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?**

main method is not declared static

Snippet 2:
```
public class Main {
    static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- **What happens when you compile and run this code?**

main method is not declared public

Snippet 3:
```
public class Main {
    public static int main(String[] args) {
        System.out.println("Hello, World!");
        return 0;
    }
```

When java program is terminated it does not return any value

```
}
```

- **What error do you encounter? Why is `void` used in the `main` method?**

Snippet 4:

to pass command-line arguments to a Java program.

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

when string is converted to any datatype then there will be no data loss
it will include all the data

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

The program will compile, but not run, because JVM will not recognize
the main() method. Remember JVM always looks for the main()
method with a string type array as a parameter.

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");
    }
}
```

When we run this Java program, the Java Virtual
Machine (JVM) will look for the main method with the
signature public static void main(String[] args) and
execute it. This is the entry point function

output:- Main method with String[] args

- **Can you have multiple `main` methods? What do you observe?**

Snippet 6:

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

```
error: cannot find symbol
 int x = y + 10;
         ^
  symbol:   variable y
  location: class Main
```

- **What error occurs? Why must variables be declared?**

so we can the data in that variable

Snippet 7:

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

Type safety means that the compiler will validate types
while compiling, and throw an error if you try to assign the
wrong type to a variable.

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

```
error: incompatible types:
String cannot be converted to
int
 int x = "Hello";
```

the types does not use the same amount of
memory and the computer needs to be made
aware

Snippet 8:

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

```
error: ')' expected
 System.out.println("Hello, World!"
                                   ^
1 error
```

- **What syntax errors are present? How do they affect compilation?** Compiler expects a closing parenthesis to complete the method call.

Snippet 9:

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

```
error: <identifier> expected
 int class = 10;
```

The compiler would not be able to distinguish between a keyword's intended purpose and its use as an identifier, leading to confusion and errors in program logic.

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

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);
    }
}
```

```
error: non-static method display() cannot be referenced from
a static context
 display();
 ^
error: non-static method display(int) cannot be referenced
from a static context
 display(5);
 ^
```

- **What happens when you compile and run this code? Is method overloading allowed?** Yes

Snippet 11:

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

```
ERROR!
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of
bounds for length 3
 at Main.main(Main.java:4)
```

- **What runtime exception do you encounter? Why does it occur?**

the size of array is 3 and program is asking for 5 index

Snippet 12:

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

inifinite Loop will run

we can give condition for while loop instead of true

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

ERROR!
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is null
 at Main.main(Main.java:4)

Snippet 13:

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

value of string variable is null
so compiler could recognize the length of string

- **What exception is thrown? Why does it occur?**

Snippet 14:

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

error: incompatible types

the types does not use the same amount of memory and the computer needs to be made aware

- **What compilation error occurs? Why does Java enforce data 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);
   }
}
```

```
 error: incompatible types: possible lossy conversion from double to int
 int result = num1 + num2;
                   ^
1 error
```

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

we can change result datatype into double
or
we can use Explicit typecasting for int

Snippet 16:

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

2.0

yes, because int/int = int
but datatype of result is double so it 2.0 instead of 2

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

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);
    }
}
```

```
error: illegal start of expression
 int result = a ** b;
                 ^
```

for exponentiation java has pow function
** operator is invalid in java

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

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);
    }
}
```

result = 10 + 5 * 2
= 10 + 10
= 20

20

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

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);
    }
}
```

ERROR!

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at Main.main(Main.java:5)
```

denominator can't be zero
we can't divide any number with zero it can not give us finite answer

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

Snippet 20:

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

```
error: ';' expected
 System.out.println("Hello, World")
                                   ^
```

; means end of that statement

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

Snippet 21:

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

```
error: reached end of file while parsing
}
 ^
```

- **What does the compiler say about mismatched braces?**

Snippet 22:

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

in java, we cannot declare method inside the methos
all the should be at class level

```
error: illegal start of expression
 static void displayMessage() {
 ^
ERROR!
error: class, interface, or enum expected
}
^
```

- **What syntax error occurs? Can a method be declared inside 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");
        }
    }
}
```

using break after each 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?

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");
        }
    }
}
```

bcoz break is not used after cases

if case is true then it will break the loop and will not check next cases

- **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?

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: incompatible types: possible lossy conversion from double to int
 switch(score) {
       ^
```

Reason:
In Java, the switch statement supports the following types:

byte, short, char, int (and their corresponding wrapper classes Byte, Short, Character, Integer)
enum types
String (since Java 7)
var (if it resolves to one of the supported types)
The switch statement does not support float, double, long, or their corresponding wrapper classes (Float, Double, Long).

- **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?

How to Modify the Code:
To make the code work, you need to either:

Use an integer type for the score variable.
Use an if-else statement instead of a switch statement if you need to work with a double.

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: duplicate case label
 case 5:
 ^
```

- **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?

  At runtime, if two cases in a switch statement have the same label, the second case will never be executed.

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"

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.

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.