Lab Assignment 2

Section 1: Error-Driven Learning in Java

**Snippet 1:**

public class Main {

public void main(String[] args) {

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

}

}

Error: The issue in the Java code is that the method main is not declared as static

Soln:

public class Main {

public static void main(String[] args) {

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

}

}

This code will now run correctly and print Hello, World! to the console

**Snippet 2:**

public class Main {

static void main(String[] args) {

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

}

}

Error: In this public access modifier is missing. In Java the main method must be public so that the JVM can access it when starting the program.

Soln:

public class Main {

public static void main(String[] args) {

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

}

}

The program will run successfully and print Hello, World! to the console.

**Snippet 3:**

public class Main {

public int main(String[] args) {

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

}

}

Error: main method has an incorrect return type. In Java, the main method must have a return type of void.

Soln:

public class Main {

public static void main(String[] args) {

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

}

}

the program will run correctly and print Hello, World! to the console.

**Snippet 4:**

public class Main {

public static void main() {

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

}

}

Error: String[] args is missing.

It is a standard method signature in Java. The Java Virtual Machine (JVM) looks for this specific method signature to start program execution. The String[] args parameter allows the Java program to accept command-line arguments when it is executed.

Soln:

public class Main {

public static void main(String[] args) {

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

}

}

**Snippet 5:**

public class Main {

public static void main(String[] args) {

System.out.println("Main method with String[] args");

}

public static void main(int[] args) {

System.out.println("Overloaded main method with int[] args");

}

}

There is no issue in syntax.

Main method is overloaded by providing another version: public static void main(int[] args). Method overloading allows you to have multiple methods with the same name but different parameter lists in the same class. To call the main(int[] args) method we need to do it explicitly

**Snippet 6:**

public class Main {

public static void main(String[] args) {

int x = y + 10;

System.out.println(x);

}

}

Error:

The variable y is used in the expression int x = y + 10..but it has not been declared or initialized anywhere in the code before it is used. variables must be declared to tell the compiler what type of data the variable will hold.

Soln:

public class Main {

public static void main(String[] args) {

int y = 5; // Declaring and initializing y

int x = y + 10;

System.out.println(x); // This will print 15

}

}

**Snippet 7:**

public class Main {

public static void main(String[] args) {

int x = "Hello";

System.out.println(x){

}

}

Error: string value cannot be assigned to a variable declared as int. Java does not allow assigning a value of one type (like String) to a variable of another incompatible type (like int)

Type safety ensures that variables are used consistently with their declared types, preventing type-related errors. This makes the code more reliable and easier to debug by catching issues at compile time.

**Snippet 8:**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World!"

}

}

Error: “)” and “;” is missing at the end of System.out.println("Hello, World!"

Missing Parenthesis: The compiler will not be able to match the opening parenthesis ( with a closing one ), resulting in a syntax error because it expects the method call to be properly closed

Missing Semicolon: Java requires a semicolon to terminate statements. Without it, the compiler won't understand where the statement ends, leading to a compilation error.

**Snippet 9:**

public class Main {

public static void main(String[] args) {

int class = 10;

System.out.println(class);

}

}

Error:

The error occurs because class is a reserved keyword in Java, and reserved keywords cannot be used as variable names (identifiers). This rule prevents conflicts and maintains clarity in the language syntax.

**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: compilation error

When we compile and run the original code, it will fail due to the attempt to call non-static methods from a static context. Method overloading is allowed, but you need to use an instance of the class to call non-static methods.

Soln:

To fix the issue create an instance of Main and use that instance to call the non-static methods:

public class Main {

public void display() {

System.out.println("No parameters");

}

public void display(int num) {

System.out.println("With parameter: " + num);

}

public static void main(String[] args) {

Main obj = new Main(); // Create an instance of Main

obj.display(); // Call the non-static method

obj.display(5); // Call the overloaded non-static method

}

}

**Snippet 11:**

public class Main {

public static void main(String[] args) {

int[] arr = {1, 2, 3};

System.out.println(arr[5]);

}

}

Error: ArrayIndexOutOfBoundsException at runtime.

The array arr is initialized with three elements: {1, 2, 3}, meaning it has indices 0, 1, 2.

