Lab Assignment 2

Snippet 1:

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

- **What error do you get when running this code?**
- **Getting run time error-** main method should be static.

Error: Main method is not static in class Main, please define the main method as:

public static void main(String[] args)

Correced Code-

```
public class Main {
  public 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?**
- **Getting run time error-** main method should be public.

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

public static void main(String[] args)

Correced Code-

```
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?**
- **Getting run time error-** main method should be public.
- **return 0 - error:** incompatible types: unexpected return value**.**
- **why void -** The **main method is called** by the **JVM to start the program**, and it is responsible for initiating the application's execution. Since the purpose of this method is to start the program and not to return a value, **void is used to indicate that no value will be returned.**

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)

Correced Code-

```
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?**
- **Compile -** The code will compile successfully because the main method with no arguments is syntactically valid Java code.
- **Run - Error:** Main method not found in class Main, please define the main method as:

  public static void main(String[] args)
- The JVM expects the main method to have a specific signature

  public static void main(String[] args) , and it will not recognize

  public static void main() as a valid entry point.

This is a standard convention, and the String[] args parameter allows the JVM to pass command-line arguments to the program.

String[] args is used to receive command-line arguments, which can be provided by the user when running the program. This feature allows the program to be flexible and receive input at runtime.

Correced Code-

```
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");
  }
}
```

- **Can you have multiple main methods? What do you observe?**
- **Output** - Main method with String[] args
- Yes we can have multiple main methods. To achieve multiple main method we need to do overload main method.

The method public static void main(int[] args) is an overloaded version of the main method. It is not used by the JVM to start the application.

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?**
- **Error:** cannot find symbol, int x = y + 10; y is not declared.

  Why:

- **Type Safety:** Java is a statically-typed language, meaning that the type of each variable must be known at compile time. Declaring variables allows the compiler to check for type compatibility and ensures that operations on variables are valid.
- **Memory Allocation:** When you declare a variable, the Java compiler allocates memory space for that variable based on its type. If you try to use a variable without declaring it, the compiler doesn't know how much memory to allocate or how to handle the variable.

Corrected Code:

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

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?**
- **Compile Time Error - Error:** incompatible types: String cannot be converted to int

  int x = "Hello";

incompatible types because the given value is **String** and it's datatype is **integer.**

Why:

Type safety also plays a role in security. By preventing type-related errors.

Type safety ensures that variables and expressions behave in predictable ways.

Memory Safety: By enforcing type safety, Java ensures that variables are assigned only compatible types, preventing memory corruption or unpredictable behavior.

Preventing Errors at Compile Time: By enforcing type safety, Java ensures that many potential errors are caught during compilation rather than at runtime.

Corrected Code-

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

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?**
- **Compiler Time Error - Error**: ')' expected

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

Affect: These syntax errors will prevent the code from compiling successfully.

Corrected Code-

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

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?**
- **Compiler Time Error - Error**: not a statement int class = 10;

Reserved keywords are predefined in the Java language to perform specific functions. For example, class is used to define a new class. Allowing these keywords to be used as identifiers (like variable names) would create confusion for the compiler and make it difficult to understand the code correctly.

Corrected Code:

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

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?**
- **Compile Error- Error**: non-static method display() cannot be referenced from a static context.
- **Method overloading allowed -** Method overloading is allowed. To call non-static methods from a static context like main, we need to create an instance of the class and use that instance to call the 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?**
- **Runtime Error -** java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

This errors occurs because the given size of array is 3 but we are trying to access index 5 which is not there. To handle this exception, we can use if-else or we can access any of the index in between 0 to 2 instead of 5th index.

Corrected Code -

Approach 1 -

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

Approach 2 -

```
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 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?**
- Runs successfully but the program enters an infinite loop.
- **To avoid** - we have to add a condition to stop infinite loop or we can use break also with condition.

