

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) { //static is missing  
        System.out.println("Hello, World!");  
    }  
}
```

What error do you get when running this code?

Output:

```
Error: Main method is not static in class Main, please define the main method as:  
public static void main(String[] args)
```

Correct Code with Output:

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

```
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?

Compile and Run successfully.

```
Hello, World!
```

Snippet 3:

```
public class Main {  
    public static int main(String[] args) { //Void is missing  
        System.out.println("Hello, World!");  
        return 0; //unexpected return value  
    } //one more closing bracket require }
```

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

It is a keyword and is used to specify that a method doesn't return anything. As the main() method doesn't return anything, its return type is void. As soon as the main() method terminates, the Java program terminates too. Hence, it doesn't make any sense to return from the main() method as JVM can't do anything with its return value of it.

If main method is not void, we will get an error.

Output:

```
Main.java:6: error: reached end of file while parsing  
    }  
    ^  
1 error
```

```
Error: Main method must return a value of type void in class Main, please  
define the main method as:  
    public static void main(String[] args)
```

```
Main.java:5: error: incompatible types: unexpected return value  
        return 0;  
        ^  
1 error
```

Correct Code with Output:

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

Output:

```
Hello, World!
```

Snippet 4:

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

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

String[] args

It **stores Java command-line arguments** and is an array of type ***java.lang.String*** class. Here, the name of the String array is *args* but it is not fixed and the user can use any name in place of it.

Output:

It compile but give Run time error

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java  
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main  
Error: 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
```

Correct Code

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

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main  
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");  
    }  
}
```

Can you have multiple main methods? What do you observe?

Output

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main  
Main method with String[] args
```

Multiple main methods means Overloading the main() method is possible in Java, meaning we can create any number of main() methods in a program.

To overload the main() method in Java, we need to create the main() method with different parameters.

I observe Java main() method is the starting point of a Java program. It is the main body that is executed by the JVM, and without the main() method no Java program can be run without it.

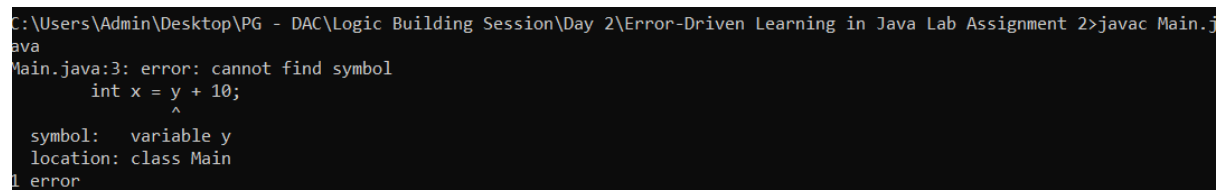
Java main method can not be int. There are two main reasons for it:

1. JVM looks for **public static void main(String[] args)** when starting the program execution as it is the standard signature for entry. Using int signature would cause confusion and compatibility issues while program execution.
2. Having void signature means the main method will not return anything, but using int signature means the main function will have to return integer, which isn't useful for JVM.

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?



```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
Main.java:3: error: cannot find symbol
    int x = y + 10;
            ^
    symbol:   variable y
    location: class Main
1 error
```

Variables must be declared

Variable/Identifiers in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. Variable is a memory location name of the data. A variable is a name given to a memory location.

Java is a statically-typed language. It means that all variables must be declared before they can be used.

Correct Code

```
public class Main {
    public static void main(String[] args) {
        int y = 5; // Declare and initialize y
        int x = y + 10;

        System.out.println(x) } }
```

Output :



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?

Type safety in Java ensures that a variable or object is only assigned values that are compatible with its declared type. This prevents type errors and enhances the reliability and robustness of the code. Java enforces type safety through its strong type system, which includes compile-time and runtime checks.

Output:

```
java  
Main.java:3: error: incompatible types: String cannot be converted to int  
    int x = "Hello";  
          ^  
1 error
```

Correct code:

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

Output: Hello

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?

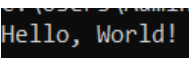
Round bracket and semicolon ; is expected

Output:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
Main.java:4: 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!")
    }
}
```

O/P: 

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?

