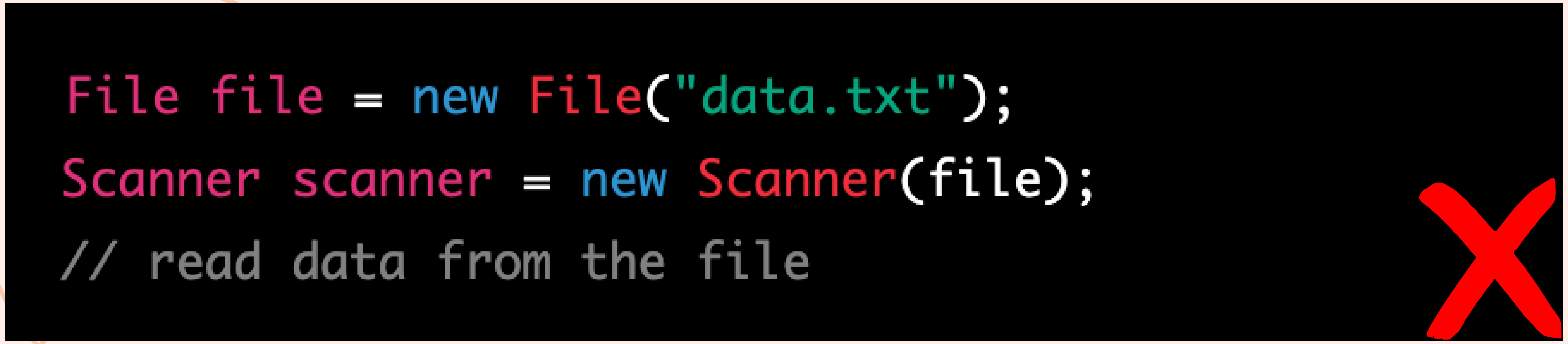
Java
Common Mistakes

Top 10 Java Developer Mistakes and How to Fix Them

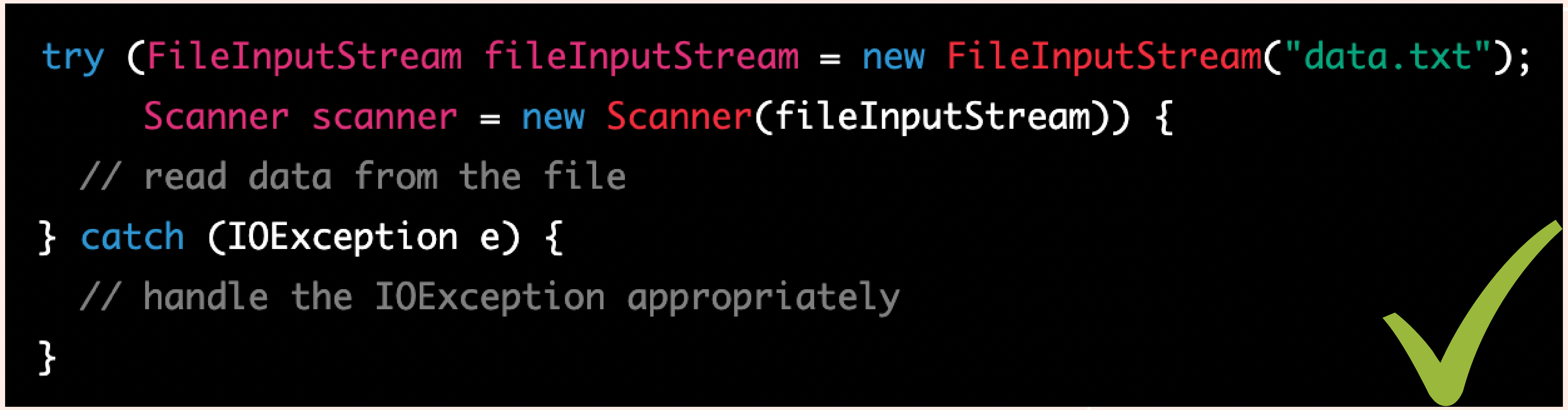
@ankitpangasa

Not properly closing resources



```java
File file = new File("data.txt");
Scanner scanner = new Scanner(file);
// read data from the file
```

The code above doesn't close the file or the scanner after reading data from the file. This can cause resource leaks and unexpected behavior. Instead, Java developers should close resources using try-with-resources blocks:

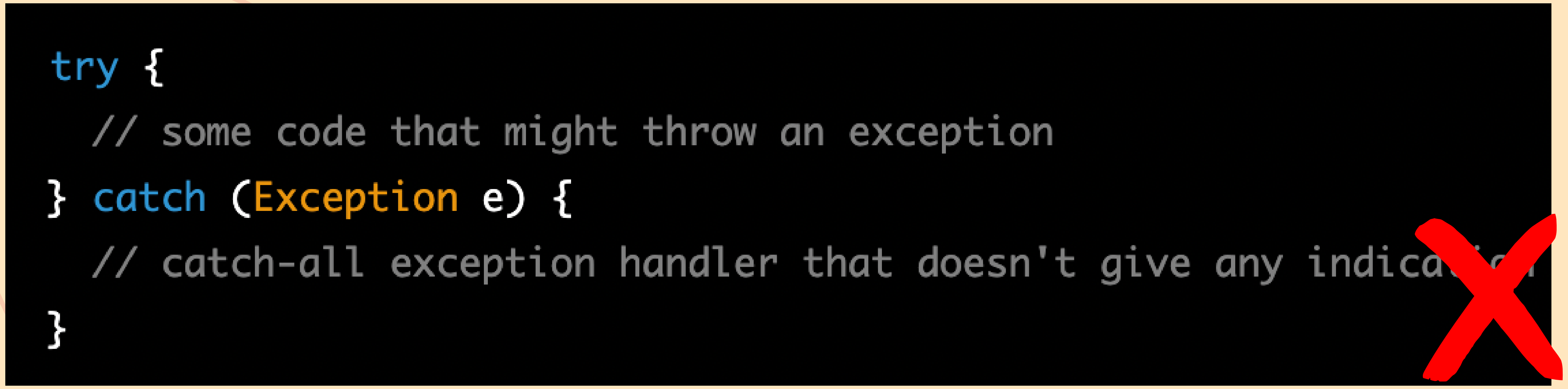


```java
try (FileInputStream fileInputStream = new FileInputStream("data.txt");
     Scanner scanner = new Scanner(fileInputStream)) {
  // read data from the file
} catch (IOException e) {
  // handle the IOException appropriately
}
```

