means binary if ur able to read all whole things then it is string format otherwise it is binary format.

## 229 Serialization

=serialization is the process of storing the state of the object and its data into the file. restoring the object from the file is known as the deserialization. this is done through the 'ObjectInputStream' and ' ObjectOutputStream' classes.



```
class MyWrite
{
    public static void main() throws Exception
    {
        FileOutputStream fo=new FileOutputStream("My.txt");
        ObjectOutputStream oos=new ObjectOutputStream(fo);
        Student s=new Student();
        s.rollno=10; s.name="John"; s.dept="CSE";
        oos.writeObject(s);
    }
}
```

```
class Student
{
    int rollno;
    String name;
    String dept;
}
```

```
class MyRead
{
```

p.s.v.main() throws Exception

```
}
```

→ FileInputStream fis=new FileInputStream("r")  
→ ObjectInputStream ois=new ObjectInputStream(fis)

Student s;

s=(Student) ois.readObject();

} , S.O.P(s.rollno + " " + s.name + " " + s.dept)

Serialization

```

class Student {
    int rollno;
    String name;
    String dept;
}

class MyRead {
    public static void main() throws Exception {
        FileInputStream fis = new FileInputStream("my.txt");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Student s = (Student) ois.readObject();
        System.out.println(s.rollno + " " + s.name + " " + s.dept);
    }
}

class MyWrite {
    public static void main() throws Exception {
        FileOutputStream fo = new FileOutputStream("my.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fo);
        Student s = new Student();
        s.name = "John";
        s.dept = "CSE";
        oos.writeObject(s);
    }
}

```

it has to implements the serializable

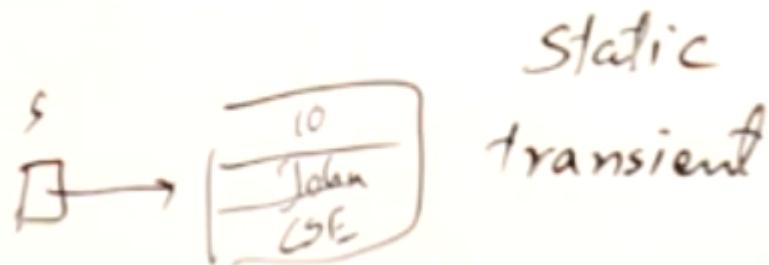
class Student implements Serializable

```

class Student implements Serializable {
    int rollno;
    String name;
    String dept;
}

```

Static  
transient



## 230 Object Streams and Serialization

```
package objectdemo;
import java.io.*;

class Student implements Serializable
{
    private int rollno;
    private String name;
    private float avg;
    private String dept;
    public static int Data=10; //static variable is not serialized //not stored in
the file
    public transient int t; //transient variable is not serialized //not stored in
the file
    public volatile int v; //volatile is serialized //stored in the file //check
once.

    public Student() //without this gets error. InvalidClassException //
serialVersionUID has to match with deserialized class

    {

    }

    public Student(int r,String n,float a,String d)
    {
        rollno=r;
        name=n;
        avg=a;
        dept=d;
        Data=500;
        t=500;
    }

    public String toString()
    {
        return "\nStudent Details\n"+
               "\nRoll "+rollno+
               "\nName "+name+
               "\nDept "+dept+
               "\nAvg "+avg+
               "\nData "+Data+
               "\nT "+t;
    }
}
```

```
        "\nAverage "+avg+
        "\nDept "+dept+
        "\nData "+Data+
        "\nTransient "+t+"\n";
    }

}

public class ObjectDemo
{
    /* public static void main(String[] args) throws Exception
    {
        FileOutputStream fos=new FileOutputStream("C:\\MyJava\\Student3.txt");
        ObjectOutputStream oos=new ObjectOutputStream(fos);

        Student s=new Student(10,"John",89.9f,"CSE");

        oos.writeObject(s);

        fos.close();
        oos.close();

    }
    */
    public static void main(String[] args) throws Exception
    {
        FileInputStream fis=new FileInputStream("C:\\MyJava\\Student3.txt");
        ObjectInputStream ois=new ObjectInputStream(fis);

        Student s=(Student)ois.readObject();

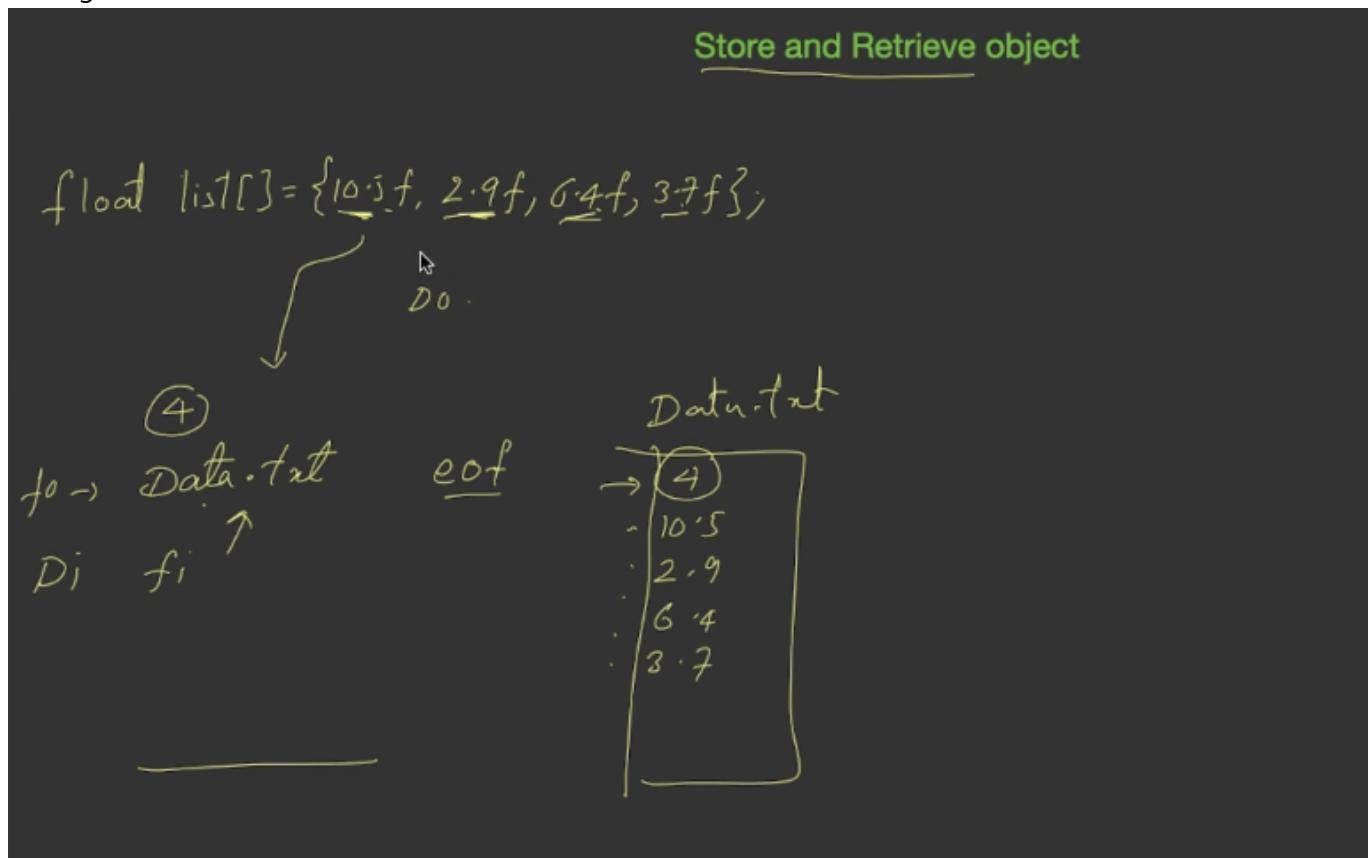
        System.out.println(s);

        fis.close();
        ois.close();

    }
}
```

## 231. sc : serialize customer IMP IMP

storing the list of floats into a file and retrieve.



```

package scio2;
import java.io.*;

public class SCIO2
{
    public static void main(String[] args) throws Exception
    {
        float list[]={1.2f , 3.45f , 6.78f , 9.01f , 2.34f};

        FileOutputStream fos=new FileOutputStream("List.txt");
        DataOutputStream dos=new DataOutputStream(fos);

        dos.writeInt(list.length);
        for(float f:list)
        {
            dos.writeFloat(f);
        }

        dos.close();
        fos.close();

        FileInputStream fis=new FileInputStream("List.txt");
        DataInputStream dis=new DataInputStream(fis);
        int length=dis.readInt();
        float data;

        for(int i=0;i<length;i++)
        {
    
```

```
        data=dis.readFloat();
        System.out.println(data);
    }
    dis.close();
    fis.close();

}
}
```

searching

```
package scio3;

import java.io.*;

class Customer implements Serializable
{
    String custID;
    String name;
    String phno;

    static int count=1;

    Customer()
    {

    }
    Customer(String n,String p)
    {
        custID="C"+count;
        count++;
        name=n;
        phno=p;
    }

    public String toString()
```

```
public static void main(String[] args) throws Exception
{
    java.util.Scanner sc=new java.util.Scanner(System.in);

    FileInputStream fis=new FileInputStream("Customer.txt");
    ObjectInputStream ois=new ObjectInputStream(fis);

    int length=ois.readInt();

    Customer list[]=new Customer[length];

    for(int i=0;i<length;i++)
    {

        Customer list[]=new Customer[length];
        |
        for(int i=0;i<length;i++)
        {
            list[i]=(Customer)ois.readObject();
        }

        System.out.println("Enter Name of Customer");
        String name=sc.nextLine();

        for(int i=0;i<length;i++)
        {
            if(name.equalsIgnoreCase(list[i].name))
                System.out.println(list[i]);
        }

        ois.close();
        fis.close();
    }
}
```



Good job!

Question 6:

What is a *buffer*?

- The cable that connects a data source to the bus.
- Any stream that deals with character IO.
- A file that contains binary data.
- A section of memory used as a temporary storing area for input or output data.



Good job!

