Errors & Testing

Camille Duquesne
1ère IB

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Any questions/difficulties you would like to share about last session’s content ?

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

B2 Programming

B2.1.3 Describe how programs use common exception handling techniques.
• Potential points of failure in a program must include unexpected inputs, resource unavailability, logic errors.
• The role of exception handling in developing programs
• Exception handling constructs that effectively manage errors must include try/catch in Java, and try/except in Python, along with the finally block.

B2.1.4 Construct and use common debugging techniques.
• Debugging techniques may include trace tables, breakpoint debugging, print statements and step-by-step code execution.

B2.3.1 Construct programs that implement the correct sequence of code instructions to meet program objectives.
• The impact of instruction order on program functionality
• Ways to avoid errors, such as infinite loops, deadlock, incorrect output

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

What is testing ?

Testing refers to an activity that checks that the software is **defect free** and meeting all client requirements.
Several types of testing happen at different stages of product development such as dry run testing, unit testing, security testing, alpha testing, beta testing, integration testing, user acceptance testing, etc.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Why is testing important ?

- To prevent end user dissatisfaction
- Enables early detection of errors
- Reduces production delay and costs
- Ensures the system meets its design specifications
- Detect security risks and privacy concerns

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Errors

Syntax errors are usually called before we run our program, when java goes through compilation

When we run our programm different reasons might cause our programs to have errors.

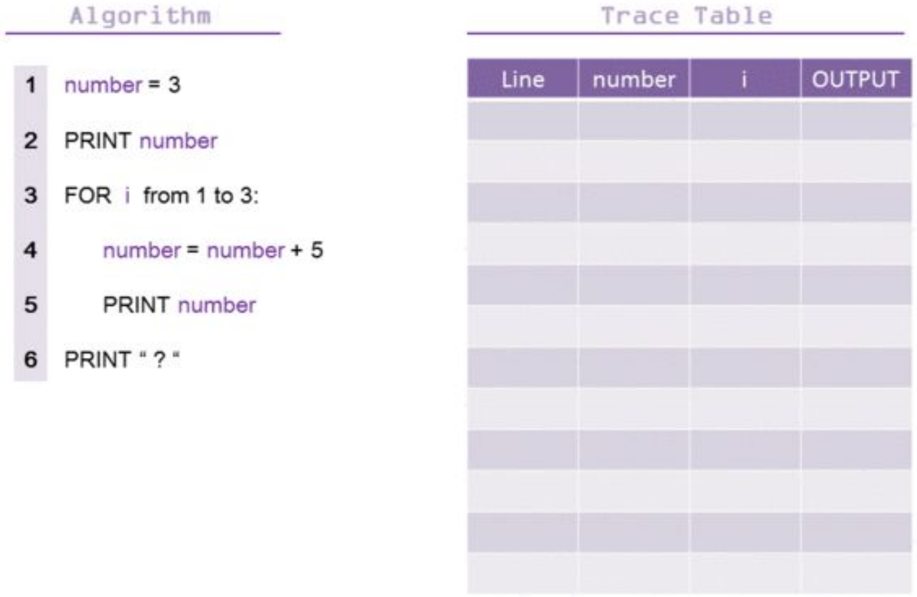
- **Logic errors**: these are caused by an incorrect logic in our program: incorrect sequence of instructions, missing instructions ,...
  They don't cause the program to crash but we don't obtain the intended output. These errors can be detected through **testing strategies**.
- **Runtime errors**: these instructions are caused as the program runs: incorrect file path, incorrect user input, division by 0, printer not ready, etc ...
  These cause the program to crash. They can be mitigated using **exception handling**

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Testing strategies - trace table

A **trace table** is a technique used to test an algorithm and predict **step by step** how the computer will run the algorithm. It can be used to understand or predict what an algorithm is doing and to identify potential **logic errors** (when the program compiles but does not produce the expected output).

Using a trace table to test an algorithm is called **dry run testing**.



Algorithm

```
1 number = 3
2 PRINT number
3 FOR i from 1 to 3:
4     number = number + 5
5     PRINT number
6 PRINT " ? "
```

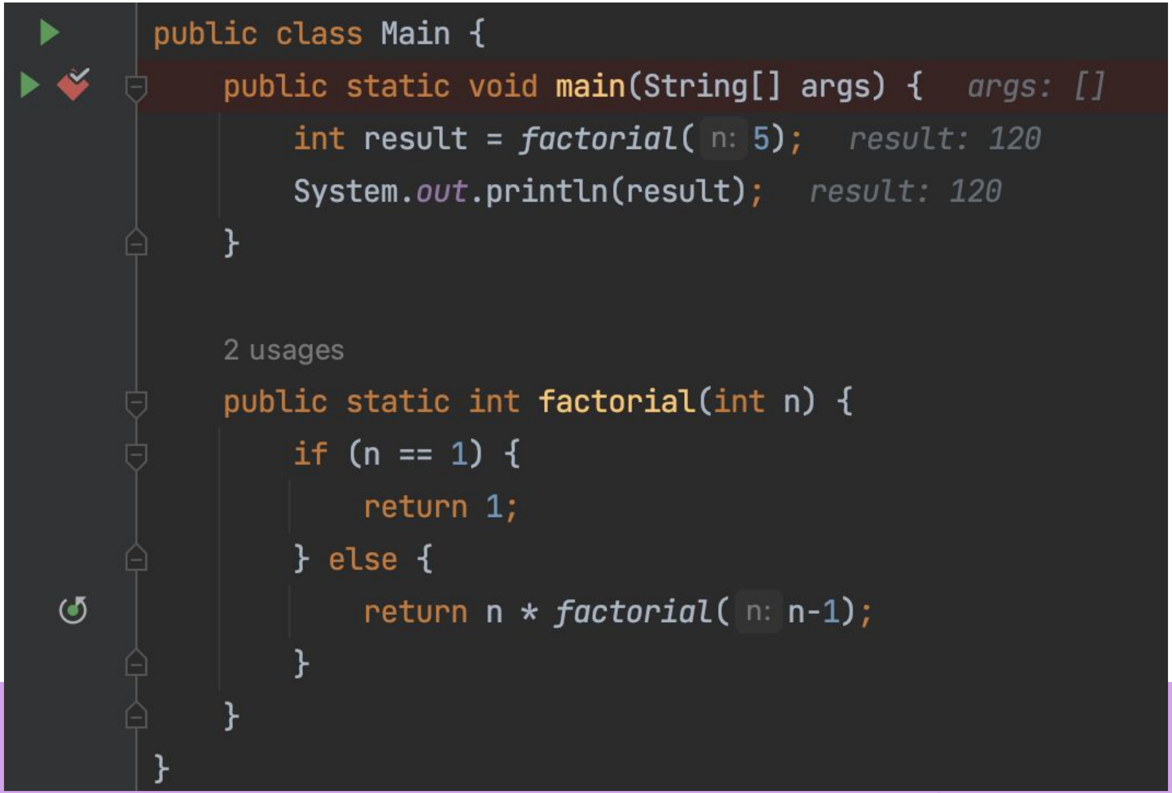
Trace Table

Line	number	i	OUTPUT

https://www.101computing.net/wp/wp-content/uploads/trace-table-s.gif

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Testing strategies - breakpoint



```
public class Main {
    public static void main(String[] args) {   args: []
        int result = factorial( n: 5);   result: 120
        System.out.println(result);   result: 120
    }

    2 usages
    public static int factorial(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * factorial( n: n-1);
        }
    }
}
```

