LOGICAL BUIDING

DAY 1----20 AUG 2024

Problem Solving

1. what is given in statements.....

2. requirements

3. solution

Algorithm

- Set of instruction.....finite....

- input then only we will output

addition 2 numbers.

1.take two numbers

2. add those two num

3. display the addition

First algorithm

1. Start

2.take 2 numbers

3. add those 2 numbers

4. display addition

5. end

------- 2nd VARIATIONS -----

1. start

2. declare number1 , number2, result

3. add number1 and number2

4. store in result

5. display result

6. end

----------3rd VARIATIONS----------

start

declare number1 , number2, result

input number 1, number2

add number1 and number2

store in result

display result

end

**** FLOWCHART ****

pictorial representation.....

---

find largest of two number

1. start

2. declare 2 number

3. input

4. compare 2 number eg..... < , >

5. display largest

6. end

----

print numbers divisible by two from 0 to 20.....

Java

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

(Demo,String,System:-each first letter is capital)

save as .java

1.to compile - javac <FileName with extention>

ex. javac Demo.java

2. to run - java <ClassName without extention>

ex. java Demo

- **SOME IMP POINTS-**

1.Raptor:-"The addition of 2 no:" +(N1+N2)

2.Notepad:-file path thevun cmd lihayache.

- Error asel tr class file open honar nahi.

- Main jis class mai hot hai vhi class lena chahiye.

-Class file proper nhi hoti.(not understandable).

-

```
int n1 = 10;
int n2 = 20;
int result ;
result = n1 + n2;

System.out.println( "Result " + result );
System.out.println( "Addtion of 2 numbers " + (n1 + n2) );
```

o/p- Result100

- **LOGICS-**
  - **1.even or odd-==========➔number%2==0**
  - **2.positive and neg-========➔n0>0**
  - **3.n1>n2>n3-==========>n1>n2….n1>n3…n2>n3**
  - **4.good morning==========➔(time<12 && time >5)**
  - **5.n>50 …put n….n-➔n+1===➔1 2 3 4 5 6 7 8 9 0**
  - **6.n>50….n->n+1…put n==== ➔2**

Food for Thought: Research and Read More About

1. History of Java: Explore the origin and development of the Java programming language. Who created Java, and why was it developed? How has it evolved over time?

A Brief History of Java: From Oak to the World

The Birth of Java

Java was originally conceived in 1991 by James Gosling, a Sun Microsystems engineer. The project, initially codenamed "Oak" (inspired by an oak tree outside Gosling's office), aimed to develop a programming language for consumer electronics devices. The goal was to create a language that was platform-independent and could run on a variety of devices.

The Shift to the Web

In the early 1990s, the internet was gaining momentum. Sun Microsystems saw an opportunity to leverage Java for web applications. They realized that Java's platform-independence could be a major advantage in the world of the web, where users were accessing content from various devices and operating systems.

The Name Change and Release

In 1995, the Oak project was renamed Java, inspired by a cup of coffee that Gosling had enjoyed during a brainstorming session. The first public release of Java, version 1.0, was released in January 1996. It quickly gained popularity due to its simplicity, reliability, and portability.

Evolution and Growth

Over the years, Java has undergone significant evolution. New features and improvements have been introduced in each major release. Some of the key milestones include:

- **Java 2 (J2SE, J2EE, J2ME):** This major release introduced three editions of Java: Standard Edition (SE), Enterprise Edition (EE), and Micro Edition (ME), catering to different use cases.
- **Java 5 (1.5):** This version introduced several new features, including generics, autoboxing/unboxing, enhanced for loops, and varargs.
- **Java 8:** A major release that brought significant changes, such as lambda expressions, method references, streams API, and a date/time API.
- **Java 11:** Introduced long-term support (LTS), making it a stable and recommended choice for production environments.
- **Java 17:** Another LTS release with features like sealed classes, records, and pattern matching for instanceof.

Why Java?

Java's success can be attributed to several factors:

- **Platform Independence:** Java's "write once, run anywhere" philosophy allows code to be compiled once and executed on any system with a Java Virtual Machine (JVM).
- **Object-Oriented Programming (OOP):** Java is a pure OOP language, promoting code reusability, modularity, and maintainability.

- **Robustness:** Java's strong type checking and memory management features help prevent common programming errors.
- **Security:** Java's security features, such as bytecode verification and sandboxing, make it suitable for enterprise applications.
- **Large Community and Ecosystem:** Java has a vast community of developers and a rich ecosystem of libraries, frameworks, and tools.

Conclusion

Java has come a long way since its inception. From a language for consumer electronics to a powerhouse for enterprise applications and web development, Java has proven its versatility and resilience. Its continued evolution and strong community support ensure that it will remain a dominant force in the programming world for years to come.

2. How Java is Useful & Problems It Solves: Research the specific problems Java addresses in software development. Why is Java preferred for certain types of projects (e.g., web development, mobile apps, enterprise systems)? What are some of its key strengths?

Java's Strengths and Applications

Java's versatility and robustness have made it a popular choice for a wide range of software development projects. Here's a breakdown of its key strengths and the problems it addresses:

Key Strengths of Java:

- **Platform Independence:** Java's "write once, run anywhere" philosophy allows developers to create applications that can run on any system with a Java Virtual Machine (JVM). This eliminates the need to develop separate versions for different platforms, saving time and effort.
- **Object-Oriented Programming (OOP):** Java is a pure OOP language, promoting code reusability, modularity, and maintainability. This makes it easier to develop complex applications and manage large codebases.
- **Robustness:** Java's strong type checking and memory management features help prevent common programming errors, leading to more reliable and stable applications.
- **Security:** Java's security features, such as bytecode verification and sandboxing, make it suitable for enterprise applications that handle sensitive data.
- **Large Community and Ecosystem:** Java has a vast community of developers and a rich ecosystem of libraries, frameworks, and tools. This provides developers with a wealth of resources and support.

Problems Java Addresses:

- **Cross-Platform Development:** Java's platform independence solves the problem of creating applications that need to run on multiple operating systems and devices.
- **Complex Software Systems:** Java's OOP features and robust design make it well-suited for developing large-scale, enterprise-level applications.
- **Performance and Scalability:** Java's Just-In-Time (JIT) compilation and garbage collection mechanisms ensure good performance and scalability, making it suitable for high-traffic applications.
- **Security Concerns:** Java's security features help address concerns related to data privacy and protection, making it a reliable choice for applications that handle sensitive information.

Types of Projects Where Java is Preferred:

- **Web Development:** Java is widely used for building web applications and web services, thanks to frameworks like Spring and Hibernate.
- **Enterprise Applications:** Java's scalability, robustness, and security make it an ideal choice for developing enterprise-level systems, such as ERP, CRM, and content management systems.

- **Mobile App Development:** While not as popular as Kotlin for Android development, Java can still be used to create Android apps, especially for large-scale projects.
- **Big Data and Analytics:** Java's ecosystem includes tools and frameworks like Hadoop and Spark, which are widely used for big data processing and analytics.
- **Embedded Systems:** Java's Embedded Systems Edition (Java ME) is used for developing applications for embedded devices, such as smartphones, IoT devices, and set-top boxes.

In summary, Java's combination of strengths makes it a versatile and powerful language that can address a wide range of software development challenges. Its platform independence, object-oriented nature, robustness, security, and large community make it a popular choice for developers worldwide

3***. Role of the Java Virtual Machine (JVM): Investigate the purpose of the JVM in the execution of Java programs. How does it enable Java's platform independence (i.e., "Write Once, Run Anywhere")?***

The Java Virtual Machine (JVM): A Platform-Independent Execution Environment

The Java Virtual Machine (JVM) is a crucial component of the Java ecosystem. It acts as an intermediary between the compiled Java bytecode and the underlying operating system. The JVM's primary role is to execute Java programs in a platform-independent manner, fulfilling the "Write Once, Run Anywhere" (WORA) principle.

How the JVM Enables Platform Independence

1. ***Bytecode Compilation: When a Java source code file is compiled, it is transformed into bytecode. Bytecode is a platform-neutral intermediate representation of the program.***
2. ***JVM Loading: The JVM loads the compiled bytecode into memory.***
3. ***Bytecode Verification: The JVM verifies the bytecode to ensure that it adheres to Java's syntax, semantics, and security rules. This helps prevent malicious code from executing.***
4. ***Just-In-Time (JIT) Compilation: The JVM uses a JIT compiler to translate the bytecode into machine code that can be directly executed by the underlying hardware. This process is performed on-the-fly, optimizing the code for the specific system.***
5. ***Execution: The machine code generated by the JIT compiler is executed by the CPU.***
6. ***Garbage Collection: The JVM automatically manages memory allocation and deallocation through garbage collection. This relieves developers from the burden of manual memory management, reducing the risk of memory leaks.***

Benefits of the JVM

- ***Platform Independence: The JVM abstracts away the underlying hardware and operating system, allowing Java programs to run on any platform that has a compatible JVM.***
- ***Portability: Java applications can be easily ported to different environments without requiring significant code changes.***
- ***Security: The JVM's bytecode verification and security mechanisms help protect against malicious code.***
- ***Performance: JIT compilation can optimize code for specific hardware, improving performance.***
- ***Memory Management: Automatic garbage collection simplifies memory management for developers.***

In conclusion, the JVM plays a vital role in enabling Java's platform independence and portability. It provides a consistent execution environment for Java programs, regardless of the underlying hardware or operating system. This has made Java a popular choice for a wide range of applications, from web development to enterprise systems.

4.Java Runtime Environment (JRE): Read about how the JRE fits into the picture when running

Java applications. What does the JRE provide, and why is it essential?

The Java Runtime Environment (JRE): A Vital Component

The Java Runtime Environment (JRE) is a critical component of the Java ecosystem. It provides the necessary environment for running compiled Java applications. Essentially, the JRE is a bundle of software that includes the Java Virtual Machine (JVM), class libraries, and other supporting components.

What the JRE Provides:

- *JVM: As discussed earlier, the JVM is the core component of the JRE. It acts as an interpreter for Java bytecode, translating it into machine code that can be executed on the underlying hardware.*
- *Class Libraries: The JRE includes a comprehensive set of class libraries that provide pre-written code for common tasks, such as input/output, networking, and data structures. These libraries save developers time and effort by providing reusable components.*
- *Other Components: The JRE may also include additional components, such as security tools, deployment tools, and Java Web Start.*

Why the JRE is Essential:

1. *Execution of Java Applications: The JRE is necessary for running any Java application. It provides the environment where the JVM can execute the compiled bytecode.*
2. *Platform Independence: The JRE's role in executing Java bytecode is crucial for maintaining platform independence. As long as a system has the JRE installed, it can run Java applications, regardless of the underlying operating system or hardware.*
3. *Access to Class Libraries: The JRE's class libraries provide developers with a rich set of tools and functionalities. This helps them build applications more efficiently and effectively.*
4. *Security: The JRE includes security features that help protect against malicious code and unauthorized access.*

In essence, the JRE is the essential component that bridges the gap between the compiled Java code and the underlying hardware. It provides the environment, tools, and libraries needed for Java applications to run smoothly and securely.

5.Difference Between JDK, JRE, and JVM: Understand the differences and relationships between the Java Development Kit (JDK), Java Runtime Environment (JRE), and Java Virtual Machine (JVM). How do these components work together when a Java program is written, compiled, and executed?

