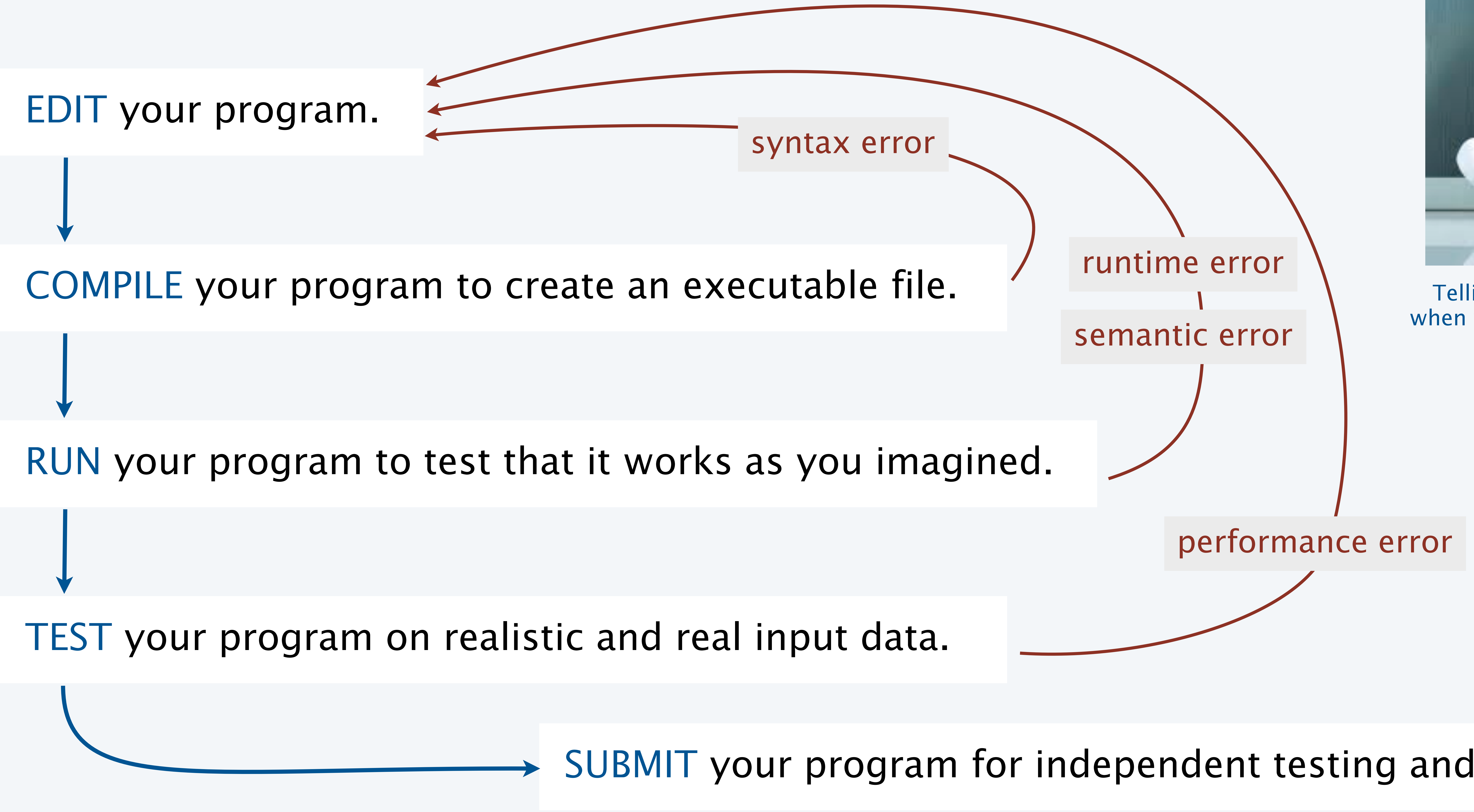
Unit Testing (Mini Lecture)

CS 121: Data Structures

Debugging your program: summary

Program development is a *four*-step process, with feedback.



EDIT your program.

COMPILE your program to create an executable file.

RUN your program to test that it works as you imagined.

TEST your program on realistic and real input data.

SUBMIT your program for independent testing and approval.

