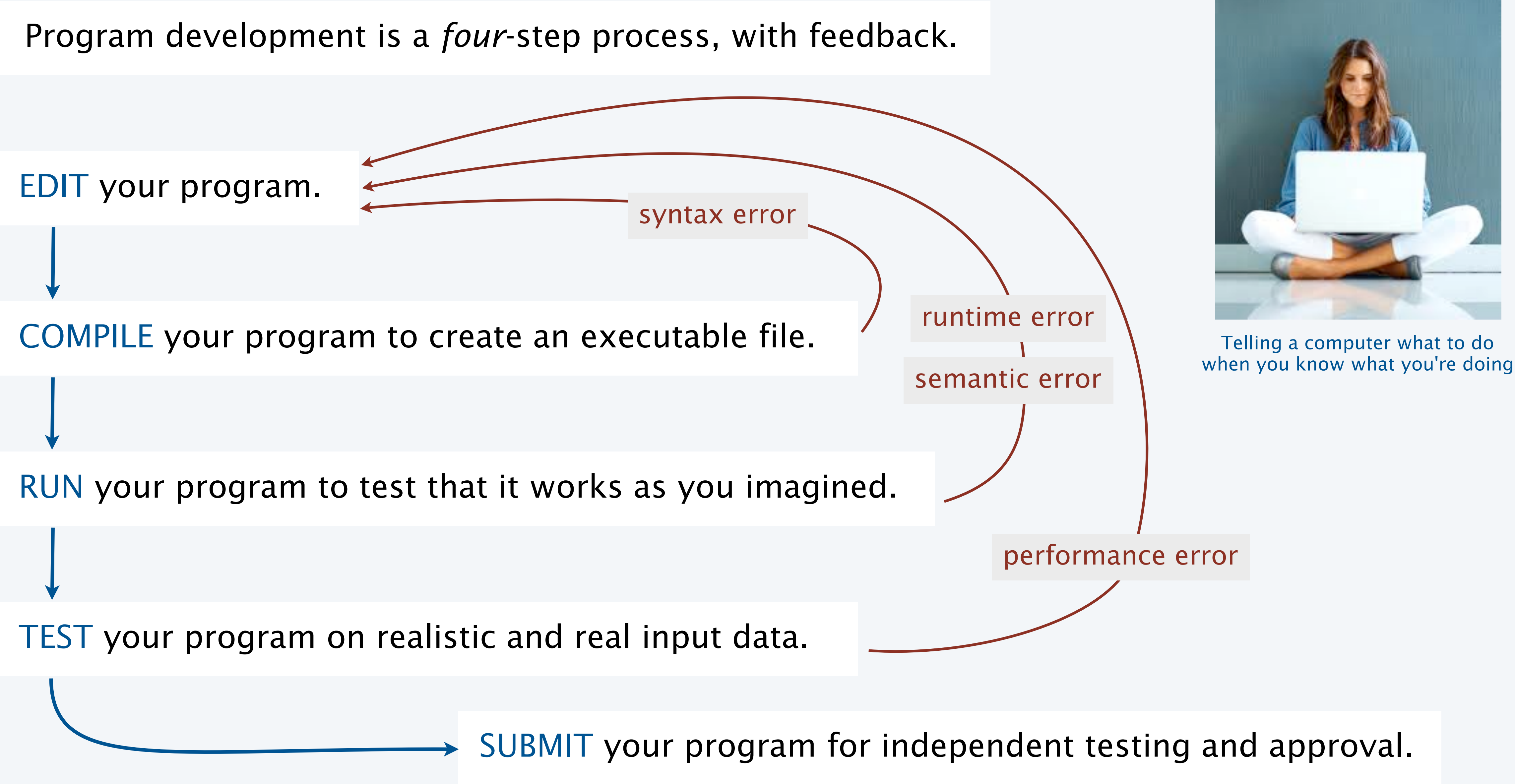


Unit Testing (Mini Lecture)

CS 121: Data Structures

Debugging your program: summary



How can we tell if our programs are correct?

