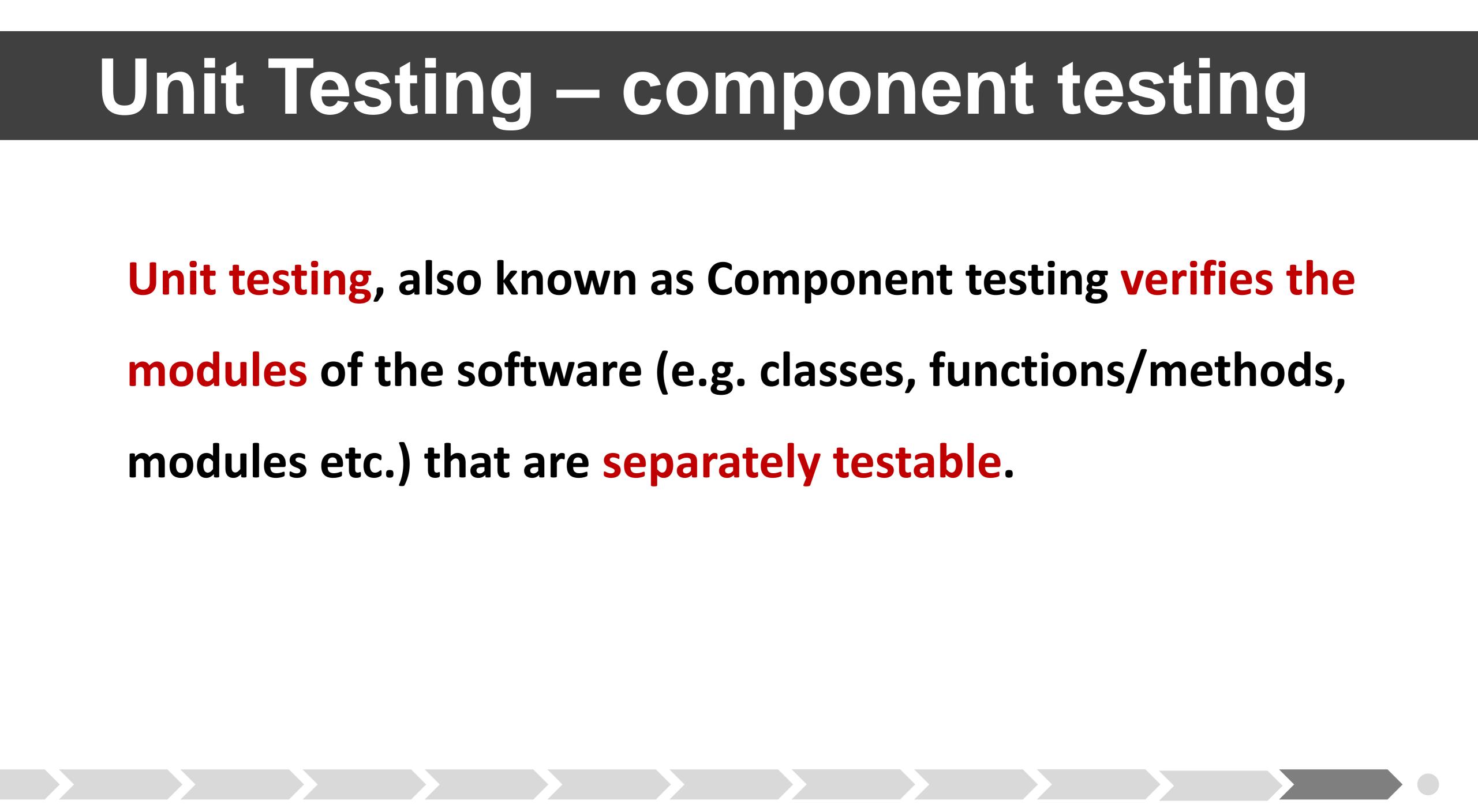
Unit Testing – component testing

modules etc.) that are separately testable.

- Unit testing, also known as Component testing verifies the
- modules of the software (e.g. classes, functions/methods,



Unit Testing – component testing

test.

Unit test framework support the developer.

system.

simulate the interface between the software components.

