Snippet 1 →

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

Here the main method not found error will come and to correct this we need to add static keyword.

Snippet 2 →

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


}
```

Main method not found in class snippet to correct this add pubic keyword.

Snippet 3 →

```
class snippet{
public static void main(String[] args) {
 System.out.println("Hello, World!");
 return 0;
 }.
```

}

Return 0 statement must be removed and in place of int keyword void should be there

The void keyword in the main method of a Java program is used to indicate that the method does not return any value. In Java, the main method serves as the entry point for the program.

Snippet 4 →

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

Error : main method not found

Solution : add String args[] in main function

String args[] → is needed to pass command-line arguments to the program.

Snippet 5 →

```
class snippet{
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");
 }

}
```

Yes we can have multiple methods named main in a single class but JVM will only recognize the standard main method with the String args[] parameter.

Output →Main method with String[] args

Snippet 6 →

```
class snippet{
public static void main(String[] args) {
int y= 0;     // declare y and initialized y with 0
int x = y + 10;
 System.out.println(x);
 }

}
```

Error : y is not declared and initialized

Output → 10

Snippet 7 →

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

Error : - String cannot be converted to int

Solution inplace of int use String

Java enforce type safety to prevent runtime errors

Snippet 8 →

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

}
```

Error : -) and ; missing

Output → Hello, World!

Snippet 9 →

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

Error : - not a statement and illegal start of expression

Reserved keyword like class cannot be used as an identifier because they have specific meaning in java

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) {
      Main obj = new Main();  // Create an instance of Main
      obj.display();          // Call instance method with no parameters
      obj.display(5);         // Call instance method with an int parameter
   }
}
```

Error :- The main method attempts to call display() and display(5), but these calls are made from a static context. The display methods are instance methods, which means they belong to an instance of the Main class, not to the class itself.

Java allows method overloading, meaning you can have multiple methods with the same name but different parameter lists in the same class.

Snippet 11 →

```
class snippet{
 public static void main(String[] args) {
 int[] arr = {1, 2, 3};
 System.out.println(arr[5]);  // 5 index is not present in array
 }
}
```

Error:- “ArrayIndexOutOfBoundsException”

The index 5 is outside the bounds of this array. Valid indices for accessing elements in the array are from 0 to arr.length - 1 (i.e., 0 to 2 in this case).

Snippet 12 →

```
class snippet{
public static void main(String[] args) {
int count = 0;
 while (count<10) {
 System.out.println("Infinite Loop");
Count++;
 }
 }
}
```

This program will print “Infinite Loop” infinite number of times because condition under while always evaluates to true.

In order to avoid infinite loop a finite condition should be present inside while condition.

Snippet 13 →

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

Error:- “NullPointerException”

In the line String str = null;, the variable str is explicitly assigned null. This means that str does not reference any String object; it is a null reference. The statement System.out.println(str.length()); attempts to call the length() method on the str variable.

Since str is null, there is no actual String object on which to call length(). Attempting to call a method on a null reference leads to a NullPointerException because you are trying to access a method or field of an object that does not exist.

Snippet 14 →

```
class snippet{

public static void main(String[] args) {

 double num = "Hello";

String str = "Hello";


 System.out.println(str);

 }




}
```

Error :- String cannot be converted to double.

Java Enforce Data Type Constraints to prevent runtime errors

Snippet 15 →

```
public class Main {

   public static void main(String[] args) {

       int num1 = 10;

       double num2 = 5.5;

      int result = num1 + num2;

int result = (int) (num1 + num2);

       System.out.println(result);
```

```
    }
}
```

Error:- possible lossy conversion from double to int

This error occurs because num1(int) + num2 (double) → result should be in double but here trying to assign the result with int data type

Using explicit type conversion we can covert the double result to int .

Snippet 16 →

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


}
```

Output → 2.0

Yes , because the final result is stored as a double that's why a decimal value is obtained

Snippet 17 →

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

```
double result = Math.pow(a, b);
        System.out.println(result);
    }
}
```

Error : - The ** operator is not a valid operator in Java.

Solution :- Math.pow() function should be used to find exponential

Snippet 18 →

```
class snippet{
  public static void main(String[] args) {
 int a = 10;
 int b = 5;
 int result = a + b * 2;
 System.out.println(result);
 }


}
```

Output → 20

In the expression a + b * 2, multiplication is performed before addition due to its higher precedence, resulting in 10 + 10, which equals 20.

Snippet 19 →

```
class snippet{
 public static void main(String[] args) {
 int a = 10;
 int b = 0;
 int result = a / b;
 System.out.println(result);
 }



}
```

Error:- “ArithmeticException”

In Java, attempting to divide an integer by zero triggers an ArithmeticException. This exception is thrown because division by zero is mathematically undefined and cannot be performed in standard arithmetic.

Snippet 20 →

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


}
```

Error : - ; expected

Semicolon ; is missing until a semicolon is not placed at every statement end compiler cant execute the expression.

Snippet 21→

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

The compiler will show error:reached end of the file while parsing

Snippet 22 →

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

   }
}
```

Results in syntax error

In Java, you cannot declare a method inside another method. The Java language specification does not allow methods to be nested within other methods. Methods must be declared at the class level, not within the body of other methods.

The static modifier is used to declare static methods, but it is not valid to use it inside another method. It should be used at the class level.

Snippet 23 →

```
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;
 }
 }
}
```

The default case is printed because there is no break statement after each case.

So in-order to prevent program from executing the default case break statement should be use 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");
 break;
 case 2:
 System.out.println("Level 2");
break;
 case 3:
 System.out.println("Level 3");
break;
 default:
 System.out.println("Unknown level");
break;
 }
 }
}
```

The default case and other cases is printed because there is no break statement after each case.

So in-order to prevent program from executing the default case and other cases break statement should be use after each case.

Break statement terminates the flow of execution once a condition evaluates to true and bring the cursor out the program.

Snippet 25 →

```
public class Switch {
   public static void main(String[] args) {
      int score = 85;  // Use an int type for the score instead of double
      switch(score) {
         case 100:
            System.out.println("Perfect score!");
            break;
         case 85:
            System.out.println("Great job!");
            break;
         default:
            System.out.println("Keep trying!");
      }
   }
}
```

Java's switch statement does not support the use of double (or any floating-point type) as the expression type. The switch statement only supports byte, short, char, int, enum, String, and some wrapper classes like Character, Integer, etc.

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;
```

```
        // You can add other unique cases if needed
        // case 6:
        //     System.out.println("Number is 6");
        //     break;
        default:
            System.out.println("This is the default case");
    }
  }
}
```

The error occurs because you have two identical case labels (case 5:) in the switch statement. In Java, each case label within a switch block must be unique. Having duplicate case labels is not allowed and causes a compilation error.