

DAY-1

Java Full Stack Development Batch 2025

What is Full Stack Development?

->it refers to the practice of **building and maintaining** all layers of software from front end to backend

- FRONT END
- BACKEND

PURPOSE: We can create an Enterprise Application

BENEFITS: Freelancing OR JOB READY

Application Development

- 1.Monolithic Architecture
- 2.Microservices Architecture

FORNT END DEVELOPMENT

->Focuses on the UI and UX

TECH: HMTL CSS JS BOOTSTRAP (20 DAYS)

Terms:

->Front End

->UI/UX

->CLIENT SIDE

->VIEW

->WEB INTERFACE

UI: Proper design, Clear Input, Color...etc

UX: Navigation, Error

BACK END DEVELOPMENT

It refers to server side of an application which handles USER data and process it.

Database layer is also comes under backend

TECH:

1.JAVA

- **CORE JAVA**
- **ADVANCED JAVA**

2.Frameworks

- **Spring**
- **Hibernate**

3.Tools:

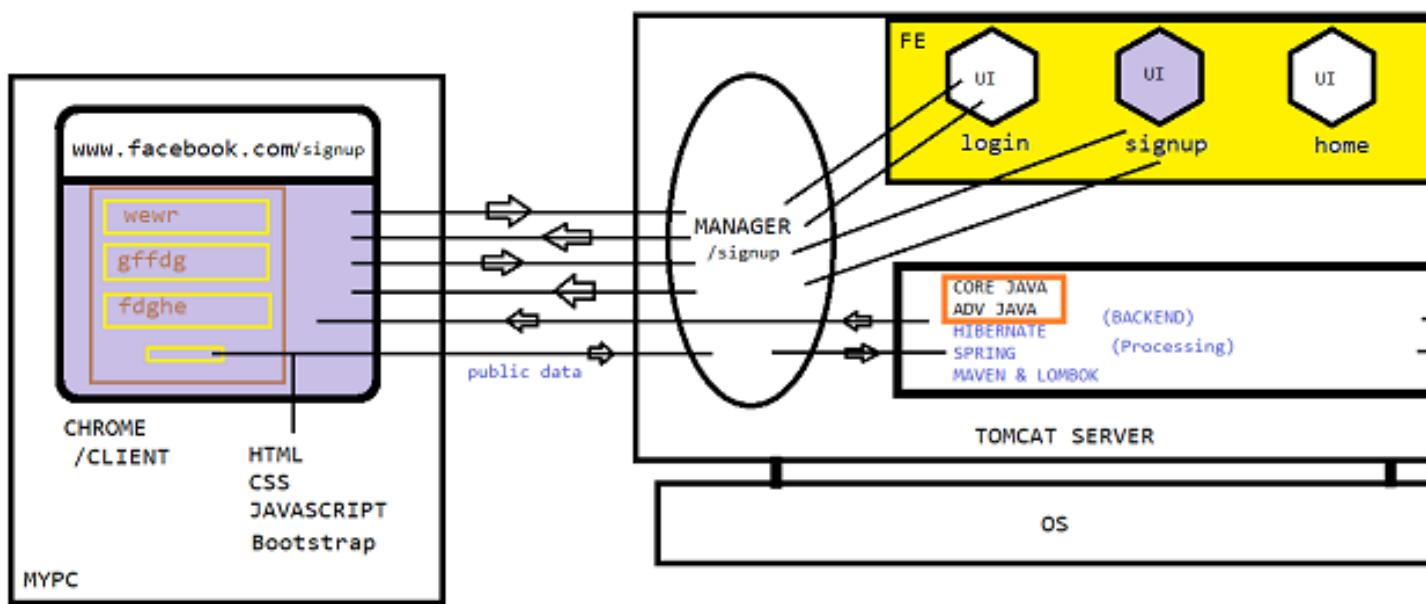
- Maven

4.Library:

- LOMBOK

DATABASE: Storage unit:

- SQL(20DAYS)
- MYSQL/ORACLE



DAY-2

Java Full Stack Development Batch 2025

Java Programming Language

Java is a **versatile, high level, object oriented programming language** designed for **ease of use, reliability** and **platform independence**.

It was developed by James Gosling and his team at SUN MICROSYSTEM (Later acquired by ORACLE CO.) and released in 1995.

What is Programming Language?

->It is a **set of instructions** that allows humans to communicate with computers. It serves as a bridge between human and machine.

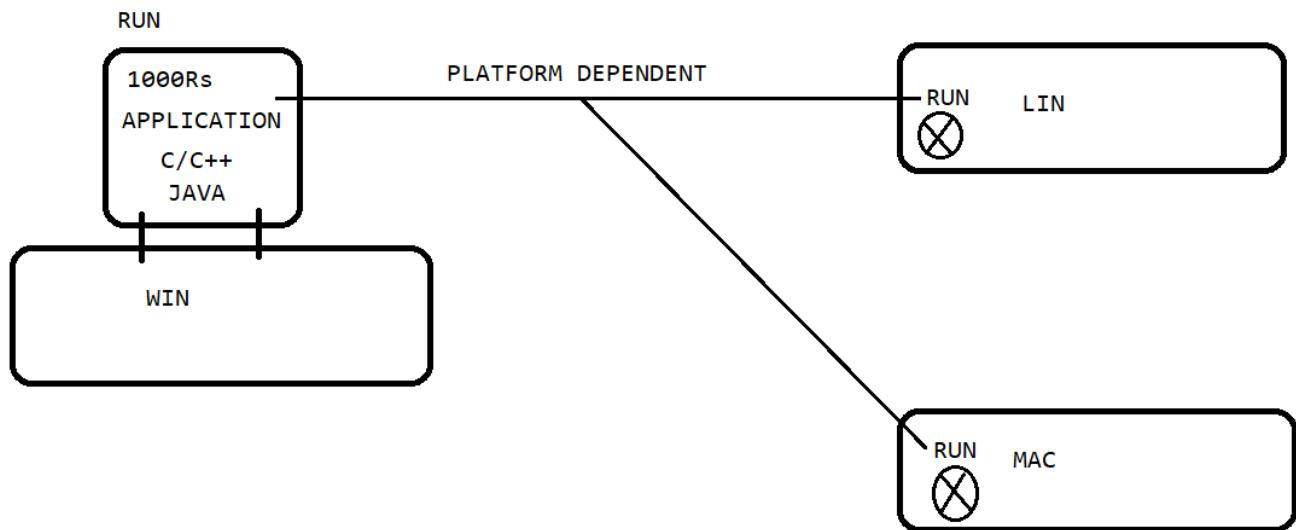
What is high level programming language?

->A high level programming language is a **type of programming** that is designed to be easy for human to read, write and understand.

- ENGLISH LIKE STATEMENTS (A-Z a-z)

- Number (0-9)
- Special Symbols: + - / * % ..etc

What is platform independent?



->Platform independence means a program or software can run on **diff diff platform (OS)** without any modification

What is Versatile?

- Able to **adapt in diff diff situations (PLATFORM IND)**
- **WORA(WRITE ONCE RUN ANYWHERE)**
- WEB DEV, MOBILE APP, BIG DATA, TESTING...etc

What is OOP's

->it is just a way of programming

What is Reliability?

- Consistently and accurately over time without failure
- Stable, Errors Handle, Memory Manage, WORA, Security

INSTALLATION

Java is higher to Lower Compatible Language

Java 17 (Installation) LTS (ORACLE: Long Term Supported)

<https://www.oracle.com/java/technologies/java/jdk17-archive-downloads.html>

IDE(Integrated Development Environment) :

Eclipse IDE

Eclipse IDE for Enterprise Java and Web Developers

544 MB 102,745 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.



[Click here](#) to raise an issue with the Eclipse Web Tools Platform. Maintainers will move opened issues to the right place.

[Click here](#) to raise an issue with the Eclipse Platform.
[Click here](#) to raise an issue with Maven integration for web projects.

[Click here](#) to raise an issue with Eclipse Wild Web Developer (incubating).



[Windows | x86_64 | AArch64](#)

[macOS x86_64 | AArch64](#)

[Linux x86_64 | AArch64 | riscv64](#)

DAY-3

Java Full Stack Development Batch 2025

How to write first java program?

Compiler

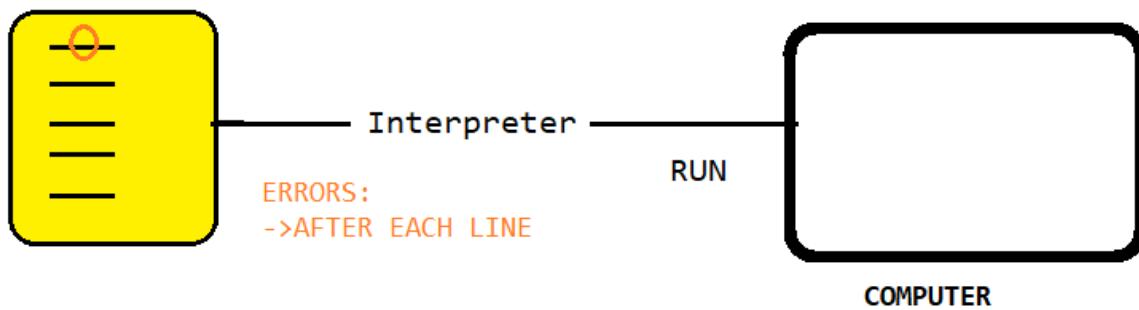
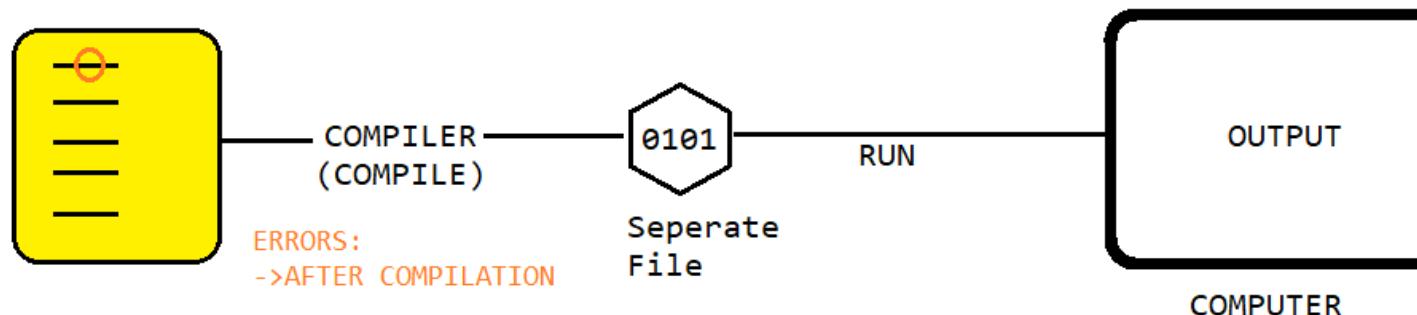
It is a program that translates **high-level source code** written in a programming language into **machine code**.

AT ONCE:

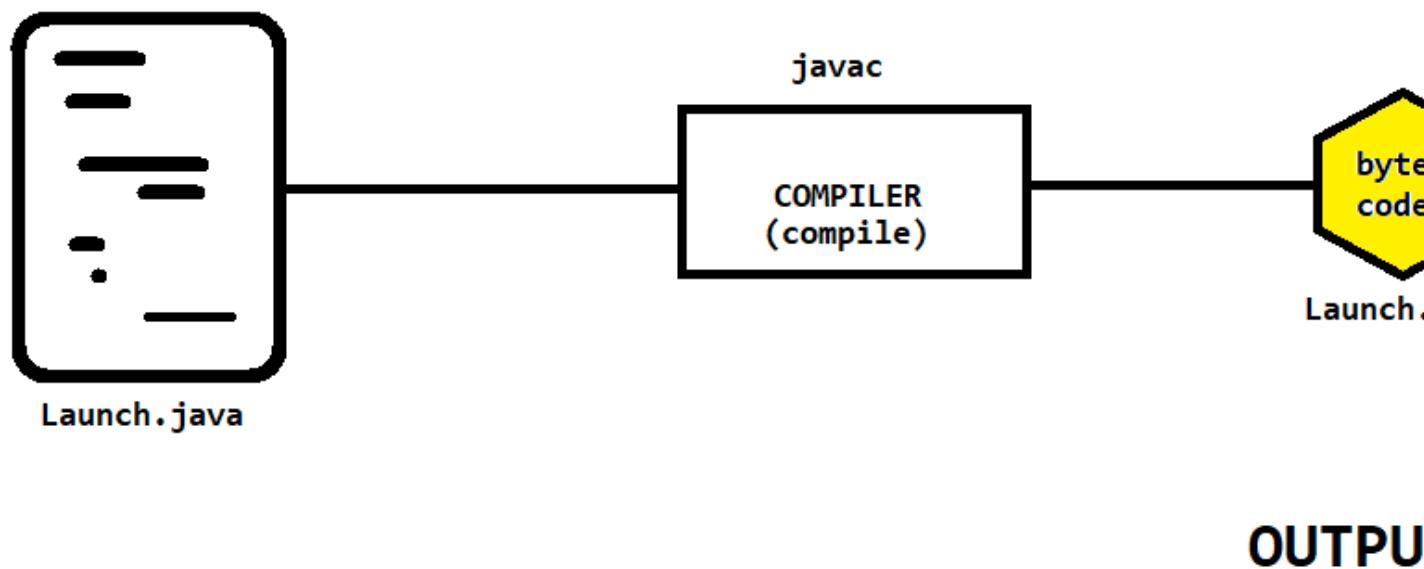
Interpreter

It is a program that translates **high-level source code** written in a programming language into **machine code**.

LINE BY LINE:



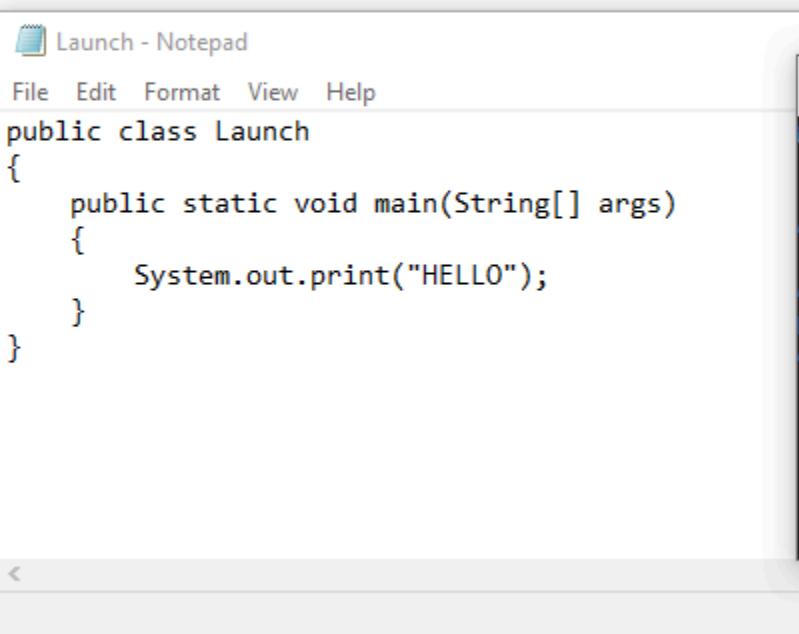
- NOTE: JAVA USES BOTH COMPILER AND INTERPRETER
- DOWNLOAD:- WE HAVE DOWNLOADED **JDK(JAVA DEVELOPMENT KIT)**
 - **COMPILER**
 - **INTERPRETER**
 - PATH: C:\Program Files\Java\jdk-17\bin



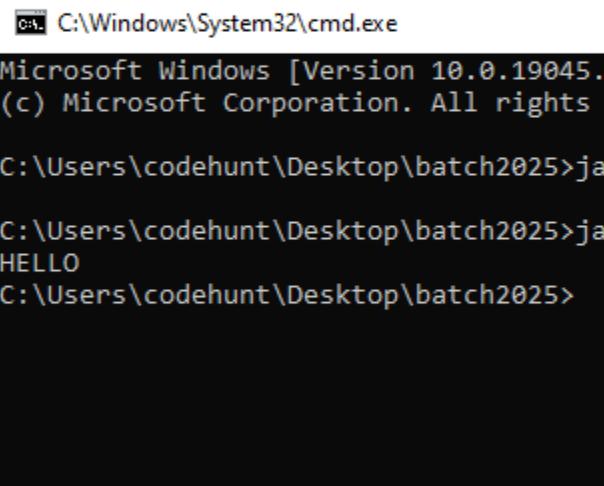
APPROACH 1 : NOTEPAD

- Select a folder
- Create a .java file
- Write your program
- Open CMD (same location)
- Compile (java c)
- Run (java)

Name	Date modified	Type	Size
Launch	1/4/2025 7:43 PM	CLASS File	1 KB
Launch	1/4/2025 7:41 PM	JAVA File	1 KB



```
Launch - Notepad
File Edit Format View Help
public class Launch
{
    public static void main(String[] args)
    {
        System.out.print("HELLO");
    }
}
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4]
(c) Microsoft Corporation. All rights reserved.

C:\Users\codehunt\Desktop\batch2025>jav
C:\Users\codehunt\Desktop\batch2025>java
HELLO
C:\Users\codehunt\Desktop\batch2025>
```

APPROACH 2 : ECLIPSE

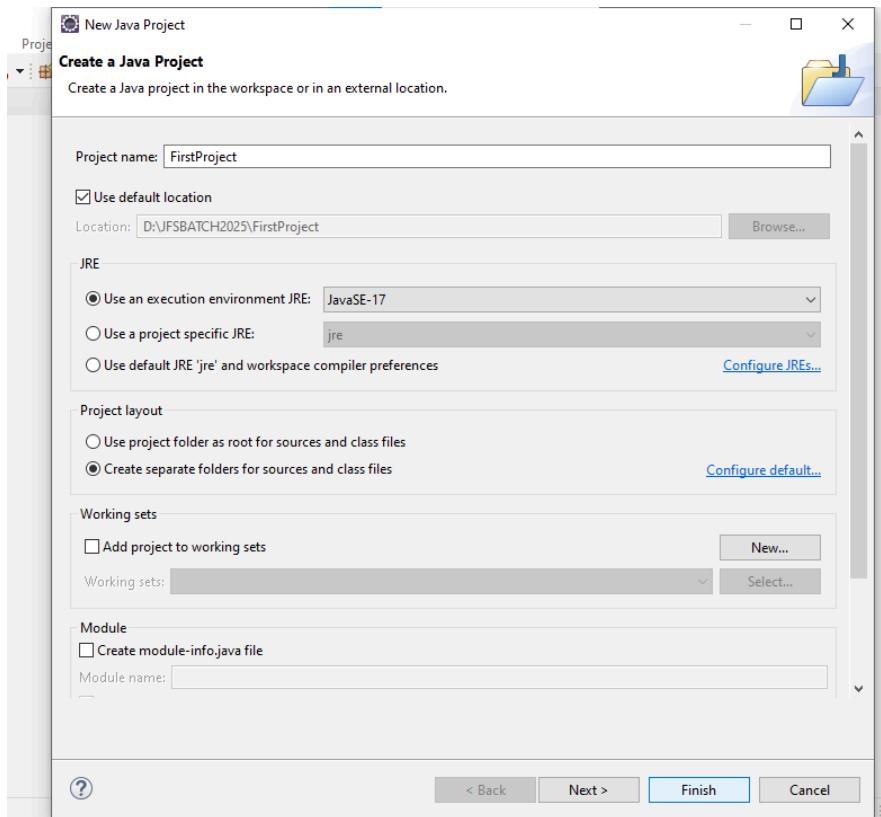
- Select a folder (CHOOSE A DIRECTORY)

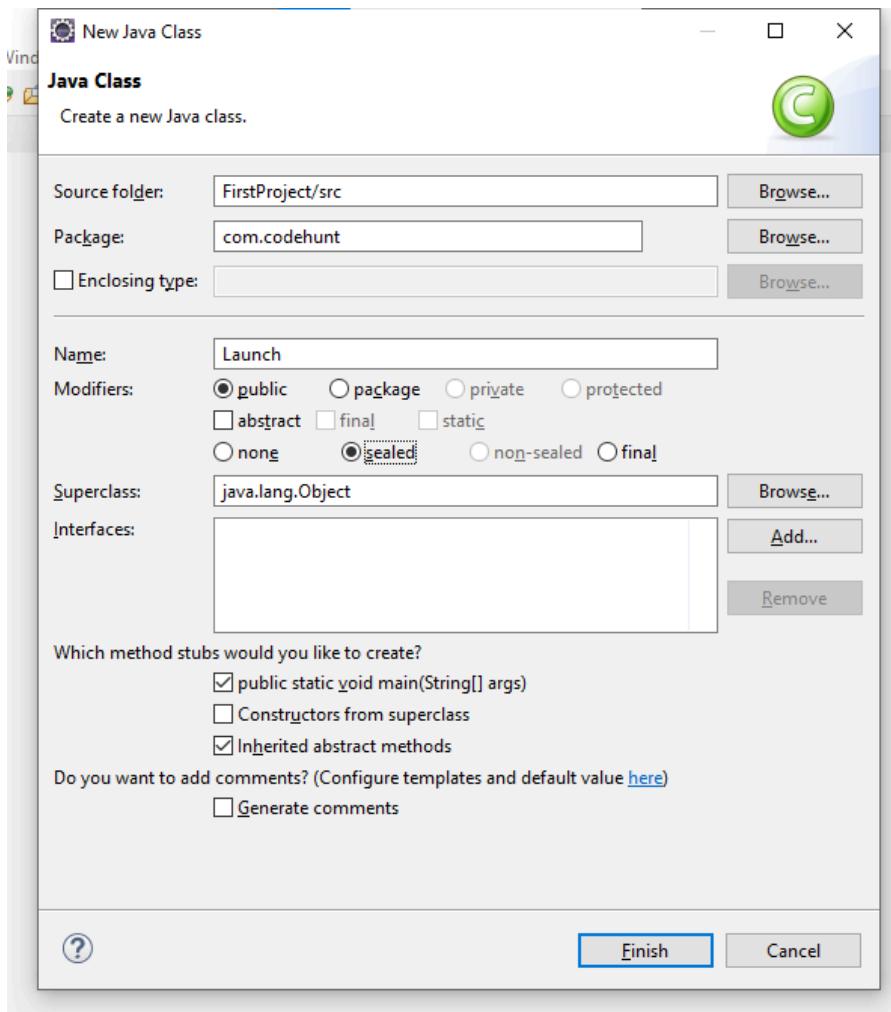
- Create a Java Project
- Create a .java file
- Write your program
- JUST CLICK ON RUN BUTTON

FIRST TIME SETUP:

By default:

- JavaEE(ADVANCED JAVA)
- You can set on Java





The screenshot shows the Eclipse IDE interface. The top window is titled "Launch.java X" and contains the following Java code:

```
1 package com.codehunt;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         System.out.println("HELLO");
7     }
8 }
9
```

The bottom window is titled "Console X" and shows the output of the application:

```
<terminated> Launch [Java Application] D:\ECLIPSE 2025\ eclipse-jee-2024-12-R-win32-x86_64\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
HELLO
```

DAY-4

Java Full Stack Development Batch 2025

History of Java

- ->Java was created by James Gosling in 1995 at SUNMICROSYSTEM (Acquired by Oracle CO. 20 APRIL 2009 7.4 BILLION \$)
- ->SUN got one project (in 1991) where they had to create software for SET OF BOXES.

- ->GREEN PROJECT
- ->GREEN TEAM (lead by JAMES gosling) other names:
Patrick Naughton ,Mike Sheredon
- ->PROJECT was initiated in 1991
- ->1995 (alpha and beta)
- ->ACCORDING TO THE TIME MAGAZINE JAVA WAS
CONSIDERED UNDER TOP 10 project of year 1995
- ->BOOK NAME: 1000 INVENTION THAT CHANGED THE
WORLD (Java)
- ->ACCORDING TO THE ORACLE: 3 BILLION RUNS JAVA
- ->1996: JAVA FIRST VERSION (JAVA1.0)

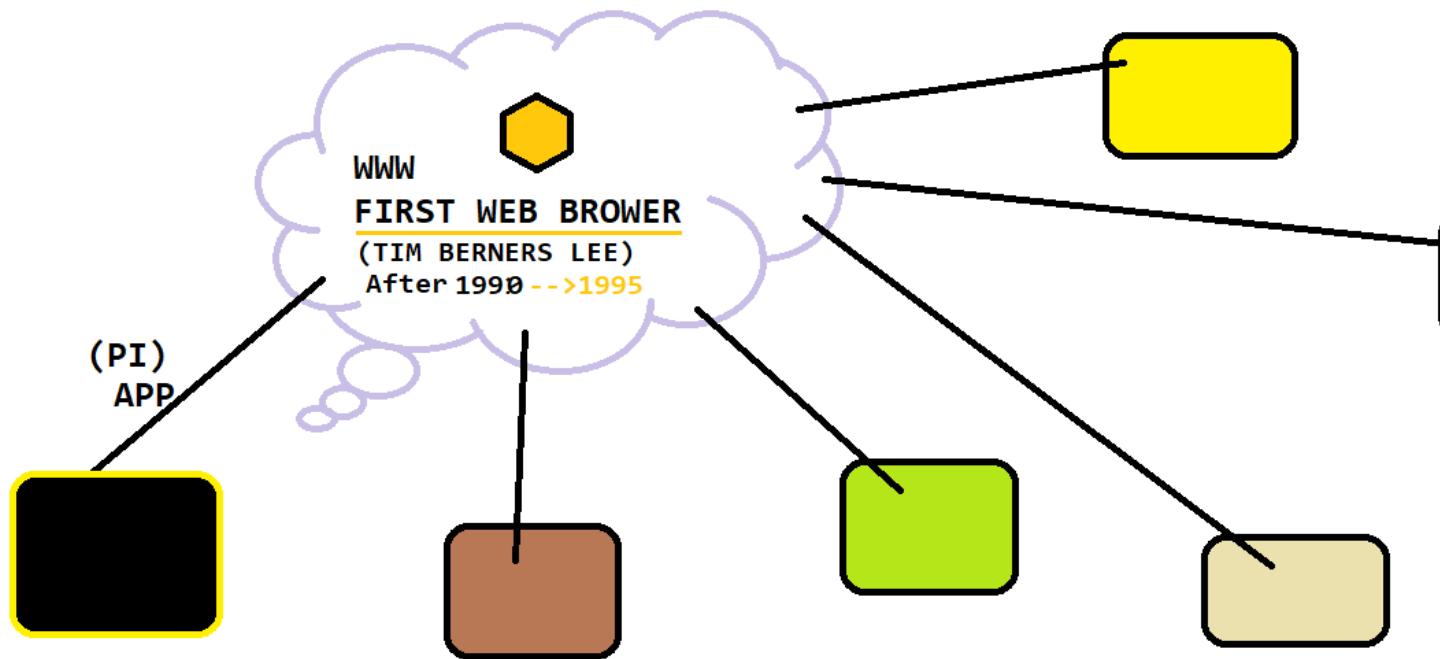
Java Naming:

- ->GREENTALK(.gt)
- ->**OAK** (RENAMED)
- ->**Java** was chosen by James gosling (Coffee)
- ->Java is an ISLAND in Indonesia
- ->1600's DUTCH EAST INDIA COMPANY
- ->FARMING OF COFFEE (Java Coffee)
- ->ESPRESSO BEAN
- ->OTHER REJECTED NAMES: dynamic, silk, jolt...etc

Popularity of Java

- World Wide Web (GLOBLIZATION)

- Regular Updates (Security)
- New Concepts (Launch new new Version)
- Large Community



DAY-5

Java Full Stack Development Batch 2025

- Core java is a term commonly used to refer fundamental concepts.

Java Edition:

- 1.J2SE(JAVA TO STANDARD EDITION)
- **JAVA SE** (JAVA TO STANDARD EDITION) : core fundamental concepts including oops, exception hadling, multithread, JDBC...
- [**Standalone application (cmd) and Desktop GUI application**] (**single system application**) : Ex. calculator
- **CORE JAVA**
- 2.J2EE (JAVA TO ENTERPRISE EDITION)
- **JAVA EE** (JAVA ENTERPRISE EDITION)
- JAKARTA EE (JAKARTA ENTERPRISE EDITION)
- **Enterprise Level Application (WEB)**
- **ADV JAVA**
- 3.J2ME
- 4.JAVA FX

Features of Java

1.Simple Language

- simple syntax
- readable language
- similar to c++
- No explicit pointer

2.Object Oriented Programming Language (OOP)

- It is a way of programming
- modularity, reusability and easier to understand

3.Platform Independent

(win->compiler->.class----->linux)

- WORA

4.Secured Language (OWAPS TOP 10) : Java 8 (Java 8 was introduced on March 18, 2014) LTS

- Byte verification

5.High Performance

6.Robust Language (MAZBOOT) : Memory Management, Can handle Faulty inputs

7.Architectural Neutral

(16 bit compiler / 32 bit compiler)

Int a=1000; int a=2000;

8.Multithreaded Language (Multithreading)

- To achieve multitasking

9.Distributed Language

- Client(machine)----->Server(machine)

10.Portable language

- Send code without compilation

11.Dynamic Language (dynamic loading)

12.Interpreted Language (Java bytecode is interpreted by the Interpreter(called by JVM))

DAY-6

Java Full Stack Development Batch 2025

FIRST JAVA PROGRAM:

In java if you want to create program you must create two things

1.class

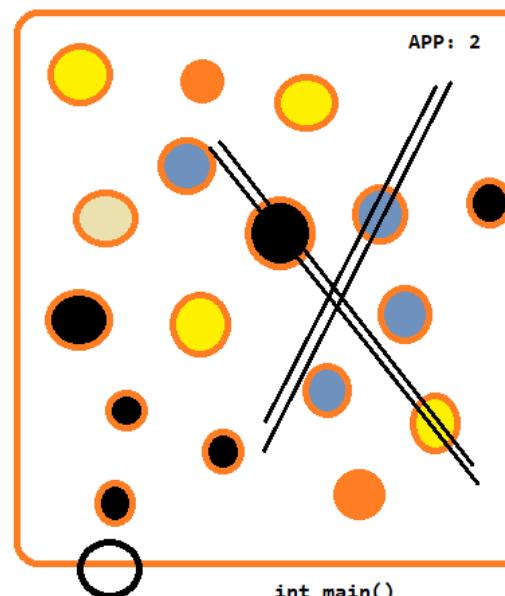
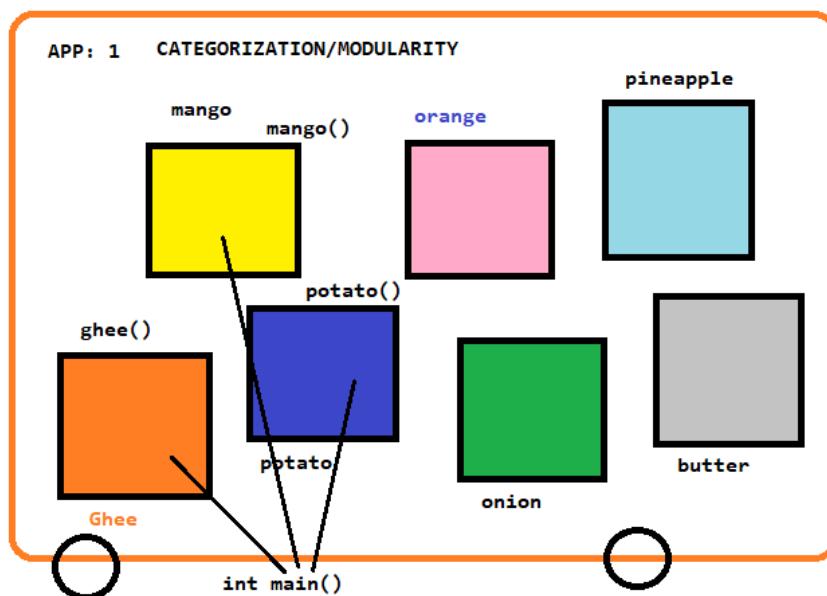
2.main method

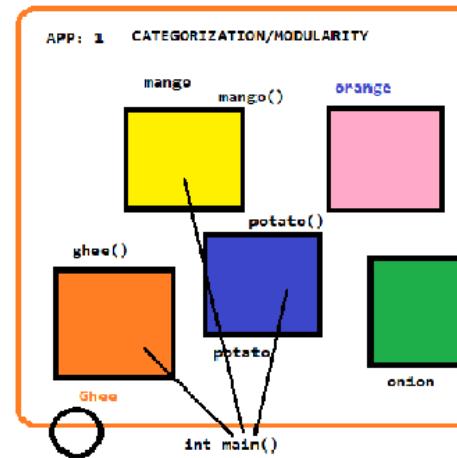
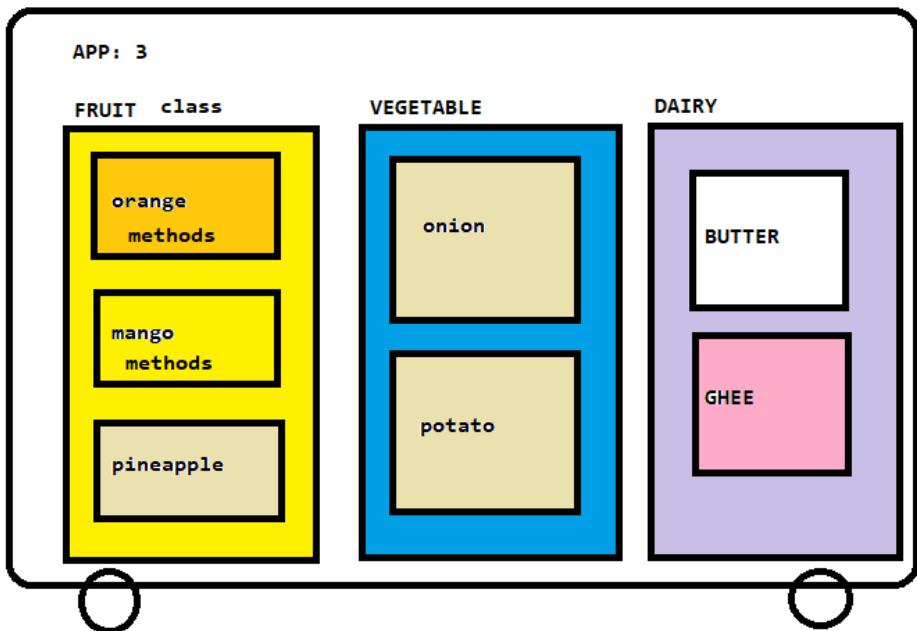
class (Later):

1.what is class?

- class is a concept of **OOP(later)** which is primarily used to achieve Categorization(Modularity)
- real-world analogy
- there is no limit of no of classes
- name should be meaningful

```
public class Launch {  
}
```





WITHOUT CLASS APPROACH

```
int main(){
}
add(){ }
sub(){ }
div(){ }
mul(){ }

logx(){}
logInverseX(){}

sinx(){}
cosx(){}
tanx{}
```

WITH CLASS

Arith

```
add(){}
sub(){}
div(){}
mul(){}
```

Trigno

```
sinx(){}
cosx(){}
tanx(){}
```

Log

```
logx(){}
logInvX(){}
```

main method:

- The main method is an entry point of a java program where the execution starts.
- We should have only one main method in a program

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

NOTE: CAN WE HAVE MULTIPLE CLASSES IN A SINGLE .java file

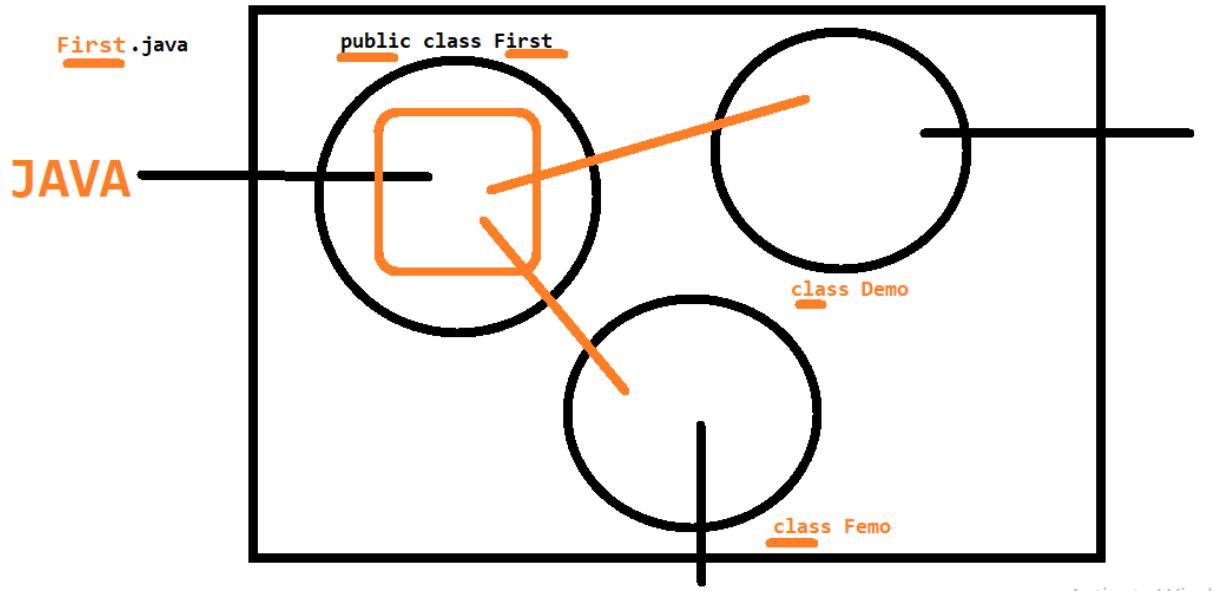
ANS: YES (IT IS NOT PREFERRED APPROACH)

The screenshot shows the Eclipse IDE interface. In the top left, there's a 'Project Explorer' view showing a project named 'rstProject' with a 'src' folder containing 'com.codehunt' and 'Launch.java'. The central part of the screen is a code editor window titled 'Launch.java' with the following content:

```
1 package com.codehunt;  
2 class Xyz{  
3  
4 }  
5 public class Launch {  
6  
7     public static void main(String[] args) {  
8         System.out.println("HELLO1");  
9     }  
10 }  
11
```

Below the code editor is a 'Console' view showing the output of the application: 'HELLO1'. The console output is as follows:

```
<terminated> Launch [Java Application] D:\ECLIPSE 2025\java-2024-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk  
HELLO1
```



DAY-7

Java Full Stack Development Batch 2025

Java Architecture

- It refers to the design and structure that defines how java program are compiled, executed and manage internally. It ensures that java program is platform independent, Robust and secured.

COMPONENTS:

- **JDK (Java Development Kit)**
 - >Compiler
 - >Necessary Tools

->Debugger

->JRE

- **JRE (Java Runtime Environment)**

->STANDARD Libraries

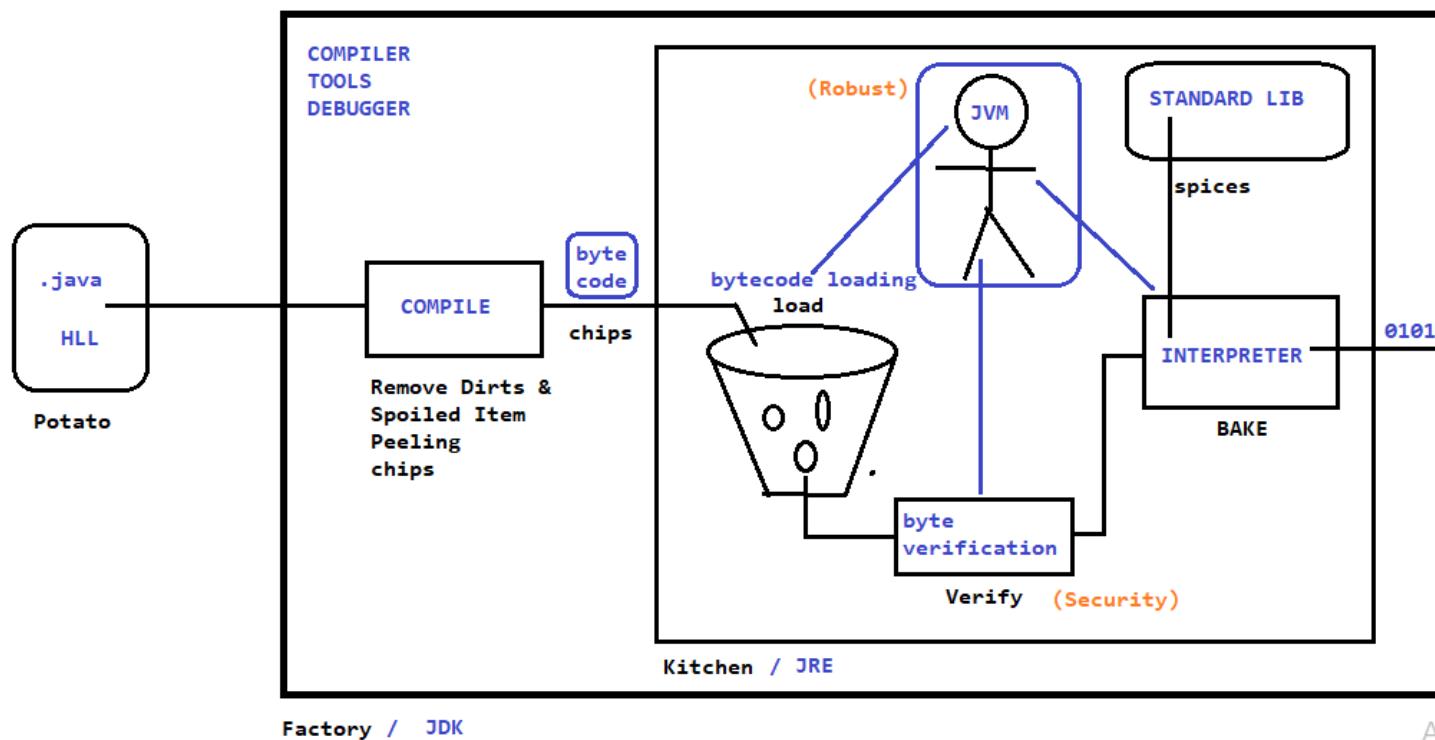
->JVM

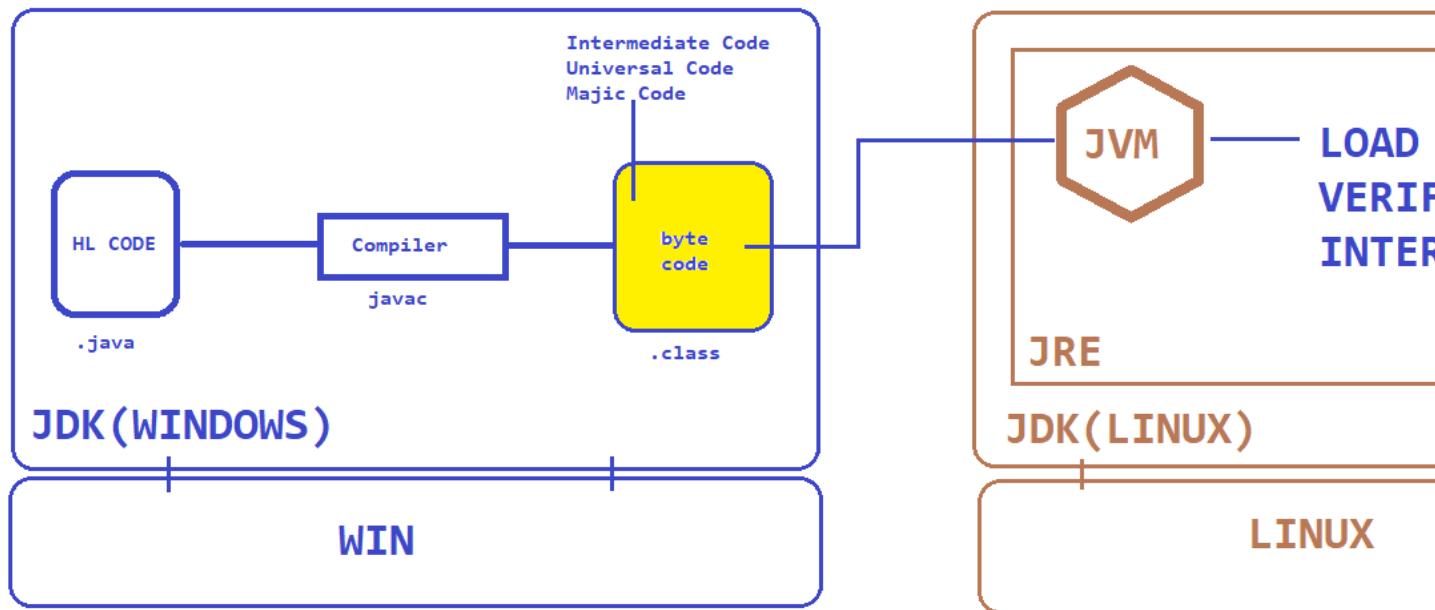
- **JVM(Java Virtual Machine)**

->It manages overall control flow of a Java Program

-> In C programming Language **main function is called By OS** coz OS is a runtime Environment for a C program

-> In Java **main method is called by JVM** coz JRE is a runtime Environment for a Java Program





Note:

- Java Application/Code is platform independent but JDK (C++) is platform dependent

CODE WRITE:

- You need JDK

CODE RUN:

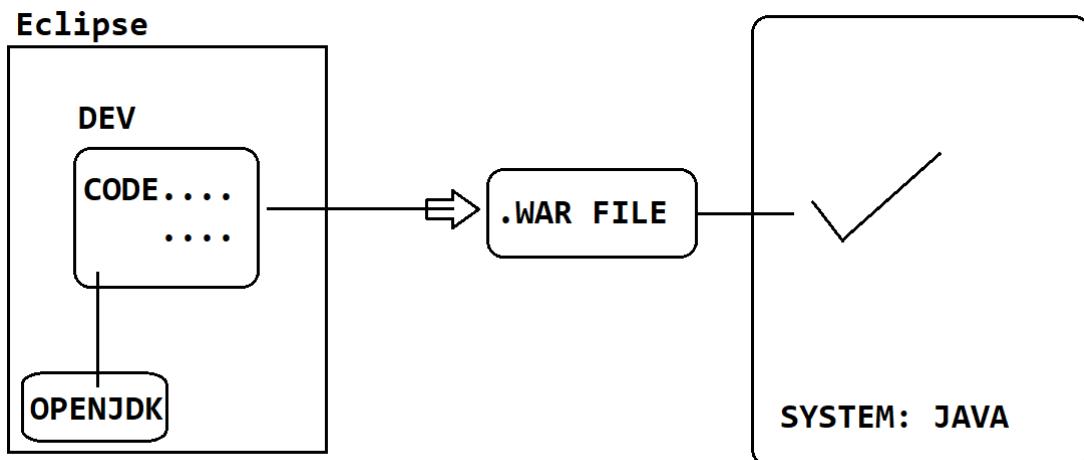
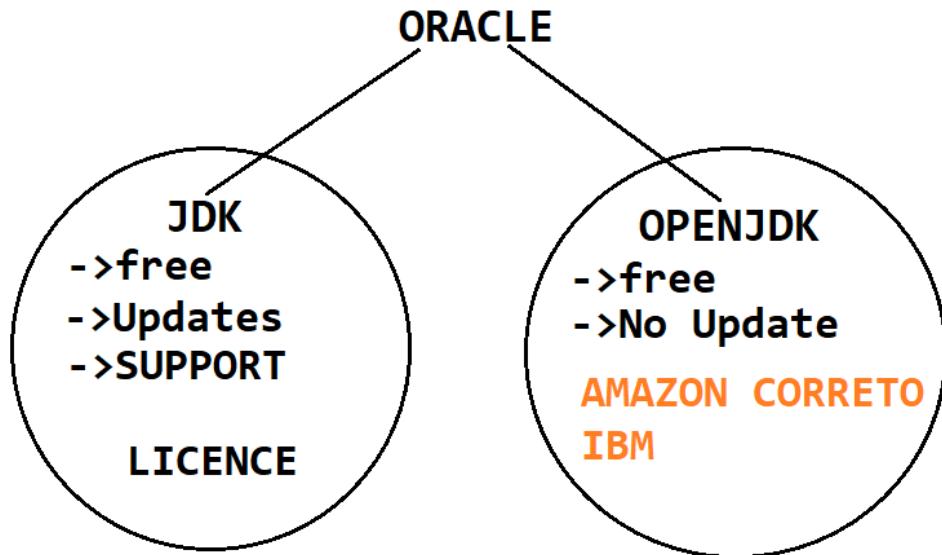
- You need only JRE (
 <https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>)

DAY-8

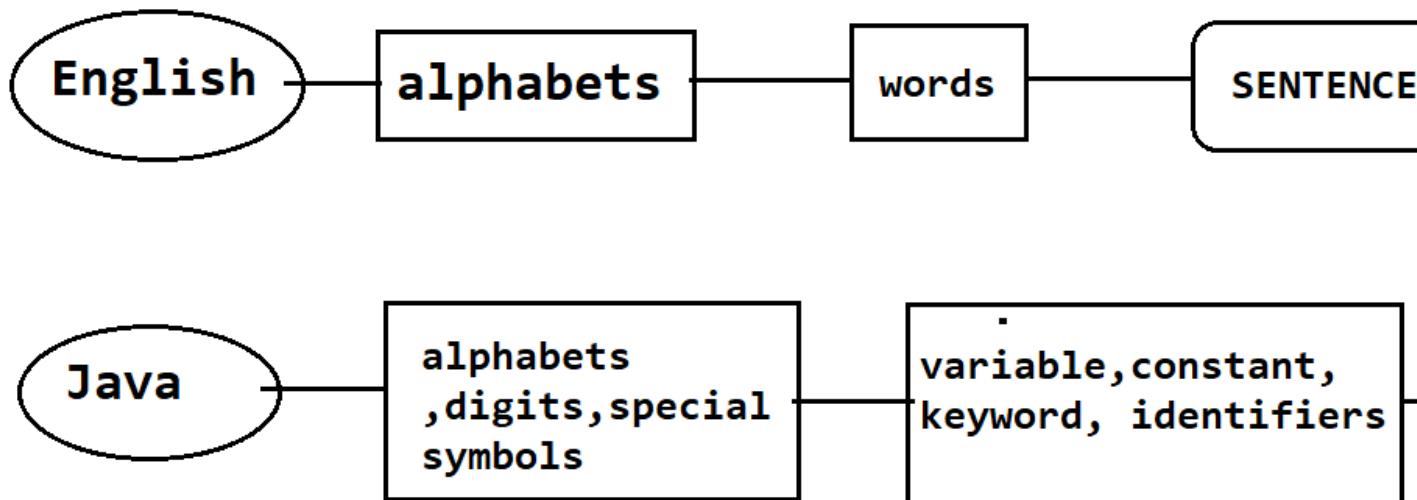
Java Full Stack Development Batch 2025

How to configure JDK in eclipse

- Latest Eclipse: Inbuilt OpenJDK



How to write a Program?



Variable Constant and Keyword

- Keyword(Reserved word)->meaning has been defined already
- In java data type checks at compile(STATICALLY TYPED LANGUAGE) time so we should provide data type with each variable

// single line comment

/*

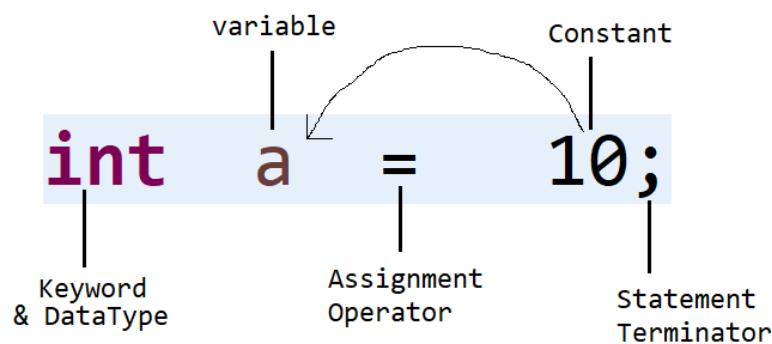
Multi line comment

*/

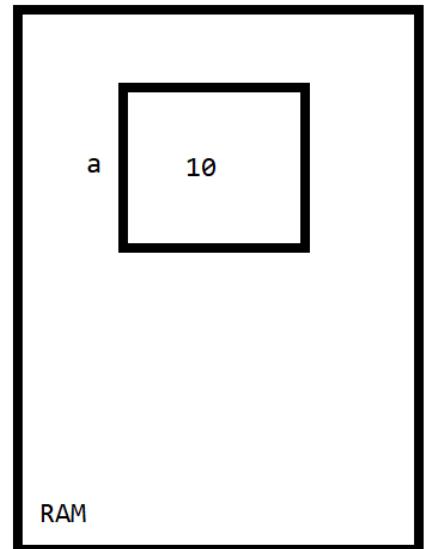
```

1 package com.codehunt;
2 public class Launch {
3
4    public static void main(String[] args) {
5
6        int a; //DECLARATION BANDAR TENSION BHAALU
7        a=10; //INITIALIZATION
8
9        //int a=10;
10    }
11 }
12

```



a==5 (Comparision)
a=5 (Assign)



NOTE:

```
Launch.java X
1 package com.codehunt;
2 public class Launch {
3
4     public static void main(String[] args) {
5         int a,b,c=30,d=40;
6         a=10;
7         b=20;
8         //System.out.println(e);
9         System.out.println(a+b+c+d);
10    }
11 }
12
```

IN A SINGLE BLOCK YOU CANNOT DECLARE TWO VARIABLE

```
Launch.java X
1 package com.codehunt;
2 public class Launch {
3     public static void main(String[] args) {
4         int a;
5         //int a; CTE
6         a=100;
7         a=67;
8         System.out.println(a);
9     }
10 }
11
```

DAY-9

Java Full Stack Development Batch 2025

Java Identifier

It is a name given to variables, methods, classes, packages...etc in a java program. It serves as a reference for programmer to interact with them.