Question 6:

What is a *buffer*?

- The cable that connects a data source to the bus.
- Any stream that deals with character IO.
- A file that contains binary data.
- A section of memory used as a temporary storing area for input or output data.



Good job!



## Question 7:

Which of these classes can write the back into InputStream?

 BufferedReader BufferedWriter PushbackReader CharArrayReader

## Question 8:

Which data member is used to enable future modifications in the Serializable classes?

 mutable serialVersionUID serializable changeLog

Question 9:

Which of these class is used to read characters and strings in Java from console?

**BufferedReader**

**FileReader**

**PrintStream**

**InputStreamReader**

Question 10:

To create a folder programmatically, object of which class is required?

**Dir**

**File**

**Folder**

**OutputStream**

## Section 24 : Generics

---

every class is directly or indirectly child of the object class. Generalization

## Generics

class Object

Object obj=new String("Hi");

Array of Object

Generic Type Array

- . Type Safety
- . Compile-Time Checking
- . No Typecasting

Object obj=new String("Hi");

String str = ~~obj~~;

String str=(String) obj;

no compile time error only runtime error

```
package genericdemo;

public class GenericDemo
{
    public static void main(String[] args)
    {
        Object obj=new String("Hello");

        obj=new Integer(10);

        String str=(String)obj;
    }
}
```

Object can be used as the generalization but there are few problems associated as the above image.

Generics

```

class Object
✓ Array of Object
Generic Type Array
    . Type Safety
    . Compile-Time Checking
    . No Typecasting

```

Object obj[] = new Object[3];

obj [ 0 ] [ 1 ] [ 2 ]

'Hi' 'Bye' 10

GenericDemo.java

```

{
    Object obj[] = new Object[3];

    obj[0] = "hi";
    obj[1] = "bye";
    obj[2] = new Integer(10);

    String str;
    for(int i=0; i<3; i++)
    {
        str = (String) obj[i];
        System.out.println(str);
    }
}

```

runtime: error

```

public class GenericDemo<T>
{
    T data[] = (T[]) new Object[3];

    public static void main(String[] args)
    {
        GenericDemo<String> gd = new GenericDemo();
    }
}

```

```
package genericdemo;

public class GenericDemo<T>
{
    T data[]={T[]} new Object[3];

    public static void main(String[] args)
    {
        GenericDemo<String> gd=new GenericDemo();

        gd.data[0]="hi";      incompatible types: Integer cannot be converted to String
        gd.data[1]="bye";
        gd.data[2]=new Integer(10);

        String str=gd.data[0];
    }
}
```

## 233 Defining the Generic Class

## Generics

```
class Data<T>
{
    1-value → data
    . getData()
    → setData()
}
```

### Examples

✓ 1. class Data

2. class MyArray

```
class Data<T>
{
    private T obj;

    public void setData(T v)
    {
        obj=v;
    }
    public T getData()
    {
        return obj;
    }
}
```

```
public class GenericDemo
{
    public static void main(String[] args)
    {
        Data<Integer> d=new Data<>();
        d.setData(new Integer(10));
        System.out.println(d.getData());
    }
}
```

```
class MyArray<T>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(T x:A)
        {
            System.out.println(x);
        }
    }
}
```

```
public static void main(String[] args)
{
    MyArray<Integer> ma=new MyArray<>();

    ma.append(10);
    ma.append(20);
    ma.append(30);

    ma.display();
}
```

## 234 Bounds of Generics

1. No Parameters ✓
2. Multiple Parameters
3. Subtypes
4. Bounded Types

MyArray<String> ma=

if parameter are not give it becomes object

```
public static void main(String[] args)
{
    MyArray ma=new MyArray();

    ma.append("Hi");
    ma.append(new Integer(10));
    ma.append("Go");

    ma.display();
```

2 parameters

```
class MyArray<T,K>
```

### 3. Subtypes

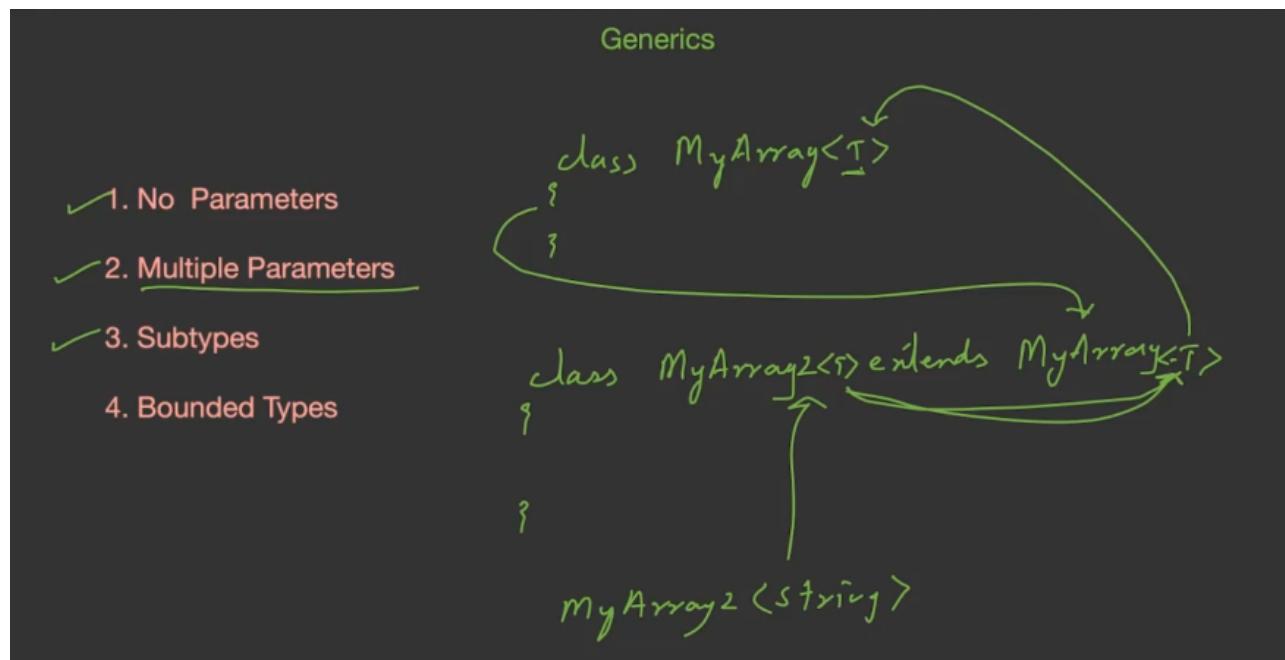
```
class MyArray2 extends MyArray<String>
{

public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray2 ma=new MyArray2();

        ma.append(10);
        ma.append("Bye");
        ma.append("Go");

        ma.display();
    }
}
```



## 4. Bounded types



```
@SuppressWarnings("unchecked")  
  
class MyArray<T extends Number>  
{  
    T A[]=(T[]) new Object[10];  
    int length =0;  
  
    public void append(T v)  
    {  
        A[length++]=v;  
    }  
  
    public void display()  
    {  
        for(int i=0;i<length;i++)  
        {  
            System.out.println(A[i]);  
        }  
    }  
}
```

only extends is used for

```
package genericdemo;  
  
interface A{}  
class B implements A{}  
class C implements A{}  
  
@SuppressWarnings("unchecked")
```

```
class MyArray<T extends A>  
{  
    T A[]=(T[]) new Object[10];  
    int length =0;  
  
    public void append(T v)  
    {  
        A[length++]=v;  
    }  
}
```

classes and interfaces

## 235 Generic Methods IMP

1. Generic Methods

2. WildCard ?

3. Lower Bound

4. Upper Bound

before the return type generic type has to be defined

```
static <T> void show(T[] list)
{
}
```

works for any types of data

```
public class GenericDemo
{
    static <E> void show(E[] list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
        show(new String[]{"Hi","Go","Bye"});
        show(new Integer[]{10,20,30,40});
    }
}
```

```
public class GenericDemo
{
    static <E> void show(E... list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
        show("Hi","Go","Bye");
        show(10,20,30,40);
    }
}
```

bound types works in

the generic methods

```
public class GenericDemo
{
    static <E extends Number> void show(E... list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
        show("Hi","Go","Bye");
        show(10,20,30,40);
    }
}
```

2. WildCard ?

3. Lower Bound

4. Upper Bound

```
static void fun(MyArray<?> obj)
{
    obj.display();
}

public static void main(String[] args)
{
    MyArray<String> ma1=new MyArray<>();
    ma1.append("Hi");
    ma1.append("Bye");

    MyArray<Integer> ma2=new MyArray<>();
    ma2.append(10);
    ma2.append(20);

    fun(ma1);
    fun(ma2);
}
```

can't use T but ? unbounded wild card

```
static void fun(MyArray<?> obj)
{
    obj.display();
}

public static void main(String[] args)
{
    MyArray<String> ma1=new MyArray<>();
    ma1.append("Hi");
    ma1.append("Bye");

    MyArray<Integer> ma2=new MyArray<>();
    ma2.append(10);
    ma2.append(20);

    fun(ma1);
    fun(ma2);
}
```

wild card with upperbound

```
static void fun(MyArray<? extends Number> obj)
{
    obj.display();
}

public static void main(String[] args)
{
    MyArray<String> ma1=new MyArray<>();
    ma1.append("Hi");
    ma1.append("Bye");

    MyArray<Integer> ma2=new MyArray<>();
    ma2.append(10);
    ma2.append(20);

    fun(ma1);
    fun(ma2);
```

super keyword (understand better with A, B , C classes)

```
static void fun(MyArray<? super Number> obj)
{
    obj.display();
}

public static void main(String[] args)
{
    MyArray<String> ma1=new MyArray<>();
    ma1.append("Hi");
    ma1.append("Bye");

    MyArray<Integer> ma2=new MyArray<>();
    ma2.append(10);
    ma2.append(20);

    fun(ma1);
    fun(ma2);
```