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

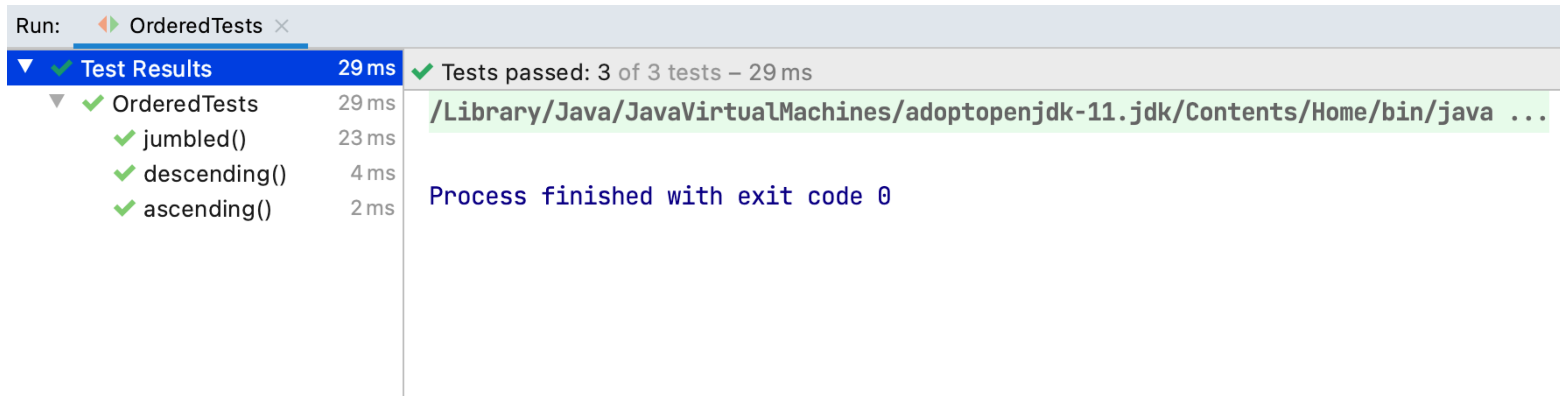
```
> java-introcs Ordered 10 17 49  
true
```

```
> 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



The screenshot shows an IDE's Run window for a test named 'OrderedTests'. The window is divided into two main sections. The left section, titled 'Test Results', shows a tree view of the test execution. The right section displays the command used to run the test and the final status.

Test Name	Duration
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

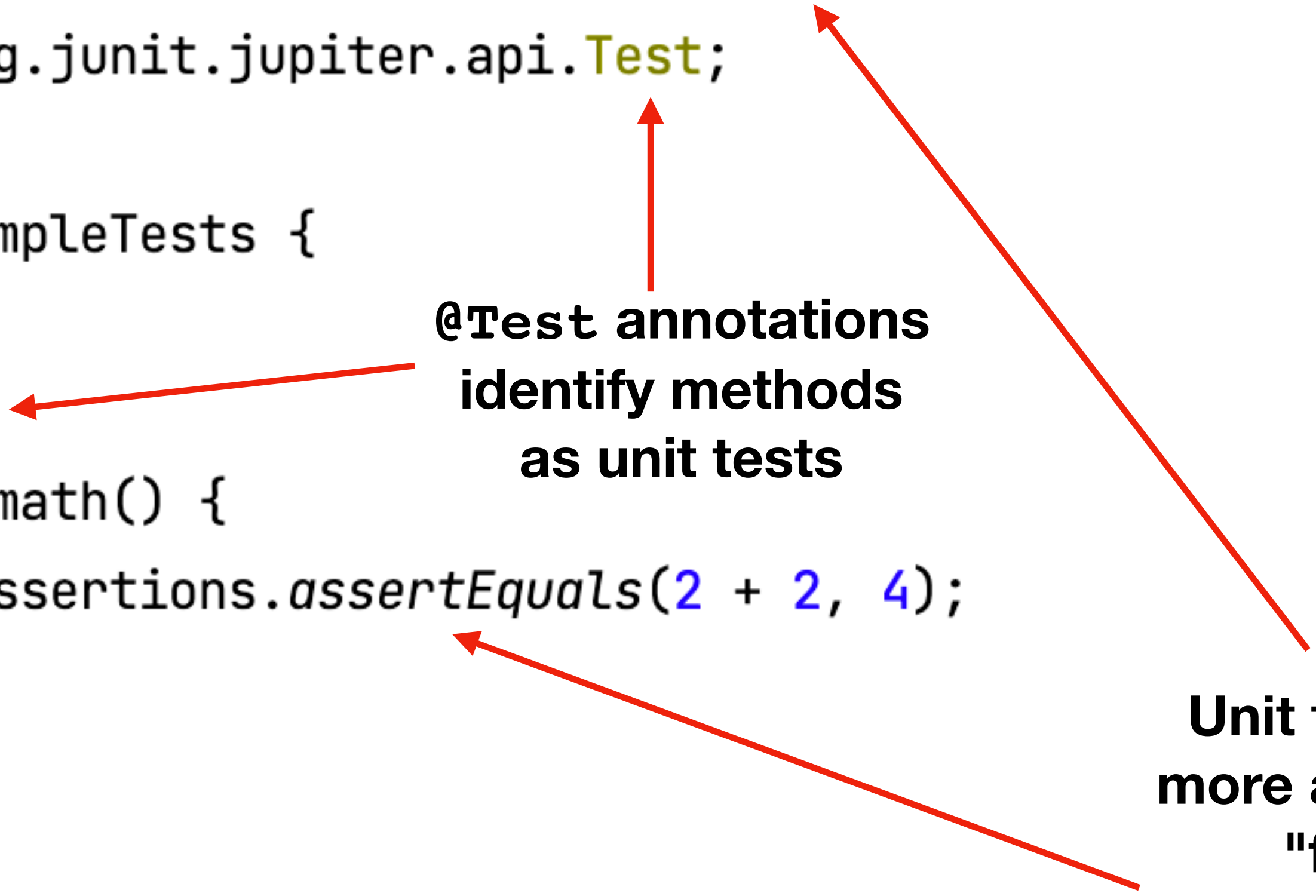
Automated Testing: Unit Testing

- 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