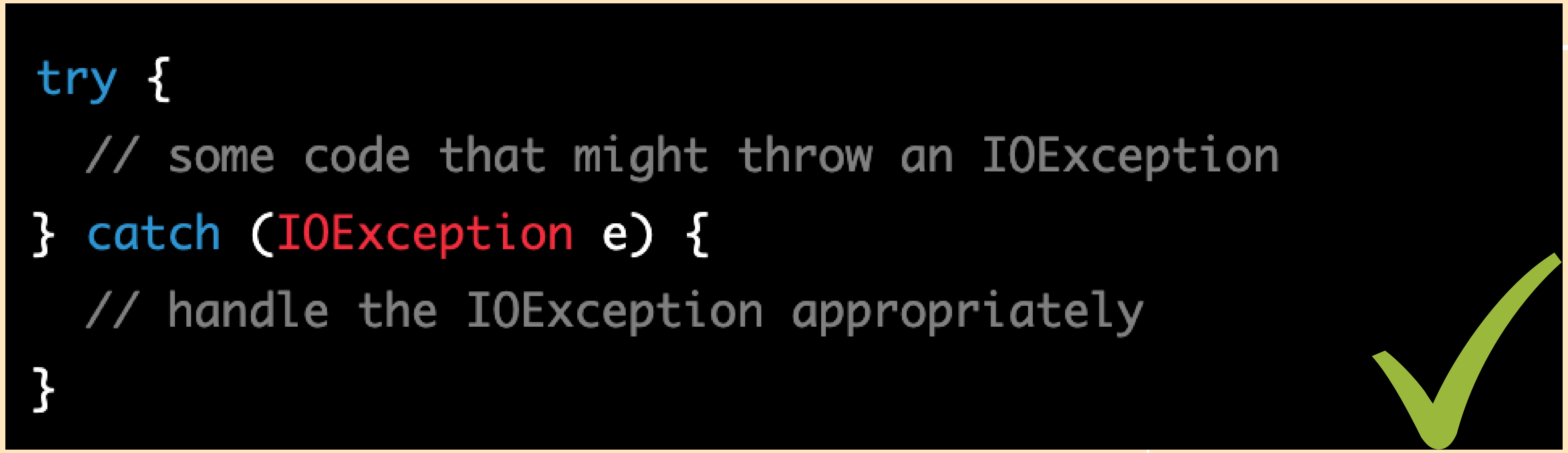
WORLD OF TECHIES EST 2023

@ankitpangasa

Not handling exceptions correctly

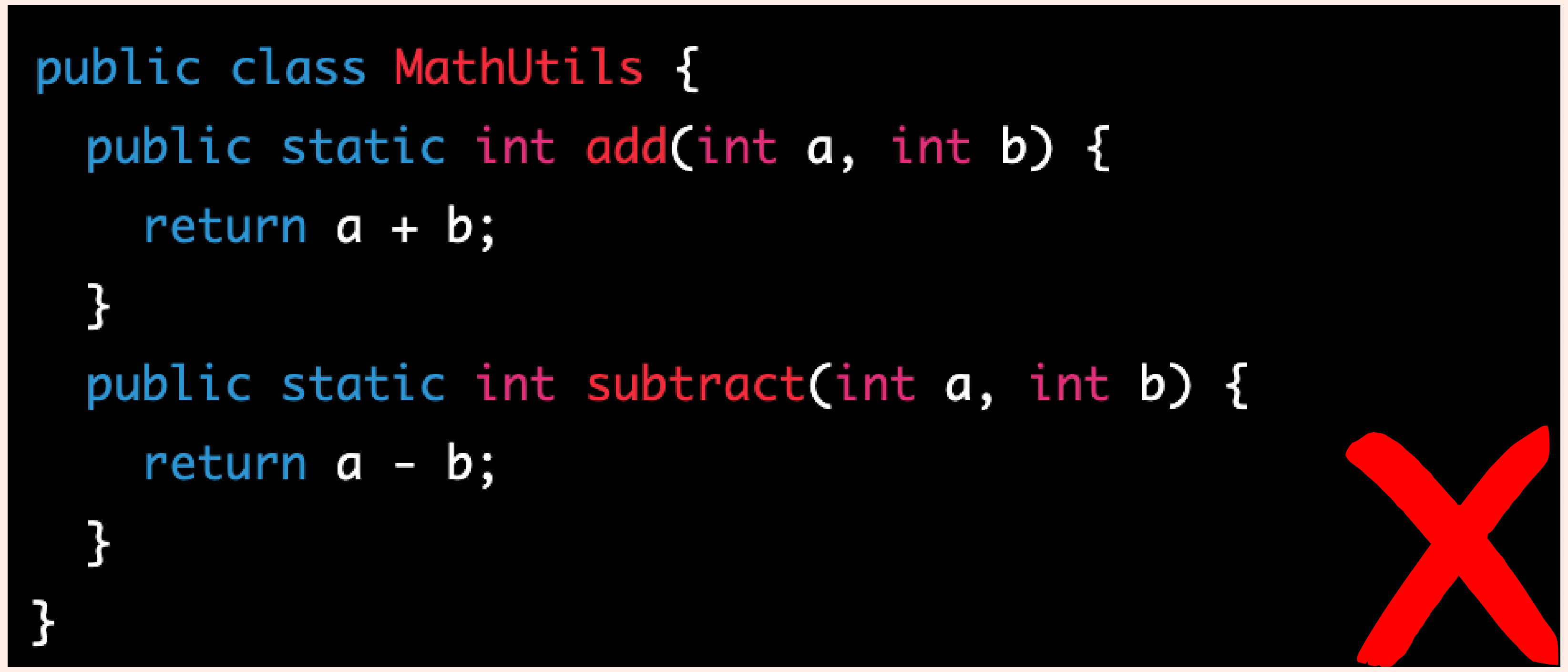

```
try {
  // some code that might throw an exception
} catch (Exception e) {
  // catch-all exception handler that doesn't give any indica[illegible]
}
```

Instead, Java developers should catch specific exceptions and handle them appropriately:

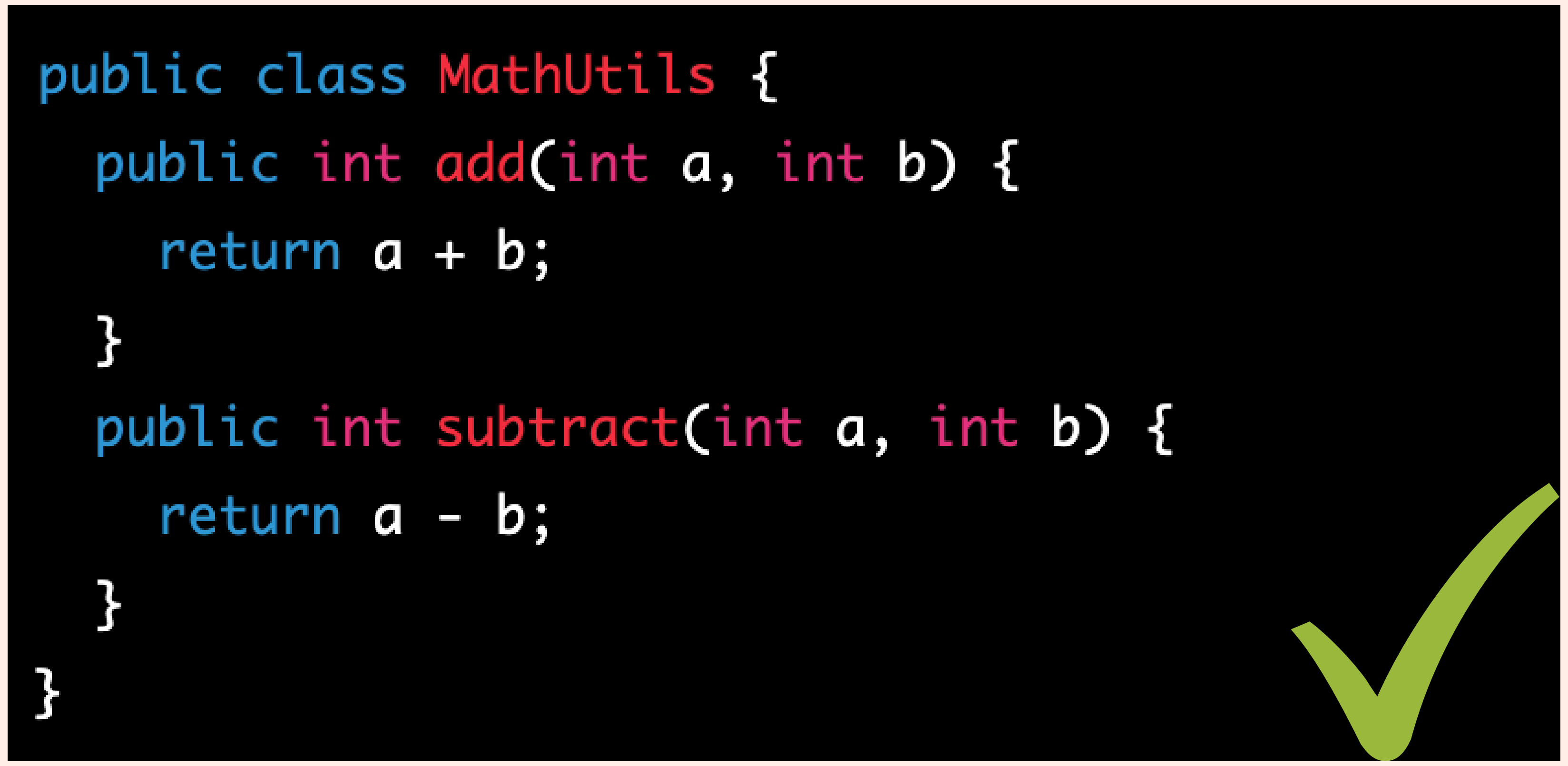

```
try {
  // some code that might throw an IOException
} catch (IOException e) {
  // handle the IOException appropriately
}
```

@ankitpangasa

Overusing or misusing static methods



```java
public class MathUtils {
  public static int add(int a, int b) {
    return a + b;
  }
  public static int subtract(int a, int b) {
    return a - b;
  }
}
```

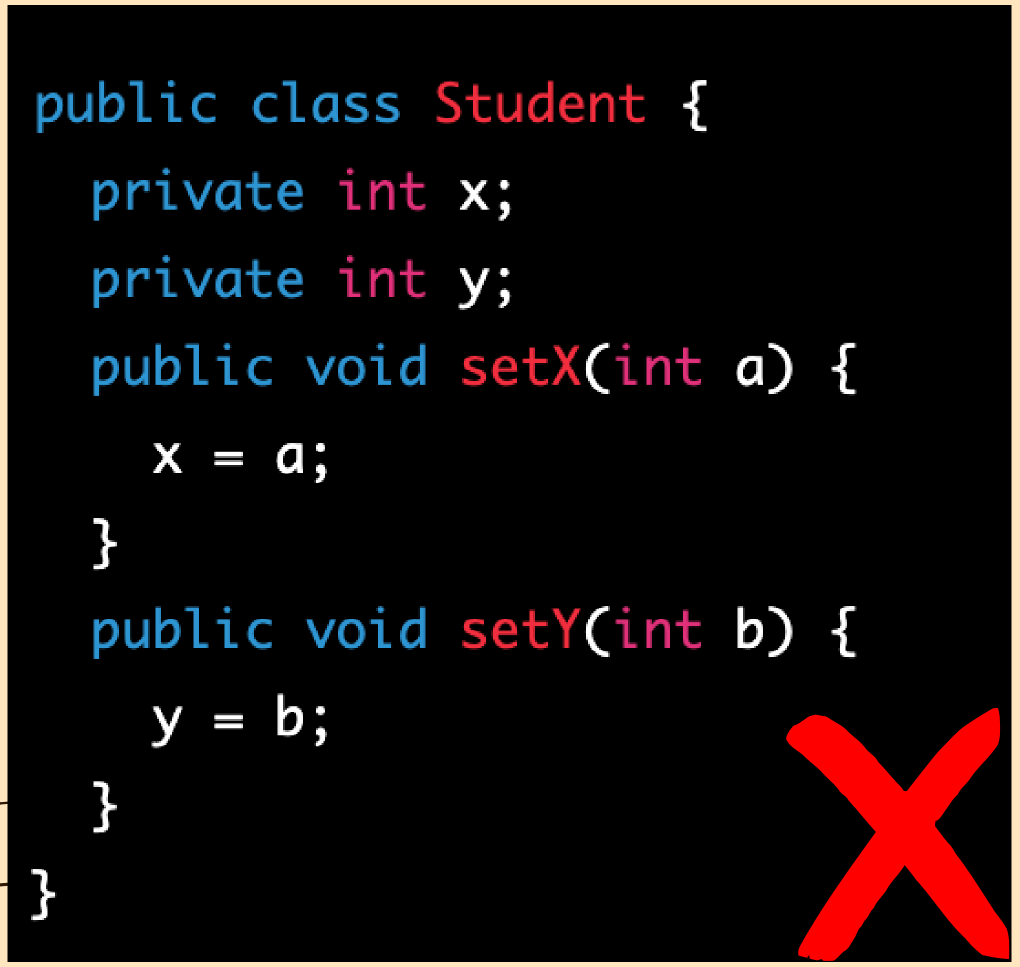


```java
public class MathUtils {
  public int add(int a, int b) {
    return a + b;
  }
  public int subtract(int a, int b) {
    return a - b;
  }
}
```

