**Name: Chinmay Gaikwad\_KH**

**CDAC Mumbai Lab Assignment**

# Section 1: Error-Driven Learning in Java

**Objective:** This assignment focuses on understanding and fixing common errors encountered in Java programming. By analyzing and correcting the provided code snippets, you will develop a deeper understanding of Java's syntax, data types, and control structures.

**Instructions:**

1. **Identify the Errors:** Review each code snippet to identify the errors or issues present.
2. **Explain the Error:** Write a brief explanation of the error and its cause.
3. **Fix the Error:** Modify the code to correct the errors. Ensure that the code compiles and runs as expected.
4. **Submit Your Work:** Provide the corrected code along with explanations for each snippet.

#### Snippet 1:

public class Main {

public void main(String[] args) { System.out.println("Hello, World!");

}

}

### What error do you get when running this code?

### error: 'main' method is not declared 'public static'

### Solution of Snippet 1:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World!");

}

}

#### Snippet 2:

public class Main {

static void main(String[] args) { System.out.println("Hello, World!");

}

}

### What happens when you compile and run this code?

### -error: class Main is public, should be declared in a file named Main.java

### Solution of Snippet 2:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World!");

}

}

#### Snippet 3:

public class Main {

public static int main(String[] args) { System.out.println("Hello, World!"); return 0;

}

### What error do you encounter? Why is void used in the main method?

* **error:** The main method should have a return type of ‘void’, not ‘int’

### Solution of Snippet 3:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World!"); return 0;

#### Snippet 4:

public class Main {

A close up of a logo

Description automatically generatedpublic static void main() { System.out.println("Hello, World!");

}

}

### What happens when you compile and run this code? Why is String[] args needed?

### error: The main method must have the exact signature: public static void main(String[] args). This specific signature is required by the JVM as the entry point of a Java application. The JVM looks for this method to start execution.

### Need: The String[] args parameter allows the JVM to pass command-line arguments to your program. This is useful for providing inputs or configuration options when starting the application.

### Solution of Snippet 4:

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

#### Snippet 5:

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");

}

}

### Can you have multiple main methods? What do you observe?

### Yes, you can have multiple main methods in a class through method overloading.

### The JVM will only recognize and execute the main(String[] args) method as the entry point for running the program.

### Overloaded main methods, like main(int[] args), can be defined and used, but they do not affect the program’s startup and are not executed by default when running the program.

#### Snippet 6:

public class Main {

public static void main(String[] args) { int x = y + 10; System.out.println(x);

}

}

### What error occurs? Why must variables be declared?

### error: The code will result in a compilation error: error: cannot find symbol.

### Here the variable y is used in the expression int x = y + 10;, but it hasn't been declared or initialized anywhere in the code. In Java, all variables must be declared (and initialized, if necessary) before they are used in expressions or operations.

### Solution of Snippet 6:

### public class Main {

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

### int y = 5; // Declare and initialize 'y'

### int x = y + 10;

### System.out.println(x);

### }

### }

#### Snippet 7:

public class Main {

public static void main(String[] args) { int x = "Hello"; System.out.println(x);

}

}

### What compilation error do you see? Why does Java enforce type safety?

### error: incompatible types: String cannot be converted to int.

### The compilation error arises because of an attempt to assign a String to an int variable, which is not allowed in Java due to type incompatibility.

### Java enforces type safety to prevent runtime errors, ensure code clarity, allow for compiler optimizations, and enhance security.

### Solution of Snippet 7:

### public class Main {

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

### String x = "Hello";

### System.out.println(x);

### }

### }

#### Snippet 8:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World!"

}

}

### What syntax errors are present? How do they affect compilation?

### Syntax errors: Missing closing Parenthesis ‘)’ and Semicolon ‘;’

### These errors prevent the code from compiling, as the compiler requires proper syntax for method calls and statement termination in Java

### Solution of Snippet 8:

### public class Main {

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

### System.out.println("Hello, World!");

### }

### }

#### Snippet 9:

public class Main {

public static void main(String[] args) { int class = 10; System.out.println(class);

}

}

### What error occurs? Why can't reserved keywords be used as identifiers?

### error cause:- The original code fails to compile because class is a reserved keyword in Java, and r reserved keywords cannot be used as identifiers.

