Snippet 1

Error: main method is not static.
Correction:

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

Snippet 2

Error: main method must be public static.
Correction:

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

Snippet 3

Error: main method must have a void return type.
Correction:

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

Snippet 4

Error: main method must have String[] args as a parameter.
Correction:

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

Snippet 5

Observation: Java **allows method overloading, but only the main(String[] args) method is used as the entry point.**
Correction:

```
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");
  }
}
```

This will execute main(String[] args) while main(int[] args) will not be called automatically.

Snippet 6

Error: y is not declared.
Correction:

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

```
    int y = 5;  // Declare y before using it
    int x = y + 10;
    System.out.println(x);
  }
}
```

Snippet 7

Error: Type mismatch (int x = "Hello").
Correction:

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

Snippet 8

Error: Missing closing parenthesis) in System.out.println("Hello, World!".
Correction:

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

Snippet 9

Error: class is a reserved keyword.
Correction:

```
public class Main {
```

```
  public static void main(String[] args) {
    int myClass = 10;  // Use a valid identifier
    System.out.println(myClass);
  }
}
```

Snippet 10

Error: display() is not static, so it cannot be called inside main().
Correction:

```
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 object to call non-static methods
    obj.display();
    obj.display(5);
  }
}
```

Method overloading is allowed, and now both methods will work correctly.

Snippet 11

Error: ArrayIndexOutOfBoundsException occurs because index 5 is out of bounds.
Correction:

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

```
    int[] arr = {1, 2, 3};
    System.out.println(arr[2]); // Access valid index
  }
}
```

Java arrays are zero-based, meaning valid indices for arr are 0, 1, 2.

Snippet 12

Observation: This code runs an infinite loop.
Solution to avoid infinite loops:

```
public class Main {
  public static void main(String[] args) {
    int i = 0;
    while (i < 5) { // Condition to exit loop
      System.out.println("Loop iteration: " + i);
      i++;
    }
  }
}
```

Always ensure loops have an exit condition.

Snippet 13

Error: NullPointerException occurs because str is null.
Correction:

```
public class Main {
  public static void main(String[] args) {
    String str = "Hello"; // Assign a valid string
    System.out.println(str.length());
  }
}
```

Null values should be checked before accessing methods like length().

Snippet 14

Error: Type mismatch (double cannot be assigned a String).
Correction:

```
public class Main {
  public static void main(String[] args) {
    double num = 10.5; // Assign a valid double value
    System.out.println(num);
  }
}
```

Java enforces type safety to prevent unintended operations.

Snippet 15

Error: Type mismatch (num1 + num2 results in a double, which cannot be assigned to int).
Correction:

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

Mixed-type arithmetic operations result in the higher precision type (double).

Snippet 16

Error: Integer division occurs (num / 4 results in an integer before assigning to double).
Correction:

```
public class Main {
  public static void main(String[] args) {
    int num = 10;
    double result = num / 4.0; // Use 4.0 to force floating-point division
    System.out.println(result);
  }
}
```

Integer division truncates decimal places. Use floating-point numbers for accurate division.

Snippet 17

Error: ** operator is not valid in Java (use Math.pow()).
Correction:

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

*Java does not support ** for exponentiation*.

Snippet 18

Observation: Operator precedence applies (* has higher precedence than +).
Correction (no error, but clarifying precedence):

```
public class Main {
  public static void main(String[] args) {
    int a = 10;
    int b = 5;
    int result = a + (b * 2); // Multiplication happens first
    System.out.println(result); // Output: 20
  }
}
```

To ensure clarity, use parentheses () when necessary.

Snippet 19

Error: ArithmeticException: / by zero.
Correction:

```
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("Cannot divide by zero.");
    }
  }
}
```

Always check for zero before division to avoid runtime exceptions.

Snippet 20

Error: Missing semicolon ; at the end of System.out.println("Hello, World").
Correction:

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

A missing semicolon causes a compilation error.

Here are the corrected versions of the code snippets along with explanations of the errors:

Snippet 21

Error: Missing closing brace (}) results in a compilation error:
Correction:

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

Java requires properly matched braces to define code blocks. The compiler will complain about "reached end of file while parsing".

Snippet 22

Error:

- static is not allowed inside another method.
- A method cannot be declared inside another method in Java.

Correction:

```
public class Main {
```

```
  public static void displayMessage() { // Moved outside main method
    System.out.println("Message");
  }

  public static void main(String[] args) {
    displayMessage(); // Calling the method
  }
}
```

Methods must be defined at the class level, not inside other methods.

Snippet 23

Error:

- **Missing break statements** cause "fall-through" execution.
- The default case executes because there is no break.

Correction:

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

```
        default:
            System.out.println("Default case");
    }
  }
}
```

The break statement prevents execution from continuing into the next case.

Snippet 24

Error:

- **Missing break statements** cause all cases after the matching one to execute.

Correction:

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

}

The break statement stops execution of remaining cases.

Snippet 25

Error:

- switch does **not** support double values.
- The compiler throws: Cannot switch on a value of type double.

Correction:

```
public class SwitchExample {
  public static void main(String[] args) {
    int score = 85; // Changed type to int
    switch(score) {
      case 100:
        System.out.println("Perfect score!");
        break;
      case 85:
        System.out.println("Great job!");
        break;
      default:
        System.out.println("Keep trying!");
    }
  }
}
```

switch supports byte, short, char, int, String, and enum but not double.

Snippet 26

Error:

- **Duplicate case labels (case 5: appears twice)** cause a compilation error: Duplicate case label.

Correction:

```
public class SwitchExample {
  public static void main(String[] args) {
    int number = 5;
    switch(number) {
      case 5:
        System.out.println("Number is 5");
        break;
      case 6: // Changed duplicate case 5 to a different value
        System.out.println("This is another case");
        break;
      default:
        System.out.println("This is the default case");
    }
  }
}
```

A switch block cannot have duplicate case labels. Each case must be unique.