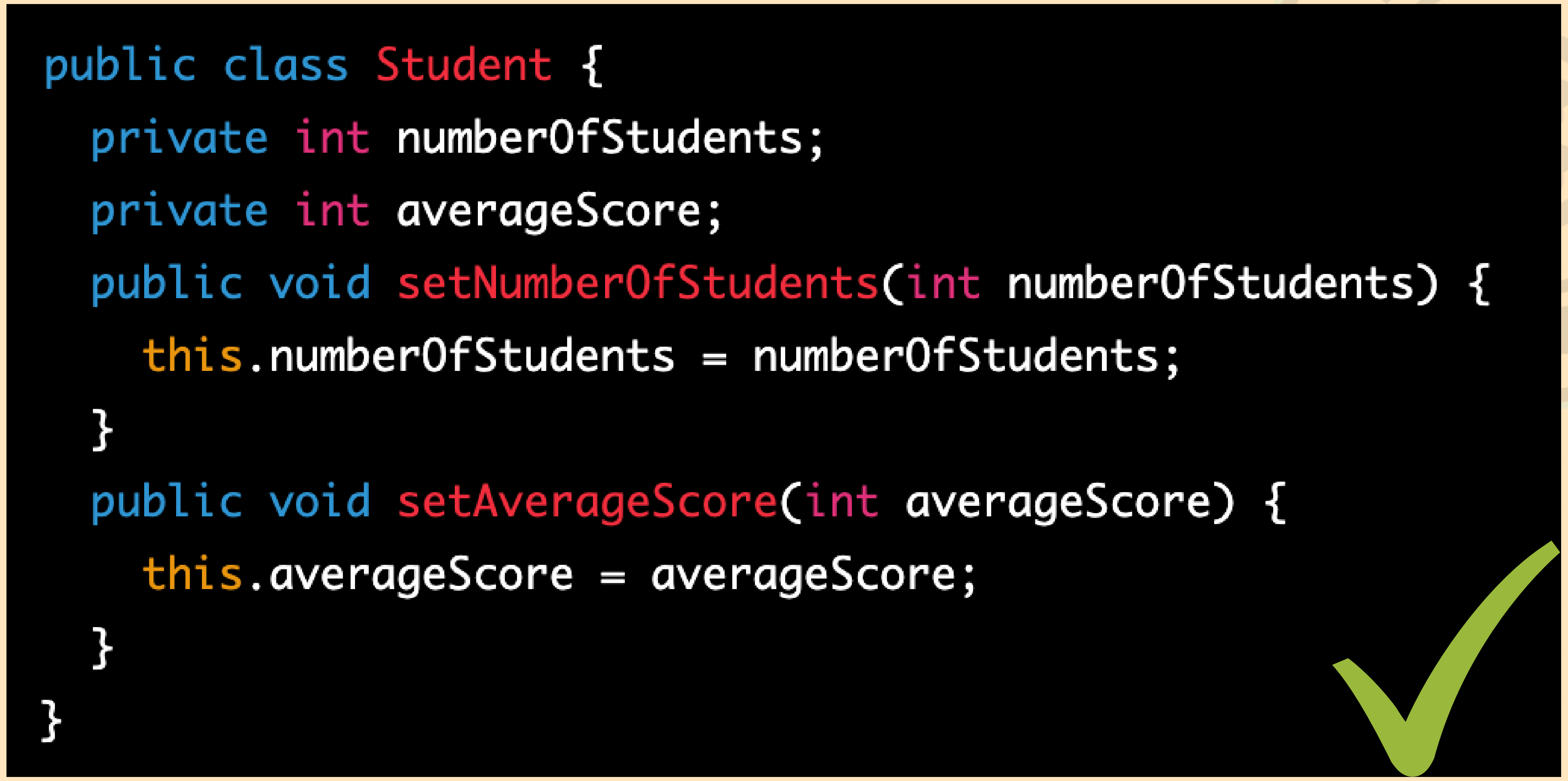
Every method in the MathUtils class is static, even though they could be instance methods. This can make the code less modular and harder to test. Instead, Java developers should use static methods only when appropriate.

Not following naming conventions

In the code sample on the left, the variable names x and y are not very descriptive. Java developers should use descriptive names that follow the standard naming conventions.



```java
public class Student {
  private int x;
  private int y;
  public void setX(int a) {
    x = a;
  }
  public void setY(int b) {
    y = b;
  }
}
```

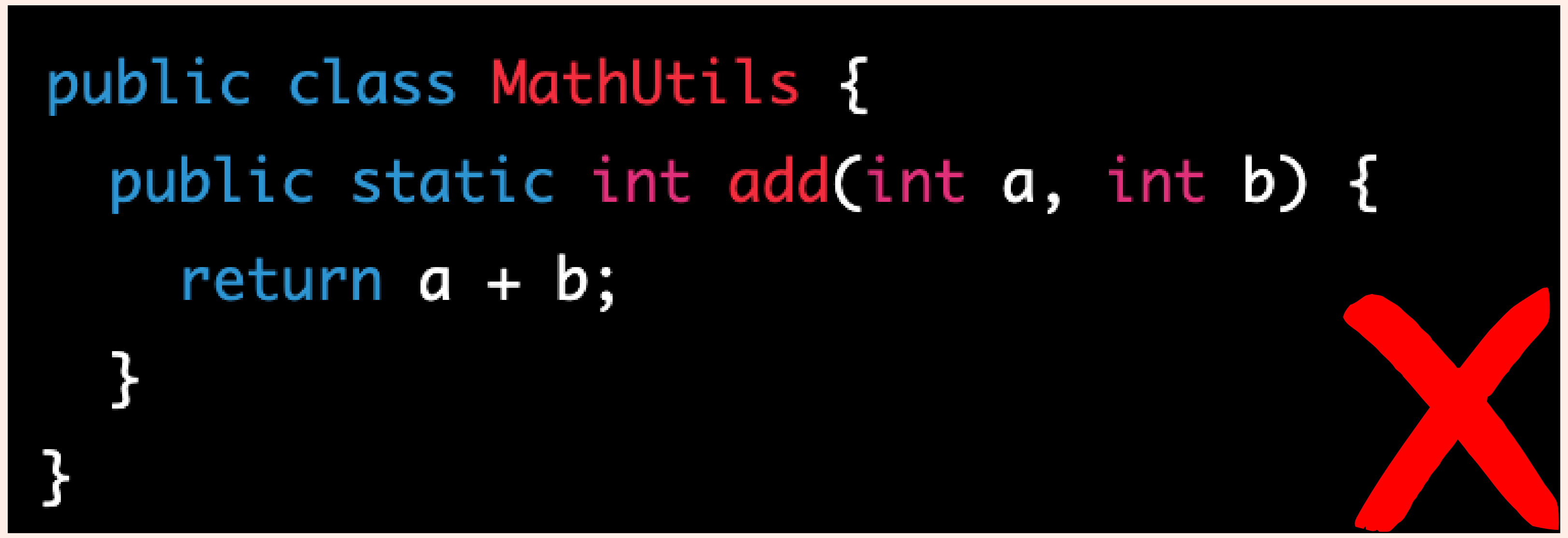


```java
public class Student {
  private int numberOfStudents;
  private int averageScore;
  public void setNumberOfStudents(int numberOfStudents) {
    this.numberOfStudents = numberOfStudents;
  }
  public void setAverageScore(int averageScore) {
    this.averageScore = averageScore;
  }
}
```