## 235 Do's and Don't of Generics

1. Only extends is allowed in Generic class definition
2. extends is used for interfaces also
3. extends from Only one class and multiple interfaces
4. extends and super are allowed with ? In methods
5. <?> will accept all types but cannot access
6. Base type of an Object should be same or ?

```
class A{}  
interface B{}  
interface C{  
  
class MyArray<T extends A & B>  
{  
    T A[]=(T[]) new Object[10];  
    int length =0;  
    public void append(T v)  
}
```

first one must be class

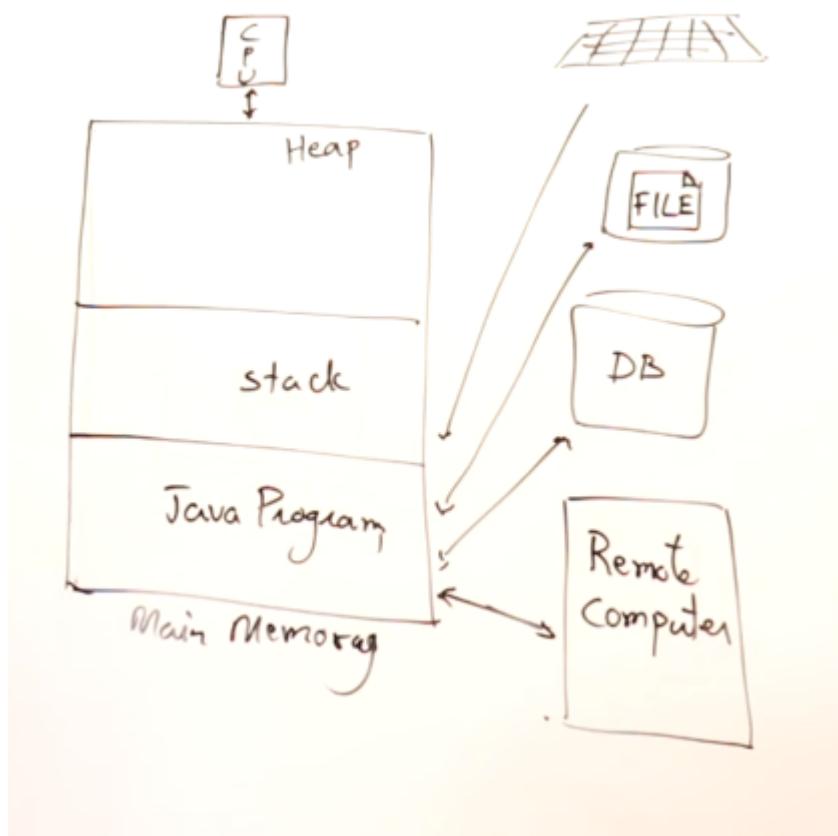
```
class A{}  
interface B{}  
interface C{}  
  
class Test extends A implements B, C  
{  
  
}  
  
class MyArray<T extends A & B & C>  
{  
    T A[]=(T[]) new Object[10];  
    int length =0;  
    public void append(T v)  
    {  
        A[length++]=v;  
    }  
  
    public void display()  
    {  
        for(int i=0;i<length;i++)  
        {  
            System.out.println(A[i]);  
        }  
    }  
}
```

```
public class GenericDemo
{
    static void fun(MyArray<?> obj)
    {
        obj.append("Hello");
    }

    public static void main(String[] args)
    {
        MyArray<String> ma1=new MyArray<>();
        fun(ma1);
    }
}
```

```
MyArray<?> ma1=new MyArray<Integer>();  
ma1.append(10);
```

## Section 25: Collection Framework

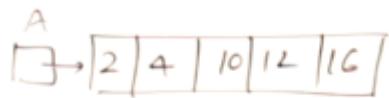


place like list of values, array of values, database, file, network, etc.

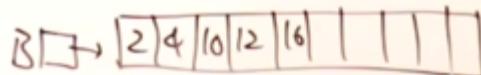
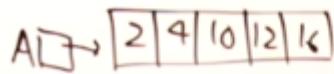
Collections Examples =list of objects of students Integers Floats Books Customers Products Accounts Movies Friends

Array is basic DataStructue. `int arr[] = new int[5]; Student std[] = new Student[10];` type of requirement we have array is not sufficient for data storing. so we have to go for the collection framework.

```
int A[] = {2, 4, 10, 12, 16};
```

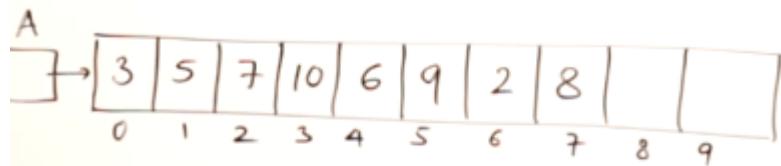


```
int B[] = new int[10];  
for(int i=0; i < A.length; i++)  
    B[i] = A[i];  
A = B;
```



Array size is fixed. You can't change the size of the array. copy and reference is changed.(this is the way)

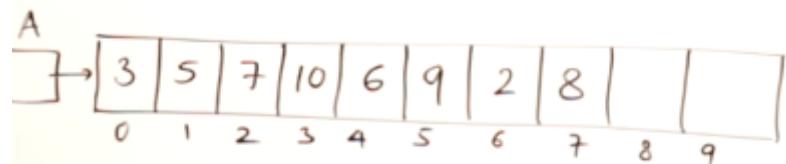
`int A[] = int A[10];`



- Variable Size Collection
- Distinct Collection
- Sorted Collection
- Insert
- Delete
- Search

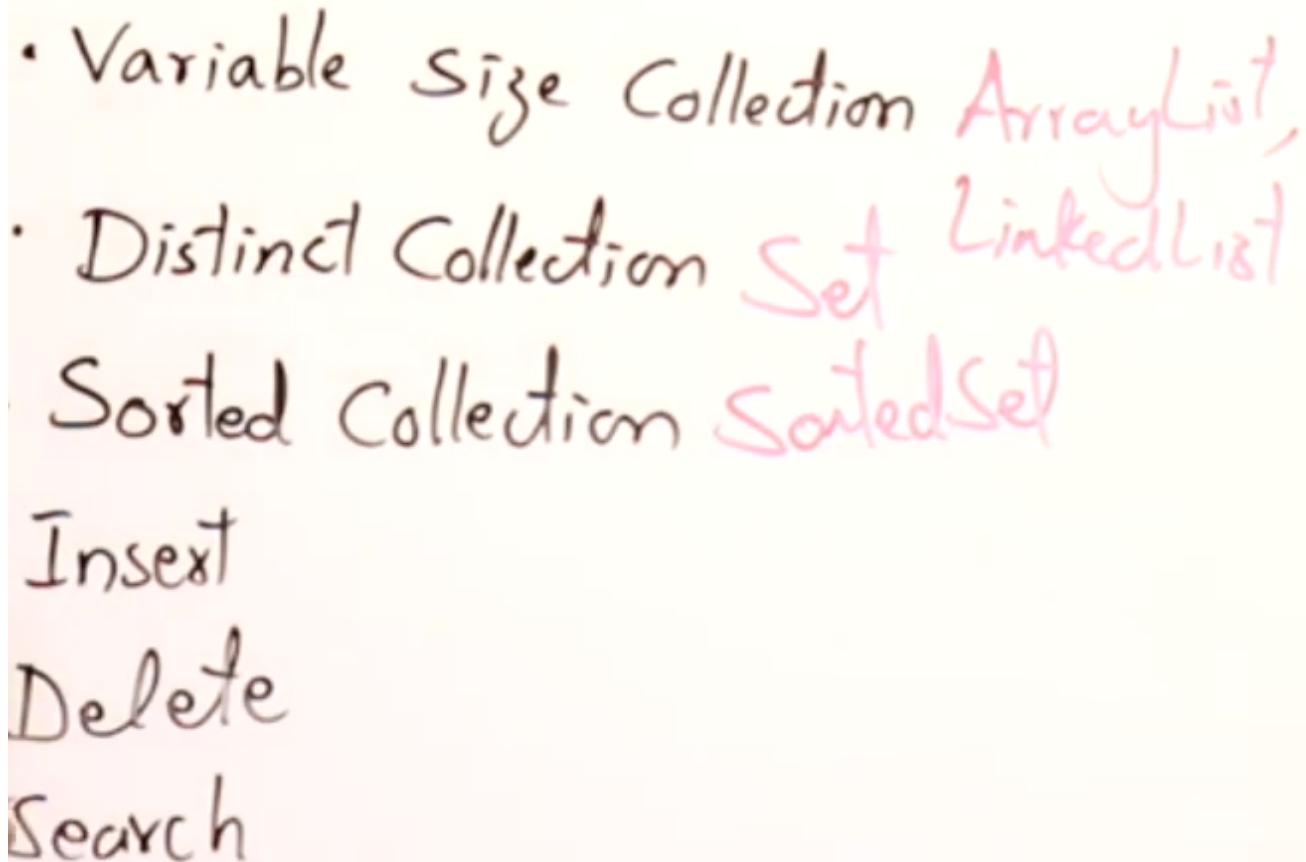
Collection FW provides ArrayList and

`int A[] = int A[10];`



- Variable Size Collection
- Distinct Collection
- Sorted Collection
- Insert
- Delete
- Search

LinkedList (variable size collection)



collection has built in linear , binary searching java provides the Hashing and sorting algorithms.

collection framework is which provides the data structures in the java java provides the collection framework in the java.util package. java provides the collection FW in the form of classes and interfaces