If you add a breakpoint (red dot on the right) and then press debug, you will be able to see the step by step execution of your program !

https://blog.jetbrains.com/idea/2020/05/debugger-basics-in-intellij-idea/

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Testing Strategies - Unit Testing

This type of testing focuses on **verifying the functionality of individual components (units)** of the system. It involves testing small, isolated sections usually individual functions or objects.

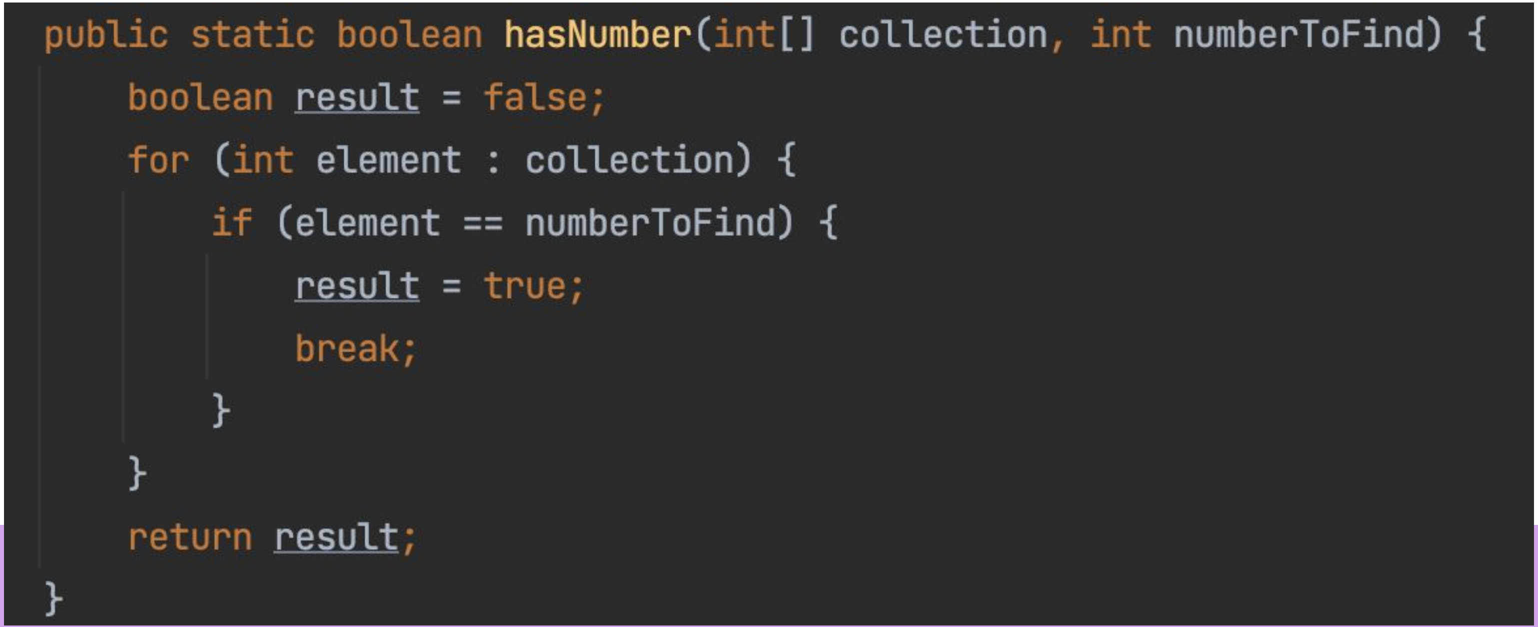
Unit tests are written **at the same time** as the objects/functions (we don’t wait for the whole system to be done to begin writing unit tests).

We usually want to test the behavior for **normal data, extreme data and abnormal data.**

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

A little more about unit testing !