### - Reserved keywords are integral to the language's syntax and allowing them as identifiers would create ambiguity and reduce the clarity and maintainability of the code.

### Solution of Snippet 9:

### public class Main {

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

### int number = 10;

### System.out.println(number);

### }

### }

#### Snippet 10:

public class Main { public void display() {

System.out.println("No parameters");

}

public void display(int num) { System.out.println("With parameter: " + num);

}

public static void main(String[] args) { display();

display(5);

}

}

### What happens when you compile and run this code? Is method overloading allowed?

### Error: Compilation Error: The code will result in a compilation error due to the way the display() methods are called in the main method.

### Solution of Snippet 10:

### public class Main {

### public void display() {

### System.out.println("No parameters");

### }

### 

### public void display(int num) {

### System.out.println("With parameter: " + num);

### }

### 

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

### Main obj = new Main(); // Create an instance of Main class

### obj.display(); // Call the instance method display() with no parameters

### obj.display(5); // Call the overloaded method display(int num) with an integer parameter

### }

#### Snippet 11:

public class Main {

public static void main(String[] args) { int[] arr = {1, 2, 3}; System.out.println(arr[5]);

}

}

### What runtime exception do you encounter? Why does it occur?

### error: When you run this code, you will encounter a java.lang.ArrayIndexOutOfBoundsException.

### - The runtime exception ArrayIndexOutOfBoundsException occurs because the code attempts to access an array index (5) that is out of bounds for the given array (arr with valid indices 0, 1, and 2).

### - Java enforces array bounds checking to prevent accessing invalid memory locations, which could lead to undefined behaviour.

### Solution of Snippet 11:

### public class Main {

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

### int[] arr = {1, 2, 3};

### System.out.println(arr[2]); // Accessing a valid index

### }

### }

#### Snippet 12:

public class Main {

public static void main(String[] args) { while (true) {

System.out.println("Infinite Loop");

}

}

}

### What happens when you run this code? How can you avoid infinite loops?

### effect: Running the given code results in an infinite loop because the condition while (true) never becomes false

### In the below example, the loop runs only 5 times before count < 5 becomes false, and the loop terminates.

### Solution of Snippet 12:

### public class Main {

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

### int count = 0;

### while (count < 5) {

### System.out.println("Loop iteration: " + count);

### count++;

### }

### }

### }

#### Snippet 13:

public class Main {

public static void main(String[] args) { String str = null; System.out.println(str.length());

}

}

### What exception is thrown? Why does it occur?

### Exception thrown: java.lang.NullPointerException is thrown when you run this code.

* In this code, the variable str is declared and explicitly set to null.
* The line System.out.println(str.length()); attempts to call the length() method on the str object.
* However, since str is null and does not reference any actual String object, there is no object to call the length() method on.
* This results in the NullPointerException being thrown at runtime

### Solution of Snippet 13:

### public class Main {

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

### String str = null;

### if (str != null) {

### System.out.println(str.length());

### } else {

### System.out.println("String is null, cannot determine length.");

### }

### }

### }

#### Snippet 14:

public class Main {

public static void main(String[] args) { double num = "Hello"; System.out.println(num);

}

}

### What compilation error occurs? Why does Java enforce data type constraints?

### error: incompatible types: String cannot be converted to double

### double num = "Hello";

* Java enforces data type constraints to ensure type safety, allowing the compiler to catch type-related errors early and making the code more predictable and easier to maintain.

### Solution of Snippet 14:

### public class Main {

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

### double num = 10.5; // Assigning a double value

### System.out.println(num);

### }

### }

### ^

#### Snippet 15:

public class Main {

public static void main(String[] args) { int num1 = 10;

double num2 = 5.5;

int result = num1 + num2; System.out.println(result);

}

}

### What error occurs when compiling this code? How should you handle different data types in operations?

- **error**: incompatible types: possible lossy conversion from double to int int result = num1 + num2;

- The code will not compile because of an attempt to assign a double result to an int variable, which is not allowed without explicit casting.