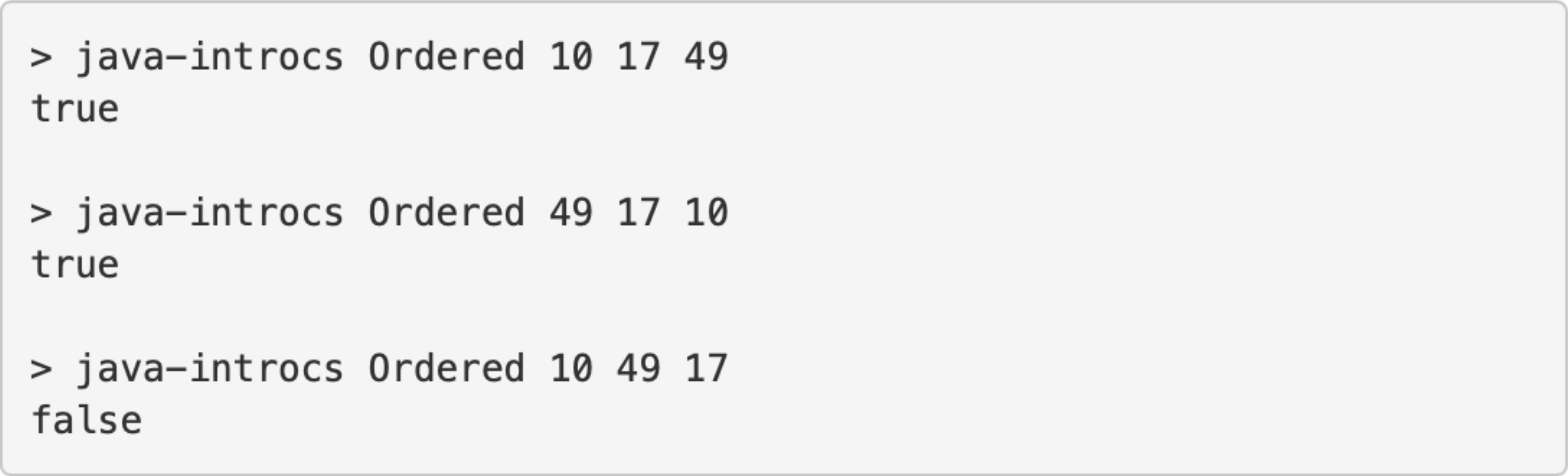
Telling a computer what to do
when you know what you're doing

Ordered.java, from HW1

“Write a program Ordered.java that takes three integer command-line arguments, x, y, and z. Define a boolean variable whose value is true if the three values are either in strictly ascending order (x < y < z) or in strictly descending order (x > y > z), and false otherwise. Then, print this boolean value.”

How can we tell if our programs are correct?

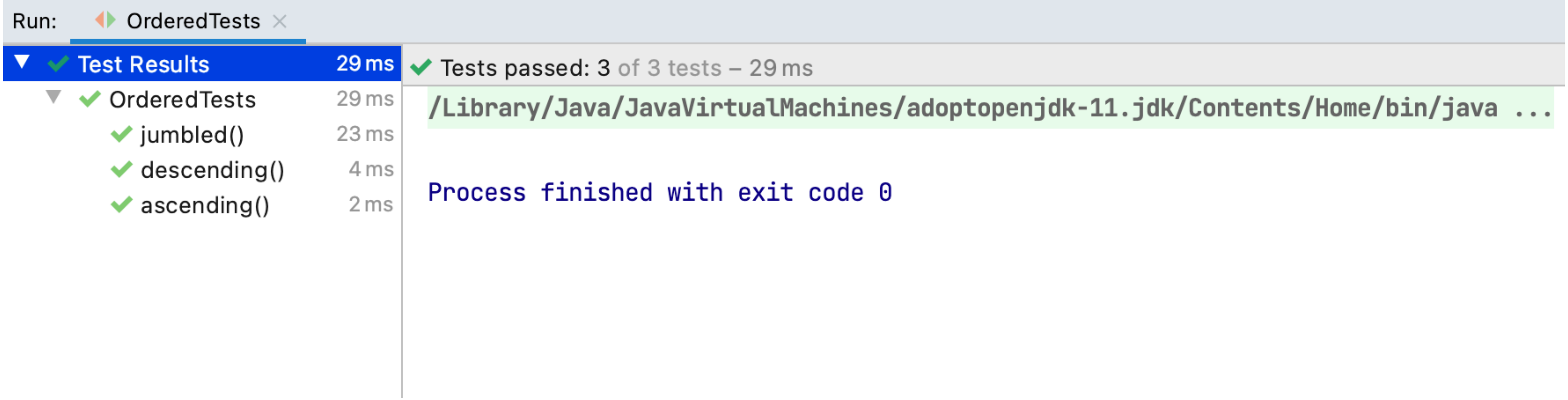
- Testing!
    - Good: Running the program ourselves, with **manually entered test data**



```
> java-introcs Ordered 10 17 49
true

> java-introcs Ordered 49 17 10
true

> java-introcs Ordered 10 49 17
false
```