Imagine we have a method that checks if an element is in an array and we want to test that method



```java
public static boolean hasNumber(int[] collection, int numberToFind) {
    boolean result = false;
    for (int element : collection) {
        if (element == numberToFind) {
            result = true;
            break;
        }
    }
    return result;
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

What is JUnit ?

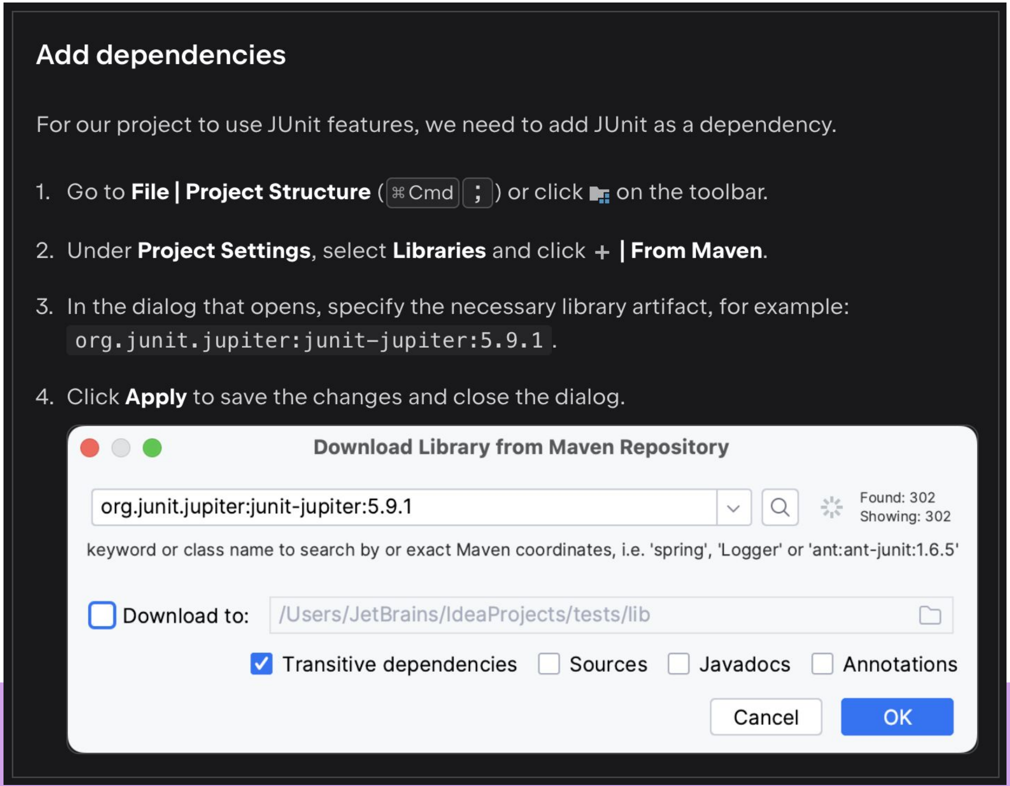
In Java we can use the package JUnit to perform unit tests !

JUnit simplifies the process of creating and executing tests, making it an essential tool for software developers to maintain code quality and detect bugs early in the development process.

https://junit.org/junit5/docs/current/user-guide/

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

How do we add JUnit ?



Add dependencies

For our project to use JUnit features, we need to add JUnit as a dependency.

1. Go to **File | Project Structure** (⌘Cmd ;) or click on the toolbar.
2. Under **Project Settings**, select **Libraries** and click + | **From Maven**.
3. In the dialog that opens, specify the necessary library artifact, for example: `org.junit.jupiter:junit-jupiter:5.9.1`.
4. Click **Apply** to save the changes and close the dialog.

Download Library from Maven Repository

org.junit.jupiter:junit-jupiter:5.9.1

Found: 302
Showing: 302

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

[ ] Download to: /Users/JetBrains/IdeaProjects/tests/lib

[✓] Transitive dependencies [ ] Sources [ ] Javadocs [ ] Annotations

https://www.jetbrains.com/help/idea/junit.html#intellij

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

How do I test my method ? The test class

Conventionally we have one test class/file for each class/file that we have defined.

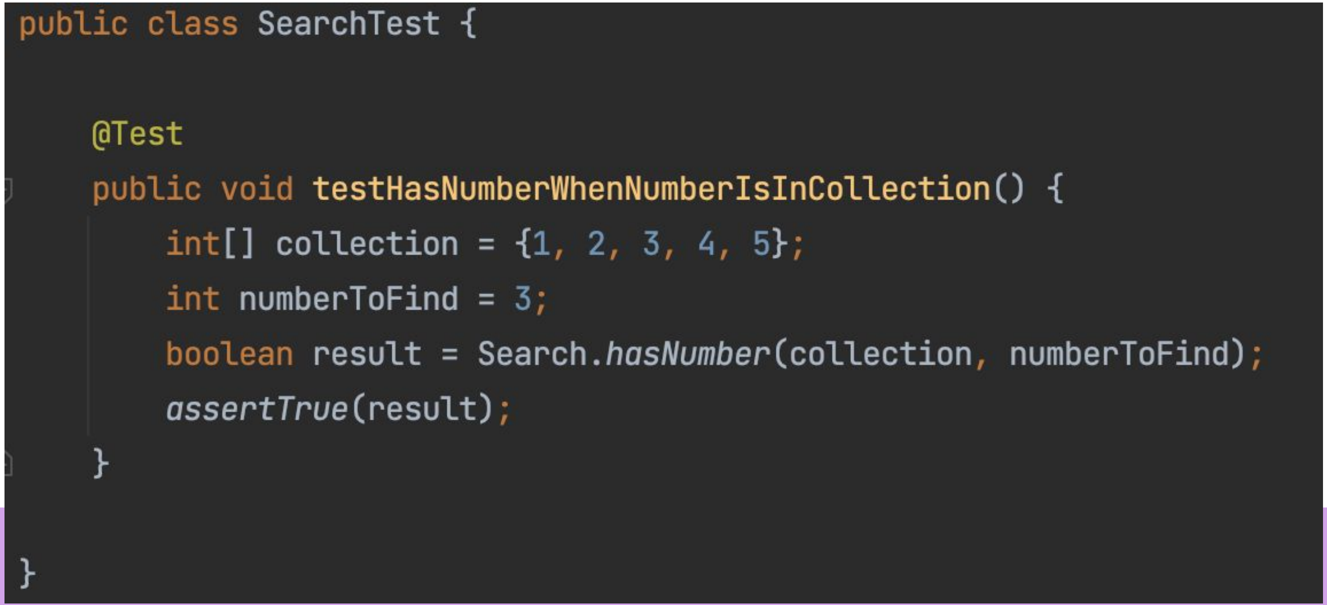
Since our previous method was in a class called Search, we conventionally call our test class SearchTest.

Otherwise you can also Right click on your class name -> generate -> tests, and IntelliJ will create the test class for you.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

How do I test my method ? The anatomy of a test

Your Unit Test are going to be methods that call your methods in specific scenarios and checks if the result is the one you intend.

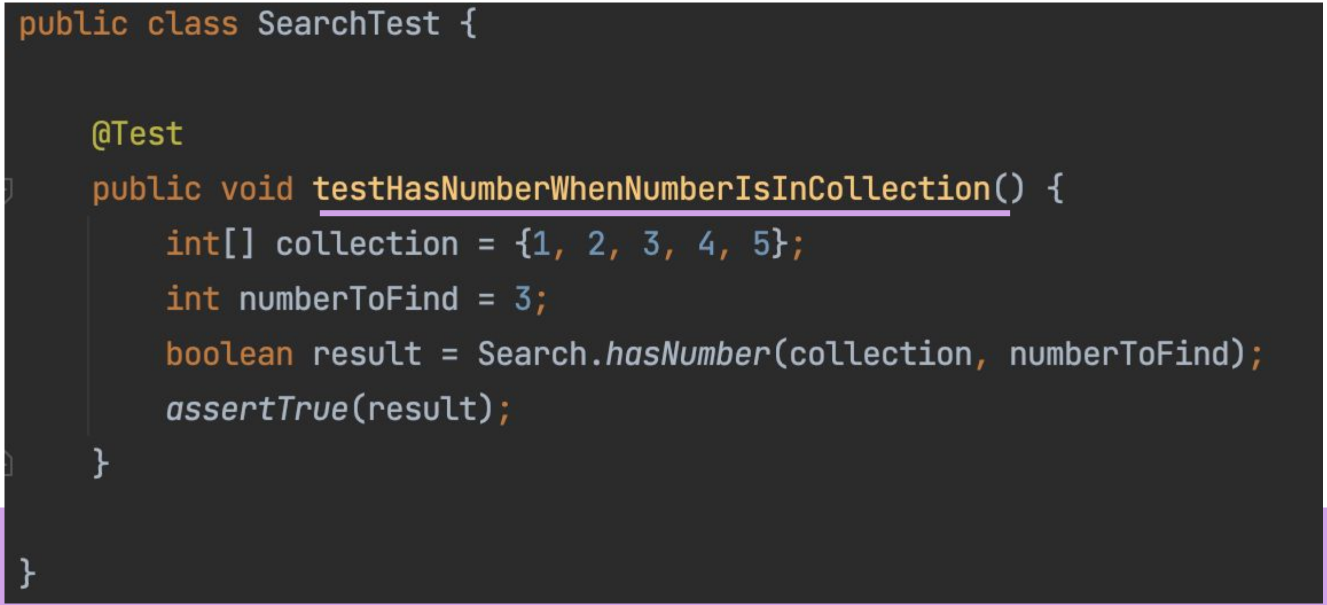