The developer writes code to test modules in the software under

Unit testing should be done in isolation from the rest of the

Stubs and drivers are used to replace the missing software and



Unit Testing – component testing

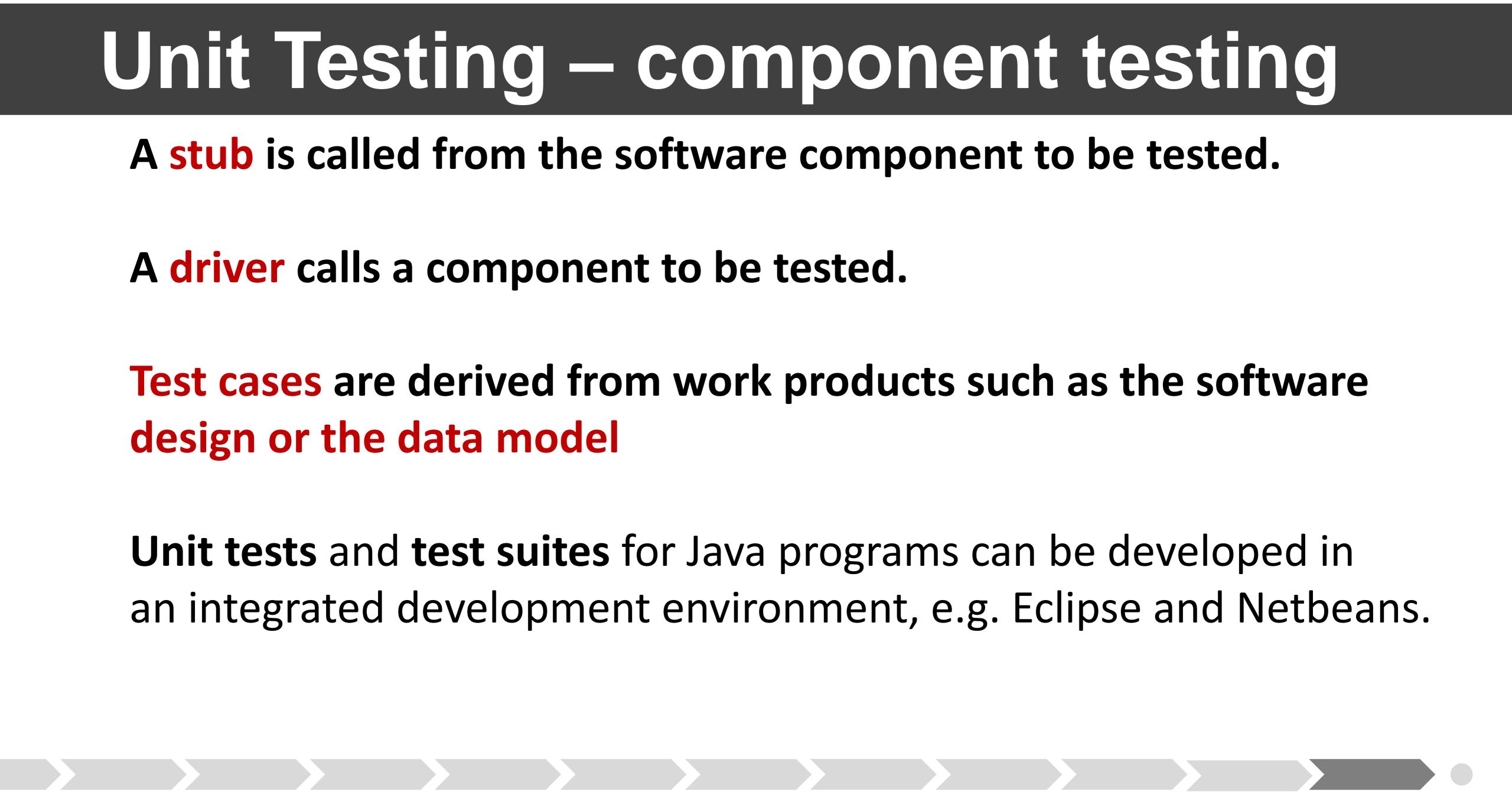
A stub is called from the software component to be tested.

A driver calls a component to be tested.

design or the data model

- **Test cases** are derived from work products such as the software

Unit tests and test suites for Java programs can be developed in an integrated development environment, e.g. Eclipse and Netbeans.