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
Main.java:4: error: not a statement
    int class = 10;
    ^
Main.java:4: error: ';' expected
    int class = 10;
    ^
Main.java:4: error: <identifier> expected
    int class = 10;
    ^
Main.java:5: error: <identifier> expected
    System.out.println(class);
    ^
Main.java:5: error: illegal start of type
    System.out.println(class);
    ^
Main.java:5: error: <identifier> expected
    System.out.println(class);
    ^
Main.java:7: error: reached end of file while parsing
}
^
7 errors
```

reserved keywords can't be used as identifiers because

In Java (and most programming languages), **reserved keywords** are predefined words that have a special meaning in the language syntax. These keywords are reserved by the language and are used to define the structure, control flow, or behavior of the program.

Examples of reserved keywords in Java include:

- class
- if
- for
- public
- static
- int
- void
- while

Because these keywords have specific roles in the language, **they cannot be used for variable names**, method names, or any other identifiers. Using them for anything else would cause confusion for the compiler, as it would not be able to distinguish between the keyword's intended meaning and an identifier you might want to use.

In summary, reserved keywords serve a special purpose within the language's grammar, and preventing them from being used as identifiers helps to avoid conflicts, ensures proper interpretation of the code, and maintains clarity.

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?

Output

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java  
Main.java:10: error: non-static method display() cannot be referenced from a static context  
    display();  
    ^  
Main.java:11: error: non-static method display(int) cannot be referenced from a static context  
    display(5);  
    ^  
2 errors
```

Is method overloading allowed?

In the main method, you're trying to call `display()` and `display(5)` directly, but the `display` method is an instance method, not a static method. You cannot call an instance method (like `display()`) without creating an object of the `Main` class, since instance methods belong to objects rather than the class itself.

To fix this, you need to create an object of the `Main` class in the main method and call the `display()` methods on that object.

Method overloading is allowed and works correctly.

The original code failed to compile because it attempted to call instance methods from a static context (the main method).

What happens when you compile and run this corrected code?

- **Method Overloading:** This is **method overloading**, which is allowed in Java. Overloading occurs when multiple methods in the same class have the same name but different parameter lists (in this case, one method takes no parameters, and the other takes an integer).

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 the Main class

obj.display(); // Call the first display method

obj.display(5); // Call the second display method with an integer parameter

}

}

```

O/p:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
No parameters
With parameter: 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?

Output:

ArrayIndexOutOfBoundsException error occur

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
    at Main.main(Main.java:5)

```

Why does it occur?

- The array arr has only 3 elements: arr[0] = 1, arr[1] = 2, and arr[2] = 3.
- In Java, array indices are zero-based, meaning the valid indices for this array are 0, 1, and 2.
- The line System.out.println(arr[5]); is trying to access arr[5], which is an index that **does not exist** in the array. Since the array has only 3 elements (indices 0, 1, and 2), trying to access arr[5] is out of bounds.

What happens:

- At runtime, Java detects that you are trying to access an index that exceeds the bounds of the array, and it throws an `ArrayIndexOutOfBoundsException` to signal that the index you're trying to access is invalid.

How to fix:

- To avoid this exception, you should ensure that you are accessing valid indices within the array. For example, you could check the length of the array before accessing an index or use an index that falls within the array bounds:

Correct Code:

```
public class Main {  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3};  
        if (arr.length > 5) {  
            System.out.println(arr[5]);  
        } else {  
            System.out.println("Index 5 is out of bounds.");  
        }  
    }  
}
```

o/p:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java  
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main  
Index 5 is out of bounds.
```

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?

The code you provided creates an **infinite loop**, which means that it will continually print "Infinite Loop" to the console until the program is manually stopped.

Here's how the code works:

1. `while (true)` is a loop condition that will always evaluate to true. Since there's no way for the condition to become false, the loop will never exit on its own.
2. `System.out.println("Infinite Loop");` prints the string "Infinite Loop" to the console.
3. This will continue endlessly, as long as the program is running.



How can you avoid infinite loops?

The loop condition can eventually become false (or there is an explicit exit condition, like `break`).

The loop does not run forever without a termination point.

In this modified version, the loop will run 10 times and then exit when count reaches 10.

Alternatively, you could use a `break` statement to exit the loop based on some other condition inside the loop.

Correct Code:

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