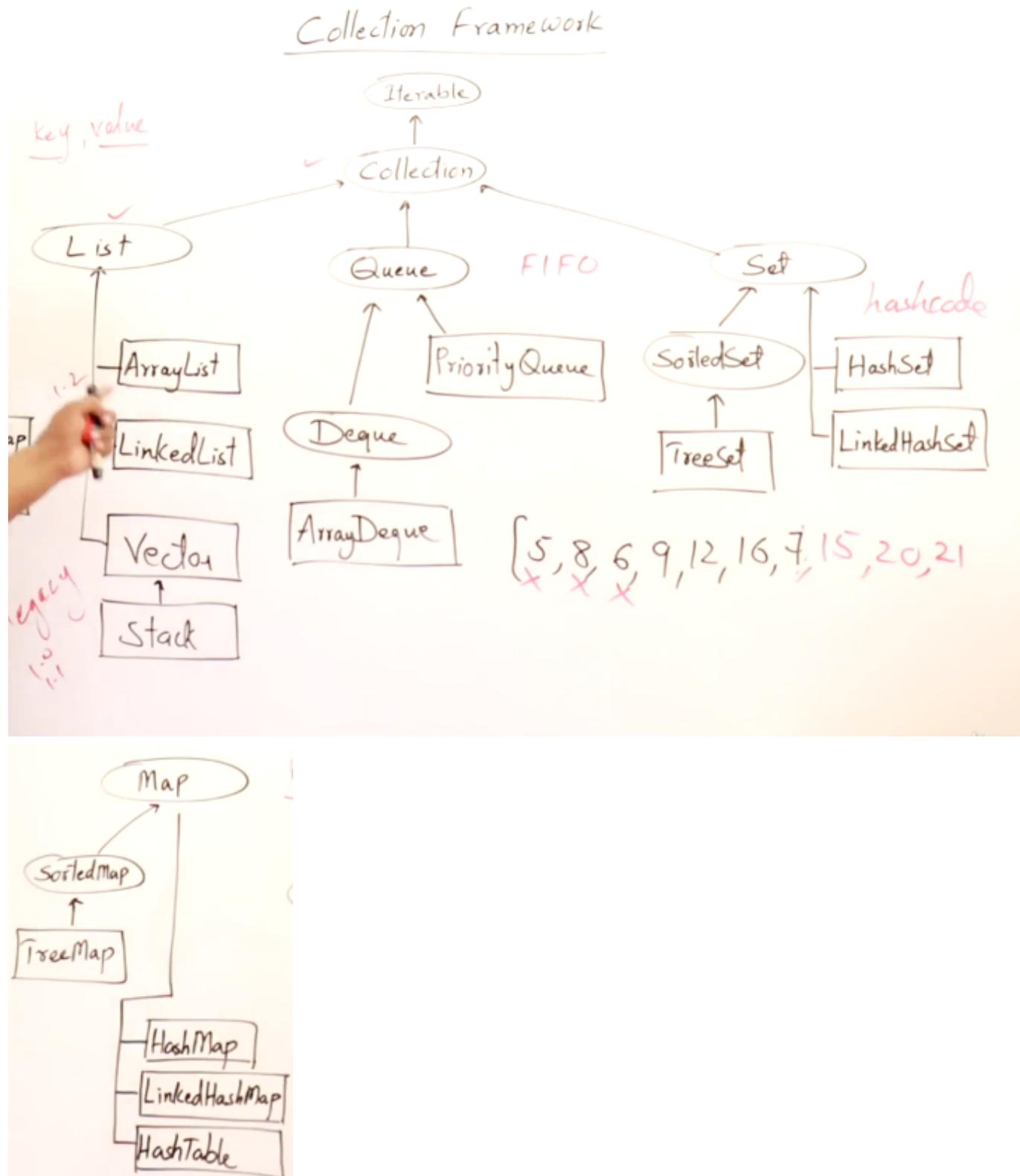
those classes and interfaces are organized in the form of hierarchy. owl boxes are interfaces rectangular boxes are the classes Iterable is the top most interface which has iterator method. Collection Interface =collection means group of elements or objects which homogenous or heterogenous. **List** means group of elements which are ordered and indexed.

- so we can insert, update, delete, search the elements.
- list can have duplicates also ArrayList (1.2) LinkedList(1.2) Vector (legacy classes)(java 1.0 or 1.1) Stack (legacy classes)(java 1.0 or 1.1)

Set is also a collection

- set will not allow the duplicates
- set is unordered and unindexed, but has the unique elements.

Queue follows FIFO mechanism



[5,2,1,4,3]

- Collection
- List
- Set
- Queue

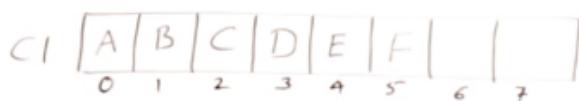
interfaces

### interface Collection

add(E e)  
addAll(Collection<E> c)  
remove(Object o)  
removeAll(Collection<E> c)  
retainAll(Collection<E> c)  
clear()  
isEmpty()  
contains(Object o)  
containsAll(Collection<E> c)  
equals(Object o)  
size()  
iterator()  
toArray()

241

## Collection Framework



interface List extends Collection

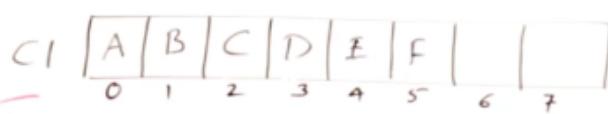
- add(int index, E e)
- addAll(int index, Collection<E> c)
- remove(int index)
- get(int index)
- set(int index, E e)
- subList(int from, int to)
- indexOf(Object o)
- lastIndexOf(Object o)
- listIterator()
- listIterator(int index)

## interface Collection

- add(E e)
- addAll(Collection<E> c)
- remove(Object o)
- removeAll(Collection<E> c)
- retainAll(Collection<E> c)
- clear()
- isEmpty()
- contains(Object o)
- containsAll(Collection<E> c)
- equals(Object o)
- size()
- iterator()
- toArray()

List has few additional methods than Collection interface Set has the same methods as the Collection interface

## Collection Framework

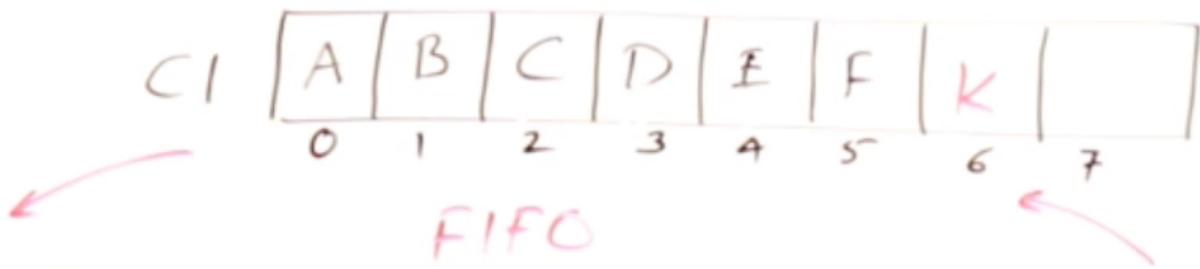


interface Queue extends Collection

- add(E e)
- poll()
- remove() throws NoSuchElementException
- peek()
- element() throws ...

## interface Collection

- add(E e)
- addAll(Collection<E> c)
- remove(Object o)
- removeAll(Collection<E> c)
- retainAll(Collection<E> c)
- clear()
- isEmpty()
- contains(Object o)
- containsAll(Collection<E> c)
- equals(Object o)
- size()
- iterator()
- toArray()



interface Queue extends Collection

- add(E e)

- poll() null

- remove() throws NoSuchElementException

- peek() null

- element() throws ...

## 242 visiting the docs

search > java 13 util list of interfaces below are classes and exceptions

## 243 ArrayList & Iterator

```
package listdemo;

import java.util.*;

public class ListDemo {

    public static void main(String[] args) {
        ArrayList<Integer> al1=new ArrayList<>(20);
        ArrayList<Integer> al2=new ArrayList<>(List.of(50,60,70,80,90));
        al1.add(10);
    }
}
```

```
al1.add(0,5);
//al1.addAll(al2);
al1.addAll(1,al2);
al1.add(5,70);
al1.set(6,100);

//al1.forEach(n->System.out.println(n));
//al1.forEach(System.out::println);
al1.forEach(n->show(n));

//System.out.println(al1.contains(25));
//System.out.println(al1.get(5));
//System.out.println(al1.indexOf(70));
//System.out.println(al1.lastIndexOf(70));

//System.out.println(al1);

/*for(int i=0;i<al1.size();i++)
    System.out.println(al1.get(i));*/

/*for(Integer x:al1)
    System.out.println(x);*/

/*for(Iterator<Integer> it= al1.iterator(); it.hasNext();)
{
    java.lang.Integer x = it.next();
    System.out.println(x);
}*/



/*al1.forEach((x)->{
    System.out.println(x);
});*/



/*for(Iterator<Integer> it=al1.listIterator();it.hasNext();)
while(it.hasNext())
{
    System.out.println(it.next());
}*/



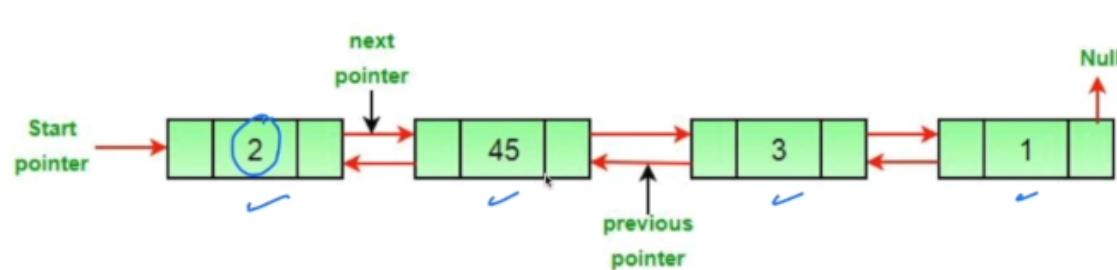
}

static void show(int n)
{
    if(n>60)
        System.out.println(n);
}

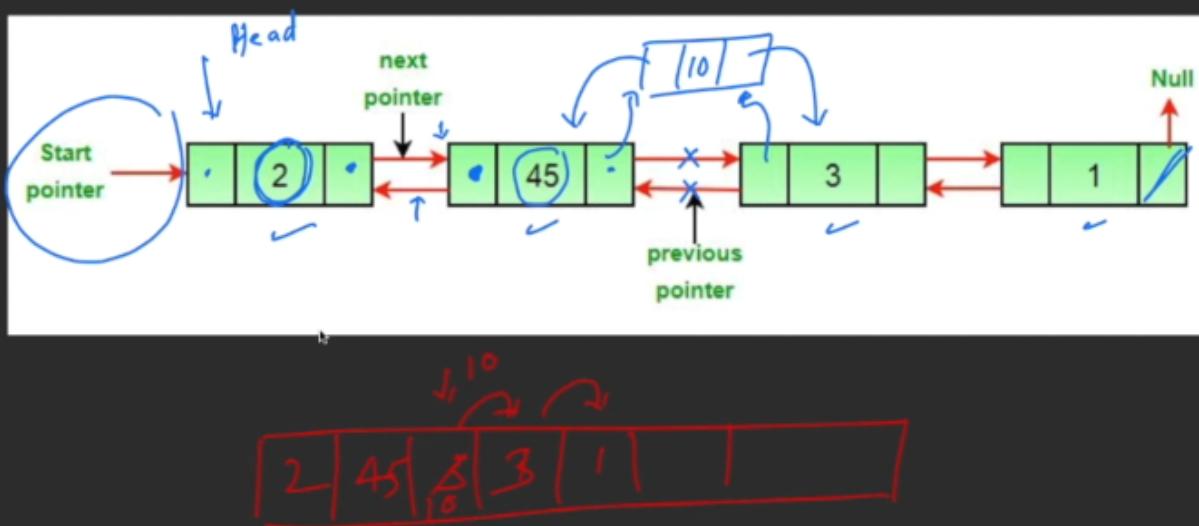
}
```