```java
public class SearchTest {

    @Test
    public void testHasNumberWhenNumberIsInCollection() {
        int[] collection = {1, 2, 3, 4, 5};
        int numberToFind = 3;
        boolean result = Search.hasNumber(collection, numberToFind);
        assertTrue(result);
    }

}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

The anatomy of a test - the method name

The method name needs to be representative of what you are testing. It's okay here if the method names are a bit longer, if it makes them clearer.

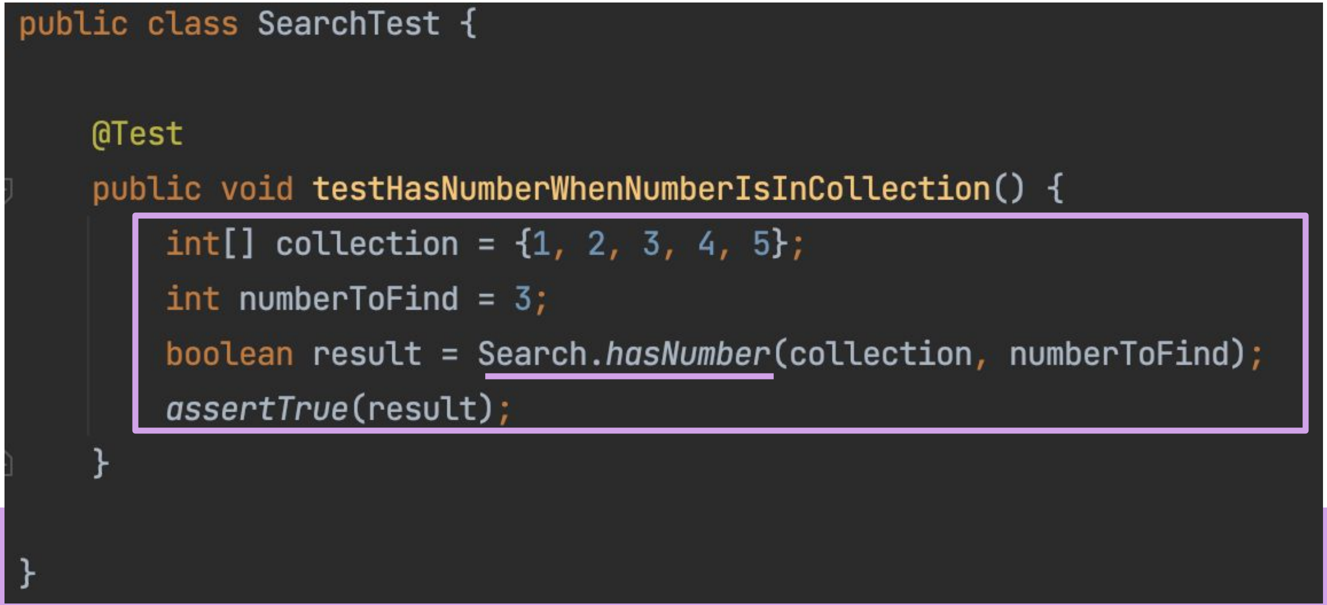


```java
public class SearchTest {

    @Test
    public void testHasNumberWhenNumberIsInCollection() {
        int[] collection = {1, 2, 3, 4, 5};
        int numberToFind = 3;
        boolean result = Search.hasNumber(collection, numberToFind);
        assertTrue(result);
    }

}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

The anatomy of a test - the body

In our test method we create sample data, here a collection and a number to find and we call our method hasNumber() with our sample data.

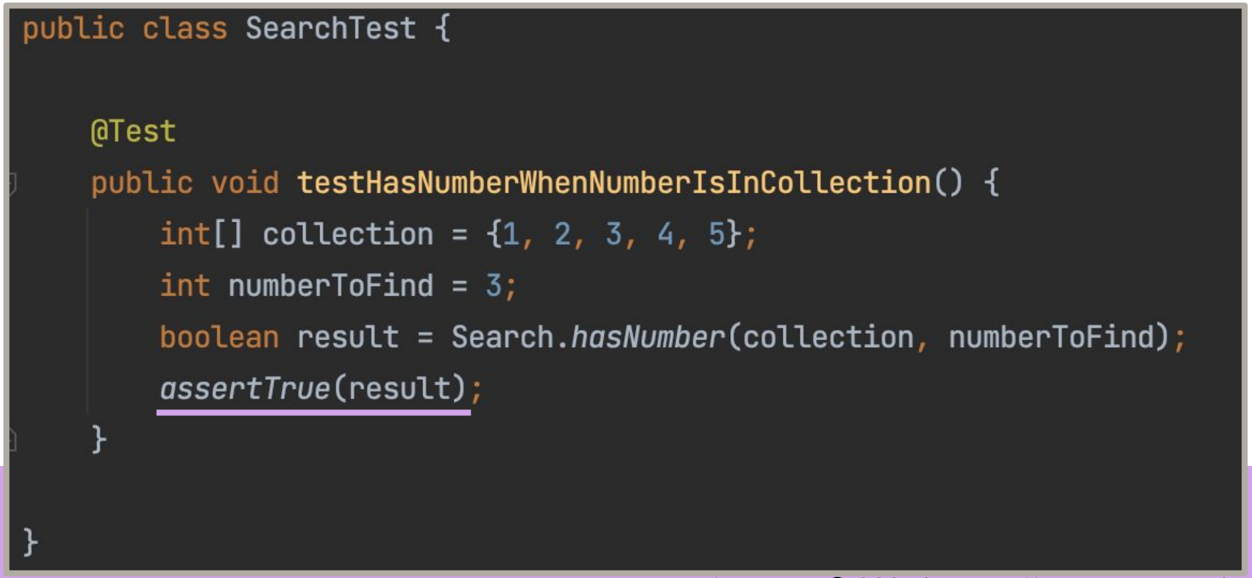


```
public class SearchTest {

    @Test
    public void testHasNumberWhenNumberIsInCollection() {
        int[] collection = {1, 2, 3, 4, 5};
        int numberToFind = 3;
        boolean result = Search.hasNumber(collection, numberToFind);
        assertTrue(result);
    }