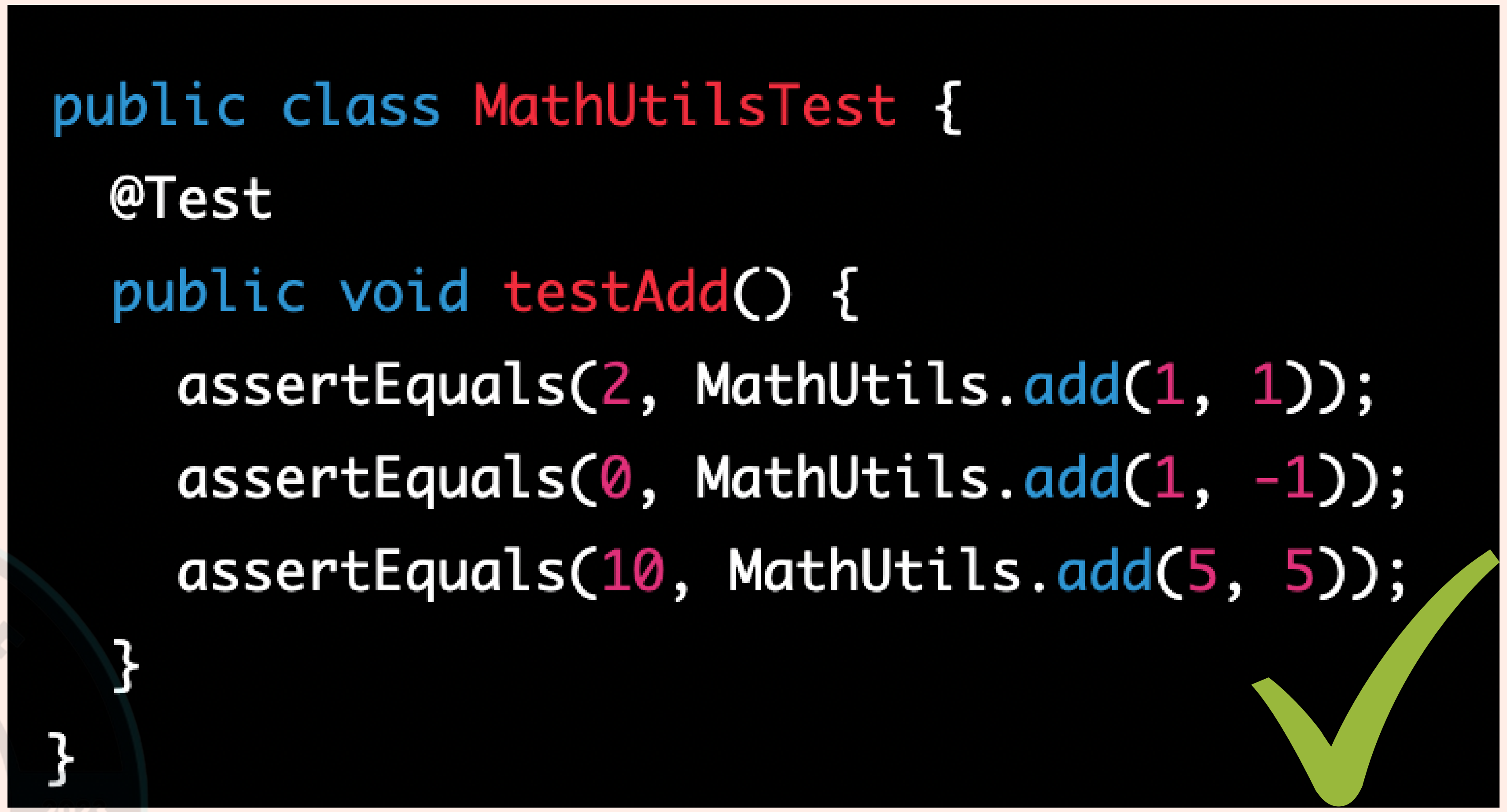
@ankitpangasa

Not writing unit tests

The code on the left doesn't have any unit tests, so it's hard to ensure that the add method works correctly in all cases. Java developers should write unit tests to ensure that their code works correctly:



```java
public class MathUtils {
  public static int add(int a, int b) {
    return a + b;
  }
}
```



```java
public class MathUtilsTest {
  @Test
  public void testAdd() {
    assertEquals(2, MathUtils.add(1, 1));
    assertEquals(0, MathUtils.add(1, -1));
    assertEquals(10, MathUtils.add(5, 5));
  }
}
```

@ankitpangasa

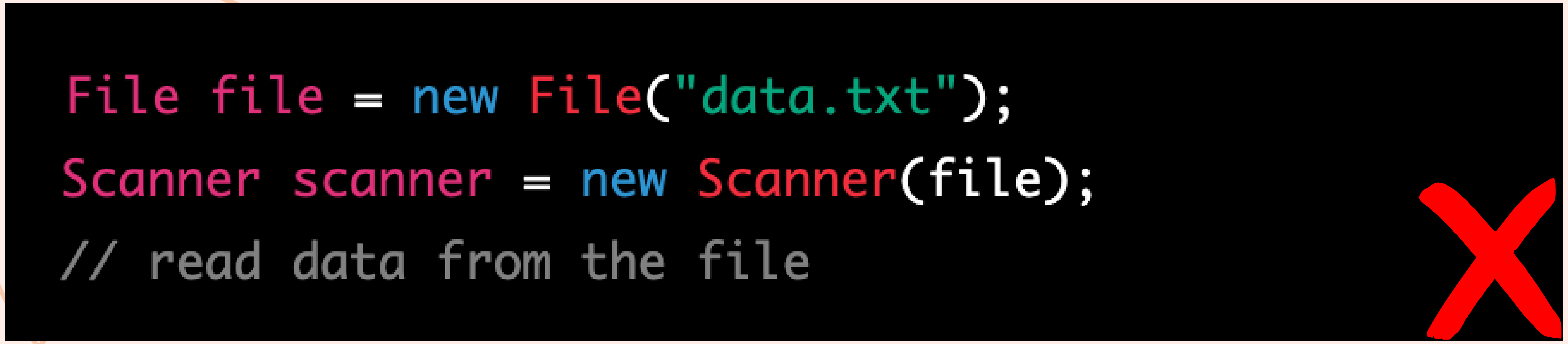
Poor exception handling

```java
try {
  // some code that might throw an exception
} catch (Exception e) {
  // ignore the exception
}
```

The code above catches an exception but doesn't do anything with it. This can make it hard to debug issues when they arise. Instead, Java developers should log or re-throw exceptions:

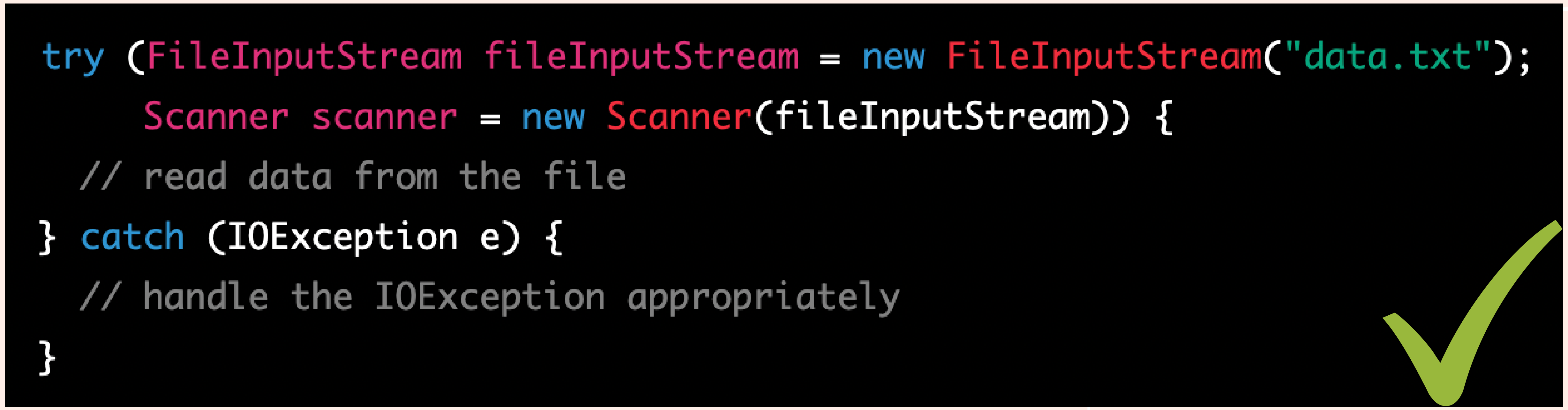
```java
try {
  // some code that might throw an exception
} catch (Exception e) {
  LOGGER.error("An error occurred: {}", e.getMessage());
  throw new RuntimeException(e);
}
```