## 244 LinkedList

all the elements should be of same type



most flexible of inserting and deleting at any position



eg. if there are 4000 elem's or more like that which used the doubly linked list (in java)

- linkedlist takes more space than the arraylist
- no need to mention size in the linkedlist

.addFirst() .addLast()

```

package listdemo;

import java.util.*;

public class ListDemo {

    public static void main(String[] args) {
        ArrayList<Integer> al1=new ArrayList<>(20);
        ArrayList<Integer> al2=new ArrayList<>(List.of(50,60,70,80,90));
        al1.add(10);
        al1.add(0,5);
    }
}
  
```

```
//al1.addAll(al2);
al1.addAll(1,al2);
al1.add(5,70);
al1.set(6,100);

//al1.forEach(n->System.out.println(n));
//al1.forEach(System.out::println);
al1.forEach(n->show(n));

//System.out.println(al1.contains(25));
//System.out.println(al1.get(5));
//System.out.println(al1.indexOf(70));
//System.out.println(al1.lastIndexOf(70));

//System.out.println(al1);

/*for(int i=0;i<al1.size();i++)
    System.out.println(al1.get(i));*/

/*for(Integer x:al1)
    System.out.println(x);*/

/*for(Iterator<Integer> it= al1.iterator(); it.hasNext();)
{
    java.lang.Integer x = it.next();
    System.out.println(x);
}*/



/*al1.forEach((x)->{
    System.out.println(x);
});*/



/*for(Iterator<Integer> it=al1.listIterator();it.hasNext();)
while(it.hasNext())
{
    System.out.println(it.next());
}*/



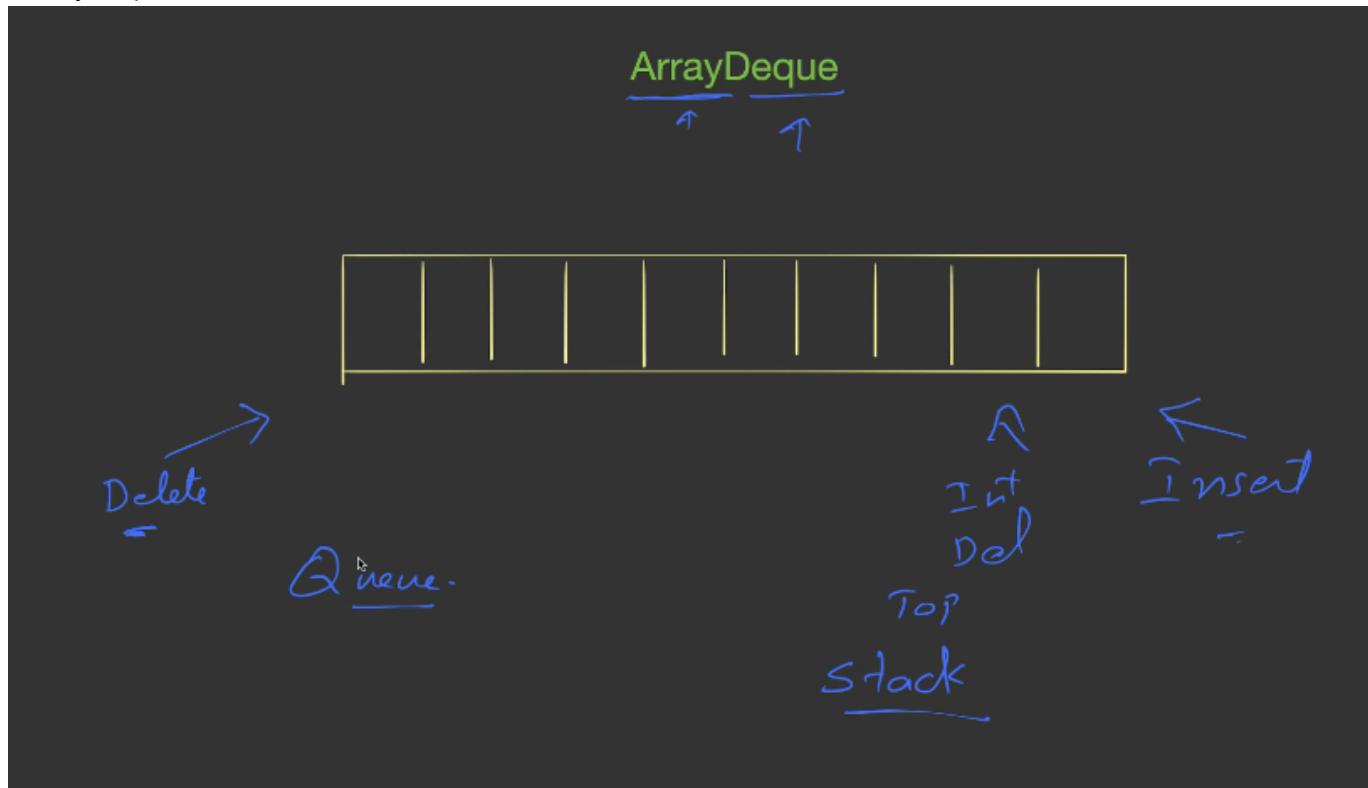
}

static void show(int n)
{
    if(n>60)
        System.out.println(n);
}

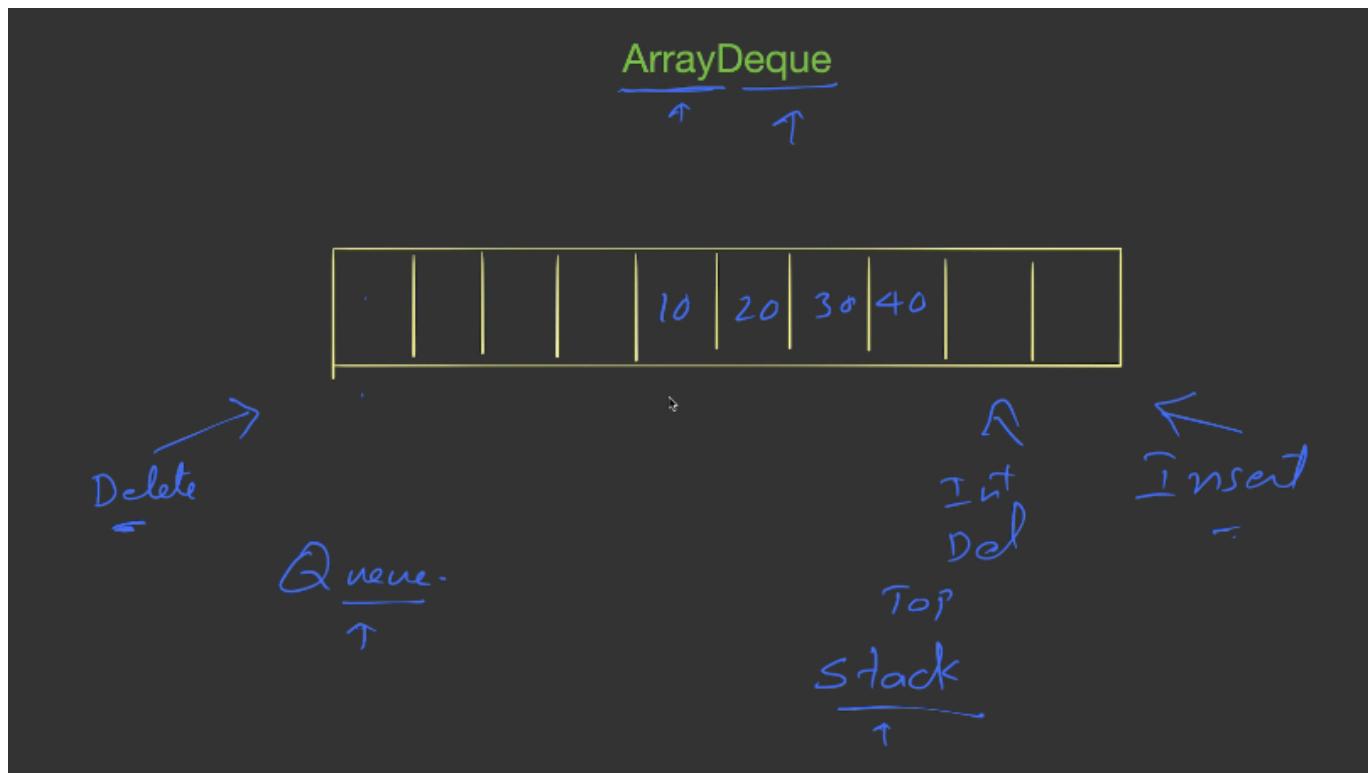
}
```

## 245 ArrayDeque

=arrayDeque follows Double ended Queue data structure



starts from the middle insertion and deletion



if array size is full, then it will internally grows the size

all the elements follows constant time complexity inserting from last, deleting at first is known as the queue.

inserting first,deleting first is known as the stack.

inserting first and deleting last, then it is a queue.