RULES OF CONSTRUCTING AN IDENTIFIER

- 1.ALLOWED: Alphabets(A-Z,a-z) , Digits(0-9), Special Symbols(\$ _)
- 2.You can't start with digit
- 3.You can't take space
- 4.No limit of length (within 15)
- 5.Meaningful
- 6.Case Sensitive
- 7.You can't take keyword name

COVENTION

- These are recommendations or best practices aim to improve code readability, maintainability and consistency
- Conventions are not enforced by Compilers but following them make the code easier to understand

1.CAMEL CASE: (Lower Camel Case)

a.UpperCamelCase : The first letter of each word is capitalized

Ex. ThisIsMyCar , MyNameIsKhan,
RajuBanGyaGentleMan2

IN JAVA : class name, interface name,
enums

b.LowerCamelCase : The first letter should be in lower case and
subsequent word starts with Upper case

Ex. thisIsMyCar , myNameIsKhan,
rajuBanGyaGentleMan2

IN JAVA : variable , method

2.PASCAL CASE: (UpperCamelCase)

The first letter of each word is capitalized

Ex. ThisIsMyCar , MyNameIsKhan, RajuBanGyaGentleMan2

IN JAVA : class name, interface name, enums

3.SNAKE CASE (Lower Snake Case)

a.Upper Snake Case : All letters are in upper case but words are separated by underscore

Ex. THIS_IS_MY_CAR

IN JAVA : STATIC FINAL VARIABLE, CONSTANT

b. Lower Snake Case: All letters are in lower case but words are separated by underscore

Ex. this_is_my_car

MOSTLY USED IN PYTHON

4. KABAB CASE

All letters are in lower case but words are separated by hyphens

Ex. this-is-my-car

MOSTLY USED IN CSS

5. TRAIN CASE

The first letter of each word is capitalized but words are separated by hyphens

Ex. This-Is-My-Car

API DOCUMENTATION

6. FLAT CASE

All letters are in lower case with no separators between words

IN JAVA: keywords (ONLY ALPHABETS)

DAY-10

Java Full Stack Development Batch 2025

Data Types

Java Data types can be categorized into **Primitive** and **Non-Primitive (later)** Data Types

1. Primitive : stored directly inside the memory

INTEGER

1. byte (1GB->1024MB , 1MB->1024KB , 1KB->1024BYTE,
1BYTE->8BIT)

2. short (2BYTE->16BIT)

3. int (4BYTE->32Bit)

4. long (8BYTE->64BIT)

Formula:

$-2^{(bit-1)}$ to $+[2^{(bit-1)}] - 1$

-2^8-1 to $+[2^8-1] - 1$

-2^7 to $+[2^7]-1$

-128 to +127

DECIMAL

1.float (4byte) : after decimal: 5 digits

2.double(8byte) : after decimal: 12-15 digits

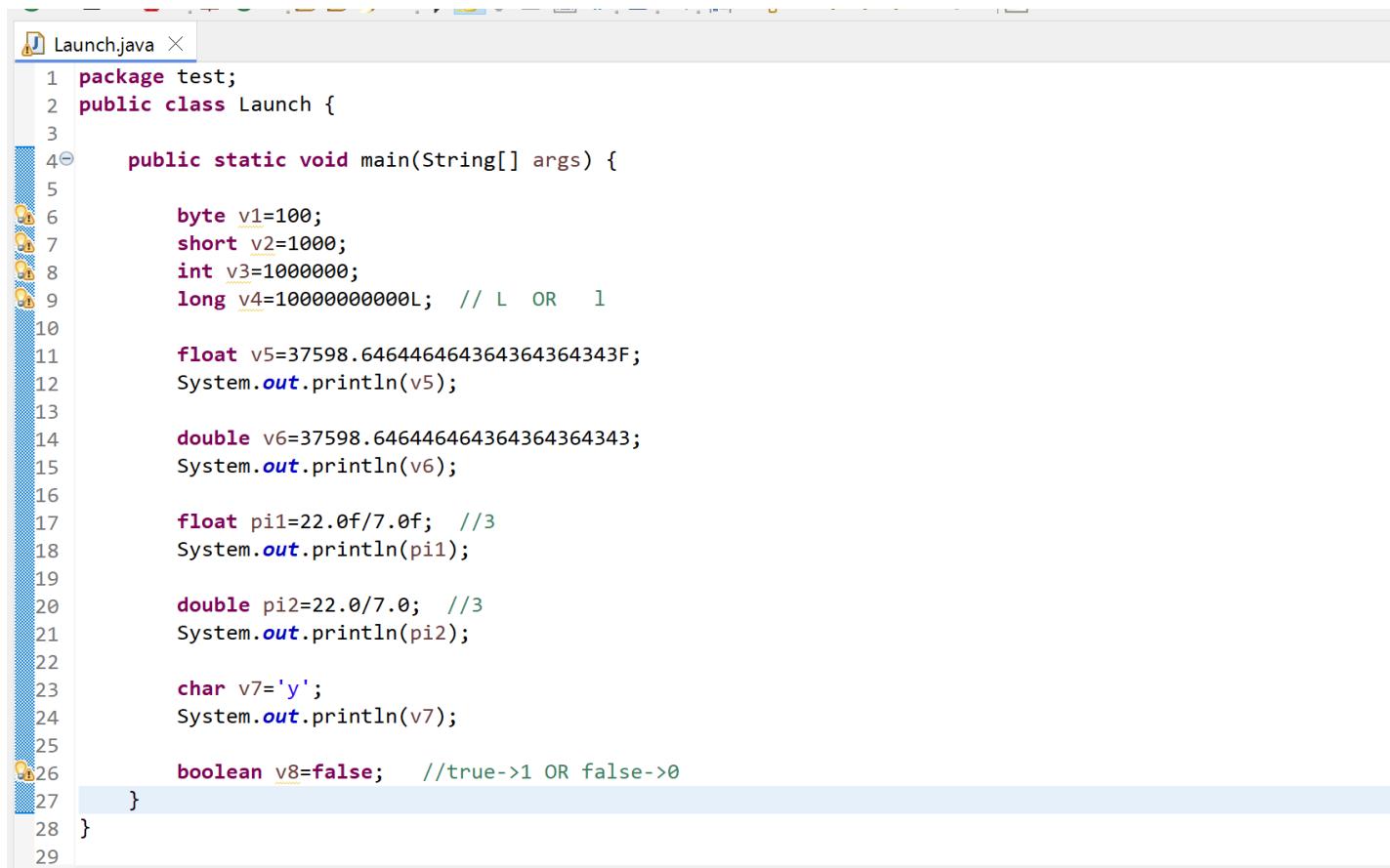
OTHER

1.char (2byte)

2.boolean(1bit)

NOTE:

integer/integer=>integer

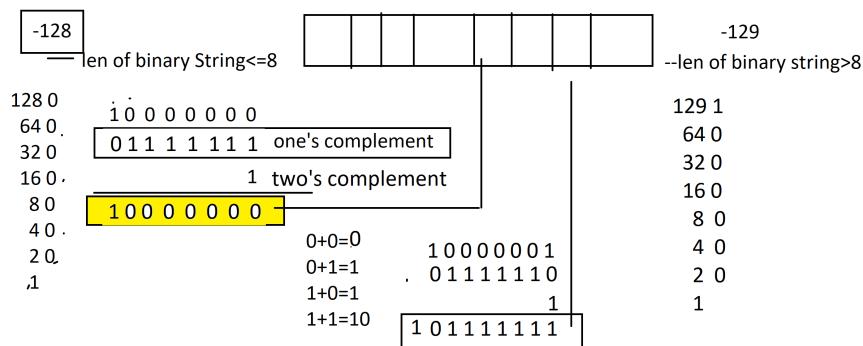
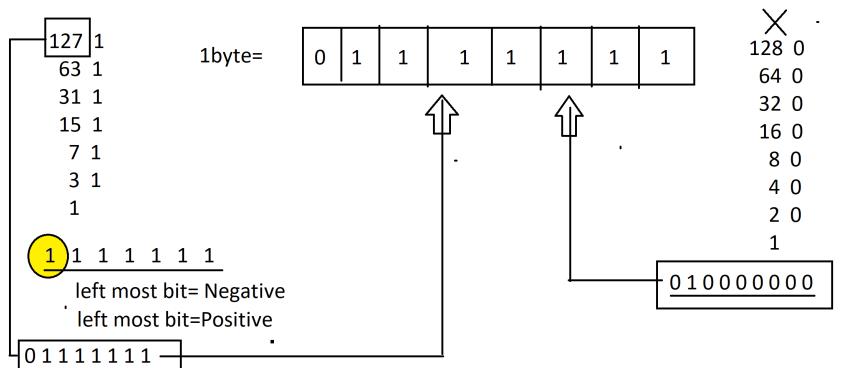


```
1 package test;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         byte v1=100;
7         short v2=1000;
8         int v3=1000000;
9         long v4=1000000000L; // L OR 1
10
11        float v5=37598.646446464364364343F;
12        System.out.println(v5);
13
14        double v6=37598.646446464364364343;
15        System.out.println(v6);
16
17        float pi1=22.0f/7.0f; //3
18        System.out.println(pi1);
19
20        double pi2=22.0/7.0; //3
21        System.out.println(pi2);
22
23        char v7='y';
24        System.out.println(v7);
25
26        boolean v8=false; //true->1 OR false->0
27    }
28
29 }
```

WHY RANGE IS FIXED ? (INTEGER)

Byte (-128 to +127)

Size: 1 byte



DAY-11

Java Full Stack Development Batch 2025

Java Control Statement

->it manages the flow of control

1.Decision Making Control Flow Statement

a.if statement

- 1.simple if statement
- 2.if else statement
- 3.If else if statement(if else ladder/else if clause)
- 4.Nested if else

b.switch statement

2.Loop Control Flow Statement

- 1.for loop statement
- 2.while loop statement
- 3.do while loop statement
- 4.For each loop statement/enhance for loop (LATER)

3.Jump Control Statement

- 1.break statement
 - 2.continue statement
-

if statement

- 1.simple if statement
- 2.if else statement
- 3.If else if statement(if else ladder/else if clause)
- 4.Nested if else

1.simple if statement

```
if(condition) {  
    if condition is true  
}
```

The screenshot shows a Java IDE interface. On the left is a code editor with a tab labeled "Launch.java". The code is as follows:

```
1 package test;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         //ELIGIBLE : PER>=60
7         int per=60;
8         if(per>=60)
9         {
10             System.out.println("ELIGIBLE1");
11             System.out.println("ELIGIBLE2");
12             System.out.println("ELIGIBLE3");
13         }
14         System.out.println("EXIT");
15     }
16 }
17
```

On the right is a "Console" window with the following output:

```
<terminated> L
ELIGIBLE
ELIGIBLE
ELIGIBLE
EXIT
```

Note: Java is a free form language

If there is no curly bracket just after is if() then
the first statement just after if will be
considered inside and rest other statement will
be considered outside

The screenshot shows a Java IDE interface. On the left, the code editor displays a file named 'Launch.java' with the following content:

```
1 package test;
2
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         // ELIGIBLE : PER>=60
8         int per = 9;
9         if (per >= 60); {
10             System.out.println("ELIGIBLE1");
11             System.out.println("ELIGIBLE2");
12             System.out.println("ELIGIBLE3");
13         }
14         System.out.println("EXIT");
15     }
16 }
17 //ctrl shift F
```

The code uses an if statement to check if the variable 'per' is greater than or equal to 60. If true, it prints "ELIGIBLE1", "ELIGIBLE2", and "ELIGIBLE3". If false, it prints "EXIT". The condition in the if statement is missing a closing brace, which is highlighted by a red squiggly underline.

On the right, the 'Console' tab shows the output of the program:

```
<terminated> Launch
ELIGIBLE1
ELIGIBLE2
ELIGIBLE3
EXIT
```

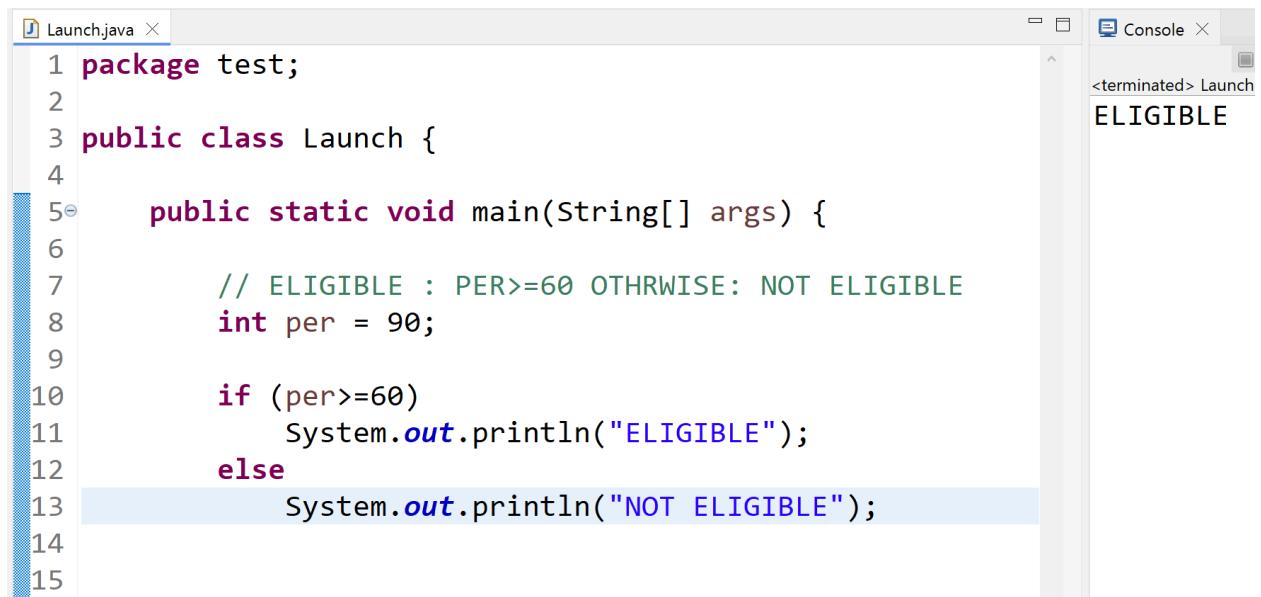
2. if else statement

- You cannot create else without if
- Else must be just after if statement

```
if (per>=60);
```

```
{
    System.out.println("ELIGIBLE");
}
```

```
else ; {  
    System.out.println("NOT ELIGIBLE1");  
    System.out.println("NOT ELIGIBLE2");  
}  
OUTPUT: compile time error
```



The screenshot shows an IDE interface with two tabs: 'Launch.java' and 'Console'. The 'Launch.java' tab contains the following Java code:

```
1 package test;  
2  
3 public class Launch {  
4  
5     public static void main(String[] args) {  
6  
7         // ELIGIBLE : PER>=60 OTHRWISE: NOT ELIGIBLE  
8         int per = 90;  
9  
10        if (per>=60)  
11            System.out.println("ELIGIBLE");  
12        else  
13            System.out.println("NOT ELIGIBLE");  
14  
15}
```

The 'Console' tab shows the output: 'ELIGIBLE'.

DAY-12

Java Full Stack Development Batch 2025

Nested if else

If you have to print a success message based of multiple conditions and you have to print one-one error message for each condition

Task:

Per>=60 , tc='y', pcm>=50

=>you are eligible for admission

Not Eligible: Low Percentage, No TC, Low PCM

USER INPUT?

```
Launch.java ×
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         Scanner scanner=new Scanner(System.in);
8         //1.byte
9         System.out.print("ENTER A DEMO BYTE VALUE: ");
10        byte b=scanner.nextByte();
11        System.out.println(b);
12
13        short st=scanner.nextShort();
14        int i=scanner.nextInt();
15        long l=scanner.nextLong();
16
17        float f=scanner.nextFloat();
18        double d=scanner.nextDouble();
19
20        boolean bool=scanner.nextBoolean();
21        char c=scanner.next().charAt(0);
22    }
23}
24
```

NOTE:

1. You need not to import classes of same package and java.lang package



The screenshot shows the Eclipse IDE interface. The Package Explorer view on the left shows a project structure with a src folder containing com.mainapp, which has two files: Launch.java and Scanner.java. The Launch.java file is open in the editor, displaying Java code. The code imports java.util.Scanner and java.util.*. It defines a public class Launch with a main method that creates a Scanner object from System.in. The code then continues with comments about Scanner classes and user input. The Console view at the bottom shows the output of the application: 'RUN'.

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {
        java.util.Scanner scan=new java.util.Scanner(System.in);
        //10 more classes from java.util
    }
    //Scanner : com.mainapp(userDefined)
    //Scanner : java.util(inbuilt) : USER INPUT
}
```

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

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

        System.out.print("ENTER YOUR PERCENTAGE : ");
        byte per=scanner.nextByte();

        System.out.print("DO YOU HAVE TC? : ");
        char tc=scanner.next().charAt(0);

        System.out.print("ENTER YOUR PCM PERCENTAGE: ");
        byte pcm=scanner.nextByte();

        if(per>=60) {
            if(tc=='Y') {
                if(pcm>=50) {
```

```
        System.out.print("U R ELIGIBLE");
    }
    else{
        System.out.print("NOT ELIGIBLE: PCM PER");
    }
}
else {
    System.out.print("NOT ELIGIBLE: TC");
}
else {
    System.out.print("NOT ELIGIBLE: OVERALL PERCENTAGE");
}
}
```

DAY-13

Java Full Stack Development Batch 2025

If else if statement(if else ladder/else if clause)

Ex.

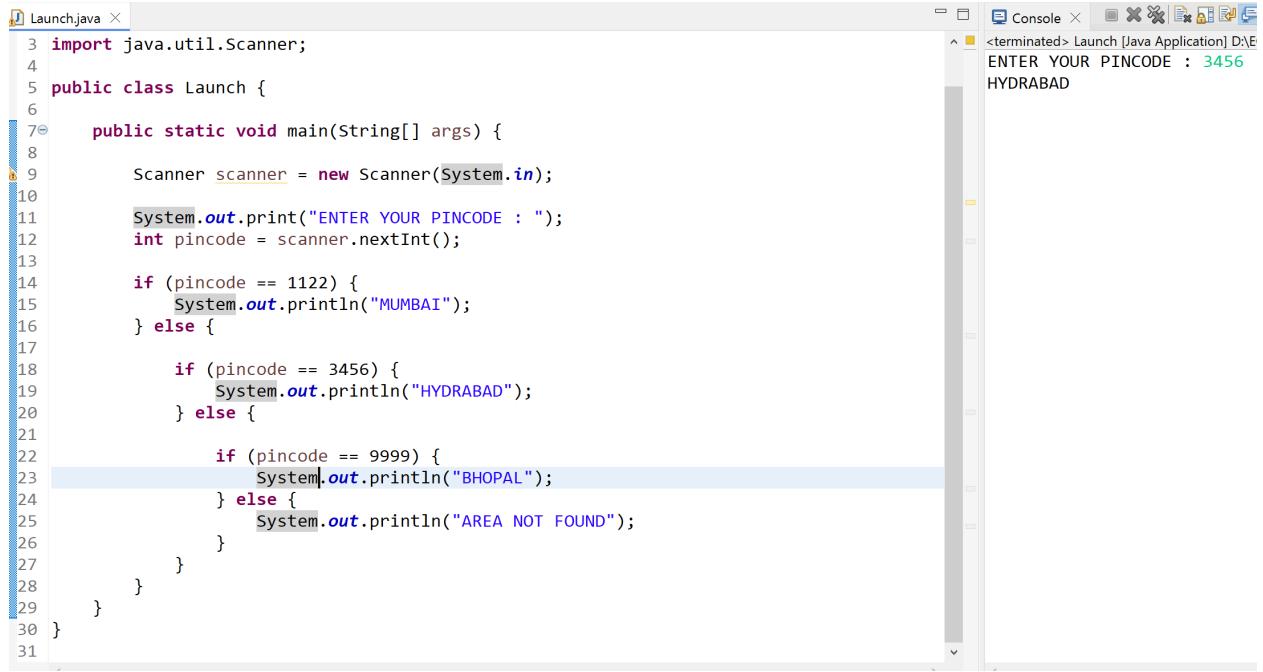
User pincode: 1122->Mumbai,3456->Hydrabad,9999->Bhopal,
AREA NOT FOUND

Problem with Nested

1.Right side creep

2. Bracket Match

Not Preferred Approach



```
Launch.java X
3 import java.util.Scanner;
4
5 public class Launch {
6
7     public static void main(String[] args) {
8
9         Scanner scanner = new Scanner(System.in);
10
11         System.out.print("ENTER YOUR PINCODE : ");
12         int pincode = scanner.nextInt();
13
14         if (pincode == 1122) {
15             System.out.println("MUMBAI");
16         } else {
17
18             if (pincode == 3456) {
19                 System.out.println("HYDRABAD");
20             } else {
21
22                 if (pincode == 9999) {
23                     System.out.println("BHOPAL");
24                 } else {
25                     System.out.println("AREA NOT FOUND");
26                 }
27             }
28         }
29     }
30 }
```

Console X <terminated> Launch [Java Application] D:\E
ENTER YOUR PINCODE : 3456
HYDRABAD

Best Way:

The screenshot shows a Java development environment with two windows. The left window is titled "Launch.java" and contains the following Java code:

```
4 public class Launch {  
5     public static void main(String[] args) {  
6         Scanner scanner = new Scanner(System.in);  
7         System.out.print("ENTER YOUR PINCODE : ");  
8         int pincode = scanner.nextInt();  
9         if (pincode == 1122)  
10        {  
11            System.out.println("MUMBAI");  
12        }  
13        else if(pincode == 3456)  
14        {  
15            System.out.println("HYDRABAD");  
16        }  
17        else if(pincode == 9999)  
18        {  
19            System.out.println("BHOPAL");  
20        }  
21        else  
22        {  
23            System.out.println("AREA NOT FOUND");  
24        }  
25    }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }
```

The right window is titled "Console" and shows the output of the application. It reads "ENTER YOUR PINCODE : 3456" and then prints "HYDRABAD".

Multiple if vs else if

The screenshot shows the Eclipse IDE interface. On the left is the code editor with a file named 'Launch.java'. The code implements a simple application that reads a pincode from the user and prints the corresponding city name based on the pincode. The code uses both traditional if-else blocks and modern ternary operators. On the right is the 'Console' window, which displays the output of the application when run. The user entered '1122', and the application printed 'MUMBAI'.

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.print("ENTER YOUR PINCODE : ");
10        int pincode = scanner.nextInt();
11
12        // if (pincode == 1122) //true
13        //{
14        //    System.out.println("MUMBAI");
15        //}
16        //if(pincode == 3456)
17        //{
18        //    System.out.println("HYDRABAD");
19        //}
20        //if(pincode == 9999)
21        //{
22        //    System.out.println("BHOPAL");
23        //}
24
25        if (pincode == 1122) //true
26        {
27            System.out.println("MUMBAI");
28        }
29        else if(pincode == 3456)
30        {
31            System.out.println("HYDRABAD");
32        }
33        else if (pincode == 9999)
34        {
35            System.out.println("BHOPAL");
36        }
37    }
38}
39
```

Console Output:

```
Launch [Java Application] D:\ECLIPSE IDE
ENTER YOUR PINCODE : 1122
MUMBAI
```

Logical Operators

AND (&&) : both condition should be true=>true

OR (||) : any one condition must be true=>true

NOT (!) : opposite

Task:

Per>=60 , tc='y', pcm>=50

=>you are eligible for admission

Otherwise: Not Eligible

Program: and operator

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        Scanner scanner=new Scanner(System.in);

        System.out.print("ENTER YOUR PERCENTAGE : ");
        byte per=scanner.nextByte();

        System.out.print("DO YOU HAVE TC? : ");
        char tc=scanner.next().charAt(0);

        System.out.print("ENTER YOUR PCM PERCENTAGE: ");
        byte pcm=scanner.nextByte();

        if(per>=60 && tc=='Y' && pcm>=50)
        {
            System.out.println("ELIGIBLE ");
        }
        else
        {
            System.out.println("NOT ELIGIBLE");
        }

//        if(per>=60) {
//        
```

```

//           if(tc=='Y') {
//
//               if(pcm>=50) {
//                   System.out.println("ELIGIBLE ");
//               }
//               else
//               {
//                   System.out.println("NOT ELIGIBLE");
//               }
//           }
//           else
//           {
//               System.out.println("NOT ELIGIBLE");
//           }
//       }
//   }
}

```

Program OR operator

```

if(per>=60 || tc=='Y' || pcm>=50)
{
    System.out.println("ELIGIBLE ");
}
else
{
    System.out.println("NOT ELIGIBLE");
}

```

Program Not Operator

```

if( !(per>=60 && tc=='Y')      )
{
    System.out.println("ELIGIBLE ");
}
else
{
    System.out.println("NOT ELIGIBLE");
}

```

Interview Q:

```
int a=10;  
  
if( a>5){  
    break;  
    Sop("Hello");  
}  
  
Sop("Exit");  
  
OP: compile time error
```

DAY-14

Java Full Stack Development Batch 2025

Switch Statement

- >In Java Switch is used to create a Menu
- >it is almost similar to if else if statement but it provides an efficient way to dispatch execution flow to diff part of program
- >if you are creating a switch statement then try to take cases in contiguous order.
- >If you take contiguous number(1,2,3...etc) then internally java compiler uses **JUMP TABLE** to store those conditions and then control will directly jump on the case which you want to execute.

- >in case of Non Contiguous case the compiler will use **LOOKUP TABLE** and still it is more efficient than if else if
- >Case must be unique
- >You can created Nested Switch
- >Primitive: byte short int char
- >Non Primitive: String enum Wrapper classes

USE of if else if:

- 1.Decimal
- 2.Object compare
- 3.Complex Expressions

```

package com.mainapp;

import java.util.Scanner;

public class Launch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("ENTER Num1:");
        int num1 = scanner.nextInt();

        System.out.print("ENTER Num2:");
        int num2 = scanner.nextInt();

        System.out.println("P1->add\nP2->sub\nP3->div\nP4->mul");

        System.out.print("Enter Your Choice: ");
        int choice = scanner.nextInt(); // 4
    }
}

```

```
switch (choice) {  
  
    case 1:  
        System.out.println(num1+num2);  
        break;  
  
    case 2:  
        System.out.println(num1-num2);  
        break;  
  
    case 3:  
        System.out.println(num1/num2);  
        break;  
  
    case 4:  
        System.out.println(num1*num2);  
        break;  
  
    default:  
        System.out.println("You have entered wrong  
choice");  
        break;  
}  
  
//  
//    if(choice==1) {  
//        System.out.println(num1+num2);  
//    }  
//    else if(choice==2) {  
//        System.out.println(num1-num2);  
//    }  
//    else if(choice==3) {  
//        System.out.println(num1/num2);  
//    }  
//    else if(choice==4) {  
//        System.out.println(num1*num2);  
//    }  
//    else {  
//        System.out.println("You have entered wrong choice");  
//    }  
}  
}
```

```
=====
=====

package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("ENTER Num1:");
        int num1 = scanner.nextInt();

        System.out.print("ENTER Num2:");
        int num2 = scanner.nextInt();

        System.out.println("P-A->add\nP-B->sub\nP-C->div\nP-D->mul");

        System.out.print("Enter Your Choice: ");
        char choice = scanner.next().charAt(0);

        switch (choice) {

            case 'A':
                System.out.println(num1+num2);
                break;

            case 'B':
                System.out.println(num1-num2);
                break;

            case 'C':
                System.out.println(num1/num2);
                break;

            case 'D':
                System.out.println(num1*num2);
                break;

            default:
                System.out.println("You have entered wrong
choice");
        }
    }
}
```

```

        break;
    }

//      if(choice==1) {
//          System.out.println(num1+num2);
//      }
//      else if(choice==2) {
//          System.out.println(num1-num2);
//      }
//      else if(choice==3) {
//          System.out.println(num1/num2);
//      }
//      else if(choice==4) {
//          System.out.println(num1*num2);
//      }
//      else {
//          System.out.println("You have entered wrong choice");
//      }
}

}

```

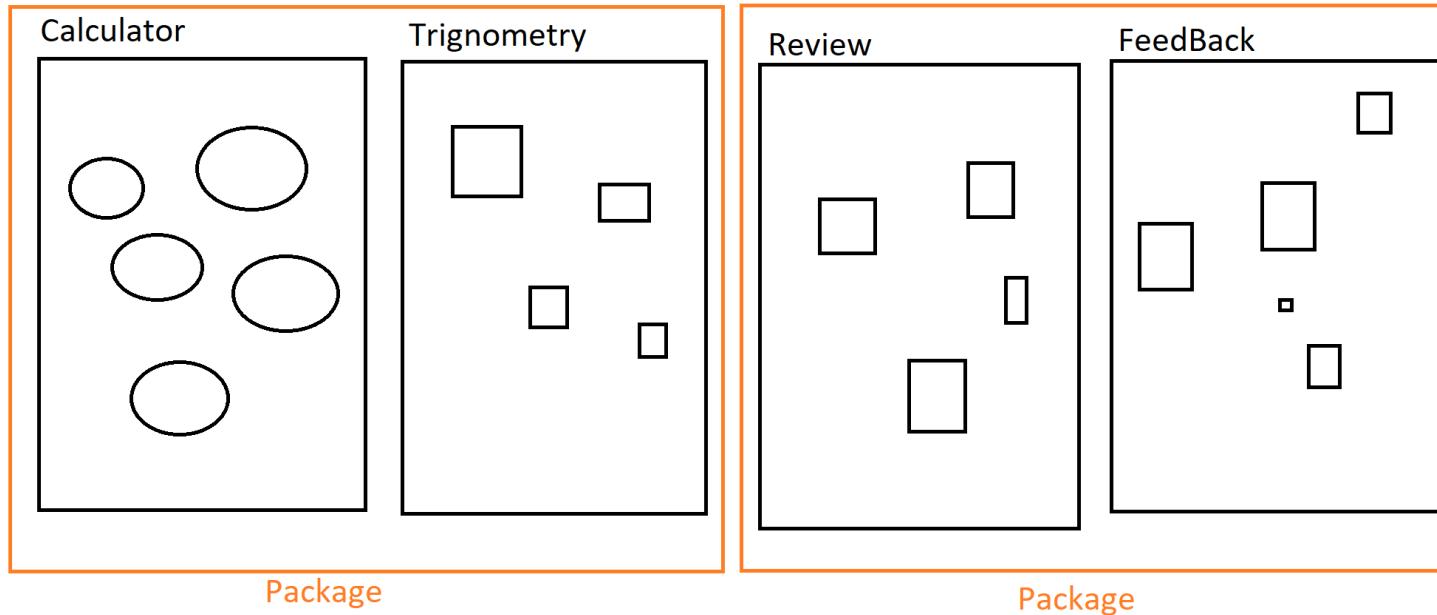
NEXT JOURNEY:

2.Loop Control Flow Statement

- 1.for loop statement
- 2.while loop statement
- 3.do while loop statement
- 4.For each loop statement/enhance for loop (LATER)

3.Jump Control Statement

- 1.break statement
- 2.continue statement



DAY-15

Java Full Stack Development Batch 2025

2.Loop Control Flow Statement : it is used to repeat/iterate some piece of code

- 1.for loop statement
- 2.while loop statement
- 3.do while loop statement
- 4.For each loop statement/enhance for loop (LATER)

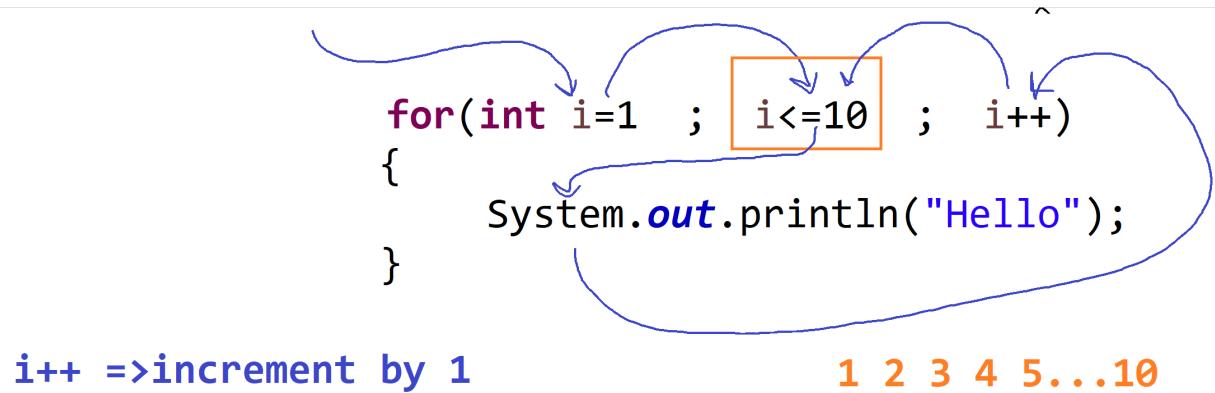
3.Jump Control Statement

- 1.break statement

- 2. continue statement

for loop statement

->when the number of iteration is fixed

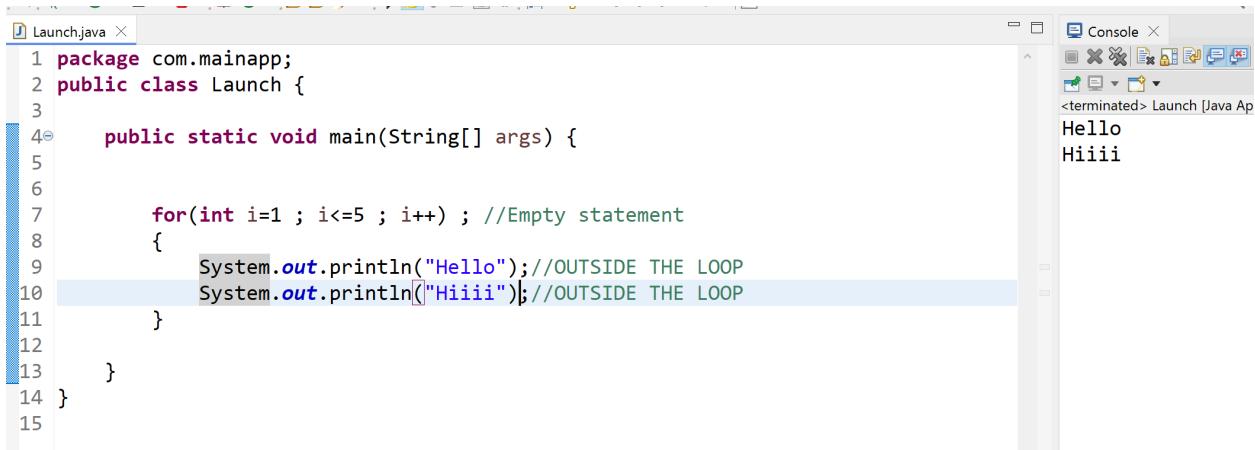


A screenshot of an IDE showing the execution of a Java program. The left pane shows the code in `Launch.java`:

```
package com.mainapp;  
public class Launch {  
    public static void main(String[] args) {  
        //TASK: Print Hello 10 Times  
        for(int i=1000 ; i<1010 ; i++)  
        {  
            System.out.println("Hello");  
        }  
        //1000 1001 1002...1009= 10 times  
    }  
}
```

The right pane shows the `Console` output, which displays the string "Hello" ten times, corresponding to the iterations of the loop.

If there is no curly bracket with for loop then the first statement just after for statement will be considered inside the loop



The screenshot shows an IDE interface with two main panes. On the left is the code editor for a file named 'Launch.java'. The code contains a for loop that prints 'Hello' and 'Hiiii' to the console. On the right is the 'Console' tab, which displays the output: 'Hello' followed by 'Hiiii'. The code in the editor is as follows:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         for(int i=1 ; i<=5 ; i++) ; //Empty statement
7         {
8             System.out.println("Hello");//OUTSIDE THE LOOP
9             System.out.println("Hiiii");//OUTSIDE THE LOOP
10        }
11    }
12}
13}
14}
15}
```

Note:

i++ : increment by 1

i-- : decrement by 1

lower to higher : condition < or <= : i++

higher to lower : condition > or >= : --

Increment and decrement operator

1.post increment(i++)

2.pre increment(++i)

1.post decrement(i--)

2.pre decrement(--i)

Ex.post increment(i++)

```
int k=10;  
//      System.out.println(k); //10  
//      System.out.println(k++); //USE then Increment 10  
//      System.out.println(k); //11
```

```
int b=k;  
System.out.println(b); //10  
b=k++; //USE then increment  
System.out.println(b); //10  
System.out.println(k); //11
```

Ex.pre increment

```
int k=10;  
//      System.out.println(k); //10  
//      System.out.println(++k); //Increment the USE 11  
//      System.out.println(k); //11  
  
int b=k;  
System.out.println(b); //10  
b=++k; //increment then USE  
System.out.println(b); //11  
System.out.println(k); //11
```

Ex.post decrement

```
int k=10;  
//      System.out.println(k); //10  
//      System.out.println(k--); //USE then decrement 10  
//      System.out.println(k); //9  
  
int b=k;  
System.out.println(b); //10  
b=k--; //USE then decrement  
System.out.println(b); //10  
System.out.println(k); //9
```

Ex. Pre decrement

```

int k=10;
//      System.out.println(k); //10
//      System.out.println(--k); //decrement then USE 9
//      System.out.println(k); //9

int b=k;
System.out.println(b); //10
b=--k; //decrement the USE
System.out.println(b); //9
System.out.println(k); //9

```

```

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

```

- i=1
- i=(i+1)
- i= 1+1
- i=2

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

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

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

//      for(int i=10; i>=5 ; i-- )
{
//      System.out.println("Hello");
}

//      for(int i=10; i>=5 ; i-- )
{
//      System.out.println("Hello");
}

//      for(int i=1; i<=5 ; ++i )
{
//      System.out.println("Hello");
```

```
//      }

//      for(int i=1; i<=5 ; i=i+2 )
//{
//      System.out.println("Hello");
}

//      for(int i=10; i>=5 ; i=i-2 )
//{
//      System.out.println("Hello");
}
```

DAY-16

●Java Full Stack Development Batch 2025

Types of Variable

1. Local variable : declared inside the method/constructor/block

Scope: within the block where it is declared

Local variable must be initialized and it has no default value

```
Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int a; //DECLARATION
7         a=100; //INITIALIZATION
8         int c; //DECLARATION
9         System.out.println(a);
10
11     {
12         int b; //DECLARATION
13         b=200; //INITIALIZATION
14         c=300; //INITIALIZATION
15
16         System.out.println(a);
17         System.out.println(b);
18         System.out.println(c);
19     }
20
21     //System.out.println(b); CTE
22     System.out.println(c);
23 }
24
25 }
```

2.Instance variable(later)

3.class/static variable(later)

Nested for loop:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int k=1;
7         for(int i=1;i<=5;i++)
8         {
9             for(int j=1 ;j<=5; j++){
10                 System.out.println("HELLO"+k);
11                 k++;
12             }
13         }
14     }
15 }
16
```

Java Jump Statement

1.break 2.continue

break : it is used to break the flow of a loop and
switch(cant use outside the loop & switch)

continue: it is used to move on to the next iteration (cant
use outside the loop)

```
Launch.java ×
```

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         for(int i=1;i<=5;i++)
7         {
8             if(i==3) {
9                 break;
10            }
11            System.out.println("HELLO");
12        }
13
14        System.out.println("EXIT");
15
16    }
17}
```

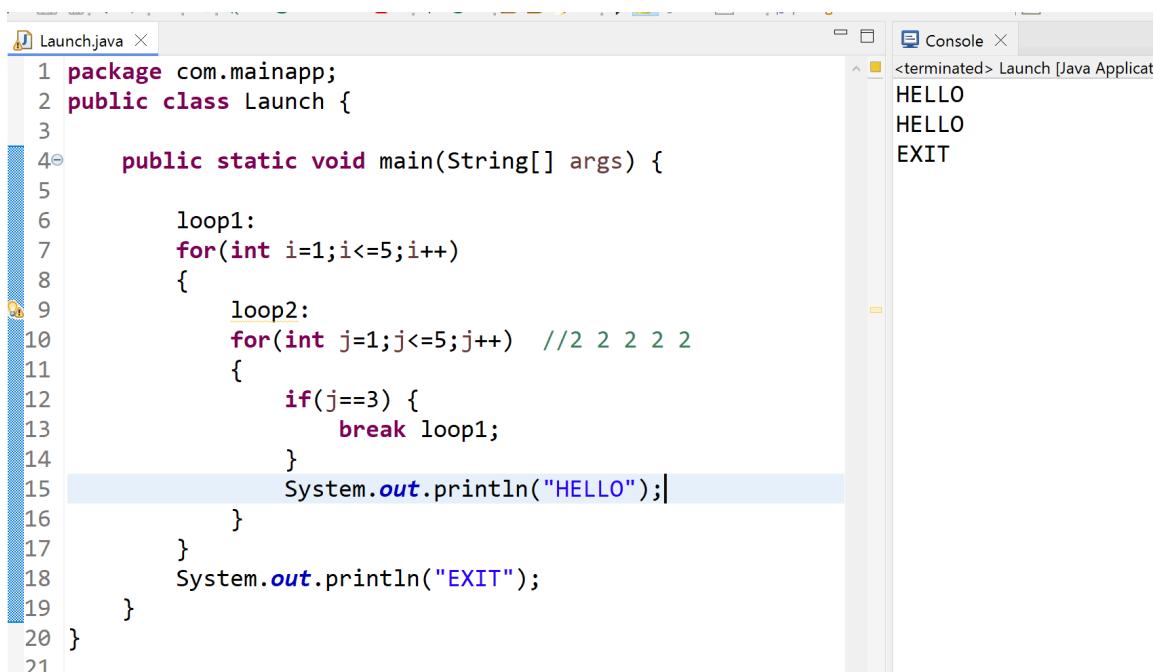
```
Console ×
<terminated> Laun
HELLO
HELLO
EXIT
```

```
Launch.java ×
```

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         for(int i=1;i<=5;i++)
7         {
8             if(i==3) {
9                 continue;
10            }
11            System.out.println("HELLO"+i);
12        }
13
14        System.out.println("EXIT");
15
16    }
17}
```

```
Console ×
<terminated> Launch [
HELLO1
HELLO2
HELLO4
HELLO5
EXIT
```

Labeled for loop:



The screenshot shows a Java development environment with two panes. The left pane, titled 'Launch.java', contains the following code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         loop1:
7             for(int i=1;i<=5;i++)
8             {
9                 loop2:
10                    for(int j=1;j<=5;j++) //2 2 2 2 2
11                    {
12                        if(j==3) {
13                            break loop1;
14                        }
15                        System.out.println("HELLO");
16                    }
17                }
18            System.out.println("EXIT");
19        }
20    }
21 }
```

The right pane, titled 'Console', shows the output of the program:

```
<terminated> Launch [Java Application]
HELLO
HELLO
EXIT
```

Pattern Programming:

Playlist Visit:

Task1: num1<num2

Enter num1: 10 (user input)

Enter num2: 14 (user input)

OUTPUT:

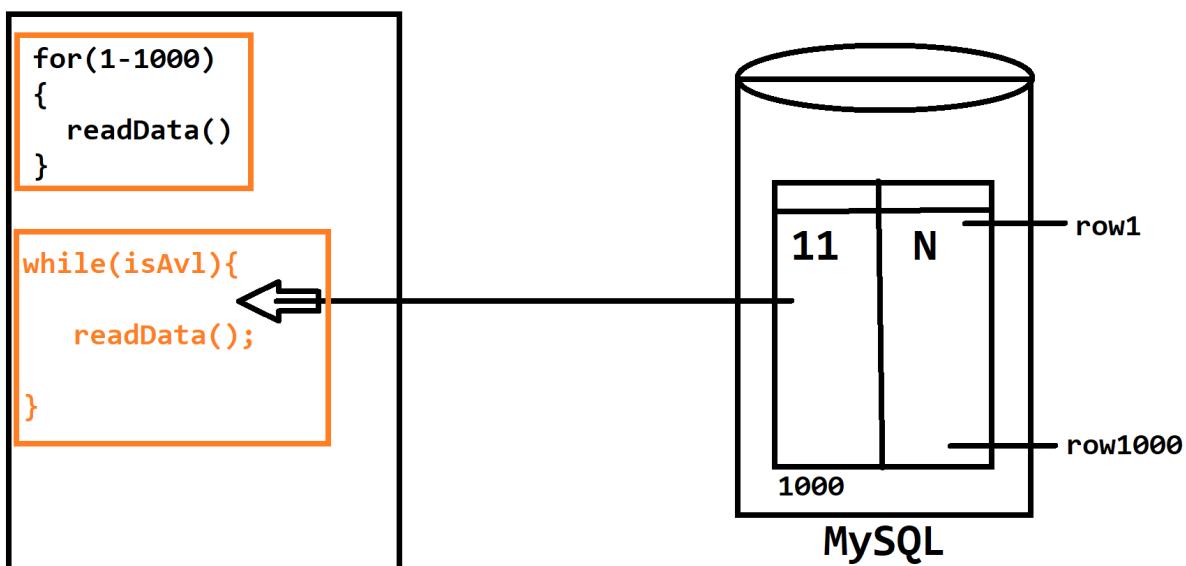
$$10+11+12+13+14=60$$

DAY-17

● Java Full Stack Development Batch
2025

While loop

When you don't know the no. of iteration



```

int i=1;
while(i<=10) {
    System.out.println("HELLO");
}

```

```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int i=1;
7
8         while(i<=10) {
9             System.out.println("HELLO");
10            i++;
11        }
12
13        System.out.println("EXIT");
14    }
15
16

```

<terminated> Launch

HELLO
EXIT

do-While loop

When you don't know the no. of iteration but you want to iterate at least once

```

int i=1;
do {
    System.out.println("HELLO");
    i++; 2
}
while(i<=10);
2 3 4 5 6 7 8 9 10=
9times+1

```

Launch.java

```

1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int i=1;
7
8         do
9         {
10             System.out.println("HELLO"); // 1 times
11             i++;
12         }
13         while(i>=10); //checking from i=2 to i=10=9 times
14
15         System.out.println("EXIT");
16     }
17 }
18

```

DAY-18

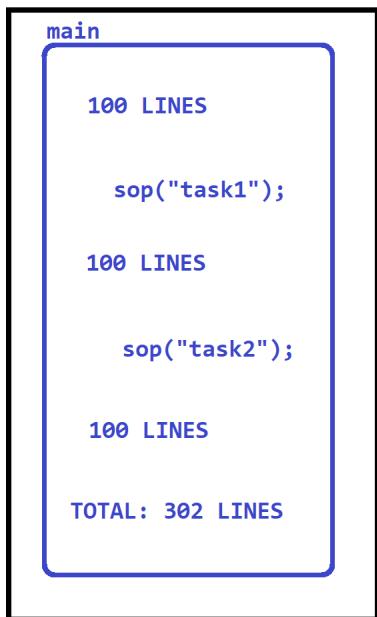
● **Java Full Stack Development Batch
2025**

Methods in Java

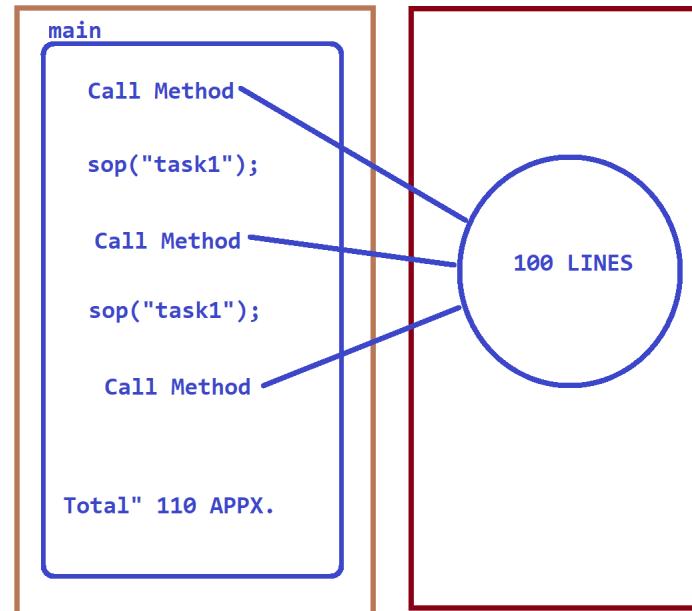
A method in Java is a block of code **designed to perform a specific task.** It is used to achieve code reusability and modularity.

Benefits:

- **code reusability**
- **modularity** (separate)
- Less Line of Code
- Reduces dependency
- Easy to debug
- Maintenance



1. MORE LINE OF CODE (Redundancy)
2. Dependency
3. Messed Up
4. Difficult to Debug
5. Maintainability



100 LINES

Types of Method:

1. Predefined Method/Built-in method/System defined method

-> Provided by Java library (JRE)

2. User Defined Method

-> Created by programmer to perform a specific task Ex. raju()

How to create a method

- Rule: method in Java must be inside a class
- You must call a method to execute its logic

```

Access modifier  Return type  Method Name (Parameters (op)) {
    Method Body
}

```

Access Modifier (from any class we can access this test() method)

```

public void test() {
    //CODE
}

```

This method returns nothing

Method Name
(Identifier)
Convention

Launch.java

```

1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5         System.out.println("HII");
6
7         PracticeMethods p=new PracticeMethods();
8         p.test();
9
10    System.out.println("EXIT");
11
12 }
13
14 }

```

PracticeMethods.java

```

1 package com.mainapp;
2
3 public class PracticeMethods {
4
5     public void test() {
6         System.out.println("TEST1");
7     }
8
9 }
10

```

Console Output:

```

HII
TEST1
EXIT

```

The screenshot shows the Eclipse IDE interface with two code editors and a console window.

Launch.java:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         System.out.println("HII");
7
8         PracticeMethods p=new PracticeMethods();
9         p.test1();
10        p.test2();
11
12        System.out.println("EXIT");
13    }
14 }
15 }
```

PracticeMethods.java:

```
1 package com.mainapp;
2
3 public class PracticeMethods {
4
5     public void test1() {
6         System.out.println("TEST1");
7     }
8     public void test2() {
9         System.out.println("TEST2");
10    }
11
12 }
13 }
```

Console Output:

```
<terminated> Launch [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jst.j.openjdk.hotspot.jre.full.win32.x86_64\jre\bin\java
HII
TEST2
EXIT
```

DAY-19

●Java Full Stack Development Batch 2025

Ways to create a method

- 1.no parameter , no return (Zero parameterized method/No Argument method)
- 2.parameter , no return (Parameterized method)
- 3.no parameter , return

4.parameter , return

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        System.out.println("HII");

        PracticeMethods p=new PracticeMethods();
        //p.test1();
        //p.test2(3);

        //int res=p.test3();
        //System.out.println("RESULT="+res);

        //System.out.println("RESULT="+p.test3());

        int res = p.test4(10, 20, 30);
        System.out.println(res);

        System.out.println("EXIT");

    }
}

package com.mainapp;
public class PracticeMethods {

    //No Parameter , No Return
    //Self Contained Logic
    public void test1() {
        for(int i=1;i<=10;i++)
            System.out.println(5*i);
    }

    //Parameter , No Return
    //When the Logic is based on the value
    //which is coming from outside
    public void test2(int k) {

        for(int i=1;i<=10;i++)
            System.out.println(k*i);
    }
}
```

```
}

//No Parameter , Return
//When we need not to send any value to
//the method but we expect some return Ex. int
a=scanner.nextInt();
public int test3() {

//        return 10;

//        int a=10;
//        return a;

int a=10;
int b=20;
return a+b;

}

//Parameter , Return
//When your logic return something and
//process on external data
public int test4(int a,int b,int c) {

    return a+b+c;
}
}
```

```

Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         System.out.println("HII");
7
8         PracticeMethods p=new PracticeMethods();
9         //p.test1();
10        //p.test2(3);
11
12        p.test3();| 10
13
14        System.out.println("EXIT");
15    }
16
17
PracticeMethods.java
14 //which is coming tr
15 public void test2(in
16
17
18
19
20
21
22 //No Parameter , Ret
23
24 public int test3() {
25
26     return 10;
27
28
29
30
31

```

Task2: WAP to create calculator:

Step 1: create a menu with infinite loop and a break condition

Create Scanner

Create calculator class : Calculator c=new
Calculator();

```

System.out.println("Press-1 : Add");
System.out.println("Press-2 : Sub");
System.out.println("Press-3 : Mul");
System.out.println("Press-4 : Div");
System.out.println("Press-5 : Exit\n");

```

Enter your choice: 1

Case 1:

Enter num1:10 (float)

Enter num2:20 (float)

```
float res = c.add(num1,num2);
Sop(res);
```

Task1Sol: num1<num2

Enter num1: 10 (user input)

Enter num2: 14 (user input)

OUTPUT:

10+11+12+13+14=60

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Num1:");
        int num1=scanner.nextInt(); //10

        System.out.print("Enter Num2:");
        int num2=scanner.nextInt(); //16

        if(num1<num2) {

            int sum=0;
            for(int i=num1 ; i<=num2 ; i++) { //10 11 12 13 14 15
                sum=sum+i;
                System.out.print(i);
                if(i!=num2)
                    System.out.print("+");
            }

            System.out.println("="+sum);
        }
    }
}
```

```

        //i=10    sum=0+10  sum=10
        //i=11    sum=10+11  sum=21
        //i=12    sum=21+12  sum=33
        //i=13    sum=33+13  sum=46
        //i=14    sum=46+14  sum=60
        //i=15    sum=60+15  sum=75
        //i=16    sum=75+16  sum=91

    }
    else {
        System.out.println("num1 must be less than num2");
    }
}
}

```

DAY-20

Java Full Stack Development Batch 2025

Recursion:

It is a programming technique where a method calls itself to solve a problem.

It is commonly used for problem that can be broken down into smaller sub problem of the same type

Ex.

Factorial of a Number:

$$5! = 5 * 4 * 3 * 2 * 1$$

Base Case: A condition that stops the recursion (Always define base case to avoid infinite recursion)

Recursive Case: A call to the Same method

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         PracticeMethods p=new PracticeMethods();
7         p.test();
8
9     }
10 }
11 
```

```
PracticeMethods.java
1 package com.mainapp;
2 public class PracticeMethods {
3
4     int count=1; //INSTANCE VARIABLE(LATER)
5     public void test() {
6         System.out.println("HELLO");
7
8         if(count<=10) {
9             count++;
10            test(); //Recursion
11        }
12    }
13 }
```

```
Console <terminated> Launch [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.openjdk.hotspot.jrepackaged\lib\jre\bin\java -Dfile.encoding=UTF-8 com.mainapp.Launch
HELLO
```

WAP to find factorial of a number using recursion

Factorial of a Number:

$$5! = 5 * 4 * 3 * 2 * 1$$

```

num=5
                                factorial of 1 and 0 = 1
public int test(int num) {
    if(num==1 || num==0)
        return 1;
    return num*test(num-1);
}

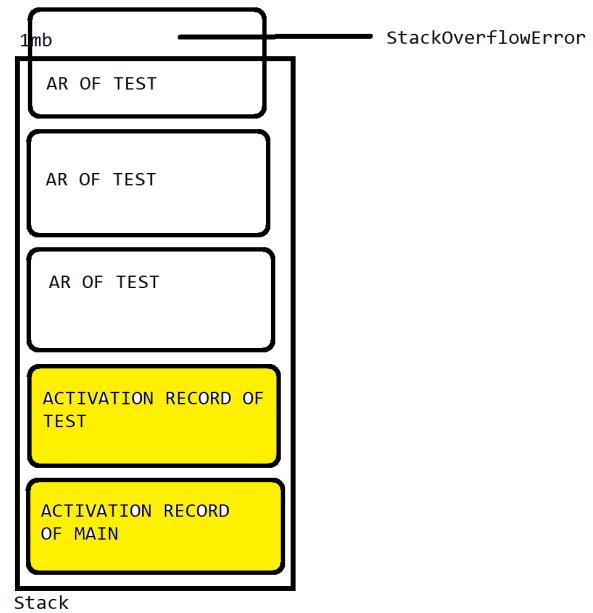
```

5 * test(4)
 4 * test(3)
 3 * test(2)
 2 * test(1)
 1

Why StackOverflowError?