}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

The anatomy of a test - the assert

The assert statement in the end allows us to check if the result we get is the intended one. Here hasNumber() should return true, hence assertTrue() checks if result is equal to true.



```
public class SearchTest {

    @Test
    public void testHasNumberWhenNumberIsInCollection() {
        int[] collection = {1, 2, 3, 4, 5};
        int numberToFind = 3;
        boolean result = Search.hasNumber(collection, numberToFind);
        assertTrue(result);
    }

}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Assertion methods

There are many more assertion methods than assertTrue() that we can use:

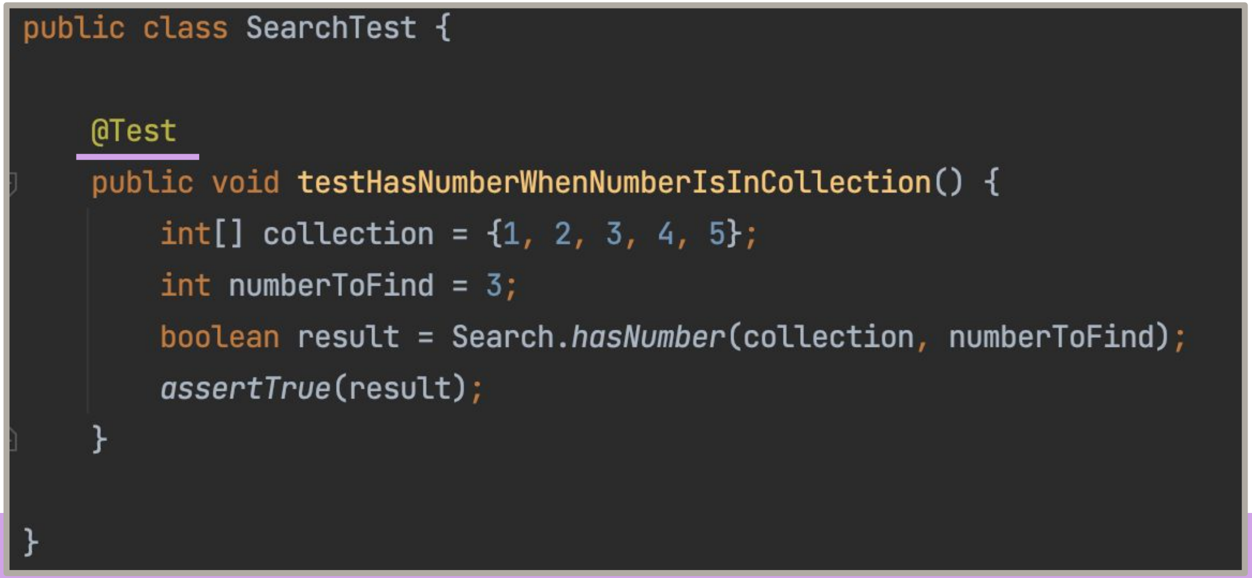
https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

Which assertion methods seem very useful ?

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

The anatomy of a test - the @Test annotation

The @Test annotation allows the test runner (provided by Junit framework) to recognize that a specific method is a test method.

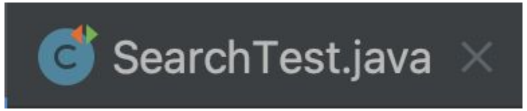


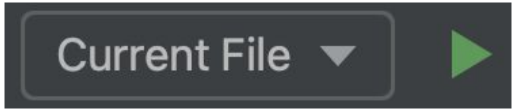
```java
public class SearchTest {

    @Test
    public void testHasNumberWhenNumberIsInCollection() {
        int[] collection = {1, 2, 3, 4, 5};
        int numberToFind = 3;
        boolean result = Search.hasNumber(collection, numberToFind);
        assertTrue(result);
    }