JDK, JRE, and JVM: A Breakdown

Java Development Kit (JDK)

- ***Purpose: A comprehensive software development kit that provides tools for creating, compiling, running, and debugging Java applications.***
- ***Components: Includes the JRE, compiler (javac), debugger (jdb), and other development tools.***

Java Runtime Environment (JRE)

- ***Purpose: The environment required to run Java applications.***
- ***Components: Includes the JVM, class libraries, and other supporting components.***

Java Virtual Machine (JVM)

- ***Purpose: The interpreter that executes Java bytecode.***
- ***Role: Translates the bytecode into machine-specific code for execution on the underlying hardware.***

How They Work Together

1. ***Writing Java Code: Developers write Java code using a text editor or integrated development environment (IDE).***
2. ***Compilation: The Java compiler (part of the JDK) takes the Java source code and compiles it into bytecode.***
3. ***Execution: The JVM (part of the JRE) loads the bytecode into memory and interprets it, translating it into machine-specific code for execution.***
4. ***Class Libraries: The JRE's class libraries provide pre-written code for common tasks, which the JVM can use during execution.***

Relationship Between the Components

- ***JDK contains JRE: The JDK includes the JRE as a subset. This means that if you have the JDK installed, you automatically have the JRE.***
- ***JRE contains JVM: The JRE includes the JVM as its core component. The JVM is responsible for executing Java bytecode.***

In summary:

- ***JDK is the complete toolkit for Java developers.***

- ***JRE is the environment needed to run Java applications.***
- ***JVM is the heart of the JRE, responsible for executing Java bytecode.***

When a Java program is written, compiled, and executed, the JDK, JRE, and JVM work together in the following sequence:

1. ***JDK: The compiler in the JDK compiles the Java source code into bytecode.***
2. ***JRE: The JRE provides the environment for running the bytecode, which includes the JVM.***
3. ***JVM: The JVM loads the bytecode and executes it, translating it into machine-specific code.***

6.Memory Areas in JVM: Explore the different types of memory areas within the JVM, such as the Heap, Stack, and Method Area. What roles do these memory areas play during the execution of a Java program?

Show drafts

Memory Areas in the JVM

The Java Virtual Machine (JVM) manages memory in different areas to efficiently execute Java programs. These areas are:

1. Heap

- *Purpose: Stores objects created during program execution.*
- *Organization: Dynamically allocated and deallocated using garbage collection.*
- *Types:*
    - *Young Generation: Stores newly created objects that are likely to be short-lived.*
    - *Old Generation: Stores objects that have survived multiple garbage collection cycles.*
    - *Permanent Generation (Deprecated in Java 8): Previously used to store class metadata, but now part of the meta space.*

2. Stack

- *Purpose: Stores information about methods being executed, including local variables, method arguments, and return values.*
- *Organization: Follows a LIFO (Last-In-First-Out) order.*
- *Lifetime: A new stack frame is created for each method call and is removed when the method returns.*

3. Method Area

- *Purpose: Stores class-level information, such as the class bytecode, runtime constant pool, field and method data, and the class initializer.*
- *Organization: Shared across all threads of a JVM.*
- *Lifetime: Remains in memory until the class is unloaded.*

4. PC Registers

- *Purpose: Points to the address of the instruction currently being executed.*

- ***Organization: Each thread has its own PC register.***

Roles of These Areas

- ***Heap: Objects are allocated and deallocated in the heap. Garbage collection is responsible for reclaiming unused memory.***
- ***Stack: The stack is used to manage method calls and local variables. When a method is called, a new stack frame is created to store its local variables and arguments. When the method returns, its stack frame is popped off the stack.***
- ***Method Area: The method area stores information about classes and methods, such as the class bytecode and the runtime constant pool. This information is used by the JVM to load and execute classes.***
- ***PC Registers: PC registers keep track of the current instruction being executed in each thread.***

In summary, these memory areas work together to provide the environment for Java programs to execute. The heap stores objects, the stack manages method calls, the method area stores class-level information, and the PC registers keep track of the current instruction being executed. Understanding these memory areas is essential for optimizing Java applications and avoiding memory-related issues.

7.Primitive Data Types in Java: Learn about Java's primitive data types. What are they, and how are they different from reference data types? List and explain the eight primitive data types in Java.

Primitive Data Types in Java

Primitive data types in Java are the most basic building blocks of the language. They represent simple values, unlike reference data types which refer to objects. Primitive data types are stored directly on the stack, while reference data types store references to objects on the heap.

Eight Primitive Data Types in Java:

1. *byte: Represents a single byte of data (8 bits). It is used for small integer values.*
2. *short: Represents a 16-bit signed integer. It is also used for small integer values.*
3. *int: Represents a 32-bit signed integer. It is the most commonly used integer data type.*
4. *long: Represents a 64-bit signed integer. It is used for very large integer values.*
5. *float: Represents a 32-bit floating-point number. It is used for decimal numbers with a fractional part.*
6. *double: Represents a 64-bit floating-point number. It is used for decimal numbers with a higher precision than float.*
7. *char: Represents a single character. It is used to store individual characters.*
8. *boolean: Represents a boolean value (true or false). It is used for logical expressions.*

Key Differences Between Primitive and Reference Data Types:

- *Storage: Primitive data types are stored directly on the stack, while reference data types store references to objects on the heap.*
- *Size: Primitive data types have fixed sizes, while the size of reference data types depends on the object they refer to.*
- *Comparison: Primitive data types are compared by value, while reference data types are compared by reference (whether they refer to the same object).*
- *Null Values: Primitive data types cannot be null, while reference data types can be null.*

Example:

Java

```
int age = 25; // Primitive data type (integer)
```

String name = "Alice"; // Reference data type (String object)

Use code with caution.

In this example, age is a primitive data type that stores the integer value 25 directly on the stack. name is a reference data type that stores a reference to a String object on the heap.

Understanding the differences between primitive and reference data types is crucial for effective Java programming. It helps you choose the appropriate data type for different scenarios and avoid common programming errors.

Sources and related content

DAY 2-21 AUG 2024

JAVA.....

Author - james gosling.

open source lang.

1995

initial version - 1

latest version - 24

java 1.8

intial name - OAK

extentions --- .java & .class , .jar(advJava)

SunMirco systems

oracle

WORA - write once run anywhere

Parts of Java

1. CoreJava

2. Adv Java

- **a. j2se java 2 std edition**
- **b. j2ee jave 2 enterprise edition**
- **c. j2me java 2 mirco edition**

jdbc

jsp

servlet

spring

spring boot

hibernate

Java Features -

1. Simple

-pointer

-multiple inheritance

2. Platform independent

- .class

javac Demo.java .... windows

.class we can run on any other OS.

3. Compiled & interpreted

4. OOP

```
class demo{


}


demo obj = new demo();
```

5. Secure

6. Dynamic language

7. high performace

8. robust

9. multithread

10. portable & distributed

11. architecutral

----

class

- **1. instance Variable**
- **2. static variable**
- **3. contr.**
- **4. Method**
- **5. Instace Block & static Block**

Variable -

Local

global

Primitve

Non-primitve

**** Data Types**

----

1. byte - 1 byte - -128 to 127

2. short - 2 byte -

3. int - 4 byte -

4. long - 8 byte -

5. float - 4 byte -

6. double - 8 byte -

7. char - 2 byte -

8. boolean - 1 bit - (true / false)

9. String -

dataType Name; // declare

dataType Name1 = 10; // Declare & initilse/definitions.

Tokens -

smallest individual part a program

indentifiers -

name given to class, method, variable....

rules

1. shld start with char or _ or $

2. numbers are not allowed at the start of variable name

3. NO duplicate

4.

5.

6.

Java Statements -

- if

- if-else

- switch

loops

- for

- while

- do-while

1.if

```
if(condtion){
statements
statements
}
```

2. if - else

```
if(condtion){
statements
}
else{
statements
statements
}
```

3. swtich

```
switch (case){
case :
        statements;
        statements;
        break;
 }
```

- **SOME POINTS-**

  1.Open source ->any one access.

  2.oracle ke pass h java abhi.

-128 to 127

128 127 ~~loop~~ +26

24 →

① String is non-primitive.
② boolean → 1 bit. (T or F)
③ ~~bcccc~~ → byte, boolean
④ Newpad++ → features ~~नहीं~~ है
⑤ byte b = (byte) 130;
space देता है

⑥ short → -32768 to 32,767
⑦ byte b2 = a;
sysout(b2); //error.
⑧ 1 bit
1 byte - 8 bit
1kb - 1024 byte
1mb - 1024 kb
1gb - 1024 mb

gb > mb > kb > byte > bit

⑨ Decimal value → binary
⑩ 10/2 = 0
5/2 = 1
1/2 = 1
(bottom to top) लिखते है

⑪ byte b2 = a;
sysout(b2); //error

⑫ char c = 'A';
sysout(c) // print

(15) $variable → चलता है।

(16) case sensitive → जिस case में किया उसी में print करें।

(17) No use keywords for variable

(18) 52 keywords in Java

(19) String is class name.
No class name as a variable or method name

(20) MAX_VALUE
MAX_PRIORITY
→ constant Variable).
in capital letters only

Logic (21) Eligible for voting →

```
int age = 17
if(age.7=18)
```

(22)

```
if (a) //error.
if (1) //error.
if(true) // ✓
if(flase)// ✓
```

Logic (23) Marks

```
else if
if (marks >= 85)
```

(24)

```
int age = 15;
if (age > 18 || age == 18) ;
```

0 + 1 = 1

||

1 * 0 = 0

To gain score

1. Eng sy nn → ~~[illegible]~~ Indiabix
2. Apti → - || -
3. Rea → - || -
4. CF → - || -

11. char c = 'A';
sy00 (int) c) → For numerical value

12. syso ((char)(66)); //B.

13. 65 to 90 → A to Z
97 to 122 → a to z
48 to 57 → 0 to 9

16. syso ((char) 125); // closing error brou
→ char उस Range में नहीं है) ?

13. ASCII → 0 to 255 range.

14. int a = 10;
Float f = a;
syso (f) // 10.0.

- Float f1 = 10.0 (double)
// f1 = 10.0f (Float).
syso (f1);
- int a1 = (int) f1;
- syso (a1); // size pt होरहा.

(26) switch → ~~no~~ case ~~में अलग~~ दिया हो and वो case में नहीं है। तब भी compile होगा है। But कोई value नहीं देता।

(27) switch → case '2' // ✓
case 2.0 // ✗ float & double not allowed
other are allowed (int, string) short, lon

(28)

① We can have multiple main methods in a Java class.
means → public static void आपको multiple times ~~लिख सकते~~

② int x = y+10 //error

③ int x = "Hello"; //error.

④ Method overloading allowed in Java.

```
public void display () {
    Syso ("No Parameter");
}
public void display (int num) {
    sysout("with parameter :" +num);
}
psvm (String [] args) {
    display ();
    display (5);          // Pt honga.
}
```

Pranjal Thiemade Madhavdas.

⑤ while (True)
{sysout} // Indefinite loop
Print statement

⑥ int result = (int) (num1 + num2));
cast to the desired data type

⑦ order of operations →
(operator precedence):
1) Parentheses ( )
2) Exponentiation ^
3) * 4 /
4) + 4 −

⑧ * Method cannot be declared within another method.

