

Assignment – 2 Solution

Snippet1 –

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

- What error do you get when running this code?

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

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

Explanation - The method main should be public static void main(String[] args). The static keyword is necessary because the main method is called by the JVM without creating an instance of the class.

Corrected code –

```
public class Main {  
    public static 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?

The code will compile without any errors because the syntax is correct. But when we, try to run the program, it will not execute as expected.

The Java Virtual Machine (JVM) looks specifically for a public static void main(String[] args) method to start the execution of the program. Since the main method in your code is missing the public access modifier, the JVM will not recognize it as the entry point for the program.

Corrected 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?

The error occurs because the main method is incorrectly declared with a return type of int instead of void. The Java Virtual Machine (JVM) expects the main method to have: `public static void main(String args[])`. The void keyword indicates that the method does not return anything.

Corrected 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?

The code will compile successfully because there are no syntax errors. We will get error at runtime. If the main method doesn't include String args[], the JVM doesn't recognize it as the correct entry point and will throw an error.

Corrected 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?

Yes, you can have multiple methods named main in the same class, but they must differ in their parameter types. This is known as method overloading in Java.

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?

The error occurs because the variable `y` is used in the expression `int x = y + 10` is not declared or initialized. In Java, all variables must be declared before they are used. You need to declare and initialize the variable `y` before using it.

Corrected code –

```
public class Main {  
    public static void main(String[] args) {  
        int y = 11;  
        int x = y + 10;  
        System.out.println(x);  
    }  
}
```



↓ Click this button to add a new journal entry

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?

```
Main.java:3: error: incompatible types: String cannot be co  
nverted to int
```

```
    int x = "Hello";  
           ^
```

```
1 error
```

The Java language, by design, enforces type safety. It implies that Java prevents the programs from accessing memory in inappropriate ways by controlling the memory access of each object.

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?

```
Main.java:3: error: ')' expected
        System.out.println("Hello, World!"
                                ^
1 error
```

The program will not compile.

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?

```
Main.java:3: error: not a statement
        int class = 10;
        ^
Main.java:3: error: ';' expected
        int class = 10;
        ^
Main.java:3: error: expected
        int class = 10;
        ^
Main.java:4: error: expected
        System.out.println(class);
```

```

      ^
Main.java:4: error: illegal start of type
    System.out.println(class);
      ^
Main.java:4: error: expected
    System.out.println(class);
      ^
Main.java:6: error: reached end of file while parsing
}
^
7 errors

```


Keywords cannot be used as identifiers **for other purposes** (e.g. variable/function/method naming). Keywords are reserved words so cannot be used as identifier.

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?

```

Main.java:9: error: non-static method display() cannot be r
eferenced 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

```

Method Overloading: Method overloading allows multiple methods with the same name but different parameter lists (different number or types of parameters) within the same class. Here display() method is overloaded:

display() with no parameters.

display(int num) with one integer parameter.

Error in main Method:

The main method attempts to call display() and display(5) directly. However, these method calls are not associated with an instance of the Main class. In Java, you need to create an instance of the class to call non-static methods, or the methods themselves need to be declared as static if you want to call them directly from the main method.

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(); // Create an instance of the M
        obj.display();          // Call display() method on th
        obj.display(5);          // Call display(int) method on
    }
}

```

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?

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

The **ArrayIndexOutOfBoundsException** occurs whenever we are trying to access any item of an array at an index which is not present in the array.

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?

It will go into infinite loop. We can avoid infinite loop by adding break statement inside while loop. break keyword breaks the current iteration of the loop.

Corrected Code -


```

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

```

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?

Exception in thread "main" java.lang.NullPointerException
at Main.main(Main.java:4)

Null pointer exception is a runtime exception. Null is a special kind of value that can be assigned to the reference of an object. Whenever one tries to use a reference that has the Null value, the NullPointerException arise.

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?

In Java, double is a primitive data type that can only hold numeric values (specifically, floating-point numbers). The value "Hello" is a String, which is an entirely different data type. Java is strongly typed, meaning that you cannot assign a value of one type to a variable of another type without explicit conversion (and in this case, there's no valid conversion from String to 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?

```
Main.java:5: error: incompatible types: possible lossy conversion from double to int  
        int result = num1 + num2;  
                           ^  
1 error
```

int result = num1 + num2; attempts to add num1 (an int) and num2 (a double). The result of this addition will be a double because, in Java, operations

involving different numeric types will promote the smaller type (int in this case) to the larger type (double).

The result of `num1 + num2` is of type double, we are trying to store it in an int variable (result). Java does not allow implicit narrowing conversions, meaning we cannot assign a double value directly to an int variable without explicit casting.

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?

The output will be 2.0.


The expected the result to be 2.5, the output is not what we expected because the division was performed as integer division before being stored as a double.

We can get the expected result by

```
double result = (double) num / 4;
```

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?

```
Main.java:5: error: illegal start of expression  
        int result = a ** b;  
                        ^  
  
1 error
```

The ** operator is not valid in Java because Java does not have a built-in exponentiation operator. Instead, Java provides the `Math.pow()` method for performing exponentiation, which is consistent with Java's design philosophy of using methods for more complex operations.

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?

The Output of this code will be 20.

1. **Multiplication:** $b * 2$ is calculated first.

$b = 5$

$b * 2 = 5 * 2 = 10$

2. **Addition:** Next, the result of the multiplication is added to a.

$a = 10$

$a + 10 = 10 + 10 = 20$

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?

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

Snippet 20:

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

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

```
Main.java:3: error: ';' expected
        System.out.println("Hello, World")
                                ^
1 error
```

It causes syntax error and syntax errors leads to affect compilation.

Snippet 21:

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

Q What does the compiler say about mismatched braces?

```
Main.java:5: error: reached end of file while parsing
    }
    ^
1 error
```

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?

```

Main.java:3: error: illegal start of expression
    static void displayMessage() {
        ^
Main.java:7: error: class, interface, or enum expected
    }
    ^
2 errors

```

Methods in Java must be declared at the class level, not within other methods. Each method is defined separately and can be called within other methods as needed.

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?

output-

Value is 2

Value is 3

Default case

The default case print after "Value is 2" because there no break statement added inside switch case statement. To prevent the program from executing the default case, we need to write break keyword after each 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 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?

There is no break keyword added to terminate the iteration of the loop after execution of condition. Hence it print "Level 1", "Level 2", "Level 3", and "Unknown level". The break statement terminate the loop after execution of condition.

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?

The above code will not compile because, the switch statement in Java does not support the double type for the switch expression. The switch statement only supports specific data types such as int, char, String, and enum, but not double. Instead of switch we can use if-else statement to make this code 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");  
                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?

```
Switch.java:8: error: duplicate case label  
    case 5:  
      ^  
1 error
```

when you have two identical case labels in the same switch block, it will give error: duplicate case label. The compiler gets confuse, which statement to execute.