}
```

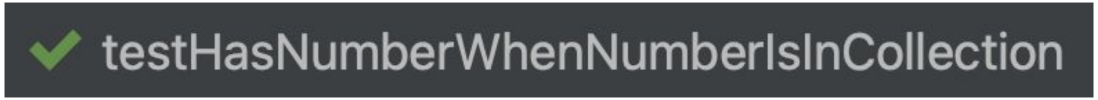
Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

The anatomy of a test - Running tests

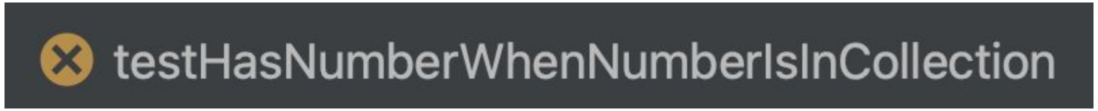
When you are on SearchTest.java you can run the file and you should see if the tests are successful or unsuccessful.







Successful test



Unsuccessful test

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Best Practices for writing tests

Test One Thing at a Time: Each test method should focus on testing one specific behavior or scenario.

Use Descriptive Test Names: Give your test methods clear and descriptive names that explain what is being tested.

Keep Tests Independent: Tests should not rely on the order of execution or the results of other tests. Tests should be able to run in any order.

Test Extreme/Abnormal Cases: Ensure that your tests cover not only typical scenarios but also extreme cases and abnormal cases.

Regularly Refactor Tests: As your code evolves, update and refactor your tests to maintain their accuracy and readability.

Review and Collaborate: Have peers review your tests to ensure their quality and effectiveness.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

What are exceptions of an algorithm ?

An **exception** represents an unexpected or abnormal situation that deviates from the algorithm's normal behavior, often resulting in errors or anomalies.

Proper **exception handling** and, when possible, preventing exceptions are crucial aspects of algorithm design to ensure reliable performance in the presence of exceptional situations.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Practice !

Identify what the exceptions are for a method that calculates the area of a rectangle.

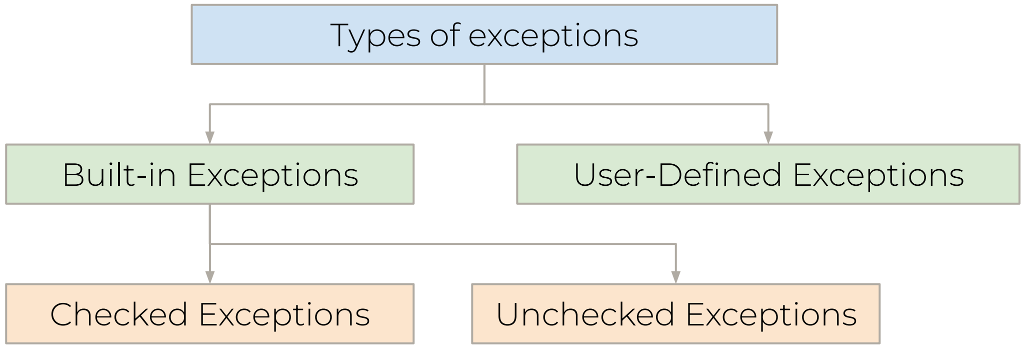
Identify what the exceptions are for an app that scans QR codes

Identify what the exceptions are when moving your character in a video game

Identify what the exceptions are for our hasNumber() method

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exceptions in java



- ClassNotFound Exception
- InterruptedException
- IOException
- InstantiationException
- SQLException
- FileNotFoundException

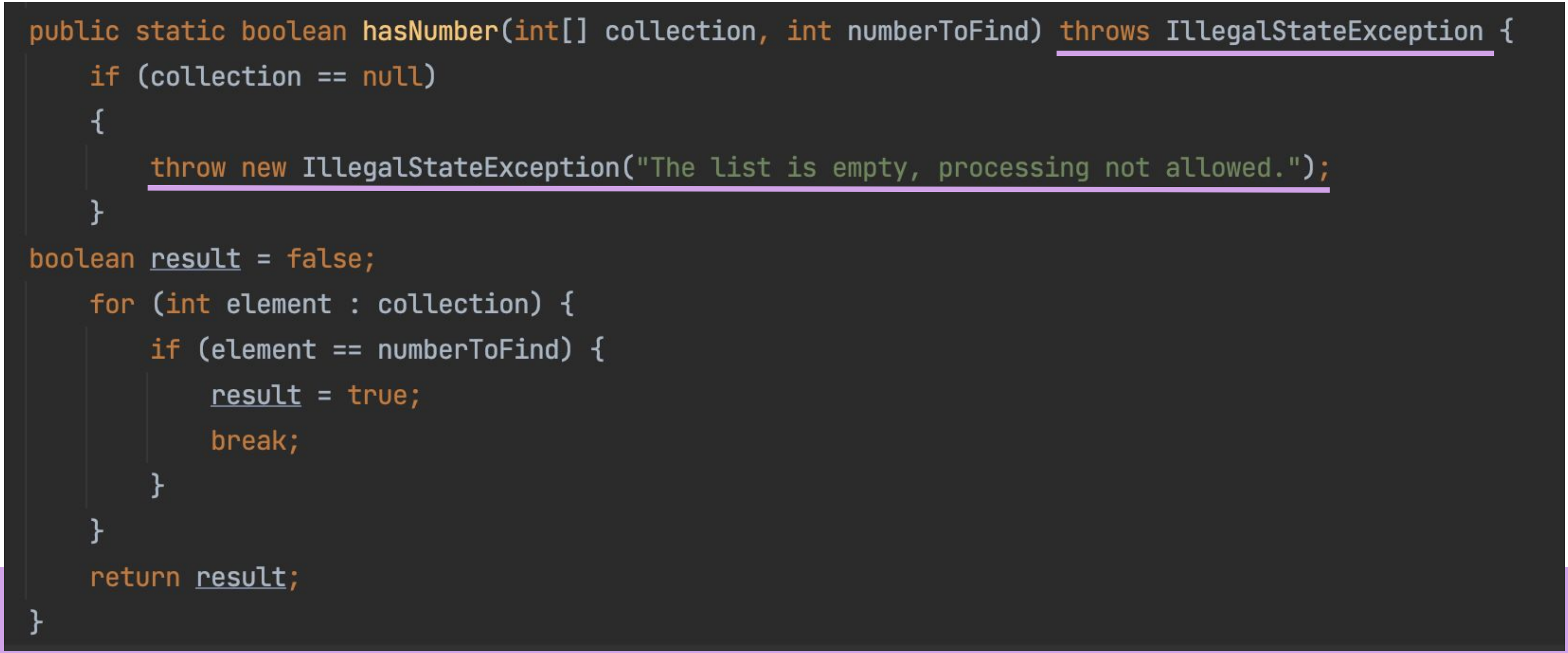
- ArithmeticException
- ClassCastException
- NullPointerException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- IllegalThreadStateException

Exceptions are used to handle unexpected or **erroneous situations** that can occur at **runtime**. When an exception is thrown, the program can take specific actions to gracefully handle the error or terminate in a controlled manner.

https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

What would be an exception to throw in our hasNumber() method ?



```java
public static boolean hasNumber(int[] collection, int numberToFind) throws IllegalStateException {
    if (collection == null)
    {
        throw new IllegalStateException("The list is empty, processing not allowed.");
    }
boolean result = false;
    for (int element : collection) {
        if (element == numberToFind) {
            result = true;
            break;
        }
    }
    return result;
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

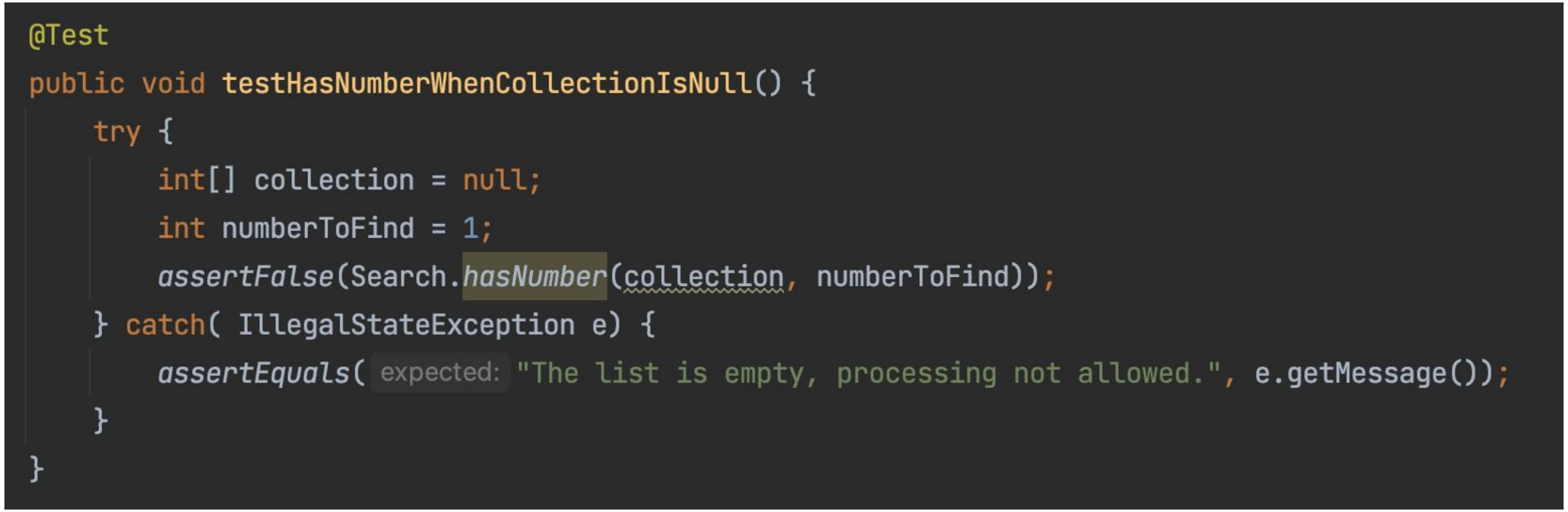
Try catch statement

```
try {
  // usual code, normal behavior
} catch (Exception e) {
  // if usual code give an exception
  we handle the exception here
} finally {
  // executes no matter what
}
```