```
1  import org.junit.jupiter.api.Assertions;
2  import org.junit.jupiter.api.Test;
3
4  class ExampleTests {
5
6      @Test
7      void math() {
8          Assertions.assertEquals(2 + 2, 4);
9      }
10
11 }
```

**@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

```
1  import org.junit.jupiter.api.Assertions;
2  import org.junit.jupiter.api.Test;
3
4  class OrderedTests {
5
6      @Test
7      void ascending() {
8          Assertions.assertTrue(OrderedRefactor.ordered(10, 17, 49));
9      }
10
11     @Test
12     void descending() {
13         Assertions.assertTrue(OrderedRefactor.ordered(49, 17, 10));
14     }
15
16     @Test
17     void jumbled() {
18         Assertions.assertFalse(OrderedRefactor.ordered(10, 49, 17));
19     }
20
21 }
```

Initial Ordered.java

```
1 public class Ordered {  
2     public static void main(String[] args) {  
3         int x = Integer.parseInt(args[0]);  
4         int y = Integer.parseInt(args[1]);  
5         int z = Integer.parseInt(args[2]);  
6         boolean ordered = ((x < y) && (y < z)) || ((x > y) && (y > z));  
7         System.out.println(ordered);  
8     }  
9 }
```


Refactored Ordered.java

```
1 public class OrderedRefactor {  
2  
3     public static boolean ordered(int x, int y, int z) {  
4         return ((x < y) && (y < z)) || ((x > y) && (y > z));  
5     }  
6  
7     public static void main(String[] args) {  
8         int x = Integer.parseInt(args[0]);  
9         int y = Integer.parseInt(args[1]);  
10        int z = Integer.parseInt(args[2]);  
11        System.out.println(ordered(x, y, z));  
12    }  
13 }
```

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