```

while (count < 10) { // Condition that will become false after 10 iterations

    System.out.println("Loop iteration " + count);

    count++; // Increment the counter

}

}

}

```

O/p:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
Loop iteration 0
Loop iteration 1
Loop iteration 2
Loop iteration 3
Loop iteration 4
Loop iteration 5
Loop iteration 6
Loop iteration 7
Loop iteration 8
Loop iteration 9

```

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?

The code you provided will throw a NullPointerException

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
Exception in thread "main" java.lang.NullPointerException
    at Main.main(Main.java:4)

```

why it occurs:

- **String str = null;;** This creates a String variable str and assigns it a null value. This means str does not reference any valid object in memory.
- **System.out.println(str.length());;** You're attempting to call the length() method on the str object. However, since str is null, there is no actual object to invoke the length() method on.

In Java, trying to call a method (like length()) on a null reference results in a NullPointerException because the Java Virtual Machine (JVM) cannot access any methods or properties on null.

Exception Thrown:

- **NullPointerException:** This is the runtime exception that occurs when an application attempts to use null where an object is required (for example, calling methods or accessing fields of null).

How to Avoid This Exception:

To avoid a NullPointerException, you can ensure that the object reference is not null before calling methods on it.

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");
        }
    }
}
```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
String 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?

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
Main.java:3: error: incompatible types: String cannot be converted to double
    double num = "Hello";
                ^

```

The code you provided will result in a **compilation error**

- double is a primitive data type used for representing floating-point numbers.
- "Hello" is a String, which is a sequence of characters.

Java enforces strict type checking and does not allow automatic conversion between these two incompatible types. In Java, you cannot assign a string to a numeric type (like double) without explicitly converting it (which is not possible in this case because "Hello" cannot be converted to a number).

Correct Code:

To fix this error, you need to assign a valid double value to the num variable.

```

public class Main {
    public static void main(String[] args) {
        double num = 42.0; // valid double value
        System.out.println(num);
    }
}

```

o/p:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
42.0

```

If want to work with the string "Hello" and somehow need a double, you'd have to explicitly convert the string to a number (though "Hello" cannot be converted to a numeric value). You might need to handle it differently depending on your use case.

```

public class Main {
    public static void main(String[] args) {
        String str = "42.0";
        double num = Double.parseDouble(str); // Convert string to double
        System.out.println(num);
    }
}

```

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
42.0

```

In this case, "42.0" is a valid number represented as a string, and Double.parseDouble(str) will convert it to a double.

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?

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main
ava
Main.java:5: error: incompatible types: possible lossy conversion from double to int
    int result = num1 + num2;
                      ^
1 error

```


How to Handle Different Data Types in Operations:

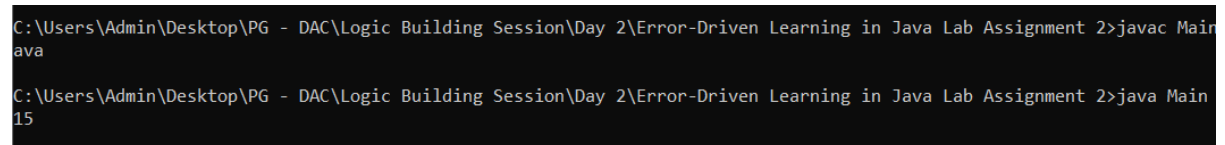
To handle operations between different data types properly, you need to ensure that the result type is compatible with the variable you're assigning it to. You can:

Correct Output:

1. **Explicitly Cast the Result:** If you want to store the result as an int (and you're okay with losing the fractional part), you can explicitly cast the double result to an int:

```
public class Main {  
    public static void main(String[] args) {  
        int num1 = 10;  
        double num2 = 5.5;  
        int result = (int) (num1 + num2); // Casting double to int  
        System.out.println(result); // Output will be 15  
    }  
}
```

O/P:



```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java  
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main  
15
```

In this case, (int) performs a type cast, truncating the decimal part of the double result and storing only the whole number in result.

