Name: Sumit Deshmukh_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?**
  Ans: Main method is not static in class Main, please define the main method as:
  public static void main(String[]args).
  Correct 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?**
  Ans: Here, public keyword is missing before static void main(String[] args)
  While running this code an error will be shown on the console **that main method not found in class Main.**

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?

Ans: The error that I encounter is:
Main method must return a value of type void in class Main, please

define the main method as:public static void main(String[] args)

Void is used in the main method to specify that the method does not return anything.

Snippet 4:

```
public class Main {
   public static void main() {
      System.out.println("Hello, World!");
   }
}
```

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

Ans: We get the following error in this code:
Main method not found in class Main, please define the main method as:
public static void main(String[] args)

Correct code:

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

String[] args is used to pass command line arguments to a java program.

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?
  Ans: Yes, we can have multiple main methods with different parameter lists.

  The overloaded `main` method with `int[] args` is not called because the JVM specifically looks for the `main` method with the signature `public static void main(String[] args)` as the entry point of the program. Other overloaded `main` methods are not recognized as entry points and thus are not executed automatically.

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?

Ans: We get the following error while running this program:

```
error: cannot find symbol

  int x = y + 10;
              ^
 symbol:   variable y
 location: class Main
```

Correct code:

```
public class Main {
public static void main(String[] args) {
 int y = 5;
 int x = y + 10;
 System.out.println(x);
}
}
```

Variables must be declared because variables are containers that store and manage data during program execution. They allow you to manipulate values, control program flow, and interact with different parts of your code. Declaring a variable in Java involves specifying its data type and name, which informs the compiler about the type of data the variable will hold and how much memory to allocate for it.

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?

Ans; We can see the following compilation error:

Main.java:3: error: incompatible types: String cannot be converted to int

```
      int x = "Hello";
                ^
```

Correct Code:

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

Type safety checks ensure that any object a method may try to manipulate is of the proper type. Suppose a program tried to apply the turnOn operation to an Applet object. If the operation were allowed to go ahead, it would do what turnOn was supposed to do, and set the first field of the object to true.

Snippet 8:

```
public class Main {

      System.out.println("Hello, World!"
   }
}
```

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

Ans: Syntax errors that are present in the code is:

the main method is not defined and the System.out.println statement is missing a closing parenthesis, and there are unmatched braces.

It will affect the compilation of the program.

At the time of compilation it will show following error:

```
Main.java:2: error: <identifier> expected

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

                  ^Main.java:2: error: illegal start of type
System.out.println("Hello, World!"

                   ^Main.java:4: error: class, interface, enum, or record expected

}

^
```

Correct code:

```
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?

Ans: Following errors are being occurred at the time of compilation:

```
Main.java:3: error: not a statement
        int class = 10;
        ^
Main.java:3: error: ';' expected
        int class = 10;
          ^
Main.java:3: error: <identifier> expected
        int class = 10;
              ^
Main.java:4: error: illegal start of expression
        System.out.println(class);
                  ^
Main.java:4: error: <identifier> expected
        System.out.println(class);
                        ^
5 errors
```

Correct Code:

```
public class Main {
  public static void main(String[] args) {
  int number = 10; // Changed variable name from 'class' to 'number'
  System.out.println(number);
}
}
```

This error indicates that int class = 10; is not a valid statement.

Keywords are reserved words so cannot be used as identifier, but in case sensitive languages, in which keywords are generally in lower case, so you can use uppercase name of keyword as an identifier.

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?

Ans: When we run this code the following error occurs:

```
Main.java:7: error: non-static method display() cannot be referenced from a static context

   public static void main(String[] args) {       display();       display(5);

                                    ^

Main.java:7: error: non-static method display(int) cannot be referenced from a static context

   public static void main(String[] args) {       display();       display(5);
```

^

2 errors

Correct Code:

```
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
obj.display();        // Call the display method without parameters
 obj.display(5);       // Call the display method with an integer parameter
   }
}
```

Java allows method overloading, which is a feature that lets programmers define multiple methods with the same name but different parameters. This can make code more readable and easier to understand, and can be useful when working with different data types or combinations of inputs.

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?

Ans: We get the following runtime exception:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

at Main.main(Main.java:3)

Correct Code:

```
public class Main {
   public static void main(String[] args) {
 int[] arr = {1, 2, 3};
// Accessing valid indices
for (int i = 0; i < arr.length; i++) {
 System.out.println(arr[i]);
 }
 }
}
```

The ArrayIndexOutOfBoundsException occurs because this code is trying to access an index that is outside the bounds of the array. In this case, the array has a length of 3, meaning valid indices are 0, 1, and 2. Attempting to access index 5, which does not exist, results in this exception.

Snippet 12:

```
public class Main {

      while (true) {
         System.out.println("Infinite Loop");
      }
   }
}
```

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

Ans: We get the following error while running and compiling the code:

```
Main.java:2: error: illegal start of type

      while (true) {

      ^

Main.java:6: error: class, interface, enum, or record expected

}

^

2 errors
```

Correct Code:

```
public class Main {
 public static void main(String[] args) {
while (true) {
System.out.println("Infinite Loop");
}
   }
}
```

Use break statements to exit loops when necessary. For example, in Java, you can place a condition within the loop that triggers a break statement when it's satisfied. In Python, you can use the break statement to exit a loop based on a certain condition.

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?

Ans: Following Exception is thrown at the time of running the program:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()"
because "<local1>" is null

        at Main.main(Main.java:3)
```

Correct Code:

```
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");
}
}
}
```

This exception is thrown because The NullPointerException occurs because our code is attempting to call the length() method on a String object that is null. This means that the variable you're trying to use has not been initialized with a valid String object. In above case, the error message indicates that the variable at Main.java:3 is null.

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?

Ans: Following compilation error occurs:

Main.java:2: error: incompatible types: String cannot be converted to double

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

Correct Code:

```
public class Main {
public static void main(String[] args) {
 double num = 10.5; // Assigning a valid double value
  System.out.println(num);
   }
}
```

Java enforce data type constraints because:

Constraints play a crucial role in Java programming, helping developers define rules and conditions that must be met for their code to function correctly. They ensure data integrity, protect against unexpected behavior, and contribute to the overall robustness of Java applications.

^

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?

Ans: Following error occurs at the time of compilation of this code:
Main.java:4: error: incompatible types: possible lossy conversion from double to int

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

                  ^
```

1 error

Correct code:

```
public class Main {
public static void main(String[] args) {
int num1 = 10;
double num2 = 5.5;
double result = num1 + num2;
System.out.println(result);
 }
}
```

To handle different data types in operations typecasting needs to be done first in order to convert all data types in one form.
We need to use double at the time of initialization and declaration of the result.

Snippet 16:

```
public class Main {

      int num = 10;
double result = num / 4;
      System.out.println(result);
   }
}
```

What is the result of this operation? Is the output what you expected?
Ans: The following is the result of this operation:
Main.java:3: error: <identifier> expected

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

^

Main.java:3: error: <identifier> expected

System.out.println(result);

^

Main.java:5: error: class, interface, enum, or record expected

}

^

3 errors

Correct Code:

```
public class Main {
public static void main(String[] args) {
  int num = 10;
double result = num / 4.0; // Ensure one operand is a double for floating-point
divisionSystem.out.println(result);
}
}
```

I was expecting num/4. The following error occurred due to missing of public static void main(String[]args) which is the main method.
Ensure that one operand is a double for floating-point division.

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?

Ans: The following compilation error is being occurred:

Main.java:2: error: illegal start of expression

int result = a ** b;

^

1 error

Correct Code:

```
public class Main {
 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);
  }
}
```

This error has occurred due to use of ** operator.
The ** operator is not valid in Java because Java does not use this operator for exponentiation. Instead, Java uses the Math.pow() method to perform exponentiation. The ** operator is commonly used in languages like Python for exponentiation, but Java has a different approach.

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?
  Ans: The Output of this code is 20.
  Operator precedence affects the order in which operations are performed in complex expressions, which can significantly change the result. For example, in the expression 1 + 5 * 3, the answer is 16 and not 18 because the multiplication operator has a higher precedence than the addition operator. This is because operator precedence specifies how tightly two expressions are bound together.

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?

Ans: The following Runtime exception is thrown:

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

at Main.main(Main.java:1)

Correct Code:

```
public class Main {
 public static void main(String[] args) {
int a = 10;
int b = 0;
try {
   int result = a / b;
  System.out.println(result);
} catch (ArithmeticException e) {
 System.out.println("Division by zero is not allowed.");
 }
 }
}
```

A divide by zero error generates a processor exception which triggers an interrupt. The interrupt is "read" by the operating system and forwarded to the program if a handler is registered. Since Java registers a handler, it receives the error and then translates it into an ArithmeticException that travels up the stack.

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?

Ans: The following Syntax error occours:

The error basically indicates semi colon missing.

```
Main.java:2: error: ';' expected

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

                                ^

1 error
```

Correct Code:

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

Missing of Semi colon demarcates a statement—the compiler is confused by its absence, thinking multiple lines are in fact a single statement. This generates a compilation error, as almost always the combined multi-line expression is invalid

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?

Ans: Compiler says the following about the mismatched braces:

Main.java:5: error: reached end of file while parsing

}

^

1 error

Correct Code:

```
public class Main {
 public static void main(String[] args) {
System.out.println("Hello, World!");
  } // Closing brace for main method
} // Closing brace for Main class
```

Closing braces is missing for the main method in the above snippet.

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?

Ans: The following Syntax error occurs:

```
Main.java:2: error: illegal start of expression

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

                                    ^

Main.java:6: error: class, interface, enum, or record expected

}

^

2 errors
```

Correct code:

```
public class Main {
// Define the displayMessage method outside of main method
    static void displayMessage() {
  System.out.println("Message");
    }
    public static void main(String[] args) {
// Call the displayMessage method from within the main method
  displayMessage();
 }
}
```

No. A method cannot be declared inside of another method in Java.

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?

  Ans:

  The reason that the default case print after "Value is 2" is the missing of the break statement.

  To prevent the program from executing the default case we need to add break statement at the end of every case.

  Correct Code:

```
public class Confusion {
   public static void main(String[] args) {
      int value = 2;
      switch(value) {
         case 1:
            System.out.println("Value is 1");
            break; // Exit the switch block after this case
         case 2:
            System.out.println("Value is 2");
            break; // Exit the switch block after this case
         case 3:
            System.out.println("Value is 3");
            break; // Exit the switch block after this case
         default:
            System.out.println("Default case");
            break; // Exit the switch block after this case
      }
   }
}
```

Snippet 24:

```
public class MissingBreakCase {
public static void main(String[] args) {
      int level = 1;
switch(level) {
case 1:
            System.out.println("Level 1");
case 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?
  Ans: The reason that it prints"Level 1","Level 2","Level 3" and "Unknown level" is that due to missing of break statement.

  The role of break statement in this situation is to terminate the code once the given condition is satisfied.

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.

  Correct Code:

```
public class Switch {
public static void main(String[] args) {
 double score = 85.0;
int scoreInt = (int) score; // Cast the double to int
switch(scoreInt) {
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.

  Correct code:

```
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 :
 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 ifelse 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){
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);
}
}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:
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:

            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

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 tradeoffs between simplicity and control over the hardware?

2. 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?

3. 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?

4. 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?

5. 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?

6. 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?

7. 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?

8. 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.

9. 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.