

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 static in class Main, please define the main method as:
`public static void main(String[] args)`
-

Snippet 2:

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

- **What happens when you compile and run this code?**
It is compiled but the error is = Main method is not static in class Main, please define the main method as: `public static void main(String[] args)`
-

Snippet 3:

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

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`

```
}
```

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

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?
-

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?

No, Main method with String[] args

Snippet 6:

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

java:3: error: cannot find
symbol int x = y + 10;
 symbol: variable y
 location: class Main

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

Snippet 7:

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

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

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

Snippet 8:

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

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

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

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?

They are having already predefined meaning which is already known to the compiler of 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);  
    }  
}
```

error: non-static method display() cannot be
referenced from a static context

display();
^

Main.java:10: error: non-static method display(int)
cannot be referenced from a static context

display(5);
^

2 errors

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

Snippet 11:

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

Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: Index 5 out of
bounds for length 3
at Main.main(Main.java:4)

Since array is of size 3 elements and we are trying to access
element no. 5, That's why we are facing this error.

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

Snippet 12:

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

Code will run into infinite loop,
we can stop by doing condition as false.

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

Snippet 13:

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

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

length is null and we are trying to get its length, so runtime error.

- 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);  
    }  
}
```

incompatible types: String
cannot be converted to double
double num = "Hello";

- 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);  
    }  
}
```

incompatible types: possible
lossy conversion from double to
int

int result = num1 + num2;

We can do simply typecasting
by doing "int(5.5)"

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

Snippet 16:

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

2.0 (answer is in float format.)

- 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;

- 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);  
    }  
}
```

20

Operator uses BODMAS method for precedence.

- 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);  
    }  
}
```

java.lang.ArithmeticException: / by zero
at Main.main(Main.java:5)

We can't divide any number by 0

- 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")

- **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  
    }
```

Code is executed by { _ }

- **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");  
        }  
    }  
}
```

error: illegal start of expression
static void displayMessage() {
^
Main.java:7: error: class, interface,
enum, or record 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");  
        }  
    }  
}
```

Value is 2
Value is 3
Default case

We can prevent it by using break statement.

- **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");
        }
    }
}
```

Level 1
Level 2
Level 3
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?
-

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: patterns in switch statements are a preview feature and are disabled by default.

```
switch(score) {
    ^
```

(use --enable-preview to enable patterns in switch statements)

Switch.java:5: error: constant label of type int is not compatible with switch selector type double

```
case 100:
    ^
```

Switch.java:8: error: constant label of type int is not compatible with switch selector type double

```
case 85:
    ^
```

- **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?
-

Snippet 26:

```
public class Switch {
    public static void main(String[] args) {
        int number = 5;
        switch(number) {
            case 5:
                System.out.println("Number is 5");
        }
    }
}
```

error: duplicate case label
case 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?

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.

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.