(if u need stack or queue then use arrayDeque)

```

package dequeudemo;

import java.util.*;

public class DequeDemo
{
    public static void main(String[] args)
    {
        ArrayDeque<Integer> dq=new ArrayDeque<>();

        dq.offerLast(10);
        dq.offerLast(20);
        dq.offerLast(30);
        dq.offerLast(40);

        dq.pollFirst();

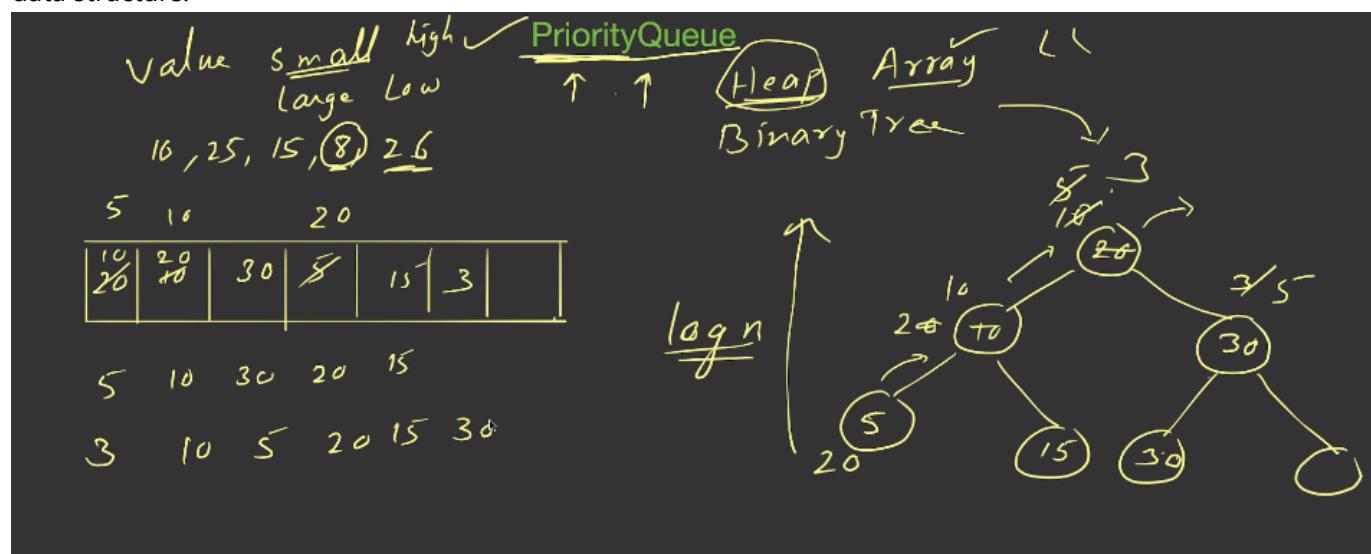
        dq.offerLast(1);
        dq.offerLast(2);
        dq.offerLast(3);
        dq.offerLast(4);

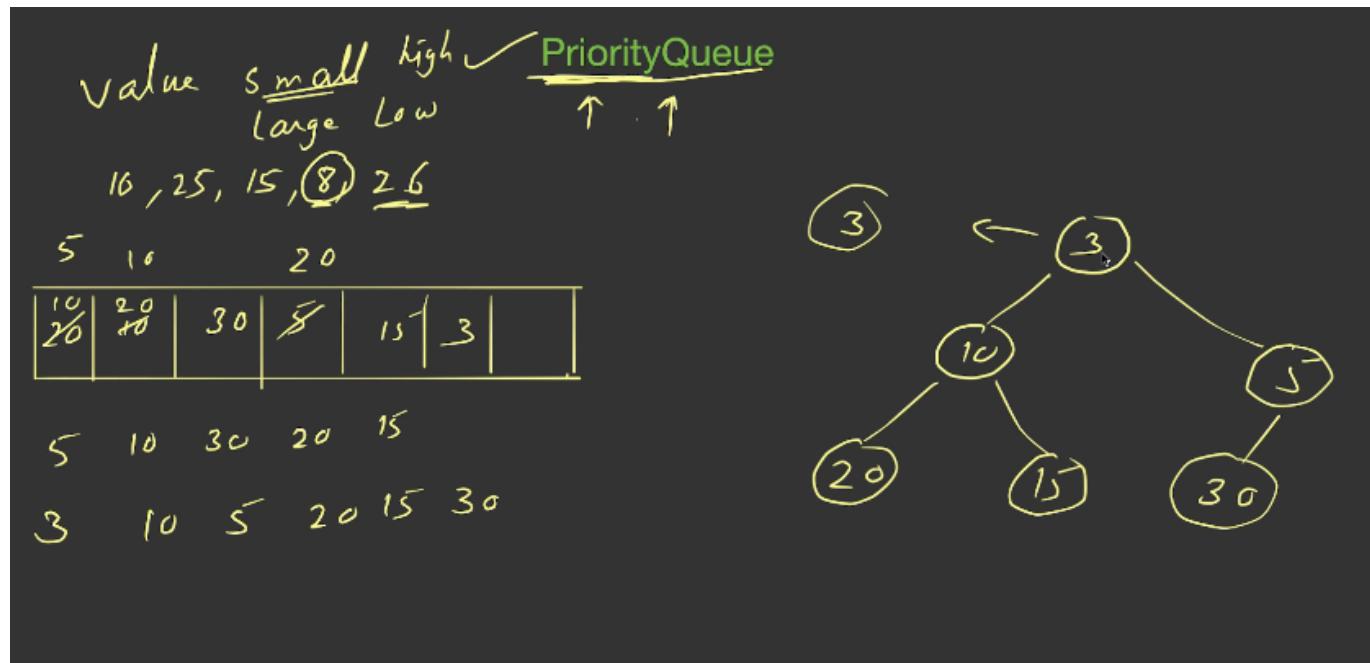
        dq.forEach((x)->System.out.println(x));
    }
}

```

## 246 Priority Queue

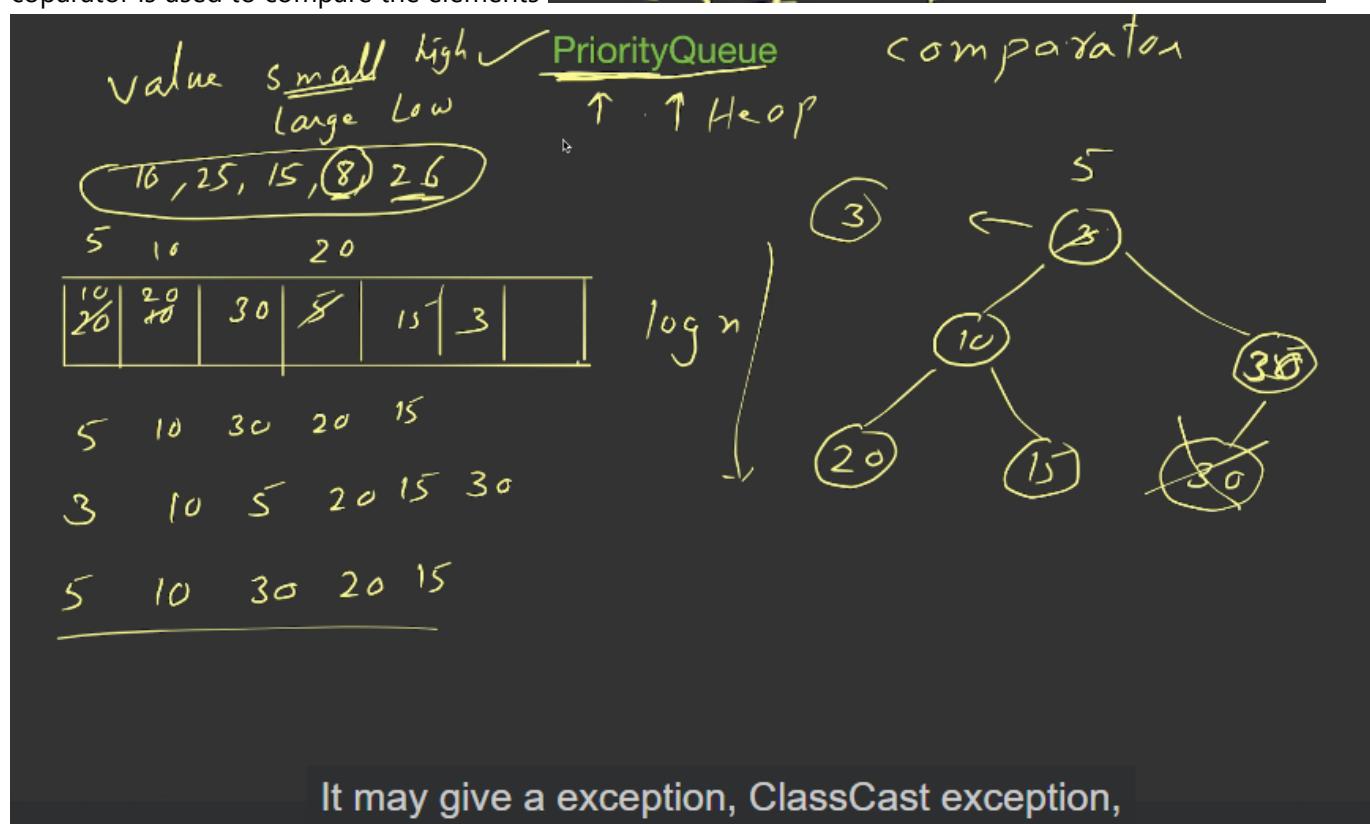
=Priority Queue is a queue which follows the priority order. priority queue is implemented using the heap data structure.





