Assignment 2

Lakshit Dogra - KH

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:

This error occurs in Java when the main method in your class isn't defined as "public static void main(String[] args)". This specific signature is required by Java to start the program.

- public: makes it accessible.
- static: allows it to run without creating an instance of the class.
- void: means it doesn't return anything.
- main: is the method name the JVM looks for.

Correction:

```
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:

The error means the main method is missing or incorrectly defined.

The main method should be written as 'public static void main(String[] args)'. If it's missing or incorrectly written (e.g., no public), the JVM can't find it to start the program.

Correction:

```
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:

The error occurs because the 'main' method is not returning 'void'. In Java, the 'main' method must have a return type of 'void', meaning it doesn't return anything.

Correction:

```
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:

The error occurred because the Java main method has a specific signature that must be followed in order to be recognized as the entry point of the program. the main method without String[] args will cause this error as Java won't be able to recognize it as the entry point for the application and will throw an error like the one you encountered, because the JVM (Java Virtual Machine) expects the exact public static void main(String[] args) signature to start the program.

Correction:

```
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");

   }

}
```

Explanation:

When you run your program, Java will use the **public static void main(String[] args)** method because that's the standard entry point. The main(int[] args) method will **not** be invoked automatically by the JVM, because the JVM doesn't know about it when starting the application.

Corrections:

You can have multiple main methods, but only one will be called by the JVM when the program starts: the one with the String[] args parameter.

The overloaded main(int[] args) method will never be called automatically.

You could manually call the overloaded main method from within the program, like this:

```
public class Main {
   public static void main(String[] args) {
      System.out.println("Main method with String[] args");
      main(new int[] { 1, 2, 3 });  // Manually calling the overloaded main method
   }

   public static void main(int[] args) {
      System.out.println("Overloaded main method with int[] args");
   }
}
```

Snippet 6:

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

Error:

```
Main.java:49: error: cannot find symbol
        int x = y + 10;
                ^
  symbol:   variable y
  location: class Main
1 error
```

Explanation:

The error indicates that the variable y is being used in the expression y + 10, but it hasn't been declared or initialized in the Main class before being used.

Correction:

```
public class Main {
    public static void main(String[] args) {
        int y = 5;
        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);
      }
}
```

Error:

```
Main.java:58: error: incompatible types: String cannot be converted to int
      int x = "Hello";
               ^
1 error
```

Explanation:

The error occurs because we're trying to assign a String (which is "Hello") to a variable of type int. In Java, we cannot assign a String directly to an int variable because they are different data types.

Correction:

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

Snippet 8:

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

Error:

```
Main.java:67: error: ')' expected
        System.out.println("Hello, World!"
                                          ^
1 error
```

Explanation:

The error you're seeing is because the closing parenthesis ‘)’ is missing in the System.out.println() statement. Every method call in Java must have matching parentheses. Also, the ‘;’ token is missing at the end of the SOP statement.

Correction:

```
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);
    }
}
```

Error:

```
Main.java:75: error: not a statement
        int class = 10;
             ^
Main.java:75: error: ';' expected
        int class = 10;
           ^
Main.java:75: error: <identifier> expected
        int class = 10;
                 ^
Main.java:76: error: <identifier> expected
        System.out.println(class);
```

Explanation:

The issue with the code lies in the use of the keyword class as a variable name. In Java, class is a reserved keyword, meaning it has a special purpose in the language (for defining classes), and we cannot use it as an identifier (like a variable name).

Correction:

```
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);
  }
}
```

Error:

```
Main.java:90: error: non-static method display() cannot be referenced from a static context
    display();
    ^
Main.java:91: error: non-static method display(int) cannot be referenced from a static context
    display(5);
    ^
2 errors
```

Explanation:

The issue you're encountering is due to trying to call non-static methods (display(), display(int)) from the static main method. In Java, non-static methods must be called on an instance of the class, not from a static context directly.

To fix this, you can either:

1. **Make the display() method static**, so it can be called from the static main method, or
2. **Create an instance of the Main class** in the main method and call the non-static display() methods on that instance.

Snippet 11:

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

Error:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of
bounds for length 3
      at Main.main(Main.java:100)
```

Explanation:

In the code, we are trying to access arr[5] in the array arr, which has only 3 elements (indexed from 0 to 2). Since the array only has indices 0, 1, and 2, attempting to access arr[5] results in an ArrayIndexOutOfBoundsException. To avoid the ArrayIndexOutOfBoundsException, ensure that the index we're accessing is within the valid range of the array. In our case, we should access indices 0, 1, or 2, as those are the valid ones for an array of length 3.

Correction:

```
public class Main {
   public static void main(String[] args) {
      int[] arr = {1, 2, 3};
      System.out.println(arr[2]); // Accessing valid index (2) will print "3"
   }
}
```

Snippet 12:

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

Execution:

When we run the code, The while (true) condition is always true, so the loop keeps executing indefinitely. The line System.out.println("Infinite Loop"); keeps printing "Infinite Loop" to the console repeatedly. This will continue until you manually stop the program (for example, by pressing Ctrl+C in the terminal or stopping it in your IDE).

Correction:

```
public class Main {
   public static void main(String[] args) {
      int a = 5;
      while (a>0) {
         System.out.println("Infinite Loop");
         a--;
      }
   }
}
```

Snippet 13:

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

Exception:

```
Exception in thread "main" java.lang.NullPointerException
        at Main.main(Main.java:119)
```

Explanation:

The issue we're encountering is a **NullPointerException** because we're trying to call the length() method on a String that is null.