Not properly closing resources



```
File file = new File("data.txt");
Scanner scanner = new Scanner(file);
// read data from the file
```

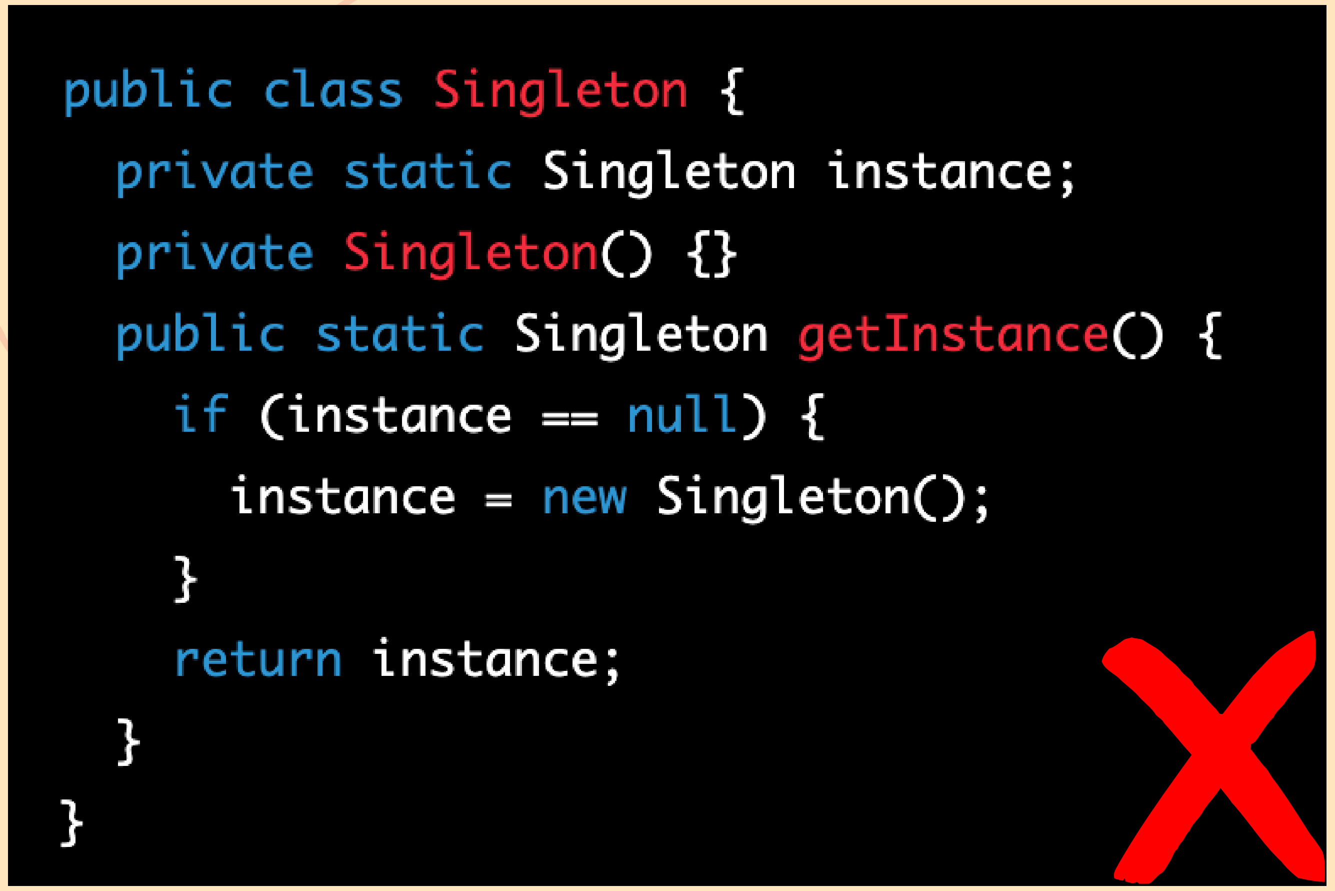
The code above doesn't close the file or the scanner after reading data from the file. This can cause resource leaks and unexpected behavior. Instead, Java developers should close resources using try-with-resources blocks:



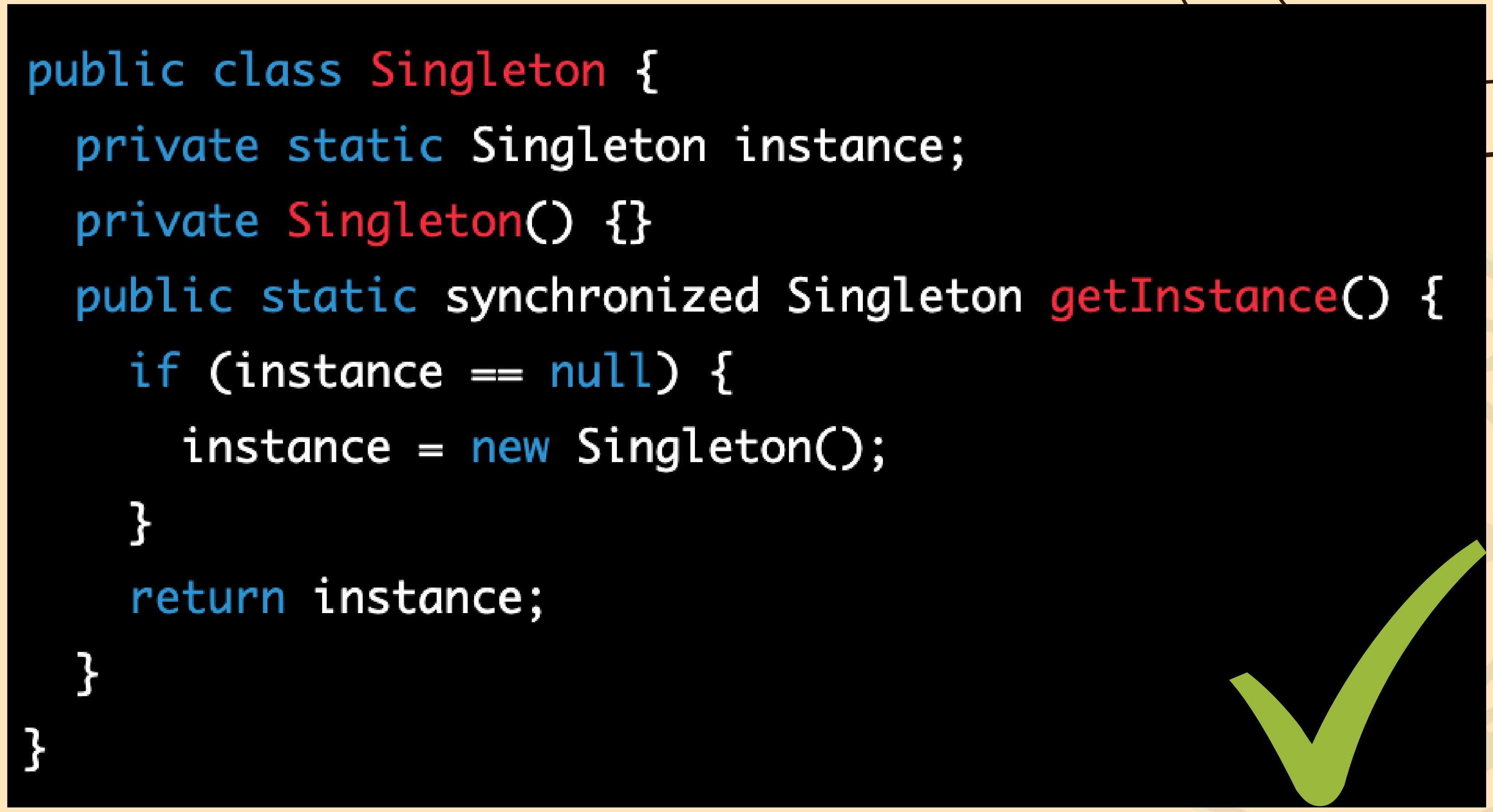
```
try (FileInputStream fileInputStream = new FileInputStream("data.txt");
     Scanner scanner = new Scanner(fileInputStream)) {
  // read data from the file
} catch (IOException e) {
  // handle the IOException appropriately
}
```