2. **Change the Result Type to double:** If you want to keep the decimal precision, you can change the type of result to double:

```
public class Main {  
    public static void main(String[] args) {  
        int num1 = 10;  
        double num2 = 5.5;  
        double result = num1 + num2; // Result will be a double  
        System.out.println(result); // Output will be 15.5  
    }  
}
```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
15.5
```

In this case, the result of $\text{num1} + \text{num2}$ is a double, so result should also be of type double to avoid any loss of precision.

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?

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
2.0
```

Explanation:

1. Variable Declaration:

- `int num = 10;` declares an integer variable `num` and assigns it the value 10.
- `double result = num / 4;` declares a double variable `result`.

2. Division Operation:

- `num / 4` performs integer division, because `num` is an `int` and `4` is also an integer. In integer division, the result is the quotient without the decimal part. So, $10 / 4$ equals 2 (the fractional part .5 is discarded).

3. Result Assignment:

- The result of `num / 4` is 2 (integer), but you are assigning this to a double variable. Java will implicitly convert the integer 2 to 2.0 when storing it in the double variable. Hence, result becomes 2.0.

4. Output:

- The `System.out.println(result);` prints the value of `result`, which is 2.0 (since `result` is of type `double`).

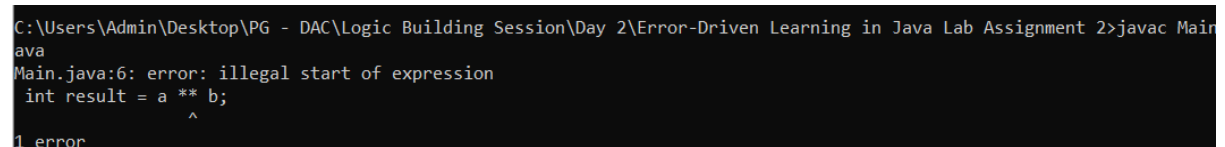
Note

Although `num / 4` is initially an integer operation, the division result is stored in a `double` variable. However, because the result of integer division (`10 / 4`) is 2, Java automatically converts it to 2.0 when assigning it to the `double` variable `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?



```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java  
Main.java:6: error: illegal start of expression  
    int result = a ** b;  
                   ^  
1 error
```

The `**` operator is **not valid** in Java because Java does not support exponentiation (raising a number to a power) using `**` syntax. In some programming languages like Python, `**` is used for exponentiation (e.g., `2 ** 3` results in 8), but **Java does not have the `**` operator** for exponentiation.

How to Perform Exponentiation in Java?

To perform exponentiation in Java, you can use the `Math.pow()` method, which is part of the Java standard library. The `Math.pow()` method takes two `double` arguments and returns the result of raising the first argument to the power of the second argument.

Here's how you can modify the code to compute `a` raised to the power of `b`:

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);    // Output will be 100000.0 (10^5)
    }
}

```

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
100000.0

```

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?

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
20

```

Operator Precedence:

- In Java, multiplication (*) has higher precedence than addition (+). This means that the multiplication operation will be performed first, before the addition operation.

- The general precedence of operators in Java follows the order:
 1. Multiplication (*), division (/), and modulus (%) have higher precedence than addition and subtraction.
 2. Addition (+) and subtraction (-) have lower precedence.

Step-by-Step Evaluation:

- $a = 10$
- $b = 5$
- 1. **First, the multiplication operation is performed because * has higher precedence than +:**
 - $b * 2 = 5 * 2 = 10$
- 2. **Then, the addition is performed:**
 - $a + 10 = 10 + 10 = 20$

Why the result is 20:

- The multiplication $b * 2$ is calculated first, resulting in 10.
- Then, the addition $a + 10$ is computed, resulting in 20.

If there is $(a+b)*2$ eg. $(10 + 5)*2$ $15*2= 30$

- Parentheses have the highest precedence in Java.

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?

O/P: error : in thread "main" java.lang.ArithmeticException: / by zero

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:5)
```

Java's int data type represents whole numbers, and when you try to divide an integer by zero, the JVM cannot compute a meaningful result.

a is 10, and b is 0.

