Assignment 2 – Day 2

Section 1: Error-Driven Learning in Java

Snippet 1:

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

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

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

Explanation :

Error is that the main method don’t have static keyword. The main() method is declared static so that JVM can call it without creating an instance of the class containing the main() method.

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!");
 } }
```

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

Explanation:

Public keyword is missing in the above code. If it's not public, then it won't be found. Making the main() method public makes it globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.

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;
 }
}
```

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)

Explanation:

As the *main()* method doesn’t return anything, its return type is **void**. 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.
2. Using int signature would cause confusion and compatability issues while program execution.

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!"); } }
```

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

Explanation :

Even when we have nothing to pass, it is important to write “String args[]” as a parameter as it is a part of the syntax in Java, and the program won't run without it.

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");
 }
}
```

Output: Main method with String[] args

Explanation :

Having multiple main methods can not show any error but it wont execute the methods except public static void main(String args[]). If you try to add multiple methods with arguments String args[] it will throw error like JVM only looks for main(String args[]) as starting point. Other main methods just regular static methods and must be called explicitely within the program.

Snippet 6:

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

Error : Main.java:4: error: cannot find symbol

```
int x = y + 10;
        ^
 symbol:   variable y
location: class Main
```

Explanation : Compiler can't find a declaration for that variable within the current scope. Also if we are assigning its value to another it should contain some value.

Corrected code:

```
public class Main {
 public static void main(String[] args) {
 int x ,y=2;
 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);
}
}
```

```
Error: Main.java:4: error: incompatible types: String cannot be converted to int
                int x = "Hello";
                        ^
               1 error
```

Explanation: Int identifier cannot hold string value.

<u>Why does Java enforce type safety?</u>

Java enforces type safety to ensure that data is used in a way that prevents errors and makes programs more predictable.

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!"
 }
}
```

```
Error: ')' expected
 System.out.println("Hello, World!"
                                    ^
```

Explanation: Syntax errors will prevent compilation, so always check for missing parentheses.

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);
 }
}
```

Errors:

```
Main.java:3: error: not a statement
 int class = 10;
 ^
Main.java:3: error: ';' expected
```

```
 int class = 10;
    ^
Main.java:3: error: <identifier> expected
 int class = 10;
          ^
Main.java:4: error: <identifier> expected
 System.out.println(class);
                   ^
Main.java:4: error: illegal start of type
 System.out.println(class);
                    ^
Main.java:4: error: <identifier> expected
 System.out.println(class);
                         ^
Main.java:6: error: reached end of file while parsing
}
 ^
7 errors
```

Explanation: We cannot use keywords like ‘class’ as identifier, Because these words have predefined meanings in the Java programming language. Naming conventions has some rules we must follow that.

Corrected code:

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

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);
 }
}
```

Errors:

```
Main.java:9: error: non-static method display() cannot be referenced from a static context
 display();
 ^
Main.java:10: error: non-static method display(int) cannot be referenced from a static context
 display(5);
 ^
2 errors
```

Snippet 11:

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

Errors:

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

Explanation: accesing 5th index when we only have 3 values in array.

Corrected code:

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

Snippet 12:

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

Explanation: It prints infinite times “Infinite Loop”. The condition in the while loop is true, which never changes also there is no exit condition or break statement. To avoid infinite statements include break, or give specific condition which will be true once.

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());
 }
}
```

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

Explanation: In this code String literal has identifier str with null value. when a variable of type String (or any object) is assigned null, it means that the variable does not reference any memory location where an actual object exists. Since null represents the absence of an object, trying to access methods or properties of a null object leads to a NullPointerException (NPE).

Corrected code:

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

Snippet 14:

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

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

```
          ^
```

Explanation:

Double num expects a numeric (floating-point) value. Java does not automatically convert a String to a double because they are fundamentally different types. Also java is a strongly typed language, meaning every variable must have a defined type.

Corrected code :

```
public class Main {
 public static void main(String[] args) {
 String num = "Hello";
 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);
 }
}
```

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

Explanation: When adding int and double, Java automatically promotes int to double (implicit conversion). The result (15.5) is of type double, but the result variable is declared as int. Java does not allow automatic conversion from double to int because it could lose data (decimal part is removed).

Corrected code:

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

<u>Snippet 16:</u>

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

Output: 2.0

Explanation:

Integer Division Happens First (num / 4)

- num = 10 (an int).
- 4 is also an int.
- When dividing two int values, Java performs integer division, meaning it discards the decimal part.
- 10 / 4 results in 2 (instead of 2.5).

Implicit Conversion to double Happens After Division

- The result of 10 / 4 is 2 (an int).
- This 2 is then assigned to result, which is of type double.
- Java automatically converts 2 to 2.0, but the decimal part is already lost

Snippet 17:

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

```
error: illegal start of expression
 int result = a ** b;
                ^
```

Explanation : Java does not support the ** operator for exponentiation (unlike Python, which uses ** for power calculations). ** is not a valid Java operator, so the compiler does not understand it. Use Math.pow(base, exponent) for power calculations.

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);

 }

}
```

Output: 20

Explanation : Operator precedence will affect the output like it will first evaluate * then +.

- ➔ a+b*2
- ➔ 10+5*2
- ➔ 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);

 }

}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero

    at Main.main(Main.java:5)
```

Explanation : It shows runtime exception because dividing by zero is mathematically undefined, and Java treats this operation as an error, causing the program to crash when it tries to execute it. We can correct it by using exception handling concept.

Snippet 20:

```
public class Main {

 public static void main(String[] args) {

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

 }

}
```

```
Error: ';' expected
```

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

Explanation: In Java, each statement must end with a semicolon (;). Java’s compiler expects a semicolon to mark the end of the System.out.println() statement. Without it, the compiler doesn't know where the statement ends, causing an error.

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

Explanation: Every opening brace { must have a corresponding closing brace }. The main method { is opened but never closed, leading to an error. A missing closing brace causes a reached end of file while parsing error.

Corrected code:

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

Snippet 22:

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

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

Explanation: In Java, methods cannot be declared inside another method. The displayMessage() method is defined inside main(), which is not allowed.

Corrected code:

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

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");
 }
 }
}
```

Output : Value is 2

Value is 3

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?

Explanation: We need to use break statement to break the flow if it matches. This is how we can avoid default case execution.

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");
 }
 }
}
```

Output:

```
Level 1

Level 2

Level 3

Unknown level
```

Error to Investigate: When level is 1, why does it print "Level 1", "Level 2", "Level 3", and "Unknown level"? What is the role of the break statement in this situation?

Explanation: Break statement stops execution when case matches. The switch statement starts execution from the matching case (case 1). However, there are no break statements, so it continues executing all subsequent cases, including case 2, case 3, and default.

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");
 }
 }
}
```

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: incompatible types: possible lossy conversion from double to int
 switch(score) {
       ^
```

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?

Explanation: byte, short, char, int these are the only datatypes case we can pass to switch. double values are stored in binary form, making exact comparisons unreliable in a switch statement.

Corrected code:

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

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: duplicate case label
 case 5:
```

Explanation:

In a switch statement, each case label must be unique within the same switch block. If two cases have the same value, the compiler throws an error. If case labels were allowed to be duplicated, Java wouldn't know which case to execute for num = 5. To avoid ambiguity, the compiler stops compilation and throws an error before running the program.

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;
 case 6:
 System.out.println("This is another case 5");
 break;
 default:
 System.out.println("This is the default case");
 }
 }
}
```