

# Unit Testing – component testing

**Unit testing**, also known as Component testing **verifies the modules** of the software (e.g. classes, functions/methods, modules etc.) that are **separately testable**.



# Unit Testing – component testing

The **developer** writes **code to test modules** in the software under test.

**Unit test framework** support the developer.

Unit testing should be done **in isolation** from the rest of the system.

**Stubs** and **drivers** are used to **replace the missing software** and **simulate** the interface between the software components.



# Unit Testing – component testing

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

A **driver** calls a component to be tested.

**Test cases** are derived from work products such as the software design or the data model

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



# Exercise: Unit Testing

The Java program : PerfectNumbers.java finds **perfect numbers** up to a given limit.

- Use **Eclipse** to develop **JUnit** test cases for the **three 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*.)



# Exercise: Unit Testing

**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>**



# Exercise: Unit Testing

What is a **perfect number**?

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

*Real factor* means a factor **less** than the **number** itself

Examples

Integer	Real factors	Sum	Perfect?
<b>4</b>	1, 2	$1 + 2 = 3$	<b>No</b> $3 \neq 4$
<b>6</b>	1, 2, 3,	$1 + 2 + 3 = 6$	<b>Yes</b> $6 = 6$
<b>12</b>	1, 2, 3, 4, 6	$1 + 2 + 3 + 4 + 6 = 16$	<b>No</b> $12 \neq 16$
<b>28</b>	1, 2, 4, 7, 14	$1 + 2 + 4 + 7 + 14 = 28$	<b>Yes</b> $28 = 28$

# Exercise: Unit Testing

## 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++ ) {  
            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++ ) {  
            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++ ) {  
            if ( perfect( i ) ) {  
                result += i + " = " + factorSum( i ) + "\n";  
            }  
        }  
        return result;  
    }
```

```
}
```

# Exercise 2: Unit Testing

## Testing *perfect*(int number)

### What to test?

Confirm perfect number is perfect

Chosen number: 6

### Variables

*result* → Holds the returned value

*expected* → Set to *true*

### Assert

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);

    }
}
```



# Exercise 2: Unit Testing

## Testing *perfect*(int number)

### What to test?

Confirm non-perfect is non-perfect

Chosen number: 7

### Variables

*result* → Holds the returned value

*expected* → Set to *false*

### Assert

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);

    }
}
```

# Exercise 2: Unit Testing

## Testing *factorSum(int number)*

### What to test?

Confirm correct sum of factors

Chosen number: 6

### Variables

*result* → Holds factor sum of 6

*expected* → 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);
    }
}
```

# Exercise 2: Unit Testing

## Testing *findPerfectNumbers(int limit)*

### What to test?

Confirm correct retrieval of PN

Chosen number: 1000

### Variables

*result* → Holds all PN within limit

*expected* → 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" +
            "\n28 = 1 + 2 + 4 + 7 + 14" +
            "\n496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248\n";

        assertEquals(expected, result);
    }
}
```

# Exercise 2: Unit Testing

## JUnit **Test Suite** for all test cases

**Where** to place test suite?

*AllTests.java*

**@RunWith**(Suite.class)

What to **include**?

*PerfectTest1.java*

*PerfectTest2.java*

*FactorSumTest.java*

*FindPerfectNumberTest.java*

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ FactorSumTest.class, FindPerfectNumberTest.class,
                PerfectTest1.class, PerfectTest2.class})

public class AllTests { }
```