What is Stack?
 Stack is a DataStr which supports FILO/LIFO
 A special type of area inside the RAM (Memory)

start: activation record create
 end: activation record delete



Untick your console limit first

You can increase your stack size (when your base case is high)

-Xss20m : Extra Stack Size 20 mb

Right click on your Launch.java -> run configuration->arguments->vm argument->-Xss20m ->apply->run

DAY-21

Java Full Stack Development Batch 2025

```
int count=1; //INSTANCE VARIABLE(LATER
public void test() {
    System.out.println("HELLO");

    if(count<=10) {
        count++;
        test(); //Recursion
    }
}
```

Types of Variable:

1. Local Variable
2. Instance Variable
3. Static Variable(later)

Local Variable:

->In Java variable which is declared inside a method, constructor or block is called local variable

Hint: outer block (declared)--->We can use it inside inner block

Inner block (declared)--->We cannot use it outside the inner block

->Scope : within the block where it is declared

->You cannot use before initialization because it does not have any default value.

->No need of Access Modifier

```
Launch.java
1 package com.start;
2 import com.mainapp.PracticeMethods;
3 public class Launch {
4
5     public static void main(String[] args) {
6         PracticeMethods p=new PracticeMethods();
7         p.test();
8     }
9 }
10
11 }
12 }

PracticeMethods.java
1 package com.mainapp;
2 public class PracticeMethods {
3
4     public void test() {
5         int a=10;
6
7         int b;
8         if(true) {
9             int c;
10            System.out.println("Value of c is "+c);
11        }
12        System.out.println("Value of b is "+b);
13    }
14 }
15
16
17
18
19
20 }
```

Console output:

```
<terminated> Launch [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jdt.debug\1.8.0_212-e4.7.0.v20180509-1000\jre\bin\java
10
10
```

Access modifier:

->it represent the scope (limit from where we can access)

- public : inside and outside the package

- private : inside the class
- protected : (LATER)
- default (DO NOT USE default KEYWORD): within the package

Instance Variable:

(Object Data, Fields, Data Member)

->In Java variable which is declared outside the method/constructor but inside the class.

->We can use it before initialization (Have some default value)

- Non Decimal : 0
- Decimal : 0.0
- char : black
- boolean: false
- Non Primitive: null

->We must provide access modifier (if we don't give automatically it will become default)

->Scope: depends on its access modifier

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with packages `EmployeeManagementSystem`, `test`, and `JRE System Library [JavaSE-1.8]`. The `src` folder contains `com.mainapp` (with `PracticeMethods.java` and `Test.java`) and `com.start` (with `Launch.java`).
- Launch.java:** Contains the following code:

```
1 package com.start;
2 import com.mainapp.PracticeMethods;
3 import com.mainapp.Test;
4 public class Launch {
5     public static void main(String[] args) {
6         PracticeMethods p=new PracticeMethods();
7         p.test1();
8         p.test2();
9         p.test3();
10        System.out.println(p.count);
11        Test t=new Test();
12        t.test();
13    }
14 }
```
- Test.java:** Contains the following code:

```
1 package com.mainapp;
2 public class Test {
3     public void test() {
4         PracticeMethods p=new PracticeMethods();
5         // int k=p.count;
6         // System.out.println(k);
7         System.out.println(p.count);
8     }
9 }
```
- PracticeMethods.java:** Contains the following code:

```
1 package com.mainapp;
2 public class PracticeMethods {
3     public int count = 1000; // INST
4
5     public void test1() {
6         int roll=19;
7         System.out.println(count);
8     }
9
10    public void test2() {
11        int eid=11;
12        System.out.println(count);
13    }
14
15    public void test3() {
16        int pin=9898;
17        System.out.println(count);
18    }
19 }
20 }
```
- Console:** Displays the output of the application:

```
<terminated> Launch [Java Application] D:\ECLIPSE IDE\...
1000
1000
1000
1000
1000
```

Note:

1. We cannot declare two local variable in a same block but we can initialize a local variable a same block multiple times.
2. We cannot declare two instance variable of same name in a same class.
3. We can create a local variable and a instance variable of same name in a class.

```
1 package com.mainapp;
2
3 public class PracticeMethods {
4
5     private int count = 1000;
6     //private int count = 1000; //CTE
7
8     public void test1() {
9
10         //int k; CTE
11         {
12             int k;
13         }
14
15         int k;
16     }
17 }
18
```

The screenshot shows the Eclipse IDE interface with three main panes:

- Package Explorer:** Shows the project structure with packages EmployeeManagementS, test, JRE System Library [JavaSE-1.8], and src containing com.mainapp (PracticeMethod) and com.start (Launch).
- Launch.java:** Contains the following code:

```
1 package com.start;
2 import com.mainapp.PracticeMethods;
3 public class Launch {
4     public static void main(String[] args) {
5
6         PracticeMethods p=new PracticeMethods();
7         p.test1();
8     }
9 }
10 }
```
- PracticeMethods.java:** Contains the following code:

```
1 package com.mainapp;
2 public class PracticeMethods {
3
4     private int count = 10000;
5
6     public void test1() {
7
8         // int count=1000;
9         // System.out.println(count);
10        // System.out.println(this.count);
11
12         System.out.println(count);
13     }
14 }
15 }
```
- Console:** Shows the output of the application:

```
<terminated> Launch [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.core\src\com\mainapp\PracticeMethods.java
10000
```

DAY-22

Java Full Stack Development Batch 2025

Type Casting:

- Types casting in Java is the process of converting a variable from one data type to another (primitive and non-primitive both)
- There are two main types of type casting

1.WIDENING(Implicit) Type Casting

2.NARROWING(Explicit) Type Casting

1.WIDENING(Implicit) Type Casting

- When converting a smaller data type into larger data type
- Automatic conversion by the compiler
- No data loss because larger types can hold all values of smaller types

byte->short->int->long->float->double

char->int->long->float->double

->No widening conversion for boolean because it is not numeric type

Character Encoding:

->It is way to convert character into its numeric form (because in computer system data travels in digital format)

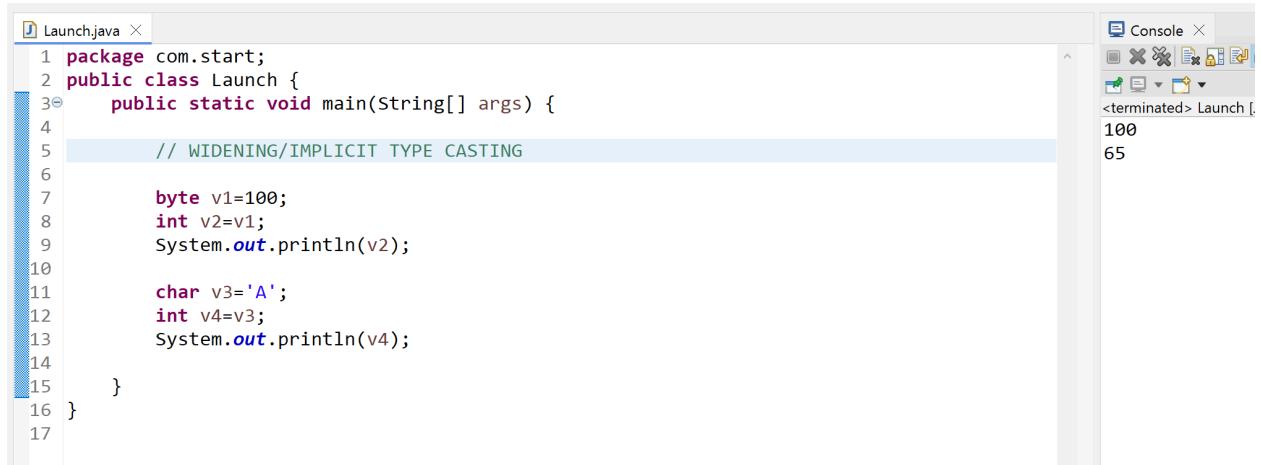
->Java supports UNICODE character encoding.

->Java supports ASCII character encoding as a subset of UNICODE

ASCII: American Standard Code for Information Interchange
(LIMITED TO ENGLISH AND SPECIAL SYMBOLS)

- A-Z (65-90)
- a-z (97-122)

Program: Widening



The screenshot shows an IDE interface with two main panes. The left pane displays the Java code for 'Launch.java'. The code defines a class 'Launch' with a main method. It includes comments for 'WIDENING/IMPLICIT TYPE CASTING' and demonstrates casting from a byte to an int, and from a char to an int. The right pane shows the 'Console' output, which prints the values 100 and 65, corresponding to the byte and char literals respectively.

```
1 package com.start;
2 public class Launch {
3     public static void main(String[] args) {
4
5         // WIDENING/IMPLICIT TYPE CASTING
6
7         byte v1=100;
8         int v2=v1;
9         System.out.println(v2);
10
11         char v3='A';
12         int v4=v3;
13         System.out.println(v4);
14
15     }
16 }
```

Console X
terminated> Launch [
100
65

2. NARROWING(Explicit) Type Casting

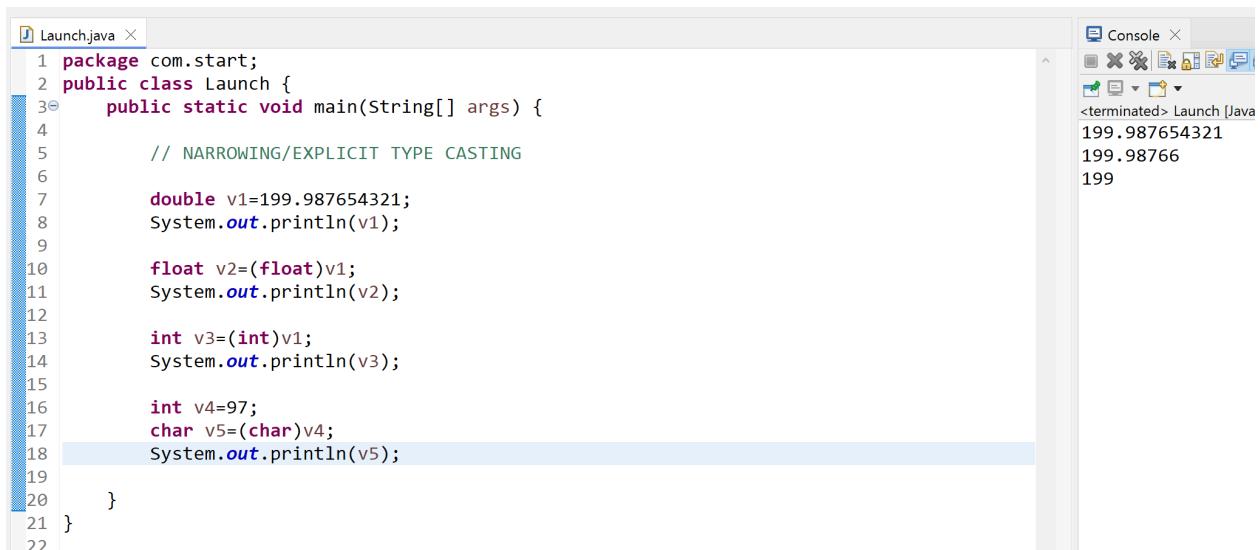
- When converting a larger data type into smaller data type
- Manually we have to convert (Not Automatically)

- Risk of data loss because smaller types cannot hold all values of larger types

->No narrowing conversion for boolean because it is not numeric type

double->float->long->int->short->byte

double->float->long->int->char



The screenshot shows an IDE interface with two main panes. The left pane is titled "Launch.java" and contains the following Java code:

```

1 package com.start;
2 public class Launch {
3     public static void main(String[] args) {
4
5         // NARROWING/EXPLICIT TYPE CASTING
6
7         double v1=199.987654321;
8         System.out.println(v1);
9
10        float v2=(float)v1;
11        System.out.println(v2);
12
13        int v3=(int)v1;
14        System.out.println(v3);
15
16        int v4=97;
17        char v5=(char)v4;
18        System.out.println(v5);
19
20    }
21
22 }
```

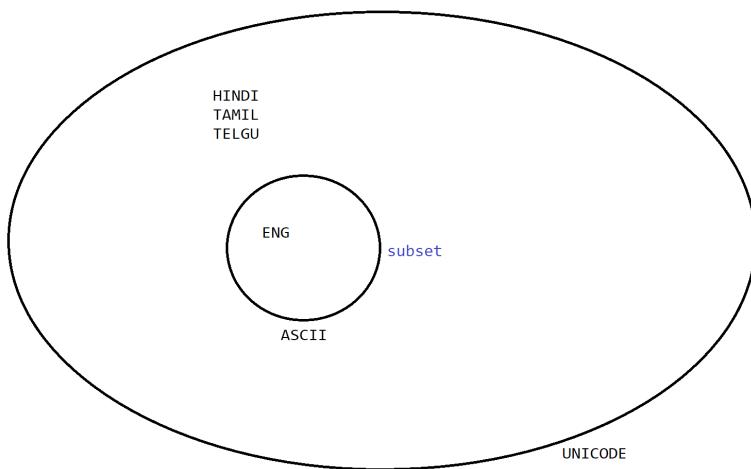
The right pane is titled "Console" and shows the output of the program:

```

<terminated> Launch [Java]
199.987654321
199.98766
199
```

UNICODE CHART:

<https://www.ssec.wisc.edu/~tomw/java/unicode.html>



Operators in Java:

Arithmetic Operators: + - / % ++ --

```

1 package com.start;
2 public class Launch {
3     public static void main(String[] args) {
4
5         int a=11;
6         int b=2;
7         int c=a%b;
8         System.out.println(c);
9
10    }
11 }
12

```

Relational Operators: == != > < <= >=

Logical Operator: && || !

Ternary Operator: It is a short way to write simple if else

The screenshot shows an IDE interface with two main panes. On the left is the code editor for a file named 'Launch.java'. The code contains Java code that prints the value of a conditional expression. On the right is the 'Console' tab, which displays the output of the program: '10'. The code in 'Launch.java' is as follows:

```
1 package com.start;
2 public class Launch {
3     public static void main(String[] args) {
4
5         int a=10;
6         int b=20;
7
8         if(a<b) {
9             System.out.println("a is less than b");
10        }
11        else {
12            System.out.println("b is less than a");
13        }
14
15        // condition ? true : false
16
17        int a=10;
18        int b=20;
19
20        int res= a<b ? a : b ;
21        System.out.println(res);
22
23        int a=10;
24        int b=20;
25
26        System.out.println(a<b ? a : b );
27    }
28 }
```

Java Number System

1.decimal (0-9)

2.binary (0 1)

3.octal (0-7)

4.hexa decimal (0-9 A B C D E F)

The screenshot shows an IDE interface with two main panes. On the left is the code editor for a file named 'Launch.java'. The code demonstrates how to convert integers from different bases to decimal and then print them. On the right is the 'Console' tab, which displays the decimal values of the converted numbers: 110, 6, 72, and 272. The code in 'Launch.java' is as follows:

```
1 package com.start;
2 public class Launch {
3     public static void main(String[] args) {
4
5         int a=110;      //DECIMAL
6
7         int b=0b110;   //BINARY      1*2^2      1*2^1      0*2^0  = 4 +2+0= 6
8         int c=0110;    //OCTAL       1*8^2      1*8^1      0*8^0  = 64 + 8 0 = 72
9         int d=0x110;   //Hexa Decimal 1*16^2     1*16^1      0*16^0 = 256 + 16 + 0=>272
10
11        System.out.println(a);
12
13        System.out.println(b);
14        System.out.println(c);
15        System.out.println(d);
16
17    }
18 }
19 
```

DAY-23

Java Full Stack Development Batch 2025

OOP's Concepts

OOP (Object Oriented Programming)

What is OOP?

- >It is a style of Programming
- >It is a way of Programming
- >It is a Programming Paradigm(Approach)
- >It is a Programming Model
- >It is a Programming methodology
- >It is a Design approach

Style of Programming: procedural Oriented, **Objected Oriented, functional Programming...etc.**

OOP is a way of programming where we relate
concepts through REAL-WORLD entities

OOP's Concepts:

- 1.class
- 2.Object
- 3.Encapsulation
- 4.Inheritance
- 5.Polymorphism
- 6.Abstraction

I want to build a Car:

- 1.Create a Design (BLUEPRINT) : tire,wind,door,engine
- 2.Arrange all parts and fix as per design (OBJECT)
- 3.Give a no to it (IDENTITY)
- 4.Use Car functions(start,accelerate,brake,stop) (BEHAVIOUR)

Object Oriented Programming Language:

->All concepts of OOP with primitive data type (Java)

Object Based Programming Language

->All concepts of OOP except inheritance and polymorphism with primitive data type (EX. VbScript, JavaScript(Before ECMA-6))

Pure Object Oriented Programming Language

->Everything is in the form of Objects(LATER) Ex. SCALA

NOTE: Java is not pure object oriented language because it has primitive data types but it is better than pure object oriented language because it has capability to convert its primitive into Objects through WRAPPER class.

Benefits of OOP:

- >Modularity
- >Reusability
- >Flexibility
- >Security

OOP's Concepts:

Pillar of OOP (Encapsulation , Inheritance, Polymorphism, Abstraction)

class & object:

class is a fundamental building block in OOP

class is used to achieve categorization (Modularity)

when you create a class it wont take any memory space inside the RAM

class is just a blueprint

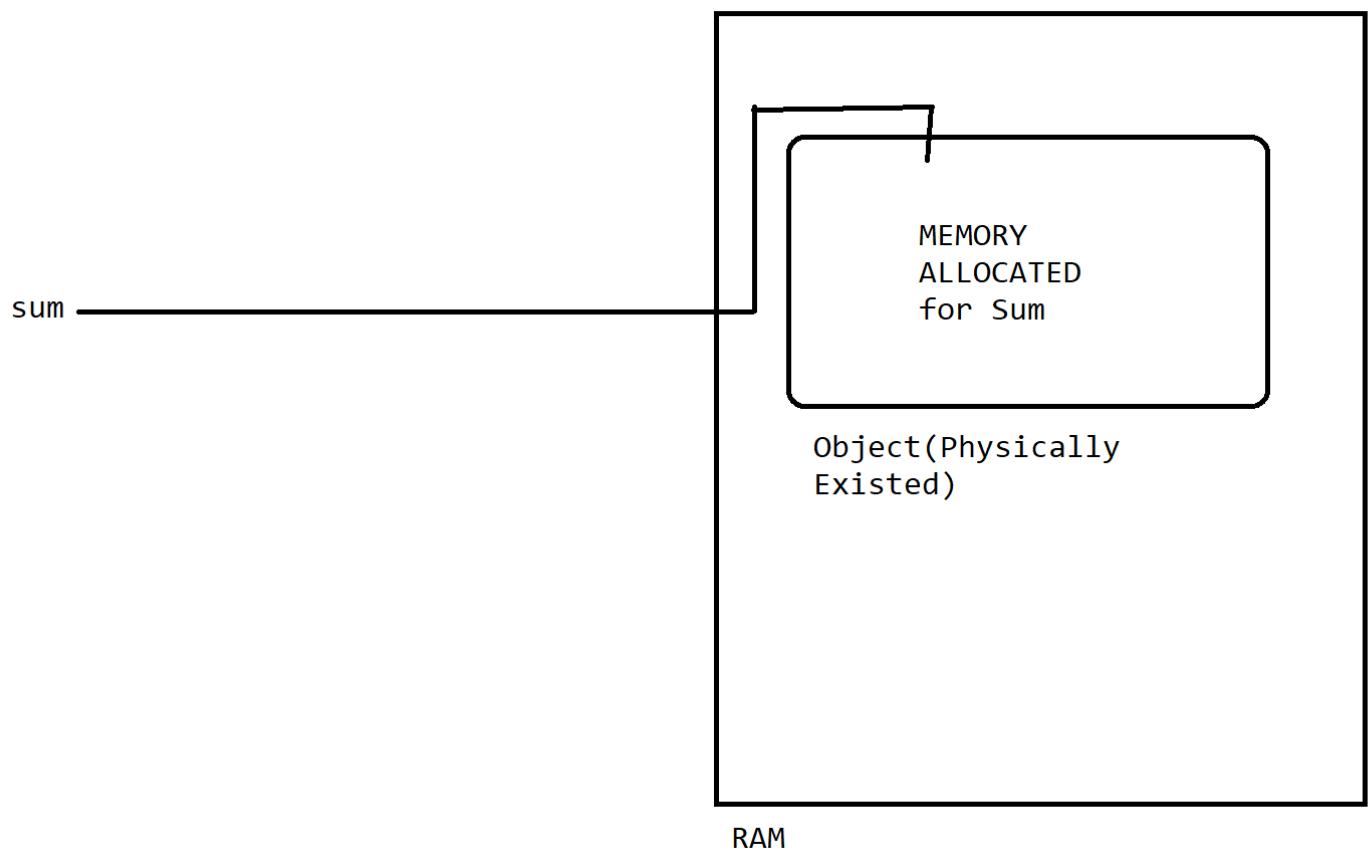
Object is a real time existence of a class inside the memory

Class is also called as user defined Data type

Object is an Instance (Mutra Roop) of a class

- RAM (EXIST) : PHYSICALLY EXISTED : OBJECT
- RAM(NOT EXIST) : PHYSICALLY NOT EXISTED : CLASS (BLUEPRINT)

To create object of a class we use new keyword



The screenshot shows a Java development environment with four code editors open:

- Launch.java**:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5         int a=10;
6
7         Sum sum = new Sum();
8         //We have created object of Sum class
9         //sum: reference variable : Sum Object
10
11        Chappal c;
12        int d=10;
13        float e=10.66f;
14
15    }
16
17 }
18 }
```
- Sum.java**:

```
1 package com.mainapp;
2 public class Sum {
3
4     public void sum() {
5
6         int a=10;
7         int b=20;
8
9         System.out.println(a+b);
10    }
11
12 }
```
- Sub.java**:

```
1 package com.mainapp;
2
3 public class Sub {
4
5 }
6
```
- Chappal.java**:

```
1 package com.mainapp;
2
3 public class Chappal {
4
5 }
6
```

DAY-24

Java Full Stack Development Batch 2025

Object

- 1.Object is an Instance of a class
- 2.Object is the Real-Time Existence of a class inside the RAM
- 3.We use new keyword to create an Object

Characteristics of Objects (CAR)

- STATE: color, speed, milage..etc
- BEHAVIOUR : autoMode, Drifting, Running..etc

- IDENTITY : Car No

Car Object:

- STATE: instance variables/data member/fields
- BEHAVIOUR : methods
- IDENTITY : Car No

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure for "EmployeeManagementSystem" with packages "com.mainapp" and "test".
- Launch.java:** Contains the main method which creates two instances of the "Car" class and prints them to the console.
- Car.java:** Contains the implementation of the "Car" class with methods for running, drifting, and enabling auto mode.
- Console:** Displays the output of the application's execution.

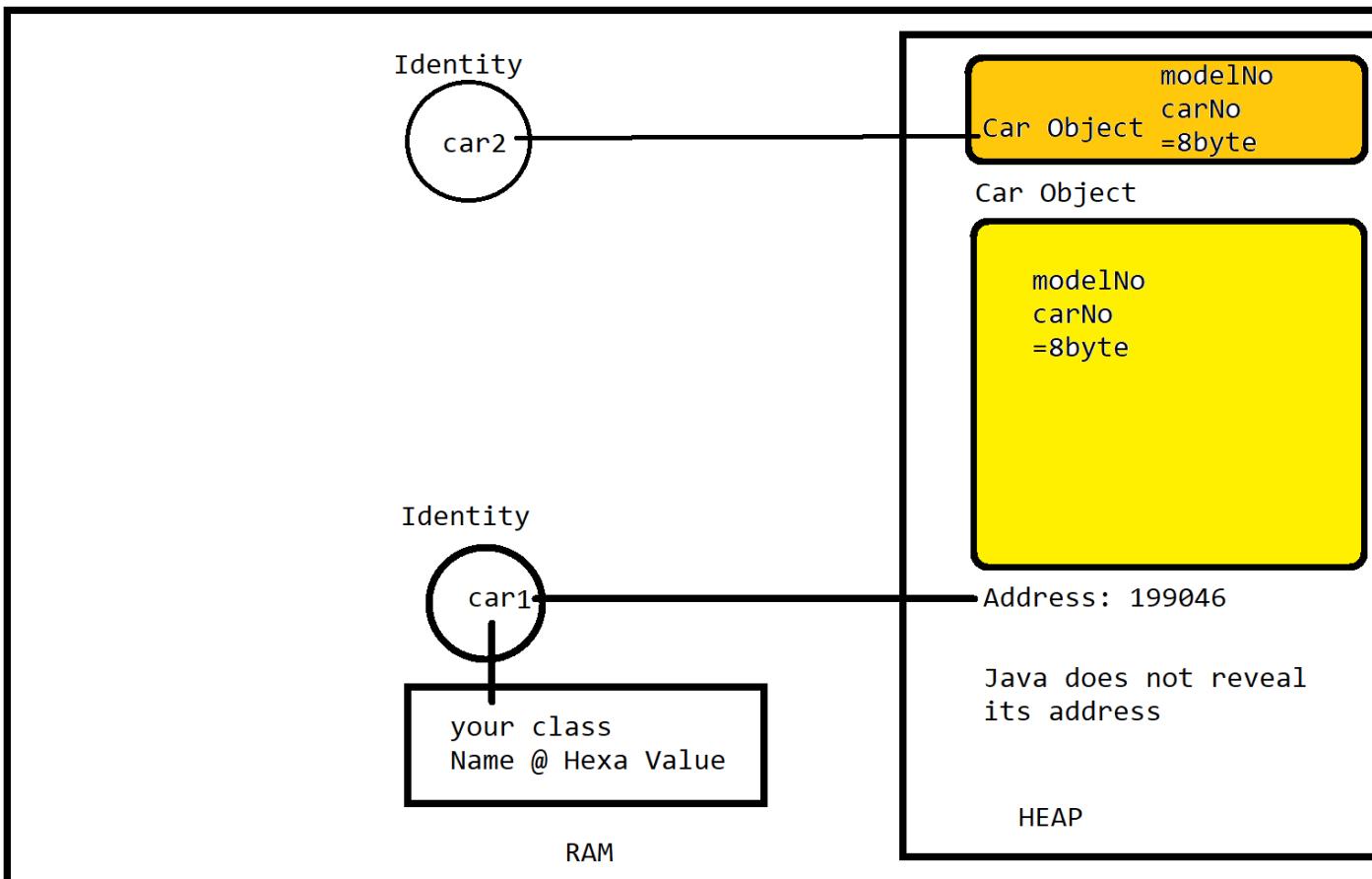
```

Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Car car1=new Car();
7         System.out.println(car1);
8         car1.run();
9
10        Car car2=new Car();
11        System.out.println(car2);
12        car2.run();
13
14    }
}

Car.java
1 package com.mainapp;
2 public class Car {
3
4     private int modelNo=1899;
5     private int carNo=9211;
6
7     public void run() {
8         System.out.println("CAR IS RUNNING"+modelNo+"|"+carNo);
9     }
10
11     public void drift() {
12         System.out.println("CAR IS DRIFTING"+modelNo+"|"+carNo);
13     }
14
15     public void autoMode() {
16         System.out.println("AUTO MODE ENABLED"+modelNo+"|"+carNo);
17     }
18 }

Console
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.core\src\com.mainapp\Car.java:18: error: cannot find symbol
    private int modelNo=1899;
                           ^
  symbol:   variable modelNo
  location: class Car
D:\ECLIPSE IDE\plugins\org.eclipse.jdt.core\src\com.mainapp\Car.java:20: error: cannot find symbol
    private int carNo=9211;
                           ^
  symbol:   variable carNo
  location: class Car
1 error

```



In Java Objects are treated as **First Class Citizen**

- 1.you can hold object inside a variable
- 2.you can pass an object to a method
- 3.you can return an object from a method

The screenshot shows the Eclipse IDE interface with two code editors and a console window.

Launch.java:

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         Scanner scanner = new Scanner(System.in);
8
9         Car car = new Car();
10        car.test(scanner);
11
12        System.out.println("EXIT");
13    }
14}
15}
16}
```

Car.java:

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Car {
4
5     public void test(Scanner scan) {
6
7         System.out.println("ENTER A NUMER");
8         int a=scan.nextInt();
9         System.out.println(a);
10    }
11}
12}
13}
14}
```

Console Output:

```
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.f
ENTER A NUMER
100
100
EXIT
```

DAY-25

Java Full Stack Development Batch 2025

Encapsulation

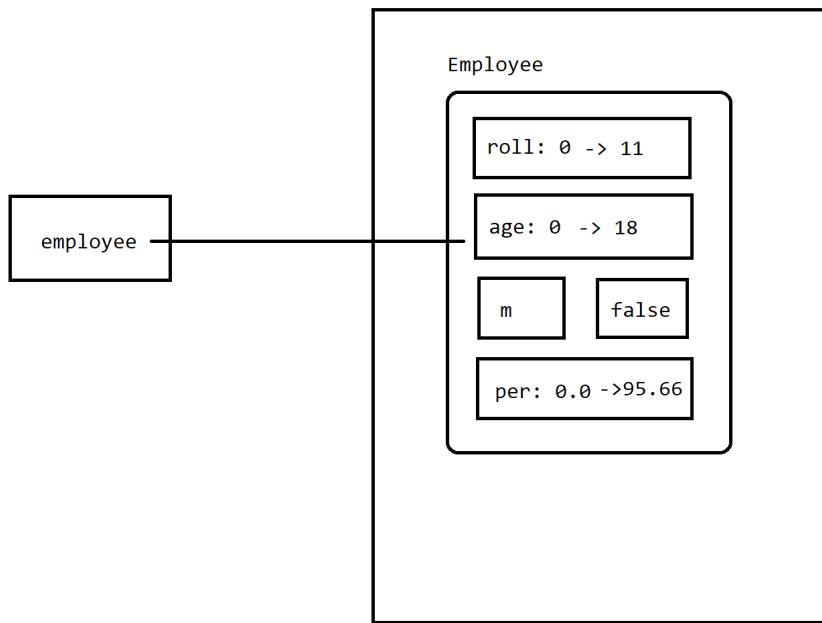
->It is a concept of OOP

->It is used to achieve **data binding** and **data hiding**

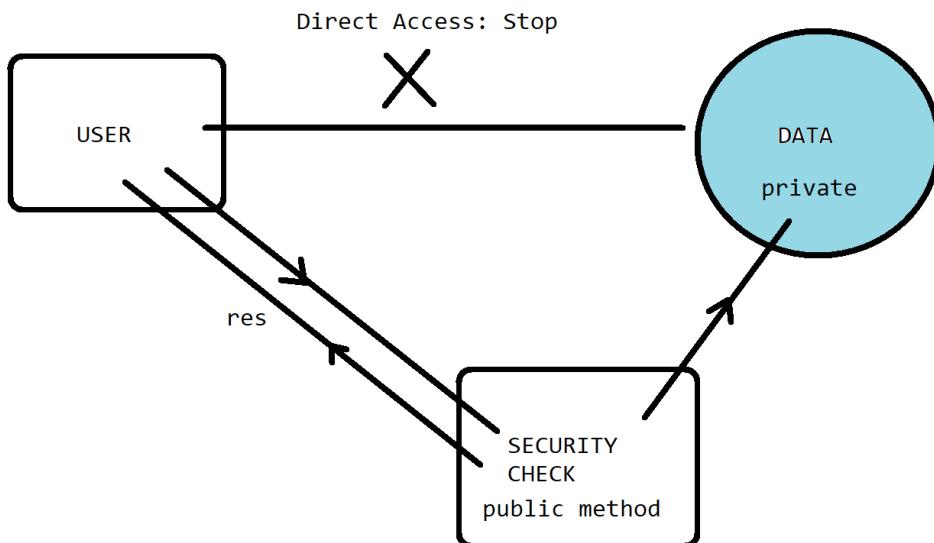
How to implement Encapsulation:

->To achieve Encapsulation you need a separate class

DATA BINDING:



DATA HIDING:



```

package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {
        Car car = new Car();

        Scanner scanner = new Scanner(System.in);

        System.out.println("ENTER ROLL");
        int roll=scanner.nextInt(); //11

        System.out.println("ENTER AGE");
        int age=scanner.nextInt(); //18

        System.out.println("ENTER GENDER");
        char gender=scanner.next().charAt(0); //m

        System.out.println("ENTER PC");
        boolean pc=scanner.nextBoolean(); //false

        System.out.println("ENTER PER");
        float pcmPer=scanner.nextFloat(); //95.66

        Employee employee = new Employee();
        employee.setData(roll,age,gender,pc,pcmPer);

        car.test(employee);
    }
}

package com.mainapp;
import java.util.Scanner;
//DTO : Data Transfer Object
public class Employee {

    private int roll;
    private int age;
    private char gender;
    private boolean pc;
}

```

```

private float pcmPer;

public void setData(int roll,int age,char gender, boolean pc,float pcmPer) {

    Scanner scanner = new Scanner(System.in);
    System.out.println("ENTER KEY");
    int key=scanner.nextInt();
    if(key==1234) {
        this.roll=roll;
        this.age=age;
        this.gender=gender;
        this.pc=pc;
        this.pcmPer=pcmPer;
    }
}

public void printData() {

    Scanner scanner = new Scanner(System.in);
    System.out.println("ENTER KEY");
    int key=scanner.nextInt();
    if(key==1234) {
        System.out.println("ROLL:"+roll);
        System.out.println("AGE:"+age);
        System.out.println("GENDER:"+gender);
        System.out.println("PC:"+pc);
        System.out.println("per:"+pcmPer);
    }
}
}

package com.mainapp;
public class Car {

//500 parameter
public void test(Employee employee)
{
    employee.printData();
}
}

```

=====

```

=====
=====

package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        Car car = new Car();

        Scanner scanner = new Scanner(System.in);

        System.out.println("ENTER ROLL");
        int roll=scanner.nextInt(); //11

        System.out.println("ENTER AGE");
        int age=scanner.nextInt(); //18

        System.out.println("ENTER GENDER");
        char gender=scanner.next().charAt(0); //m

        System.out.println("ENTER PC");
        boolean pc=scanner.nextBoolean(); //false
    }
}

```

```

        System.out.println("ENTER PER");
        float pcmPer=scanner.nextFloat(); //95.66

        Employee employee = new Employee();
        employee.setRoll(roll);
        employee.setAge(age);
        employee.setGender(gender);
        employee.setPc(pc);
        employee.setPcmPer(pcmPer);

        car.test(employee);

    }
}

package com.mainapp;
public class Car {

    //500 parameter
    public void test(Employee employee)
    {
        System.out.println("ROLL:"+employee.getRoll());
        System.out.println("AGE:"+employee.getAge());
        System.out.println("GENDER:"+employee.getGender());
        System.out.println("PC:"+employee.isPc());
        System.out.println("per:"+employee.getPcmPer());

        employee.setPcmPer(77.65f);

        System.out.println("ROLL:"+employee.getRoll());
        System.out.println("AGE:"+employee.getAge());
        System.out.println("GENDER:"+employee.getGender());
        System.out.println("PC:"+employee.isPc());
        System.out.println("per:"+employee.getPcmPer());

    }
}

package com.mainapp;
//DTO : Data Transfer Object
//POJO class: Plain old Java Object
public class Employee {

    private int roll;
    private int age;
    private char gender;
    private boolean pc;
    private float pcmPer;

    //GETTER / ACCESSOR
    public int getRoll() {
        return roll;
    }

    //SETTER / MUTATOR
    public void setRoll(int roll) {
        this.roll = roll;
    }

    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public char getGender() {
        return gender;
    }
    public void setGender(char gender) {
        this.gender = gender;
    }
}

```

```
public boolean isPc() {
    return pc;
}
public void setPc(boolean pc) {
    this.pc = pc;
}
public float getPcmPer() {
    return pcmPer;
}
public void setPcmPer(float pcmPer) {
    this.pcmPer = pcmPer;
}
//alt shift s
}
```

DAY-26

Java Full Stack Development Batch 2025

Constructor

- Constructor is just a block of code like a method
- Constructor is used to initialize the Object(Data Member/Fields/Instance Variable Initialization)
- You need not to call constructor but it will get called automatically by JVM to perform initialization
- JVM call constructor just after object Creation
- You can have multiple constructor in a class
- If you don't have created a constructor in a class then in such case JVM will create one Default Constructor (NO ARG CONSTRUCTOR)

Rules of Constructor:

1. constructor must be inside the class

2.constructor name must be same as class name(SENSITIVE)

3.constructor has no Return Type

Note: Constructor is used to initialize the object (Called Once for an object) while setter is used to update the initialized value (Can be called any no times)

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        Employee employee = new Employee(10,18,10.9f);
        employee.test();

        employee.setAge(20);

        employee.test();

    }
}
```

```

package com.mainapp;
public class Employee {

    private int id;
    private int age;
    private float avg;

    public Employee(int id,int age,float avg) {
        this.id=id;
        this.age=age;
        this.avg=avg;
    }

    public void test() {
        System.out.println("THIS IS MY METHOD");
        System.out.println(id+"|"+age+"|"+avg);
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setAvg(float avg) {
        this.avg = avg;
    }
}

```

Constructor Chaining:

- **this vs this()**
- **this(); “this constructor call” : it is used to call current class constructor**

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5         Employee employee = new Employee(10);
6     }
7
8 }
9
10
```

```
Employee.java
1 package com.mainapp;
2 public class Employee {
3
4     public Employee(int id, int age) {
5         this(10, 20);
6     }
7
8     public Employee() {
9         System.out.println("Employee created");
10    }
11
12    public Employee() {
13        System.out.println("Employee created");
14    }
15
16 }
17
18 }
```

RULES OF `this()`:

- 1.it must be the first statement inside the constructor
- 2.`this()` is only applicable inside the constructor

TASK:

- Employee class : id age avg gender
- Initialize all data through a constructor (USER INPUT)
- Print all data using getters
- Update avg and age using setter (USER INPUT)
- Print all data using getters

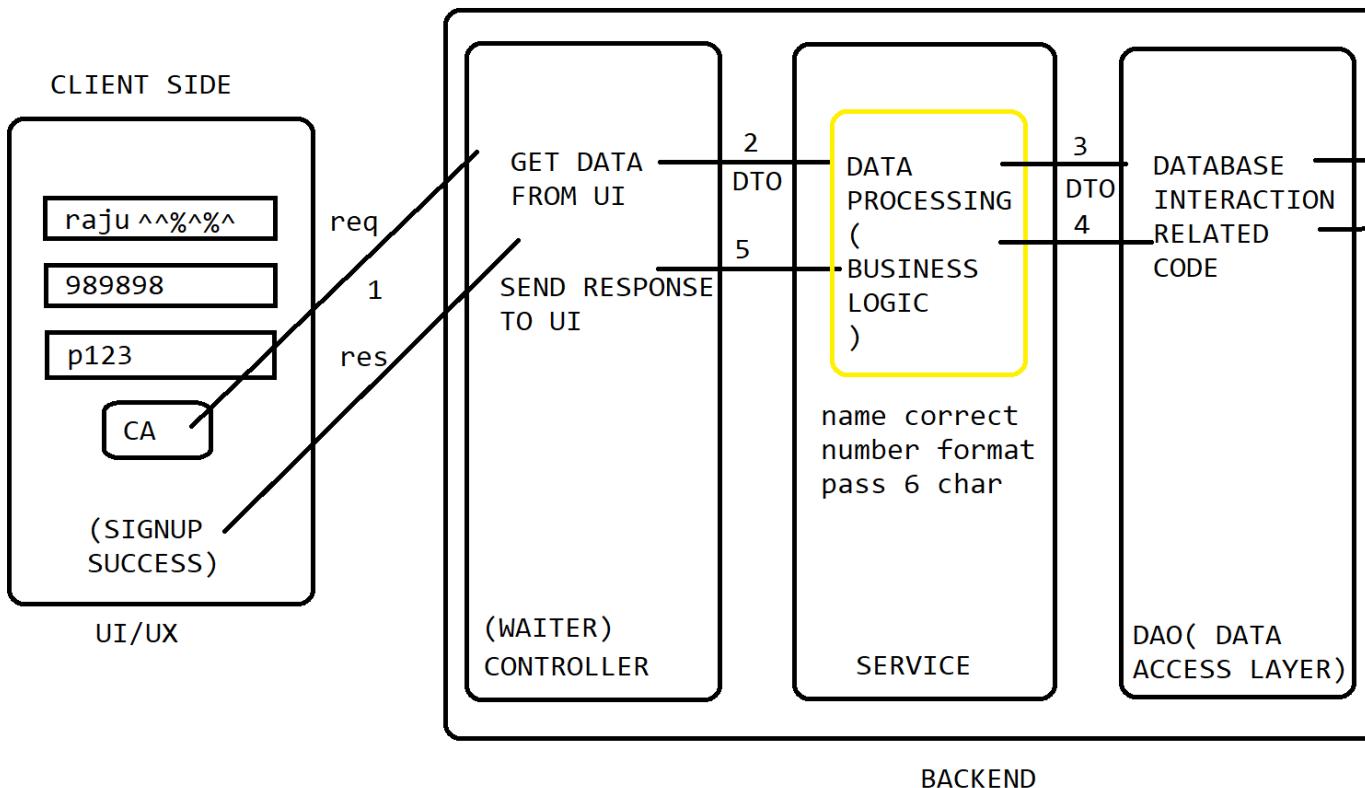
DAY-27

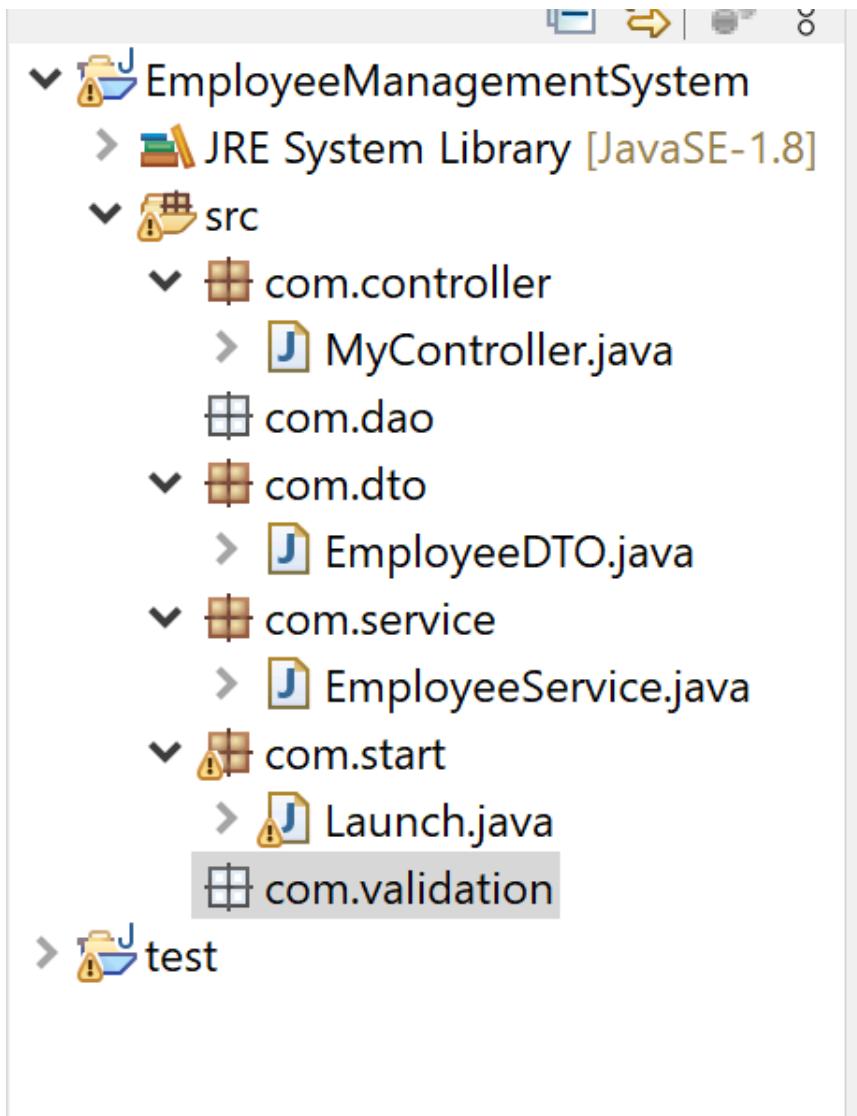
Java Full Stack Development Batch 2025

How to work on project (Individually)

Enterprise applications(LARGE SCALE APPLICATION) are categorized into layers

1. CONTROLLER LAYER : it interacts with UI (get data from UI, send response to UI)
2. SERVICE LAYER: it is used to write business logic(Data Processing/**Data Validation**)
3. DATA ACCESS OBJECT LAYER: it is used to write code related to DATABASE interaction
4. ALWAYS PASS DATA FROM ONE LAYER TO ANOTHER IN THE FORM DTO





```
package com.start;
import java.util.Scanner;
import com.controller.MyController;
import com.dto.EmployeeDTO;

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

        //CLIENT SIDE

        System.out.println("*****WELCOME TO EMPLOYEE MANAGEMENT
SYSTEM*****\n");
        MyController controller = new MyController();

        while (true) {
```

```

System.out.println("Press-1 : Add Employee");
System.out.println("Press-2 : Read All Employee");
System.out.println("Press-3 : Delete Single
Employee");
System.out.println("Press-4 : Update Single
Employee");
System.out.println("Press-5 : Exit\n");

Scanner scanner = new Scanner(System.in);
System.out.print("Enter Your Choice: ");
int choice = scanner.nextInt();

if(choice==5) {
    System.out.println("THANKS FOR USING");
    break;
}

switch (choice) {

case 1:
    System.out.println("Add Employee\n");

    System.out.println("ENTER EID");
    int eid=scanner.nextInt();

    System.out.println("ENTER EAGE");
    int eage=scanner.nextInt();

    System.out.println("ENTER EPINCODE");
    int epincode=scanner.nextInt();

    EmployeeDTO edto = new EmployeeDTO(eid, eage,
epincode);
    int res=controller.addEmployee(edto);
    if(res==100) {
        System.out.println("DATA INSERTED
SUCCESSFULLY");
    }
    else {
        System.out.println("SOMETHING WENT WRONG");
    }
    break;

case 2:
    System.out.println("Read All Employee\n");
}

```

```

        break;

    case 3:
        System.out.println("Delete Single Employee\n");
        break;

    case 4:
        System.out.println("Update Single Employee\n");
        break;

    default:
        System.out.println("Wrong Choice\n");
        break;
    }

    //end of switch

}

//end of while

}

//end of main
}
package com.controller;
import com.dto.EmployeeDTO;
import com.service.EmployeeService;

public class MyController {

    private EmployeeService employeeService;

    public MyController() {
        employeeService=new EmployeeService();
    }

    public int addEmployee(EmployeeDTO edto) {

        int res= employeeService.addEmployee(edto);
        return res;
    }
}
package com.service;
import com.dto.EmployeeDTO;

public class EmployeeService {

    public int addEmployee(EmployeeDTO edto) {

        //DATA PROCESSING:
    }
}

```

```

        int res=validateEmployee(edto);
        if(res==100) {
            //DAO LAYER : LATER
            return res;//JUST TO GET RID OF ERROR
        }
        else {
            return res;
        }
    }

public int validateEmployee(EmployeeDTO edto)
{
    if(edto.getEid()<1 || edto.getEid()>99) {
        return 4041;
    }
    else if(edto.getEage()<18 || edto.getEage()>40) {
        return 4042;
    }
    else if(edto.getEpicode()<100000 || edto.getEpicode()>999999)
    {
        return 4043;
    }
    else {
        return 100;
    }
}
}

package com.dto;
public class EmployeeDTO {

    private int eid;
    private int eage;
    private int epicode;

    public EmployeeDTO(int eid, int eage, int epicode) {
        this.eid = eid;
        this.eage = eage;
        this.epicode = epicode;
    }

    public int getEid() {
        return eid;
    }

    public void setEid(int eid) {

```

```
        this.eid = eid;
    }

    public int getEage() {
        return eage;
    }

    public void setEage(int eage) {
        this.eage = eage;
    }

    public int getEpicode() {
        return epicode;
    }

    public void setEpicode(int epicode) {
        this.epicode = epicode;
    }
}
```

DAY-28

Java Full Stack Development Batch 2025

this keyword

- it provides current class object
- Generally it is used to differentiate between local & instance variable
- You can use this keyword inside both method and constructor

```
Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Employee employee = new Employee();
7         employee.test();
8
9     }
10}
11
```

```
Employee.java ×
1 package com.mainapp;
2 public class Employee {
3
4     private int a=1000;
5
6     public void test(Employee employee) {
7
8         int a=100;
9
10        System.out.println(a);
11        System.out.println(employee.a);
12
13        //test: same object
14    }
15
16}
17
18
```

```
Console ×
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.core\src\com\mainapp\Launch.java
100
1000
```

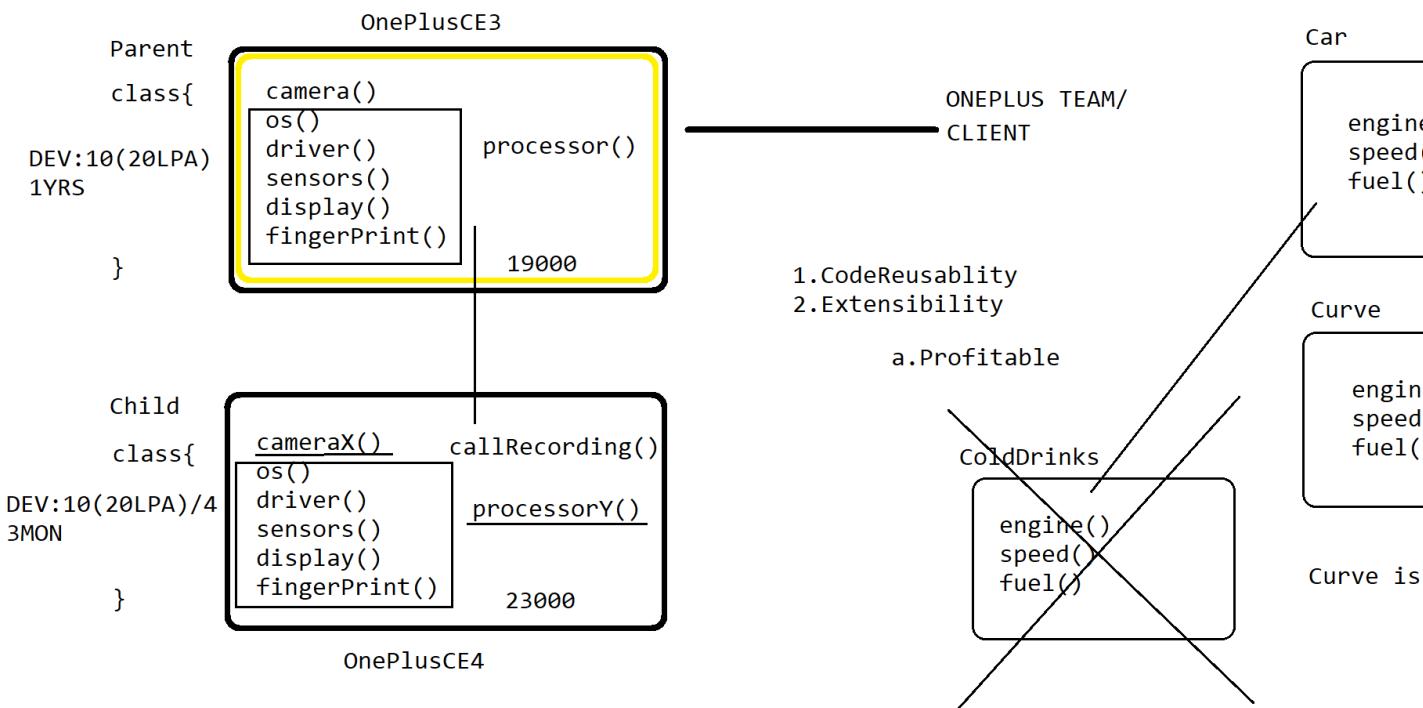
```
Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Employee employee = new Employee();
7         employee.test();
8
9     }
10}
11
```