-To handle different data types in operations, use explicit type casting if necessary, or choose a variable type that matches the result of the operation to avoid loss of data and ensure correct program behaviour

### Solution of Snippet 15:

public class Main {

public static void main(String[] args) {

int num1 = 10;

double num2 = 5.5;

int result = (int) (num1 + num2); // Explicitly cast the result to int

System.out.println(result);

}

}

#### Snippet 16:

public class Main {

public static void main(String[] args) { int num = 10;

double result = num / 4; System.out.println(result);

}

}

### What is the result of this operation? Is the output what you expected?

### The resulut of this operation is 2.0

### The output is 2.0, which might not be what you expected if you anticipated the operation to consider the fractional part (like 2.5). To achieve 2.5, you would need to make sure that one of the operands is a double or float

### Solution of Snippet 16:

public class Main {

public static void main(String[] args) { int num = 10;

double result = (double)num / 4; System.out.println(result);

}

A close up of a logo

Description automatically generated}

### This approach will yield 2.5 as the output.

#### Snippet 17:

public class Main {

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

int b = 5;

int result = a \*\* b; System.out.println(result);

}

}

### What compilation error occurs? Why is the \*\* operator not valid in Java?

### Error: illegal start of expression

*  **Invalid Operator (\*\*)**: In Java, the \*\* operator is not recognized or supported for exponentiation (raising a number to a power). Java does not have a built-in operator for exponentiation like some other languages (e.g., Python). The \*\* operator is not valid in Java, which is why the compiler throws an error.
*  **Exponentiation in Java**: To perform exponentiation in Java, you need to use the Math.pow() method. This method takes two arguments, the base and the exponent, and returns the result as a double.

### Solution of Snippet 17:

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

### int a = 10;

### int b = 5;

### double result = Math.pow(a, b); // Use Math.pow for exponentiation

### System.out.println(result);

### }

### int result = (int) Math.pow(a, b);

### System.out.println(result);

#### Snippet 18:

public class Main {

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

int b = 5;

int result = a + b \* 2; System.out.println(result);

}

}

### What is the output of this code? How does operator precedence affect the result?

### The result of the expression a + b \* 2 is 20, which is what the code will output.

### Operator Precedence: In Java (and many other programming languages), operators have a specific order of precedence, which dictates the order in which operations are performed.

### Multiplication (\*) Before Addition (+): Multiplication has higher precedence than addition. Therefore, in the expression a + b \* 2, the multiplication b \* 2 is performed first, before the addition.

### Step-by-Step Evaluation:

### b \* 2 is evaluated first: 5 \* 2 = 10.

### Then, a + 10 is evaluated: 10 + 10 = 20.

#### Snippet 19:

public class Main {

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

int b = 0;

int result = a / b; System.out.println(result);

}

}

### What runtime exception is thrown? Why does division by zero cause an issue in Java?

Exception in thread "main" java.lang.ArithmeticException: / by zero

Issue for cause:

**Mathematical Reason**: In mathematics, division by zero is undefined because there is no number that, when multiplied by zero, gives a non-zero number. This is why any attempt to divide by zero leads to an error.

**Java's Handling**: Java throws an ArithmeticException at runtime to prevent the program from continuing with an undefined or invalid operation

### Solution of Snippet 19: To avoid this exception, you should always check if the divisor is zero before performing the division.

### if (b != 0) {

### int result = a / b;

### System.out.println(result);

### } else {

### System.out.println("Cannot divide by zero.");

### }

#### Snippet 20:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World")

}

}

### What syntax error occurs? How does the missing semicolon affect compilation?

### Explanation:

*  **Missing Semicolon**: In Java, every statement must end with a semicolon (;). The semicolon acts as a statement terminator, telling the compiler where one statement ends and the next begins.
*  **Effect on Compilation**: Without the semicolon, the Java compiler does not recognize the end of the statement, leading to a syntax error. The compiler expects a semicolon at the end of the System.out.println("Hello, World") statement. Because of this error, the program will not compile, and you'll need to fix the code before it can run.