A try-catch statement in Java is used to handle exceptions. The code within the "try" block is executed, and if an exception occurs, it is caught and handled by the code in the "catch" block, allowing for graceful error handling and recovery in Java programs.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Testing with exceptions



```java
@Test
public void testHasNumberWhenCollectionIsNull() {
    try {
        int[] collection = null;
        int numberToFind = 1;
        assertFalse(Search.hasNumber(collection, numberToFind));
    } catch( IllegalStateException e) {
        assertEquals( expected: "The list is empty, processing not allowed.", e.getMessage());
    }
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

To go further

You can automate your Unit tests to be run on github whenever you push some new code: https://docs.github.com/en/actions/automating-builds-and-tests

This ensures that you always have a functional version and don’t break previous units !

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Pause & Recall

Created by Victoruler
from Noun Project

Close your eyes and try to recall as many things as possible that were covered during this lesson.

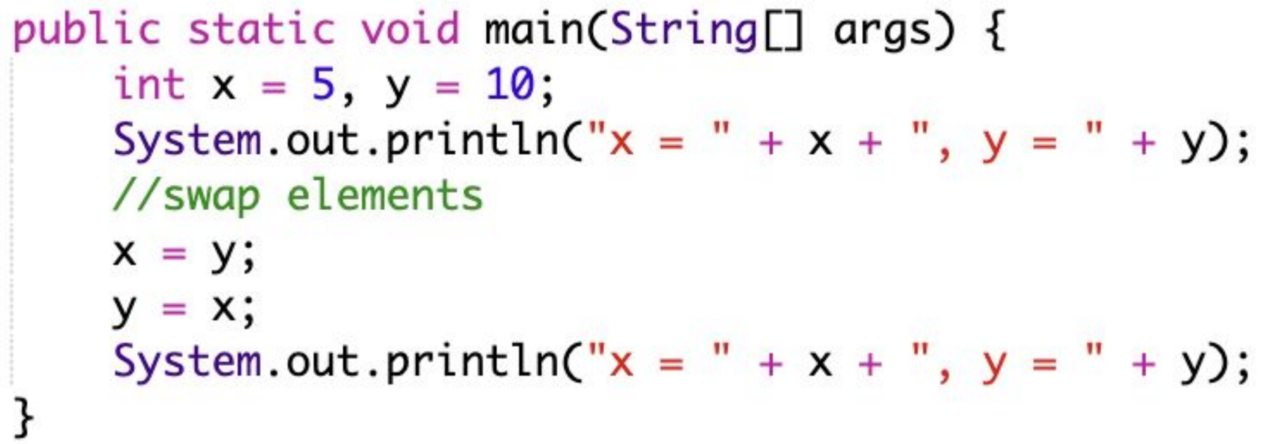
Alternatively, you can keep your eyes open and write down as many things you remember on a piece of paper.

This will help strengthen your memory of key concepts 💪

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 1 *(GenAI 🟠 Orange)*

Write a trace table for the following algorithm. Does it work as intended ? If it doesn't can you explain what the error is and how to fix it ?

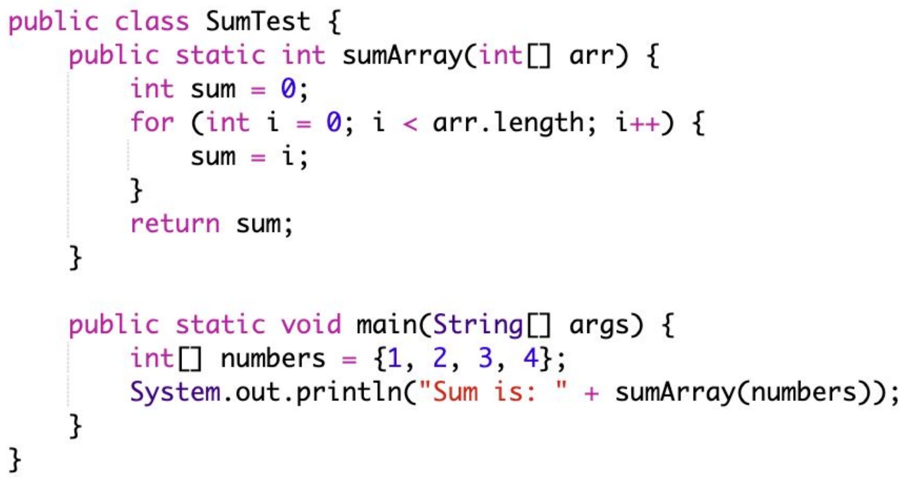


```java
public static void main(String[] args) {
    int x = 5, y = 10;
    System.out.println("x = " + x + ", y = " + y);
    //swap elements
    x = y;
    y = x;
    System.out.println("x = " + x + ", y = " + y);
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 2 *(GenAI 🟠 Orange)*

Write a trace table for the following algorithm. Does it work as intended ? If it doesn't, can you explain what the error is and how to fix it ?