⑨

Hackerrank
① space
sysout ("

```java
        //type casting
        byte b1 = 127; // implicit
        byte b = (byte)130; // explicit
        //System.out.println(b);

        int a = 10;
        //byte b2 = a; error
        //System.out.println(b2);// error

        char c = 'A';
        System.out.println(c);          // A
        System.out.println((int)c);     //65
        System.out.println((char)66);   // B
    System.out.println((char)125);  // closing curly brace
    System.out.println((char)500); //
    System.out.println((char)499); //
    System.out.println((char)255); //
    System.out.println((char)256); //

    //ASCII - 0 to 255 range

/*1 bit
1 byte - 8 bit
1 kb  - 1024 byte
1 mb  - 1024 kb
1 gb - 1024 mb

Binary - 0's and 1's

10 ( Decimal ) - 1010

10 % 2   = 0
5 % 2 = 1
2 % 2 = 0
1 % 2 = 1
    byte b3 = 10;
    int a1 = b3;
    System.out.println(a1);

    }
}
```

```java
/*int a = 10;
    float f = a;

    System.out.println(f);
    //20.8978
```

```
❖
❖     float f1 = 10.0f;
❖     System.out.println(f1); //10
❖
❖     int a1 = (int)f1;
❖     System.out.println(a1);  //4
❖
❖     //int 1variable = 10;
❖     //System.out.println(1variable); //ERROR
❖
❖     int $variable = 10;
❖     //int $variable = 10;  //EXECUTE
❖     System.out.println($variable); // 10
❖     $variable = 20;
❖     System.out.println($variable); // 20
❖
❖
❖     //int @variable = 10;
❖     //System.out.println(@variable);  //NOT
❖
❖     float FloatMarks = 10.5f;
❖     System.out.println(FloatMarks);
❖     float floatMarks1 = 10.5f;
❖     System.out.println(floatMarks1);*/
❖
❖
❖     //int int = 10;
❖     //System.out.println(int);
❖
❖     //int String = 100;
❖     //System.out.println(String);
❖
❖     //MAX_VALUE
❖     //MAX_PRIORITY
❖     //MIN_PRIORITY
❖
❖     }
❖
❖ }
```

```
❖ /*error
❖             case 2.0:
❖             System.out.println("double 2");
❖                 break;
❖             */
❖ Question 3: Calculator
❖ // Write a program that acts as a simple calculator. It should accept two numbers and
  an operator
```

```java
// (+, -, *, /) as input. Use a switch statement to perform the appropriate operation. Use nested if-
// else to check if division by zero is attempted and display an error message.

// import java.util.Scanner;

// class Programs {
//     public static void main(String[] args) {
//         char operator;
//         int n1, n2, result;

//         // Create an object of Scanner class
//         Scanner input = new Scanner(System.in);

//         // Ask users to enter operator
//         System.out.println("Choose operator: +, -, *, /");
//         operator = input.next().charAt(0);   // carefully baghane

//         // Ask users to enter numbers
//         System.out.println("Enter first number");
//         n1 = input.nextInt();

//         System.out.println("Enter second number");
//         n2 = input.nextInt();

//         switch (operator) {
//             case '+':
//                 result = n1 + n2;
//                 System.out.println(n1 + " + " + n2 + " = " + result);
//                 break;
//             case '-':
//                 result = n1 - n2;
//                 System.out.println(n1 + " - " + n2 + " = " + result);
//                 break;
//             case '*':
//                 result = n1 * n2;
//                 System.out.println(n1 + " * " + n2 + " = " + result);
//                 break;
//             case '/':
//                 if (n2 == 0) {
//                     System.out.println("Error: Division by zero is not allowed.");
//                 } else {
//                     result = n1 / n2;
//                     System.out.println(n1 + " / " + n2 + " = " + result);
```

```
Question 4: Discount Calculation
// Write a program to calculate the discount based on the total purchase amount. Use the following
// criteria:
// • If the total purchase is greater than or equal Rs.999, apply a 10% discount.
// • If the total purchase is less than Rs.500, apply a 5% discount.
// Additionally, if the user has a membership card, increase the discount by 5%.

// import java.util.Scanner;

// class Programs{
//     public static void main (String args[]){
//         double totalpurchase,discount=0;//discount=0; initial karan ajarurui hai
//         boolean hasmembershipcard;

//         Scanner input =new Scanner(System.in);

//         //ask the user for the total purchase amount
//         System.out.println("Entert total purchase");
//         totalpurchase=input.nextDouble();

//         // ask the user if they have a membership card
//         System.out.println("Do you have membership card?(yes/no)");
//         hasmembershipcard=input.nextBoolean();


//         if (totalpurchase>=1000) {
//             discount=20;
//         }
//         else if (totalpurchase>=500 && totalpurchase<=999) {
//                 discount=10;
//         }
//         else if (totalpurchase<500) {
//             discount=5;
//     }
//          if  (hasmembershipcard) {
//                     discount += 5;

//          }
//          double finalprice=totalpurchase-(totalpurchase * discount / 100);
//          System.out.println("totalpurchase="+totalpurchase);
//          System.out.println("discount="+ discount + "%");
//          System.out.println("final price after discount :"+finalprice);


// }
// }
```

```java
// Question 5: Student Pass/Fail Status with Nested Switch
// Write a program that determines whether a student passes or fails based on their grades in three
// subjects. If the student scores more than 40 in all subjects, they pass. If the student fails in one or
// more subjects, print the number of subjects they failed in.
         import java.util.Scanner;
         class Programs{
            public static void main(String args[]){
            int eng ,math,phy,chem,failedcount=0;
            try (Scanner input = new Scanner(System.in)) {//carefully see why try is here used
                {
                    System.out.println("Enter the marks of Eng ");
                                eng=input.nextInt();
                                System.out.println("Enter the marks of phy ");
                                phy=input.nextInt();
                                System.out.println("Enter the marks of chem");
                                chem=input.nextInt();
                                System.out.println("Enter the marks of math ");
                                math=input.nextInt();
                }
            }
                 switch(eng>40?1:0){
                 case 0:
                failedcount++;
            }

            switch(math>40?1:0){
                case 0:
                failedcount++;
           }

           switch(phy>40?1:0){
            case 0:
            failedcount++;
        }

        switch(chem>40?1:0){
         case 0:
         failedcount++;
    }
           switch(failedcount){
            case 0:
            System.out.println("Student pass");
            break;

            default:
            System.out.println("student fail-" +failedcount +"subject");
            break;
```

```java
class SwtichDemo{
    Run | Debug
    public static void main(String args[]){

        int day = 2;

        switch(day){
            default:
                System.out.println(x:"invalid input");
                break;
            case 3:
                System.out.println(x:"Today is Wed");
                break;
            case 1:
                System.out.println(x:"Today is Monday");
                break;
            case 2:
                System.out.println(x:"Today is Tues");
                System.out.println(x:"Today is Tues");
                System.out.println(x:"Today is Tues");System.out.println(x:"Today is Tues");System.out.println(x:"Today is Tues");
                break;
            case '2':
                System.out.println(x:"char 2");
                break;
            /*error
            case 2.0:
            System.out.println("double 2");
                break;
            */
                System.out.println(x:"int switch");

        }
        System.out.println(x:"after switch");
```

```
LEMS 5  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
ning] cd "d:\CDAC\Logical Building\Day 2\Program\" && javac SwtichDemo.java && java SwtichDemo
chDemo.java:29: error: unreachable statement
            System.out.println("int switch");
            ^
ror
```

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

    String day = "true";