@ankitpangasa

Not using design patterns



```
public class Singleton {
  private static Singleton instance;
  private Singleton() {}
  public static Singleton getInstance() {
    if (instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

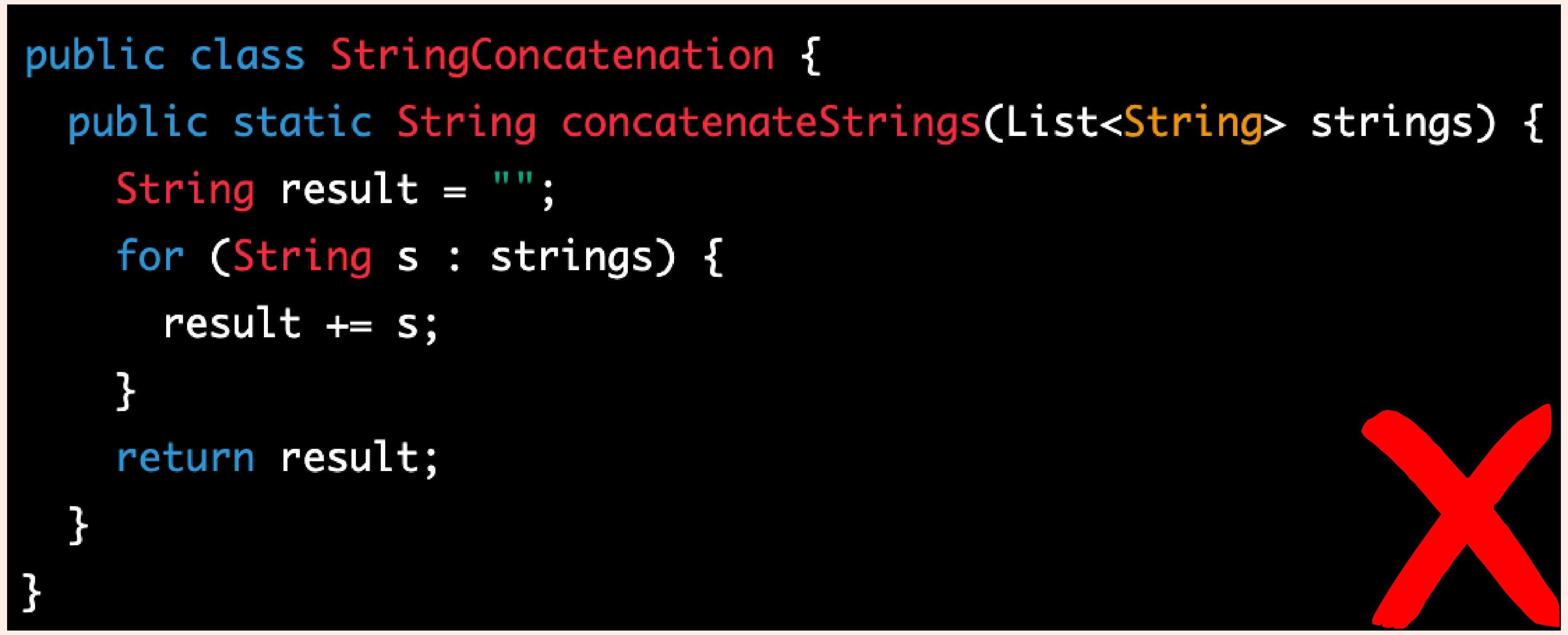