How can we tell if our programs are correct?

- Testing!
  - Good: Running the program ourselves, with **manually entered test data**
  - Better: **Automatically running the program multiple times**, with different combinations of test data



```
▼ ✓ Test Results        29 ms
  ▼ ✓ OrderedTests      29 ms
      ✓ jumbled()       23 ms
      ✓ descending()     4 ms
      ✓ ascending()      2 ms
```

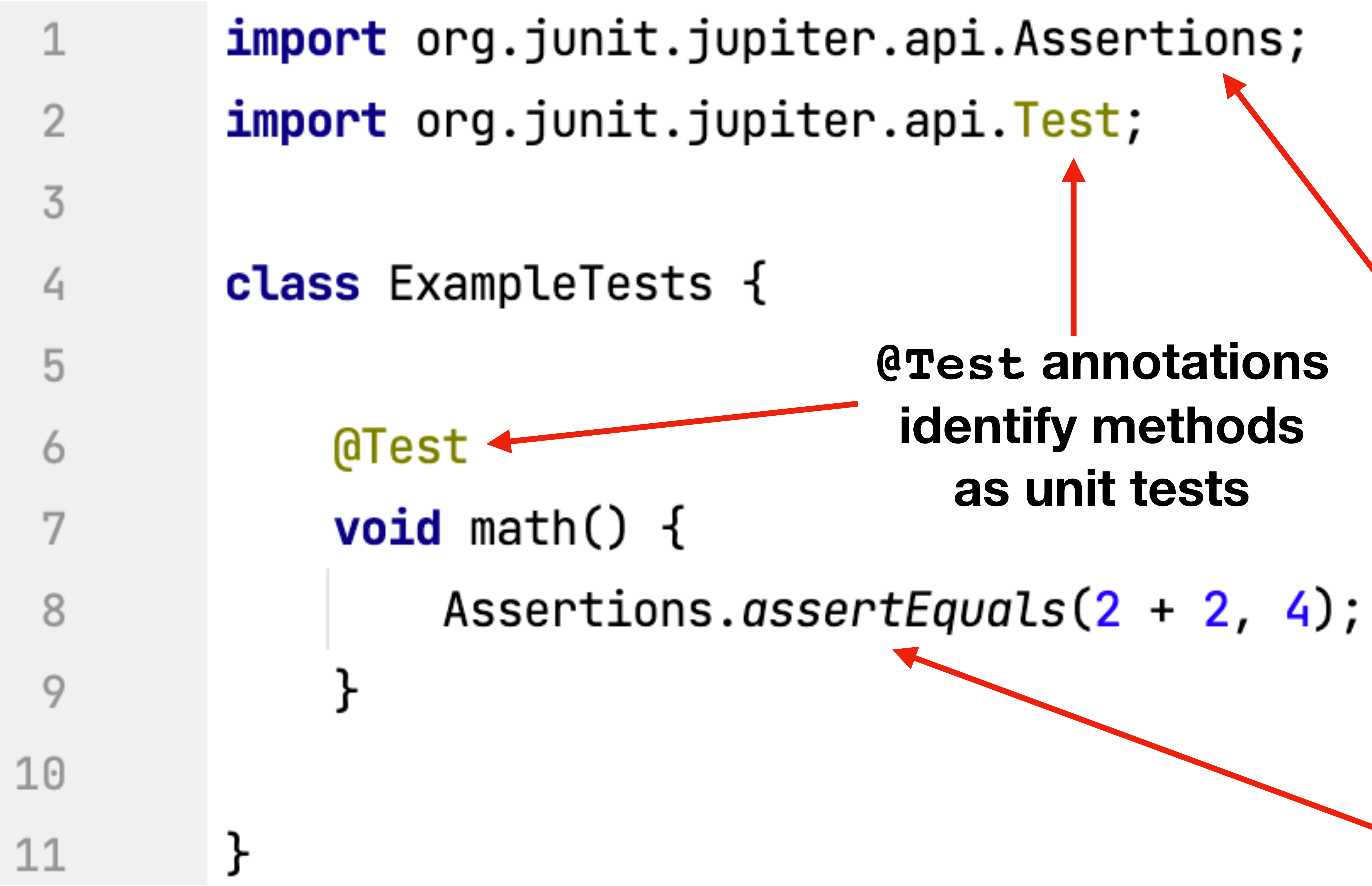
```
✓ Tests passed: 3 of 3 tests – 29 ms
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Automated Testing: Unit Testing

- A unit testing framework will execute a method multiple times, with different inputs, and check the outputs

- If the outputs differ from what it expects, **the program is wrong**

Anatomy of a Unit Test

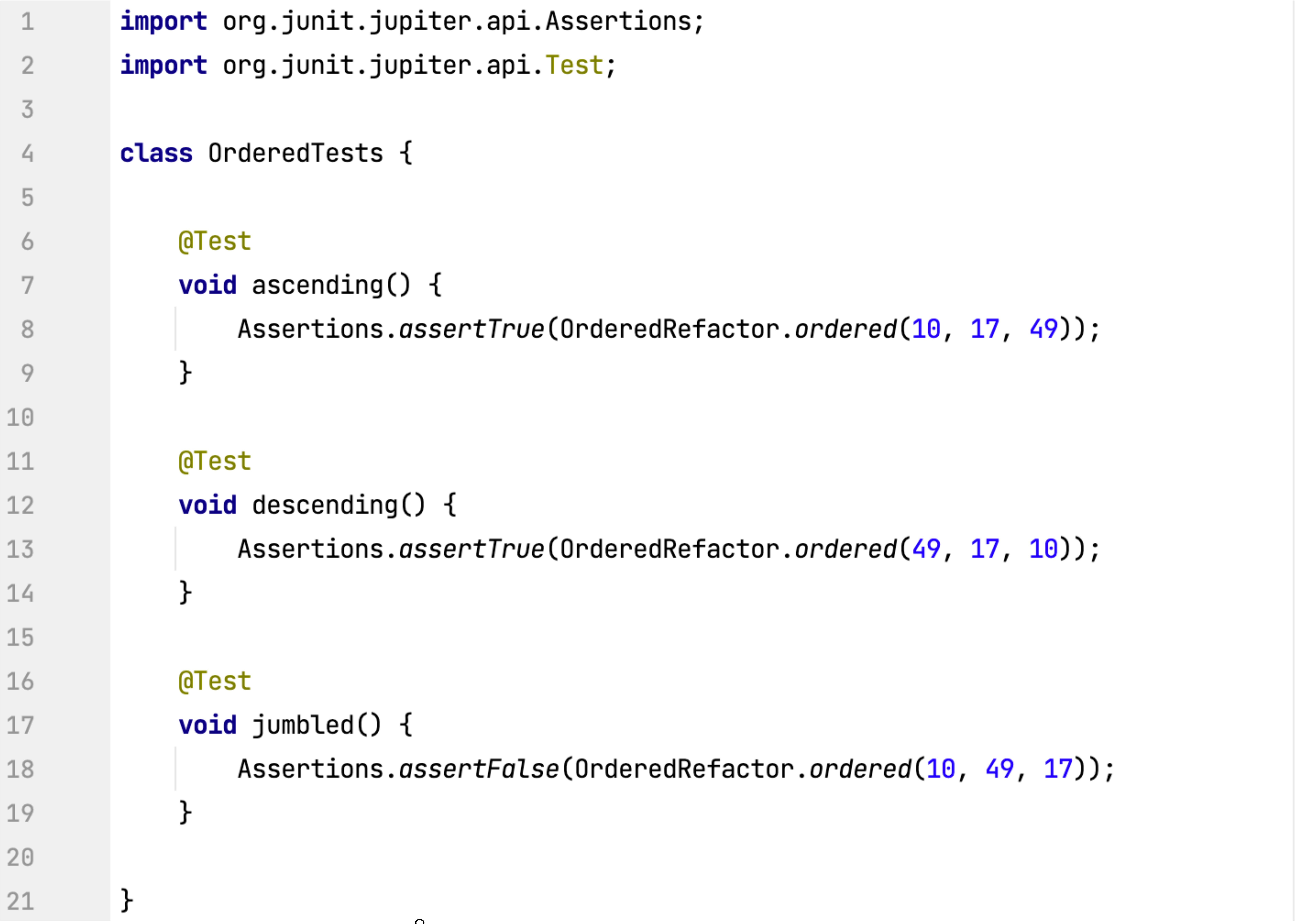