Corrected Code -

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

```
        int counter = 0;
        while (true) {
            System.out.println("Loop iteration: " + counter);
            counter++;
            if (counter >= 5) {
                break; // Exit 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?**
- **Runtime Error -** java.lang.NullPointerException
- Whenever you try to perform an operation on an object reference that is null. In this case, trying to call str.length() on a null reference results in this exception.
- We can initialize the variable with a default value instead of null ***(String str = "";)***. This way, the variable is never null, and you can safely call methods on it**.**

Corrected 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("The 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?**
- **Compile Error:** incompatible types: String cannot be converted to double.

Type constraints also plays a role in security. By preventing type-related errors.

Type constraints ensures that variables and expressions behave in predictable ways.

Memory Safety: By enforcing type constraints , Java ensures that variables are assigned only compatible types, preventing memory corruption or unpredictable behavior.

Preventing Errors at Compile Time: By enforcing type safety, Java ensures that many potential errors are caught during compilation rather than at runtime.

Correced Code -

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

```
        double num = 2.45; // Assign a double value
        System.out.println(num);
    }
}
```

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?**
- **Compile Error:** incompatible types: possible lossy conversion from double to int
- **Handling Different Data Types:**

  Use type casting when converting between types.

  Use the appropriate data type for the result to avoid loss of precision.

  Ensure that operations are performed with compatible types to maintain precision and correctness.

  Corrected Code -

```
public class Main {
    public static void main(String[] args) {
        int num1 = 10;
        double num2 = 5.5;
```

```
            int result = (int) (num1 + num2);
            System.out.println(result);
        }
}
```

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?**
- **Output -** 2.0
- **No, the Output May Not Be as Expected:** If you expected a floating-point result (such as 2.5), you need to perform floating-point division. To ensure that division yields a double result, at least one operand in the division should be a double.

Corrected Code -

```
public class Main {
public static void main(String[] args) {
    int num = 10;
            double result = (double) num / 4;
              System.out.println(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?**
- **Compile Time Error -** error: illegal start of expression

  int result = a ** b;

The ** operator is not recognized in Java.

Corrected Code-

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        double result = Math.pow(a, b);
        System.out.println(result);
    }
}
```

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?**

output is 20

Multiplication (*) has higher precedence than addition.

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?**
- Error- java.lang.ArithmeticException: / by zero
- Division by zero is mathematically undefined

Corrected Code-

```
public class Main {

    public static void main(String[] args) {

        int a = 10;

        int b = 0;

        if (b != 0) {

            int result = a / b;

            System.out.println(result);

        } else {

            System.out.println("Division by zero is not allowed.");

        }
```

```
    }
}
```

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?**
- **Error: ';' expected**

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

- In Java, every statement must end with a semicolon (;). This semicolon is used to indicate the end of a statement to the compiler.

Corrected Code-

```
public class Main {
public static void main(String[] args) {
System.out.println("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?**
- **Error**: reached end of file while parsing

Corrected Code-

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

```
}
```

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?**
- **Error: illegal start of expression**
- **In Java, you cannot declare a method inside another method.**

Corrected Code-

```
public class Main {

    public static void main(String[] args) {

        displayMessage();

    }

    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");
}
}
}
```

- **Error to Investigate: Why does the default case print after "Value is 2"? How can you prevent the program from executing the default case ?**
- There is no break statement at the end of case 2:, the program continues to execute the case 3: and default: cases.

Corrected Code-

```
public class Confusion {
    public static void main(String[] args) {
        int value = 2;
        switch(value) {
            case 1:
                System.out.println("Value is 1");
                break;
            case 2:
                System.out.println("Value is 2");
                break;
            case 3:
                System.out.println("Value is 3");
                break;
            default:
                System.out.println("Default case");
                break;
        }
    }
}
```

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?**
- The break statement is used to terminate a case within a switch statement.

Corrected Code-

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

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?**
- **Error**- incompatible types: possible lossy conversion from double to int

Corrected Code-

```
public class Switch {
```

```
	public static void main(String[] args) {
		int score = (int) 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!");
		}
	}
}
```

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?**
- **Error:** duplicate case label: 5
- When the compiler detects duplicate case labels, it cannot determine which block of code should execute when the switch expression matches the duplicate value.
- To resolve this error, we need to ensure that all case labels in a switch block are unique.

Corrected Code-

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

```
                // Removed the duplicate case 5
                default:
                    System.out.println("This is the default case");
            }
        }
}
```