#### Snippet 21:

public class Main {

public static void main(String[] args) { System.out.println("Hello, World!");

// Missing closing brace here

}

### What does the compiler say about mismatched braces?

### The compiler will typically produce an error like:

### error: reached end of file while parsing.

#### Snippet 22:

public class Main {

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

System.out.println("Message");

}

}

}

### What syntax error occurs? Can a method be declared inside another method?

### error: illegal start of expression

### static void displayMessage() {

*  **Method Declaration Rules**: In Java, methods cannot be declared inside other methods. All methods must be declared at the class level, directly inside the class body, but not inside other methods or constructors.
*  **Why This is an Error**: The main method is trying to declare a new method (displayMessage) within its body, which violates the rules of method declaration in Java. The keyword static and the method signature are not allowed within the scope of another method.

### Solution of Snippet 22:

### public class Main {

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

### displayMessage(); // Call the method

### }

### static void displayMessage() {

### System.out.println("Message");

### }

### }

#### Snippet 23:

public class Confusion {

public static void main(String[] args) { int value = 2;

switch(value) { case 1:

System.out.println("Value is 1");

case 2:

System.out.println("Value is 2");

case 3:

System.out.println("Value is 3"); default:

System.out.println("Default case");

}

}

}

•**Error to Investigate**: Why does the default case print after "Value is 2"? How can you prevent the program from executing the default case?

* **Explanation:**
* **Switch Case Fall-Through**: In Java, once a matching case is found in a switch statement, all the subsequent cases are executed until a break statement or the end of the switch block is encountered. This is known as "fall-through."
* **Why Default Case Prints**: Since there is no break statement after case 2, the program continues to execute case 3 and then the default case, printing all of them.
* **How to Prevent Fall-Through:**
* To prevent the program from executing subsequent cases after a match, you should add break statements at the end of each case:

**Solution of Snippet** 23**:**

public class Confusion {

public static void main(String[] args) {

int value = 2;

switch(value) {

case 1:

System.out.println("Value is 1");

break;

case 2:

System.out.println("Value is 2");

break;

case 3:

System.out.println("Value is 3");

break;

default:

System.out.println("Default case");

break;

}

}

}

#### Snippet 24:

public class MissingBreakCase {

public static void main(String[] args) { int level = 1;

switch(level) { case 1:

System.out.println("Level 1");

A close up of a logo

Description automatically generatedcase 2:

System.out.println("Level 2");

case 3:

System.out.println("Level 3"); default:

System.out.println("Unknown level");

}

}

}

•**Error to Investigate**: When level is 1, why does it print "Level 1", "Level 2", "Level 3", and "Unknown level"? What is the role of the break statement in this situation?

* **Switch Case Fall-Through**: In Java, when a switch statement executes, it starts at the matching case and continues executing subsequent cases until it encounters a break statement or reaches the end of the switch block. This is known as "fall-through."

### Role of the break Statement:

* **Preventing Fall-Through**: The break statement is used to terminate the switch case after the matching case has been executed. When the break statement is encountered, the program exits the switch block, preventing any further cases from being executed.
* **Correct Usage**: To stop the fall-through and ensure that only the matching case is executed, you should add break statements at the end of each case.

**Solution of Snippet** 24**:**

public class MissingBreakCase {

public static void main(String[] args) {

int level = 1;

switch(level) {

case 1:

System.out.println("Level 1");

break;

case 2:

System.out.println("Level 2");

break;

case 3:

System.out.println("Level 3");

break;

default:

System.out.println("Unknown level");

break;

}

}

}

#### Snippet 25:

public class Switch {

public static void main(String[] args) { double score = 85.0;

switch(score) { case 100:

System.out.println("Perfect score!"); break;

case 85:

System.out.println("Great job!"); break;

default:

System.out.println("Keep trying!");

}

}

}

•**Error to Investigate**: Why does this code not compile? What does the error tell you about the types allowed in switch expressions? How can you modify the code to make it work?