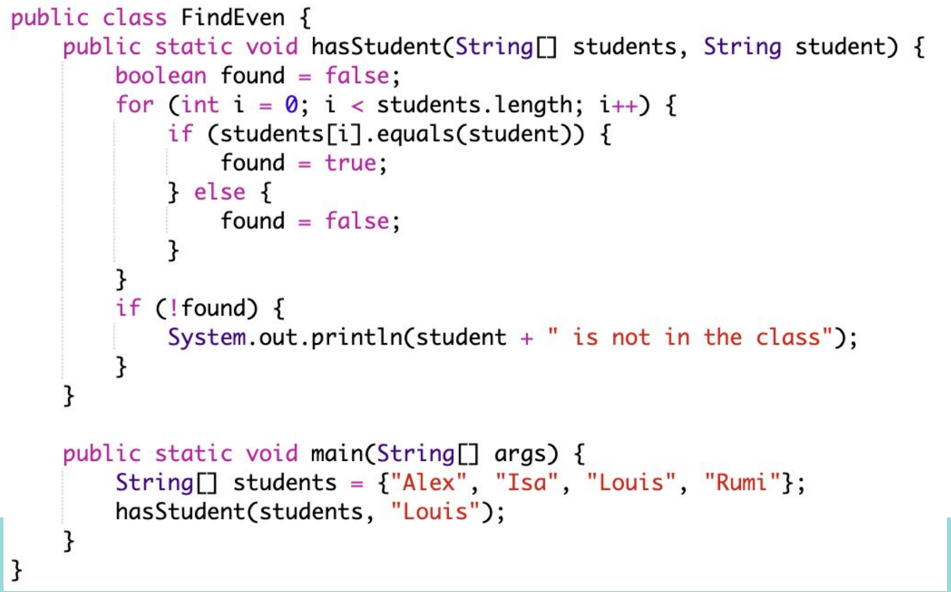


```java
public class SumTest {
    public static int sumArray(int[] arr) {
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum = i;
        }
        return sum;
    }

    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4};
        System.out.println("Sum is: " + sumArray(numbers));
    }
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 3 *(GenAI 🟠 Orange)*

Write a trace table for the following algorithm. Does it work as intended ? If it doesn’t, can you explain what the error is and how to fix it ?

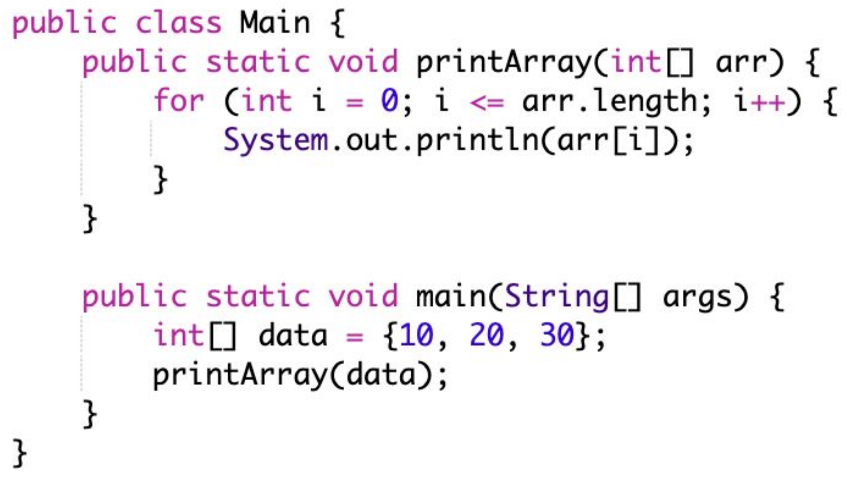


```java
public class FindEven {
    public static void hasStudent(String[] students, String student) {
        boolean found = false;
        for (int i = 0; i < students.length; i++) {
            if (students[i].equals(student)) {
                found = true;
            } else {
                found = false;
            }
        }
        if (!found) {
            System.out.println(student + " is not in the class");
        }
    }

    public static void main(String[] args) {
        String[] students = {"Alex", "Isa", "Louis", "Rumi"};
        hasStudent(students, "Louis");
    }
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 4 *(GenAI 🟠 Orange)*

Write a trace table for the following algorithm. Does it work as intended ? If it doesn't, can you explain what the error is and how to fix it ?

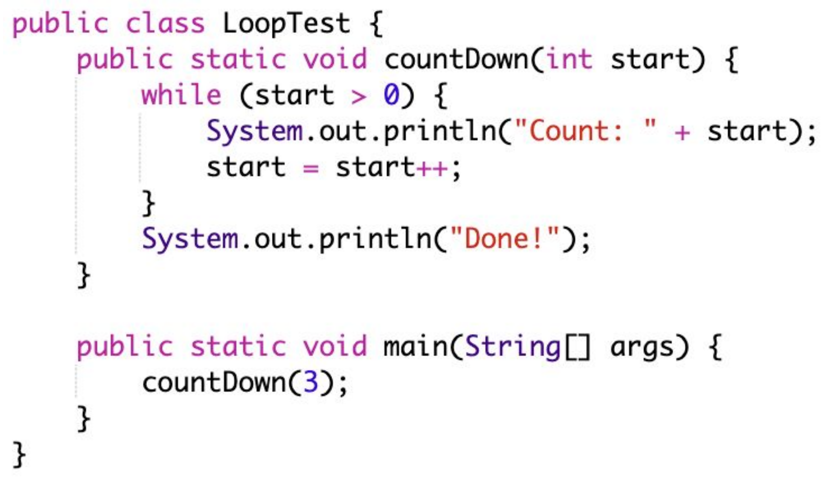


```java
public class Main {
    public static void printArray(int[] arr) {
        for (int i = 0; i <= arr.length; i++) {
            System.out.println(arr[i]);
        }
    }

    public static void main(String[] args) {
        int[] data = {10, 20, 30};
        printArray(data);
    }
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 5 *(GenAI 🟠 Orange)*

Write a trace table for the following algorithm. Does it work as intended ? If it doesn’t, can you explain what the error is and how to fix it ?



```java
public class LoopTest {
    public static void countDown(int start) {
        while (start > 0) {
            System.out.println("Count: " + start);
            start = start++;
        }
        System.out.println("Done!");
    }

    public static void main(String[] args) {
        countDown(3);
    }
}
```

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 6 *(GenAI 🟠 Orange)*

Explain how exceptions differ from errors.

Explain why it is important to handle exceptions in your code.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 7 *(GenAI 🟠 Orange)*

Write a program that asks the user to input two integers and divides the first by the second. Use a try/catch block to handle division by zero. Print a message when an exception occurs.

Bonus: Add a finally block that prints "Operation completed" regardless of exception.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 8 *(GenAI 🟠 Orange)*

Write a method that takes a String input and attempts to convert it to an integer using `Integer.parseInt()`. Use a try/catch to catch `NumberFormatException` and print "Invalid number format". Use a finally block to print "Parsing attempt finished".

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 9 *(GenAI 🟠 Orange)*

1. Checks if a file named password.txt exists in the current working directory.
2. If the file does not exist, create it.
3. Generate 10 random strings, each 8 characters long. You can use letters and digits.
4. Write these 10 random strings into the password.txt file, one string per line.
5. Use try/catch blocks to handle potential IOExceptions.
6. Use a finally block to close any open resources and print "File writing operation completed" regardless of success or failure.

Hints:

- Use java.io.File to check file existence.
- Use java.io.FileWriter or BufferedWriter to write to the file.
- Use java.util.Random or java.util.UUID for generating random strings.
- Remember to properly close file streams in the finally block

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 10 *(GenAI 🟠 Orange)*

Implement all the Unit tests that are relevant for the hasNumber() method to cover normal, abnormal and extreme cases.

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Exercise 11 & co *(GenAI 🟠 Orange)*

Many more exercises on code wars !

https://www.codewars.com

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**

Homework

Finish all the exercises

Create flashcards for the following terms:
Logic error
Runtime error
Exception
Exception handling
Debugging
Trace table
Breakpoint
try/catch/finally
IDE

Errors and Testing © 2025 by **Camille Duquesne** is licensed under **CC BY-NC-SA 4.0**