```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class ExampleTests {

    @Test
    void math() {
        Assertions.assertEquals(2 + 2, 4);
    }

}
```

`@Test` annotations identify methods as unit tests

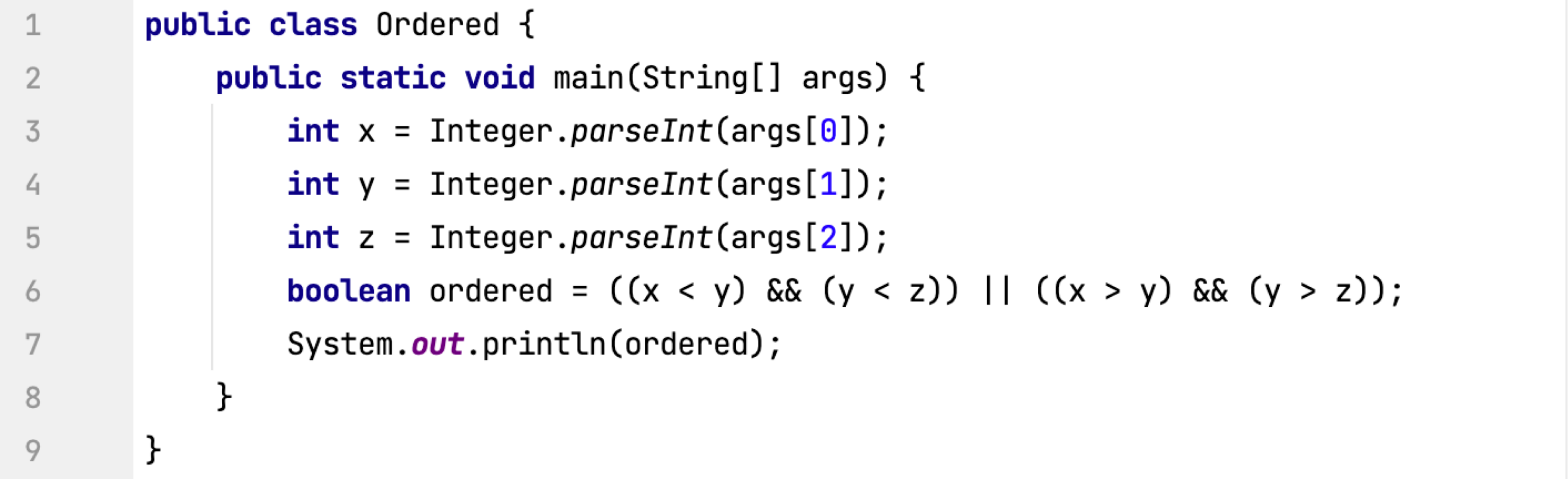
Unit tests should have one or more assertions. If an assertion "fails," the test "fails."

Unit Testing Ordered.java



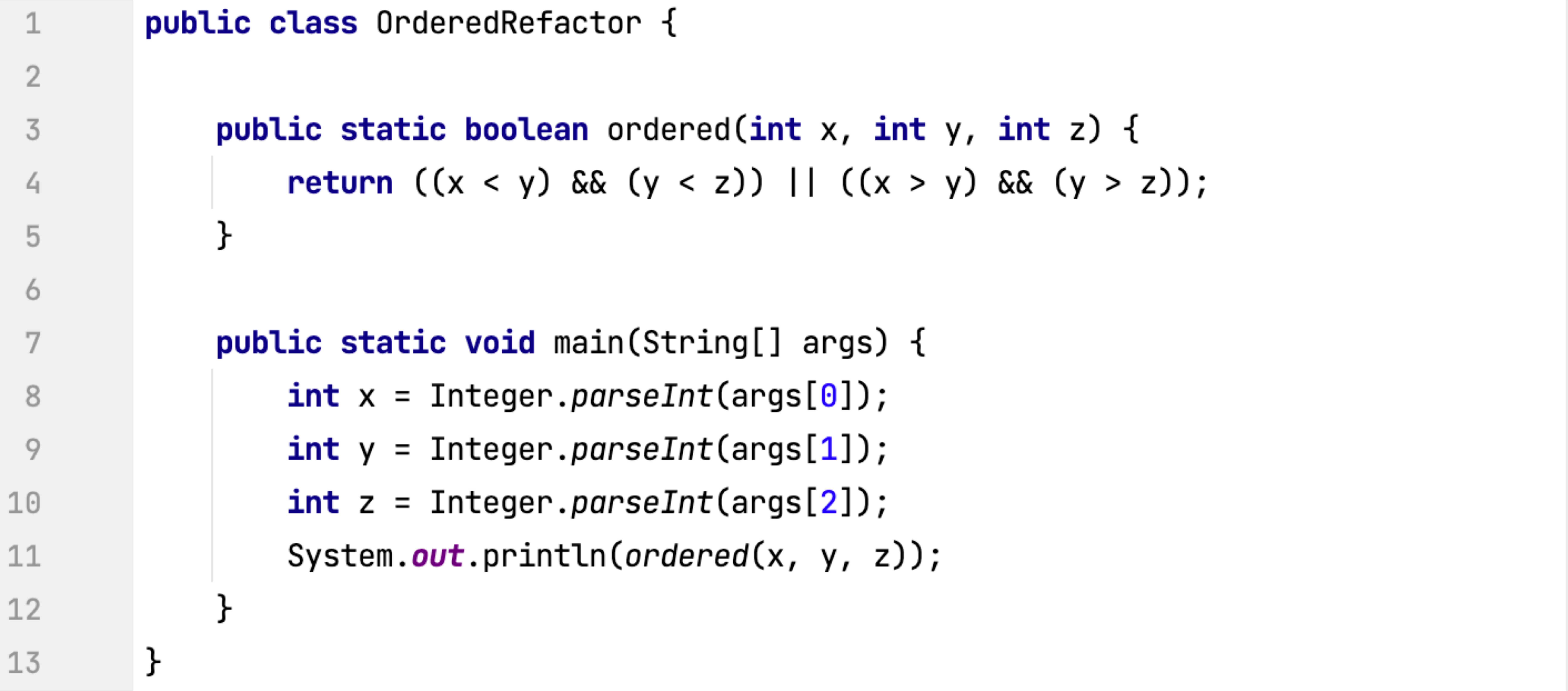
```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

class OrderedTests {

    @Test
    void ascending() {
        Assertions.assertTrue(OrderedRefactor.ordered(10, 17, 49));
    }

    @Test
    void descending() {
        Assertions.assertTrue(OrderedRefactor.ordered(49, 17, 10));
    }

    @Test
    void jumbled() {
        Assertions.assertFalse(OrderedRefactor.ordered(10, 49, 17));
    }

}
```

Initial Ordered.java



```java
public class Ordered {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        int z = Integer.parseInt(args[2]);
        boolean ordered = ((x < y) && (y < z)) || ((x > y) && (y > z));
        System.out.println(ordered);
    }
}
```

Refactored Ordered.java



```java
public class OrderedRefactor {

    public static boolean ordered(int x, int y, int z) {
        return ((x < y) && (y < z)) || ((x > y) && (y > z));
    }

    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        int z = Integer.parseInt(args[2]);
        System.out.println(ordered(x, y, z));
    }
}
```

How can we tell if our programs are correct?

- Testing!
  - Good: Running the program ourselves, with **manually entered test data**
  - Better: **Automatically running the program multiple times**, with different combinations of test data
  - Best: Write tests **before you write your program** (test-driven development)
    - In TDD, tests describe what the program should do, before you even start writing the program