small high ✓  
large low

comparator is used to compare the elements



```
package prioritydemo;

import java.util.*;
```

```
class MyCom implements Comparator<Integer>
{
    public int compare(Integer o1,Integer o2)
    {
        if(o1<o2) return 1;
        if(o1>o2) return -1;
        return 0;
    }
}

public class PriorityDemo
{
    public static void main(String[] args)
    {
        PriorityQueue<Integer> p=new PriorityQueue<>(new MyCom());

        p.add(20);
        p.add(10);
        p.add(30);
        p.add(5);
        p.add(15);
        p.add(3);

        System.out.println(p.peek());

        p.forEach((x)->System.out.println(x));

        p.poll();

        System.out.println("After Deletion");
        p.forEach((x)->System.out.println(x));
    }
}
```

comparator for max-heap

```
import java.util.*;

class MyCom implements Comparator<Integer>
{
    public int compare(Integer o1, Integer o2)
    {
        if(o1<o2) return 1;
        if(o1>o2) return -1;
        return 0;
    }

public class PriorityDemo
{
    public static void main(String[] args)
    {
        PriorityQueue<Integer> p=new PriorityQueue<>();
PriorityQueue<Integer> p=new PriorityQueue<>(new MyComp());
```

247 hashing Technique

$$h(x) = x \% 10$$

$$\lambda = 0.75$$

HashSet  
 HashMap  
 LinkedHashSet  
 LinkedHashMap  
 Hashtable  
 properties

|   |  |
|---|--|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |

## Hashing

|   |  |
|---|--|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |

$$h(x) = x \% 10$$

$$\lambda = 0.75$$

Key  
Value

15

28

42

35

17

52

|   |           |
|---|-----------|
| 0 | Key value |
| 1 |           |
| 2 |           |
| 3 |           |
| 4 |           |
| 5 |           |
| 6 | >         |
| 7 |           |
| 8 |           |
| 9 |           |

## Hashing

Hash Table  
 size 16

$$\underline{h(x) = x \% 10}$$

$$\lambda = 0.75$$

Value  
Key

15

28.

42

35

17

52

|   | key value |
|---|-----------|
| 0 |           |
| 1 |           |
| 2 | 42        |
| 3 |           |
| 4 |           |
| 5 | 15        |
| 6 | 35        |
| 7 |           |
| 8 | 28        |
| 9 |           |

## Hashing

|   |  |
|---|--|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |

Hash Table

size 16

a little bit of sequential search for reaching a value.

$$\underline{h(x) = x \% 10}$$

$$\lambda = 0.75$$

Value  
Key

15

28.

42

35

17

52

45

|   | key value |
|---|-----------|
| 0 |           |
| 1 |           |
| 2 | 42        |
| 3 |           |
| 4 |           |
| 5 | 15        |
| 6 | 35        |
| 7 | —         |
| 8 | 28        |
| 9 |           |

## Hashing

|   |  |
|---|--|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |

Hash Table

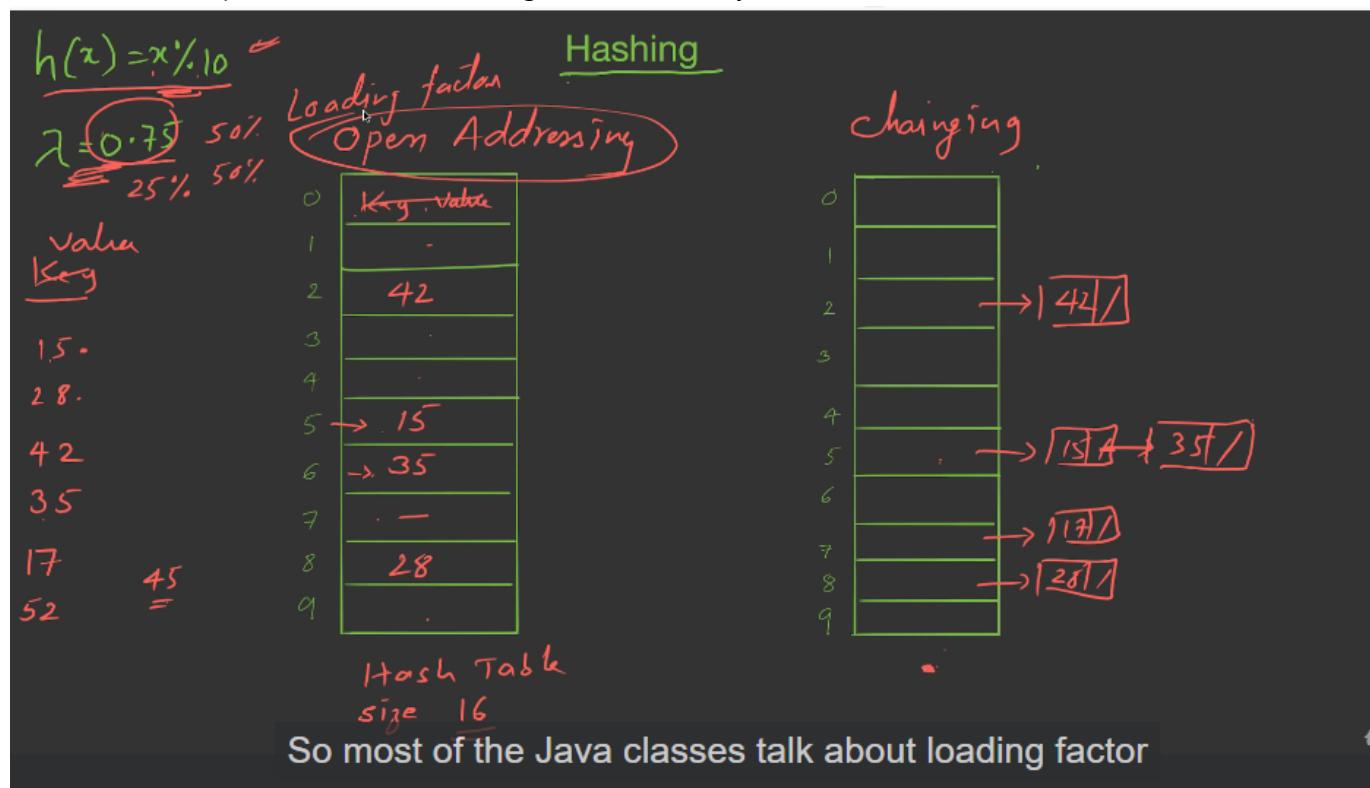
size 16

Check the next location, next location is vacant, blank.

$$\lambda = \underline{0.75} \text{ so } 50\% \\ 25\% \text{ blank}$$

25% blank is called as the load factor

combination of open address and chaining it used internally.



initial capacity is 16

## HashSet

set means it doesn't allow duplicates. hash means it uses the hash table.

- it takes the constant time.
- by default loadfactor is 0.75
- you can mention 0.5 if you want.

```
public class SetDemo
{
    public static void main(String[] args)
    {
        HashSet<Integer> hs=new HashSet<>(20,0.25f);

        hs.add(10);
        hs.add(20);
        hs.add(30);
        hs.add(10);

        System.out.println(hs);
    }
}
```

- if you need to add and remove elements in the constant time then you can take it.

```
package setdemo;

import java.util.*;

public class SetDemo {

    public static void main(String[] args) {

        HashSet<Integer> hs=new HashSet<>(20,0.75f);

        hs.add(10);
        hs.add(20);
        hs.add(30);
        hs.add(10);

        System.out.println(hs);
    }
}
```

## 249. TreeSet.

tree is a datastructure. there are various types of data structures. like BST, AVL tree, red trees, black trees, play trees.

This implementation provides guaranteed  $\log(n)$  time cost for the basic operations (add, remove and search).

BST are good at searching  $\log(n)$  times

```
import java.util.*;

public class SetDemo
{
    public static void main(String[] args)
    {
        TreeSet<Integer> ts=new TreeSet<>(List.of(10,30,50,70,10,40));

        ts.add(25);

        System.out.println(ts.ceiling(55));
        System.out.println(ts);
    }
}
```

```
public class SetDemo
{
    public static void main(String[] args)
    {
        SortedSet<Integer> ts=new TreeSet<>(List.of(10,30,50,70,10,40));
        ts.add(25);

        System.out.println(ts);
    }
}
```

```
package setdemo1;

import java.util.*;

public class SetDemo1 {

    public static void main(String[] args) {
        TreeSet<Integer> ts=new TreeSet<>(List.of(10,30,50,70,10,40));
        ts.add(25);

        //ts.ceiling(55);

        System.out.println(ts);
    }
}
```

## 250 Comparable Interface

```
package setdemo2;

import java.util.*;

class Point implements Comparable
{
    int x;
    int y;
    public Point(int x,int y)
    {
        this.x=x;
        this.y=y;
    }
}
```

```
public String toString()
{
    return "x="+x+"y="+y;
}
public int compareTo(Object o)
{
    Point p=(Point)o;
    if(this.x<p.x)
        return -1;
    else if(this.x>p.x)
        return 1;
    else
    {
        if(this.y<p.y)
            return -1;
        else if(this.y>p.y)
            return 1;
        else
            return 0;
    }
}

public class SetDemo2 {

    public static void main(String[] args) {

        TreeSet<Point> ts=new TreeSet<>();

        ts.add(new Point(1,1));
        ts.add(new Point(5,5));
        ts.add(new Point(5,2));

        System.out.println(ts);
    }
}
```

250 to 260 pause

\*\*\*\*\*

## Section 26 : Date and Time API

## 261 Deprecated Date Class

## Old Date and Time Classes

java.util ✓

java.time — 8

1. Date Representation
2. Date Class
3. Calendar Class
4. TimeZone Class

date is represent in long from 1 jan 1970 (starting time) (starting calendar is 1900)

```
package datedemo;

import java.util.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        System.out.println(System.currentTimeMillis());
    }
}
```

```
Output - DateDemo
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name:init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/bu...
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/
compile:
run:
1596012315992
```

s. cms() / 1000 / 60 / 60 / 24 / 365

```
import java.util.*; //old date class
class Main{
    public static void main(String args[]){
        System.out.println(System.currentTimeMillis()/1000/60/60/24/365.25);

        Date d = new Date();
    }
}
```

```

        System.out.println(d);

                // MM/DD/YYYY
        Date sg = new Date("09/09/2001");
        System.out.println(sg);

        System.out.println((d.getTime() - sg.getTime()) / 1000 / 60 / 60 / 24 / 365.25);

        System.out.println(d.getYear() + 1900);
    }
}