When you attempt to divide 10 by 0, the JVM detects the illegal operation and throws an **ArithmeticException**.

If you're working with **floating-point numbers** (float or double), division by zero does not throw an exception.

Correct Code:

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

        if (b != 0) { // checks if b is zero before attempting to divide, avoiding the exception and
//giving a more meaningful message.
            int result = a / b;
            System.out.println(result);
        } else {
            System.out.println("Cannot divide by zero");
        }
    }
}
```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main.java
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Main
Cannot divide by zero
```

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?

In Java, each statement must be terminated with a **semicolon (;)**.

Correct Code:

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

o/p:

```
C:\Users\Admin\Desktop\PG -
Hello, World
```

Snippet 21:

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

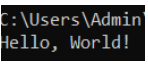
What does the compiler say about mismatched braces?

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac Main
ava
Main.java:6: error: reached end of file while parsing
    }
    ^
1 error
```

Correct Code:

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

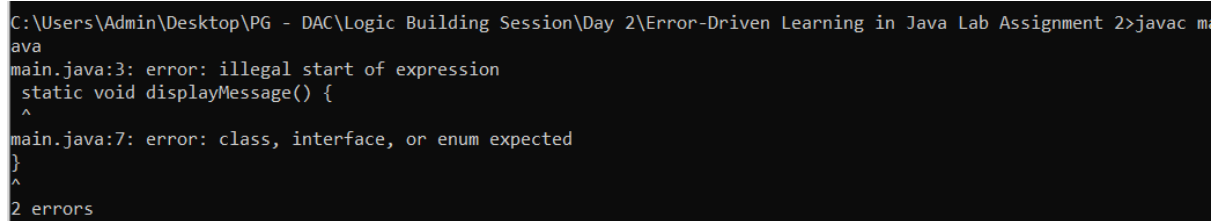
```
System.out.println("Hello, World!");
```

} // Closing brace added here } **o/p:**  To resolve the issue, you need to add the closing brace } for the main method at the correct place

Snippet 22:

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

o/p:



```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac m  
ava  
main.java:3: error: illegal start of expression  
    static void displayMessage() {  
    ^  
main.java:7: error: class, interface, or enum expected  
    }  
    ^  
2 errors
```

What syntax error occurs? Can a method be declared inside another method?

Can a Method Be Declared Inside Another Method?

No,

Why Does This Error Occur?

The error occurs because **methods cannot be declared inside another method** in Java. In Java, a method must be declared directly within a class or interface, not inside another method. Java does not allow a method declaration within the body of another method.

1. methods cannot be declared inside other methods. So, we need to move the displayMessage() method outside the main() method.
2. The static Keyword: The static keyword is fine as long as it is used at the class level and not within a method. We can keep static for the displayMessage() method.

Correcting the Code:

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



```

        displayMessage(); // Call the displayMessage method from within the main method
    }

    static void displayMessage() { // Method defined at the class level

        System.out.println("Message");

    }
}

```

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Ses
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");
        }
    }
}

```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>java Confusion
Value is 2
Value is 3
Default case
```

Error to Investigate: Why does the default case print after "Value is 2"?

The reason the default case prints after "Value is 2" is due to fall-through behavior in the switch statement.

When the value is 2, the program enters the case 2 block, and since there is no break statement after case 2, the program continues to execute the code for the next case (i.e., case 3) and then the default case.

This behavior occurs because, in Java, if there is no break statement at the end of a case, execution falls through to the next case, regardless of whether the value matches the condition or not. As a result:

1. The code enters case 2, prints "Value is 2".
2. Because of the fall-through, the program continues into case 3, prints "Value is 3".
3. The program continues again into the default case and prints "Default case".

How can you prevent the program from executing the default case?

To prevent the program from executing the default case, you can use a break statement after each case block to ensure that only the matching case is executed, and the switch block exits once that case is completed.

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; // Prevent fall-through

            case 2:

                System.out.println("Value is 2");

                break; // Prevent fall-through
```

```

        case 3:
            System.out.println("Value is 3");
            break; // Prevent fall-through
        default:
            System.out.println("Default case");
    }
}
}

```

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java
on
Value is 2

```

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");
        }
    }
}

```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Lo
BreakCase
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"?