```
Employee.java ×
1 package com.mainapp;
2 public class Employee {
3
4     private int a=1000;
5
6     public void test() {
7
8         int a=100;
9
10        System.out.println(a);
11        System.out.println(this.a);
12
13    }
14
15}
```

```
Console ×
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.core\src\com\mainapp\Launch.java
100
1000
```

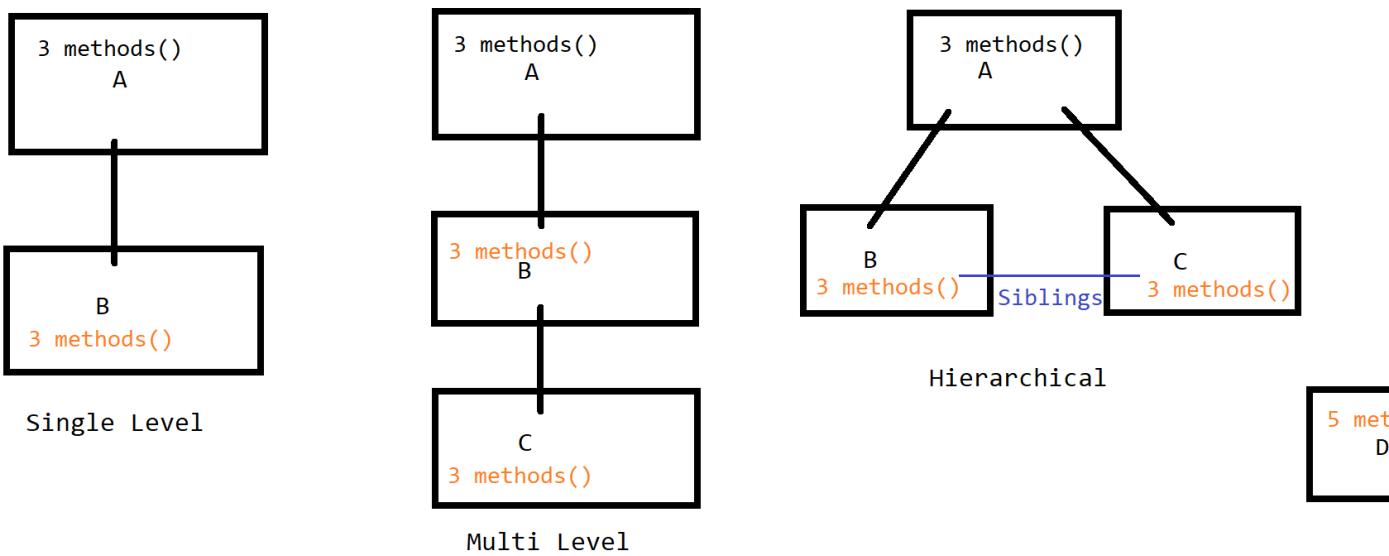
Inheritance in Java(Pillar of OOP)

- >The concept Inheritance is also taken from the REAL-WORLD
- >In java inheritance is a mechanism in which **one Object** can inherit all the methods and attributes(instance variable) of a **parent Object**
- >In java inheritance is a mechanism in which one Object can inherit properties (instance variable) and behaviors of a parent Object
- >Child class **is a type of the Parent class**
- >Inheritance is also called **IS-A** relationship



- Parent Child
- SuperClass Subclass
- BaseClass DerivedClass

Types of Inheritance:



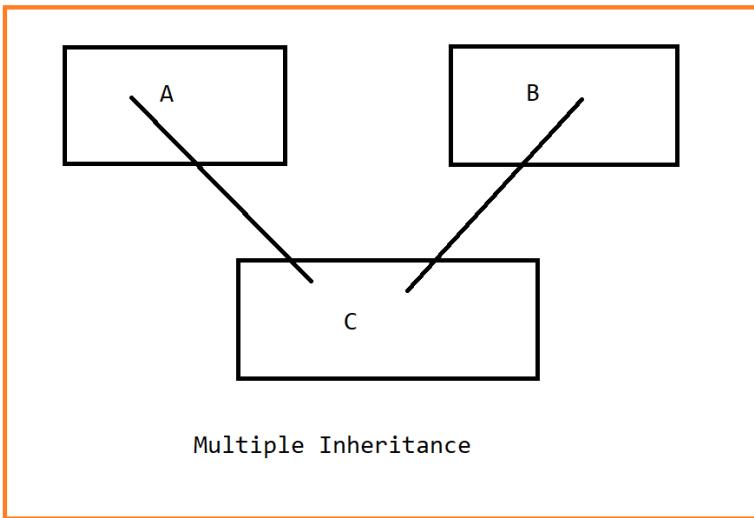
DAY-29

Java Full Stack Development Batch 2025

- In java we can achieve Inheritance through **class and interface**
- When we achieve inheritance through class then we cannot achieve multiple inheritance but it is possible through interface

Multiple Inheritance:

We cannot achieve this through class but through interface we can



In java we use `extends` keyword to achieve inheritance

- extend
 - **extends**
 - extent
 - extents
-
- When you create object of child class in java then automatically parent class will get instantiated
 - For a valid Inheritance child class constructor must call parent class constructor because parent class

need to be instantiated so that child object can inherit parent object

super()

->this() is also a constructor call
constructor call

->this() should be first statement
be first statement

->this() is allowed only inside constructor
allowed only inside the constructor

->it is used to call constructor
constructor

->same class constructor
constructor

->super() is a

-> super() should

->super() is

->it is used to call

->parent class

Eclipse IDE screenshot showing three Java files and their output in the console.

Launch.java

```

1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.speed();
8         bmw.milage();
9         bmw.color();
10
11    }
12 }

```

Car.java

```

1 package com.mainapp;
2 public class Car {
3
4     public Car(int a) {
5         System.out.println("PCC");
6     }
7
8     public void speed() {
9         System.out.println("SPEED: 100km/h");
10    }
11
12     public void milage() {
13         System.out.println("MILAGE: 10km/L");
14    }
15
16     public void color() {
17         System.out.println("BLACK");
18    }
19 }

```

Bmw.java

```

1 package com.mainapp;
2 public class Bmw extends Car {
3
4     public Bmw() {
5         super(10);
6         System.out.println("CCC");
7     }
8 }

```

Console Output

```

<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\eclipse\plugins\org.eclipse.jst.jdt.openjdk.hc
PCC
CCC
SPEED: 100km/h
MILAGE: 10km/L
BLACK

```

```

public class Car {

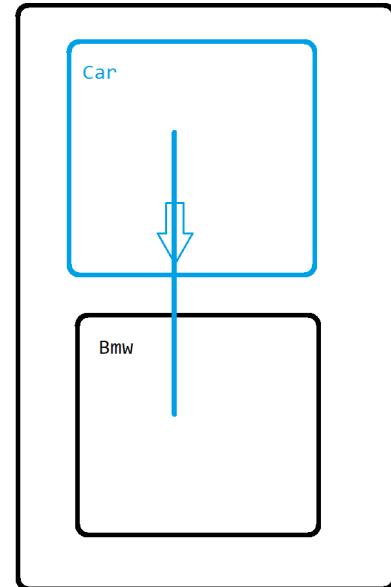
    public void speed() {
        System.out.println("SPEED: 100km/h");
    }

    public void milage() {
        System.out.println("MILAGE: 10km/L");
    }

    public void color() {
        System.out.println("BLACK");
    }
}

public class Bmw extends Car {
}

```



```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.speed();
8         bmw.milage();
9         bmw.color();
10
11    }
12
13 }
14

Car.java ×
1 package com.mainapp;
2 public class Car {
3
4     public Car() {
5         System.out.println("PCC");
6     }
7
8     public void speed() {
9         System.out.println("SPEED: 100km/h");
10    }
11
12     public void milage() {
13         System.out.println("MILAGE: 10km/L");
14    }
15
16     public void color() {
17         System.out.println("BLACK");
18    }
19
20 }

Bmw.java ×
1 package com.mainapp;
2 public class Bmw extends Car {
3
4     public Bmw() {
5         System.out.println("CCC");
6     }
7
8     public void speed() {
9         System.out.println("SPEED: 100km/h");
10    }
11
12     public void milage() {
13         System.out.println("MILAGE: 7km/L");
14    }
15
16     public void color() {
17         System.out.println("BLACK");
18    }
19
20     public void aiFeatures() {
21         System.out.println("AUTOMODE");
22    }
23
24 }
25

```

The diagram illustrates the inheritance relationship between the three classes. A blue oval surrounds the `Car` and `Bmw` classes, indicating they inherit from the same base class. Blue arrows point from the `bmw` object in the `Launch` code to the `speed()`, `milage()`, and `color()` methods in the `Car` class, and from the `Bmw` constructor to its own `speed()`, `milage()`, `color()`, and `aiFeatures()` methods.

Import terms in inheritance: inherited, overridden, specialized method

```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.speed();
8         bmw.milage();
9         bmw.color();
10        bmw.aiFeatures();
11
12    }
13
14 }

Car.java × Bmw.java ×
1 package com.mainapp;
2 public class Bmw extends Car {
3
4     //AAYA HAI : Inherited Method -> speed color
5
6     @Override //Optional
7     public void milage() { //OVERRIDDEN METHOD
8         System.out.println("MILAGE: 7km/L");
9     }
10
11     //SPECIALIZED METHOD
12     public void aiFeatures() {
13         System.out.println("AUTOMODE");
14     }
15
16 }
17

```

The diagram shows the same code structure as the previous one, but with annotations. The `milage()` method in `Bmw` is annotated with `//OVERRIDDEN METHOD`. The `aiFeatures()` method in `Bmw` is annotated with `//SPECIALIZED METHOD`.

final keyword in Java:

- class : you can't extends
- method : you cant override
- variable : you cant update

- private : cant inherit cant override
- final : can inherit cant override
- public: can inherit can inherit

The screenshot shows the Eclipse IDE interface with four open files:

- Launch.java**:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.speed();
8         bmw.milage();
9         //bmw.color();
10    }
11 }
```
- Car.java**:

```
1 package com.mainapp;
2 public class Car {
3
4     public void speed() {
5         System.out.println("SPEED: 100km/h");
6     }
7     public final void milage() {
8         System.out.println("MILAGE: 10km/L");
9     }
10    private void color() {
11        System.out.println("BLACK");
12    }
13 }
```
- Bmw.java**:

```
1 package com.mainapp;
2 public class Bmw extends Car {
3
4
5     // public void milage() { CTE
6     //     System.out.println("MILAGE: 1km/L");
7     // }
8
9 }
```
- Console**:
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.java.core
SPEED: 100km/h
MILAGE: 7km/L
BLACK
AUTOMODE

```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         // final int a=10;
7         // a=20; CTE
8
9         Bmw bmw = new Bmw();
10
11
12     }
13 }
```



```

Bmw.java ×
1 package com.mainapp;
2 public class Bmw {
3
4     private final int a=20;
5
6     // public void setA(int a) {
7     //     this.a=a; CTE
8     // }
9
10
11
12 }
13
14 }
```



```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         // final int a=10;
7         // a=20; CTE
8
9         Bmw bmw = new Bmw(10);
10
11
12     }
13 }
14 }
```



```

Bmw.java ×
1 package com.mainapp;
2 public class Bmw {
3
4     private final int a;
5
6     public Bmw(int a) { //INITIALIZE
7         this.a = a;
8     }
9
10 // public void setA(int a) { //UPDATE
11 //     this.a = a;
12 // }
13
14 }
15 }
```

DAY-30

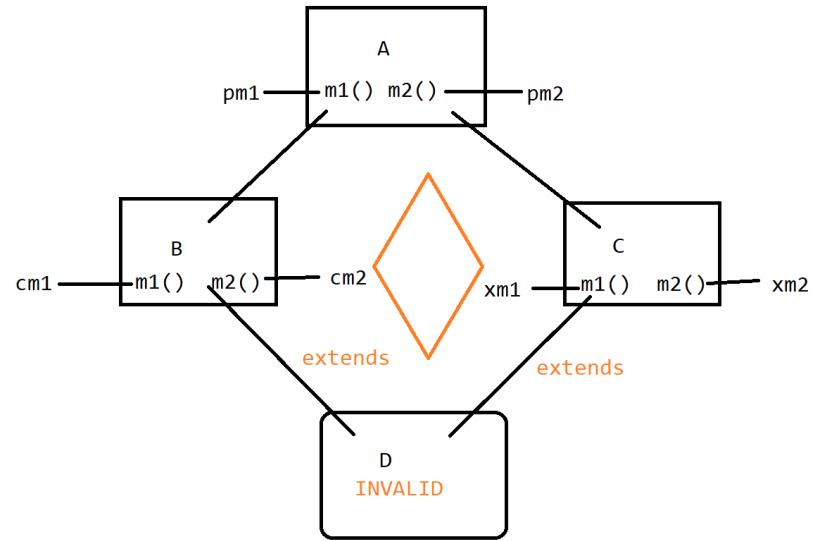
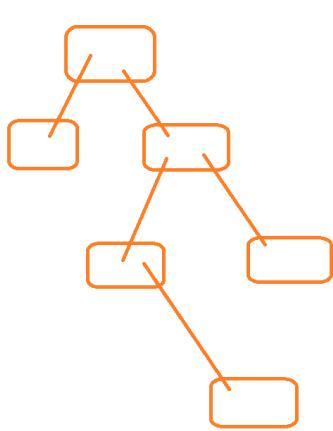
Java Full Stack Development Batch 2025

1.CAN WE INHERIT CONSTRUCTOR ?

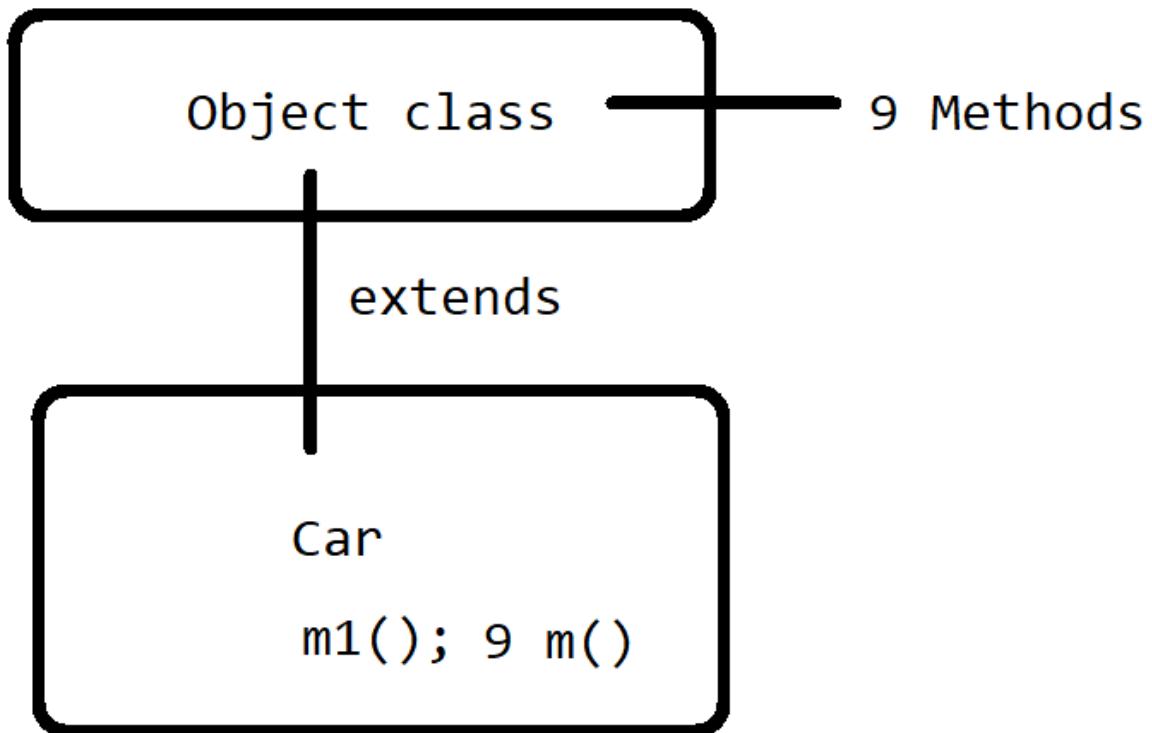
->NO but you must call parent class constructor from the child class

2.WHY MULTIPLE INHERITANCE IS NOT POSSIBLE THROUGH CLASS?

->it leads to DIAMOND SHAPE PROBLEM



Note: In java every individual class extends Object



The screenshot shows the Eclipse IDE interface with two open files:

- Launch.java** (Left):

```

1 package com.mainapp;
2 import java.lang.String;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         Car car1 = new Car();
8         Car car2 = new Car();
9         //System.out.println(car.getClass());
10
11         if(car1.equals(car2)) {
12             System.out.println("EQ");
13         }
14         else {
15             System.out.println("NEQ");
16         }
17     }
18 }
19
20

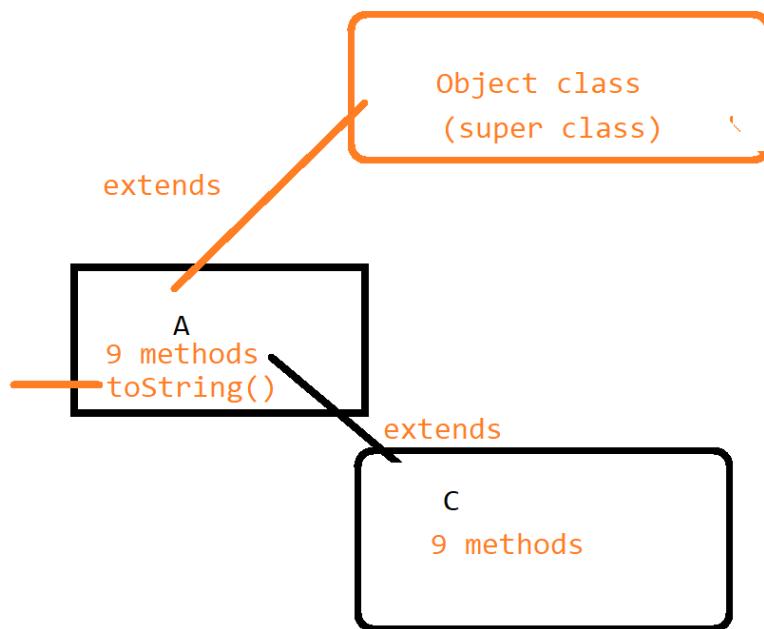
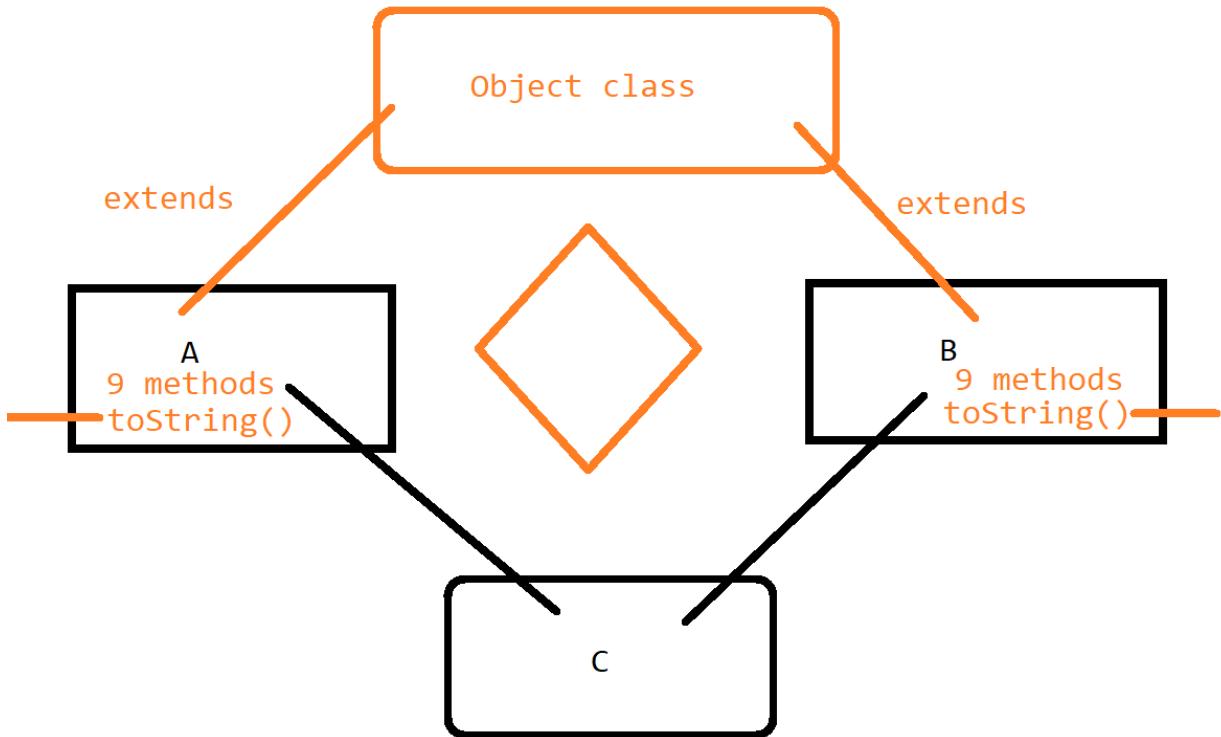
```
- Car.java** (Right):

```

1 package com.mainapp;
2 public class Car extends Object {
3
4     public void m1() {
5         System.out.println("PM1");
6     }
7 }
8

```

At the bottom, the "Console" tab is active, showing the output: "NEQ".



```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         new Bmw();
7
8     }
9 }
10

Car.java ×
1 package com.mainapp;
2 public class Car extends Object {
3
4     // public Car() {}
5     // super();
6     //
7
8     public void m1() {
9         System.out.println("PM1");
10    }
11 }
12

Bmw.java ×
1 package com.mainapp;
2
3 public class Bmw extends Car {
4
5     // public Bmw() {}
6     // super();
7     //
8
9 }
10

```

super keyword

->it provides parent class object

```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.m1();
8     }
9 }
10

Car.java ×
1 package com.mainapp;
2 public class Car extends Object {
3
4     public int a=10;
5
6     public void m1() {
7         System.out.println("PM1");
8     }
9 }
10

Bmw.java ×
1 package com.mainapp;
2 public class Bmw extends Car {
3
4     private int a=100;
5
6     @Override
7     public void m1() {
8         int a=1000;
9         System.out.println("CM1");
10        System.out.println(a);
11        System.out.println(this.a);
12        System.out.println(super.a);
13    }
14 }
15

```

The screenshot shows the Eclipse IDE interface with three code editors open:

- Launch.java**:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Bmw bmw = new Bmw();
7         bmw.m1();
8     }
9
10}
```
- Car.java**:

```
1 package com.mainapp;
2 public class Car extends Object {
3
4     public void m1() {
5         System.out.println("PM1");
6     }
7
8}
```
- Bmw.java**:

```
1 package com.mainapp;
2 public class Bmw extends Car {
3
4
5     @Override
6     public void m1() {
7         super.m1();
8         System.out.println("CM1");
9     }
10}
```

Below the code editors is a **Console** tab with the following output:
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.op
PM1
CM1

DAY-31

Java Full Stack Development Batch 2025

Polymorphism(POLY: MANY , MORPHISM: FORM)

- In java if a single statement can perform many task then we can say we have achieve polymorphism
- It reduces the line of code

It has two types

1.Compile time polymorphism

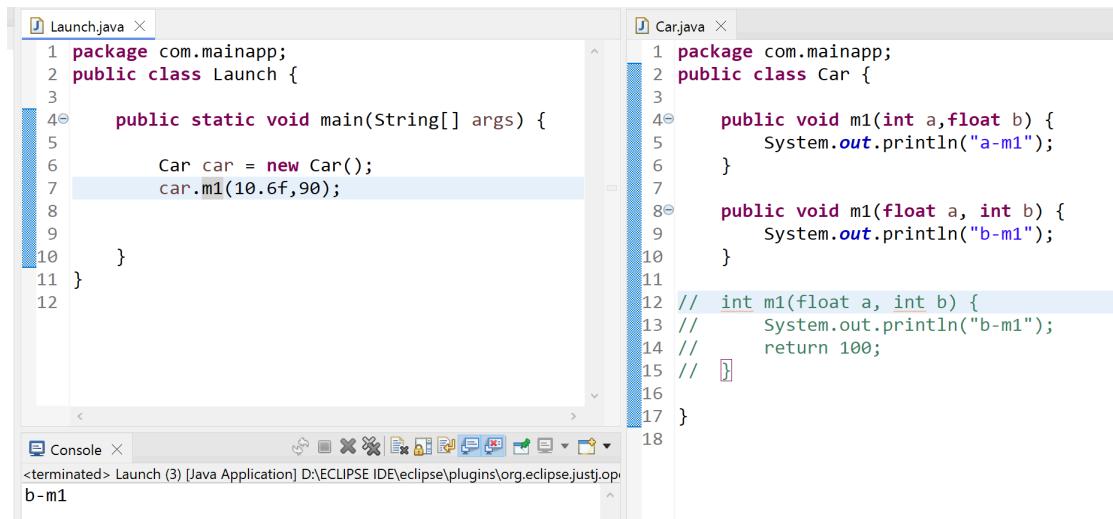
- Through method **Overloading**

2.Runtime polymorphism

- Through method Overriding

COMPILE TIME POLYMORPHISM: (Early Binding/Compile Time Binding/Static Polymorphism/Static Binding)

- Method Overloading: it is a technique in which we create more than one method with same name in a same class
- It occurs at compile time



```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Car car = new Car();
7         car.m1(10.6f,90);
8
9     }
10 }
11 }

Car.java ×
1 package com.mainapp;
2 public class Car {
3
4     public void m1(int a,int b) {
5         System.out.println("a-m1");
6     }
7
8     public void m1(float a, int b) {
9         System.out.println("b-m1");
10    }
11
12 // int m1(float a, int b) {
13 //     System.out.println("b-m1");
14 //     return 100;
15 // }
16 }
17 }

Console ×
terminated (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.op
b-m1

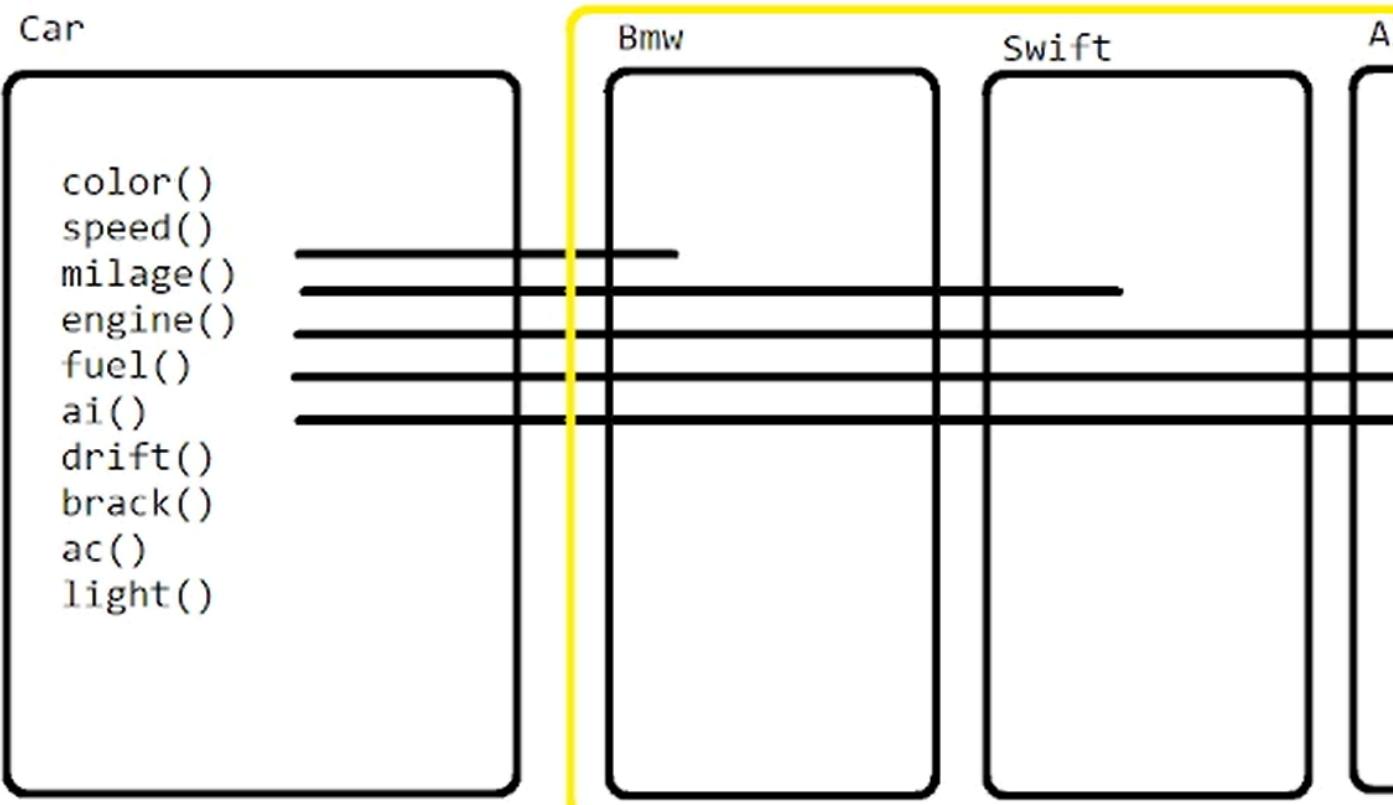
```

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Car car = new Car();
7         car.m1(10);
8
9     }
10}
11
```

```
Car.java
1 package com.mainapp;
2 public class Car {
3
4     public void m1(int a) {
5         System.out.println("LOGIC1");
6     }
7
8     public void m1(float a) {
9         System.out.println("LOGIC2");
10    }
11
12    public void m1(boolean a) {
13        System.out.println("LOGIC3");
14    }
15}
16
17
```

Console <terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.op LOGIC1

RUNTIME POLYMORPHISM (Late Binding/Dynamic Polymorphism/Dynamic Method Dispatch/Dynamic Binding))
(70% LINE CODE REDUCE)



Launch: create obj of child classes
->call all methods

DAY-32

Java Full Stack Development Batch 2025

Runtime Polymorphism:

1.Method Overriding

2.Upcasting (Holding child's object inside the Parent's reference)

There are two very important points here

1.Method Checking (COMPILE TIME) : without object (SEE REFERENCE)

2.Method Running (RUNTIME) : depends object

CALCULATION:

20 CAR CLASS : 10 METHODS

LAUNCH:-

- 20 LINES OBJECT CREATION
- $20 \times 10 = 200$
- Total=220

=====

- 21 line obj creation
- Method calling (test) : 20 Line
- Extra class: 10 line method calling + 5 extra
- TOTAL: 56LINES

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        Extra extra = new Extra();

        Bmw bmw=new Bmw();
        extra.test(bmw);

        Curve curve = new Curve();
        extra.test(curve);

        Swift swift = new Swift();
        extra.test(swift);

        Alto alto = new Alto();
        extra.test(alto);

        Thar thar = new Thar();
        extra.test(thar);

    }
}

package com.mainapp;

public class Extra {

    public void test(Car c) {

        c.color();
        c.speed();
        c.milage();
        c.engine();
        c.fuel();
        c.ai();
        c.drift();
        c.brake();
        c.ac();
        c.light();
    }
}

package com.mainapp;
public class Car {
```

```
public void color() {
    System.out.println("color");
}

public void speed() {
    System.out.println("speed");
}

public void milage() {
    System.out.println("milage");
}

public void engine() {
    System.out.println("engine");
}

public void fuel() {
    System.out.println("fuel");
}

public void ai() {
    System.out.println("ai");
}

public void drift() {
    System.out.println("drift");
}

public void brake() {
    System.out.println("brake");
}

public void ac() {
    System.out.println("ac");
}

public void light() {
    System.out.println("light");
}
}

package com.mainapp;
public class Bmw extends Car {

    @Override
    public void color() {
        System.out.println("BMW color");
    }
}
```

```
}

@Override
public void speed() {
    System.out.println("BMW speed");
}

@Override
public void milage() {
    System.out.println("BMW milage");
}

@Override
public void engine() {
    System.out.println("BMW engine");
}

@Override
public void fuel() {
    System.out.println("BMW fuel");
}

@Override
public void ai() {
    System.out.println("BMW ai");
}

@Override
public void drift() {
    System.out.println("BMW drift");
}

@Override
public void brake() {
    System.out.println("BMW brake");
}

@Override
public void ac() {
    System.out.println("BMW ac");
}

@Override
public void light() {
    System.out.println("BMW light");
}
```

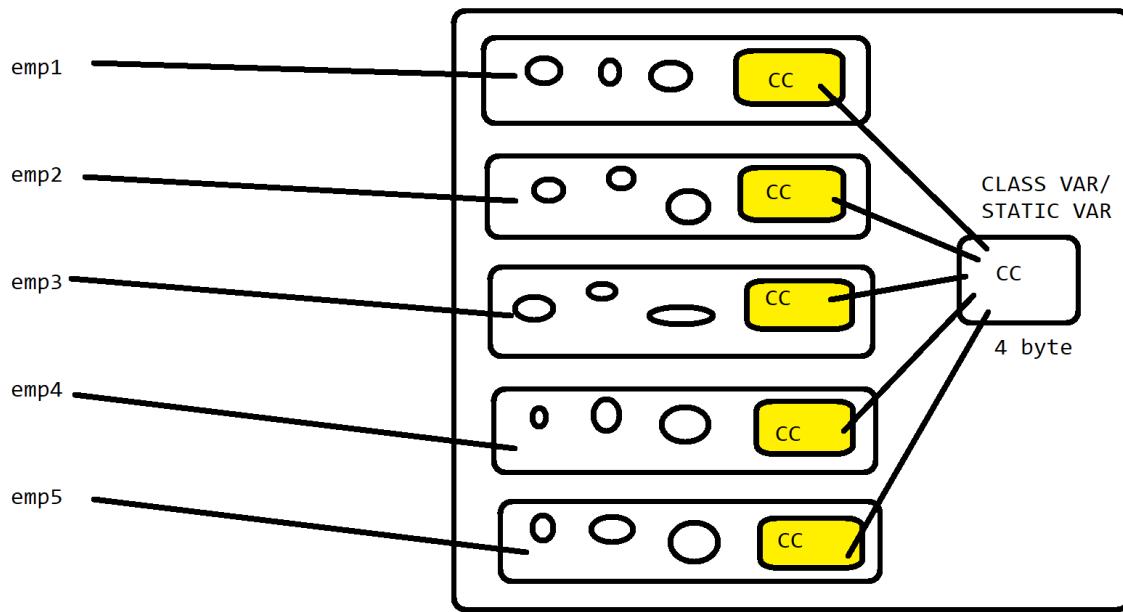
}

DAY-33

Java Full Stack Development Batch 2025

static keyword in Java

- static keyword in java is used to make your program memory efficient
- we can apply static keyword with
 - a.variable (outside the method)
 - b.method
 - c.block
 - d.nested class (LATER)
- static is nowhere related to the object



Program 1: static variable

```

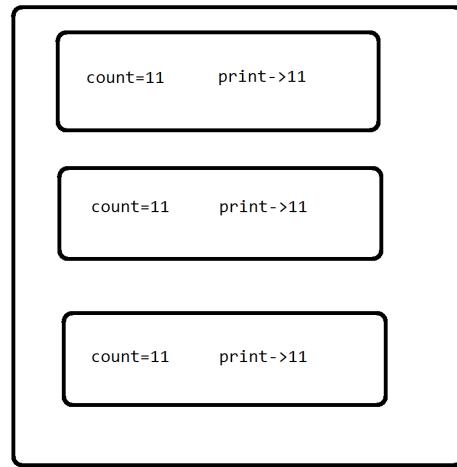
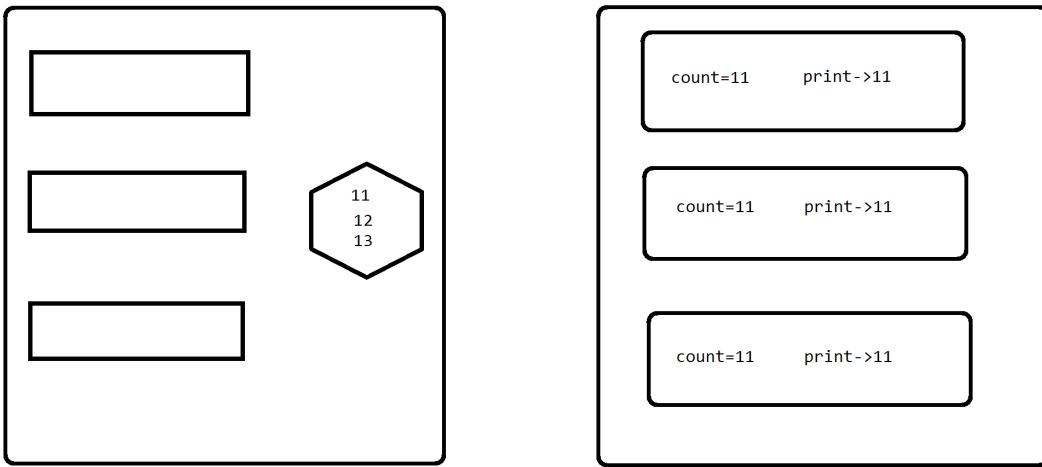
Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Employee emp1 = new Employee(6, 18, 'm');
7         Employee emp2 = new Employee(7, 19, 'm');
8         Employee emp3 = new Employee(8, 20, 'f');
9         Employee emp4 = new Employee(9, 21, 'f');
10        Employee emp5 = new Employee(10, 22, 'f');
11
12        emp1.printAllData();
13        emp2.printAllData();
14        emp3.printAllData();
15        emp4.printAllData();
16        emp5.printAllData();
17    }
18 }
19

Employee.java ×
1 package com.mainapp;
2 //INDIAN EMPLOYEES
3 public class Employee {
4
5     private int id;
6     private int age;
7     private char gender;
8
9     private static int countryCode=909090; //CLASS VARIABLE
10
11     public Employee(int id, int age, char gender) {
12         super();
13         this.id = id;
14         this.age = age;
15         this.gender = gender;
16     }
17
18     public void printAllData() {
19         System.out.println("ID: "+id);
20         System.out.println("AGE: "+age);
21         System.out.println("GENDER: "+gender);
22         System.out.println("COUNTRYCODE: "+countryCode);
23     }
24
25
26 }

Console ×
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.jdt.openjdl
ID: 6
AGE: 18
GENDER: m
COUNTRYCODE: 909090

```

Program 2: prove common memory



```

Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         Employee emp1 = new Employee();
7         emp1.print();
8         Employee emp2 = new Employee();
9         emp2.print();
10        Employee emp3 = new Employee();
11        emp3.print();
12    }
13 } //11 11 11, 11 12 13, 10 11 12,
14

Employee.java
1 package com.mainapp;
2 //INDIAN EMPLOYEES
3 public class Employee {
4
5     private static int count=10;
6
7     public Employee() {
8         count++;
9     }
10
11     public void print() {
12         System.out.println("COUNT="+count);
13     }
14 }

```

Console

```

<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjls
COUNT=11
COUNT=12
COUNT=13

```

Program 3: static block

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4         Employee emp1 = new Employee(6,18);
5         emp1.print();
6         Employee emp2 = new Employee(7,19);
7         emp2.print();
8         Employee emp3 = new Employee(8,20);
9         emp3.print();
10    }
11 }
12 
```

```
Employee.java
1 package com.mainapp;
2 import java.util.Scanner;
3 //INDIAN EMPLOYEES
4 public class Employee {
5     private int id;
6     private int age;
7     private static int countyCode;
8     static {
9         System.out.println("ENTER COUNTRY CODE");
10        Scanner scanner = new Scanner(System.in);
11        countyCode=scanner.nextInt();
12    }
13    public Employee(int id, int age) {
14        super();
15        System.out.println("CONSTRUCTOR");
16        this.id = id;
17        this.age = age;
18    }
19    public void print() {
20        System.out.println("ID="+id);
21        System.out.println("AGE="+age);
22        System.out.println("CC="+countyCode);
23    }
24 }
```

```
Console
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.h
ENTER COUNTRY CODE
9090
CONSTRUCTOR
ID=6
AGE=18
CC=9090
CONSTRUCTOR
ID=7
AGE=19
CC=9090
CONSTRUCTOR
ID=8
AGE=20
CC=9090
```

DAY-34

Java Full Stack Development Batch 2025

STATIC IS NOWHERE RELATED TO THE OBJECT

STATIC METHOD

Objects can access common data(static data)

Inside non static method we can access static variable

1.to call a static method we need not to create Object because static is nowhere related to the object

2.we should create static method only if we want to perform any logic which is common for all objects

3.Objects can access common data (non static method can access static variables)

The screenshot shows the Eclipse IDE interface with two open files and a console window.

Launch.java (Left Editor):

```
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4         Employee.commonLogic(100, 200);
5
6         Employee emp1 = new Employee();
7
8         emp1.objectSpecificLogic(10, 20);
9         emp1.printobjectSpecificLogic();
10
11        Employee emp2 = new Employee();
12        emp2.printobjectSpecificLogic();
13
14        emp1.printCommonLogic();
15        emp2.printCommonLogic();
16    }
17}
18}
```

Employee.java (Right Editor):

```
1 package com.mainapp;
2 public class Employee {
3
4     private int result;
5     private static int commonResult;
6
7     public void objectSpecificLogic(int a, int b) {
8         result = a * b;
9     }
10
11     public void printobjectSpecificLogic() {
12         System.out.println(result);
13     }
14
15     public void printCommonLogic() {
16         System.out.println(commonResult);
17     }
18
19     public static void commonLogic(int a, int b) {
20         commonResult = a * b;
21     }
22 }
```

Console (Bottom Window):

```
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.h
200
0
20000
20000
```

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Employee employee = new Employee();
6         System.out.println(employee.result);
7
8         //System.out.println(employee.commonResult);
9         System.out.println(Employee.commonResult);
10
11         employee.test();
12
13         //employee.demo();
14         Employee.demo();
15
16     }
17 }
Employee.java
1 package com.mainapp;
2 public class Employee {
3
4     public int result=10;
5     public static int commonResult=1000;
6
7     public void test() {
8
9     }
10
11     public static void demo() {
12
13     }
14 }
```

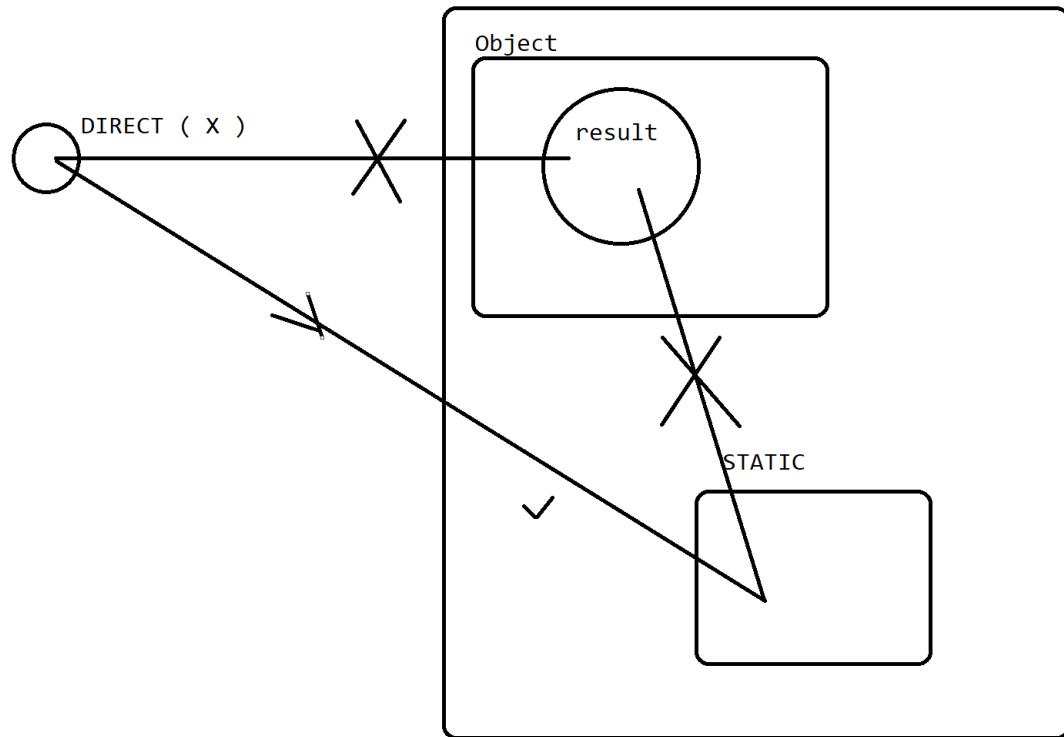
Console

```
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.
```

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         // Employee employee = new Employee();
6         // employee.test();
7
8         Employee.demo();
9
10    }
11 }
Employee.java
1 package com.mainapp;
2 public class Employee {
3
4     private int result=10;
5     private static int commonResult=1000;
6
7     public void test() { //OS--->COMMONDA
8         System.out.println(result);
9         System.out.println(commonResult);
10        demo();
11        //You can access static from non static
12    }
13
14    public static void demo() { //COMMON-
15        System.out.println(commonResult);
16        //System.out.println(result);
17        //you cannot even this keyword
18        //you cannot call not static met
19    }
20
21 }
```

Console

```
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.
1000
```



DAY-35

Java Full Stack Development Batch 2025

Interface

- >It is just a block of code
- >interface is also a keyword which is used to create an Interface

->**interface is just a blueprint of a class**(we can also say that Interface provides STANDARDIZATION)

->Interface contains **abstract method** which is used to provide method signature

->abstract method : method without body (NO { })

->to create an abstract method we **use abstract keyword** but in interface it is optional

->inside an interface all methods are public only so it is optional to use public keyword

-

Launch

prashant:

Problem
1.Lot of diff diff
methods

speed
milage
engine
color

Bmw

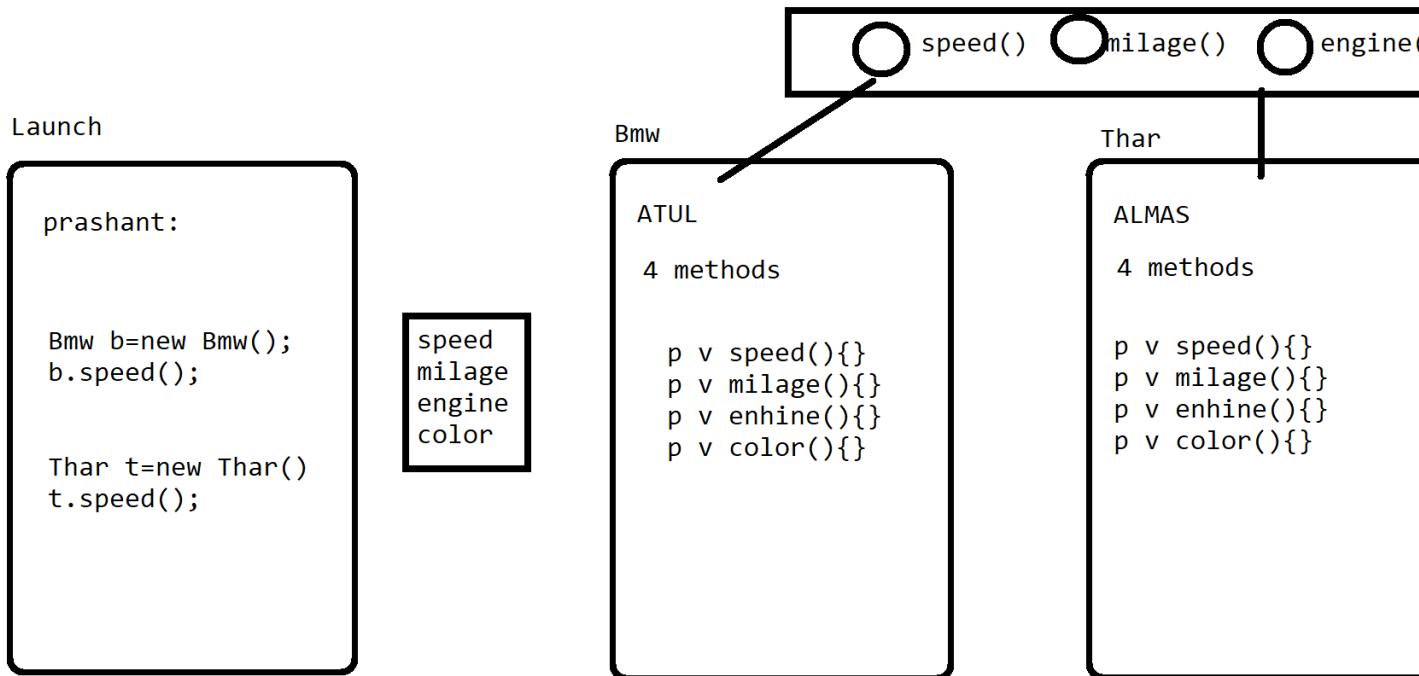
ATUL

```
4 methods
p v bmwSpeed(){}
}
p v bmwMilage(){}
}
p v engine(){}
p v color(){}
```

Thar

ALMAS

```
4 methods
p int speed(){}
}
p int milage(){}
}
p int engine(){}
p int color(){}
```



```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        test(new Bmw());
        test(new Thar());
        test(new Curve());
    }

    public static void test(MyInterface i) {

        i.speed();
        i.milage();
        i.engine();
        i.color();
    }
}
  
```

```

package com.mainapp;
public interface MyInterface {
  
```

```

        void speed();
        void milage();
        void engine();
        void color();

    }

package com.mainapp;
public class Bmw implements MyInterface {

    @Override
    public void speed() {
        System.out.println("bmw speed");
    }

    @Override
    public void milage() {
        System.out.println("bmw milage");
    }

    @Override
    public void engine() {
        System.out.println("bmw engine");
    }

    @Override
    public void color() {
        System.out.println("bmw color");
    }

}

```

DAY-36

Java Full Stack Development Batch 2025

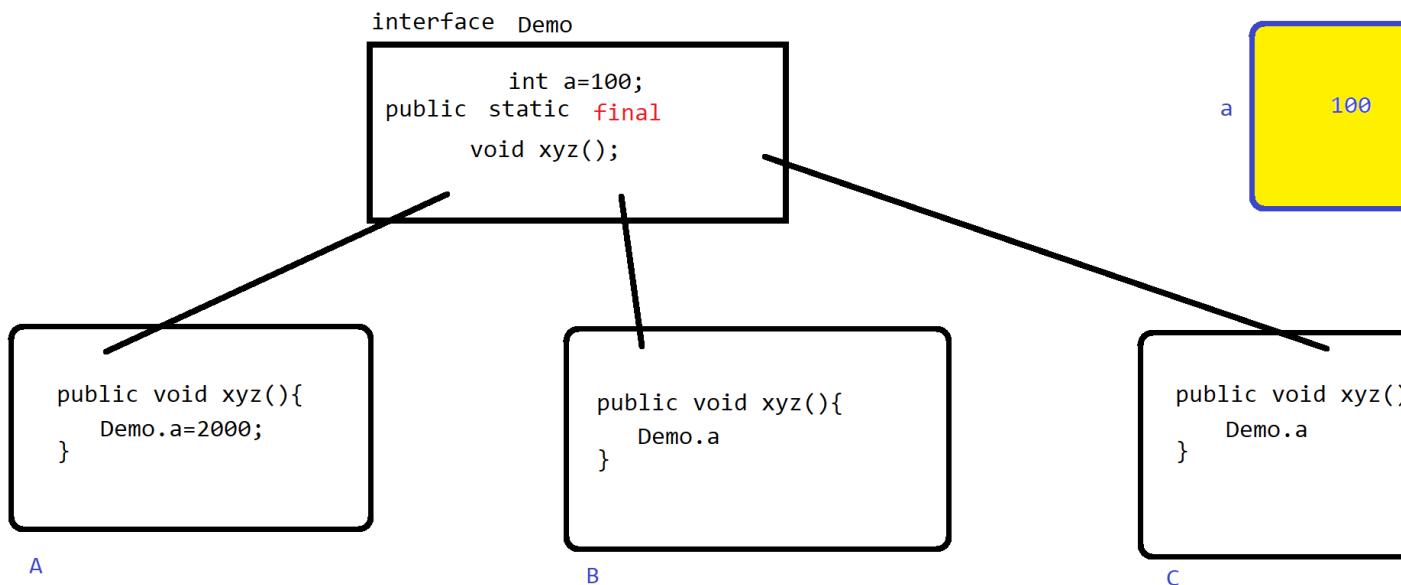
- Interface contains abstract method(only signature)
- We cannot create object of an Interface

- Inside an interface we can create variable also but that must be public static final

```
package com.mainapp;
public interface MyInterface {

    public static final int a=100;

}
```



Through Interface we can also achieve multiple inheritance

The screenshot shows a Java code editor with four tabs open:

- MyInterface.java**: A public interface with a single method `int speed();`.
- Test.java**: A public interface with a single method `void speed(int a);`.
- Child.java**: A public class that implements both `MyInterface` and `Test`. It overrides the `speed()` method from `MyInterface` and provides its own implementation for the `speed(int a)` method.
- Xyz.java**: A public class that extends `Object` and implements `Test`. It contains a method `pqrs()` that prints "LOGIC" to the console.

```

MyInterface.java:
1 package com.mainapp;
2 public interface MyInterface {
3     int speed();
4 }
5
6 }

Test.java:
1 package com.mainapp;
2 public interface Test {
3
4     void speed(int a);
5
6 }
7

Child.java:
1 package com.mainapp;
2 public class Child implements MyInterface,Test {
3
4     @Override
5     public int speed() {
6         return 0;
7     }
8
9     @Override
10    public void speed(int a) {
11
12    }
13
14 }

```

We can extends as well as implement at a time

The screenshot shows a Java code editor with four tabs open:

- MyInterface.java**: A public interface with a single method `int speed();`.
- Test.java**: A public interface with a single method `void speed(int a);`.
- Xyz.java**: A public class that extends `Object` and implements `Test`. It contains a method `pqrs()` that prints "LOGIC" to the console.
- Child.java**: A public class that extends `Xyz` and implements `MyInterface` and `Test`. It overrides the `speed(int a)` method from `Test` and provides its own implementation for the `speed()` method.

```

MyInterface.java:
1 package com.mainapp;
2 public interface MyInterface {
3
4     int speed();
5
6 }
7

Test.java:
1 package com.mainapp;
2 public interface Test {
3
4     void speed(int a);
5
6 }
7

Xyz.java:
1 package com.mainapp;
2 public class Xyz extends Object {
3
4     public void pqrs() {
5         System.out.println("LOGIC");
6     }
7
8 }

Child.java:
1 package com.mainapp;
2 public class Child extends Xyz implements MyInterface,Test {
3
4     @Override
5     public void speed(int a) {
6
7     }
8
9     @Override
10    public int speed() {
11         return 0;
12     }
13
14 }

```

When we extends a class with implementing an interface there will be problem of Runtime Polymorphism and it can be solve through abstract class(later)

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Xyz x;
        x=new Child();
        x.pqrs();

        MyInterface m=(MyInterface) x;
        m.insert();
        m.read();
    }
}

package com.mainapp;
public class Xyz extends Object {

    public void pqrs() {
        System.out.println("LOGIC");
    }

    //insert() read()
}
package com.mainapp;
public interface MyInterface {

    void insert();
    void read();

}

package com.mainapp;
public class Child extends Xyz implements MyInterface  {

    @Override
    public void pqrs() {
        System.out.println("LOGIC U1");
    }
}
```

```

@Override
public void insert() {
    System.out.println("IN C1");
}

@Override
public void read() {
    System.out.println("RD C1");
}

}

package com.mainapp;
public class Child2 extends Xyz implements MyInterface {
    @Override
    public void pqrs() {
        System.out.println("LOGIC U2");
    }

    @Override
    public void insert() {
        System.out.println("IN C2");
    }

    @Override
    public void read() {
        System.out.println("RD C2");
    }
}

```

DAY-37

Java Full Stack Development Batch 2025

1. STANDARDIZATION (BLUEPRINT)

2. MULTIPLE INHERITANCE

3. LOOSE COUPLING

4.THROUGH AN INTERFACE WE CAN PERFORM SOME EXTRA INTERNAL OPERATION(BY JVM) ON A CLASS.(MARKER INTERFACE)

COUPLING:

- Tight coupling
- Loose coupling

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Test t = new Child();
6         t.insert(); //NO ERROR
7     }
8 }
9
10

Child.java
1 package com.mainapp;
2 public class Child {
3
4     @Override
5     public void insert() {
6         System.out.println("INSERT");
7     }
8
9
10

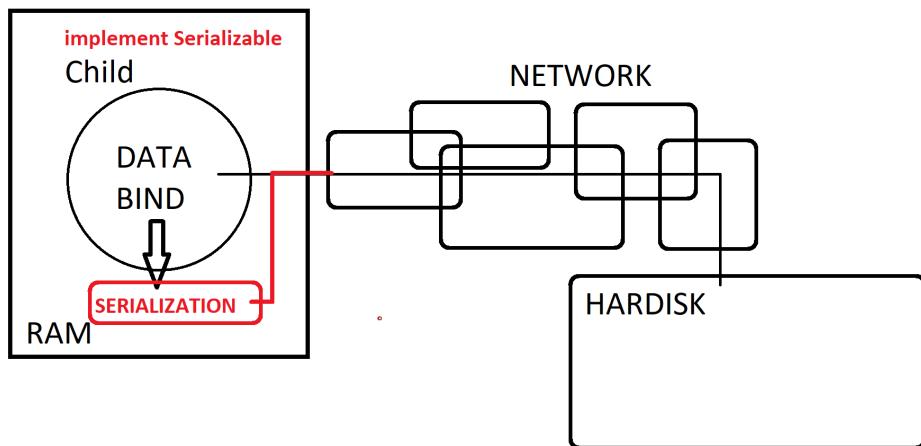
Test.java
1 package com.mainapp;
2 public class Test {
3
4     @Override
5     public void insert() {
6         System.out.println("NO ERROR");
7     }
8
9
10
11
12
13
14
15
16
17
18 }
```

Console

```
<terminated> Launch (3) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jdk11\bin\java -jar C:\Users\DELL\OneDrive\Desktop\MarkerInterface.jar
INSERT
```

MARKER INTERFACE

Empty interface



PROBLEM:

The screenshot shows a Java development environment with four code files:

- Launch.java**:

```
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         MyInterface m=new Child();
6         m.insert();
7         m.read();
8         m.pqrs();
9         m.ijkl();
10    }
11
12 }
```
- MyInterface.java**:

```
1 package com.mainapp;
2 public interface MyInterface {
3
4     void insert();
5     void read();
6     public void pqrs();
7     public void ijk1();
8
9 }
10
```
- Child.java**:

```
1 package com.mainapp;
2 public class Child extends Xyz implements MyInterface {
3
4     @Override
5     public void pqrs() {
6         System.out.println("LOGIC");
7     }
8
9     @Override
10    public void insert() {
11        System.out.println("IN C1");
12    }
13
14    @Override
15    public void read() {
16        System.out.println("RD C1");
17    }
18 }
```
- Child2.java**:

```
1 package com.mainapp;
2 public class Child2 extends Xyz implements MyInterface {
3
4     @Override
5     public void pqrs() {
6         System.out.println("LOGIC");
7     }
8
9     @Override
10    public void insert() {
11        System.out.println("IN C2");
12    }
13
14    @Override
15    public void read() {
16        System.out.println("RD C2");
17    }
18 }
```

ABSTRACT CLASS:

- class->concrete method
- interface->abstract method
- abstract-> concrete method & abstract method

- use abstract keyword(mandatory) to create abstract class

- inside an abstract class you can use other access modifiers for abstract method except private
- you cannot instantiate abstract class directly but if you create object of its child then automatically its constructor will call

```

Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String
4
5         MyAbstract m=new Child();
6         m.insert();
7         m.read();
8         m.pqrs();
9         m.ijkl();
10    }
11 }
12

Child.java
1 package com.mainapp;
2 public class Child extends MyAbstract {
3
4     public Child() {
5         System.out.println("C
6     }
7
8     @Override
9     public void pqrs() {
10        System.out.println("L
11    }
12
13     @Override
14     public void insert() {
15        System.out.println("I
16    }
17
18     @Override
19     public void read() {
20        System.out.println("R
21    }
22 }
23

```

Console

```

<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\eclipse\plugins\com.eclipsesource.jdt.ls\2.1.0\jdt.ls\java\src\com\mainapp\Launch.java
PCC
CCC
IN C1
RD C1
LOGIC U1
LOGICY

```

ABSTRACTION:

ACCESS MODIFIER

JAVA MEMORY MANAGEMENT

DAY-38

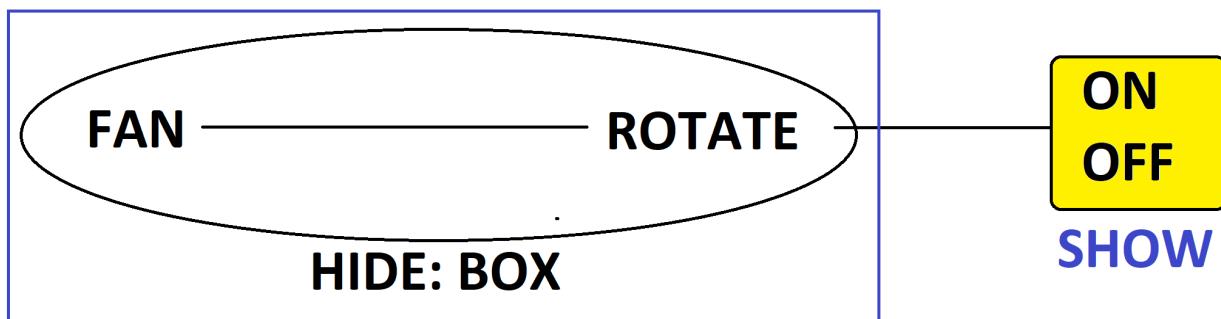
Java Full Stack Development Batch 2025

ABSTRACTION

Encapsulation: DATA BINDING/DATA HINDING

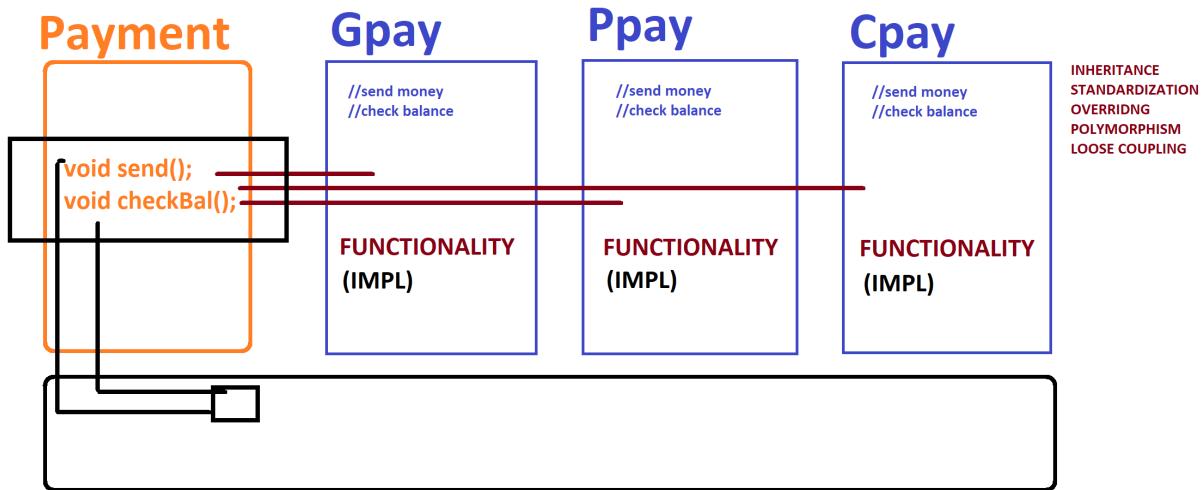
->SHOWING ONLY NECESSARY DETAILS AND **HIDING THE
IMPLEMENTATION**

->WE USE THE FUNCTIONALITY BUT WE DON'T CARE HOW IT HAS
BEEN IMPLEMENTED



->WE CAN ACHIEVE ABSTRACTION IN TWO WAYS

- 1. THROUGH INTERFACE (100% ABSTRACTION)
- 2. THROUGH ABSTRACT CLASS (0 to 100%)



Program: Through Interface

```

Launch.java ×
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Payment p = new Gpay();
6         p.send();
7         p.check();
8         p.miniStmt();
9         p.last5Trs();
10
11     }
12 }
13

Payment.java ×
1 package com.mainapp;
2 public interface Payment {
3     void send();
4     void check();
5     void miniStmt();
6     void last5Trs();
7 }

Gpay.java ×
1 package com.mainapp;
2 public class Gpay implements Payment {
3     @Override
4     public void send() {
5         System.out.println("GPAY SEND");
6     }
7     @Override
8     public void check() {
9         System.out.println("GPAY CHECK");
10    }
11
12    @Override
13    public void miniStmt() {
14        System.out.println("GPAY MINI");
15    }
16
17    @Override
18    public void last5Trs() {
19        System.out.println("GPAY LAST5");
20    }
21 }

```

Console ×
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hot
GPAY SEND
GPAY CHECK
GPAY MINI
GPAY LAST5

FACTORY DESIGN PATTERN (WE TAKE OBJECT FROM ANOTHER CLASS)

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Payment p = ObjectFactory.getObject(1111);
        if (p == null) {
            System.out.println("WRONG KEY");
        } else {
            p.send();
            p.check();
            p miniStmt();
            p.last5Trs();
        }
    }
}

package com.mainapp;

public class ObjectFactory {

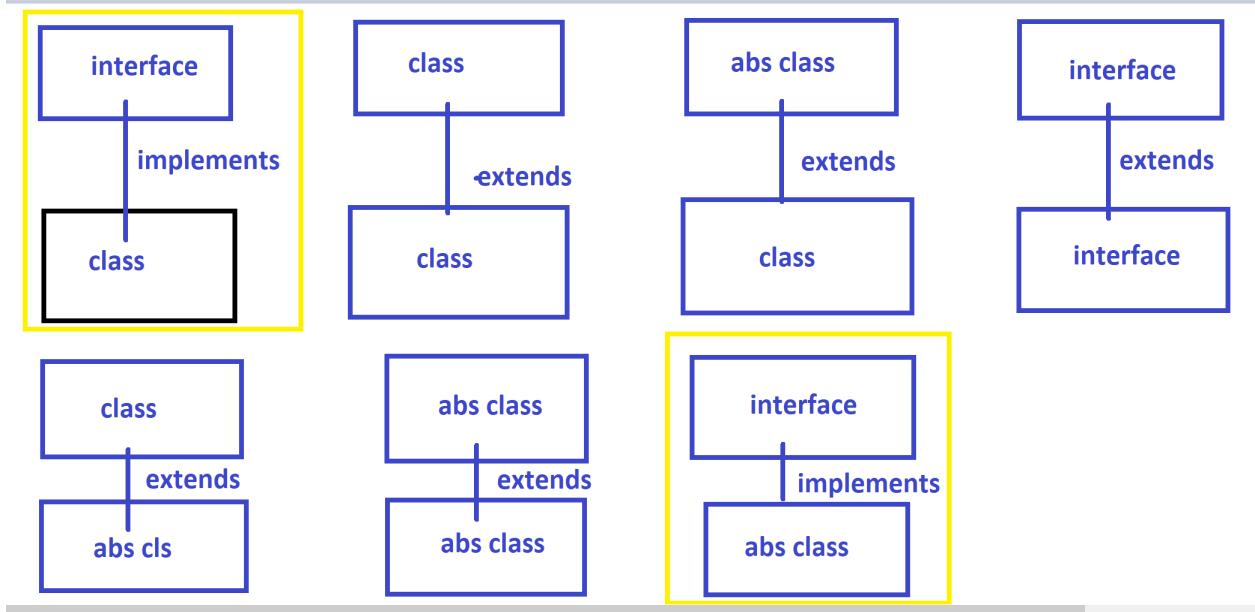
    public static Payment getObject(int key) {

        if(key==1111) {
            return new Gpay();
        }
        else if(key==2222) {
            return new Ppay();
        }
        else if(key==3333) {
            return new Cpay();
        }
        else {
            return null;
        }
    }
}

package com.mainapp;
public interface Payment {
    void send();
    void check();
```

```
    void miniStmt();
    void last5Trs();
}
package com.mainapp;
public class Gpay implements Payment {
    @Override
    public void send() {
        System.out.println("GPAY SEND");
    }
    @Override
    public void check() {
        System.out.println("GPAY CHECK");
    }
    @Override
    public void miniStmt() {
        System.out.println("GPAY MINI");
    }
    @Override
    public void last5Trs() {
        System.out.println("GPAY LAST5");
    }
}
```

NOTE:



PROGRAM: ABSTRACTION USING ABSTRACT CLASS

```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Payment p = ObjectFactory.getObject(1111);
        if (p == null) {
            System.out.println("WRONG KEY");
        } else {
            p.send();
            p.check();
            p miniStmt();
            p.last5Trs();
            p.policy();
        }
    }
}

```

```
package com.mainapp;
public abstract class Payment {

    public abstract void send();
    public abstract void check();
    public abstract void miniStmt();
    public abstract void last5Trs();

    public void policy() {
        System.out.println("POLICY");
    }
}
package com.mainapp;

public class ObjectFactory {

    public static Payment getObject(int key) {

        if(key==1111) {
            return new Gpay();
        }
        else if(key==2222) {
            return new Ppay();
        }
        else if(key==3333) {
            return new Cpay();
        }
        else {
            return null;
        }
    }
}
package com.mainapp;
public class Gpay extends Payment {
    @Override
    public void send() {
        System.out.println("GPAY SEND");
    }
    @Override
```

```
public void check() {  
    System.out.println("GPAY CHECK");  
}  
  
@Override  
public void miniStmt() {  
    System.out.println("GPAY MINI");  
}  
  
@Override  
public void last5Trs() {  
    System.out.println("GPAY LAST5");  
}  
}
```

DAY-39

Java Full Stack Development Batch 2025

1. Access Modifier

2. Java Memory Management

Access Modifier

1.private (within the class)

2.public (inside outside the package)

3.default(within the package)

4.protected (within the package and outside side the package(CHILD CLASS))

The screenshot shows an IDE interface with three code editors:

- Launch.java:** A public class Launch with a main method. It imports com.test.Test and creates a new Test object to call its testMethod.
- Gpay.java:** A protected void send() method within the Gpay class, which prints "GPAY".
- Test.java:** A public class Test that extends Gpay. It overrides the send() method and calls it from its own testMethod.

```
Launch.java:
1 package com.mainapp;
2 import com.test.Test;
3
4 public class Launch {
5     public static void main(String[] args) {
6
7         Test test = new Test();
8         test.testMethod();
9     }
10
11 }
12
13 }
14

Gpay.java:
1 package com.mainapp;
2 public class Gpay {
3
4     protected void send() {
5         System.out.println("GPAY");
6     }
7
8 }
9
10

Test.java:
1 package com.test;
2 import com.mainapp.Gpay;
3
4 public class Test extends Gpay {
5
6     public void testMethod() {
7         send();
8     }
9     //Gpay send()
10 }
11
```

JAVA MEMORY MANAGEMENT

->IT IS A PROCESS OF ALLOCATION AND DEALLOCATION OF OBJECTS INSIDE THE MEMORY, IT ALSO INCLUDES STATIC ALLOCATION

1.CLASS LOADING

- >CLASSES ARE LOADED INTO THE JVM THROUGH CLASS LOADER (A PART OF JRE)
- >BYTE VERIFICATION (.class)
- >METHOD AREA: STORE META DATA WITH STATIC DATA

2.OBJECT CREATION

- >JVM ALLOCATES MEMORY FOR THE OBJECTS INSIDE THE HEAP MEMORY
- >AFTER MEMORY ALLOCATION JVM PUTS DEFAULT VALUE (LIKE FOR INT->0)
- >AFTER SETTING DEFAULT VALUES IT WILL CALL USER DEFINED CONSTRUCTOR (IF AVAILABLE)

3.OBJECT REFERENCE

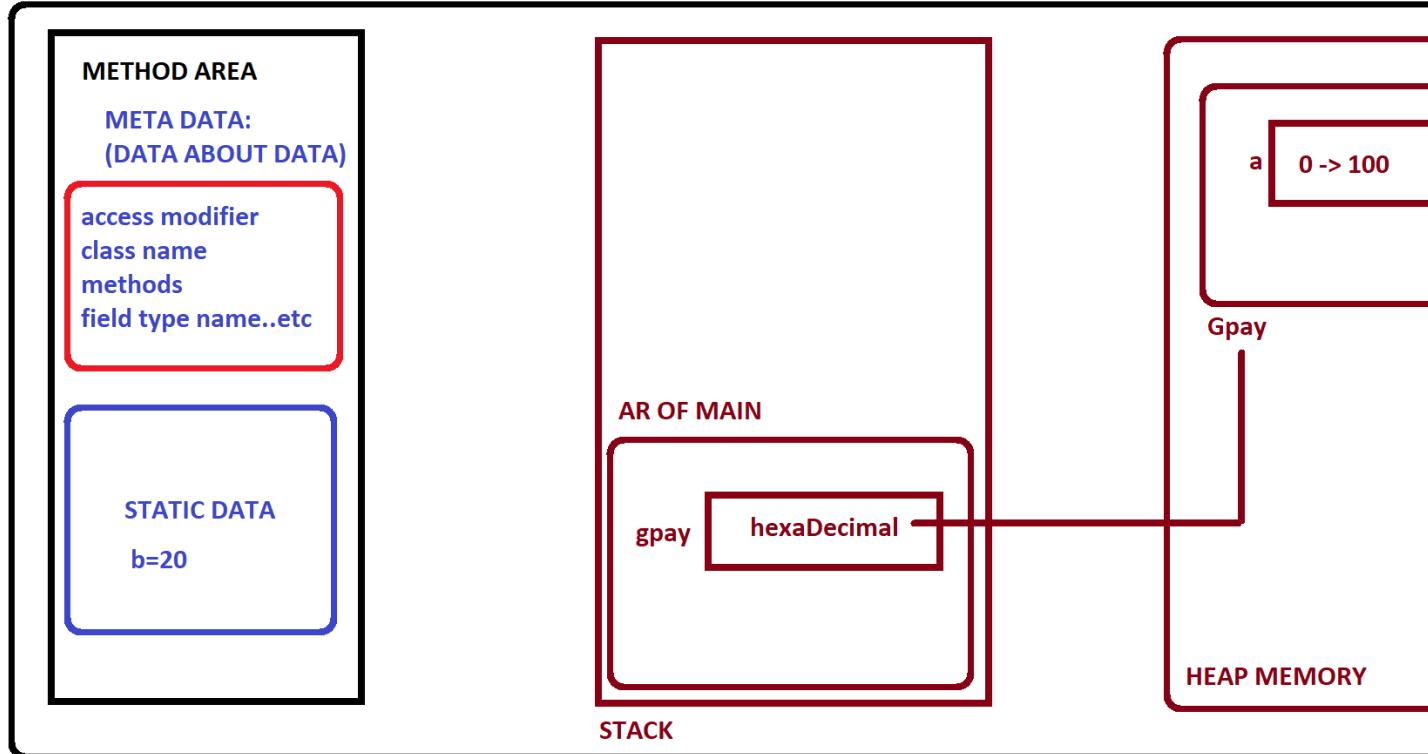
- >A REFERENCE VAR IS ASSIGNED TO POINT THE OBJECT (INTERNALY USES POINTERS)
- >IF YOUR REFERENCE VARIABLE IS A LOCAL VARIABLE THEN IT WILL GET STORED WITH ACTIVATION RECOND INSIDE THE STCACK
- >ACTIVATION: local variable, method parameter, controls....etc

4.OBJECT ACCESS:

-> YOU CAN CALL METHODS (NOT STATIC) OR YOU CAN DIRECTLY ACCESS FIELDS

5.GARBAGE COLLECTION (IT IS A PROCESS OF CLEANUP)

->JVM KEEPS TRACK THE DEFERENECED OBJECT AND IF JVM FINDS ANY OBJECT WITHOUT REFERENCE IT WILL START CLEANUP PROCESS
->TERMINATE (COMPLETE CLEANUP)



DAY-40

Java Full Stack Development Batch 2025

Explain Destructor in Java?

->JAVA DOES NOT SUPPORT DESTRUCTOR LIKE C++

->INSTEAD JAVA HAS GARBAGE COLLECTOR THAT AUTOMATICALLY DEALLOCATES MEMORY FOR UNUSED OBJECTS

1. ASSOCIATION & COMPOSITION

ASSOCIATION:

- Association represents a relationship between two independent classes.
- There is no ownership: both objects can exist independently

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Car car = new Car(1111, 989800);
        Employee emp = new Employee(11, 18, car);

        System.out.println("ID->" + emp.getId());
        System.out.println("AGE->" + emp.getAge());
        System.out.println("CN->" + emp.getCar().getCarNo());

        System.out.println("CM->" + emp.getCar().getCarModelNo());
```

```
    }
}

package com.mainapp;
public class Employee {

    private int id;
    private int age;
    private Car car; //For Employee Car is a Dependency
    //ASSOCIATION

    public Employee(int id,int age,Car car) {
        this.id=id;
        this.age=age;
        this.car=car;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
        this.car = car;
    }
}
```

```

    }
}

package com.mainapp;
public class Car {

    private int carNo;
    private int carModelNo;

    public Car(int carNo, int carModelNo) {
        this.carNo = carNo;
        this.carModelNo = carModelNo;
    }

    public int getCarNo() {
        return carNo;
    }

    public void setCarNo(int carNo) {
        this.carNo = carNo;
    }

    public int getCarModelNo() {
        return carModelNo;
    }

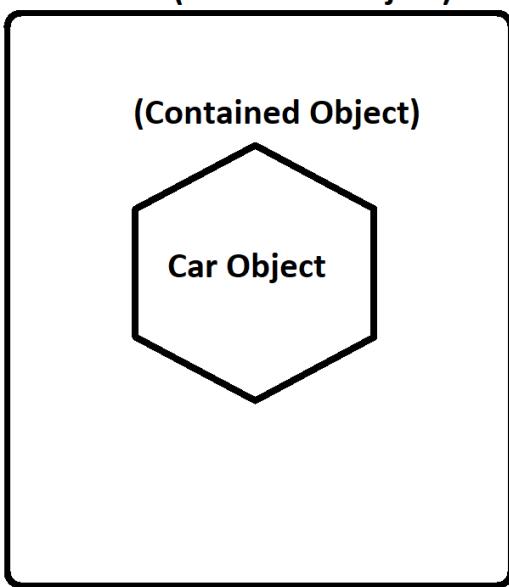
    public void setCarModelNo(int carModelNo) {
        this.carModelNo = carModelNo;
    }
}

```

COMPOSITION:

- Composition is strong form of association where one object owns another object.
- If Container object is destroyed then the contained object also get destroyed

Employee (Container Object)



```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Employee emp = new Employee(11, 18);

        System.out.println("ID->" + emp.getId());
        System.out.println("AGE->" + emp.getAge());
        System.out.println("CN->" + emp.getCar().getCarNo());
        System.out.println("CM->" + emp.getCar().getCarModelNo());

    }
}
package com.mainapp;
public class Employee {

    private int id;
    private int age;
    private Car car; //For Employee Car is a Dependency

    public Employee(int id,int age) {
        this.id=id;
        this.age=age;
        this.car=new Car(9090, 987654);
        //COMPOSITION
    }

}
```

```
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public Car getCar() {
    return car;
}

public void setCar(Car car) {
    this.car = car;
}
}

package com.mainapp;
public class Car {

    private int carNo;
    private int carModelNo;

    public Car(int carNo, int carModelNo) {
        this.carNo = carNo;
        this.carModelNo = carModelNo;
    }

    public int getCarNo() {
        return carNo;
    }

    public void setCarNo(int carNo) {
        this.carNo = carNo;
    }

    public int getCarModelNo() {
```

```
        return carModelNo;
    }

    public void setCarModelNo(int carModelNo) {
        this.carModelNo = carModelNo;
    }
}
```

DAY-41

Java Full Stack Development Batch 2025

ARRAY

DATA HOLD

- primitive variable Ex. int roll=10;
- Object Ex. [id,name,address,salary] Employee
emp1=new Employee(11,"raju","add",1000);

USE CASE

- 1. I want to store 1000 roll no. **SOL: Array**
- 2. I want to store 1000 employee object. **SOL: Array**

In java Array is an Object

Array is a collection of similar data/homogeneous data

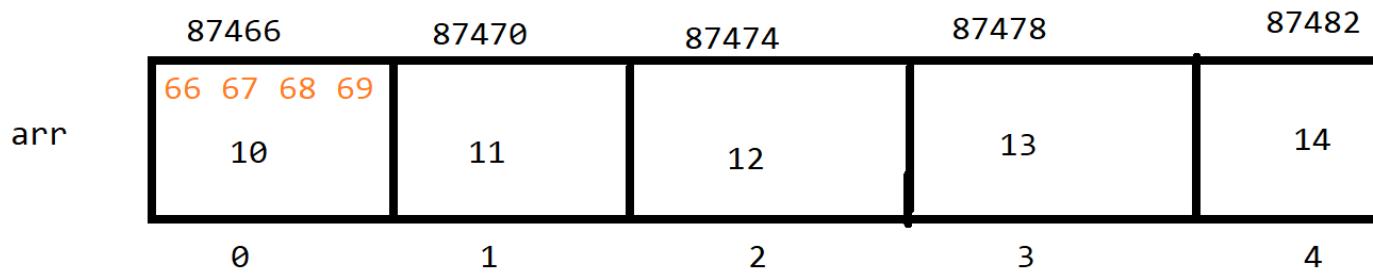
Through Array we can hold multiple values in a single variable

**We cannot change the size of array at runtime
(IMMUTABLE)**

It uses contiguous memory location for fast access

Array supports INDEX (from zero)

```
int arr[ ]= {10,11,12,13,14};
```



TYPES OF ARRAY

1.SINGLE DIMENTIONAL ARRAY (1D)

2. MULTI DIMENTIONAL ARRAY(2D,3D...etc)

SINGLE DIMENTIONAL ARRAY (1D)

1. PROGRAMMER : SIZE & ELEMENT(INPUT)

2. USER: SIZE & ELEMENT(INPUT)

1D ARRAY (PROGRAMMER INPUT)

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        // int arr[] = {};//empty array
        // int arr[] = {11,22,33,44,55};
        // int[] arr = {11,22,33,44,55};

        int arr[] = {11,22,33,44,55};
        //
        System.out.println(arr[0]);
        //
        System.out.println(arr[1]);
        //
        System.out.println(arr[2]);
        //
        System.out.println(arr[3]);
        //
        System.out.println(arr[4]);

        //ARRAY ITERATION
        for(int i=0;i<arr.length;i++) {
            System.out.println(arr[i]);
        }

        //FOR EACH LOOP : ENHANCE FOR LOOP : FROM WHERE , WHAT I'LL
        GET

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

        arr[2]=800;
        arr[5]=900;
```

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

**WAP TO STORE 5 EMPLOYEE OBJECT INSIDE AN
ARRAY (1D: PROGRAMMER INPUT)**

AND ITERATE THAT ARRAY USING FOR EACH LOOP

DAY-42

Java Full Stack Development Batch 2025

**1D ARRAY (PROGRAMMER
INPUT) : NOT PRIMITIVE**

```
Console Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Employee emp1 = new Employee(1, 18);
6         Employee emp2 = new Employee(2, 19);
7         Employee emp3 = new Employee(3, 20);
8         Employee emp4 = new Employee(4, 21);
9         Employee emp5 = new Employee(5, 22);
10
11         Employee emp[] = {emp1,emp2,emp3,emp4,emp5};
12
13         for(Employee employee :emp ) {
14             System.out.println("ID="+employee.getId());
15             System.out.println("AGE="+employee.getAge());
16             System.out.println("*****");
17         }
18     }
19 }
20

Employee.java
1 package com.mainapp;
2 public class Employee {
3
4     private int id;
5     private int age;
6
7     public Employee(int id, int age) {
8         super();
9         this.id = id;
10        this.age = age;
11    }
12
13    public int getId() {
14        return id;
15    }
16
17    public void setId(int id) {
18        this.id = id;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28 }
```

1D ARRAY (USER INPUT)

->create array using new keyword

The screenshot shows an IDE interface with two main panes. The left pane is titled "Launch.java" and contains the following Java code:

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4     public static void main(String[] args) {
5
6         Scanner scanner = new Scanner(System.in);
7
8         System.out.print("ENTER ARRAY SIZE: ");
9         int arraySize=scanner.nextInt();
10
11        int arr[]=new int[arraySize]; //DEF VALUE->0
12
13        for(int i=0;i<arr.length;i++) {
14            System.out.print("ENTER arr["+i+"]=");
15            arr[i]=scanner.nextInt();
16        }
17
18        //ITERATION
19        for(int k:arr)
20            System.out.print(k+" ");
21    }
22}
```

The right pane is titled "Console" and displays the output of the program. It shows the user entering the array size as 7, followed by seven integer values (11, 12, 13, 14, 15, 16, 17), and finally the printed output showing the array elements separated by spaces.

```
<terminated> Launch (2) [Java Application] D:\ECLI
ENTER ARRAY SIZE: 7
ENTER arr[0]=11
ENTER arr[1]=12
ENTER arr[2]=13
ENTER arr[3]=14
ENTER arr[4]=15
ENTER arr[5]=16
ENTER arr[6]=17
11 12 13 14 15 16 17
```

1D ARRAY (USER INPUT) : NON PRIMITIVE

->create array using new keyword

The screenshot shows a Java development environment with two panes. The left pane displays the code for a class named 'Launch'. The right pane shows the 'Console' output.

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4     public static void main(String[] args) {
5
6         Scanner scanner = new Scanner(System.in);
7
8         System.out.print("ENTER NO OF EMPLOYEE: ");
9         int count=scanner.nextInt(); //5
10
11        Employee emp[]=new Employee[count]; //5
12        for(int i=0;i<count;i++) { //0 to 4
13
14            System.out.print("ENTER ID OF EMP-"+i+":");
15            int id=scanner.nextInt();
16            System.out.print("ENTER AGE OF EMP-"+i+":");
17            int age=scanner.nextInt();
18
19            Employee employee=new Employee(id, age);
20            emp[i]=employee;
21        }
22        //ITERATION
23        for(Employee employee:emp) {
24            System.out.print("ID: "+employee.getId()+" ");
25            System.out.println("AGE: "+employee.getAge());
26            System.out.println("*****");
27        }
28    }
29 }
30 }
```

The console output shows the following interaction:

```
<terminated> Launch (2) [Java]
ENTER NO OF EMPLOYEE:
ENTER ID OF EMP-0:1
ENTER AGE OF EMP-0:18
ENTER ID OF EMP-1:2
ENTER AGE OF EMP-1:19
ENTER ID OF EMP-2:3
ENTER AGE OF EMP-2:20
ENTER ID OF EMP-3:4
ENTER AGE OF EMP-3:21
ENTER ID OF EMP-4:5
ENTER AGE OF EMP-4:22
ID: 1 AGE: 18
*****
ID: 2 AGE: 19
*****
ID: 3 AGE: 20
*****
ID: 4 AGE: 21
*****
ID: 5 AGE: 22
*****
```

TASK:

1.WAP TO PRINT AN ARRAY IN REVERSE ORDER

```
int arr[]={11,22,33,44,55}
```

**2.WAP TO PRINT ALTERNATE ELEMENTS OF AN
ARRAY IN REVERSE ORDER**

```
int arr[]={11,22,33,44,55}
```

3.WAP TO PRINT TO SUM UP ALL THE ELEMENTS OF THE GIVEN ARRAY

```
int arr[]={11,22,33,44,55}
```

4.WAP TO PRINT TO SUM UP ALL THE EVEN ELEMENTS OF THE GIVEN ARRAY

```
int arr[]={11,22,33,44,55}
```

5.WAP TO PRINT TO REVERSE THE GIVEN ARRAY

```
int arr[]={11,22,33,44,55}
```

CONCEPT BASED:

WAP TO INSERT EMPLOYEE DATA (id name address salary) IN THE ARRAY

- EMPLOYEE DATA: USER INPUT
- NO OF EMPLOYEE: USER INPUT
- IF NO OF EMPLOYEE IS MORE THAN 10: PRINT: LIMIT EXCEEDED

DAY-43

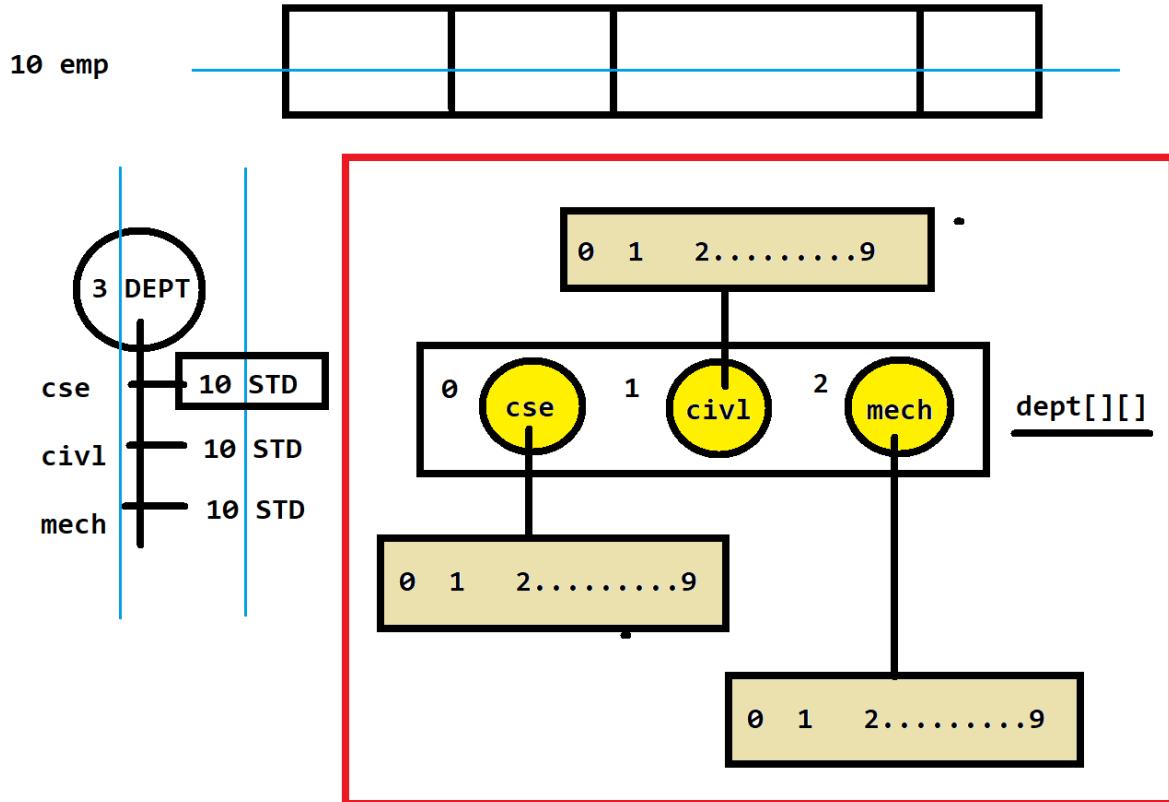
Java Full Stack Development

Batch 2025

Multidimensional Array (2D,3D....ND)

2 D ARRAY

- a. Regular Array**
 - b. Irregular Array(JAGGED ARRAY)**
- >SET OF 1D ARRAY

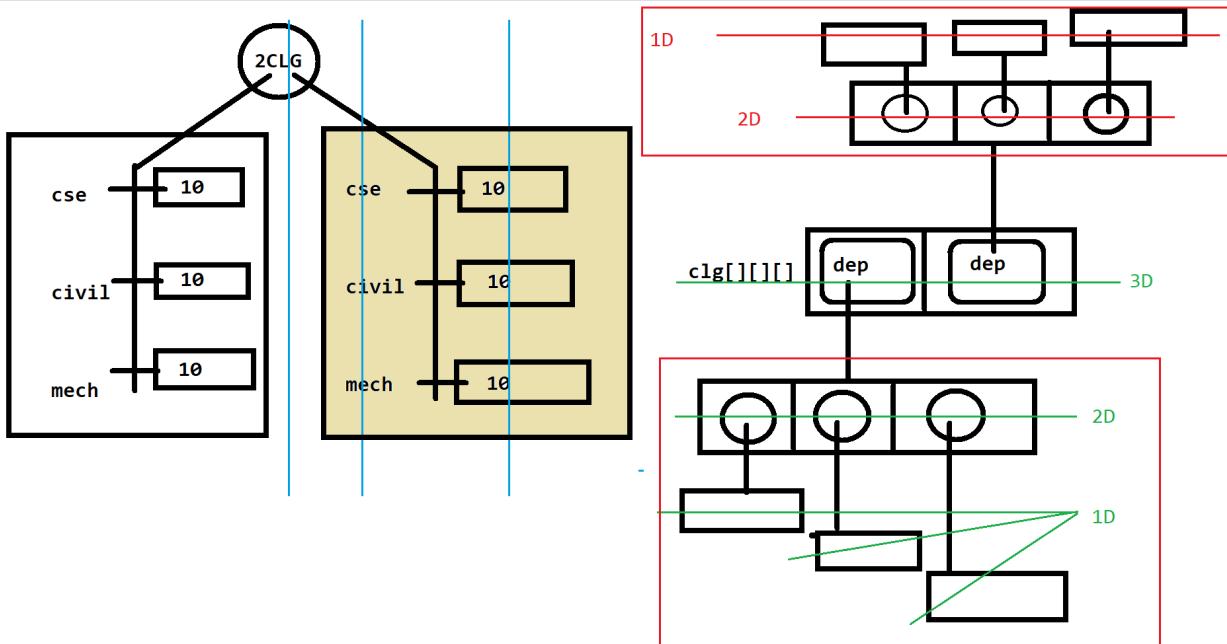


3 D ARRAY

a. Regular Array

b. Irregular Array(JAGGED ARRAY)

->SET OF 2D ARRAY



EXAMPLE:

- Roll no of raju -> Variable
- Employee data(id,name,address,salary) -> Employee Object (ref variable)
- Roll no of 10 student -> 1D ARRAY
- Roll no of 1000 student -> 1D ARAAY
- 10 EMPLOYEE OBJECT -> 1D ARRAY
- 1000 EMPLOYEE OBJECT -> 1D ARRAY
- 2 DEP -> 10,10 EMPLOYEES -> 2D ARRAY
- 2 CLG -> 3,3 DEPT -> 10 STD IN EACH DEP -> 3D ARRAY

2D ARRAY | REGULAR | Programmer Input

2D ARRAY | JAGGED | Programmer Input

2D ARRAY | REGULAR | USER Input

2D ARRAY | JAGGED | USER Input

3D ARRAY | REGULAR | Programmer Input

3D ARRAY | JAGGED | Programmer Input

3D ARRAY | REGULAR | USER Input

3D ARRAY | JAGGED | USER Input

ARRAY PASSING

ANONYMOUS ARRAY

DAY-44

Java Full Stack Development

Batch 2025

2D ARRAY | REGULAR |

Programmer Input

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //3 DEP -> 4 STUDENTS IN EACH DEP

        int arr[][]= {

            {11,22,55,86}, //dep0
            {66,77,21,55}, //dep1
            {70,50,10,77} //dep2
        };

        for(int i=0 ;i<arr.length ; i++) { //0 1 2
            for(int j=0 ; j < arr[0].length ; j++ ) { //0
1 2 3
                System.out.print(arr[i][j]+ " ");
            }
            System.out.println();
        }

        // System.out.println(arr.length);
        // System.out.println(arr[1][2]);
        // System.out.println(arr[2][0]);
    }
}
```

```

//      System.out.println(arr.length);
//      System.out.println(arr[2].length);
/*
 *   0,0   0,1   0,2   0,3
 *   1,0   1,1   1,2   1,3
 *   2,0   2,1   2,2   2,3
 */
}
}

```

2D ARRAY | JAGGED | Programmer Input

```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //3 DEP -> 4 STUDENTS IN EACH DEP

        int arr[][]= {

            {11,22,55,86,90,65,89},  //dep0
            {66,77},    //dep1
            {70,50,10,77}   //dep2
        };

        for(int i=0 ;i<arr.length ; i++) { //0 1 2
    
```

```

1 2 3 4 5 6
    for(int j=0 ; j < arr[i].length ; j++ ) { //0
        System.out.print(arr[i][j]+" ");
    }
    System.out.println();
}
}
}

```

PROGRAM2:

```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //3 DEP -> 4 STUDENTS IN EACH DEP

        int arr[][]= {

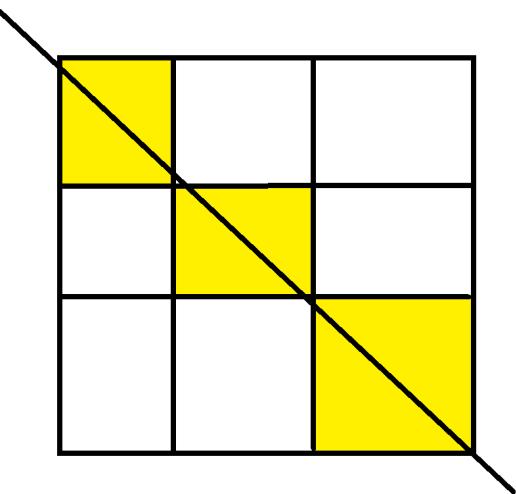
            {11,22,55,86,90,65,89},   //dep0
            {66,77},      //dep1
            {70,50,10,77}   //dep2
        };

        arr[2][2]=1000;

        for( int k[] : arr ) {
            for( int elm : k ) {
                System.out.print(elm+" ");
            }
            System.out.println();
        }
    }
}

```

TASK:



```
{  
    {11,22,33},  
    {11,22,33},  
    {11,22,33}  
}
```

output: 66

TASK:

**WAP to update(1000) the value of 2D array
whose index sum is eq to 5;**

```
int arr[][]= {
```

```
    {11,22,55,86,90,65,89}, //dep0  
    {66,77}, //dep1
```

```
    {70,50,10,77}    //dep2  
};
```

2D ARRAY | REGULAR | USER Input

```
package com.mainapp;  
import java.util.Scanner;  
public class Launch {  
    public static void main(String[] args) {  
  
        // 3 DEP -> 4 STUDENTS IN EACH DEP  
  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.println("ENTER NO OF DEP");  
        int dept=scanner.nextInt();//3  
  
        System.out.println("ENTER NO OF STD IN EACH DEPT");  
        int std=scanner.nextInt();//4  
  
        int arr[][] = new int[dept][std]; //3 4  
  
        for (int i = 0; i < arr.length; i++) {  
  
            for (int j = 0; j < arr[i].length; j++) {  
  
                System.out.print("ENTER STD"+j+" IN  
DEPT"+i+"->");  
                arr[i][j]=scanner.nextInt();  
            }  
            System.out.println();  
        }  
    }  
}
```

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

//          for(int i=0 ;i<arr.length ; i++) { //0 1 2
//          //
//          for(int j=0 ; j < arr[i].length ; j++ ) { //0 1 2 3 4
//              //
//              System.out.print(arr[i][j]+" ");
//          }
//          System.out.println();
//      }
}

//
```

DAY-45

Java Full Stack Development

Batch 2025

2D ARRAY | JAGGED |

USER Input

```
package com.mainapp;
import java.util.Scanner;
public class Launch {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("ENTER NO OF DEP");
        int dept=scanner.nextInt(); //3

        int arr[][]=new int [dept][];
        for(int i=0 ; i<dept ; i++) { //0 1 2

            System.out.print("ENTER NO OF STD IN
DEPT-"+i+" :");
            int std=scanner.nextInt();
            arr[i]=new int[std];
        }

        // arr[0]=new int[5]; //DEPT-0 STD-5
        // arr[1]=new int[2]; //DEPT-1 STD-2
        // arr[2]=new int[3]; //DEPT-2 STD-3

        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                System.out.print("ENTER STD"+j+" IN
DEPT"+i+"->");
                arr[i][j]=scanner.nextInt();
            }
        }
    }
}
```

```

        }
        System.out.println();
    }

    for( int k[] : arr ) {
        for( int elm : k ) {
            System.out.print(elm+" ");
        }
        System.out.println();
    }

//    for(int i=0 ;i<arr.length ; i++) { //0 1 2
//
//        for(int j=0 ; j < arr[i].length ; j++ ) { //0
1 2 3 4 5 6
//
//            System.out.print(arr[i][j]+" ");
//
//        }
//        System.out.println();
//
//    }
}

}

```

ENTER NO OF DEP

3

ENTER NO OF STD IN DEPT-0 :5
ENTER NO OF STD IN DEPT-1 :2
ENTER NO OF STD IN DEPT-2 :3
ENTER STD0 IN DEPT0->11
ENTER STD1 IN DEPT0->67
ENTER STD2 IN DEPT0->56
ENTER STD3 IN DEPT0->32
ENTER STD4 IN DEPT0->78

ENTER STD0 IN DEPT1->59
ENTER STD1 IN DEPT1->43

```
ENTER STD0 IN DEPT2->43
ENTER STD1 IN DEPT2->87
ENTER STD2 IN DEPT2->54
```

```
11 67 56 32 78
59 43
43 87 54
```

3D ARRAY | REGULAR | Programmer Input

Ex. 2 CLG 3DEPT 4STD

- SET OF 2D ARRAY

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //2 CLG 3DEPT 4STD
        int arr[][][] = {

            {
                {11,22,33,44},
                {44,55,66,77},
                {88,66,55,44}
            },
            {
                {17,98,76,43},
                {65,78,43,56},
                {11,88,65,32}
            }
        };
    }
}
```

```
        }
    };

    for(int i=0 ; i<arr.length ; i++) { //0 1
        for(int j=0 ; j<arr[0].length ; j++) { //0 1 2
            for(int k=0 ; k<arr[0][0].length ; k++) { //0 1 2 3
                System.out.print(arr[i][j][k]+" ");
            }
            System.out.println();
        }
        System.out.println("\n");
    }
}
```

3D ARRAY | JAGGED |

Programmer Input

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //2 CLG 3DEPT 4STD
        int arr[][][] = {
            {
                {11, 22, 33, 44},
                {44, 55, 66}
            },
            {
        }
```

```

        {17,98},
        {65,78,43,56},
        {11,88,65,32}
    }
};

for(int i=0 ; i<arr.length ; i++) { //0 1

    for(int j=0 ; j<arr[i].length ; j++) {//i=0 j=
0 1 | i=1 j=0 1 2

        for(int k=0 ; k<arr[i][j].length ; k++)
{ //0 1 2 3
            System.out.print(arr[i][j][k]+" ");
        }
        System.out.println();
    }
    System.out.println("\n");
}
}
}

```

DAY-46

Java Full Stack Development
Batch 2025

3D ARRAY | REGULAR |

User Input

```
package com.mainapp;

import java.util.Scanner;

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

        // 2 CLG 3DEPT 4STD

        Scanner scanner = new Scanner(System.in);

        int clg,dept,std;
        System.out.println("ENTER CLG");
        clg=scanner.nextInt();
        System.out.println("ENTER DEPT IN EACH CLG");
        dept=scanner.nextInt();
        System.out.println("ENTER STD IN EACH DEPT");
        std=scanner.nextInt();

        int arr[][][] = new int[clg][dept][std]; // TOTAL=24

        for (int i = 0; i < arr.length; i++) {

            for (int j = 0; j < arr[i].length; j++) {

                for (int k = 0; k < arr[i][j].length; k++) {

                    System.out.println("ENTER STD:" + k + " IN CLG-" + i
+ " DEPT-" + j);
                    arr[i][j][k] = scanner.nextInt();
                }
                System.out.println();
            }
            System.out.println("\n");
        }

        for (int k[][] : arr) {

            for (int l[] : k) {

                for (int elm : l) {
                    System.out.print(elm + " ");
                }
            }
        }
    }
}
```

```
        System.out.println();
    }
    System.out.println("\n");
}

// for (int i = 0; i < arr.length; i++) {
//
//     for (int j = 0; j < arr[i].length; j++) {
//
//         for (int k = 0; k < arr[i][j].length; k++) {
//
//             System.out.print(arr[i][j][k] + " ");
//         }
//         System.out.println();
//     }
//     System.out.println("\n");
}

}
```

3D ARRAY | JAGGED |

User Input

```
int arr[][][] = new int[2][][]; //2 CLG

arr[0]=new int[3][]; //CLG-0 : 3 DEP
arr[1]=new int[2][]; //CLG-1 : 2 DEP

arr[0][0]=new int[5]; //CLG-0 ,DEPT-0 : 5 STD
arr[0][1]=new int[2]; //CLG-0 ,DEPT-1 : 2 STD
arr[0][2]=new int[3]; //CLG-0 ,DEPT-2 : 3 STD

arr[1][0]=new int[4]; //CLG-1 ,DEPT-0 : 4 STD
arr[1][1]=new int[2]; //CLG-1 ,DEPT-1 : 2 STD
```

```
package com.mainapp;
```

```

import java.util.Scanner;
public class Launch {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("ENTER NO CLG");
        int clg=scanner.nextInt();

        int arr[][][] = new int[clg][][];
        //2 CLG

        //FOR DECIDING NO OF DEPT IN CLG
        for(int i=0 ; i<clg ; i++) { //0 1

            System.out.println("ENTER NO OF DEPT IN CLG-"+i);
            int dept=scanner.nextInt();
            arr[i]=new int[dept][];
            //clg0-3dept , clg1-4dept
        }

        //FOR NO OF STD IN CLG-DEP
        for(int i=0 ; i<arr.length ; i++) { //0 1
            for(int j=0 ; j<arr[i].length ; j++) { //i=0, j=0 1 2
                System.out.println("ENTER NO OF STD IN CLG-"+i+""
DEPT-"+j);
                int std=scanner.nextInt();
                arr[i][j]=new int[std];
            }
        }

        for (int i = 0; i < arr.length; i++) {

            for (int j = 0; j < arr[i].length; j++) {

                for (int k = 0; k < arr[i][j].length; k++) {

                    System.out.println("ENTER STD:" + k + " IN
CLG-" + i + " DEPT-" + j);
                    arr[i][j][k] = scanner.nextInt();
                }
                System.out.println();
            }
            System.out.println("\n");
        }

        for (int k[][] : arr) {

            for (int l[] : k) {

```

```
        for (int elm : l) {
            System.out.print(elm + " ");
        }
        System.out.println();
    }
}
```

```
ENTER NO CLG
2
ENTER NO OF DEPT IN CLG-0
3
ENTER NO OF DEPT IN CLG-1
4
ENTER NO OF STD IN CLG-0 DEPT-0
3
ENTER NO OF STD IN CLG-0 DEPT-1
4
ENTER NO OF STD IN CLG-0 DEPT-2
2
ENTER NO OF STD IN CLG-1 DEPT-0
1
ENTER NO OF STD IN CLG-1 DEPT-1
2
ENTER NO OF STD IN CLG-1 DEPT-2
4
ENTER NO OF STD IN CLG-1 DEPT-3
3
ENTER STD:0 IN CLG-0 DEPT-0
12
ENTER STD:1 IN CLG-0 DEPT-0
45
ENTER STD:2 IN CLG-0 DEPT-0
32

ENTER STD:0 IN CLG-0 DEPT-1
45
ENTER STD:1 IN CLG-0 DEPT-1
65
```

ENTER STD:2 IN CLG-0 DEPT-1

32

ENTER STD:3 IN CLG-0 DEPT-1

46

ENTER STD:0 IN CLG-0 DEPT-2

34

ENTER STD:1 IN CLG-0 DEPT-2

76

ENTER STD:0 IN CLG-1 DEPT-0

76

ENTER STD:0 IN CLG-1 DEPT-1

45

ENTER STD:1 IN CLG-1 DEPT-1

43

ENTER STD:0 IN CLG-1 DEPT-2

54

ENTER STD:1 IN CLG-1 DEPT-2

67

ENTER STD:2 IN CLG-1 DEPT-2

87

ENTER STD:3 IN CLG-1 DEPT-2

54

ENTER STD:0 IN CLG-1 DEPT-3

43

ENTER STD:1 IN CLG-1 DEPT-3

56

ENTER STD:2 IN CLG-1 DEPT-3

78

12 45 32

45 65 32 46

34 76

76

45 43

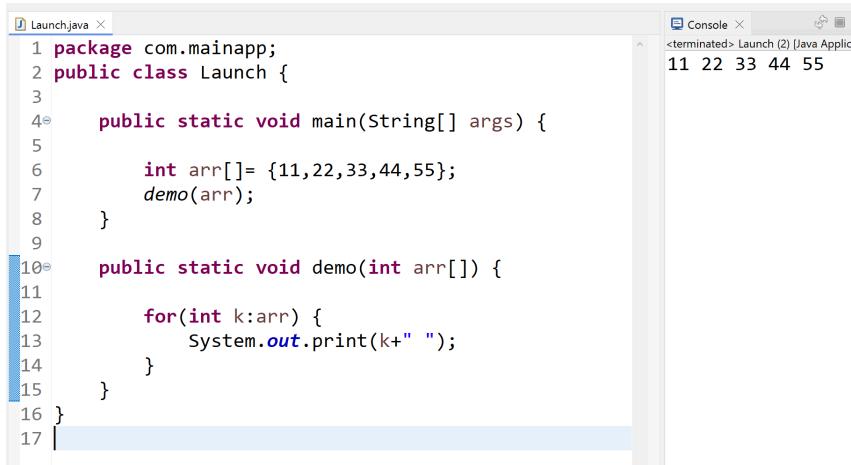
54 67 87 54

43 56 78

DAY-47

Java Full Stack Development Batch 2025

1. Array Passing



The screenshot shows a Java development environment with two panes. The left pane, titled 'Launch.java', contains the following code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int arr[] = {11,22,33,44,55};
7         demo(arr);
8     }
9
10    public static void demo(int arr[]) {
11
12        for(int k:arr) {
13            System.out.print(k+" ");
14        }
15    }
16 }
17 |
```

The right pane, titled 'Console', shows the output of the program:

```
<terminated> Launch (2) Java Application
11 22 33 44 55
```

The screenshot shows a Java IDE interface with two tabs: 'Launch.java' and 'Console'. The 'Launch.java' tab contains the following code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int arr[] = demo();
7         for(int k: arr) System.out.println(k);
8     }
9
10    public static int[] demo() {
11
12        int arr[] = {11, 22, 33, 44, 55};
13        return arr;
14
15    }
16}
17
```

The 'Console' tab shows the output of the program:

```
11
22
33
44
55
```

2. ANONYMOUS ARRAY

The screenshot shows a Java IDE interface with two tabs: 'Launch.java' and 'Console'. The 'Launch.java' tab contains the following code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         int getarr[] = demo(10);
7         for(int k: getarr) System.out.println(k);
8     }
9
10    public static int[] demo(int size) {
11
12        return new int[size]; // ANONYMOUS ARRAY
13
14    }
15}
16
```

The 'Console' tab shows the output of the program:

```
0
0
0
0
0
0
0
0
0
0
```

//40 MIN -> 2 QUESTION (1 SOLVE): 1
EASY-MID , 1 MID-HARD

Neil Sardesai owns Kiddeo corp, a shop for kid's accessories. There are two categories of items with an id number associated with them as follows:

- If the first digit in the id is 1, then the item cost is 50 each.
- If the first digit in the id is 2, then the item cost is 100 each.