The code attempts to access arr[5], which is out of bounds since the array only has elements at indices 0, 1, and 2.

**Snippet 12:**

public class Main {

public static void main(String[] args) {

while (true) {

System.out.println("Infinite Loop");

}

}

}

Error: Infinite loop

So we can use a counter that increments with each iteration and exits the loop when it reaches a certain value.

int counter = 0;

while (counter < 10) {

System.out.println("Loop iteration: " + counter);

counter++;

}

Or we can also use a break statement inside the loop to exit based on a specific condition.

while (true) {

System.out.println("Loop iteration: " + counter);

if (counter == 9) {

break;

**Snippet 13:**

public class Main {

public static void main(String[] args) {

String str = null;

System.out.println(str.length());

}

}

Error: The code will throw a NullPointerException

This error occurs because you are trying to call a method (length()) on a null reference, which is not allowed in Java.

**Snippet 14:**

public class Main {

public static void main(String[] args) {

double num = "Hello";

System.out.println(num);

}

}

Error:

The error occurs because we are trying to assign a String value ("Hello") to a variable of type 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);

}

}

Error: The error occurs because we are assigning the result of int (num1) and a double (num2) to an int variable (result).

In Java, when you perform arithmetic operations between different data types (e.g., int and double), the result is automatically promoted to the more precise type, which in this case is double.

int num1 = 10;

double num2 = 5.5;

int result = (int) (num1 + num2); // Explicitly cast the result to int

System.out.println(result); // This will print 15 by ignoring decimals

}

}

**Snippet 16:**

public class Main {

public static void main(String[] args){

int num = 10;

double result = num / 4;

System.out.println(result);

}

}

The output will be 2.0

when you divide two integers, the result is also an integer. This means that the fractional part of the division is discarded.

**Snippet 17:**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = 5;

double result = a\*\*b;

System.out.println(result);

}

}

\*\* operator is not a valid operator in Java

We need to use the Math.pow() method which is a part of java.lang.Math class

Math.pow() method can handle both integer and floating-point exponentiation.

Soln:

double result = Math.pow(a, b);

System.out.println(result); // This will print 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);

}

}

Output: 20

multiplication operator (\*) has higher precedence than the 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);

}

}

Error: ArithmeticException . dividing an integer by zero is mathematically undefined and cannot be performed in programming.

**Snippet 20:**

public class Main {

public static void main(String[] args) {

System.out.println("Hello, World")

}

}

Error: semicolon is missing

The missing semicolon causes a syntax error because the Java compiler expects a semicolon to terminate the statement.

**Snippet 21:**

public class Main {

public static void main(String[] args) {

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

}

Error: Missing closing brace here

braces { } define the boundaries of code blocks, such as methods, classes, and loops.

**Snippet 22:**

public class Main {

public static void main(String[] args) {

static void displayMessage() {

System.out.println("Message");

}

}

}

Error: Methods must be declared at the class level and not within other methods.

Soln: public class Main {

public static void main(String[] args) {

// Call the displayMessage method from within main

displayMessage();

}

// Method declaration at the class level

static void displayMessage() {

System.out.println("Message");

}

}

**Snippet 23:**

public class Confusion {

public static void main(String[] args) {

int value = 2;

switch(value) {

case 1:

System.out.println("Value is 1");

case 2:

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

case 3:

System.out.println("Value is 3");

default:

System.out.println("Default case");

}

}

}

There is no break statement after System.out.println("Value is 2") so it also prints "Value is 3" and "Default case"

**Snippet 24:**

public class MissingBreakCase {

public static void main(String[] args) {

int level = 1;

switch(level) {

case 1:

System.out.println("Level 1");

case 2:

System.out.println("Level 2");

case 3:

System.out.println("Level 3");

default:

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

}

}

}

Error: absence of break

**Snippet 25:**

public class Switch {

public static void main(String[] args) {

double score = 85;

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: Allowed Types for switch: int, byte, short, char, and enum. Floating-point types like double are not allowed.

**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 6");

break;

default:

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

}

}

}

Error: case label is same.

In a switch statement each case label must be unique within the same switch block