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.

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).

Corrected 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. Main method not found in class Main, please define the main method as:
public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application.

Corrected 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?**

Ans. Main method must return a value of type void in class Main, please define the main method as:

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

Corrected snippet 3:

```
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?**

Ans. Error: Main method not found in class Main, please define the main method as:

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

Corrected snippet 3:

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

The JVM is designed to recognize public static void main(String[] args) as the entry point for any Java application. If this is not present, then JVM won't know where to start the execution.

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. In the above snippet, first main method is will be considered as the signature method which is the starting point of execution. Second method will not be executed since it is not called from the first main method.

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. Error : cannot find symbol. This error occurred since the variable y is not declared in the snippet.

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. error: incompatible types: String cannot be converted to int. Since the variable x is assigned a string and its data type is declared as int, that's why the error is thrown.Type safety in java ensures that variables do not access memory in inappropriate ways.

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?**
  Ans. There is syntax error is that closing parenthesis is missing at the end of SOP statement.When we compile the given code it throws error by showing to add missing parenthesis.

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. error: <identifier> expected

The above snippet throws error since ‘class’ is the reserved keyword in java, we cannot use reserved keywords as variable names in java. Using these keywords is not allowed in java because they have predefined meanings in java.

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. The above snippet throws some errors as main method is static, but the display methods are non-static. In java, we cannot call non-static methods directly from a static method like main method without creating an instance of the class. We need to create the object for the non-static methods.
  Corrected 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();
      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?**
  Ans. Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3. This exception occurred due to the size provided while printing the array. There are only 3 elements present in the array.

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?**
  Ans. The while loop created an infinite loop, continuously printing “infinite loop” to the console. This while loop will create a loop that will never terminate because the condition is true always. We need to give the condition to terminate the loop.

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. Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is null.
  This occurred because we are trying to call the length() method on a null reference.

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. error: incompatible types: String cannot be converted to double.
  This snippet throws error because we are trying to assign a string value to a double variable. In java, we cannot directly assign a string to a double due to type constraints.

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. error: incompatible types: possible lossy conversion from double to int
  This snippet throws an compilation error because we are trying to assign the result of adding an int and a double to an int variable without explicit casting.

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?**
  Ans. Output : 2.0 . I expected the output as 2.5, but then after researching I found that integer division discards the fractional part.

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. error: illegal start of expression. In java ** operator is not defined for exponentiation.java does not have a built-in operator for raising a number to power. Instead we need to use Math.pow method.

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. Output: 20 . In java operators with higher precedence are evaluated before operators with lower precedence. Multiplications operator has higher precedence than addition operator.

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

  In mathematics, dividing any number by zero is undefined. To prevent undefined behavior and potential errors, Java throws an ArithmeticException when an integer division by zero is attempted.

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. error: ';' expected . In java, a missing semicolon at the end of a statement causes a compilation error because the semicolon is used to terminate statements.

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. error: reached end of file while parsing. This happens because the compiler expects every opening brace { to have a corresponding closing brace }.

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. error: illegal start of expression . In java, methods cannot be declared within other methods. Java does not directly supports nested methods. Methods must be declared within a class, not within another method.

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. When value is 2, the switch statement jumps to case 2 and prints “value is 2”. Since there is no break statement after case 2, the execution continues to case 3 and prints “value is 3”. We need to add the break statement at the end of each case to prevent the program from executing the default case or any other cases.

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. When level is 1, the switch statement jumps to case 1 and prints “Level 1”.Since there is no break statement after case 1, the execution continues to case 2 and prints “Level 2”.It then continues to case 3 and prints “Level 3”.Finally, it reaches the default case and prints “Unknown level”. The break statement is used to terminate a case in the switch block.

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. Code does not compile because switch statement in java does not support double type. error: incompatible types: possible lossy conversion from double to int. To make the code work, we can use an int type for the score variable.

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. error: duplicate case label . In a switch statement, each case label must be unique. The switch statement uses these labels to determine which block of code to execute based on the value of the expression. Having duplicate case labels would create ambiguity, as the compiler wouldn’t know which block to execute for a given value.

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"

Questin 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.

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.

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%.

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.

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.