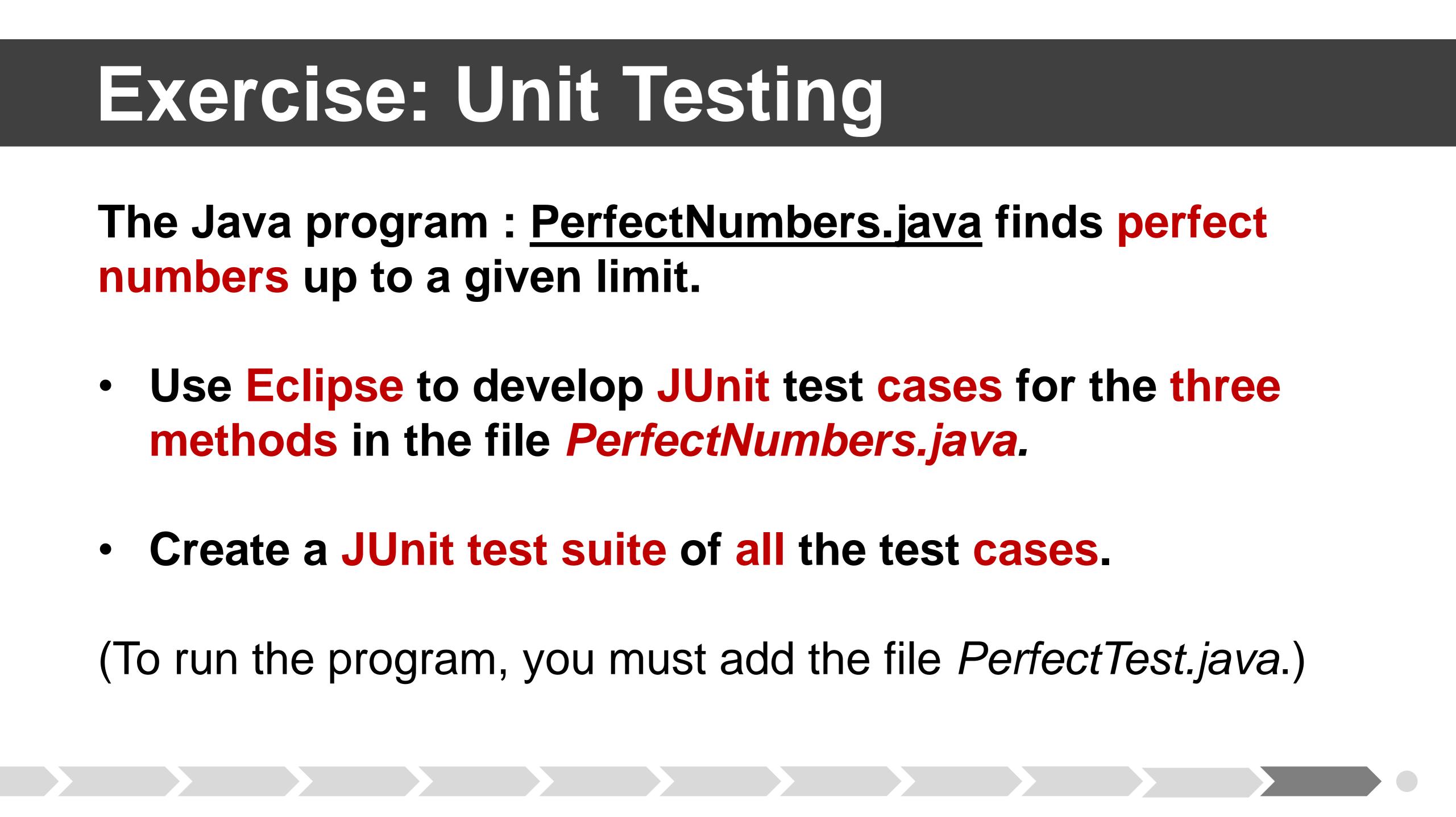
numbers up to a given limit.

- methods in the file PerfectNumbers.java.
- Create a JUnit test suite of all the test cases.

(To run the program, you must add the file *PerfectTest.java*.)