    //int a = 3, b= 1;
    switch(day){
```

//CASE true HAI TOH HONGA EXECUTE

Section 3: Food for Thought: Research and Read More

About

1. Evolution of Programming Languages

 Research Topic: Explore the different levels of programming languages: Low-level, High-level, and Assembly-level languages.

o Questions to Ponder:

 What is a Low-level language? Give examples and explain how they work.

 What is a High-level language? How does it differ from a low-level language in terms of abstraction and usage?

 What is an Assembly-level language, and what role does it play in programming?

 Why do we need different levels of programming languages? What are the trade-offs between simplicity and control over the hardware?

Evolution of Programming Languages: A Deep Dive

Low-Level Languages

Low-level languages are programming languages that are very close to the machine code that a computer's processor can directly understand. They offer a high degree of control over the hardware but are often difficult to read and write.

Examples:

- Machine Code: The most basic form of programming language, directly executable by the processor.
- Assembly Language: A human-readable representation of machine code, using mnemonics to represent instructions.

How they work:

- Low-level languages require programmers to specify instructions at a very granular level, dealing with memory addresses, registers, and machine operations.
- This level of detail gives programmers maximum control over the hardware, but it also makes programming more complex and time-consuming.

High-Level Languages

High-level languages are programming languages that are designed to be easier for humans to read and write. They abstract away the complexities of machine code, making it easier to focus on the problem being solved rather than the underlying hardware.

Examples:

- Python: A popular general-purpose language known for its readability and simplicity.
- Java: A versatile language used for a wide range of applications, including web development, mobile app development, and enterprise software.
- C++: A powerful language that combines elements of both high-level and low-level programming.

Differences from low-level languages:

- Abstraction: High-level languages provide a higher level of abstraction, allowing programmers to focus on the logic of their programs rather than the underlying hardware details.
- Usage: High-level languages are generally easier to learn and use, making them more accessible to a wider range of programmers.

Assembly-Level Language

Assembly-level language is a low-level language that uses mnemonics to represent machine code instructions. It is a bridge between high-level languages and machine code.

Role in programming:

- Optimization: Assembly language can be used to optimize performance-critical sections of code.
- System programming: Assembly language is often used for system programming tasks that require direct interaction with the hardware.
- Reverse engineering: Assembly language is used to analyze and understand the behavior of compiled programs.

Why different levels of programming languages?

Different levels of programming languages are needed to address various needs and trade-offs:

- Simplicity: High-level languages are easier to learn and use, making them suitable for a wider range of programmers.
- Control: Low-level languages offer greater control over the hardware, allowing for highly optimized and efficient programs.
- Performance: Assembly language can be used to achieve the highest possible performance, but it requires a deep understanding of the hardware and is more time-consuming to write.

The choice of programming language depends on the specific requirements of the project, including factors such as performance, portability, and development time.

2. Different Programming Languages and Their Usage

 Research Topic: Explore different programming languages and understand their use cases.

o Questions to Ponder:

 What are the strengths and weaknesses of languages like C, Python, Java, JavaScript, C++, Ruby, Go, etc.?

 In which scenarios would you choose a specific language over others? For example, why would you use JavaScript for web development but Python for data science?

 Can one programming language be used for all types of software development? Why or why not?

Different Programming Languages and Their Usage

C

- Strengths: Fast, efficient, low-level control, widely used for system programming and embedded systems.
- Weaknesses: Complex syntax, prone to memory management errors, requires more manual effort for development.
- Use cases: Operating systems, device drivers, compilers, embedded systems, performance-critical applications.

Python

- Strengths: Readable, easy to learn, versatile, large standard library, popular for data science, machine learning, and web development.
- Weaknesses: Can be slower than compiled languages, not suitable for performance-critical applications.
- Use cases: Data science, machine learning, web development, scripting, automation, scientific computing.

Java

- Strengths: Object-oriented, platform-independent, robust, widely used for enterprise applications and Android development.
- Weaknesses: Can be verbose, slower than compiled languages like C++, requires a virtual machine for execution.
- Use cases: Enterprise applications, Android development, web development, big data, scientific computing.

JavaScript

- Strengths: Client-side scripting for web pages, versatile, used for both front-end and back-end development (Node.js).
- Weaknesses: Can be less performant than compiled languages, can have quirks and inconsistencies.
- Use cases: Web development (front-end and back-end), game development, mobile app development (React Native).

C++

- Strengths: Powerful, efficient, object-oriented, can be used for both system programming and application development.
- Weaknesses: Complex syntax, requires careful memory management, can be more difficult to learn than higher-level languages.
- Use cases: System programming, game development, high-performance computing, scientific computing.

Ruby

- Strengths: Ruby on Rails framework for web development, dynamic, easy to learn, emphasizes productivity and developer happiness.
- Weaknesses: Can be slower than compiled languages, not as widely used as other languages.
- Use cases: Web development (Ruby on Rails), scripting, automation.

Go

- Strengths: Efficient, concurrent, easy to learn, suitable for building scalable and reliable systems.
- Weaknesses: Relatively new, smaller ecosystem compared to more established languages.
- Use cases: Web development, cloud computing, networking, system programming.

Choosing the Right Language

The choice of programming language depends on various factors, including:

- Project requirements: The specific needs and goals of the project.
- Performance: The required performance level of the application.
- Development time: The desired development speed and productivity.
- Team expertise: The skills and experience of the development team.
- Ecosystem: The availability of libraries, tools, and community support.

Examples:

- For web development, JavaScript is a popular choice for both front-end and back-end development. Python with frameworks like Django or Flask is also widely used for web development, especially for data-driven applications.
- For data science and machine learning, Python is a leading choice due to its extensive libraries and tools like NumPy, Pandas, and TensorFlow.
- For system programming and performance-critical applications, C or C++ are often preferred for their efficiency and low-level control.

Can one programming language be used for all types of software development?

While it's possible to use a single programming language for many different types of software development, there are often trade-offs involved. For example, Python is a versatile language that can be used for a wide range of tasks, but it may not be the best choice for performance-critical applications.

In general, it's often more efficient to use the language that is best suited for the specific task at hand. This can help to improve development speed, code quality, and maintainability.

3. Which Programming Language is the Best?

 Research Topic: Investigate the debate around the "best" programming language.

o Questions to Ponder:

 Is there truly a "best" programming language? If so, which one, and why?

 If a language is considered the best, why aren't all organizations using it? What factors influence the choice of a programming language in an organization (e.g., cost, performance, ecosystem, or community support)?

 How do trends in programming languages shift over time? What are some emerging languages, and why are they gaining popularity?

The "Best" Programming Language: A Myth Debunked

Is there truly a "best" programming language?

The short answer is no. There is no single "best" programming language that can be universally applied to all software development projects. The "best" language depends on various factors, including:

- **Project requirements:** The specific needs and goals of the project.
- **Performance:** The required performance level of the application.
- **Development time:** The desired development speed and productivity.
- **Team expertise:** The skills and experience of the development team.
- **Ecosystem:** The availability of libraries, tools, and community support.

Why aren't all organizations using the "best" language?

Even if a language were considered the "best" in all respects, there are several factors that influence the choice of a programming language in an organization:

- **Cost:** Some languages or frameworks may have licensing costs or require specialized hardware, which can increase development costs.
- **Performance:** Certain languages or frameworks may not be suitable for performance-critical applications.
- **Ecosystem:** The availability of libraries, tools, and community support can significantly impact development efficiency and productivity.
- **Legacy systems:** Organizations may be constrained by existing legacy systems that use specific programming languages.

- **Team expertise:** The skills and experience of the development team can influence the choice of language.

How do trends in programming languages shift over time?

Trends in programming languages can shift over time due to various factors, including:

- **Technological advancements:** New technologies and platforms can drive the adoption of new languages.
- **Community growth:** A growing community of developers can contribute to the popularity of a language.
- **Ease of use:** Languages that are easier to learn and use can become more popular.
- **Performance improvements:** Improvements in language performance can make them more attractive for certain use cases.

Emerging languages and their popularity:

Some emerging programming languages that are gaining popularity include:

- **Rust:** Known for its safety, speed, and concurrency features, Rust is gaining traction for systems programming and embedded systems.
- **Kotlin:** A modern language designed to interoperate with Java, Kotlin is becoming popular for Android development and backend services.
- **TypeScript:** A superset of JavaScript that adds static typing, making it more suitable for large-scale projects.
- **Swift:** A modern language developed by Apple for iOS and macOS development.
- **Go:** A language designed by Google for building scalable and reliable systems, gaining popularity for cloud computing and networking.

The popularity of these languages is driven by factors such as their features, performance, and community support. As technology continues to evolve, new programming languages may emerge and become popular in the future.

4. Features of Java

 Research Topic: Dive deep into the features of Java.

o Questions to Ponder:

 Why is Java considered platform-independent? How does the JVM contribute to this feature?

 What makes Java robust? Consider features like memory management, exception handling, and type safety. How do these features contribute to its robustness?

 Why is Java considered secure? Explore features like bytecode verification, automatic garbage collection, and built-in security mechanisms.

? Analyze other features like multithreading, portability, and simplicity. Why are they important, and how do they impact Java development

Features of Java

Platform Independence

Java is considered platform-independent because it can run on any system that has a Java Virtual Machine (JVM) installed. The JVM acts as an intermediary between the Java code and the underlying hardware. It translates the Java bytecode (the compiled form of Java code) into machine code that can be executed on the specific platform. This allows Java programs to be written once and run anywhere (WORA).

Robustness

Java is known for its robustness due to several features:

- Memory Management: Java uses automatic garbage collection to manage memory allocation and deallocation. This reduces the risk of memory leaks and other memory-related errors.
- Exception Handling: Java provides a robust exception handling mechanism that allows programmers to catch and handle errors gracefully, preventing program crashes.
- Type Safety: Java is a strongly typed language, which means that variables must be declared with a specific data type. This helps to prevent type-related errors and makes code more reliable.

Security

Java has built-in security features that help to protect against malicious code:

- Bytecode Verification: The JVM verifies the bytecode of Java programs to ensure that it is safe to execute. This helps to prevent malicious code from being executed on a system.
- Automatic Garbage Collection: By managing memory automatically, Java reduces the risk of memory-related security vulnerabilities.
- Security APIs: Java provides a rich set of security APIs that can be used to implement various security measures, such as encryption, authentication, and authorization.

Other Important Features

- Multithreading: Java supports multithreading, allowing multiple threads of execution to run concurrently within a single process. This can improve performance and responsiveness.
- Portability: Java programs can be easily ported to different platforms thanks to the JVM. This reduces development and maintenance costs.
- Simplicity: Java has a relatively simple syntax and a large standard library, making it easy to learn and use.

These features collectively contribute to Java's popularity and widespread use in various domains, including enterprise applications, web development, Android app development, and scientific computing.

5. Role of public static void main(String[] args) (PSVM)

 Research Topic: Analyze the structure and purpose of the main method in Java.

o Questions to Ponder:

 What is the role of each keyword in public static void main(String[] args)?

 What would happen if one of these keywords (public, static, or void) were removed or altered? Experiment by modifying the main method and note down the errors.

 Why is the String[] args parameter used in the main method? What does it do, and what happens if you omit it?

The Role of public static void main(String[] args) in Java

Keyword Breakdown

- public: This keyword makes the main method accessible from anywhere within the class or package. It allows other classes to call the main method.
- static: This keyword allows the main method to be called without creating an instance of the Main class. This is essential because the JVM starts execution by calling the main method, and there is no instance of the Main class at that point.
- void: This keyword indicates that the main method does not return any value. The main method is typically used to execute the main logic of the application, and it doesn't need to return a value.
- main(String[] args): This is the name of the method. The String[] args parameter is used to pass command-line arguments to the Java application. When you run a Java program from the command line, you can provide arguments after the class name. These arguments are stored in the args array and can be accessed and processed within the main method.

Consequences of Modifying Keywords

- Removing public: If you remove the public keyword, the main method will only be accessible within the same class. This means it cannot be called from other classes.
- Removing static: If you remove the static keyword, the main method will require an instance of the Main class to be created before it can be called. This is not possible at the beginning of program execution, so the JVM will not be able to find a suitable entry point.
- Removing void: If you remove the void keyword, you must specify a return type for the main method. However, the JVM expects the main method to return void, so this will result in a compilation error.
- Altering main(String[] args): If you change the name of the main method or the parameter types, the JVM will not be able to recognize it as the entry point of the application. This will result in a compilation error.

The String[] args Parameter

The String[] args parameter in the main method is used to pass command-line arguments to the Java application. When you run a Java program from the command line, you can provide arguments after the class name. These arguments are stored in the args array and can be accessed and processed within the main method.

For example, if you run the following command:

Bash

java MyProgram argument1 argument2

The args array in the main method will contain the elements "argument1" and "argument2". You can then use these arguments within the main method to customize the behavior of your program.

If you omit the String[] args parameter, the main method will not be able to receive any command-line arguments. This may limit the flexibility and functionality of your application.

6. Can We Write Multiple main Methods?

 Research Topic: Experiment with multiple main methods in Java.

o Questions to Ponder:

 Can a class have more than one main method? What would happen if you tried to define multiple main methods in a single class?

 What happens if multiple classes in the same project have their own main methods? How does the Java compiler and JVM handle this situation?

 Investigate method overloading for the main method. Can you overload the main method with different parameters, and how does this affect program execution?

Multiple main Methods in Java

Can a class have more than one main method?

Yes, a Java class can have multiple main methods, but only one of them can be declared as public static void main(String[] args). This is because the Java Virtual Machine (JVM) uses this specific signature to identify the entry point of the application.

What happens if you try to define multiple main methods in a single class?

If you define multiple main methods in a single class, the compiler will generate an error message indicating that the main method is already defined. Only one main method with the correct signature can be present in a class.

What happens if multiple classes in the same project have their own main methods?

If multiple classes in the same project have their own main methods, you can choose which one to execute by specifying the class name when running the Java program from the command line. For example, if you have two classes named Class1 and Class2, you can run the main method in Class1 by executing:

Bash

java Class1

This will execute the main method in the Class1 class.

Investigate method overloading for the main method.

Yes, you can overload the main method with different parameters. However, only one of the overloaded main methods can have the signature public static void main(String[] args). This is the signature that the JVM uses to identify the entry point of the application.

Here's an example of overloading the main method:

Java