He also provides a discount of 10% if the total purchase cost is more than 5,000.

Given the item id and the number of items, display the total amount to be paid by the customer.

Input

- Single line containing item id followed by a number of items. Both values must be given space-separated as shown in the example.

Output:

- Single integer displaying the total amount to be paid.

Example 1

- Input: 12345 12
- Output: 600

Example 2

- Input: 26951 60
- Output: 5400

```
package com.mainapp;
public class Launch {
```

```

public static void main(String[] args) {

    int id=234567;    //6digit
    int noOfItems=60;

    int startID=id/100000;
    if(startID==1) {
        //CAL
        int totalcost=noOfItems*50;
        if(totalcost>5000) {
            //10 DISCOUNT
            int tenper=(totalcost*10)/100;
            System.out.println(totalcost-tenper);
        }
        else {
            System.out.println(totalcost);
        }
    }
    else {
        //CAL
        int totalcost=noOfItems*100;

        if(totalcost>5000) {
            //10 DISCOUNT
            int tenper=(totalcost*10)/100;
            System.out.println(totalcost-tenper);
        }
        else {
            System.out.println(totalcost);
        }
    }
}

```

Ema Harper, a teacher, uses an array to track student attendance in a class. The array contains a series of 1s and 0s, where:

- 1 represents that the student was present.
- 0 represents that the student was absent.

The variable k represents the minimum percentage of classes the student must attend to be eligible for an upcoming examination.

Write a program to determine if the student is eligible for the exam based on the attendance record.

Input:

- A string attendance consisting of 1s and 0s, representing the attendance record.
- An integer k representing the minimum percentage of classes the student must attend.

Output:

- Output "YES" followed by the actual attendance percentage if the student is eligible for the exam.
- Output "NO" followed by the actual attendance percentage if the student is not eligible for the exam.

Example 1:

- Input: attendance = [1,0,1,0,1,0,1], k = 70
- Output: NO 57

Example 2:

- Input: attendance = [1,1,1,0,1,0,1], k = 60
- Output: YES 71

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        int arr[] = {1,1,1,0,1,0,1};
        int k = 60;
```

```

int present=0;
for(int i=0;i<arr.length;i++) {
    if(arr[i]==1) {
        present++;
    }
}

int per=(present*100)/arr.length;

if(per<k) {
    System.out.println("NO "+per);
}else {
    System.out.println("YES "+per);
}
}
}

```

TASK:

There are 8 prison cells in a row and each cell is either occupied or vacant.

Each day, whether the cell is occupied or vacant changes according to the following rules:

If a cell has two adjacent neighbors that are both occupied or both vacant, then the cell becomes occupied.

Otherwise, it becomes vacant.

Note that because the prison is a row, the first and the last cells in the row can't have two adjacent neighbors.

You are given an integer array cells where cells[i] == 1 if the i-th cell is occupied and cells[i] == 0 if the i-th cell is vacant, and you are given an integer n.

Return the state of the prison after n days (i.e., n such changes described above).

Example 1:

Input: cells = [0,1,0,1,1,0,0,1], n = 7

Output: [0,0,1,1,0,0,0,0]

[0,1,0,1,1,0,0,1] after 7 days

[0,1,1,0,0,0,0,0] day-1

[0,0,0,0,1,1,1,0] day-2

[0,1,1,0,0,1,0,0] day-3

[0,0,0,0,0,1,0,0] day-4

[0,1,1,1,0,1,0,0] day-5

[0,0,1,0,1,1,0,0] day-6

[0,0,1,1,0,0,0,0] day-7

DAY-48

Java Full Stack Development Batch 2025

String

->Strings are set of character

Ex. itscodehunt@gmail.com

->Widely used in software Ex. name

->String are object in Java

->String manipulation

Types of String

1. Mutable String Ex. CURRENT ADDRESS

2. Immutable String Ex.PAN NO

Immutable String

1. with new keyword

2. without new keyword

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String s="raju";
        System.out.println(s);

        String k=new String("raju");
        System.out.println(k);
    }
}
```

String literals

String s="raju";

String k="raju";

String p=new String("INDIA");
System.out.println(p);

String q=new String("INDIA");
System.out.println(q);

RAM HEAP

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String s="raju";
        System.out.println(s);

        String k="raju";
        System.out.println(k);

        if(s==k) {
            System.out.println("REF ARE EQ");
        }

        String p=new String("INDIA");
        System.out.println(p);

        String q=new String("INDIA");
        System.out.println(q);
    }
}
```

```
        if(p==q) {
            System.out.println("REF ARE EQ");
        }
        else {
            System.out.println("REF ARE NOT EQ");
        }
    }
}
```

```
String s="raju"; //1st store
String k="raju"; //ref pass
String extra = s.concat("HIII"); //rajuHIII
System.out.println(s); //HIII RAJU , RAJU HIII, HIII, RAJU
System.out.println(k); //HIII RAJU , RAJU HIII, HIII, RAJU
```

RAM HEAP

```
package com.mainapp;
public class Launch {
```

```

public static void main(String[] args) {

    String s="raju"; //1st store

    String k="raju"; //ref pass

    String extra = s.concat("HIII"); //rajuHIII

    System.out.println(s); //HIII RAJU , RAJU HIII, HIII, RAJU
    System.out.println(k); //HIII RAJU , RAJU HIII, HIII, RAJU

}
}

```

String methods (String Manipulation)

1.concat return Type: String

```

package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String s="raju";
        String k="kaju";

        String res=s.concat(k);
        System.out.println(res);

    }
}

```

2.equals return Type: boolean

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String s=new String("raju");
        String k=new String("raju");

        if(s.equals(k)) {
            System.out.println("STRINGS ARE EQ");
        }
        else {
            System.out.println("STRING ARE NOT EQ");
        }

    }
}
```

NOTE:

String name=scanner.next(); //without space

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("ENTER UR NAME");
        String name=scanner.next();

        System.out.println(name);

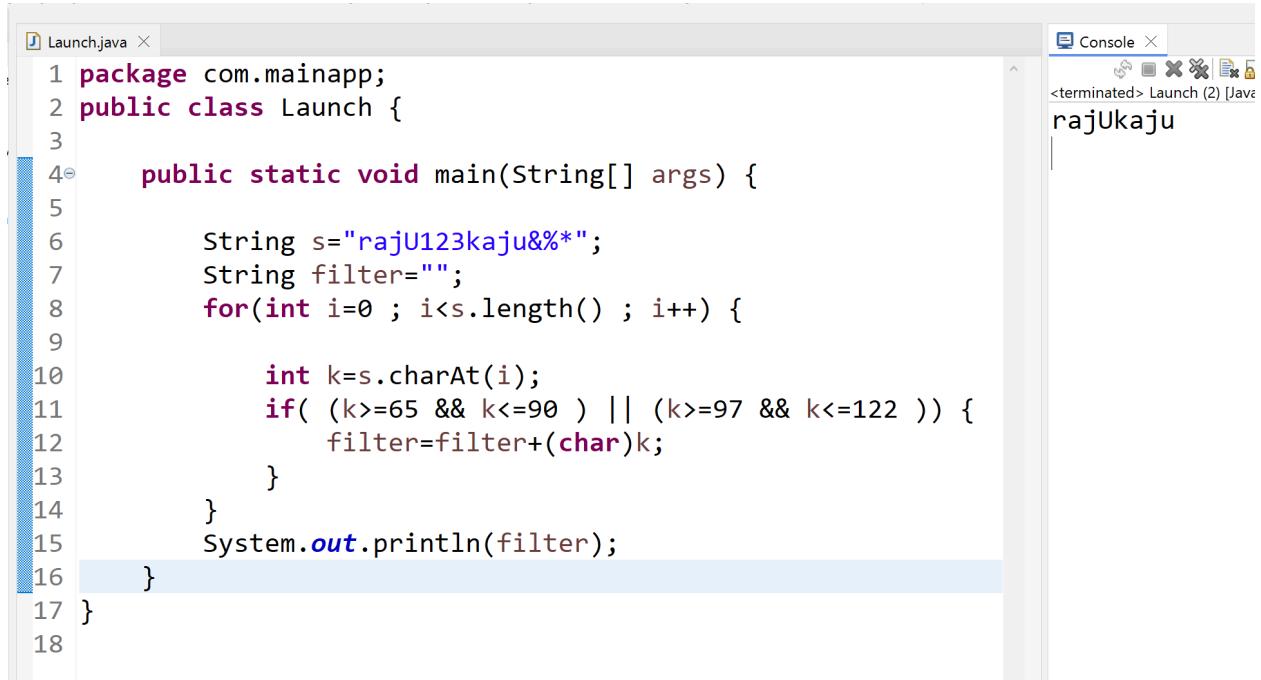
    }
}
```

DAY-49

Java Full Stack Development Batch 2025

3.length return Type: int

4.charAt return Type: char

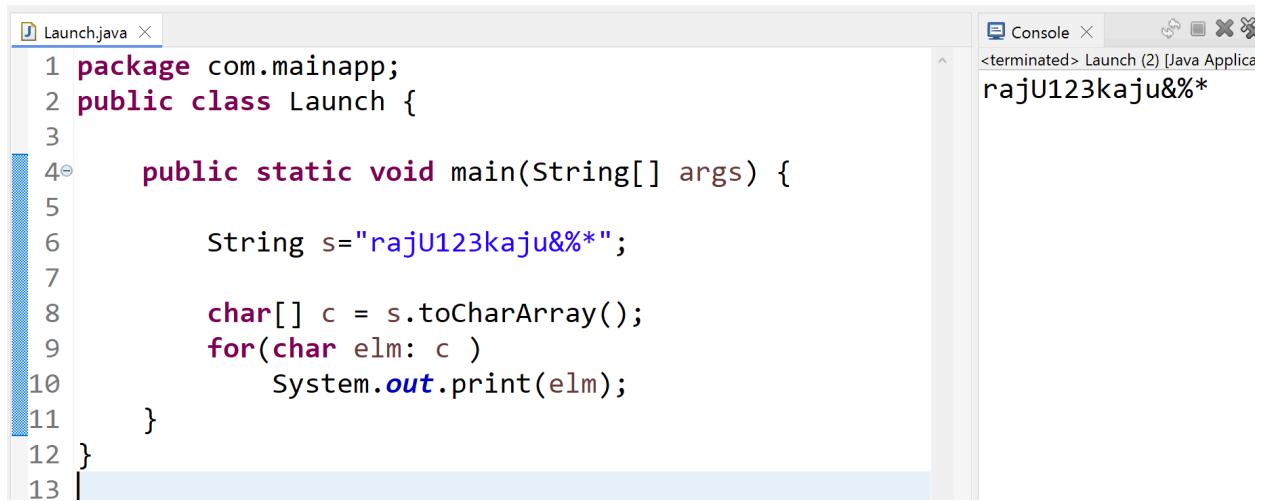


The screenshot shows an IDE interface with two main panes. The left pane displays the code for a Java class named 'Launch'. The right pane shows the 'Console' output.

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s="rajU123kaju&%*";
7         String filter="";
8         for(int i=0 ; i<s.length() ; i++) {
9
10             int k=s.charAt(i);
11             if( (k>=65 && k<=90 ) || (k>=97 && k<=122 ) ) {
12                 filter=filter+(char)k;
13             }
14         }
15         System.out.println(filter);
16     }
17 }
18 }
```

The console output shows the string "rajU123kaju" printed, which is the result of filtering non-alphabetic characters from the input string "rajU123kaju&%*".

5.toCharArray return Type: char[]



The screenshot shows a Java code editor and a terminal window. The code in the editor is:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s="rajU123kaju&%*";
7
8         char[] c = s.toCharArray();
9         for(char elm: c )
10            System.out.print(elm);
11    }
12 }
13 }
```

The terminal window shows the output of the program:

```
Console <terminated> Launch (2) [Java Application]
rajU123kaju&%*
```

6.endsWith return Type: boolean

7.startsWith return Type: boolean

```
package com.mainapp;
import java.util.Scanner;
public class Launch {

    public static void main(String[] args) {

        // Scanner scan=new Scanner(System.in);
        // System.out.println("ENTER UR EMAIL");
        // String email=scan.next();

        String email="raju@gmail.com";

        boolean endsWith = email.endsWith("co");
        if(endsWith){
            System.out.println("VALID EMAIL");
        }
        else {
```

```

        System.out.println("INVALID EMAIL");
    }

}

Launch.java ×
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         Scanner scan=new Scanner(System.in);
8         System.out.println("ENTER UR EMAIL");
9         String email=scan.next();
10
11         String email="empraju@gmail.com";
12
13         boolean endsWith = email.startsWith("emp");
14         if(endsWith){
15             System.out.println("VALID EMAIL");
16         }
17         else {
18             System.out.println("INVALID EMAIL");
19         }
20
21     }
22 }

```

The screenshot shows a Java application running in an IDE. The code in the editor checks if an input string starts with 'emp'. If it does, it prints 'VALID EMAIL'; otherwise, it prints 'INVALID EMAIL'. The output window shows the string 'INVALID EMAIL'.

8.equalsIgnoreCase return Type: boolean

```

package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String username="abcd123"; //database

        String userinput="ABCd123";

```

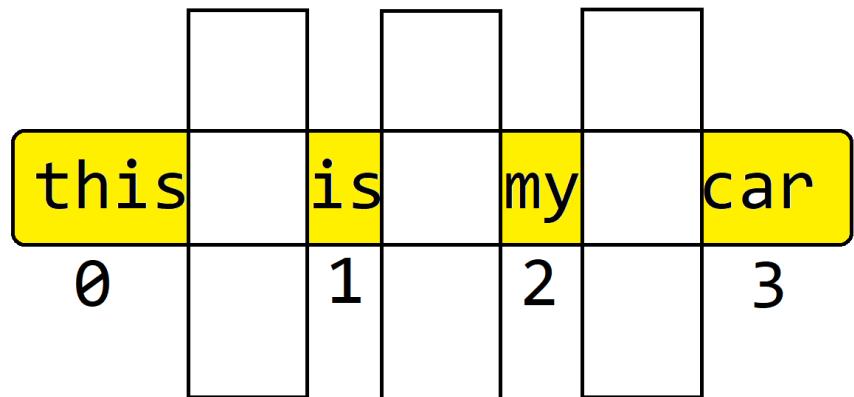
```

        boolean b = userinput.equalsIgnoreCase(username);
        System.out.println(b);
    }
}

```

9.split return Type: String[]

String[]



```

Launch.java ×
1 package com.mainapp;
2
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         String msg = "this is my car";
8         String[] split = msg.split(" ");
9         for (String elm : split)
10             System.out.println(elm);
11
12     }
13 }
14

```

Console × <terminated> Launch (2) [JRE]
this
is
my
car

TASK:

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String msg = "this is my car";
        int k=3;
        //this is my car

    }
}
```

TASK:

```
package com.mainapp;
public class Launch {

    public static void main(String[] args) {

        String s = "abcdzAXZ";
        int shift=5; //0=<shift<=26
        //fghieFCE

    }
}
```

TASK:

```
Launch.java ×
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6
7         String s = "malayalam";
8         String rev="";//dcba
9         |
10        for(int i=s.length()-1 ; i>=0 ; i--) {
11            rev=rev+s.charAt(i);
12        }
13
14        if(s.equals(rev)) {
15            System.out.println("PALINDORME");
16        }else {
17            System.out.println("NOT PALINDORME");
18        }
19    }
20}
21
```

Console ×
<terminated> Launch (2)
PALINDORME

DAY-50

**Java Full Stack Development
Batch 2025**

10.getBytes return byte[]

The screenshot shows a Java code editor with a file named "Launch.java". The code prints lowercase letters and specific characters from a string to the console. The console output shows the expected characters.

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6
7         String s = "abcdzAXZ%$qqqq";
8         byte[] b = s.getBytes();
9         for(byte elm:b) {
10             if( (elm>=65 && elm<=90) ||(elm>=97 && elm<=122))
11                 System.out.println((char)elm);
12         }
13     }
14 }
15 }
```

Console Output:

```
a
b
c
d
z
A
X
Z
q
q
q
q
```

11.toUpperCase return String

12.toLowerCase return String

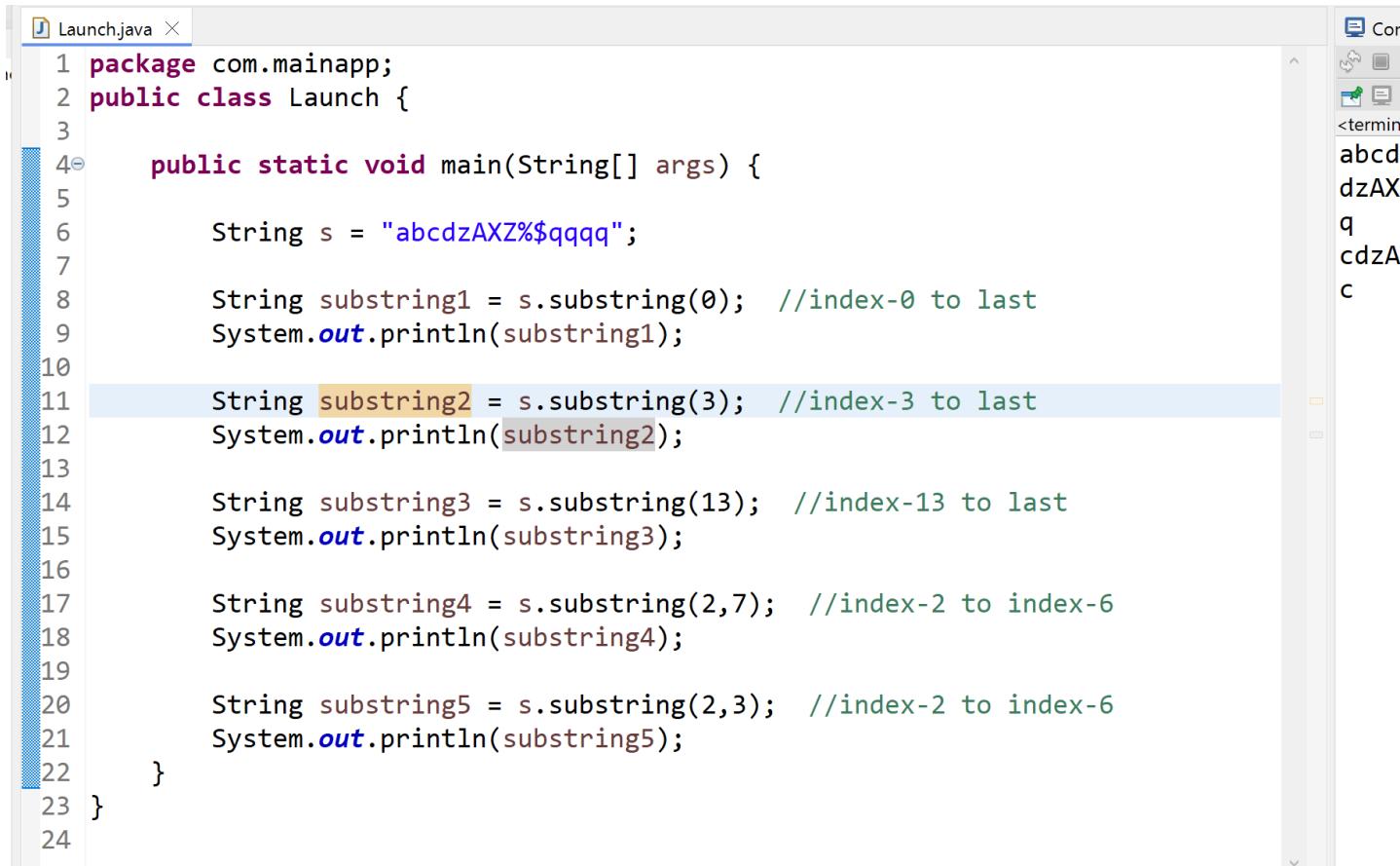
The screenshot shows a Java code editor with a file named "Launch.java". The code demonstrates the use of `toUpperCase()` and `toLowerCase()` methods on a string. The console output shows the uppercase and lowercase versions of the input string.

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6
7         String s = "abcdzAXZ%$qqqq";
8
9         String upperCase = s.toUpperCase();
10        System.out.println(upperCase);
11
12        String lowerCase = s.toLowerCase();
13        System.out.println(lowerCase);
14
15    }
16 }
17 }
```

Console Output:

```
ABCDZAXZ%$QQQQ
abcdzaxz%$qqqq
```

13.substring return String



The screenshot shows a Java code editor with the file 'Launch.java' open. The code demonstrates various uses of the `substring` method on a string 's' containing the value 'abcdzAXZ%\$qqqq'. The code prints five different substrings to the console:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s = "abcdzAXZ%$qqqq";
7
8         String substring1 = s.substring(0); //index-0 to last
9         System.out.println(substring1);
10
11        String substring2 = s.substring(3); //index-3 to last
12        System.out.println(substring2);
13
14        String substring3 = s.substring(13); //index-13 to last
15        System.out.println(substring3);
16
17        String substring4 = s.substring(2,7); //index-2 to index-6
18        System.out.println(substring4);
19
20        String substring5 = s.substring(2,3); //index-2 to index-6
21        System.out.println(substring5);
22    }
23
24 }
```

The code editor interface includes tabs for 'Launch.java', 'Console', and 'Terminal', and a sidebar with file navigation.

14.contains return boolean

The screenshot shows a Java code editor with a file named "Launch.java" open. The code contains a main method that prints "true" to the console. The "Console" tab shows the output "true".

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s = "abcdzAXZ%$qqqq";
7
8         boolean contains = s.contains("abc");
9         System.out.println(contains);
10    }
11 }
12 |
```

15.indexOf return int

The screenshot shows a Java code editor with a file named "Launch.java" open. The code contains a main method that prints the index of the character 'b' in the string "abcdzAXZ%\$qqqq" to the console. The "Console" tab shows the output "1".

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s = "abcdzAXZ%$qqqq";
7
8         int indexOf = s.indexOf("b");
9         System.out.println(indexOf);
10
11 //TASK: FIND ALL THE INDEX OF b
12    }
13 }
14 |
```

16.lastIndexOf return int

The screenshot shows a Java code editor with a file named "Launch.java". The code contains a main method that prints the last index of the character 'b' in a string. The output window shows the number 5, indicating the index of the last 'b' character.

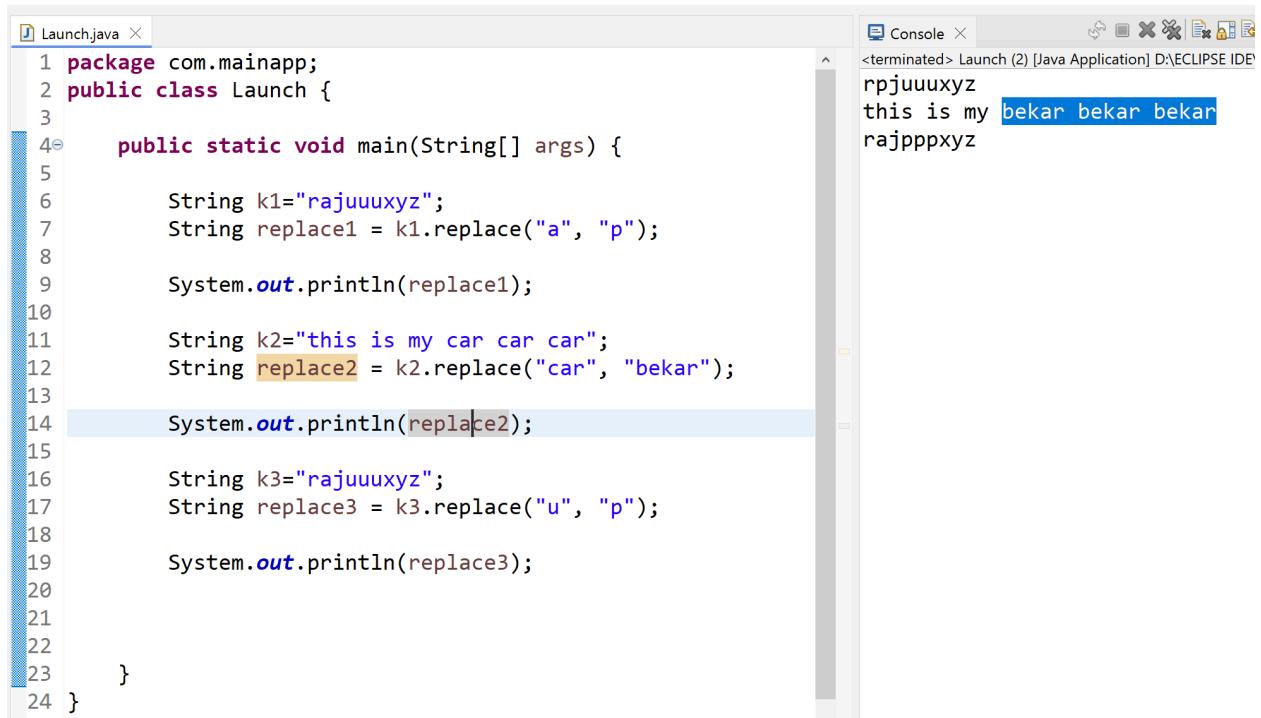
```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String s = "abcbdbzAXZ%$qqqq";
7
8         int indexOf = s.lastIndexOf("b");
9         System.out.println(indexOf);
10    }
11 }
12
```

17.intern (FROM NON CONSTANT TO CONSTANT)

The screenshot shows a Java code editor with a file named "Launch.java". The code demonstrates the use of the intern() method. It creates a constant string "raju" and a new string "raju", then prints the interned version of the second string. The output window shows the word "raju", indicating that both strings point to the same entry in the constant pool.

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String k="raju";//CONSTANT pool
7
8         String s=new String("raju");
9         String p = s.intern();
10
11         System.out.println(p);
12
13    }
14 }
```

18.replace return String



The screenshot shows the Eclipse IDE interface. On the left, the code editor window titled "Launch.java" displays the following Java code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String k1="rajuuuxyz";
7         String replace1 = k1.replace("a", "p");
8
9         System.out.println(replace1);
10
11        String k2="this is my car car car";
12        String replace2 = k2.replace("car", "bekar");
13
14        System.out.println(replace2);
15
16        String k3="rajuuuxyz";
17        String replace3 = k3.replace("u", "p");
18
19        System.out.println(replace3);
20
21
22    }
23
24 }
```

On the right, the "Console" window shows the output of the application:

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE
rpjuuuxyz
this is my bekar bekar bekar
rajpppxyz
```

19.trim return String

The screenshot shows a Java IDE interface. On the left, the code editor window titled "Launch.java" contains the following Java code:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String k1="    raju uu xyz    ";
7         String trim = k1.trim();
8         System.out.println(trim);
9
10    }
11
12 }
```

On the right, the "Console" window shows the output of the program:

```
<terminated> Launch (2)
raju uu xyz
```

20.compareTo return int

Lexicographically (it compares Unicode value of each character one by one and where it will find difference at that point only this method returns the difference)

0 : if both string are eq

Negative: if first string's character is less than second

Positive: if first string's character is greater than second

The screenshot shows a Java IDE interface. On the left is the code editor with a file named "Launch.java". The code contains a main method that compares two strings ("abcdefg" and "abcdefgh") using the compareTo method and prints the result (-3). On the right is the "Console" window, which displays the output "3", indicating a bug or unexpected behavior.

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args) {
5
6         String k1="abcdefg"; //100
7         String k2="abcdefgh"; //103
8
9         int compareTo = k1.compareTo(k2); //100-103
10        System.out.println(compareTo); //-3
11    }
12 }
13
14
```

TASK:

String s="this is my car";

Output: This Is My Car

- String buffer builder (MUTABLE)
- String user input
- Escape
- Task

DAY-51

Java Full Stack Development Batch 2025

MUTABLE STRING

In Java we can create mutable String in two ways

1.StringBuffer (class: java.lang)

- Since early version
- Synchronized
- Slower than StringBuilder
- USE IN MULTITHREADED ENV (MULTITASKING)

2.StringBuilder (class: java.lang)

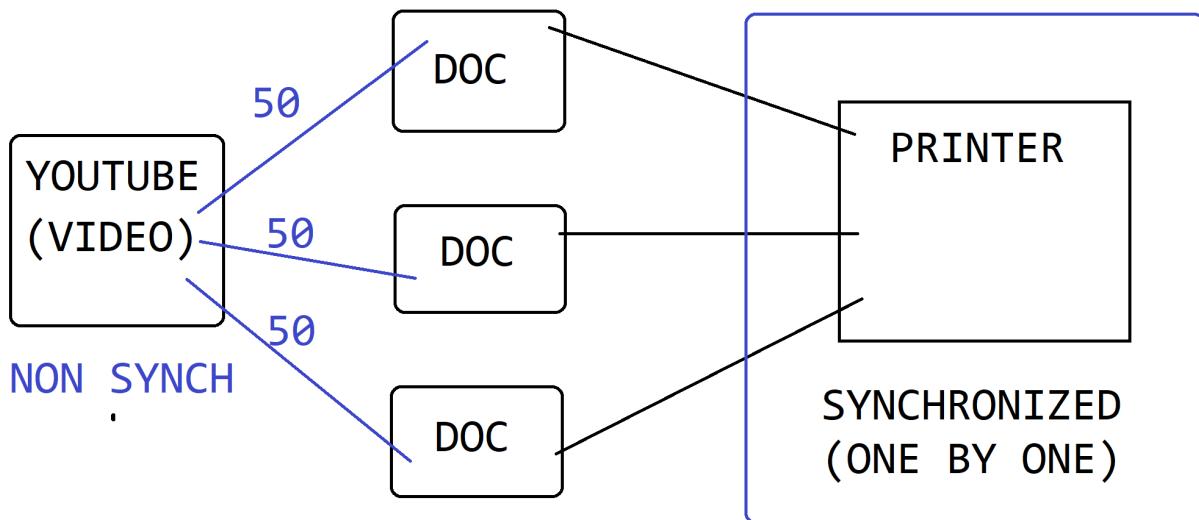
- From java 1.5
- Non Synchronized
- Faster Than StringBuffer
- USE IN SINGLE THREADED ENV (SINGLE TASKING)

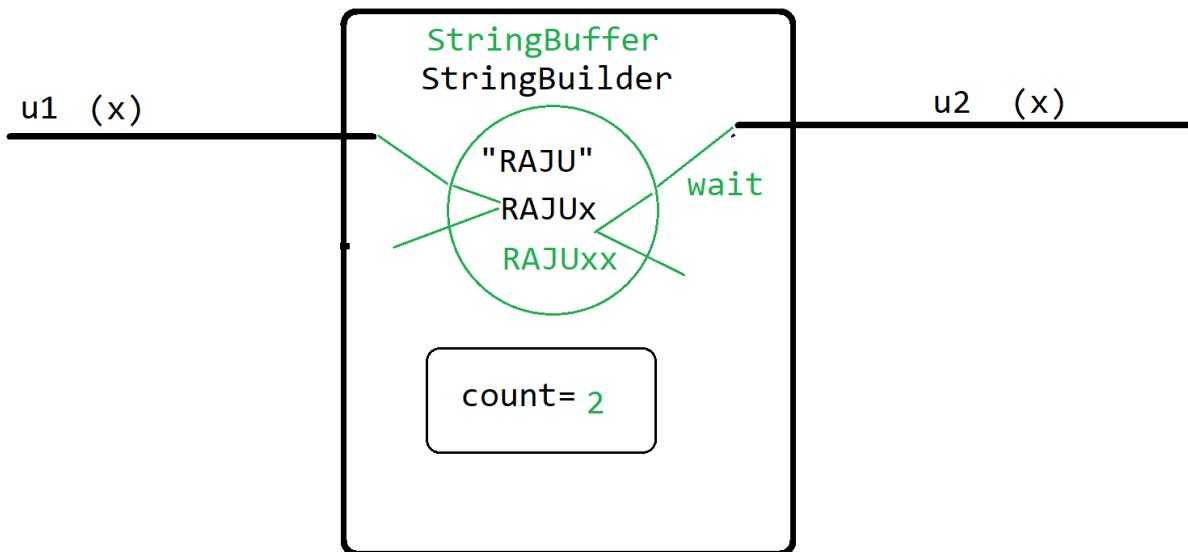
SYNCRONIZATION

If a user is accessing a resource R then at that time other users can't access the same resource

NON SYNCHRONIZATION

If multiple users can access a resource R at the same time.





```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        //        StringBuffer s = new StringBuffer("raju");
        //        System.out.println(s);
        //        s.append("abcd");
        //        System.out.println(s);
        //        s.insert(2, "**");
        //        System.out.println(s);
        //        s.delete(4, 6);
        //        System.out.println(s);
        //        s.reverse();
        //        System.out.println(s+" "+s.length());

        StringBuilder s = new StringBuilder("raju");
        System.out.println(s);
        s.append("abcd");
        System.out.println(s);
        s.insert(2, "**");
        System.out.println(s);
        s.delete(4, 6);
        System.out.println(s);
        s.reverse();
        System.out.println(s+" "+s.length());
    }
}

```

}

DAY-52

Java Full Stack Development Batch 2025

1. How to take String type user input

1.word (next())

2.sentence (nextLine())

```
package com.mainapp;
import java.util.Scanner;
public class Launch {
    public static void main(String[] args) {

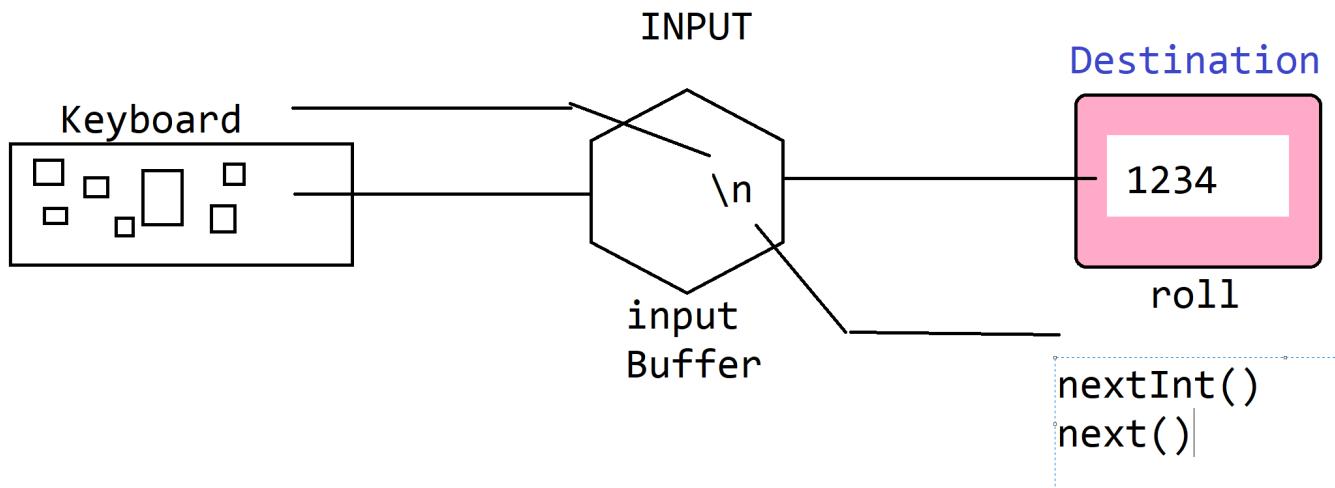
        Scanner scanner = new Scanner(System.in);

        System.out.println("ENTER YOUR ROLL");
        int roll=scanner.nextInt(); //1234 \n
        System.out.println(roll);

        //INPUT BUFFER: \n
        scanner.nextLine(); //IT IS USED TO CONSUME \n

        System.out.println("ENTER YOUR FULL NAME");
        String name1=scanner.nextLine();
        System.out.print(name1);
        System.out.println(name1.isEmpty());

        System.out.println("ENTER YOUR NAME");
        String name2=scanner.next();
        System.out.print(name2);
    }
}
```



Where it will get stored?

Ans: NON CONSTANT POOL

1.String pool lookups are computationally expensive

```

Launch.java ×
1 package com.mainapp;
2 import java.util.Scanner;
3
4 public class Launch {
5     public static void main(String[] args) {
6
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.println("ENTER YOUR NAME");
10        String name=scanner.next(); //raju NON CONSTANT
11        System.out.println(name);
12
13        String x="raju"; //CONSTANT POOL
14
15        System.out.println(name==x);
16    }
17 }

```

Console ×

<terminated> Launch

ENTER YOUR NAME
raju
raju
false

2.when u create a string using concat(), substring()..etc it will get stored inside the NON CONSTANT POOL

The screenshot shows the Eclipse IDE interface. On the left is the code editor with a file named 'Launch.java'. The code contains a main method that creates two strings: 's' with value 'raju' and 'k' with value 'ra' concatenated with 'ju'. A comment indicates 'k' is a non-constant string. The code then prints the result of the comparison 's==k'. On the right is the 'Console' view, which shows the output: '<terminated> Launch false'. This demonstrates that the two strings are different objects in memory.

```
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         String s="raju";
6         String k="ra".concat("ju"); //NON CONSTANT
7
8         System.out.println(s==k);
9
10    }
11 }
12 }
```

```
<terminated> Launch
false
```

String ESCAPE SEQUENCE:

These are some special characters used to represent certain characters within a string

The screenshot shows the Eclipse IDE interface. On the left is the code editor with a file named 'Launch.java'. The code uses several System.out.println statements to print different strings with escape sequences. The sequences include '\n' (new line), '\t' (tab), '\"' (double quotes), '\\\' (backslash), and '\u265A' (unicode character). On the right is the 'Console' view, which shows the output: 'HELLO\nWORLD', 'HELLO\tWORLD', 'HELLO \"WORLD\"', 'HELLO WORLD:XYZ\\\'', 'C:\\Users\\DELL\\Desktop\\JARS', and 'UNICODE: \u265A'. This demonstrates how escape sequences are interpreted by the Java runtime.

```
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         System.out.println("HELLO\nWORLD");
6         System.out.println("HELLO\tWORLD");
7         System.out.println("HELLO \"WORLD\"");
8         System.out.println("HELLO WORLD:XYZ\\\'");
9         System.out.println("C:\\Users\\DELL\\Desktop\\JARS");
10        System.out.println("UNICODE: \u265A");
11
12    }
13 }
14 }
```

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE
HELLO
WORLD
HELLO\tWORLD
HELLO "WORLD"
HELLO WORLD:XYZ\\'
C:\\Users\\DELL\\Desktop\\JARS
UNICODE: \u265A
```

WAP TO PRINT:

HELLO\nWORLD

WAP TO PRINT ONLY REPEATED CHAR

String s="aaabbccdefbbgg

Output: abcg

DAY-53

Java Full Stack Development

Batch 2025

Exception Handling

- Exception in Java is an unexpected event or condition during the execution of a program
- Due to Exception our program terminates abnormally
- Exception occurs due to faulty inputs
- Exceptions are also called Runtime Errors
- In Java Exception Handling is a mechanism to handle Runtime Errors / Exception

- **Exception Handling ensure your program must terminate Normally even if exception occurred**

Types of Exception:

1. **Checked Exception (COMPILE TIME)**
2. **Unchecked Exception (RUNTIME)**

```
package com.mainapp;
import java.util.Scanner;
public class Launch {
    public static void main(String[] args) {

        //PROGRAM: TERMINATE(Abnormal Termination)

        // Scanner scanner = new Scanner(System.in);
        // System.out.println("ENTER YOUR AGE");
        // int age=scanner.nextInt(); //Exception in thread "main"
        java.util.InputMismatchException
        // System.out.println("YOUR AGE: "+age);

        // String s=null;
        // s.length(); //java.lang.NullPointerException:

        // int arr[] = {11,22,33};
        // System.out.println(arr[3]); //java.lang.ArrayIndexOutOfBoundsException:

        // Scanner scanner = new Scanner(System.in);
        // System.out.print("ENTER NUM1: ");
        // int num1=scanner.nextInt();
        // System.out.print("ENTER NUM1: ");
        // int num2=scanner.nextInt();
        //
        // int num3=num1/num2; //java.lang.ArithmetricException:
        // System.out.println(num3);

    }
}
```

How to Handle Exception:

1.try catch block (BEST)

2.throws keyword (DUCKING THE EXCEPTION)

try catch block

```
Launch.java
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Demo demo = new Demo();
6         demo.test();
7         System.out.println("EXIT");
8     }
9 }
10 }

Demo.java
1 package com.mainapp;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4 public class Demo {
5
6     public void test() {
7
8         try {
9
10             Scanner scanner = new Scanner(System.in);
11
12             System.out.print("ENTER NUM1: ");
13             int num1 = scanner.nextInt();
14
15             System.out.print("ENTER NUM2: ");
16             int num2 = scanner.nextInt();
17
18             int num3 = num1 / num2;
19             System.out.println(num3);
20
21         } catch (InputMismatchException | ArithmeticException e) {
22             System.out.println("CHECK PROG CAREFULLY");
23         }
24     }
25 }
26 
```

Console

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk
ENTER NUM1: 10
ENTER NUM2: 0
CHECK PROG CAREFULLY
EXIT
```

```

1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Demo demo = new Demo();
6         demo.test();
7         System.out.println("EXIT");
8     }
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 }

```

Console X

<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk
ENTER NUM1: 10
ENTER NUM2: 0
DO NOT DIVIDE BY ZERO
EXIT

```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Demo demo = new Demo();
        demo.test();
        System.out.println("EXIT");

    }
}

```

```

package com.mainapp;
import java.util.InputMismatchException;
import java.util.Scanner;
public class Demo {

    public void test() {

        try {

            Scanner scanner = new Scanner(System.in);
            System.out.print("ENTER NUM1: ");

```

```
        int num1 = scanner.nextInt();

        System.out.print("ENTER NUM2: ");
        int num2 = scanner.nextInt();

        int num3 = num1 / num2;
        System.out.println(num3);

    }
    catch (InputMismatchException e) {
        System.out.println(e);
        //e.printStackTrace();
        System.out.println("TAKE NUMBERS ONLY");
    }
    catch (ArithmaticException e) {
        System.out.println(e);
        //e.printStackTrace();
        System.out.println("DO NOT DIVIDE BY ZERO");
    }
    catch (Exception e) {
        System.out.println(e);
        //e.printStackTrace();
        System.out.println("SOMETHING WENT WRONG");
    }
}
}
```

DAY-54

Java Full Stack Development Batch 2025

finally block

it is used to execute important code such as Resource de-allocation, regardless of whether exception occurs or not

```
package com.mainapp;
import java.util.Scanner;
public class Demo {

    public int test() {

        Scanner scanner = new Scanner(System.in);
        //10 RES
        try {

            System.out.print("ENTER NUM1: ");
            int num1 = scanner.nextInt();
            System.out.print("ENTER NUM2: ");
            int num2 = scanner.nextInt();
            return num1+num2;

        }
        catch (Exception e) {
            System.out.println("SOMETHING WENT WRONG");
            return 0;
        }
        finally {
            System.out.println("CLOSING...");
            scanner.close();
        }
    }
}
```

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Demo demo = new Demo();
        int test = demo.test();
        System.out.println(test);
        System.out.println("EXIT");
```

```
    }  
}
```

try : invalid

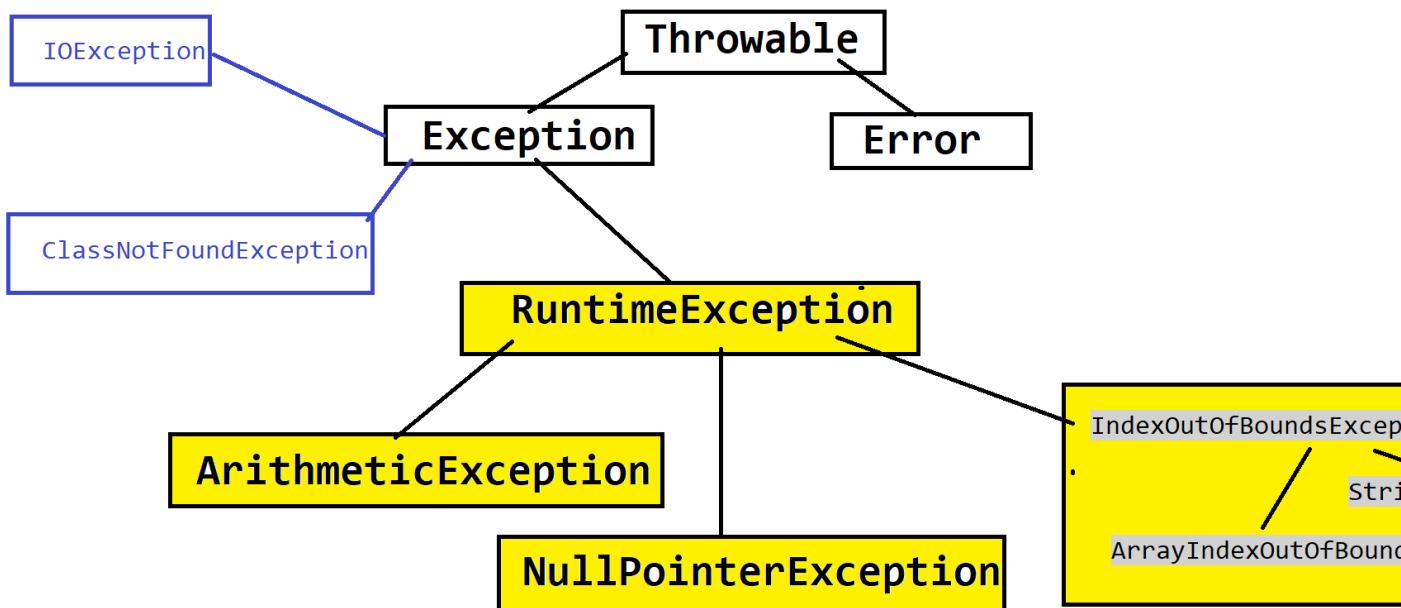
try catch: valid

try catch finally: valid

try finally:valid

- **try->try catch finally**
- **catch->try catch finally**
- **finally->try catch finally**

Exception Hierarchy



Checked Exception

- Exception occurs at compile time
- Throws keyword is used to duck the exception

The screenshot shows the Eclipse IDE interface with two open files and a console window.

Launch.java

```
1 package com.mainapp;
2 public class Launch {
3     public static void main(String[] args) {
4
5         Demo demo = new Demo();
6         int test;
7
8         try {
9             test = demo.test(); //CALLER
10            System.out.println(test);
11        } catch (Exception e) {
12            System.out.println("SOMETHING WENT WRONG");
13        }
14
15        System.out.println("EXIT");
16    }
17}
18}
```

Demo.java

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Demo {
4
5     public int test() throws
6
7
8
9
10    Scanner scanner = ne
11
12    System.out.print("EN
13    int num1 = scanner.n
14    System.out.print("EN
15    int num2 = scanner.n
16    scanner.close();
17    return num1 + num2;
18}
```

Console

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wir
ENTER NUM1: ten
SOMETHING WENT WRONG
EXIT
```

The screenshot shows the Eclipse IDE interface with two Java files open in the editor and their output in the console.

Launch.java

```
1 package com.mainapp;
2
3 public class Launch {
4     public static void main(String[] args) {
5         Demo demo = new Demo();
6         int test;
7
8         test = demo.test(); // CALLER
9         System.out.println(test);
10
11        System.out.println("EXIT");
12
13    }
14
15 }
16
```

Demo.java

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Demo {
4
5     public int test() {
6
7         Scanner scanner = new Scanner(System.in);
8         int num1 = scanner.nextInt();
9         int num2 = scanner.nextInt();
10        scanner.close();
11        return num1 + num2;
12
13    }
14
15 }
16
17 }
```

Console

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wir
ENTER NUM1: 10
ENTER NUM2: 20
30
EXIT
```

The screenshot shows an IDE interface with three main panes. The top pane displays the code for `Launch.java`, which contains a `main` method that attempts to load the `Demo` class from the `com.mainapp` package. The middle pane displays the code for `Class.class`, specifically the `forName` method of the `Class` class. The bottom right pane is the `Console`, which shows the output of the program execution.