The Java program : <u>PerfectNumbers.java</u> finds perfect

Use Eclipse to develop JUnit test cases for the three



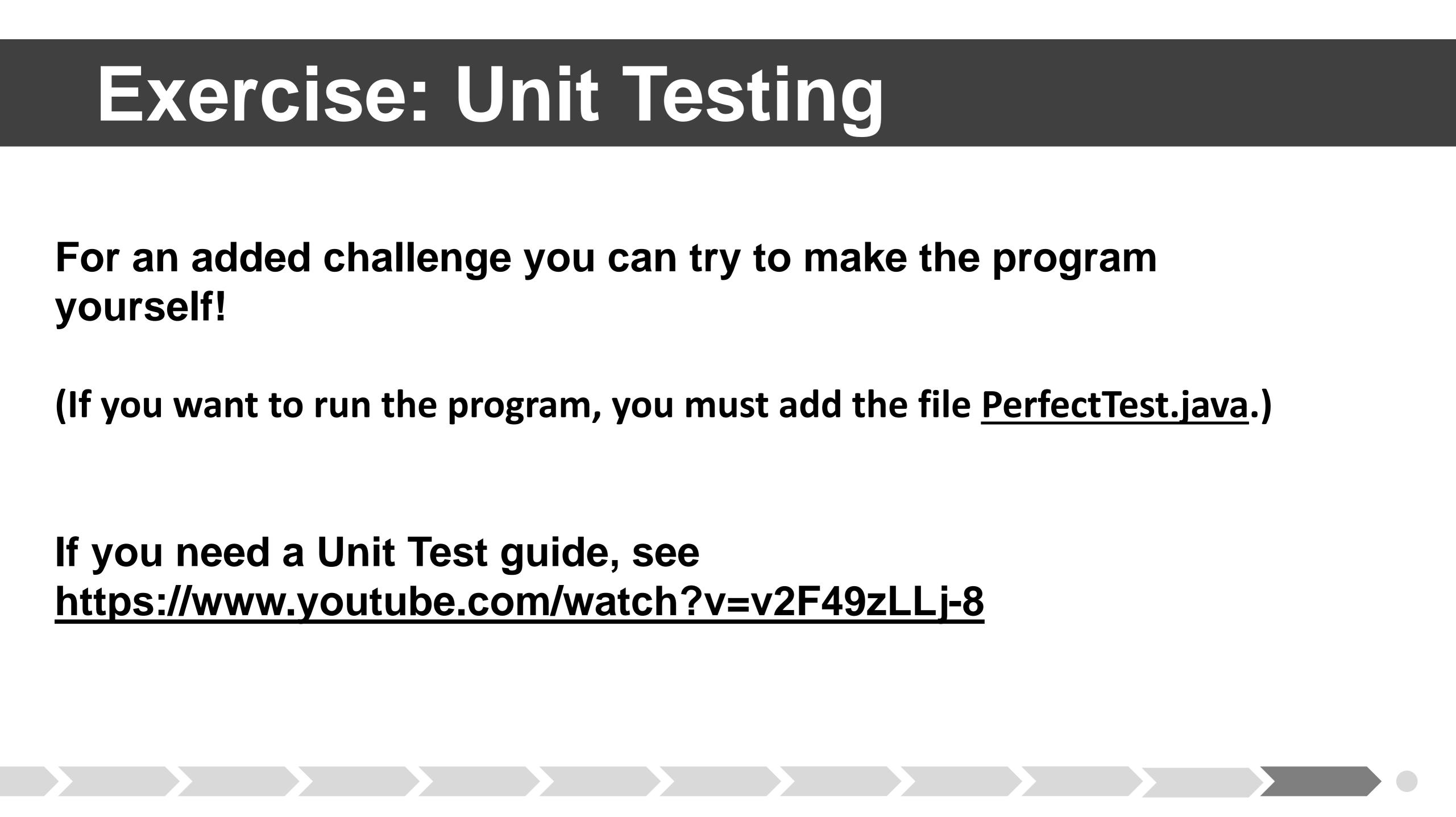
For an added challenge you can try to make the program yourself!

(If you want to run the program, you must add the file PerfectTest.java.)

