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?**
  1)Error: Main method not found in class Main, please define the main method as:
  public static void main(String[] args)
  2)Error: Main method not found in class Main, please define the main method as:
  public static void main(String[] args)

  3)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?**
  while compiling it give:
  error: class Main is public, should be declared in a file named Main.java
  public class Main {

if, we run code it give:-
Error: Could not find or load main class Main

```
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: Could not find or load main class Main

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

  }
  }

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?**
  while compiling:-
  error: class Main is public, should be declared in a file named Main.java
  public class Main {
  while running:-
  Error: Could not find or load main class Main

  String[] arg is needed in Java to allow your program to accept and process command-line arguments, providing flexibility in how the program can be executed and used.

```
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");
   }
}
```

output:-

Main method with String[] args

- **Can you have multiple main methods? What do you observe?**
  Yes we have multiple main method but JVM compiler always run first main method the other main methods will not be used by the JVM as entry points.

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


}
```

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: cannot find symbol

  Variables must be declared to allocate memory, enforce type safety, improve code readability, define scope, enable compiler optimizations, and prevent errors.

```
public class Main {
   public static void main(String[] args)
      {int y=10;
    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

  Java enforces type safety to prevent errors, improve code quality, enhance security, support object-oriented programming principles, enable compiler optimizations, and provide a safe runtime environment.

```
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?**
  demo.java:3: error: ')' or ',' expected
  The missing closing parenthesis in the system.out.println statement will result in a syntax error during compilation.

```
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?**

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

Reserved keywords cannot be used as identifiers because they have specific, predefined meanings in the language's syntax. Allowing them as identifiers would lead to ambiguity for the compiler, reduce code clarity, violate language rules, introduce potential logical errors, and hinder code portability.

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

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?**
  demo.java:10: error: non-static method display() cannot be referenced from a static context
  display();
  ^
  demo.java:11: error: non-static method display(int) cannot be referenced from a static context
  display(5);

  Method overloading is allowed and works correctly when the code is properly adjusted.

```
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();
      obj.display(5);
   }
}
```

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?**
  Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
  at Main.main(demo.java:4)
  Because there the array of sized is only 3.

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

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?**

It print “Infinite loop” in infinite time.we have to false while condition to stop loop.

```
public class Main {
    public static void main(String[] args) {
    int n=5;
    while (n>0) {
            System.out.println("Infinite Loop");
        n–;
        }
    }
}
```

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?**

1)Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is null
    at Main.main(demo.java:4)
2)Because string is null.

```
3)public class Main {
    public static void main(String[] args) {
        String str = “String”l;
        System.out.println(str.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?**

1)demo.java:3: error: incompatible types: String cannot be converted to double

2)double num = "Hello";

3)Java enforces strict data type constraints to ensure type safety, efficient memory management, and to prevent undefined behavior, making code more reliable and maintainable.

4)
```
public class Main {
    public static void main(String[] args) {
        double num = 10.5;
        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?**
    1) demo.java:5: error: incompatible types: possible lossy conversion from double to int
    int result = num1 + num2;
    2) Store in a double: Safest approach if you need to preserve the decimal part.
    Cast to int: Use this if you only care about the integer part of the result, but be aware of the data loss.

```
public class Main {
    public static void main(String[] args) {
        double num1 = 10;
        double num2 = 5.5;
        double result = num1 + num2;
        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?**
  1)output:-2.0
  2)expected:-2.5

```
3)public class Main {
   public static void main(String[] args) {
      double num = 10;
      double result = num / 4;
      System.out.println(result);
   }
}
```

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?**
  demo.java:5: error: illegal start of expression
  int result = a ** b;

  Java provides a set of predefined operators for arithmetic, logical, bitwise, and other operations, but it does not include an operator for exponentiation (“**”). Java instead relies on methods from its standard library for operations like exponentiation.

```
public class Main {
   public static void main(String[] args) {
      int a = 10;
      int b = 5;
      int result = 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?**

1)output:-20

2)The multiplication operator (“*”) has higher precedence than the addition operator (“+”), so” b* 2” is evaluated before” a+” This order of operations is crucial for correctly interpreting and calculating the result of expressions.

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
  at Main.main(demo.java:5)
  Issue with Division by Zero: Division by zero is mathematically undefined and causes a runtime exception to prevent undefined behavior and potential system instability. Proper checks should be in place to handle such cases safely.

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;

        if (b != 0) {
            int result = a / b;
            System.out.println(result);
        } else {
            System.out.println("Division by zero is not allowed.");
        }
    }
}
```

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?**
  error:-
  demo.java:3: error: ';' expected
  System.out.println("Hello, World")

  The missing semicolon causes the compiler to throw a syntax error, stopping the compilation process until the error is corrected. In Java, semicolons are essential for properly terminating statements.

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

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?**
  demo.java:5: error: reached end of file while parsing
  }
  ^

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

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?**
  demo.java:3: error: illegal start of expression
  static void displayMessage() {
  ^
  demo.java:7: error: class, interface, enum, or record expected
  }

```
^
2 errors
```

Methods must be declared at the class level in Java, not within other methods. The corrected code shows the proper way to declare and use a static method

```
public class Main {
    public static void main(String[] args) {
        displayMessage();
    }

    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?
  The default case and subsequent cases were executed due to the lack of “break” statements, leading to fall-through behavior.Adding “break” statements after each case prevents fall-through, ensuring only the matched case's code is executed.