```
1 package com.mainapp;
2 public class Launch {
3     //JVM
4     public static void main(String[] args) throws ClassNotFoundException {
5         try {
6             Class.forName("com.mainapp.Demo");
7             System.out.println("****");
8         }
9         catch (Exception e) {
10            System.out.println("CLASS NOT FOUND");
11        }
12    }
13 }
14 }
```

```
406 * @jls 12.4 Initialization of Classes and Interfaces
407 */
408 @CallerSensitive
409 public static Class<?> forName(String className)
410     throws ClassNotFoundException {
411     Class<?> caller = Reflection.getCallerClass();
412     return forName(className, caller);
413 }
414 }
```

```
<terminated> Launch
****
```

DAY-55

Java Full Stack Development
Batch 2025

throw keyword is used to invoke an exception(INBUILT OR CUSTOM)

The screenshot shows the Eclipse IDE interface with two open files and a console window.

Launch.java:

```
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args){
5         try {
6             new Demo().test();
7         } catch (ClassNotFoundException e) {
8             e.printStackTrace();
9         }
10    }
11 }
```

Demo.java:

```
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Demo {
4
5     public void test() throws ClassNotFoundException {
6
7         Scanner scanner = new Scanner(System.in);
8         System.out.println("Enter Age");
9         int age=scanner.nextInt();
10        if(age<18) {
11            //throw new ArithmeticException("Age less than 18");
12            throw new ClassNotFoundException("AGE >= 18");
13        }
14        System.out.println("YOUR AGE :" +age);
15    }
16 }
17 }
```

Console Output:

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\javaw.exe (02)
Enter Age
12
java.lang.ClassNotFoundException: AGE >= 18
    at com.mainapp.Demo.test(Demo.java:12)
    at com.mainapp.Launch.main(Launch.java:8)
```

CUSTOM EXCEPTION

- GYM APPLICATION:
- Enter age: (if age<18 : 0,1,2,3...17-> FAULTY INPUT)
- throw: InvalidAgeException (Checked or Unchecked)

- checked exception ----extends----Exception
- unchecked exception----extends-----RuntimeException

The screenshot shows the Eclipse IDE interface with three windows:

- Launch.java** (Left Window): Contains code for a main application. It imports `java.util.Scanner` and defines a `main` method that calls `Demo.test()`. A try-catch block handles `InvalidAgeException` by printing the stack trace.
- Demo.java** (Right Window): Contains code for a utility class. It imports `java.util.Scanner` and defines a `test` method. This method reads an age from `System.in` using `Scanner`, checks if it's less than 18, and if so, throws a new `InvalidAgeException` or prints "Age is valid".
- Console** (Bottom Window): Shows the application's output. It prompts for an age (Enter Age), receives the input "20", and prints "YOUR AGE:20".

```

Launch.java
1 package com.mainapp;
2 public class Launch {
3
4     public static void main(String[] args){
5
6         try {
7             new Demo().test();
8         } catch (InvalidAgeException e) {
9             // TODO Auto-generated catch block
10            e.printStackTrace();
11        }
12    }
13}
14}
15}

Demo.java
1 package com.mainapp;
2 import java.util.Scanner;
3 public class Demo {
4
5     public void test(){
6
7         Scanner scanner = new Scanner(System.in);
8         System.out.print("Enter Age");
9         int age=scanner.nextInt();
10        if(age<18) {
11            System.out.println("Age is invalid");
12        } else {
13            System.out.println("Your Age:" + age);
14        }
15    }
16}
17}
18}
19}

```

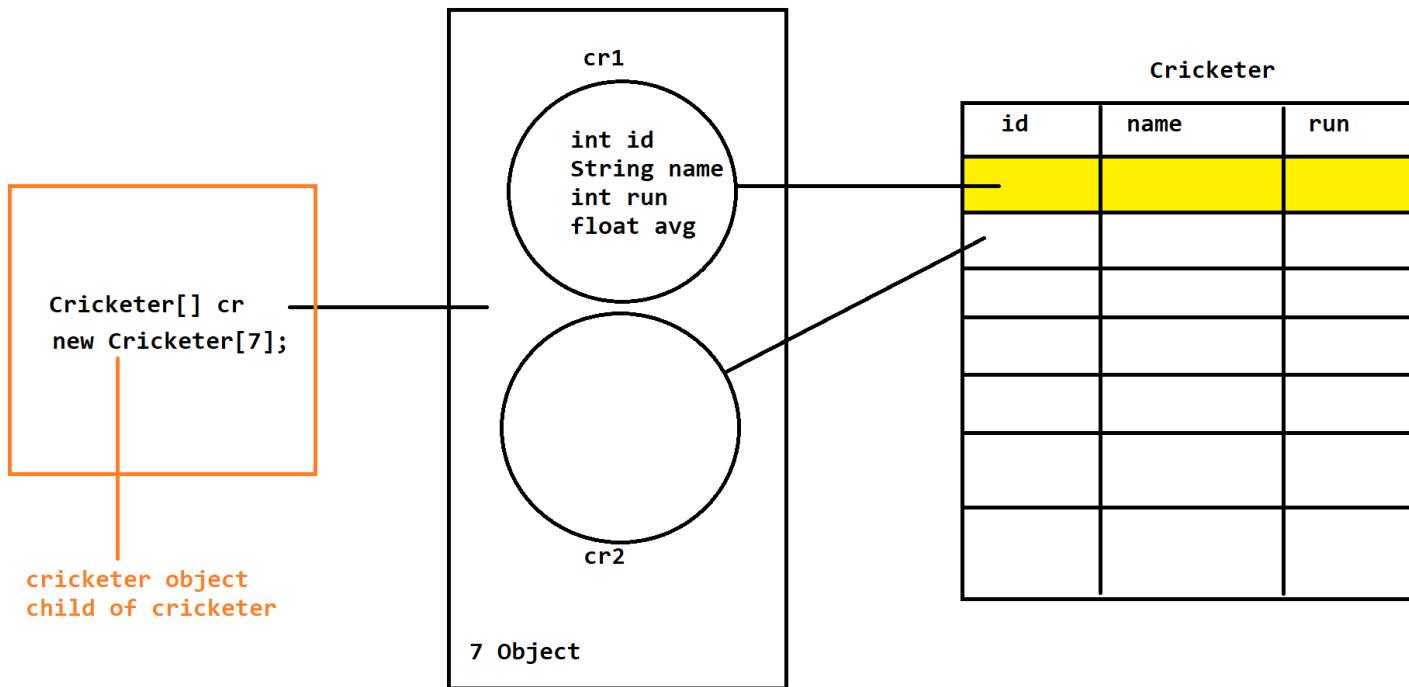
- throw throws throwable
- final finally finalize

DAY-56

Java Full Stack Development Batch 2025

Collection Framework

In java, Collection is an architecture to store and manipulate the group of Object.



Array: (Faster than Collection)

- 1.size fixed**
- 2.Homogeneous**
- 3.complex manipulation**

Collection:

- 1.Rrowsable**
- 2.Heterogeneous**
- 3.easy manipulation**

- IN COLLECTION YOU CAN ONLY INSERT OBJECT
(COLLECTION DOES NOT SUPPORT PRIMITIVE DATA
TYPE)**

WRAPPER CLASS:

- byte -> Byte**
- short -> Short**
- int -> Integer**
- long -> Long**
- float -> Float**
- double -> Double**
- char -> Character**

- boolean -> Boolean

```
int k = 100 ; //primitive data type
```

Integer //Wrapper class

We have two ways to convert primitive into its object form

1.MANUAL BOXING & MANUAL UNBOXING (BEFORE JAVA 5)

```
17 //BEFORE JAVA 5
18
19 int k=100;
20 Integer i=new Integer(k); //MANUAL BOXING
21 System.out.println(i);
22
23 int p = i.intValue(); //MANUAL UNBOXING
24 System.out.println(p);
25
26
27
```

<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v202410

100
100

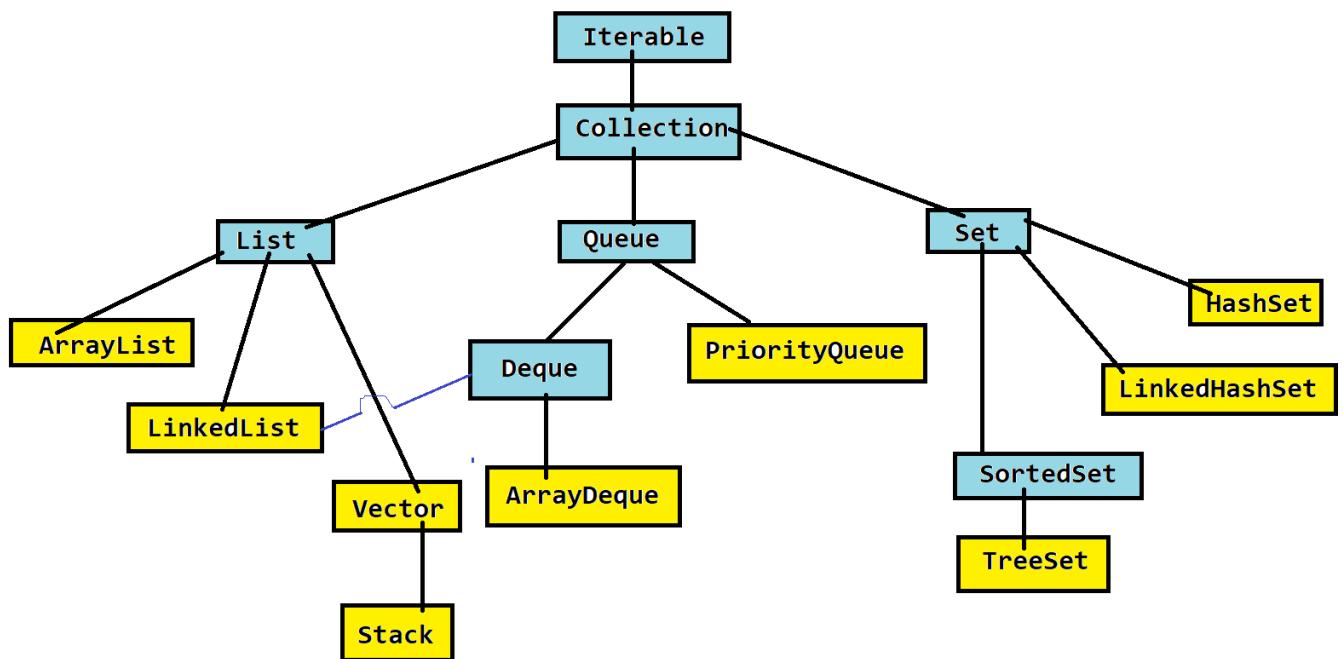
2.AUTO BOXING & AUTO UNBOXING (SINCE JAVA 5)

```
18
19     int k=100;
20     Integer i=k; //AUTO BOXING
21     System.out.println(i);
22     int p=i; //AUTO UNBOXING
23
24
25 }
26 }
```

Console <terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hel

100

Collection Hierarchy



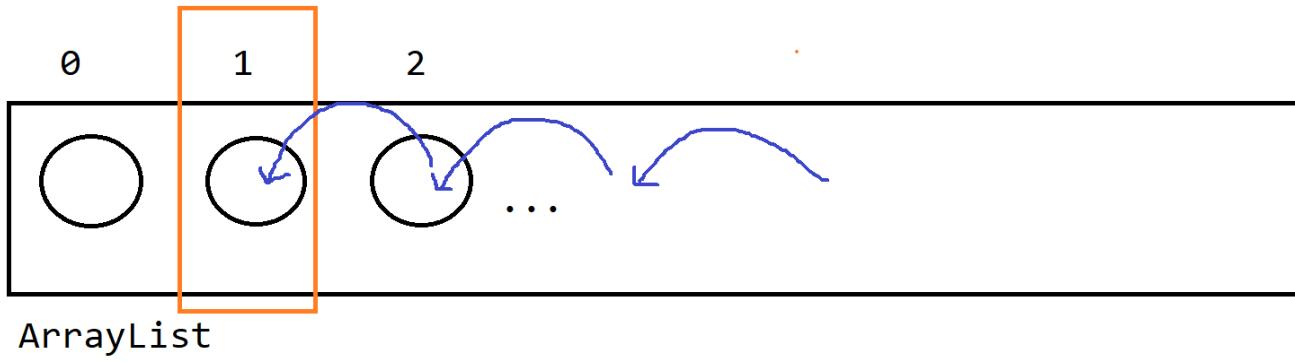
DAY-57

Java Full Stack Development Batch 2025

ArrayList:

->Data Store

- INSERTION ORDER: PRESERVED
- INDEX WISE OPERATION: YES
- DUPLICATES: YES
- RANDOM OPERATION: YES
- NULL : YES
- BEST SUITABLE FOR FAST DATA ACCESS
- DO NOT USE IT FOR DATA MODIFICATION



```

package com.mainapp;
import java.util.ArrayList;
import java.util.Iterator;

public class Launch {

    public static void main(String[] args) {

        //      ArrayList test = new ArrayList();
        //      test.add("test1");
        //      test.add("test2");
        //
        //      ArrayList al = new ArrayList();
        //      al.add(1234);
        //      al.add(1234.56f);
        //      al.add("xyz");
        //      al.add('a');
        //      al.add(false);
        //
        //      System.out.println(al);
        //      al.add(2,8000);
        //      System.out.println(al);
        //
        //      al.addAll(test);
    }
}

```

```
//      System.out.println(al);
//
//      al.addAll(0,test);
//      System.out.println(al);
//      //System.out.println(al.get(2));
//
//      //
=====

====

//      ArrayList al = new ArrayList();
//
//      // ->
//
//      al.add(1234);
//      al.add(1234.56f);
//      al.add("xyz");
//      al.add('a');
//      al.add(false);
//
//      System.out.println(al.get(3));
//
//      for(int i=0; i<al.size() ; i++) {
//          System.out.println(al.get(i));
//      }
//
//      Iterator iterator = al.iterator();
//      while(iterator.hasNext()) {
//          System.out.println(iterator.next());
//      }
//
//      //
=====

//      ArrayList test = new ArrayList();
//      test.add(false);
//      test.add('a');
//      test.add("test2");
//
```

```
//      ArrayList al = new ArrayList();
//
//      al.add(1234);
//      al.add(1234.56f);
//      al.add("xyz");
//      al.add('a');
//      al.add("xyz");
//      al.add(false);
//
//
//      al.remove("xyz");
//      al.remove(0);
//      al.removeAll(test);
//
//      System.out.println(al);
```

```
=====
=====
```

```
//      ArrayList al = new ArrayList();
//
//      al.add(123);
//      al.add(55);
//      al.add(90);
//      al.add(20);
//      al.add(2);
//      al.add(80);
//
//      Integer i=2;
//      al.remove(i);
//      //index->primitive
//      //data->object
//
//      System.out.println(al);
```

```

//=====
==

ArrayList al = new ArrayList();

al.add(null);
al.add(55);
al.add(90);
al.add(null);
al.add(2);
al.add(80);

System.out.println(al.isEmpty());
System.out.println(al.contains(90));
al.set(2, 1000);
}

}

```

LinkedList:

->Data Store

- **INSERTION ORDER: PRESERVED**
- **INDEX WISE OPERATION: YES**
- **DUPLICATES: YES**
- **RANDOM OPERATION: YES**
- **NULL : YES**

DAY-58

Java Full Stack Development Batch 2025

```
java.lang.ClassCastException: class
java.lang.Character cannot be cast to class
java.lang.String
```

Sol: generic collection (Specific Type)

```
//GENERIC COLLECTION
    ArrayList<String> al = new
ArrayList<String>(); //NAMES
    al.add("xyz1");
    al.add("xyz2");
    al.add("xyz3");
    al.add("xyz4");

    for(int i=0 ; i<al.size() ; i++) {
        String names = (String) al.get(i);
        System.out.println(names);
    }
```

NOTE:

```

//NON GENERIC COLLECTION
ArrayList al = new ArrayList(); //NAMES
al.add("xyz1");
al.add("xyz2");
al.add("xyz3");
al.add("xyz4");
al.add('a');

for(int i=0 ; i<al.size() ; i++) {
    Object object = al.get(i);
    if(object instanceof String) {
        String name=(String) object;
        System.out.println(name);
    }else {
        Character c=(Character) object;
        System.out.println(c);
    }
}

```

WILD CARD GENERICS

```

package com.mainapp;
import java.util.List;
public class Launch {

    public static void main(String[] args) {

        List<?> data = new Service().readData();

    }
}

```

```

package com.mainapp;
import java.util.List;
public class Service {

    public List<?> readData() {

        ReadMyData r = new ReadMyData();

```

```

        return r.readData(5);
    }

}

package com.mainapp;
import java.util.Arrays;
import java.util.List;

public class ReadMyData {

    List<String> list1=Arrays.asList("11","12","13");
    List<Integer> list2=Arrays.asList(11,22,33);

    public List<?> readData(int key) {

        if(key==1) {
            return list1;
        }
        else {
            return list2;
        }
    }
}

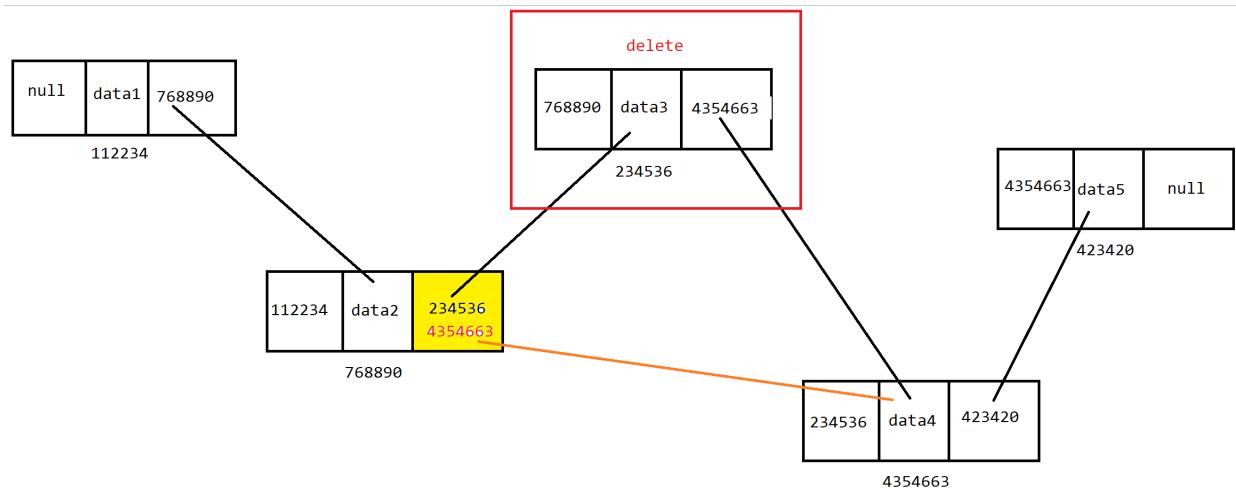
```

LinkedList:

->Data Store

- **INSERTION ORDER: PRESERVED**
- **INDEX WISE OPERATION: YES**
- **DUPLICATES: YES**
- **RANDOM OPERATION: YES**

- **NULL : YES**
- **it support doubly linked list internally**
- **Best suitable for data manipulation**
- **It takes more memory**



```

LinkedList<String> list = new
LinkedList<String>();
    list.add("data1");
    list.add("data2");
    list.add("data3");
    list.add("data4");
    list.add("data5");

System.out.println(list);

list.add(1,"XXXX");

System.out.println(list);

```

Vector:

->Data Store (Same as ArrayList)

- INSERTION ORDER: PRESERVED
- INDEX WISE OPERATION: YES
- DUPLICATES: YES
- RANDOM OPERATION: YES
- NULL : YES
- Synchronized (bari bari->kai log(multiple threads))

```
Vector<String> list = new Vector<String>();
list.add("data1");
list.add("data2");
list.add("data3");
list.add("data4");
list.add("data5");

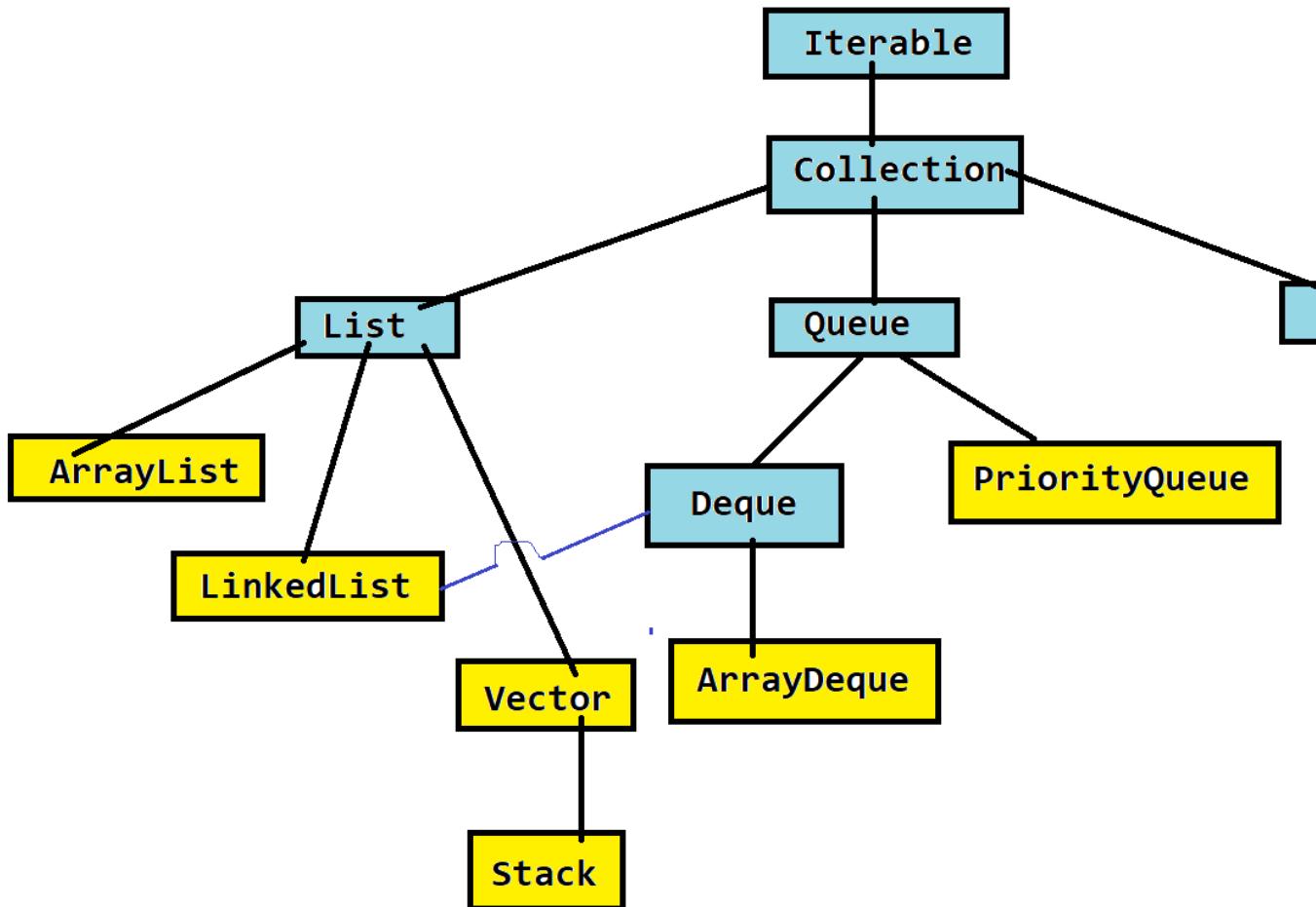
System.out.println(list);

list.add(1, "XXXX");

System.out.println(list);
```

DAY-59

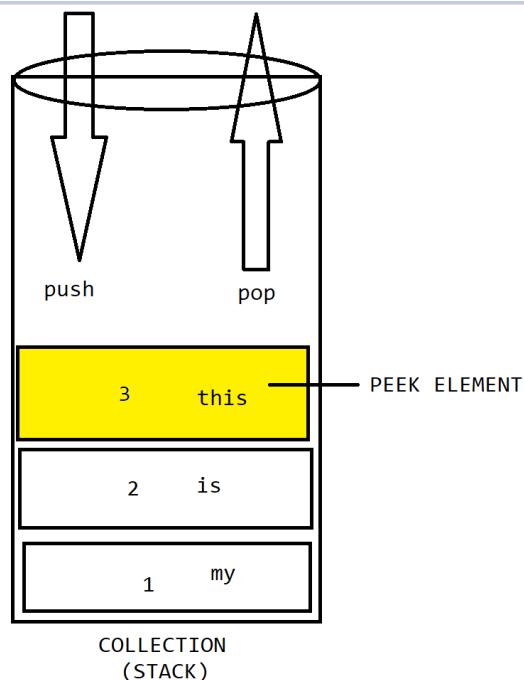
Java Full Stack Development Batch 2025



Stack

- INSERTION ORDER: PRESERVED
- INDEX WISE OPERATION: NO
- DUPLICATES: YES
- NULL : YES
- USE: FILO/LIFO

**Example: Activation Record in Java, Text editors
(delete/undo)**



The screenshot shows a Java code editor with a file named 'Launch.java'. The code creates a stack of strings and performs various operations on it. The console output shows the stack's state after each operation.

```
4
5 public static void main(String[] args) {
6
7     Stack<String> stack = new Stack<String>();
8     stack.push("data1");
9     stack.push("data2");
10    stack.push("data3");
11    stack.push("data4");
12    stack.push("data5");
13
14    System.out.println(stack);
15
16    stack.pop();
17
18    System.out.println(stack);
19
20    System.out.println(stack.peek());
21
22    System.out.println(stack.search("data3"));
23
24    System.out.println(stack);
25
26 }
27 }
```

Console Output:

```
<terminated> Launch (2) [Java Application]
[data1, data2, data3, data4
[data1, data2, data3, data4
2
[data1, data2, data3, data4
```

Queue

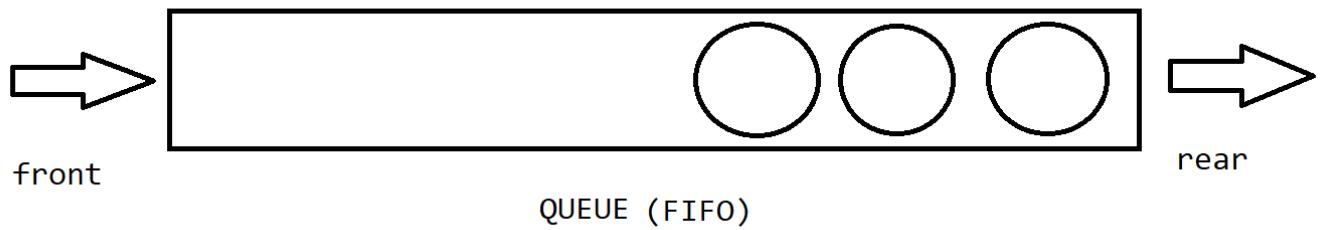
1.Queue

2.Deque

3.PriorityQueue

Queue

- INSERTION ORDER: PRESERVED
- INDEX WISE OPERATION: NO
- DUPLICATES: YES
- NULL : YES
- USE: FIFO/LIFO



Ex. Ticket Booking / Telephonic software

How to implement Queue?

We can use any implementation class of Queue

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays `Launch.java` with the following content:

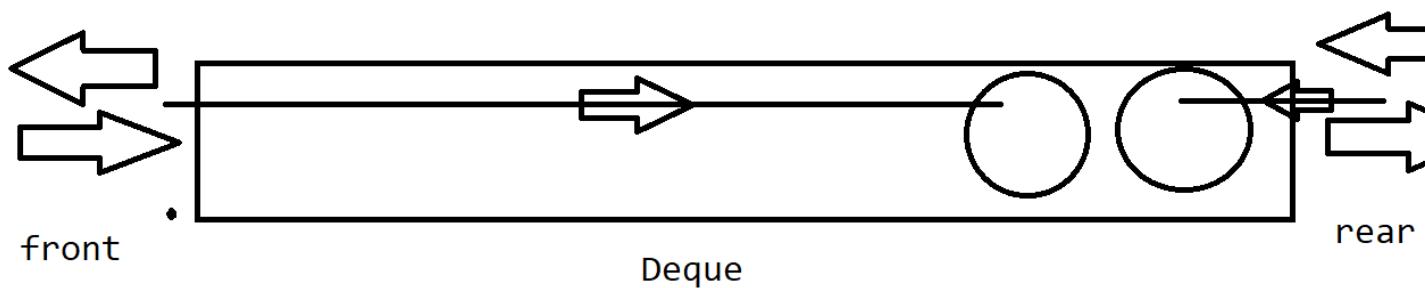
```
public static void main(String[] args) {
    Queue<String> queue = new ArrayDeque<String>();
    queue.add("data1");
    queue.add("data2");
    queue.add("data3");
    queue.add("data4");
    queue.add("data5");

    System.out.println(queue);
    queue.poll();
    System.out.println(queue);
    System.out.println(queue.peek());
    System.out.println(queue);
    System.out.println(queue.element());
}
}
```

The code uses an `ArrayDeque` to demonstrate various methods: `add`, `poll`, `peek`, and `element`. The right side of the interface shows the `Console` tab with the following output:

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\workspace\plus
[data1, data2, data3, data4, data5]
[data2, data3, data4, data5]
data2
[data2, data3, data4, data5]
data2
```

DeQueue



How to implement Deque?

We can use any implementation class of Deque interface

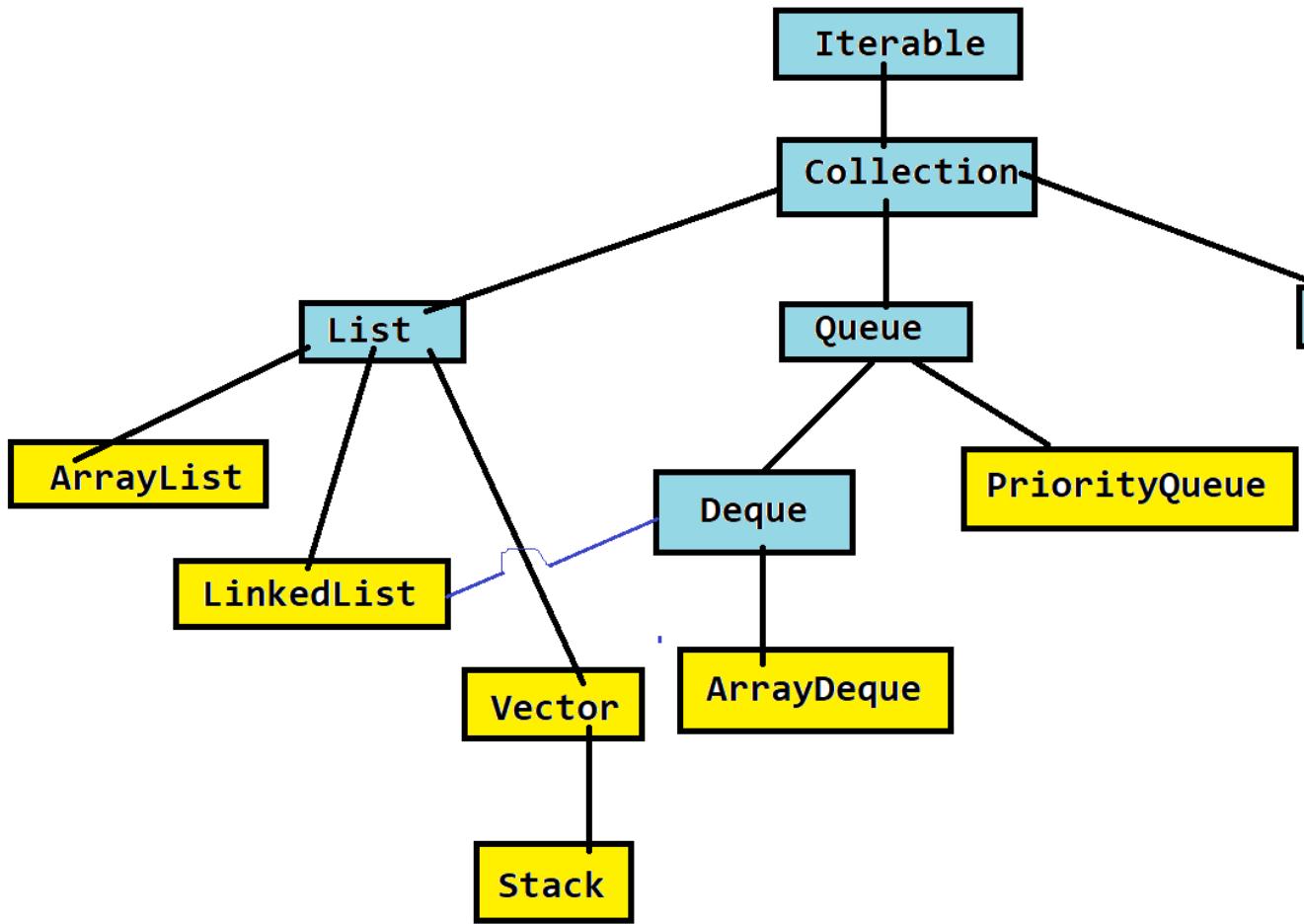
The screenshot shows the Eclipse IDE interface. On the left, the code editor window titled "Launch.java" displays Java code demonstrating the use of ArrayDeque. The code creates a deque, adds four elements ("data1" through "data4"), and then prints the queue, its first element, and its last element. On the right, the "Console" window shows the execution output: the deque's state, followed by the printed values "data1", "data3", and "data3".

```
1 package com.mainapp;
2 import java.util.ArrayDeque;
3 import java.util.Deque;
4 public class Launch {
5
6     public static void main(String[] args) {
7
8         Deque<String> queue = new ArrayDeque<String>();
9         queue.addFirst("data1");
10        queue.addFirst("data2");
11
12        queue.addLast("data3");
13        queue.addLast("data4");
14
15        System.out.println(queue);
16
17        queue.pollFirst();
18
19        System.out.println(queue);
20
21        queue.pollLast();
22        System.out.println(queue);
23
24        System.out.println(queue.peekFirst());
25        System.out.println(queue.peekLast());
26    }
27 }
28
```

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE
[data2, data1, data3, data4]
[data1, data3, data4]
[data1, data3]
data1
data3
```

DAY-60

Java Full Stack Development Batch 2025



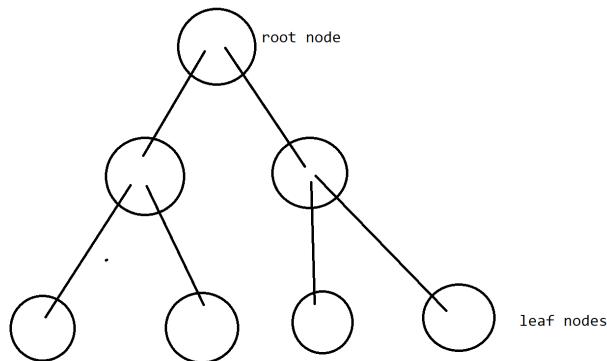
PriorityQueue

Requirement: ALWAYS LOWERST/HEIGHT ELEMENT

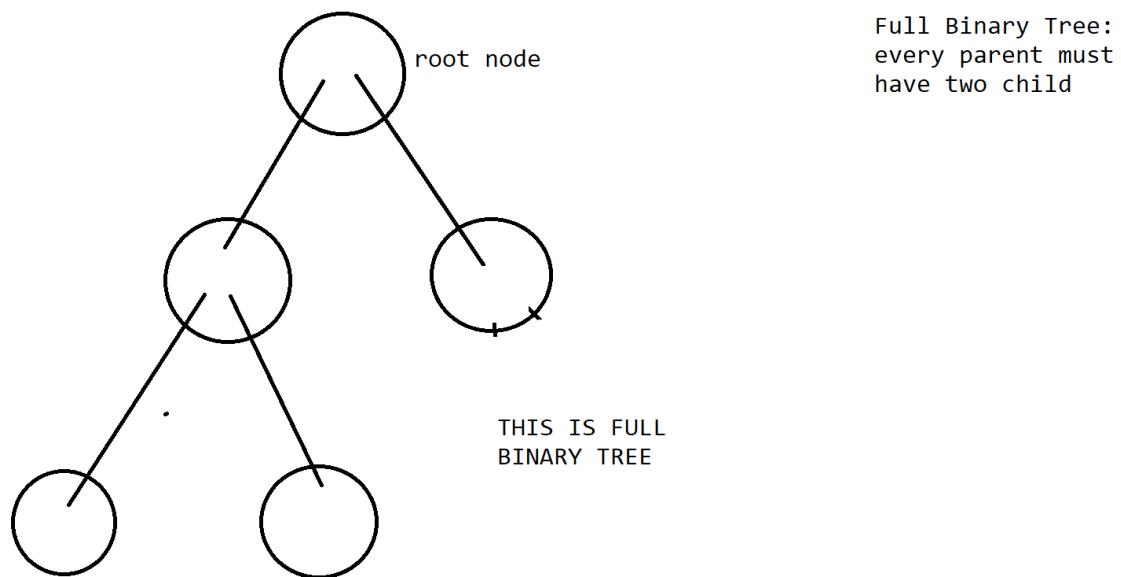
- **INSERTION ORDER: BASED ON PRIORITY (LOWEST => TOP) : DEAFULT ORDER: ASCENDING**
- **INDEX SUPPORTED: NO**
- **DUPLICATES : ALLOWED**
- **NULL: NOT ALLOWED**

- Priority Queue uses **complete binary tree (MIN HEAP)** to store its element

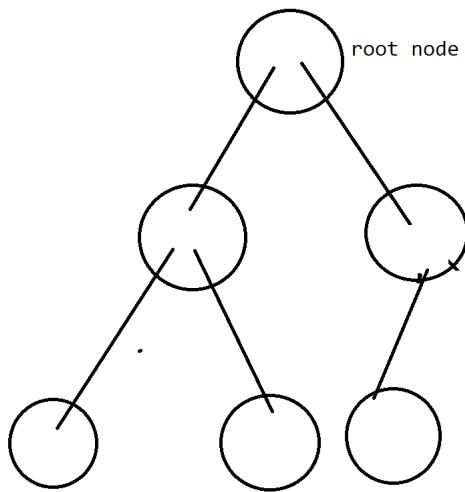
Binary tree:



Full Binary Tree:



Complete Binary Tree:



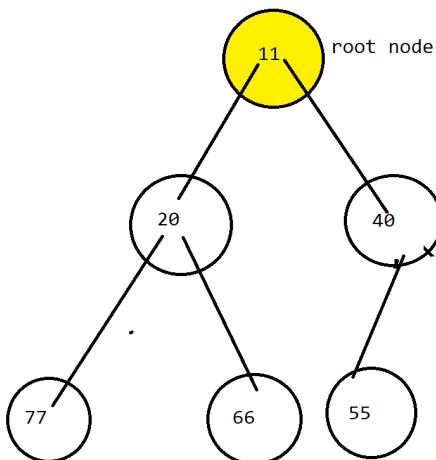
Full Binary Tree:
every parent must
have two child

Complete binary tree:
from root to last leaf there must
not be any in between empty node

This is not a full
binary tree

MIN HEAP : PARENT < CHILD

MAX HEAP: PARENT > CHILD



```
PriorityQueue<Integer> pq = new  
PriorityQueue<Integer>(); //MIN HEAP ( LOWEST ON TOP  
)
```

```
PriorityQueue<Integer> pq = new  
PriorityQueue<Integer>(Collections.reverseOrder());  
(HIGHEST ON TOP )
```

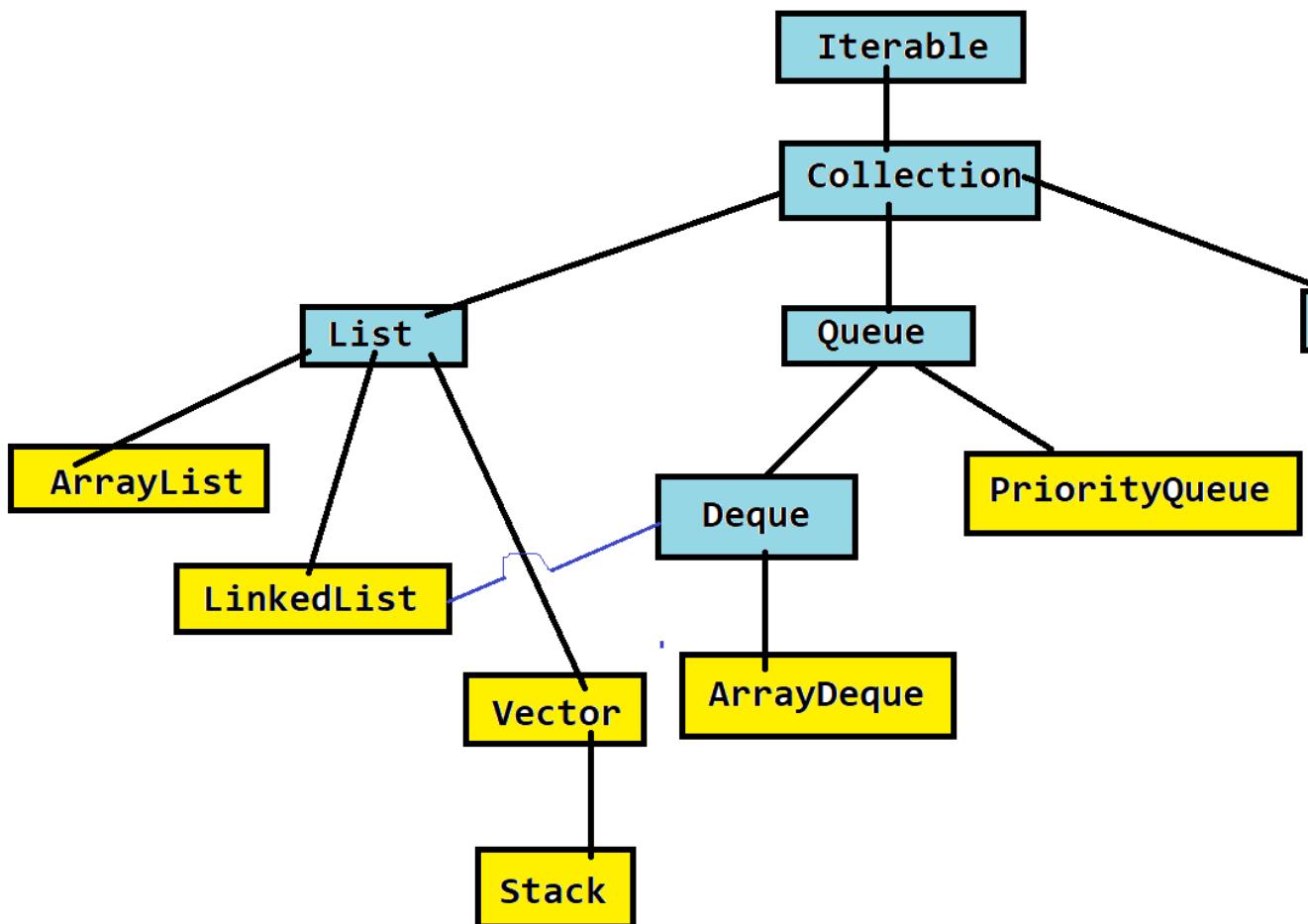
```
package com.mainapp;  
import java.util.Collections;  
import java.util.PriorityQueue;  
public class Launch {  
  
    public static void main(String[] args) {  
  
        PriorityQueue<Integer> pq = new  
PriorityQueue<Integer>(Collections.reverseOrder()); //LOWERST FIRST  
        pq.add(123);  
        pq.add(900);  
        pq.add(766);  
        pq.add(5);  
        pq.add(500);  
        pq.add(99);  
        pq.add(345);  
        pq.add(12);  
        pq.add(1);  
        pq.add(1000);  
  
        System.out.println(pq);  
  
        pq.poll();  
  
        System.out.println(pq);  
  
        System.out.println(pq.peek());  
    }  
}
```

NOTE:-

- **PriorityQueue is designed to be Generic (COZ IT COMPARES DATA)**
- **PriorityQueue supports only object which is of type Comparable**

DAY-61

Java Full Stack Development Batch 2025



HashSet

- When we have unique data
- Insertion order: RANDOM ORDER (HASH FUNCTION)
- INDEX BASED OPERATION: NO
- DUPLICATES: NOT ALLOWED
- NULL: ALLOWED

LinkedHashSet

- When we have unique data
- Insertion order: PRESERVED
- INDEX BASED OPERATION: NO
- DUPLICATES: NOT ALLOWED
- NULL: ALLOWED
- Extra operation: to maintain order

TreeSet

- When we have unique data
- Insertion order: ASCENDING (DEFAULT ORDER)
- INDEX BASED OPERATION: NO
- DUPLICATES: NOT ALLOWED
- NULL: NOT ALLOWED (because one data is comparing with another)
- Extra operation: to maintain order

- DESIGNED TO BE GENERIC (OBJECT should be COMPARABLE)

TASK: WAP to remove duplicates

String s="aaaabbbtttttttrrrrrrrdddddkkkk";

Hint: use HashSet

HashSet:

The screenshot shows the Eclipse IDE interface with two tabs open: "Launch.java" and "Console".

Launch.java:

```
4 public class Launch {  
5     public static void main(String[] args) {  
6         HashSet<String> hs = new HashSet<String>();  
7         hs.add("data1");  
8         hs.add("data2");  
9         hs.add("data2");  
10        hs.add("data2");  
11        hs.add("dataxyz");  
12        hs.add("pqrs");  
13        hs.add("11111");  
14  
15        System.out.println(hs);  
16        hs.remove("pqrs");  
17        System.out.println(hs);  
18  
19        Iterator<String> iterator = hs.iterator();  
20        while(iterator.hasNext()) {  
21            System.out.println(iterator.next());  
22        }  
23    }  
24}
```

Console:

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64  
[pqrs, data2, data1, 11111, dataxyz]  
[data2, data1, 11111, dataxyz]  
data2  
data1  
11111  
dataxyz
```

LinkedHashSet:

The screenshot shows a Java development environment with two windows. On the left is the code editor for `Launch.java`, and on the right is the console window.

Launch.java:

```
1 package com.mainapp;
2 import java.util.ArrayList;
3 import java.util.Iterator;
4 import java.util.LinkedHashSet;
5 public class Launch {
6     public static void main(String[] args) {
7
8         ArrayList<String> arrayList = new ArrayList<String>();
9         arrayList.add("a11");
10        arrayList.add("a11");
11        arrayList.add("a12");
12        arrayList.add("a13");
13        arrayList.add("a13");
14        arrayList.add("a14");
15        arrayList.add("a14");
16
17        LinkedHashSet<String> hs = new LinkedHashSet<String>();
18        hs.add("data1");
19        hs.add("data2");
20        hs.add("data2");
21        hs.add("data2");
22        hs.add("dataxyz");
23        hs.add("pqrs");
24        hs.add("11111");
25
26        hs.addAll(arrayList);
27
28        System.out.println(hs);
29
30        Iterator<String> iterator = hs.iterator();
31        while(iterator.hasNext()) {
32            System.out.println(iterator.next());
33        }
34    }
35
36}
```

Console Output:

```
<terminated> Launch (2) [Java App]
[data1, data2, dataxyz, pqrs,
data1
data2
dataxyz
pqrs
11111
a11
a12
a13
a14]
```

TreeSet

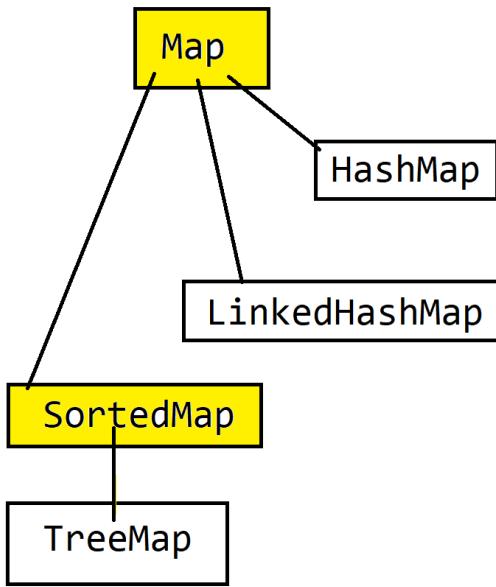
The screenshot shows the Eclipse IDE interface. The top tab bar has 'Launch.java ×' selected. Below it is the code editor window containing the following Java code:

```
1 package com.mainapp;
2 import java.util.Collections;
3 import java.util.TreeSet;
4 public class Launch {
5     public static void main(String[] args) {
6
7         TreeSet<Integer> ts = new TreeSet<Integer>(Collections.reverseOrder());
8         ts.add(1323);
9         ts.add(33);
10        ts.add(67);
11        ts.add(98);
12        ts.add(1);
13        ts.add(9);|
14        ts.add(5000);
15        ts.add(50009);
16        System.out.println(ts);
17    }
18 }
19
```

The cursor is at line 13, position 10, after the 'ts.add(9);' line. The bottom part of the interface shows the 'Console' tab is active, displaying the output of the program:

```
<terminated> Launch (2) [Java Application] D:\ECLIPSE IDE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.5.v20241023-1957\jre\bin\java
[50009, 5000, 1323, 98, 67, 33, 9, 1]
```

MAP INTERFACE (Key-value pair)



HashMap

- When we have data in form of key-value pair
- Insertion order: RANDOM ORDER (HASH FUNCTION)
- INDEX BASED OPERATION: NO
- DUPLICATES: KEY SHOULD BE UNIQUE BUT VALUE CAN BE DUPLICATE
- Null : Multiple null is allowed for value and single null is allowed for key

LinkedHashMap

- When we have data in form of key-value pair
- Insertion order: PRESERVED
- INDEX BASED OPERATION: NO

- DUPLICATES: KEY SHOULD BE UNIQUE BUT VALUE CAN BE DUPLICATE
- Null : Multiple null is allowed for value and single null is allowed for key

TreeMap

- When we have data in form of key-value pair
- Insertion order: ASCENDING (DEFAULT ORDER) : BASED ON KEY
- INDEX BASED OPERATION: NO
- DUPLICATES: KEY SHOULD BE UNIQUE BUT VALUE CAN BE DUPLICATE
- NULL IS NOT ALLOWED AS A KEY

HashMap:

```
Launch.java ×
1 package com.mainapp;
2 import java.util.HashMap;
3 import java.util.Iterator;
4 import java.util.Map.Entry;
5 import java.util.Set;
6 public class Launch {
7     public static void main(String[] args) {
8
9         HashMap<Integer, String> hashMap = new HashMap<Integer, String>();
10        hashMap.put(110, "data2"); //Entry<Integer, String>
11        hashMap.put(120, "data3");
12        hashMap.put(13, "data3");
13        hashMap.put(14, "data4");
14
15
16        Set<Entry<Integer, String>> entrySet = hashMap.entrySet();
17        Iterator<Entry<Integer, String>> iterator = entrySet.iterator();
18
19        while(iterator.hasNext()) {
20
21            Entry<Integer, String> entry = iterator.next();
22            System.out.println(entry.getKey()+"->" +entry.getValue());
23
24        }
25
26 //        System.out.println(hashMap);
27 //        hashMap.remove(110);
28 //
29 //        boolean containsValue = hashMap.containsKey(1200);
30 //        System.out.println(containsValue);
31    }
32}
```

DAY-62

Java Full Stack Development

Batch 2025

LinkedHashMap

```
package com.mainapp;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map.Entry;
import java.util.Set;
public class Launch {
    public static void main(String[] args) {

        LinkedHashMap<Integer, String> hashMap = new
LinkedHashMap<Integer, String>();
        hashMap.put(110, "data2"); //Entry<Integer, String>
        hashMap.put(120, "data3");
        hashMap.put(13, "data3");
        hashMap.put(14, "data4");

        Set<Entry<Integer, String>> entrySet = hashMap.entrySet();
        Iterator<Entry<Integer, String>> iterator =
entrySet.iterator();

        while(iterator.hasNext()) {

            Entry<Integer, String> entry = iterator.next();
            System.out.println(entry.getKey()+"->"+entry.getValue());
        }
    }
}
```

TreeMap

```
package com.mainapp;
import java.util.Collections;
import java.util.Iterator;
```

```

import java.util.LinkedHashMap;
import java.util.Map.Entry;
import java.util.Set;
import java.util.TreeMap;
public class Launch {
    public static void main(String[] args) {

        TreeMap<Integer, String> hashMap = new TreeMap<Integer,
String>(Collections.reverseOrder());
        hashMap.put(110, "data2"); //Entry<Integer, String>
        hashMap.put(120, "data3");
        hashMap.put(13, "data3");
        hashMap.put(14, "data4");
        hashMap.put(15, null);
        hashMap.put(16, null);

        Set<Entry<Integer, String>> entrySet = hashMap.entrySet();
        Iterator<Entry<Integer, String>> iterator =
entrySet.iterator();

        while(iterator.hasNext()) {

            Entry<Integer, String> entry = iterator.next();

            System.out.println(entry.getKey()+"->"+entry.getValue());
        }
    }
}

```

- **ArrayList**
- **LinkedList**
- **Vector**
- **Stack**
- **PriorityQueue**
- **ArrayDeque (Queue/Deque)**
- **HashSet**
- **LinkedHashSet**

- TreeSet
- HashMap
- LinkedHashMap
- TreeMap

Comparable vs Comparator

- Both are interfaces in Java
- Comparable interface (compareTo) :- for default natural sorting order
- Comparator interface:- for custom order
- **POSITIVE: right**
- **NEGATIVE: left**
- **ZERO: order in which u are inserting**

```
package com.mainapp;
import java.util.TreeSet;
public class Launch {
    public static void main(String[] args) {

        Employee e1 = new Employee(11, "name1", 40);
        Employee e2 = new Employee(12, "name2", 20);
        Employee e3 = new Employee(13, "name3", 19);
        Employee e4 = new Employee(14, "name4", 18);

        TreeSet<Employee> ts = new TreeSet<Employee>(new
        DescOrderBasedOnAge());
        ts.add(e2); // [e2 ]
        ts.add(e1); // [e2 e1 ]
        ts.add(e3); // [e3 e2 e1]
        ts.add(e4); // [e4 e3 e2 e1]
```

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

package com.mainapp;

//ACS : BASED ON ID
public class Employee implements Comparable<Employee> {

    private int id;
    private String name;
    private int age;

    public Employee(int id, String name, int age) {
        super();
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", age=" +
age + "]";
    }
}
```

```

//DEFAULT NATURAL SORTING ORDER
@Override
public int compareTo(Employee o) { //PRE EXISTING
    return this.id-o.id;
}
}

package com.mainapp;

import java.util.Comparator;

//CUSTOM ORDER:
public class DescOrderBasedOnAge implements Comparator<Employee>{

    @Override           //CURRENT OBJ           EXISTING OBJ
    public int compare(Employee current,      Employee existing) {

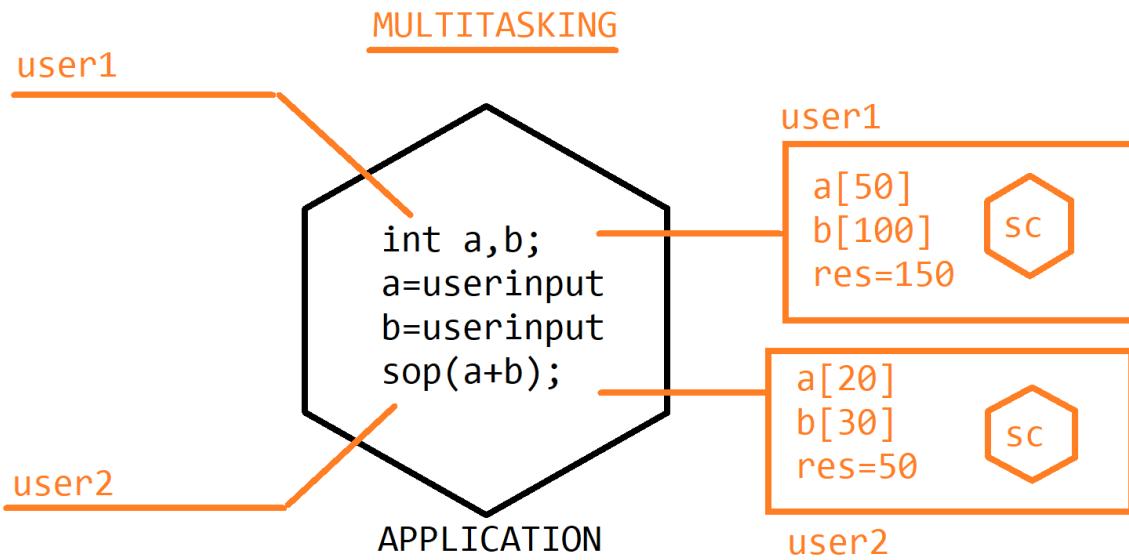
        return current.getAge()-existing.getAge();
        // 18 - 20 = neg // 18-19 = NEG
    }
}

```

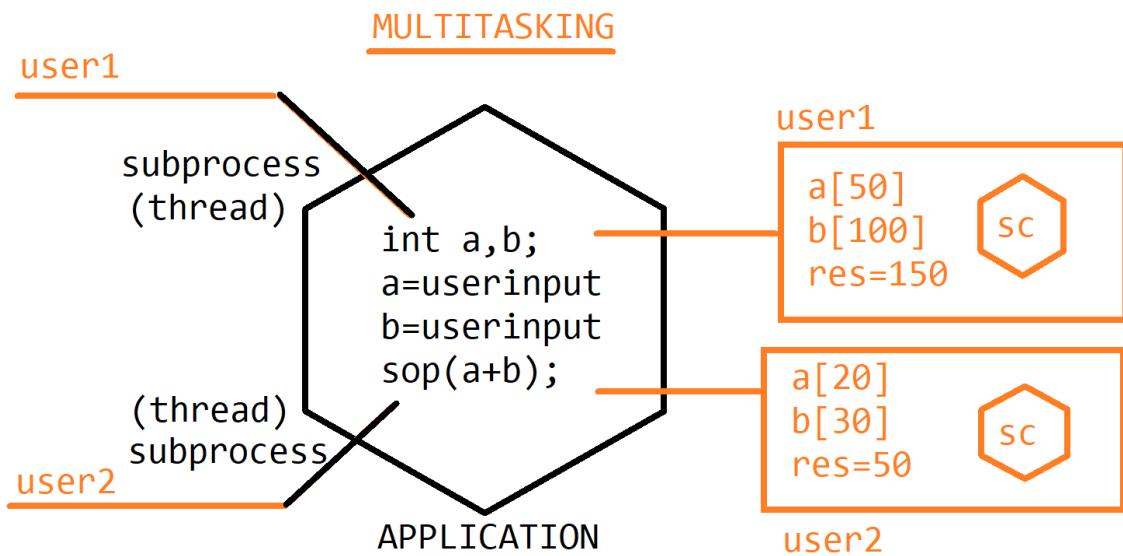
DAY-63

Java Full Stack Development Batch 2025

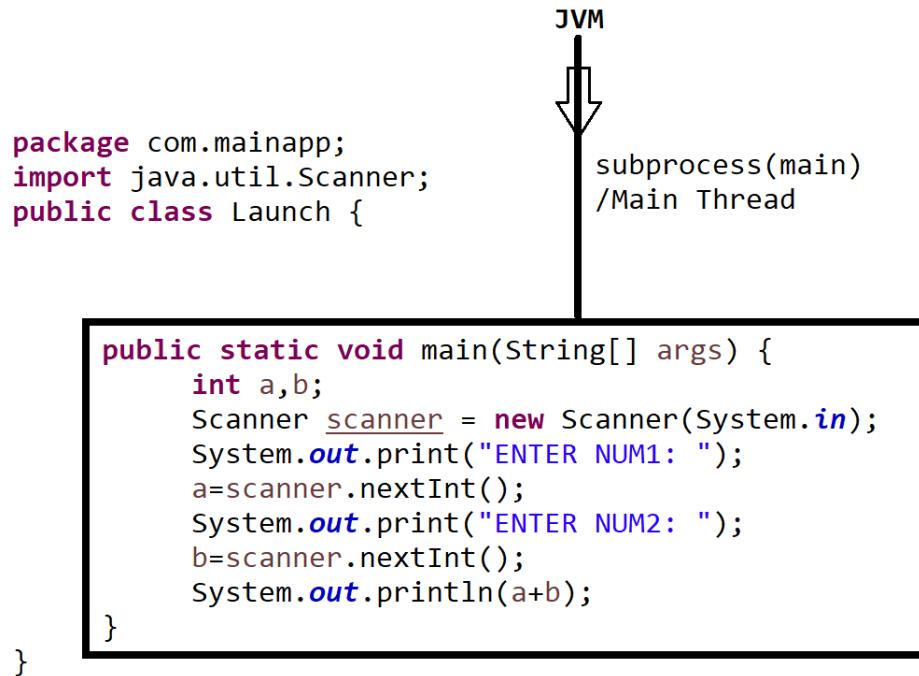
Multithreading



- Multithreading is a technique to perform multitasking
- Thread is simply a light weight process
- Multithreading is the process of executing multiple threads simultaneously



SINGLE THREADED APPLICATION



NOTE: Through multithreading we can utilize CPU time efficiently

HOW CAN WE ACHIEVE MULTITHREADING IN JAVA?