If you need a Unit Test guide, see https://www.youtube.com/watch?v=v2F49zLLj-8







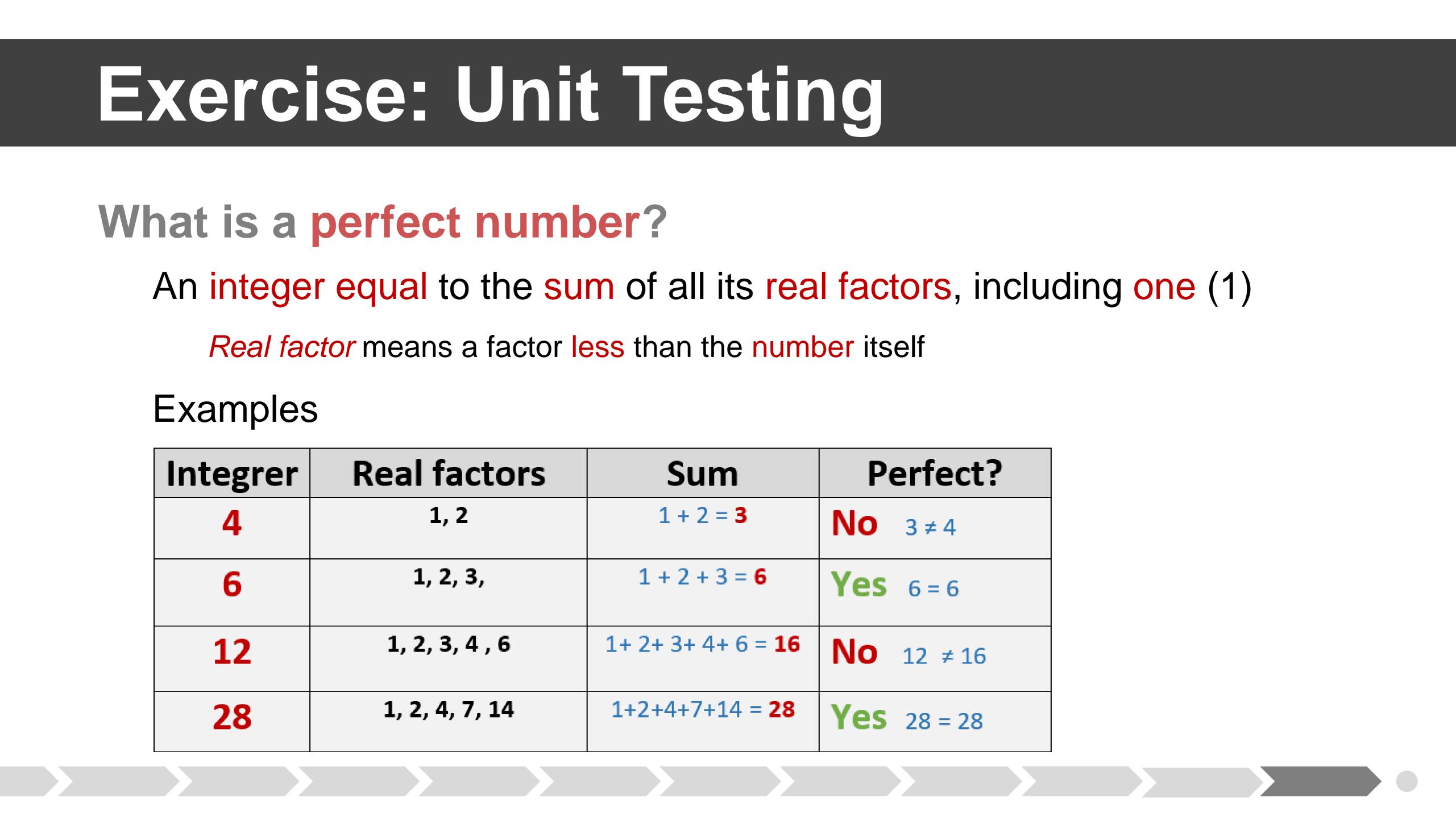
What is a perfect number?

Real factor means a factor less than the number itself

Examples

Integrer	Real factors	Sum	Perfect?
4	1, 2	1 + 2 = 3	No 3 ≠ 4
6	1, 2, 3,	1 + 2 + 3 = 6	Yes 6 = 6
12	1, 2, 3, 4 , 6	1+ 2+ 3+ 4+ 6 = 16	NO 12 ≠ 16
28	1, 2, 4, 7, 14	1+2+4+7+14 = 28	Yes 28 = 28

An integer equal to the sum of all its real factors, including one (1)



PerfectNumbers.java

Calculates perfect numbers

perfect(int number): boolean

Is the given number perfect?

factorSum(int number): String

Calculate factor sum of number

findPerfectNumbers(int limit)