```
public class Confusion {
    public static void main(String[] args) {
        int value = 2;
        switch(value) {
            case 1:
                System.out.println("Value is 1");
                break;  // Stops after executing case 1
            case 2:
                System.out.println("Value is 2");
                break;  // Stops after executing case 2
            case 3:
                System.out.println("Value is 3");
                break;  // Stops after executing case 3
            default:
                System.out.println("Default case");
                break;  // Stops after executing the default 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?
  Without “break” statements, the “switch” block continues to execute all subsequent cases, leading to the unintended output of "Level 1", "Level 2", "Level 3", and "Unknown level" when “level1”is 1.

```
public class MissingBreakCase {
   public static void main(String[] args) {
      int level = 1;
      switch(level) {
         case 1:
            System.out.println("Level 1");
            break;  // Stops after executing case 1
         case 2:
            System.out.println("Level 2");
            break;  // Stops after executing case 2
         case 3:
            System.out.println("Level 3");
            break;  // Stops after executing case 3
         default:
            System.out.println("Unknown level");
            break;  // Stops after executing the default case
      }
   }
}
```

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?

  The”switch”statement does not support “double” as a type for its expression, leading to a compilation error.
  switch supports int,byte String ,short,char and enum types.
  Solution: Either cast the double to an int (if appropriate) or use an if-else structure for floating-point comparisons.

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

        if (score == 100.0) {
            System.out.println("Perfect score!");
        } else if (score == 85.0) {
            System.out.println("Great job!");
        } else {
            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?

  The compiler complains about duplicate case labels because each “case”label must be unique within a “switch” block.

  Duplicate labels cause ambiguity in determining which block of code should be executed for a given value.

  Ensure that each “case “ label is unique, or use different logic to handle similar values.

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"

```
import java.util.*;
class Main {



  public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the student's score: ");
      int score = scanner.nextInt();

      if (score >= 90) {
          System.out.println("A");
      } else if (score >= 80) {
          System.out.println("B");
      } else if (score >= 70) {
          System.out.println("C");
      } else if (score >= 60) {
```

```
            System.out.println("D");
        } else {
            System.out.println("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.

```
import java.util.*;

class Main1 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);




        System.out.print("Enter a number : ");

        int dayNumber = scanner.nextInt();



        // Outer switch to determine the day of the week

        switch (dayNumber) {

            case 1:

                System.out.println("Monday");

                // Nested switch to determine if it's a weekday or weekend

                switch (dayNumber) {



                    default:

                        System.out.println("Weekday");
```

```
            break;
        }
        break;
    case 2:
        System.out.println("Tuesday");
        switch (dayNumber) {
            default:
                System.out.println("Weekday");
                break;
        }
        break;
    case 3:
        System.out.println("Wednesday");
        switch (dayNumber) {
            default:
                System.out.println("Weekday");
                break;
        }
        break;
    case 4:
        System.out.println("Thursday");
        switch (dayNumber) {
            default:
                System.out.println("Weekday");
                break;
        }
```

```
    break;
case 5:
    System.out.println("Friday");
    switch (dayNumber) {
        default:
            System.out.println("Weekday");
            break;
    }
    break;
case 6:
    System.out.println("Saturday");
    switch (dayNumber) {

        default:
            System.out.println("Weekend");
            break;
    }
    break;
case 7:
    System.out.println("Sunday");
    switch (dayNumber) {
        default:
            System.out.println("Weekend");
            break;
    }
    break;
```

```
            default:
                System.out.println("Invalid input.");
        }



    }
}
```

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.

```
import java.util.Scanner;


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


        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();


        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();


        System.out.print("Enter an operator (+, -, *, /): ");
        char operator = scanner.next().charAt(0);
```

```
double result;


switch (operator) {
    case '+':
        result = num1 + num2;
        System.out.println("The result is: " + result);
        break;
    case '-':
        result = num1 - num2;
        System.out.println("The result is: " + result);
        break;
    case '*':
        result = num1 * num2;
        System.out.println("The result is: " + result);
        break;
    case '/':
        // Nested if-else to check for division by zero
        if (num2 != 0) {
            result = num1 / num2;
            System.out.println("The result is: " + result);
        } else {
            System.out.println("Error: Division by zero is not allowed.");
        }
        break;
    default:
        System.out.println("Error: Invalid operator entered.");
```

```
        }
    }
}
```

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%.

```
import java.util.Scanner;

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

        System.out.print("Enter the total purchase amount in Rs: ");
        double totalPurchase = scanner.nextDouble();

        System.out.print("Do you have a membership card? (yes/no): ");
        String membershipResponse = scanner.next();
        boolean hasMembership = membershipResponse.equalsIgnoreCase("yes");

        double discount = 0;

        if (totalPurchase >= 1000) {
            discount = 0.20;  // 20% discount
        } else if (totalPurchase >= 500) {
            discount = 0.10;  // 10% discount
        } else {
            discount = 0.05;  // 5% discount
        }

        if (hasMembership) {
            discount += 0.05;  // Increase discount by 5%
        }



        double discountAmount = discount * totalPurchase;
        double finalAmount = totalPurchase - discountAmount;



        System.out.println("Total Purchase Amount: Rs " + totalPurchase);
        System.out.println("Discount Applied: Rs " + discountAmount + " (" + (discount * 100) +
"%)");
        System.out.println("Final Amount to Pay: Rs " + finalAmount);
    }
}
```

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;

public class StudentResult {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);



        System.out.print("Enter the grade for Subject 1: ");

        int subject1 = scanner.nextInt();



        System.out.print("Enter the grade for Subject 2: ");

        int subject2 = scanner.nextInt();



        System.out.print("Enter the grade for Subject 3: ");

        int subject3 = scanner.nextInt();



        int failCount = 0;



        if (subject1 <= 40) {

            failCount++;

        }

        if (subject2 <= 40) {

            failCount++;
```

```
        }
        if (subject3 <= 40) {
            failCount++;
        }
        if (failCount == 0) {
            System.out.println("Pass");
        } else {
            System.out.println("Fail in " + failCount + " 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.
    - **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?

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.