The reason this code prints "Level 1", "Level 2", "Level 3", and "Unknown level" when level is 1 is due to the fall-through behavior in the switch statement. (And break statement is not used)

What is the role of the break statement in this situation?

The break statement is used to terminate the execution of the switch block once a matching case has been executed. Without the break statement, the program continues executing the following cases, regardless of whether they match the value.

In this situation **No break statement:** Since there is no break statement after case 1, the program falls through to the next case, even though level is not equal to 2. This means the code inside case 2 gets executed, and "Level 2" is printed.

Using break

```
public class MissingBreakCase {
    public static void main(String[] args) {
        int level = 1;
        switch(level) {
            case 1:
                System.out.println("Level 1");
                break; // Prevent fall-through
            case 2:
                System.out.println("Level 2");
                break; // Prevent fall-through
            case 3:
                System.out.println("Level 3");
                break; // Prevent fall-through
            default:
```

```

        System.out.println("Unknown level");
    }
}
}

```

O/P: Level 1 (msg is print Since the value of level matches case 1, the program executes the code inside this case, printing "**Level 1**".)

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!");
        }
    }
}

```

O/P:

```

C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assi
.java
Switch.java:5: error: incompatible types: possible lossy conversion from double to int
    switch(score) {
        ^
1 error

```

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?

- **switch limitations:** Java switch expressions are limited to certain types (e.g., int, String), and do not support double, float, or long.
- **Alternative for double:** For floating-point types like double, an if-else statement is typically used to handle conditional checks.
- **Type casting:** You can convert a double to an int (if appropriate) to use it in a switch statement, but this will lose the decimal part.
- **Option 1: Use an if-else statement**
Can replace the switch statement with an if-else block, which supports comparison with double values.

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

        if (score == 100) {
            System.out.println("Perfect score!");
        } else if (score == 85) {
            System.out.println("Great job!");
        } else {
            System.out.println("Keep trying!");
        }
    }
}
```

O/P:

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building
Great job!
```

Option 2: Convert the double to an int

If we want to stick with the switch statement, you can convert the double value to an int by casting it. This is only appropriate if you don't need to deal with fractional parts and just want to use the integer portion of the number.

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

        // Convert the double score to an int
        int intScore = (int) score;

        switch(intScore) { // Now using int, which is valid for switch
            case 100:
                System.out.println("Perfect score!");
                break;
            case 85:
```

```

        System.out.println("Great job!");
        break;
    default:
        System.out.println("Keep trying!");
    }
}
}
}
O/P:

```

```

C:\Users\Admin\Desktop\PG - DAO
Great job!

```

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");

        }

    }

}

O/P:

```

```
C:\Users\Admin\Desktop\PG - DAC\Logic Building Session\Day 2\Error-Driven Learning in Java Lab Assignment 2>javac S
.java
Switch.java:9: error: duplicate case label
case 5:
^
1 error
```

Error to Investigate: Why does the compiler complain about duplicate case labels?

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

This error means that you cannot have two case labels with the same value (in this case, both are 5) within the same switch statement.

What happens when you have two identical case labels in the same switch block?

When you have two identical case labels, the Java compiler will reject the code because it would create a situation where the program can't decide which case to execute when the value matches the duplicate case.

For example, if the value of number is 5, which case 5 should be executed? The compiler needs a clear answer, but with duplicate case labels, it can't determine which block should run, so it results in a compilation error.

Correct Code:

```
public class Switch {

    public static void main(String[] args) {

        int number = 5;

        switch(number) {

            case 5:

                System.out.println("Number is 5");

                break;

            case 10: // Change the value of the second case

                System.out.println("This is case 10");

                break;

            default:
```



```
System.out.println("This is the default case");
```

```
}
```

```
} } O/P: C:\Users\Admin\Desktop  
Number is 5
```