The variable str is **initialized to null**. When we try to call str.length(), Java attempts to access the length() method of str. Since str is null, it does not point to any valid object, leading to a **NullPointerException**.

Correction:

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

Snippet 14:

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

Error:

```
Main.java:127: error: incompatible types: String cannot be converted to double
        double num = "Hello";
                     ^
1 error
```

Explanation:

The error occurs because you are trying to assign a String value ("Hello") to a double variable (num). In Java, a String cannot be directly converted to a double unless you explicitly parse it, and the value "Hello" cannot be converted to a numeric type like double.

Correction:

```
public class Main {
    public static void main(String[] args) {
        double num = 5.5;
        String grt = “Hello”;
        System.out.println(num);
        System.out.println(grt);
    }
}
```

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:

```
Main.java:138: error: incompatible types: possible lossy conversion from double to int
        int result = num1 + num2;
                          ^
1 error
```

Explanation:

The error occurs because we're trying to assign the result of the expression num1 + num2 (which is a double) to a variable result of type int. In Java, we cannot directly assign a double value to an int because it involves a **lossy conversion** (i.e., the decimal part would be lost).

Correction:

```
public class Main {
    public static void main(String[] args) {
        int num1 = 10;
        double num2 = 5.5;
        double result = num1 + num2; // result is now of type double
        System.out.println(result); // Output: 15.5
    }
}
```

Snippet 16:

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

Output:

```
2.0
```

Desired output:

```
2.5
```

Explanation:

The variable num is an int, and 4 is also an int. When you divide two integers in Java (i.e., num / 4), the result will also be an integer. Java **performs integer division** here, meaning it discards the decimal part and only keeps the whole number part of the result.

So, 10 / 4 is 2 because integer division truncates the result (the decimal part 0.5 is discarded).

Correction:

```
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);
    }
}
```

Error:

```
Main.java:159: error: illegal start of expression
        int result = a ** b;
                        ^
1 error
```

Explanation:

The error occurs because the operator ** is not valid in Java. Specifically, Java does not have a built-in exponentiation operator like **. In many other programming languages (such as Python), ** is used for exponentiation (raising a number to a power), but in Java, you'll need to use a different method to perform exponentiation.

Correction:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        int 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:

In the given Java code, we're performing an arithmetic operation where we calculate a + b * 2. Here's how the expression is evaluated based on operator precedence:

- The multiplication (*) operator has higher precedence than the addition (+) operator.
- Therefore, b * 2 is calculated first, then a is added to the result.

Breakdown of the code:

- a = 10
- b = 5
- b * 2 = 5 * 2 = 10
- a + (b * 2) = 10 + 10 = 20

hence the output is 20.

Snippet 20:

```
public class Main {

    public static void main(String[] args) {

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

    }

}
```

Error:

```
Main.java:179: error: ';' expected

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

                                       ^

1 error
```

Explanation:

The semicolon (;) in Java is important because it serves as a **statement terminator**. In Java, each statement needs to be terminated with a semicolon to indicate where the statement ends. Without it, the compiler cannot determine where one instruction ends and another begins.

Correction:

```
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:189: error: reached end of file while parsing

}

 ^

1 error
```

Explanation:

The error in the code is caused by a **missing closing brace** } at the end of the main method. In Java, each block of code, such as methods and loops, must be enclosed in curly braces { }.

In the code, we've opened the main method with {, but we did not provide the closing brace } for it. As a result, the compiler doesn't know where the main method ends, leading to a **syntax error**.

Correction:

```
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");

        }

    }

}
```

Error:

```
Main.java:195: error: illegal start of expression

        static void displayMessage() {

        ^
```

Explanation:

The error you're encountering happens because the static void displayMessage() method is **defined inside the main method**, which is not allowed in Java. In Java, you cannot define methods inside other methods. Methods must be defined directly inside a class, not inside other methods.

Correction:

```
public class Main {

    static void displayMessage() {

        System.out.println("Message");

    }

    public static void main(String[] args) {

        displayMessage();

    }

}
```

Snippet 23:

```
public class Main {

    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
```

Explanation:

In the Java code, the default case executes after "Value is 2" because **Java's switch statement does not automatically break after each case block**. Without a break statement, the program continues to execute the subsequent case blocks, a behavior known as "fall-through."

Snippet 24:

```
public class Main {

    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
```

Explanation:

In the Java code, when `level` is set to `1`, the program prints "Level 1", "Level 2", "Level 3", and "Unknown level" because **the `switch` statement lacks `break` statements** at the end of each `case` block. In Java, **without a `break` statement**, the program continues to execute the subsequent `case` blocks—a behavior known as **fall-through**.

Snippet 25:

```
public class Main {

    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:

```
Main.java:242: error: incompatible types: possible lossy conversion from double to int

        switch(score) {

               ^
```

Explanation:

The error in your code arises because **Java's switch statement does not support double or float types**. The switch statement in Java is designed to work with certain data types, and double is not among them.

Snippet 26:

```
public class Main{

    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:

```
Main.java:265: error: duplicate case label

            case 5:

            ^
```

Explanation:

The error in your code occurs because **Java does not allow duplicate case labels within a switch statement**. Each case label must be unique to ensure that the switch statement can unambiguously determine which block of code to execute.