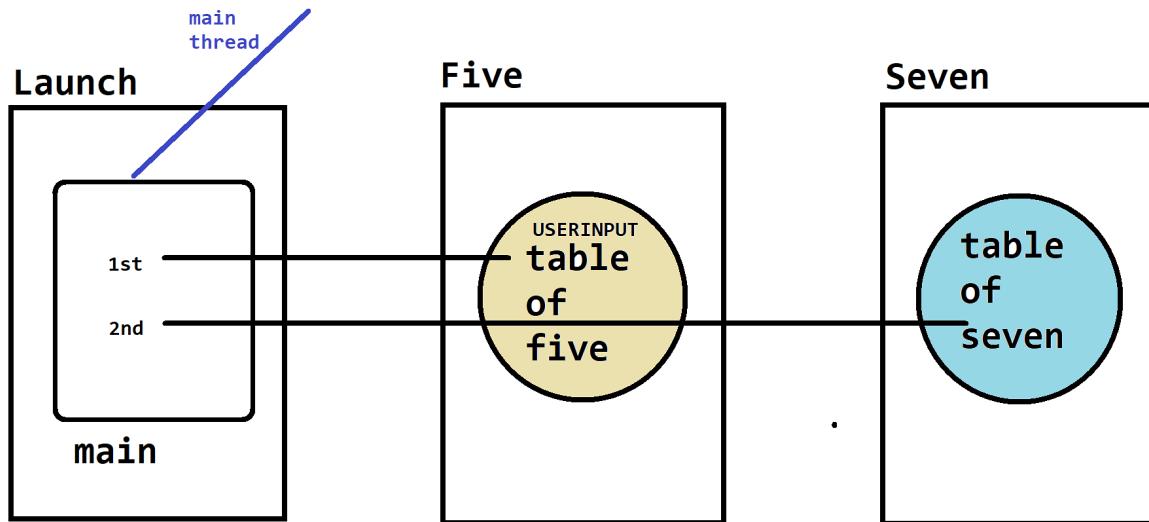
IN JAVA WE CAN ACHIEVE MULTITHREADING IN TWO WAYS

1. Thread class

2. Runnable Interface

Thread class

1. MULTIPLE CLASSES WITH ONE-ONE THREAD (Thread class)



THREAD SCHEDULER WILL DECIDE WHICH THREAD WILL EXECUTE FIRST

- 1.first come first serve
- 2.high priority

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        Five five = new Five();
        //THREAD CREATED

        Seven seven = new Seven();
        //THREAD CREATED

        five.start();
        seven.start();
    }
}
```

```
        }
    }

package com.mainapp;

public class Five extends Thread{

    @Override
    public void run() {

        for(int i=1;i<=10;i++) {
            System.out.println(5*i);
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

package com.mainapp;

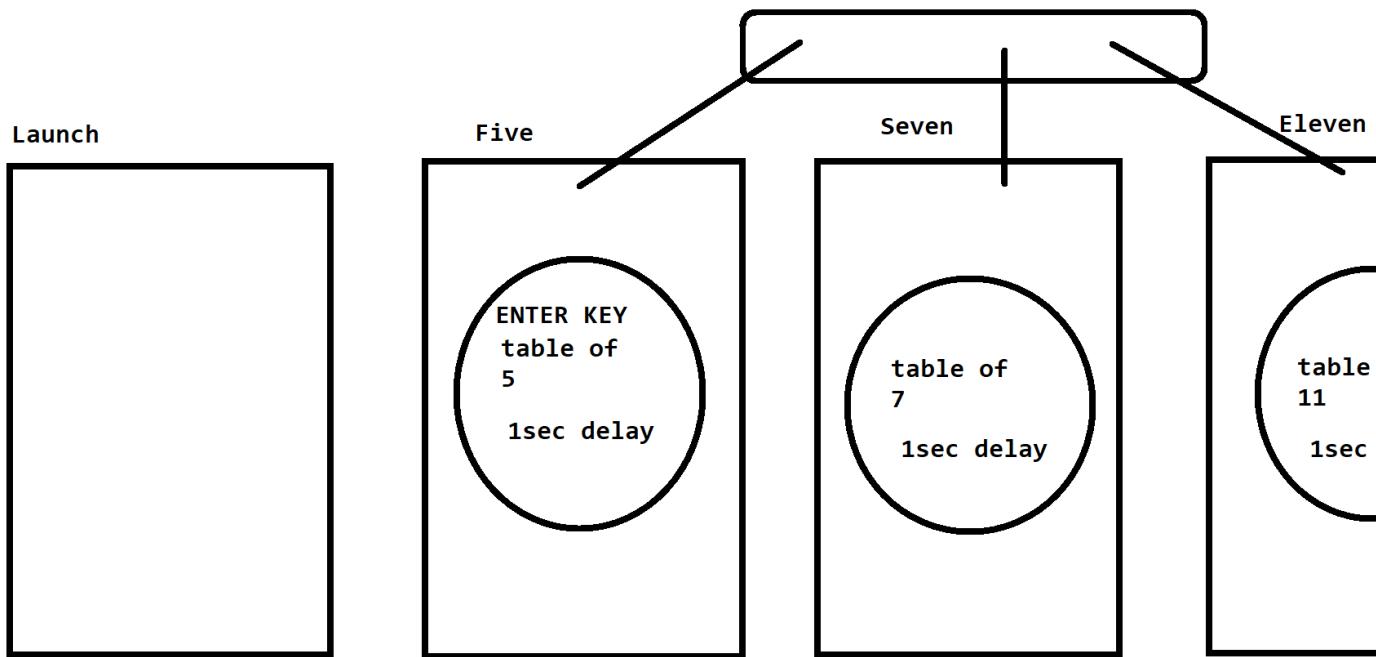
public class Seven extends Thread {

    @Override
    public void run() {

        for(int i=1;i<=10;i++) {
            System.out.println(7*i);
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

}

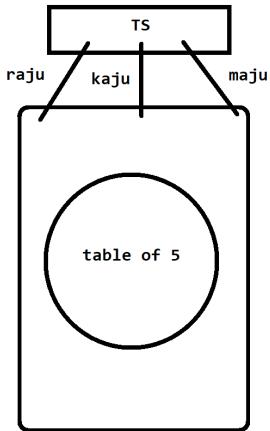
TASK:



DAY-64

Java Full Stack Development
Batch 2025

2. ONE CLASS MULTIPLE THREAD (Thread class)



```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {
```

```
        Five five1 = new Five(); //THREAD
        Five five2 = new Five(); //THREAD
        Five five3 = new Five(); //THREAD
```

```
        five1.start();
        five2.start();
        five3.start();
    }
}
```

```
package com.mainapp;

public class Five extends Thread{
```

```
    @Override
    public void run() {
```

```
        xyz();
    }
```

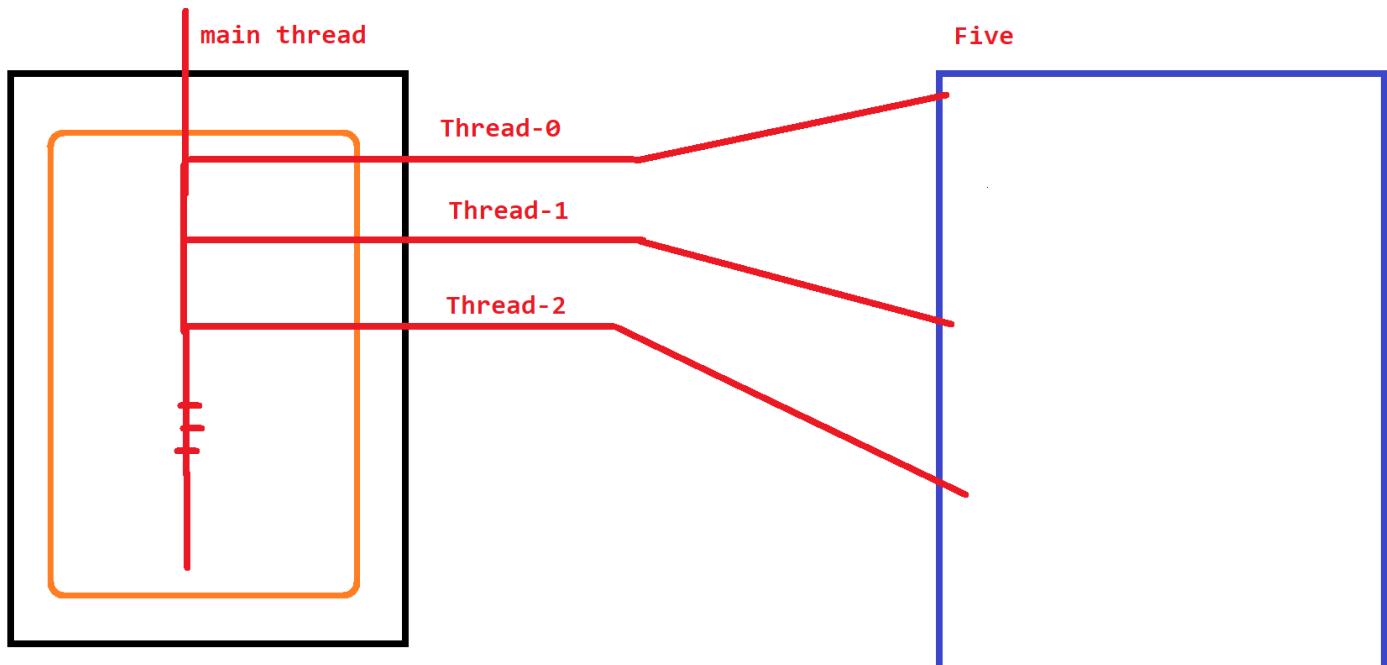
```
    private void xyz() {
```

```

        for(int i=1;i<=10;i++) {
            System.out.println(5*i);
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

THREAD NAMING



```

package com.mainapp;
public class Launch {
    public static void main(String[] args) {

```

```

        Five five1 = new Five(); //THREAD
        Five five2 = new Five(); //THREAD
        Five five3 = new Five(); //THREAD

        five1.setName("raju");
        five2.setName("kaju");
        five3.setName("maju");

        five1.start();
        five2.start();
        five3.start();

        System.out.println("HELLO");
    }
}

package com.mainapp;

public class Five extends Thread{

    @Override
    public void run() {

        Thread thread = currentThread();
        String name = thread.getName();

        xyz(name);

    }

    private void xyz(String name) {

        for(int i=1;i<=10;i++) {
            System.out.println(name+":"+ (5*i));
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

Thread Priority

1(lowest) - 10(highest)

Main thread : 5

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) {

        System.out.println(Thread.currentThread().getName());

        Five five1 = new Five(); //THREAD
        Five five2 = new Five(); //THREAD
        Five five3 = new Five(); //THREAD

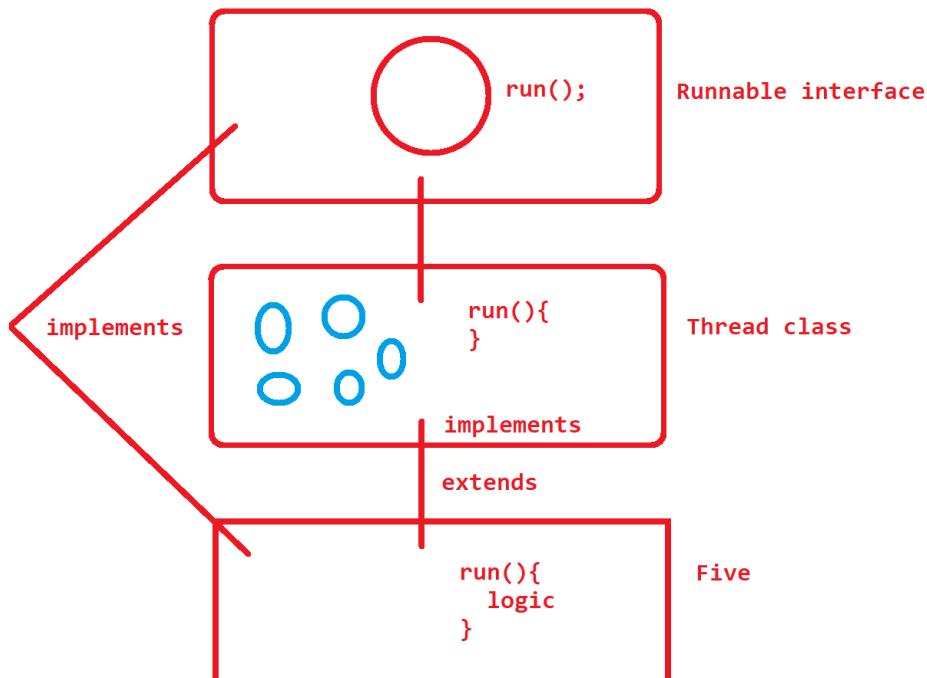
        five1.setName("raju");
        five2.setName("kaju");
        five3.setName("maju");

        five3.setPriority(10);
        five2.setPriority(5);
        five1.setPriority(1);

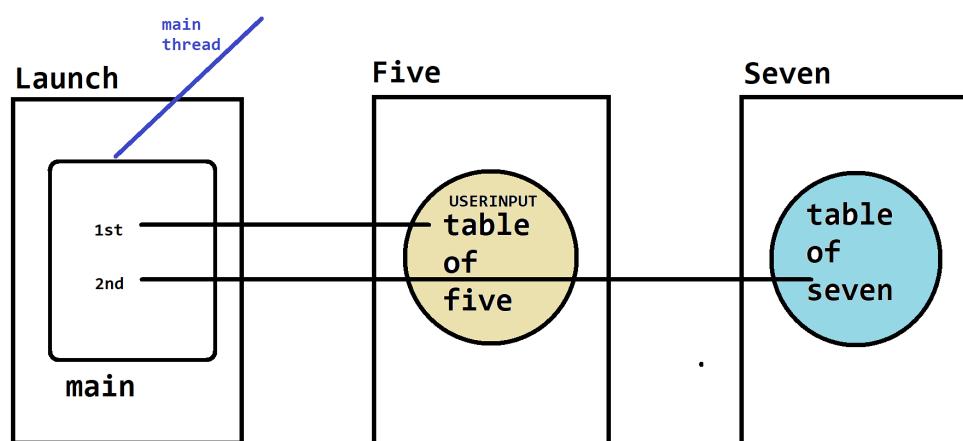
        five1.start();
        five2.start();
        five3.start();

        System.out.println("HELLO");
    }
}
```

Runnable Interface



1. MULTIPLE CLASSES WITH ONE-ONE THREAD (Thread class)



Join method()

```
package com.mainapp;
```

```

public class Launch {
    public static void main(String[] args) throws
InterruptedException {

    System.out.println("START");

    Five five1 = new Five(); //Runnable target
    Thread thread1 = new Thread(five1); //Thread Created

    Seven seven = new Seven(); //Runnable target
    Thread thread2 = new Thread(seven); //Thread Created

    thread1.setName("raju");
    thread2.setName("kaju");

    thread2.setPriority(Thread.MAX_PRIORITY);
    thread1.setPriority(Thread.MIN_PRIORITY);

    thread1.start();
    thread2.start();

    thread1.join();
    thread2.join();

    System.out.println("EXIT");
}
}

```

```

package com.mainapp;

public class Five implements Runnable{

    @Override
    public void run() {

        String name = Thread.currentThread().getName();

        for(int i=1;i<=10;i++) {
            System.out.println(name+":"+ (5*i));

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
            }
        }
    }
}

```

```
        e.printStackTrace();
    }
}

}

package com.mainapp;

public class Seven extends Parent implements Runnable{

    @Override
    public void run() {

        test();
    }
}

package com.mainapp;

public class Parent {

    public void test() {
        String name = Thread.currentThread().getName();

        for(int i=1;i<=10;i++) {
            System.out.println(name+":"+ (7*i));

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

DAY-65

Java Full Stack Development Batch 2025

1. MULTIPLE CLASSES WITH ONE-ONE THREAD (Thread class)

```
package com.mainapp;
public class Launch {
    public static void main(String[] args)  {

        Table table = new Table();

        Thread t1 = new Thread(table);
        Thread t2 = new Thread(table);
        Thread t3 = new Thread(table);

        t1.setName("raju");
        t2.setName("kaju");
        t3.setName("maju");

        t1.start();
        t2.start();
        t3.start();
    }
}
package com.mainapp;

public class Parent {

    public void test(int num) {
        String name = Thread.currentThread().getName();

        for(int i=1;i<=10;i++) {
            System.out.println(name+":"+ (num*i));

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {

```

```

        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}
package com.mainapp;

public class Table extends Parent implements Runnable {

    @Override
    public void run() {

        String name = Thread.currentThread().getName();

        if(name.equals("raju")) {
            five();
        }
        else if(name.equals("kaju")) {
            seven();
        }
        else {
            eleven();
        }
    }

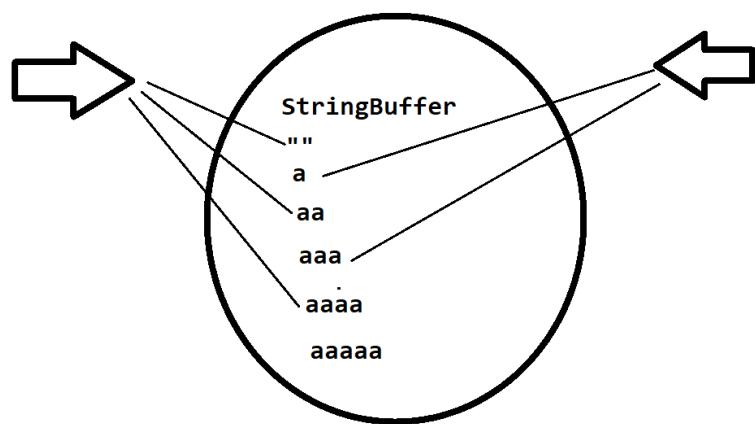
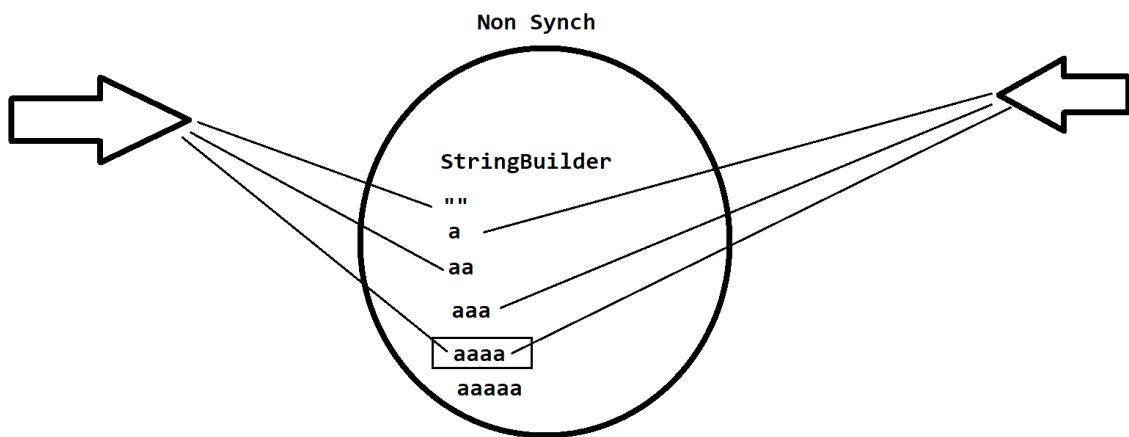
    private void five() {
        super.test(5);
    }

    private void seven() {
        super.test(7);
    }

    private void eleven() {
        super.test(11);
    }
}

```

RACE CONDITION:



In java race condition occurs when more than one thread is trying to change a common data simultaneously and then output of program might be unexpected.

Solution: Synchronization

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) throws InterruptedException {
        Account account = new Account();
        MyThreads table = new MyThreads(account);
```

```

        Thread t1 = new Thread(table);
        Thread t2 = new Thread(table);

        t1.setName("raju");
        t2.setName("kaju");

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println(account.get());

    }
}

package com.mainapp;
public class MyThreads implements Runnable {

    private Account acc;

    public MyThreads(Account acc) {
        this.acc=acc;
    }

    @Override
    public void run() {

        for(int i=1;i<=100000;i++) {
            acc.change();
        }
    }
}

package com.mainapp;

public class Account {

    private StringBuffer name=new StringBuffer("");

    public void change() {
        name.append("a");
    }

    public int get() {
        return name.length();
    }

}

```

Synchronization in Java

We can achieve synchronization in two ways

1. Synchronized method : BROAD SCOPE

2. Synchronized block: CUSTOM SCOPE

Program: Synchronized method

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) throws
InterruptedException {
    Resource account = new Resource();
    MyThreads table = new MyThreads(account);

    Thread t1 = new Thread(table);
    Thread t2 = new Thread(table);

    t1.setName("raju");
    t2.setName("kaju");

    t1.start();
    t2.start();

    t1.join();
    t2.join();
}
}

package com.mainapp;
public class MyThreads implements Runnable {

    private Resource res;

    public MyThreads(Resource res) {
        this.res=res;
    }

    @Override
```

```

public void run() {
    String name = Thread.currentThread().getName();
    if(name.equals("raju")) {
        res.printer(name);
    }
    else {
        res.printer(name);
    }
}
}

package com.mainapp;

public class Resource {

    public synchronized void printer(String name) {

        for(int i=1;i<=10;i++) {
            System.out.println(name+" using printer...\"");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

DAY-66

Java Full Stack Development Batch 2025

Synchronized block: CUSTOM SCOPE

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) throws InterruptedException {

        Test test=new Test();

        MyThread t1 = new MyThread(test);
        MyThread t2 = new MyThread(test);

        t1.setName("raju");
        t1.setName("kaju");

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println(test.getCount());
        System.out.println(test.getCount()-test.getRace());

    }
}

package com.mainapp;

public class MyThread extends Thread {

    private Test test;

    public MyThread(Test test) {
        this.test=test;
    }

    @Override
    public void run() {

        for(int i=1 ; i<=100000 ; i++) {
            test.change();
        }
    }
}

package com.mainapp;
//RESOURCE
public class Test {

    private StringBuffer
    sb=new StringBuffer("");

    private int count=0;
    private int race=0;

    public void change() {
```

```

sb.append("a");

//ENTIRE OBJECT LOCK
synchronized (this) {
    count++; //CRITICAL CODE
}

race++; //race condition
}

public int getCount(){
    return count;
}
public int getRace(){
    return race;
}
}

```

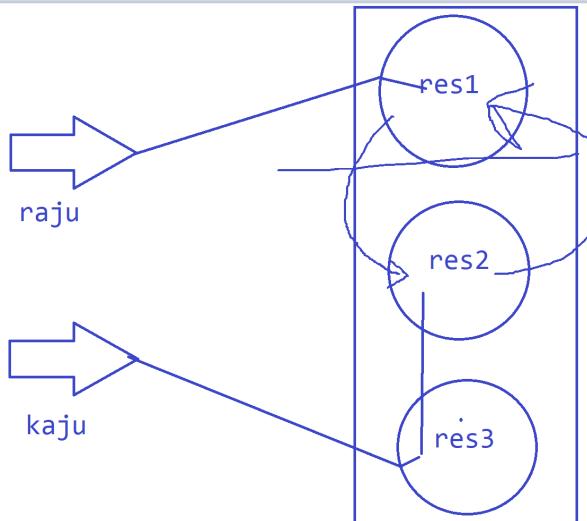
DEAD LOCK CONDITION:

order
 raju->res1
 raju->res2
 raju->res3

 kaju->res1
 kaju->res2
 kaju->res3

 order
 raju->res1
 raju->res2
 raju->res3

 kaju->res3
 kaju->res1
 kaju->res1



```

package com.mainapp;
public class Launch {
    public static void main(String[] args) throws
InterruptedException {

    Test test=new Test();

    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();

    t1.setName("raju");
    t2.setName("kaju");
}

```

```

        t1.start();
        t2.start();
    }
}

package com.mainapp;
public class MyThread extends Thread {

    private String res1="r1";
    private String res2="r2";
    private String res3="r3";

    @Override
    public void run() {

        String name = currentThread().getName();
        if(name.equals("raju")) {
            rajuAccess(name);
        }
        else {
            kajuAccess(name);
        }
    }

    private void rajuAccess(String name) {

        synchronized (res1) {
            System.out.println(name+" using "+res1);

            synchronized (res2) {
                System.out.println(name+" using "+res2);

                synchronized (res3) {
                    System.out.println(name+" using "+res3);
                }
            }
        }
    }

    private void kajuAccess(String name) {

        synchronized (res3) {
            System.out.println(name+" using "+res3);

            synchronized (res2) {

```

```
        System.out.println(name+" using "+res2);

    synchronized (res1) {
        System.out.println(name+" using "+res1);
    }
}
}
```

Object Level Lock & Class Level Lock

Object Level Lock:

```
package com.mainapp;
public class Launch {
    public static void main(String[] args) throws
InterruptedException {
    Test test1=new Test();
    Test test2=new Test();

    MyThread t1 = new MyThread(test1,test2);
    MyThread t2 = new MyThread(test1,test2);

    t1.setName("raju");
    t2.setName("kaju");

    t1.start();
    t2.start();
}
}

package com.mainapp;
public class MyThread extends Thread {

    private Test test1;
    private Test test2;
```

```

public MyThread(Test test1 , Test test2) {
    this.test1=test1;
    this.test2=test2;
}

@Override
public void run() {
    String name = currentThread().getName();
    if(name.equals("raju")) {
        test1.printer(name);
    }
    else {
        test2.printer(name);
    }
}
package com.mainapp;

//RESOURCE
public class Test {

    public void printer(String name) {
        synchronized (this) {
            for (int i = 1; i <= 10; i++) {
                System.out.println(name+" using printer");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}

```

Class level lock:

```

package com.mainapp;
public class Launch {
    public static void main(String[] args) throws
InterruptedException {

```

```

Test test1=new Test();
Test test2=new Test();

MyThread t1 = new MyThread(test1,test2);
MyThread t2 = new MyThread(test1,test2);

t1.setName("raju");
t2.setName("kaju");

t1.start();
t2.start();
}
}

package com.mainapp;
public class MyThread extends Thread {

private Test test1;
private Test test2;

public MyThread(Test test1 , Test test2) {
    this.test1=test1;
    this.test2=test2;
}

@Override
public void run() {
    String name = currentThread().getName();
    if(name.equals("raju")) {
        test1.printer(name);
    }
    else {
        test2.printer(name);
    }
}

package com.mainapp;
//RESOURCE
public class Test {

public void printer(String name) {
synchronized (Test.class) {
    for (int i = 1; i <= 10; i++) {
        System.out.println(name+" using printer");
        try {
            Thread.sleep(500);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
}

```

```
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

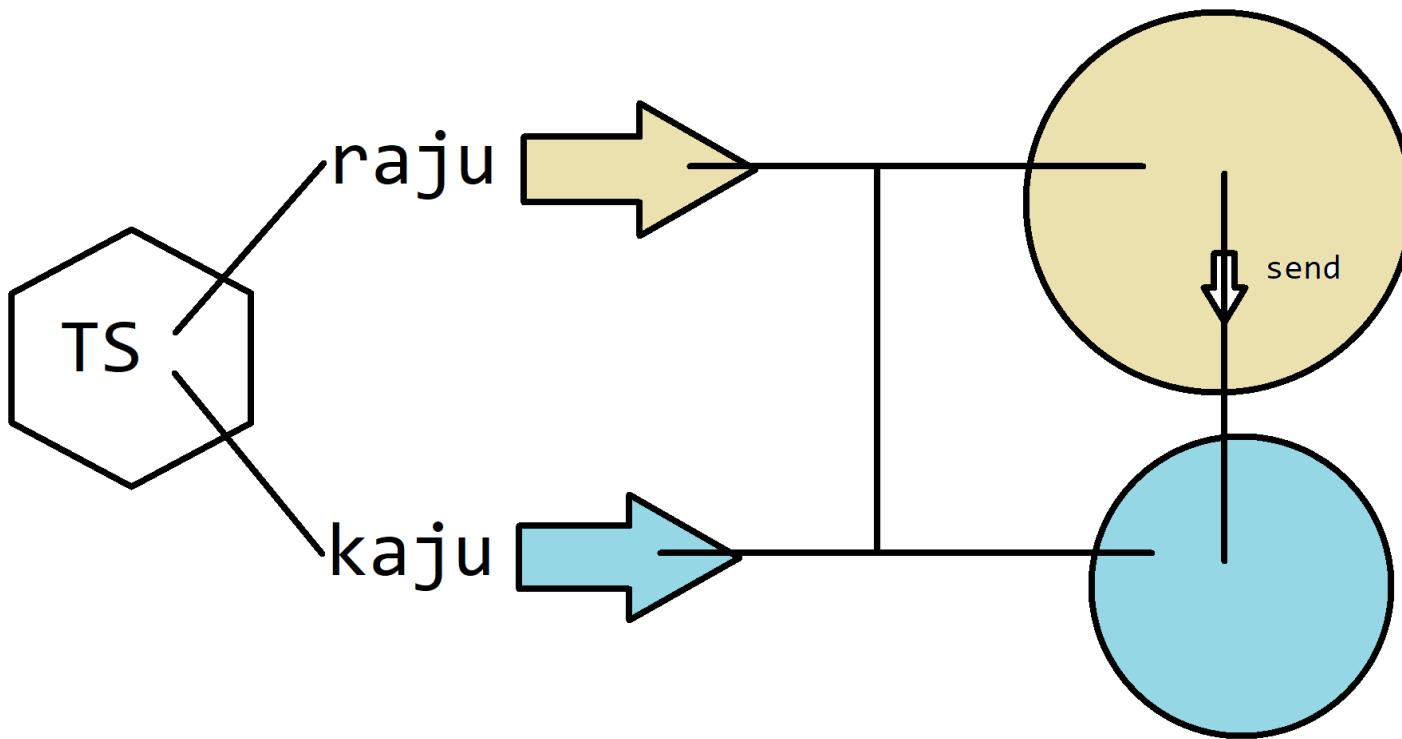
DAY-67

Java Full Stack Development Batch 2025

Inter thread communication

Problem: CONSUMER SENDER PROBLEM

wait() and notify()



```

package com.mainapp;
public class Launch {
    public static void main(String[] args)  {

        Message message=new Message();

        User user1 = new User(message);
        User user2 = new User(message);

        user1.setName("raju");
        user2.setName("kajju");

        user1.start();
        user2.start();

    }
}

package com.mainapp;

public class User extends Thread {

    private Message message;
  
```

```
public User(Message message) {
    this.message=message;
}

@Override
public void run() {

    String name = currentThread().getName();
    if(name.equals("raju")) {

        for(int i=1;i<=5;i++) {
            message.sendMessage("HELLO-"+i);
        }

    }else {
        for(int i=1;i<=5;i++) {
            message.readMessage();
        }
    }
}

package com.mainapp;
public class Message {

    private String message;
    private boolean isSent;

    public synchronized void sendMessage(String message) {

        while(isSent==true) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println(message+" send by RAJU");
    }
}
```

```

    this.message=message;

    isSent=true;
    notify(); //WAKES UP OTHER THREAD ON CURRENT OBJ
}

//KAJU
public synchronized void readMessage() {

    while (isSent==false) {

        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    System.out.println(message+" read by KAJU");
    isSent=false;
    notify();
}
}

```

DAEMON THREAD: SUPPORTING TASK (Ex. Garbage Collector)

THREAD LIFE CYCLE

1.NEW STATE (when we create a thread -> Thread t=new Thread (user))

2.ACTIVE STATE (when u invoke start())

- RUNNING STATE : WHEN YOUR THREAD IS CURRENTLY RUNNING
- RUNNABLE STATE : WHEN YOUR THREAD IS READY TO RUN

3.TIMED WAITED STATE (when u invoke sleep())

4.BLOCK STATE (when a thread is waiting for other thread to release synch resource)

5.WAIT STATE (when u invoke wait() or join())

6.TERMINATED STATE (WHEN YOUR THREAD COMPLETED OR FINISHED IT'S TASK)

- NORMAL TERMINATION : NO EXCEPTION
- ABNORMAL TERMINATION : EXCEPTION

DAY-68

**Java Full Stack Development
Batch 2025**

FILE INPUT OUTPUT (FILE IO)

File IO is used to achieve file persistency
(PERMANENT STORAGE)



OPERATION(ON FILE): READ WRITE DELETE

1. How to create a file

```
package com.mainapp;
import java.io.File;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {
        try {
            for (int i = 1; i <= 1000; i++) {
                File file = new
File("C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\abcd"+i+".txt");
                boolean b = file.createNewFile();
                if (b)
                    System.out.println("FILE CREATED");
                else
                    System.out.println("FAILED TO CREATED");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. How to delete a file

```

package com.mainapp;
import java.io.File;
public class Launch {
    public static void main(String[] args) {

        try {

            for (int i = 1; i <= 1000; i++) {
                File file = new
File("C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\abcd"+i+".txt");
                boolean b = file.delete();
                if (b)
                    System.out.println("FILE DELETED");
                else
                    System.out.println("FAILED TO DELETE");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

WINDOWS:

File->Delete Button Click---->[(FILE COPY PASTE TO RECYCLE BIN) -> DELETE]

3.HOW TO WRITE A FILE

```

package com.mainapp;
import java.io.FileOutputStream;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {
        try {

            String s = "THIS IS MY XYZ";
            byte[] b = s.getBytes();
            FileOutputStream fos = new
FileOutputStream("C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\mydata.txt");
            fos.write(b);
            System.out.println("DATA SENT SUCCESSFULLY");

        } catch (IOException e) {

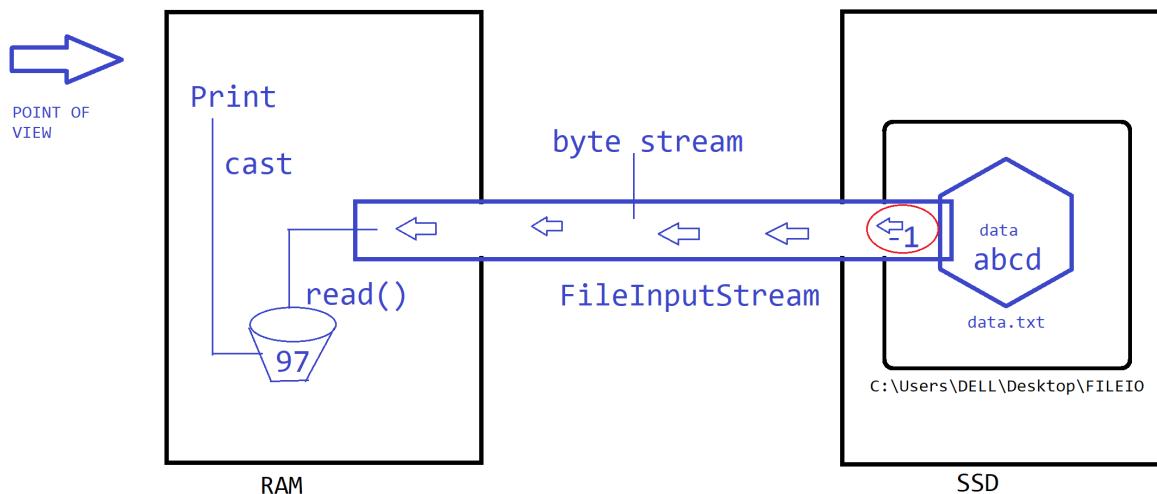
```

```

        e.printStackTrace();
    }
}
}

```

4. HOW TO READ A FILE



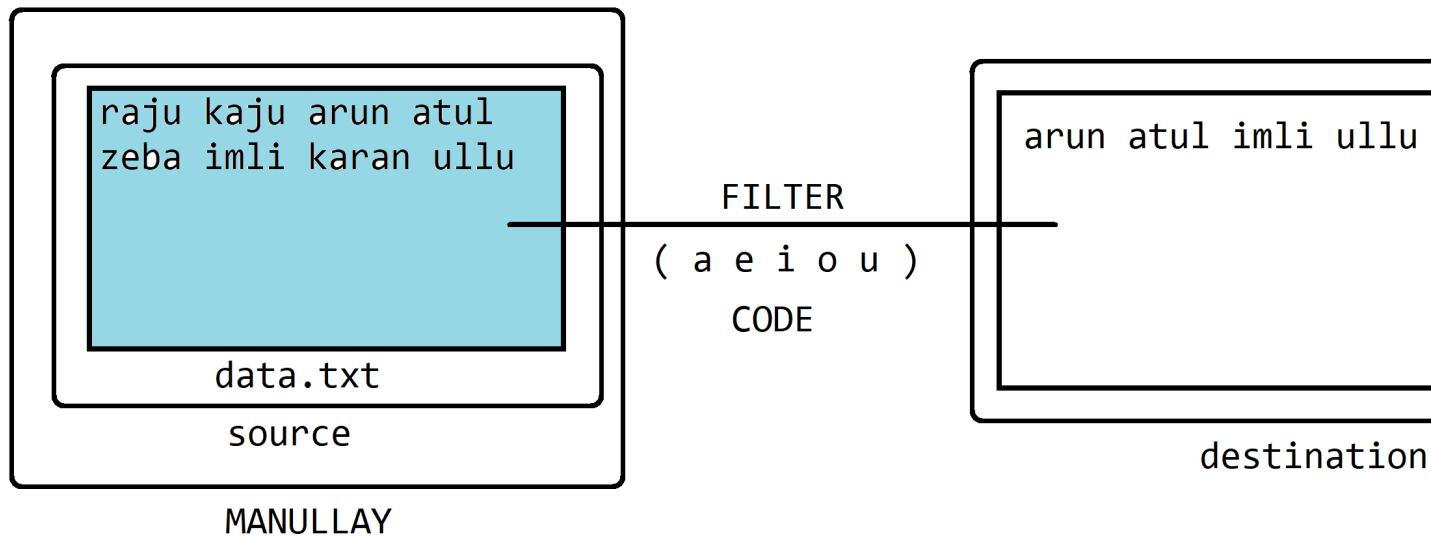
```

package com.mainapp;
import java.io.FileInputStream;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {
        try {

            FileInputStream fis = new
FileInputStream("C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\mydata.txt");
            while(true) {
                int i = fis.read();
                if(i==-1)break;
                System.out.print((char)i);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

TASK:



DAY-69

Java Full Stack Development

Batch 2025

FILE COPY PASTE

```
package com.mainapp;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {
        try {
```

```
String source="C:\\Users\\DELL\\Desktop\\FILEIO\\SOURCE\\img.png";
String destination="C:\\Users\\DELL\\Desktop\\FILEIO\\DESTINATION\\img.png";

    FileInputStream fis = new FileInputStream(source);
    byte[] b = fis.readAllBytes();

    FileOutputStream fos = new FileOutputStream(destination);
    fos.write(b);

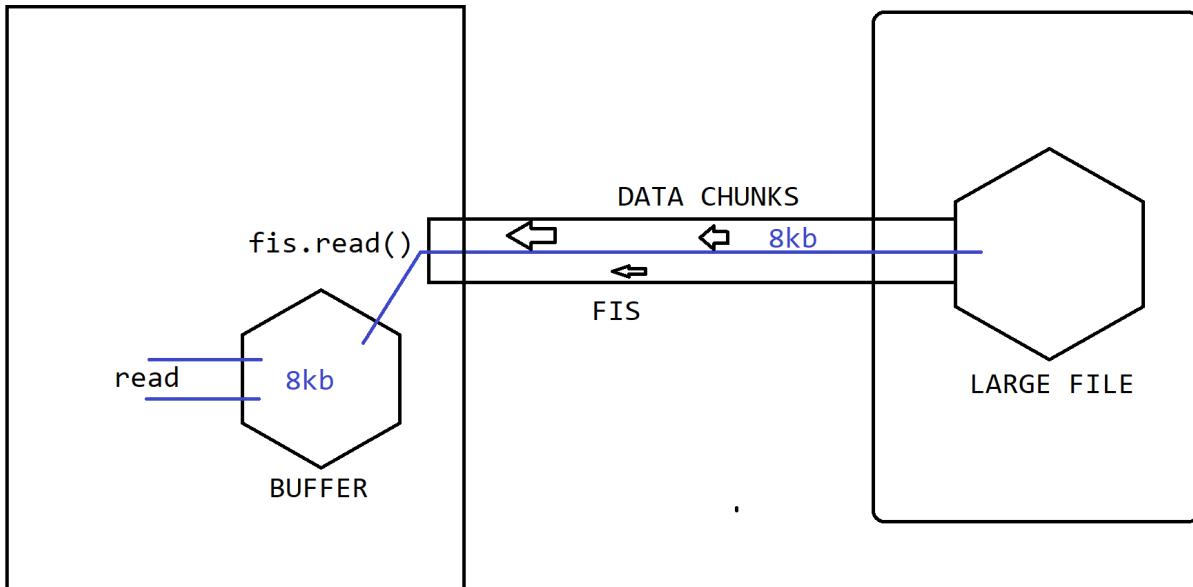
    fis.close();
    fos.close();

    File file = new File(source);
    file.delete();
    System.out.println("FILE MOVED");

} catch (IOException e) {
    e.printStackTrace();
}
}
```

BufferedInputStream & BufferedOutputStream

Note: FileInputStream and FileOutputStream is created to work on byte oriented data (like image, pdf....etc)



```

package com.mainapp;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {
        try {
            String
source="C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\abc.txt";
            FileInputStream fis = new FileInputStream(source);
            BufferedInputStream bis = new
BufferedInputStream(fis);

            while(true) {
                int i = bis.read();
                if(i==-1)break;
                System.out.print((char)i);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

package com.mainapp;

```

```

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class Launch {
    public static void main(String[] args) {

        FileOutputStream fos=null;
        BufferedOutputStream bos=null;
        try {

            String data="MYDATA";
            String des="C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\xyz.txt";

            fos = new FileOutputStream(des);
            bos = new BufferedOutputStream(fos);

            bos.write(data.getBytes());// ->exp

            //close

        } catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            try {
                bos.close();
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

NOTE: DO NOT USE FileInputStream & FileOutputStream for textual data

FileReader and FileWriter

->designed to work with textual data

```

package com.mainapp;
import java.io.FileReader;
public class Launch {
    public static void main(String[] args) {

        try {

```

```

        String
source="C:\\Users\\DELL\\Desktop\\FILEIO\\xyz.txt";
        FileReader fileReader = new FileReader(source);
        int temp=0;
        while((temp=fileReader.read())!=-1) {
            System.out.print((char)temp);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
//Connection PreparedStatement

```

DAY-70

Java Full Stack Development Batch 2025

1. How to create a folder/directory



The screenshot shows an IDE interface with two main panes. The left pane displays a Java code editor for a file named 'Launch.java'. The code creates a new file object pointing to 'C:\\Users\\DELL\\Desktop\\FILEIO\\myfolder' and checks if it was created successfully by printing 'CREATED' or 'FAILED' to the console. The right pane shows the 'Console' tab with the output 'CREATED'.

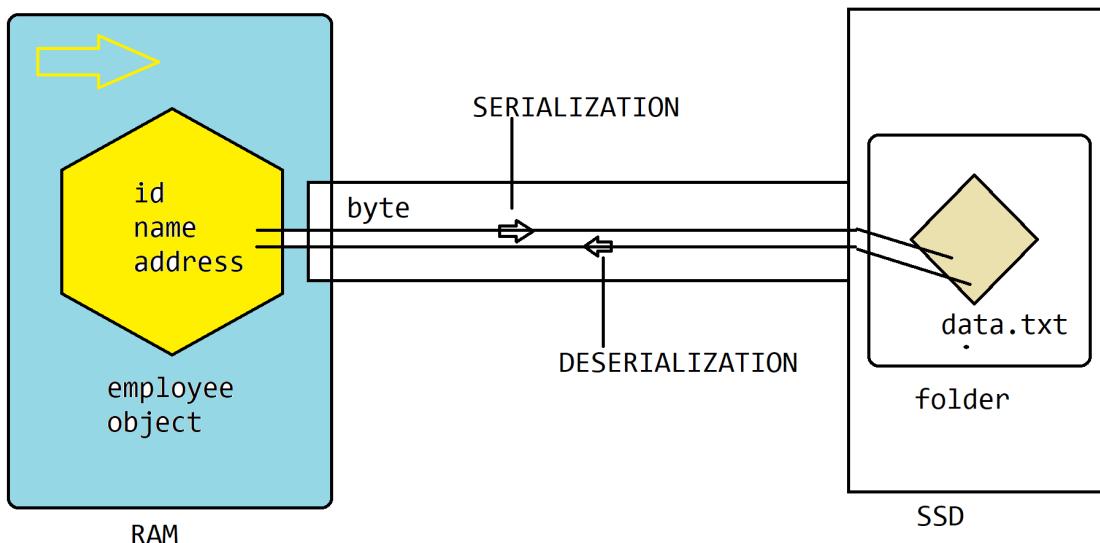
```

Launch.java ×
1 package com.mainapp;
2 import java.io.File;
3 public class Launch {
4
5     public static void main(String[] args) {
6
7         File file = new File("C:\\Users\\DELL\\Desktop\\FILEIO\\myfolder");
8         boolean mkdir = file.mkdir();
9
10        if(mkdir)System.out.println("CREATED");
11        else System.out.println("FAILED");
12    }
13 }
14

```

Console ×
 <terminated> Launch
 CREATED

SERIALIZATION & DESERIALIZATION



- **Serialization:** process of converting object into byte stream
- **Deserialization:** Reconstruction of Object from that byte stream
- **NOTE:** You cannot use FileReader & FileWrite to perform
 Serialization & Deserialization

```
package com.mainapp;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Launch {

    public static void main(String[] args) {
        // Employee employee = new Employee(11, "raju", 1000);
        // String path="C:\\Users\\DELL\\Desktop\\FILEIO\\DESTINATION\\data.txt";
        //
        // try
        // {
        //     FileOutputStream fos = new FileOutputStream(path);
        //     ObjectOutputStream oos = new ObjectOutputStream(fos);
        //     oos.writeObject(employee);
        //     System.out.println("SERIALIZED...");
        //     oos.close();
        //     fos.close();
    }
}
```

```

        }
        //
        // catch (Exception e) {
        //     e.printStackTrace();
        }

String path="C:\\\\Users\\\\DELL\\\\Desktop\\\\FILEIO\\\\DESTINATION\\\\data.txt";

try
{
    FileInputStream fis = new FileInputStream(path);
    ObjectInputStream ois = new ObjectInputStream(fis);

    Employee emp = (Employee) ois.readObject();
    System.out.println(emp.getId());
    System.out.println(emp.getName());
    System.out.println(emp.getSalary());

    ois.close();
    fis.close();
}
catch (Exception e) {
    e.printStackTrace();
}
}

}

package com.mainapp;
import java.io.Serializable;
public class Employee implements Serializable {

private int id;
private String name;
private int salary;

public Employee(int id, String name, int salary) {
    super();
    this.id = id;
    this.name = name;
    this.salary = salary;
}

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public int getSalary() {

```

```
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

**NOTE: you can skip any field being participate in
Serialization**

```
public class Employee implements Serializable {

    private int id;
    private String name;
    private transient int salary;

}
```

TASK:

WAP to achieve the functionality of RECYCLE BIN