```
public class Main {
   public static void main(String[] args) {
      System.out.println("Main method with String[] args");
   }

   public static void main(int[] args) {
      System.out.println("Overloaded    main method with int[] args");
   }
}
```

In this example, the main method is overloaded with two different signatures. The first main method takes a String[] array as input, while the second main method takes an int[] array as input. Only the first main method can be executed directly from the command line. The second main method can only be called explicitly from within other methods within the class.

7. Naming Conventions in Java

 Research Topic: Investigate Java's naming conventions.

o Questions to Ponder:

 Why do some words in Java start with uppercase (e.g., Class names) while others are lowercase (e.g., variable names and method names)?

 What are the rules for naming variables, classes, and methods in Java, and why is following these conventions important?

 How do naming conventions improve code readability and maintainability,

especially in large projects?

Naming Conventions in Java

Case Sensitivity and Naming Conventions

Java is a case-sensitive language, meaning that variableName and variablename are considered different identifiers. This case sensitivity extends to all elements of Java code, including class names, variable names, and method names.

Naming Conventions:

- Class names: Start with an uppercase letter and use PascalCase (e.g., MyClassName).
- Variable names and method names: Start with a lowercase letter and use camelCase (e.g., myVariableName).
- Constants: Use all uppercase letters with underscores between words (e.g., MAX_VALUE).

Why Follow Naming Conventions?

Following naming conventions in Java is important for several reasons:

- Readability: Consistent naming conventions make code easier to read and understand. When you see a variable named customerName, you immediately know that it likely represents the name of a customer.
- Maintainability: Well-named code is easier to maintain and modify. When you need to find a specific variable or method, you can use a consistent naming convention to search for it more efficiently.
- Collaboration: Naming conventions can help improve collaboration among team members. When everyone follows the same naming conventions, it becomes easier to understand and contribute to the codebase.

Impact on Code Readability and Maintainability

By following Java's naming conventions, you can significantly improve the readability and maintainability of your code. This is especially important in large projects where many developers may be working on the same codebase. Consistent naming conventions make it easier for developers to understand the purpose of different code elements and navigate the codebase more efficiently.

In addition to improving readability and maintainability, following naming conventions can also help to prevent errors. For example, if you use a consistent naming convention for variables, it is less likely that you will accidentally use the wrong variable in your code.

8. Java Object Creation and Memory Management

 Research Topic: Understand Java’s approach to objects and memory.

o Questions to Ponder:

 Why are Java objects created on the heap, and what are the implications of this?

 How does Java manage memory, and what role does the garbage collector play?

 What are the differences between method overloading and method overriding in

Java?

 What is the role of classes and objects in Java? Explore how they support the

principles of object-oriented programming (OOP), such as encapsulation,

inheritance, and polymorphism

Java Object Creation and Memory Management

Object Creation on the Heap

In Java, objects are created on the heap, a region of memory that is dynamically allocated at runtime. This means that the size and location of objects on the heap are not determined until the program is executed.

Implications of Heap Allocation:

- Dynamic memory allocation: Objects can be created and destroyed as needed, providing flexibility and efficiency.
- Garbage collection: Java's automatic garbage collector handles memory management, freeing up memory that is no longer in use.
- Reference-based access: Objects are accessed through references, which are pointers to the object's location on the heap.

Garbage Collection

Java's garbage collector is a background process that automatically identifies and reclaims memory that is no longer in use. This helps to prevent memory leaks and improve the overall performance of Java applications.

The garbage collector uses algorithms to determine which objects are no longer reachable by the program. These objects are then marked for deletion and their memory is reclaimed for future use.

Method Overloading vs. Method Overriding

- Method Overloading: This occurs when a class has multiple methods with the same name but different parameters. The compiler determines which method to call based on the number and types of arguments passed to it.
- Method Overriding: This occurs when a subclass defines a method with the same name and signature as a method in its superclass. When an object of the subclass calls the method, the overridden version is executed.

Classes and Objects in Java

In Java, classes are blueprints for creating objects. Objects are instances of classes and represent real-world entities.

Classes and objects support the principles of object-oriented programming (OOP) in the following ways:

- Encapsulation: Classes encapsulate data and behavior, making it easier to manage and maintain code.
- Inheritance: Classes can inherit properties and methods from other classes, promoting code reuse and creating hierarchies of related classes.

- Polymorphism: Objects of different classes can be treated as if they were objects of a common superclass, allowing for flexible and extensible code.

9. Purpose of Access Modifiers in Java

 Research Topic: Explore the purpose of access modifiers in Java.

o Questions to Ponder:

 What is the purpose of access modifiers (e.g., public, private) in controlling access to classes, methods, and variables?

 How do access modifiers contribute to encapsulation, data protection, and security in object-oriented programming?

 How do access modifiers influence software design and maintenance?

 Consider potential challenges or limitations of automatic memory management.

Access Modifiers in Java

Purpose:

Access modifiers in Java control the visibility of classes, methods, and variables within a program. They determine which parts of the code can access these elements.

Types of Access Modifiers:

- public: Accessible from anywhere within the program.
- private: Accessible only within the same class.
- protected: Accessible within the same package or any subclass of the class.
- default: Accessible within the same package (no explicit modifier).

Impact on Encapsulation, Data Protection, and Security:

- Encapsulation: Access modifiers are essential for encapsulating data and behavior within a class. By using private modifiers for instance variables and methods, you can control access to the internal state of the object, preventing unauthorized modifications.
- Data Protection: Access modifiers help protect sensitive data from unauthorized access. By declaring data members as private, you can ensure that they are only accessible through public methods that provide controlled access.
- Security: Access modifiers can be used to implement security mechanisms. For example, you can use private to restrict access to critical methods or data, preventing unauthorized users from modifying or accessing sensitive information.

Impact on Software Design and Maintenance:

- Modularity: Access modifiers can be used to create modular and maintainable software by defining clear boundaries between different parts of the code.
- Code Reusability: By using protected access modifiers, you can create reusable code that can be inherited by other classes.
- Flexibility: Access modifiers allow you to control the level of abstraction and flexibility in your code. For example, you can make certain methods or variables public to allow external access, while keeping others private to maintain control over the internal implementation.

Challenges of Automatic Memory Management:

While automatic memory management (garbage collection) is a powerful feature of Java, it can also introduce potential challenges:

- Performance Overhead: The garbage collector can sometimes introduce performance overhead, especially in applications with high memory usage or frequent object creation and destruction.
- Memory Leaks: If objects are not properly referenced or if there are circular references, the garbage collector may not be able to reclaim their memory, leading to memory leaks.
- Determinism: The exact timing of garbage collection can be unpredictable, which can make it difficult to analyze and optimize the performance of some applications.

To mitigate these challenges, it is important to write code that is mindful of memory usage and avoids creating unnecessary objects. Additionally, tools and techniques can be used to analyze memory usage and identify potential memory leaks.

Day 3: 22 Aug 2024

for(init;condi; incre/decre){

}

int a = 10;

a>11;

a = a+1;

---

while

While(condition){

statements

statements

}

Operators

1. arithmetic + , - , / , * , %

2. logical &&, ||, !

3. relational < , >, <= , >= , == , !=

4. assignment = , += , -=, *= , /=, %=

```
int a =1;
a = a +10;
a+=10;
```

5. bitwise &, |, ^, << , >>

ex. 5 & 5

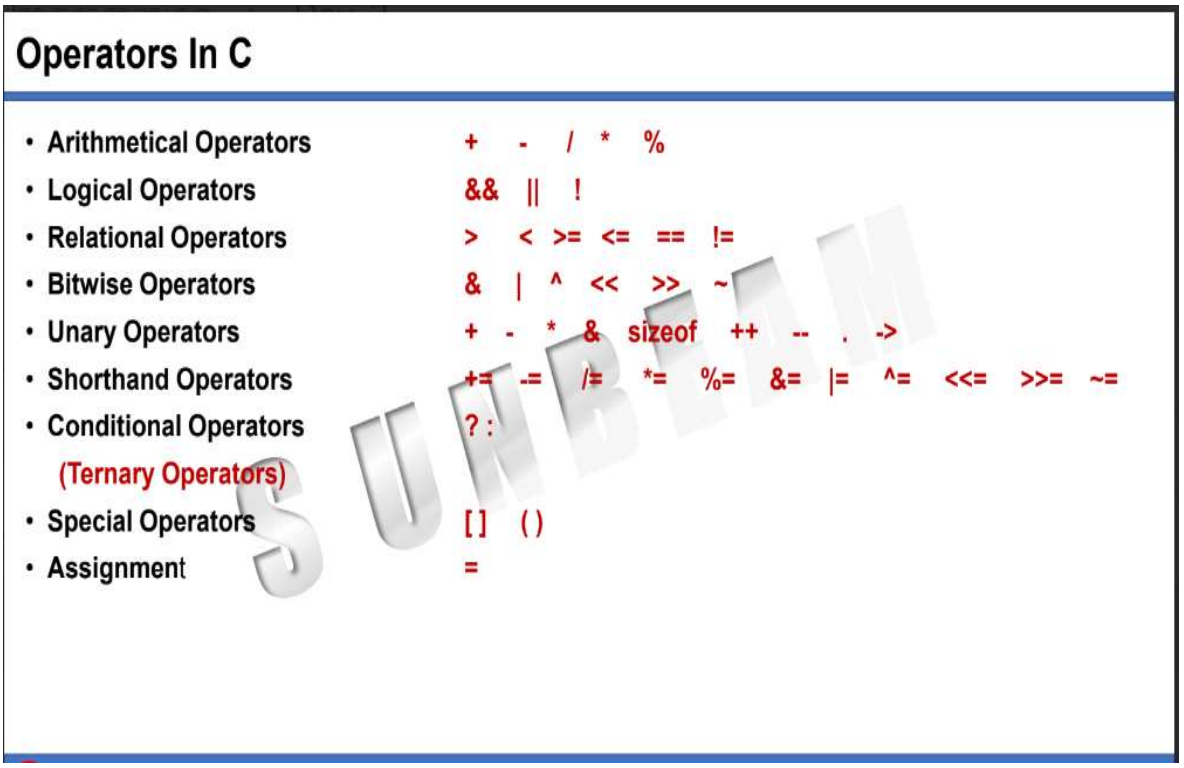
101 & 101

6. unary + , - , ++ , --

post

pre

7. ternary



Operator Precedence and Associativity

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (type) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Day-9

(1) 4+2-2+4 we can use in switch case ;

(2) class में main होता है। Main कैसे से सकते हैं

(3) ~~initiation~~ initiatialization; condition; increament (For loop)

(4) for loop में (; ;) 3 ही चलेगा। 4 आये तो error देगा।

(5) We can int a=10, b=20.
~~But~~ we can~~not~~ int i; int j;

(6) ~~int i~~; for(i=0, j=0; i<5; i++){
printf("—");
// print होगा 5 times.

(7) condition में i<5, j<5 नहीं चलेगा।
हम initialization & increament में दे सकते हैं। Two operations

But we use operators in condition
i<5 && j<5
1 or 0

(8) 7 नहीं condition (&&, ||) ऐसा चलेगा।

(8) for(int i=0, j=0; ; i++, j++){
}
// condition नहीं दी। infinite times चलेगा

C++ → int i // garbage value
Java → int i // कुछ भी value नहीं देगा
या तो 0 या garbage
error देगा।

(9) for (i=0, j=0; i<5 && j<5;) {
}
// ∞ ∞
infinite तक चलेगा।

(10) for (; ;) // Infinite loop

(11) int i=0;
for (Sysso("init"); i<5; i++) {
Sysso ("inside the loop")
}.

// init
inside loop 5 times print.

(12) Sysso print ("init");
Sysso printJn ("PRANJAL");
Sysso printIN ("Bala");

// init PRANJAL
Bala.

print (print होगा)
But second print भी उसी line पर print होगा।
printIN लगाया तो Next line पे

(13) for(int i=0; i<5; i++);
{ Sysso ("Bala"); // one time execute
}
Bcoz of semicolon.

(14)
```
int i;
for(i=0; i<5 && i<5)
for(int i=0; i<5; i++)
{
  for (int i=0; i<5; i++)
  {
    syso(" ");
  }
// compile time error
  i दो बार declare हुआ है।
```

(15)
```
for (int i=0; i<5; i++)
{
  for(int i=0; i<5; i++)
  {
    syso ("Bala");
  }
    syso ("Aruna");
```

// dry run

i	j	i<5	j<5	Bala
0	0	0<5	0<5	Bala
	1		1<5	Bala
	2		2<5	Bala
	3		3<5	Bala
	4		4<5	Bala
	5		5<5 X	
1	0	1<5	0<5	Bala
	1		1<5	Bala
	2		2<5	Bala
	3		3<5	Bala

1	i	i<5	j<5	Pf.
	4		4<5	Bala.
	5		5<5 (X)	
2	0	2<5	0<5	B
	1		1<5	B
	2		2<5	B
	3		3<5	B
	4		4<5	B
	5		5<5 (X)	
3	0	3<5	0<5	B
	1		1<5	B
	2		2<5	B
	3		3<5	B
	4		4<5	B
	5		5<5 (X)	B
4	0	4<5	0<5	B
	1		1<5	B
	2		1<5	B
	3		2<5	B
	4		3<5	B
	5		4<5	B
			5<5 (X)	
5	0	5<5 (X)		

ARONA Print: বের হ

(1)
```
for (int i=0; i<5 ; i++) {
    for (i=0 ; i<5 ; i++) {
        syso (" _ ");
    }
}
```
// 5 times _ _ _ _ _

// print होगा।

(2)
```
for (int i=0; i<5 ; i++)
    { for (int j=0 ; j<5 ; j++) {
        syso (" + ");
        syso (" - ")
    }
```
// + - + - + -

(3)
```
for (int i=0 ; i<5 ; i++)
{
  for (int j=0 ; i<5 ; i++)
    {
      syo print (" * ");
    }
    syso println (" + ");
  }
}
```
// * * * * * +
* * * * * +
* * * * * +
* * * * * +
* * * * * +

आज 23/8 →

(14) Patterns →