```

## 262 Calendar and Time Zone

java provides the abstract class called the calendar solar calendar, lunar calendar, jewish calendar, arab calendar, japanese calendar globally gregorian calendar is used. isLeapYear();

```

package datedemo;

import java.util.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        GregorianCalendar gc=new GregorianCalendar(
            System.out.println(gc.get(Calendar.YEAR));
    }
}

```

```

ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-deps-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
6
BUILD SUCCESSFUL (total time: 0 seconds)

```

gc.get(Calendar.YEAR); //and a lot of other

//wikipedia | offset search > timezones

list of timezones list of tz timezones for names

tz.getDisplayName(); tz.getID();

gc.setTimeZone(TimeZone.getTimeZone("America/Los\_Angeles")); above all are present in the java 7 java 8 gave the separate API java.time

## 263 Joda Time API

## Joda Date and Time API

✓ 1. Why new API

→ 2. LocalDate

→ 3. LocalTime

→ 4. LocalDateTime

## Joda Date and Time API



Date d = new Date();

1. millis 1, Jan 1970

2. Data & Time

3. d.sel

✓ 1. Why new API

→ 2. LocalDate ✓

→ 3. LocalTime ✓

→ 4. LocalDateTime ✓

↑  
Seconds, nano seconds

## Joda Date and Time API



Date d = new Date(); ✓

1. millis 1, Jan 1970 ←

2. Data & Time

3. d.sel

✓ 1. Why new API

✗ → 2. LocalDate ✓

→ 3. LocalTime ✓

→ 4. LocalDateTime ✓

↑  
Seconds, nano seconds

```
package datedemo;

import java.util.*;
import java.time.*;
import java.time.temporal.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        Date d=new Date();
        System.out.println(d);
    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/dep
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/bui
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/bui
compile:
run:
Wed Jul 29 19:36:25 IST 2020
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
import java.util.*;
import java.time.*;
import java.time.temporal.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        Date d=new Date();
        d.setHours(21);
        System.out.println(d);

        LocalDate dt=new LocalDate();
    }
}
```

```
15     LocalDate dt=LocalDate.now();
16     System.out.println(dt);
17 }
18 }
19 }
20 }
21 }
22 }

init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDem
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/Date
Note: /Users/abdulbari/NetBeansProjects/DateDemo/src/datedemo/Dat
Note: Recompile with -Xlint:deprecation for details.
compile:
run:
Wed Jul 29 21:38:55 IST 2020
2020-07-29
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
12     LocalDate d=LocalDate.now(Clock.systemDefaultZone());
13     System.out.println(d);
14 }
15 }
16 }
17 }
18 }
19 }

Output - DateDemo (run)
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
2020-07-29
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1     LocalDate d=LocalDate.now(ZoneId.of("Asia/Kolkata"));
2     System.out.println(d);
```

```
LocalDate d=LocalDate.of(2020, Month.MARCH, 10);
System.out.println(d);
```

```
LocalDate d=LocalDate.ofEpochDay(1);
System.out.println(d);

}
```

```
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
1970-01-02
```

```
LocalDate d=LocalDate.parse("2020-01-03");
System.out.println(d);
```

```
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
2020-01-03
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
LocalDate d=LocalDate.parse("2020-01-03");
d.plusMonths(6);
System.out.println(d);
```

take in new obj

```
LocalDate d=LocalDate.parse("2020-01-03");
LocalDate d1=d.plusMonths(6);
System.out.println(d1);
```

```
13     LocalTime t=LocalTime.now();
14     System.out.println(t);
15     LocalTime t1=t.minusHours(3);
16     System.out.println(t1);
17
18     LocalDateTime dt=LocalDateTime.now();
19     System.out.println(dt);
20
21 }
22 }
```

```
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.ac
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jardeps.jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/compile:
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo
compile:
run:
19:58:41.752
16:58:41.752
2020-07-29T19:58:41.752
```

## java.time Class

time zone is not present in LocalDateTime classes

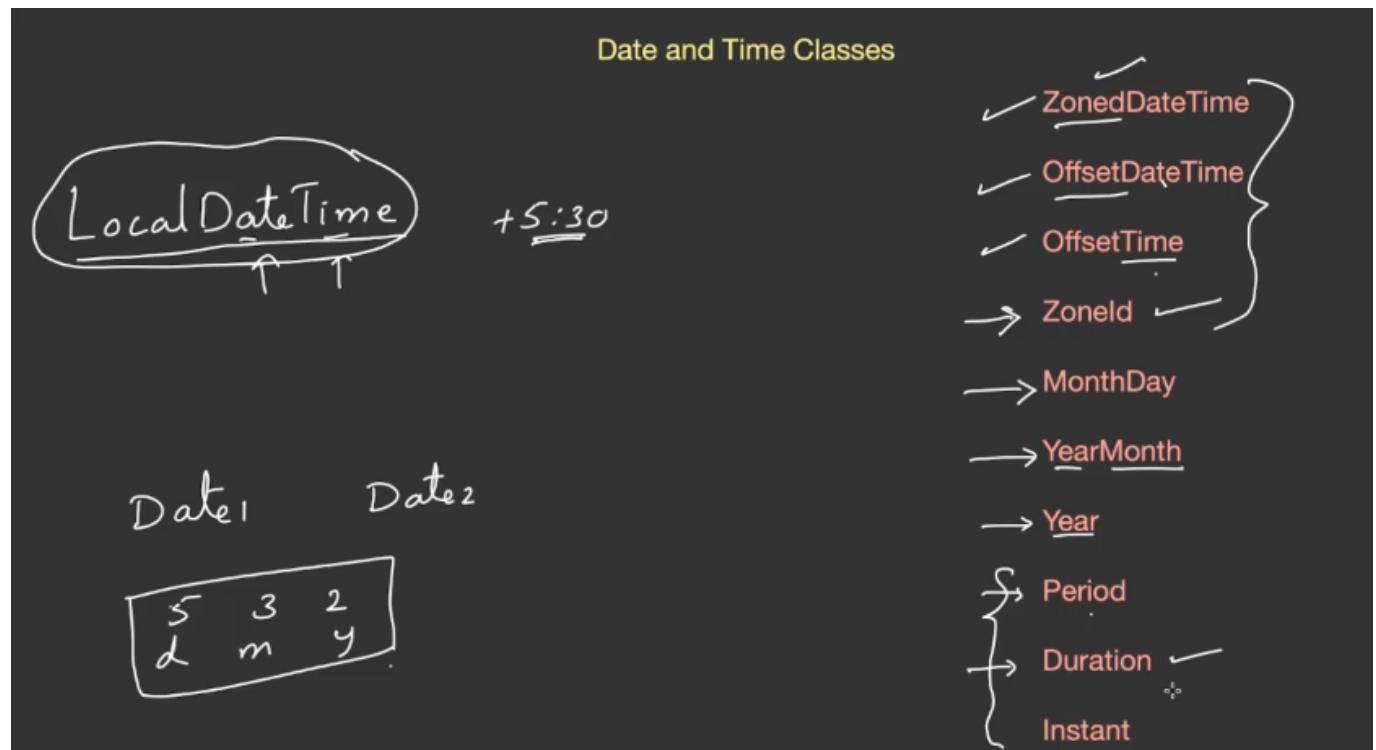
### Date and Time Classes



```
package datedemo;

import java.util.*;
import java.time.*;
import java.time.temporal.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        ZonedDateTime zdt=ZonedDateTime.now(ZoneId.of("America/Los_Angeles"));
        System.out.println(zdt);
        MonthDay md=MonthDay.now();
    }
}
```



```

import java.util.*;
import java.time.*;
import java.time.temporal.*;

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        ZonedDateTime zdt=ZonedDateTime.now(ZoneId.of("America/Los_Angeles"));
        System.out.println(zdt);
        MonthDay md=MonthDay.now();

        Period p=Period.of(2, 2, 10);
        System.out.println(p.addTo(LocalDate.now()));

    }
}

```

Output - DateDemo (run)

```

ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
2020-07-30T04:51:27.796-07:00[America/Los_Angeles]
2022-10-10
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        ZonedDateTime zdt=ZonedDateTime.now(ZoneId.of("America/Los_Angeles"));
        System.out.println(zdt);
        MonthDay md=MonthDay.now();

        Period p=Period.of(2, 2, 10);
        System.out.println(p.addTo(LocalDate.now()));

    }
}

```

Output - DateDemo (run)

```

ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
compile:
run:
2020-07-30T04:52:30.811-07:00[America/Los_Angeles]
2022-10-10
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

20
21     Instant i=Instant.now();
22

ant -f /Users/abdulbari/NetBeansProjects/DateDemo -
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDem
deps-jar:
Updating property file: /Users/abdulbari/NetBeansPr
Compiling 1 source file to /Users/abdulbari/NetBean
compile:
run:
2020-07-30T04:53:34.181-07:00[America/Los_Angeles]
2022-10-10
2020-07-30T11:53:34.232Z
BUILD SUCCESSFUL (total time: 0 seconds)
instant

```

## 265 Date formatter

```
strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        LocalDateTime dt=LocalDateTime.now();

        DateTimeFormatter df=DateTimeFormatter.ofPattern("dd/MM/yyyy");

        System.out.println(df.format(dt));
    }
}
```

```
Output - DateDemo (run)
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
30/07/2020;
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        ZonedDateTime dt=ZonedDateTime.now();

        DateTimeFormatter df=DateTimeFormatter.ofPattern("dd/MM/yyyy hh:mm:ss z Z");

        System.out.println(df.format(dt));
    }
}

Output - DateDemo (run)
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
30/07/2020 07:11:34 IST +0530
BUILD SUCCESSFUL (total time: 0 seconds)

strictfp public class DateDemo
{
    public static void main(String[] args)
    {
        LocalDateTime dt=LocalDateTime.now();

        System.out.println(dt.get(ChronoField.DAY_OF_MONTH));
    }
}

Output - DateDemo (run)
ant -f /Users/abdulbari/NetBeansProjects/DateDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/DateDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/DateDemo/build/classes
compile:
run:
chronos fields 30
```