**Ans:** The code does not compile because the switch statement in Java does not support double (or float) values. The switch statement can only be used with the following types: byte, short, int, char, enum, String and var.

-To make the code work, you can either convert the double value to an int, use a different type supported by switch, or use an if-else structure instead.

**Solution of Snippet** 25**:**

public class Switch {

public static void main(String[] args) {

int score = 85; // Change to int

switch(score) {

case 100:

System.out.println("Perfect score!");

break;

case 85:

System.out.println("Great job!");

break;

default:

System.out.println("Keep trying!");

}

}

}

#### Snippet 26:

public class Switch {

public static void main(String[] args) { int number = 5;

switch(number) { case 5:

System.out.println("Number is 5");

break; case 5:

System.out.println("This is another case 5"); break;

default:

System.out.println("This is the default case");

}

}

}

•**Error to Investigate:** Why does the compiler complain about duplicate case labels? What happens when you have two identical case labels in the same switch block?

**Ans:** The compiler complain about duplicate case labels because:

In Java, within a switch statement, each case label must be unique within the switch block. The compiler complains about duplicate case labels because each case label must be distinct; otherwise, the compiler cannot determine which block of code to execute.

When you have two identical case labels in the same switch block in Java, the compiler will generate an error. This is because having duplicate case labels creates ambiguity in which block of code should be executed, making it impossible for the compiler to determine which code should run when the switch statement matches that value.

**Solution of Snippet** 26**:**

public class Switch {

public static void main(String[] args) { int number = 5;

switch(number) { case 5:

System.out.println("Number is 5");

// No need for another case 5 break;

default:

System.out.println("This is the default case");

}

}

}

# Section 2: Java Programming with Conditional Statements

## Question 1: Grade Classification

Write a program to classify student grades based on the following criteria:

* + If the score is greater than or equal to 90, print "A"
  + If the score is between 80 and 89, print "B"
  + If the score is between 70 and 79, print "C"
  + If the score is between 60 and 69, print "D"
  + If the score is less than 60, print "F"

**Ans:**

**class Grade{**

**public static void main(String args[]){ int marks = 50;**

**if(marks>=90){ System.out.println("Grade A");**

**}else if(marks>=80 && marks<=89){ System.out.println("Grade B");**

**}else if(marks>=70 && marks<=79){ System.out.println("Grade C");**

**}else if(marks>=60 && marks<=69){ System.out.println("Grade D");**

**}else if(marks<60){ System.out.println("Grade F");**

**}**

**}**

**}**

## Question 2: Days of the Week

Write a program that uses a nested switch statement to print out the day of the week based on an integer input (1 for Monday, 2 for Tuesday, etc.). Additionally, within each day, print whether it is a weekday or weekend.

**Ans:**

**class Days{**

**public static void main(String args[]){ int day = 4;**

**switch(day){ case 1 :**

**System.out.println("Today is Monday"); break;**

**case 2 :**

**System.out.println("Today is Tuesday"); break;**

**case 3 :**

**System.out.println("Today is Wednesday"); break;**

**case 4 :**

A close up of a logo

Description automatically generated**System.out.println("Today is Thursday") break;**

**case 5 :**

**System.out.println("Today is Friday"); break;**

**case 6 :**

**System.out.println("Today is Saturday"); break;**

**case 7 :**

**System.out.println("Today is Sunday"); break;**

**case 8 : System.out.println("Invalid Day"); break;**

**}**

**}**

**}**

## Question 3: Calculator

Write a program that acts as a simple calculator. It should accept two numbers and an operator (+, -, \*, /) 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.