```
for (int i=0 ; i<5 ; i++) {
    for (int j=0 ; j<5 ; j++) {
        syso print ("+");
    }
    syso println ("+");
}
}
```

→ println उसके बाद + + + + + उसके बाद O/P

↑ उसके chars किया था

```
// +
   + +
   + + +
   + + + +
   + + + + +
```

```
+
+ -
+ + -
+ + + -
+ + + + -
```

i	j	i<5	j<5	
0	0	0<5	0<0 (x)	+
1	0	1<5	0<1	+ +
2	0	2<5	0<2	+
2	1	2<5	1<2	+ + +
3	0	3<5	0<3	+ + + +
3	1	3<5	1<3	
	2		2<3	
	3		3<3 (x)	
4	0	4<5	0<4	+ + + + +
	1		1<4	
	2		2<4	
	3		3<4	
5		5<5 (x)		

- ```
for (int i=0 ; i<5 ; i++)
{
  for(int j=0 ; j<=i ; j++){
    sys print ("+");
  }
    syso println ("+");
}
```

```
+
+ +
+ + +
+ + + +
+ + + + +
```

- ~~For int~~ sysoprint(i) →

```
0
1
2 2
3 3 3
4 4 4 4
```

- sysoprint(j) →

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
```

- ```
int count=0;
for (int i=0; i<5; i++)
{
for (int j=0; j<=i; j++)
  {
  sys print (++count + " ");
  }
  syso println (" ");
}
```

//

i	j	i<5	j<=i	
0	0	0<5	0<=0	1
	1		1<=0 (x)	
1	0	1<5	0<=1	2 3
	1		1<=1	
2	0	2<5	0<=2	4 5 6
	1		1<=2	
	2		2<=2	

Horizontal, Vertical, pyramid) company for.

Alphabet →

```
int count=65;
for(int i=0 ; i<5 ; i++)   // row
{
  for(int i=0 ; i<5 ; i++)   // colum
  {
    sysout.print((char)++count+" ");
  }
  sysout(" ");
}
```

// A B C D E F G H I Jk L M N O P Q

; char type cast करना पड़ता है

★ WHILE LOOP →

① While (cond) {
  stat
  State
}

② while(1) {   // CTE
  sysout(" ");
}

③ while (true) {   // infinite.
  sysout(" ")
}

IMP ④ while (false) {   // CTE
  sysout(" ")   unreachable statement
}

⑤ boolean b = false;
while(b){
syso println("inside while");
}

⑥ boolean b = false; // compile
while(b){ [print नहीं]
syso println("_");
}

⑦ int b = false // CTE.

⑧ int i=0;
while(i<5){ // 5 Times
syso(" ");
i++;
}

⑨ int i=0; // Inifinite times
while(i<5){ print होगा
syso(" ")
}
i++;

⑩ int i=0;
while(i<5)
syso(" ");
i--; // Inifinite.
}

0 1 2 3 4 continue

0 1 2 3 4

① boolean b=true; // one time execute
while(b){
syso(" ");
b=false;

② DO WHILE.

① do{
syso(" "); // " ".
}while(false);

② do{
syso(" "); // Infinite
}while(true);

③ while(1)/(0) → CTE.

④ do{
syso
i--; ← increament/decrement.
}while[false); ← semicolon

⑤ a>=0 (0 भी लेते हैं)

⑥ int a=0; // 0 0 00 (infinite)
while(a<10){
syso(a)}

⑦ int i=0;
while(i<10)
{ if(i==5){
continue;
}
syso(i++); // 0 1 2 3 4
5 6 7 8 9

⑧ while(i<10)
{ if(i==5)
{continue;
i++; → unreach able error.
}
syso(i++);

21 + 12 + 12
11 + 10 + 11 + 24 + 21 = 45

a = 12
b = 12
c = 5

classmate Date Page

29/8

① while (i<10) {
if (i==5) {
i++;
continue;
}
sysO(i++);

// 0 1 2 3 4 6 7 8 9

② → अंत print

③ Operators →

① a+=10 int a=1
a = a+10
= 1+10
a = 11

② sysO(5 & 5); // 5

③ int a = -10.
sysO(a) // -10.

int a = +10;
sysO(a) // 10.

④ assi increa → post a++
incre use → pre. ++a

⑤

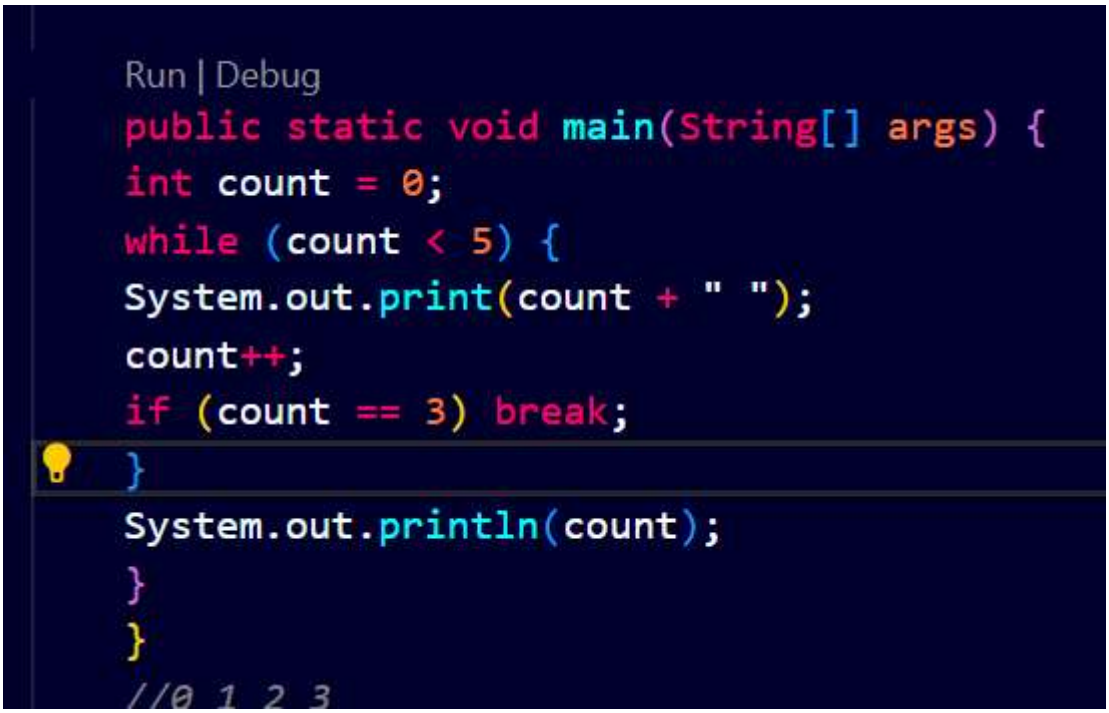
```java
public class Programs {
    Run | Debug
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 2; j++) {
        System.out.print(i + " " + j + " ");
        }
        System.out.println();
        }
        }
        }

// 1 1 1 2
// 2 1 2 2
// 3 1 3 2
```

\

```java
public class Programs {

Run | Debug
public static void main(String[] args) {
int total = 0;
for (int i = 5; i > 0; i--) {
total += i;
if (i == 3) continue;
total -= 1;
}
System.out.println(total);
}
}
// Guess the output of this loo 11
```



Snippet 4:

public class DoWhileLoop {

public static void main(String[] args) {

int i = 1;

do {

System.out.print(i + " ");

i++;

} while (i < 5);

System.out.println(i);

}

}

// Guess the output of this do-while loop

1 2 3 4

5

Snippet 5:

```
public class ConditionalLoopOutput {
public static void main(String[] args) {
int num = 1;
for (int I = 1; I <= 4; i++) {
if (I % 2 == 0) {
num += I;
} else {
num -= I;
}
}
System.out.println(num);
}
}
// Guess the output of this loop
```

3

Snippet 6:

```
public class IncrementDecrement {
public static void main(String[] args) {
int x = 5;
int y = ++x – x-- + --x + x++;
System.out.println(y);
}
}
// Guess the output of this code snippet
```

8

```
```

Snippet 7:

```
public class NestedIncrement {
public static void main(String[] args) {
int a = 10;
int b = 5;
int result = ++a * b-- - --a + b++;
System.out.println(result);
}
}
```

// Guess the output of this code snippet

Final Calculation:

- result = 11 * 5 – 10 + 4
- result = 55 – 10 + 4
- result = 45 + 4
- result = 49

LINEWISE KELELE

Snippet 8:

```
public class LoopIncrement {
public static void main(String[] args) {
int count = 0;
for (int I = 0; I < 4; i++) {
count += i++ - ++I;
}
System.out.println(count);
}
}
```

// Guess the output of this code snippet

-4

Day 4-23 Aug 2024

Day4

```
int a = 0;
for(a>10;a++){
SOP("ajkadh");
}

for(int i = 0 ; i<5 ; i++){
        if(true){
        break;
        }
        sop("safkhjkas");
}
```