Find perfect numbers given limit

public class PerfectNumbers {

```
public static boolean perfect( int number ) {
int factorSum = 1;
for ( int divisor = 2; divisor <= number / 2; divisor++ ) {</pre>
  if ( number % divisor == 0 )
    factorSum += divisor;
return (factorSum == number);
```

```
public static String factorSum( int number ) {
 String sum = "1";
 for ( int divisor = 2; divisor <= number / 2; divisor++ ) {</pre>
    if ( number % divisor == 0 ) {
      sum += " + " + divisor;
 return sum;
```

```
public static String findPerfectNumbers( int limit ) {
 String result = "perfect number less or equals " + limit + "\n";
 for ( int i = 2; i <= limit; i++ ) {</pre>
   if ( perfect( i ) ) {
     result += i + " = " + factorSum( i ) + "\n";
  1
 return result;
```









Testing *perfect(int number)* What to test?

Confirm perfect number is perfect

Chosen number: 6

Variables

result \rightarrow Holds the returned value

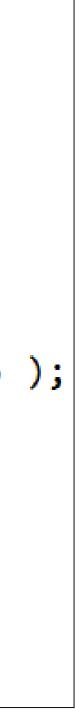
expected \rightarrow Set to true

Asser

Check that the two values match



import static org.junit.Assert.*; import org.junit.Test; public class PerfectTest1 { @Test public void perfectTest1() { boolean result = PerfectNumbers.perfect(6); boolean expected = true; assertEquals(result, expected);





Testing *perfect(int number)* What to test?

Confirm non-perfect is non-perfect

Chosen number: 7

Variables

result \rightarrow Holds the returned value

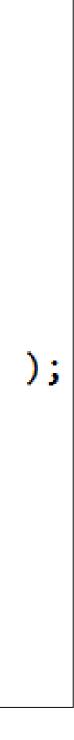
expected \rightarrow Set to false

Asser

Check that the two values match



import static org.junit.Assert.*; import org.junit.Test; public class PerfectTest2 { @Test public void perfectTest2() { boolean result = PerfectNumbers.perfect(7); boolean expected = false; assertEquals(result, expected);





Testing *factorSum(int number)*

What to test?

Confirm correct sum of factors

Chosen number: 6

Variables

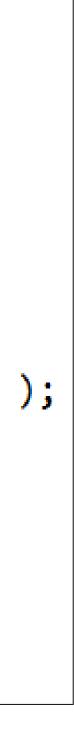
result \rightarrow Holds factor sum of 6

expected \rightarrow Set to "1 + 2 + 3"

Assert

Check that the two values match

```
import static org.junit.Assert.*;
import org.junit.Test;
public class FactorSumTest {
   @Test
    public void test() {
     String result = PerfectNumbers.factorSum( 6 );
     String expected = "1 + 2 + 3";
     assertEquals(expected, result);
```



Testing findPerfectNumbers(int limit) What to test?

Confirm correct retrieval of PN

Chosen number: 1000

Variables

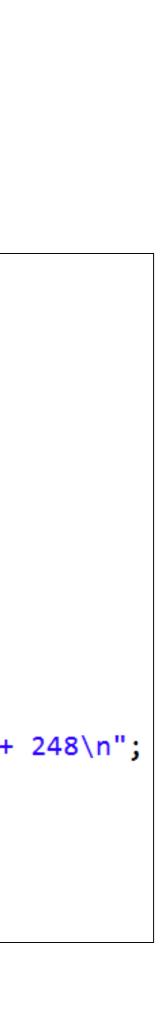
result \rightarrow Holds all PN within limit

expected \rightarrow Set to 6, 28, and 496

Assert

Check that the two values match

```
import static org.junit.Assert.*;
import org.junit.Test;
public class FindPerfectNumberTest {
   @Test
   public void findPerfectNumberTest() {
    String result = PerfectNumbers.findPerfectNumbers( 1000 );
     String expected = "perfect number less or equals 1000" +
                        "\n6 = 1 + 2 + 3" +
                        \sqrt{n496} = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248 n'';
       assertEquals(expected, result);
```



JUnit Test Suite for all test cases Where to place test suite? AllTests.java import org.junit.runner.RunWith; import org.junit.runners.Suite; @RunWith(Suite.class) import org.junit.runners.Suite.SuiteClasses; @RunWith(Suite.class) What to include? @SuiteClasses({ FactorSumTest.class, FindPerfectNumberTest.class, PerfectTest1.class, PerfectTest2.class}) PerfectTest1.java public class AllTests { } PerfectTest2.java FactorSumTest.java FindPerfectNumberTest.java