**Ans:**

**import java.util.Scanner; class Calculator{**

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

**double num1 , num2 , result; char operator;**

**|System.out.println("Enter first number : "); num1 = sc.nextDouble(); System.out.println("Enter Operator : "); operator = sc.next().charAt(0);**

**System.out.println("Enter second number : "); num2 = sc.nextDouble();**

**if(operator == '+'){ result = num1 + num2;**

**}else if(operator == '-'){ result = num1 - num2;**

**}else if(operator == '\*'){ result = num1 \* num2;**

**}else if(operator == '/'){ if(num2 != 0){**

A close up of a logo

Description automatically generated**result = num1 / num2;**

**}else{**

**System.out.println("Error: Division by zero not allowed"); return;**

**}**

**}else{**

**System.out.println("Invalid Operator"); return;**

**}**

**System.out.println("Result : " + 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 to Rs.1000, apply a 20% discount.
  + If the total purchase is between Rs.500 and 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%.

**Ans:**

**class Discount{**

**public static void main(String args[]){ int price = 200;**

**double total;**

**boolean membership = false; if(price>=1000){ if(membership == true){ total = price \* 0.25;**

**System.out.println("Total : " + total);**

**}else{**

**total = price \* 0.2;**

**System.out.println("Total : " + total);**

**}**

**}else if(price>=500 && price<=999){ if(membership == true){**

**total = price \* 0.15;| System.out.println("Total : " + total);**

**}else{**

**total = price \* 0.1; System.out.println("Total : " + total);**

**}**

A close up of a logo

Description automatically generated**}else if(price<500){ if(membership == true){ total = price \* 0.1;**

**System.out.println("Total : " + total);**

**}else{**

**total = price \* 0.05; System.out.println("Total : " + total);**

**}**

**}**

**}**

**}**

## 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.

**Ans:**

**public class StudentPassFail {**

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

**// Example grades for three subjects int grade1 = 45; // Score for Subject 1 int grade2 = 38; // Score for Subject 2 int grade3 = 50; // Score for Subject 3**

**// Initialize the count of failed subjects int failedCount = 0;**

**// Nested switch case to check grades for each subject for (int subject = 1; subject <= 3; subject++) {**

**int grade;**

**// Determine which subject's grade to use switch (subject) {**

**case 1:**

**grade = grade1; break;**

**case 2:**

**grade = grade2; break;**

**case 3:**

**grade = grade3; break;**

**default:**

A close up of a logo

Description automatically generated**grade = 0; // Default case, though this should not occur**

**}**

**// Nested switch case to determine if the grade is passing or failing switch (grade) {**

**case 0: // Special case if grades are outside expected range (for safety) case 1:**

**case 2:**

**case 3:**

**case 4:**

**case 5:**

**case 6:**

**case 7:**

**case 8:**

**case 9:**

**case 10:**

**case 11:**

**case 12:**

**case 13:**

**case 14:**

**case 15:**

**case 16:**

**case 17:**

**case 18:**

**case 19:**

**case 20:**

**case 21:**

**case 22:**

**case 23:**

**case 24:**

**case 25:**

**case 26:**

**case 27:**

**case 28:**

**case 29:**

**case 30:**

**case 31:**

**case 32:**

**case 33:**

A close up of a logo

Description automatically generated

**case 34:**

**case 35:**

**case 36:**

**case 37:**

**case 38:**

**case 39:**

**case 40:**

**// If grade is 40 or less, consider as fail failedCount++;**

**break; default:**

**// If grade is more than 40, consider as pass break;**

**}**

**}**

**// Determine and print the result based on the count of failed subjects if (failedCount == 0) {**

**System.out.println("The student passes.");**

**} else {**

**System.out.println("The student failed in " + failedCount + " subject(s).");**

**}**

**}**

**}**

# Section 3: Food for Thought: Research and Read More About

#### Evolution of Programming Languages

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

### 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?

#### Different Programming Languages and Their Usage

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

### 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?

#### Which Programming Language is the Best?

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

### 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?

#### Features of Java

* + **Research Topic:** Dive deep into the features of Java.

### 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?

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

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

### 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?

#### Can We Write Multiple main Methods?

* + **Research Topic:** Experiment with multiple main methods in Java.

### 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?

#### Naming Conventions in Java

* + **Research Topic:** Investigate Java's naming conventions.

### 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?

#### Java Object Creation and Memory Management

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

### 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.

#### Purpose of Access Modifiers in Java

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

### 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.