```
public class Singleton {
  private static Singleton instance;
  private Singleton() {}
  public static synchronized Singleton getInstance() {
    if (instance == null) {
      instance = new Singleton();
    }
    return instance;
  }
}
```

The code on the left implements the Singleton pattern, but it doesn't ensure thread safety, which can result in multiple instances of the Singleton being created. Java developers should ensure that their use of design patterns is appropriate and correct:

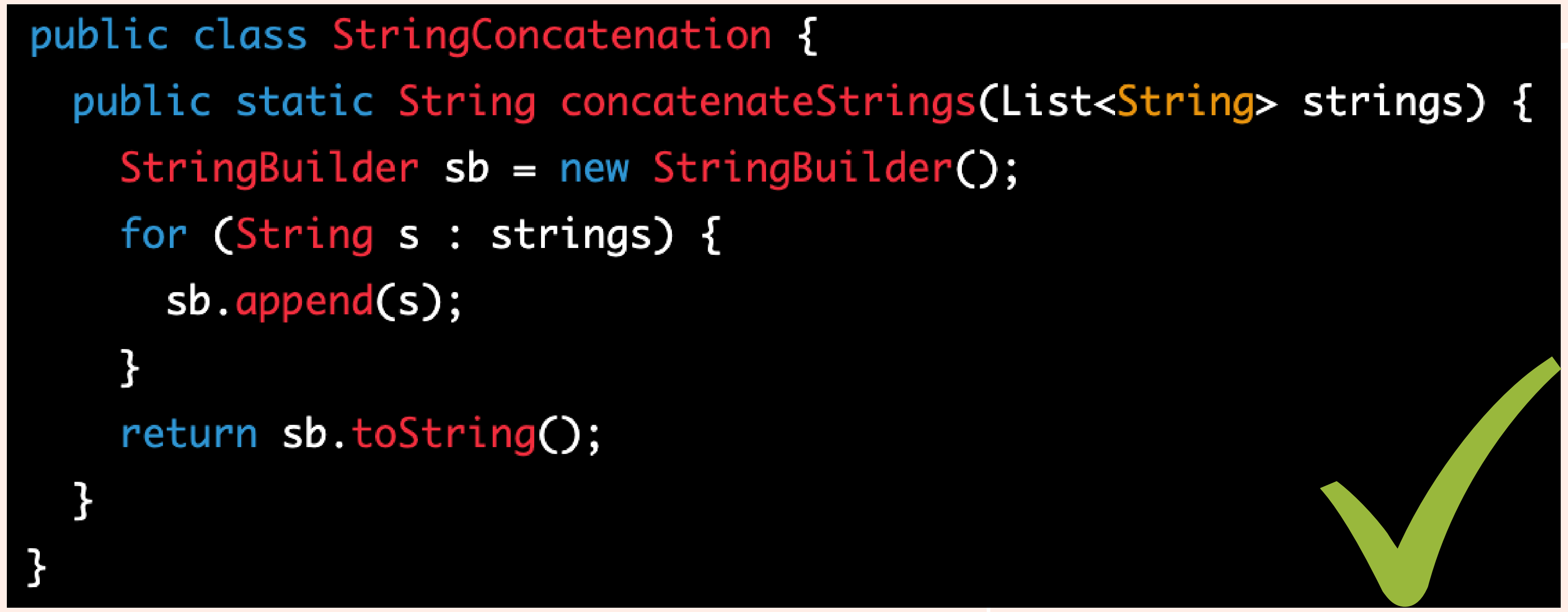
@ankitpangasa

Not optimizing code for performance



```java
public class StringConcatenation {
  public static String concatenateStrings(List<String> strings) {
    String result = "";
    for (String s : strings) {
      result += s;
    }
    return result;
  }
}
```

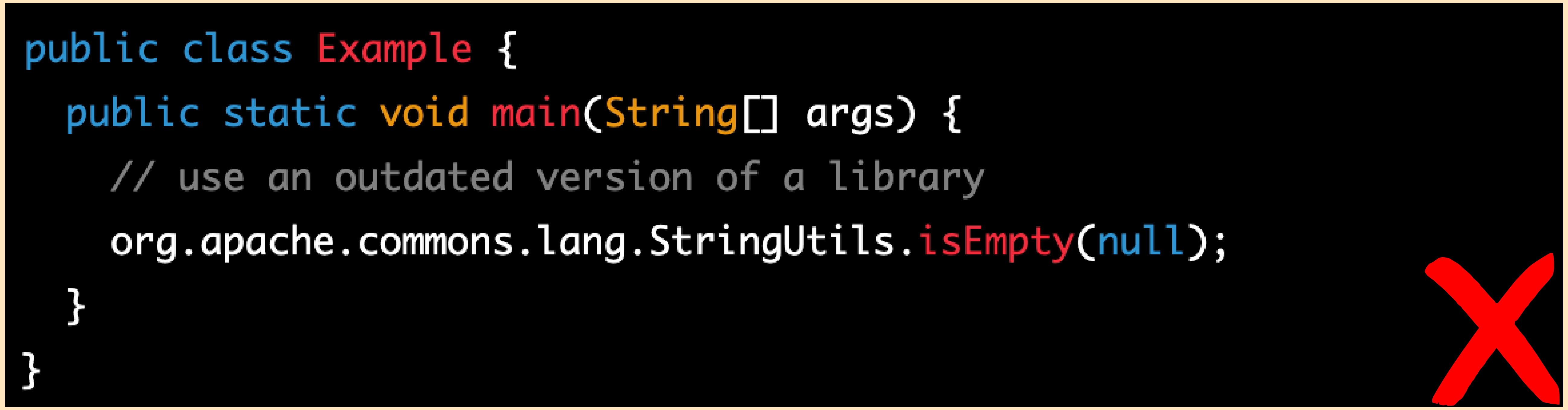
The code above uses the + operator to concatenate strings in a loop, which can be slow and inefficient. Java developers should use StringBuilder to optimize string concatenation:



```java
public class StringConcatenation {
  public static String concatenateStrings(List<String> strings) {
    StringBuilder sb = new StringBuilder();
    for (String s : strings) {
      sb.append(s);
    }
    return sb.toString();
  }
}
```

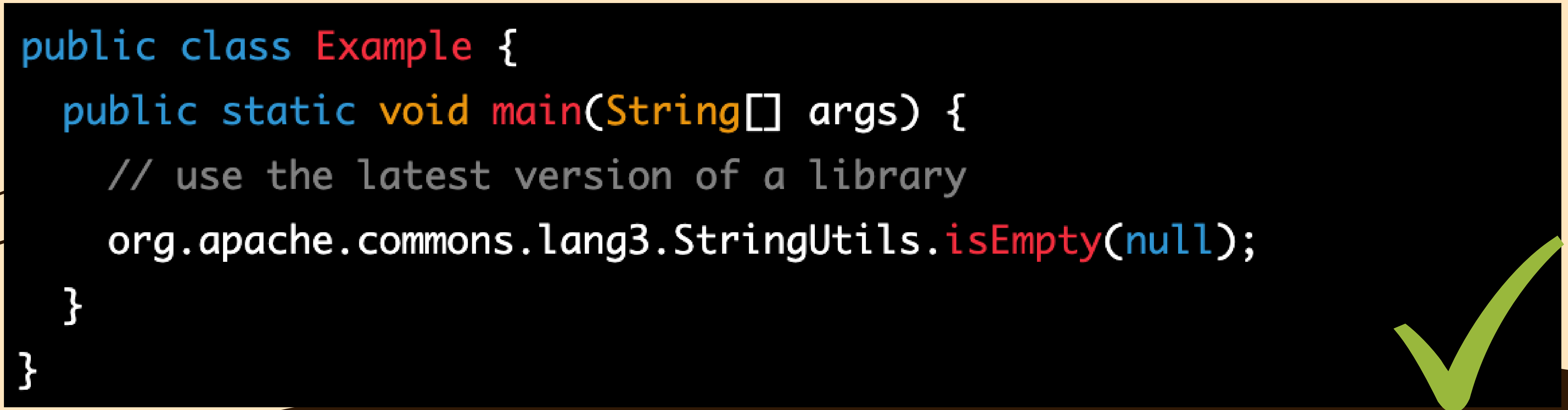
@ankitpangasa

Not keeping up with updates and security patches



```
public class Example {
  public static void main(String[] args) {
    // use an outdated version of a library
    org.apache.commons.lang.StringUtils.isEmpty(null);
  }
}
```

The code above uses an outdated version of the Apache Commons Lang library that has a known security vulnerability. Java developers should keep their libraries and frameworks up to date to ensure that their code is secure:



```
public class Example {
  public static void main(String[] args) {
    // use the latest version of a library
    org.apache.commons.lang3.StringUtils.isEmpty(null);
  }
}
```

@ankitpangasa

That's a wrap!

Thank you for reading!