Class & object

OOPS - object oriented prog.

c

c++

java

top-bottom

bottom-top

class

---

is collection objects....

its blueprint

will its own property and behavior

property - data members
behavior - members func/mthods

laptop-----
P - memory ,ModelName, Processor, OS, battery, ScreenResolution
M - WatchMoviw, clickPhoto, typeNotepad, Sound, Gaming

notebooks ----
P - name, noOfPages, type,quality
M - makingNotes, writing, drawing

---
method syntax

<non-access><Access-Specifiers><return type> nameOfMethod () { }

RT values
pritmitive DT
non-primitve DT

---

in java

local , global
instance var
static var

---
constructor -
---
same name as a classname
special method without RT.
used to initialse the instance variable.

types :

1. default
2. parameter
    a. zero parameter
    b. one parameter

```
class Laptop{
    int memory;
    String modelName;
    String processor;
    String os;
    float price;

    //clickphoto and typrnotepad and sound ye methods hai
    void clickPhoto(){
        System.out.println("inside ClickPhoto()");
    }


    String typeNotepad(){
        return "Return from typeNotepad";
    }


    void sound(){
        System.out.println("inside sound()");
    }

}



public class Demo{
    public static void main(String args[]){
        int a=10;  // primitive DT
        // user defined data type
        Laptop laptopObj = new Laptop();   // default,

    //Laptop laptopObj; //error ayega(memory chahiye).
    //classN  ref of obj =keyword

    //new is a keyword , used allocate memory
    // laptop();

    laptopObj.sound();
    laptopObj.typeNotepad();//kuch bhi show nhi hoga
```

```
    String s=laptopObj.typeNotepad();//he line nhi likhi to bhi chalega
    System.out.println(laptopObj.typeNotepad());//print honga idhar typenotepad mai jo hai vo
    //jisake classes banaye nahiye vo access nhai kar sakte agr kara toh error ayega

    }

}

```

```
class Laptop{
    int memory;
    String modelName;
    String processor;
    String os;
    float price;  // instance variable

    Laptop(){    // zero parameter
        memory=8;
        modelName = "HP";
        processor = "i5";
        os="windows";
        price=50000.0f;
    }

    Laptop(int a , String b){  // two parameter
        memory = a;
        modelName = b;
    }

    Laptop(int a, String b, String c, String d, float e){  // 5 parameter
        memory = a;
       b modelName =;
        processor =  c;
        os = d;
        price = e;

    }

    void clickPhoto(){
        System.out.println("inside ClickPhoto()");
    }

    String typeNotepad(){
        return "Return from typeNotepad()";
    }
```

```
    void sound(){
        System.out.println("inside sound()");
    }
}

public class Demo1{
    public static void main(String args[]){
        int a=10;  // primitive DT  // local var
        // Object are stored on Heap area

        Laptop laptopObj1 = new Laptop();
        Laptop laptopObj2 = new Laptop(4,"Dell","I7","Win", 60000.0f);
        Laptop laptopObj3 = new Laptop(16,"Asus");

        System.out.println(laptopObj1.memory);
        System.out.println(laptopObj1.modelName);
        System.out.println(laptopObj1.price);

        System.out.println(laptopObj2.memory+ " " + laptopObj2.modelName+" "+laptopObj2.price);
        System.out.println(laptopObj3.memory+ " " + laptopObj3.modelName+" "+ laptopObj3.price);




        //laptopObj1.clickPhoto();
        //laptopObj2.clickPhoto();




    }

}
```

```


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

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

```
            //System.out.println("inside for loop");
            break;
        }
    System.out.println("inside for loop");
    }

    System.out.println("outside for loop");
}

}
```

```
class Student{
    int id;
    String name;
    float marks;
    long mob;
    String address;
    static String instituteName= "CDAC Mumbai";

    Student(){}


    Student(int a, String b, float c, long d, String e){
        id = a;
        name = b;
        marks = c;
        mob = d;
        address = e;

    }

    float displayMarks(){

        return marks;
    }

    void calculatePercentage(){
            System.out.println("percentage");
    }

    void displayDetails(){
        System.out.println(id+ " " +name+ " "+address+ " "+mob+" "+marks);

    }
```

```
class Student1{
    int id;
    String name;
    float marks;
    long mob;
    String address;
    static String instituteName= "CDAC Mumbai";

    Student1(){}


    Student1(int a, String b, float c, long d, String e){
        id = a;
        name = b;
        marks = c;
        mob = d;
        address = e;

    }

    static void instituteName1(){
        System.out.println(instituteName);
        System.out.println(name);

    }

     float displayMarks(){
         System.out.println(instituteName+" PG_DAC AUG24");
        int grace = 1;
        marks+=grace;

        return marks;
    }

    void calculatePercentage(){
            System.out.println("percentage");
    }

    void displayDetails(){
        System.out.println(id+ " " +name+ " "+address+ " "+mob+" "+marks);
    }

}

class StudentDemo1{
```

```
    public static void main(String args[]){
        Student1 obj1 = new Student1();
        //System.out.println(obj1.displayMarks());
        obj1.instituteName1();

        Student1 obj2 = new Student1(1,"abc",70.0f,9L,"mumbai");
        //System.out.println(obj2.displayMarks());
        obj2.instituteName1();



    }


}
```

23 Aug 2024

① For (int i=0; i<5; i++){
    if (true){
        break;
    }
    sysо ("inside the loop");
}
}

// nothing pt.

② for (i=0 ; i<5; i++) {
    if (true) {
        break;
    }
    sysо (" Bala");
}
sysо ("Pranjal");
}

// pranial.

③ for (int i=0 ; i<5 ; i++) {
    if (false){
        break;
    }
    sysо (" bala");
}
    sysо (" pranjal");
}
}

// bala
bala
bala
bala Pranjal.

④
```
int i=1;
while (i<=3){
  int j=1;
    while (j<=i) {
      System.out.print("*");
      j++;
    }
  System.out.println();
  i++;
}
}
```

```
// i  i<=3  j  j<=i  j++  i++
   1  1<=3  1  1<=1  2    2
            2  2<=1 (X)
```

so on.

⑤ C → Top-to-bottom approach
Java → bottom to up approach.

⑥ C, CPP, Java (OOPS)

⑦ class → coll^{n} of objects.
- Its blueprint.
- will its own property & behaviour.
- property → data mem
  behaviour → member fun/ methods

⑧ Laptop → memory, Model name, processor, os, battery, screensolution (properties)

typeModel
↳ capital

Laptop behaviours
watchmovies, clickphoto
typepad, sound, gaming

② Notebooks →
P → Name, no-of pages, type, quality
B → Makingnotes, writting, drawing

③ Method syntax →
~~<return type>~~
<non-access> <Access> specifi
<return type> nameofMethod()
{
}

class
⑪ ~~class~~ Laptop{ Demo. Java,
PHOTO int memory;
Demo, String modelName;
Java String Preprocessor;

- ~~Laptop~~
typeNotepad() → Method,
typenotepad → Variable.

⑫ Demo1.java

- In Java
  local, global,
  instance variable.
- int a=10; //local.
  at 5 variables //insitance variable
- objects are stored in Heap area.
- instance variable ~~को~~ access ~~करना~~ है।
  (LaptopObj.memory);
  (laptopObj.modeName);
- default → int → 0
  datatypes string → Null
  float → 0.0
  short →
  byte →
  char → NULL.

- StudentDemo.java

(2) Photo → StudentDemo.java (1)

- declare किया।
- marks a, b, c, d, e Variable कि
- void calculatepercentage() {
  syso ("perce"); // access किया।
  }
- void displayMarks() {
  return Marks(); } // access किया।
  }

- static string InstituteName;
- (Student. InstituteName) syso.
  // print करना पड़ेगा (access)

- static string InstituteName = "cdac mum";
  Student studentobj
  = new student()
  syso (studentobj. Name).

—— x ——

PROGRAMS-

class Programs{

// 1. Sum of the first 50 natural numbers:

```
//    public static void main(String[] args) {
//        int sum = 0;
//        for (int i = 1; i <= 50; i++) {
//            sum += i;
//        }
//        System.out.println("Sum of the first 50 natural numbers: " + sum);
//    }
```

```
//  2. Compute the factorial of 10:
//    public static void main(String[] args) {
//        int number = 10;
//        int factorial = 1;
//        for (int i = 1; i <= number; i++) {
//            factorial *= i;
//        }
//        System.out.println("Factorial of 10: " + factorial);
//    }
```

```
//  3. Print all multiples of 7 between 1 and 100:

//    public static void main(String[] args) {
//        for (int i = 7; i <= 100; i += 7) {
//            System.out.println(i);
//        }
//    }
```

```
//  4. Reverse the digits of the number 1234:

//    public static void main(String[] args) {
//        int number = 1234;
//        int reversed = 0;
```

```
//        while (number != 0) {
//            int digit = number % 10;
//            reversed = reversed * 10 + digit;
//            number /= 10;
//        }
//        System.out.println("Reversed number: " + reversed);
//    }


// 5. Print the Fibonacci sequence up to the number 21:

//    public static void main(String[] args) {
//        int a = 0, b = 1;
//        System.out.print("Fibonacci sequence up to 21: " + a + " " + b);
//        while (b < 21) {
//            int next = a + b;
//            a = b;
//            b = next;
//            if (b <= 21) {
//                System.out.print(" " + b);
//            }
//        }
//        System.out.println();
//    }


// 6. Find and print the first 5 prime numbers:

//    public static void main(String[] args) {
//        int count = 0, number = 2;
//        while (count < 5) {
//            boolean isPrime = true;
//            for (int i = 2; i <= Math.sqrt(number); i++) {
//                if (number % i == 0) {
//                    isPrime = false;
//                    break;
//                }
//            }
//            if (isPrime) {
```

```
//             System.out.print(number + " ");
//             count++;
//          }
//          number++;
//       }
//       System.out.println();
//    }


// 7. Calculate the sum of the digits of the number 9876:

//    public static void main(String[] args) {
//       int number = 9876;
//       int sum = 0;
//       while (number != 0) {
//          sum += number % 10;
//          number /= 10;
//       }
//       System.out.println("Sum of the digits: " + sum);
//    }


// 8. Count down from 10 to 0:

//    public static void main(String[] args) {
//       for (int i = 10; i >= 0; i--) {
//          System.out.println(i);
//       }
//    }


// 9. Find and print the largest digit in the number 4825:

//    public static void main(String[] args) {
//       int number = 4825;
//       int largest = 0;
//       while (number != 0) {
//          int digit = number % 10;
//          if (digit > largest) {
```

```
//             largest = digit;
//         }
//         number /= 10;
//     }
//     System.out.println("Largest digit: " + largest);
// }


// 10. Print all even numbers between 1 and 50:

// public static void main(String[] args) {
//     for (int i = 2; i <= 50; i += 2) {
//         System.out.println(i);
//     }
// }


// 11. Demonstrate pre-increment and post-decrement operators:

// public static void main(String[] args) {
//     int x = 5;
//     int y = ++x - x-- + --x + x++;
//     System.out.println("Result: " + y);
// }


// 12. Draw the pattern:

// *****
// *****
// *****
// *****
// *****

// public static void main(String[] args) {
//     for (int i = 0; i < 5; i++) {
//         for (int j = 0; j < 5; j++) {
//             System.out.print("*");
//         }
```

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


//  13. Print the pattern:

// 2*2
// 3*3*3
// 4*4*4*4
// 5*5*5*5*5
// 5*5*5*5*5
// 4*4*4*4
// 3*3*3
// 2*2
// 1

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


//  14. Print the pattern:

// *
// **
// ***
```

```
// ****
// *****
// ******
// *******
// *********

//    public static void main(String[] args) {
//        for (int i = 1; i <= 8; i++) {
//            for (int j = 1; j <= i; j++) {
//                System.out.print("*");
//            }
//            System.out.println();
//        }



// 15. Print the pattern:

// *
// **
// ***
// ****
// *****

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


//  16. Print the pattern:

// *
// ***
// *****
```

```
// *******
// *********

//    public static void main(String[] args) {
//        int[] counts = {1, 3, 5, 7, 9};
//        for (int count : counts) {
//            for (int j = 1; j <= count; j++) {
//                System.out.print("*");
//            }
//            System.out.println();
//        }
//    }



// 17. Print the pattern:

// *****
// ****
// ***
// **
// *

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



//  18. Print the pattern:

// *
// ***
// *****
// *******
```

```
// *****
// ***
// *

//    public static void main(String[] args) {
//        int[] counts = {1, 3, 5, 7, 5, 3, 1};
//        for (int count : counts) {
//            for (int j = 1; j <= count; j++) {
//                System.out.print("*");
//            }
//            System.out.println();
//        }



//  19. Print the pattern:

// 1
// 1*2
// 1*2*3
// 1*2*3*4
// 1*2*3*4*5

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



// 20. Print the pattern:

// 5*4
// 5*4*3
// 5*4*3*2
```

```
// 5*4*3*2*1

//    public static void main(String[] args) {
//        for (int i = 5; i >= 1; i--) {
//            for (int j = 5; j >= i; j--) {
//                System.out.print(j + (j > i ? "*" : ""));
//            }
//            System.out.println();
//        }
//    }




//  21. Print the pattern:

// 1
// 1*3
// 1*3*5
// 1*3*5*7

//    public static void main(String[] args) {
//        int num = 1;
//        for (int i = 1; i <= 5; i++) {
//            for (int j = 1; j <= i; j++) {
//                System.out.print(num + (j < i ? "*" : ""));
//                num += 2;
//            }
//            num = 1;
//            System.out.println();
//        }
//    }


// 22. Print the pattern:

// *********
// *******
```

```
// *****
// ***
// *
// ***
// *****
// *******
// *********

//    public static void main(String[] args) {
//        int[] counts = {9, 7, 5, 3, 1};
//        for (int count : counts) {
//            for (int j = 1; j <= count; j++) {
//                System.out.print("*");
//            }
//            System.out.println();
//        }
//        for (int count : new int[]{3, 5, 7, 9}) {
//            for (int j = 1; j <= count; j++) {
//                System.out.print("*");
//            }
//            System.out.println();
//        }



//  23. Print the pattern:

// 11111
// 22222
// 33333
// 44444
// 55555

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

```
//        }
//    }



//  24. Print the pattern:

// 22
// 333
// 4444
// 55555

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



//  25. Print the pattern:

// 1
// 12
// 123
// 1234
// 12345

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

```
// 26. Print the pattern:

// 1
// 2 3
// 4 5 6
// 7 8 9 10
// 11 12 13 14 15

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

}
```

Day 5-24 aug 2024

Day 5

---

ex.

book

library

```
import java.util.Scanner;

class ArrayDemo{

// array is used to store similar type of data.

// datatype nameOfArray[] = {};
// dt name[] = new dt[];
// dt[] n = new dt[];

 //ex. int a[] = {1,2,3,4};
   //int a[] = {1.0,1,2 3}  error

// [] == subscript operator

public static void main(String arg[]){

   Scanner sc = new Scanner(System.in);

   /*int arr[] = {1,2,3};
   System.out.println(arr);  // hashcode
   System.out.println(arr[0]);   // 1
   System.out.println(arr[1]);   // 2
   System.out.println(arr[2]);   // 3
   */


   /*float arr1[] = {1.0f,2.0f,3.0f};

   System.out.println(arr1);  //hashcode
   System.out.println(arr1[0]+ " " + arr1[1]+" "+arr1[2]);*/

   int arr[] = new int[5];
   /*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]); */

    /*float arr1[] = new float[5];
    System.out.println(arr1[0]);
    System.out.println(arr1[1]);
    System.out.println(arr1[2]);
    System.out.println(arr1[3]);
    System.out.println(arr1[4]); */

    // for input
    for(int i = 0; i<=5; i++){
        arr[i] = 10;
    }

    // for output
    for(int i = 0; i<5; i++){
        System.out.println(arr[i]);
    }



}
```

```
import java.util.Scanner;

class ArrayDemo1{
```

```
public static void main(String arg[]){

   Scanner sc = new Scanner(System.in);
   /*sc.nextInt();
     sc.nextFloat();
   sc.next();
   sc.nextLine();
   sc.nextLong();
   char c = sc.next().charAt(0);  // pls enter 1st value
   */

   int arr[] = new int[5];

   // for input
   for(int i = 0; i<5; i++){
      System.out.println("Pls enter "+ (i+1) +"st value");
      arr[i] = sc.nextInt();
   }

   // for output
   for(int i = 0; i<5; i++){
      System.out.println(arr[i]);
   }




}




}
```

```
import java.util.Scanner;
class NestedSwitchDemo{
   public static void main(String arg[]){
   /*Scanner sc = new Scanner(System.in);

   char c = sc.next().charAt(0);

   switch(c){

   case '+':
      System.out.println("pls enter how many numbers u want to add");
      int i = sc.nextInt();
      switch(i){
            case 1:
               System.out.println("one number cannot be added");
               break;
            case 2:
               int a1 = sc.nextInt();
               int a2 = sc.nextInt();
               int result = a1 + a2;
               System.out.println("Result "+result);
               break;
            default:
               System.out.println("we can add only 2 numbers");
               break;
         }

      break;
   case '-' :
      System.out.println("substrat");
      break;
   default :
      System.out.println("Invlaid input");
      break;

   }
   */
```

```
//ternary
//  condition ?(true) :(flase) ;

   int i = 10 ;

   /*(i<10)?(System.out.print("i is less 10");
      System.out.print("asfhakjds");  )
       :((i>10)? (System.out.print("i is greater 10");)
            :(System.out.print("i is eual 10");)   );*/

   /*(i<10)?(System.out.print("i is less 10");
      System.out.print("asfhakjds");) :((i>10)?(System.out.print("i is greater
10");:(System.out.print("i is eual 10"));));*/

   }


}
```

```
class Person1
{
   String name;
   String address;



  Person1(){   }

   Person1(String a, String b){
   name = a;
   address = b;
```

```
    }

    void showData()
    {
       System.out.println("Name of Person : " + name);
       System.out.println("Address of person : " + address);

    }

    String showData(String a){
    System.out.println("parameter value : " + a);
     System.out.println("Name of Person : " + name);

    return a+" "+name;
    }

// return statement shld always return only one value.


    public static void main(String arg[])
     {
       Person1 person1 =  new Person1("Ram","MH");
       person1.showData();
    person1.showData("abc");

       }
}
```

```
class Person
{
    String name;
    String address;
    static String COUNTRYNAME;

    {
```

```
    name= "ABC";
    address= "XYZ";
  }

  static
  {
    COUNTRYNAME = "India";
  }

 Person(){   }

  Person(String a, String b){
  name = a;
  address = b;
  }

  void showData()
  {
    System.out.println("Name of Person : " + name);
    System.out.println("Address of person : " + address);
    System.out.println("Country of person : "+  COUNTRYNAME);
  }

  void showData(){
   System.out.println("Name of Person : " + name);
  }

}

class PersonDemo{
public static void main(String arg[])
  {
    Person person1 =  new Person("Ram","MH");
    person1.showData();

    Person person2 =  new Person("Shyam","MH");
    person2.showData();

   Person person3 =  new Person();
```

```
        person3.showData();
    }
}


class PersonOverloading
{
    String name;
    String address;



   PersonOverloading(){ }

    PersonOverloading(String a, String b){
    name = a;
    address = b;
    }

    void showData()
    {
        System.out.println("Name of Person : " + name);
        System.out.println("Address of person : " + address);

    }

    String showData(String a){
    System.out.println("parameter value : " + a);
     System.out.println("Name of Person : " + name);

    return a;
    }

    /*String showData(String a){
    System.out.println("parameter value : " + a);
     System.out.println("Name of Person : " + name);

    return a;
    }*/
```

```
    // method showData(String) is already defined


    public static void main(String arg[])
     {
       PersonOverloading person1 =  new PersonOverloading("Ram","MH");
       //person1.showData();
    person1.showData("abc");

     PersonOverloading person2 =  new PersonOverloading();
      // person2.showData();
    person2.showData("abc");
       }
}
```

```
import java.util.Scanner;

class ScannerDemo{
public static void main(String arg[]){

   Scanner sc = new Scanner(System.in);
   /*sc.nextInt();
     sc.nextFloat();
   sc.next();
   sc.nextLine();
   sc.nextLong();
   char c = sc.next().charAt(0);  // pls enter 1st value
   */

   //String s1 = sc.nextLine();
   //System.out.println(s1);

   //String s = sc.next();// it will take only first word in input string, and ignores after space
//  System.out.println(s);

   //sc.next();
```

```
   char c = sc.next().charAt(5);
   System.out.println(c);

}
}
```

```
class Student{

int id;
String name;
String address;
float marks;
static String iName = "CDAC Mumbai";
static int year = "2024";
static string batch = "Aug";


Student(){}

Student (int id){
this.id = id;
}

Student(int a , String b, String c, float d){
id = a;
name = b;
address =  c;
marks = d;
}

void display(){
System.out.println("Student Id " + id);

System.out.println("Institue " + iName);
```

```
}

static void print(){
System.out.println("Student Id " + id);

System.out.println("Institue " + iName);
}


}

class StudentDemo{

public static void main(String[] args){
Student obj1 = new Student();
Student obj2 = new Student(1);
obj1.display();
obj2.display();


}

}

/*
class - blueprint

class -  inst. var
   static var  -- class
   methods
   constr.
   inst block
   static block

object - instance  --
*/
```

```
class Student1{

int id;
String name;
String address;
float marks;
static String iName;

   {
      this.id = 1;
      System.out.println("Inside INIT");
   }

   static{

      System.out.println("Inside Static");
   }

   Student1(){
      this.id = 2;
      iName = "CDAC";
      System.out.println("Inside Constr.");
   }

   static void display(){
      System.out.println("Inside Display()");
   }

   void display(int i){
   System.out.println(i);

   }

}
class StudentDemo1{
   public static void main(String[] args){

   Student1 obj1 = new Student1();
   /*Student1 obj2 = new Student1();
```

```
    Student1 obj3 = new Student1();

    System.out.println(obj1.id);
    System.out.println(obj2.id);
    System.out.println(obj3.id);
    System.out.println(obj1.iName);
    System.out.println(obj2.iName);
    System.out.println(obj3.iName);
    */

    Student1.display();  // static
    //Student1.display(1); // non-static
    obj1.display(1);



    }
}
```