Génie logiciel pour la conception d'un Système d'Information

CSC4521

Voie d'Approfondissement
Intégration et Déploiement de Systèmes d'Information
(VAP DSI)

Proxy Design Pattern

http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC4521/

…/CSC4521/CSC4521-Proxy.pdf

The Proxy Design Pattern

cartoon stock.com

The Proxy Design Pattern

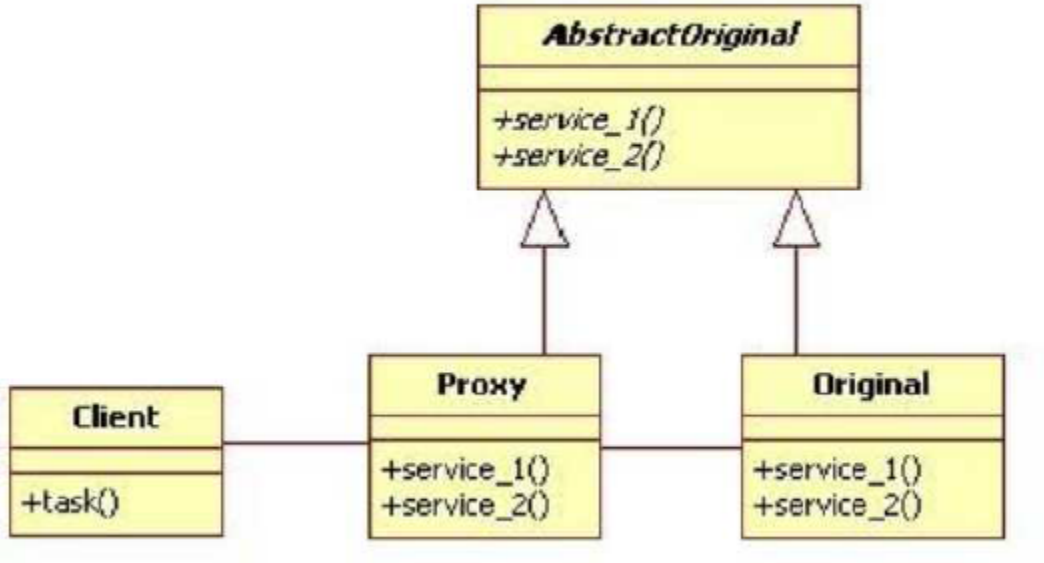
Proxy will almost certainly be useful in your **architectures**:

- **Logical**
- **Physical**

It may also be useful in **deployment** patterns

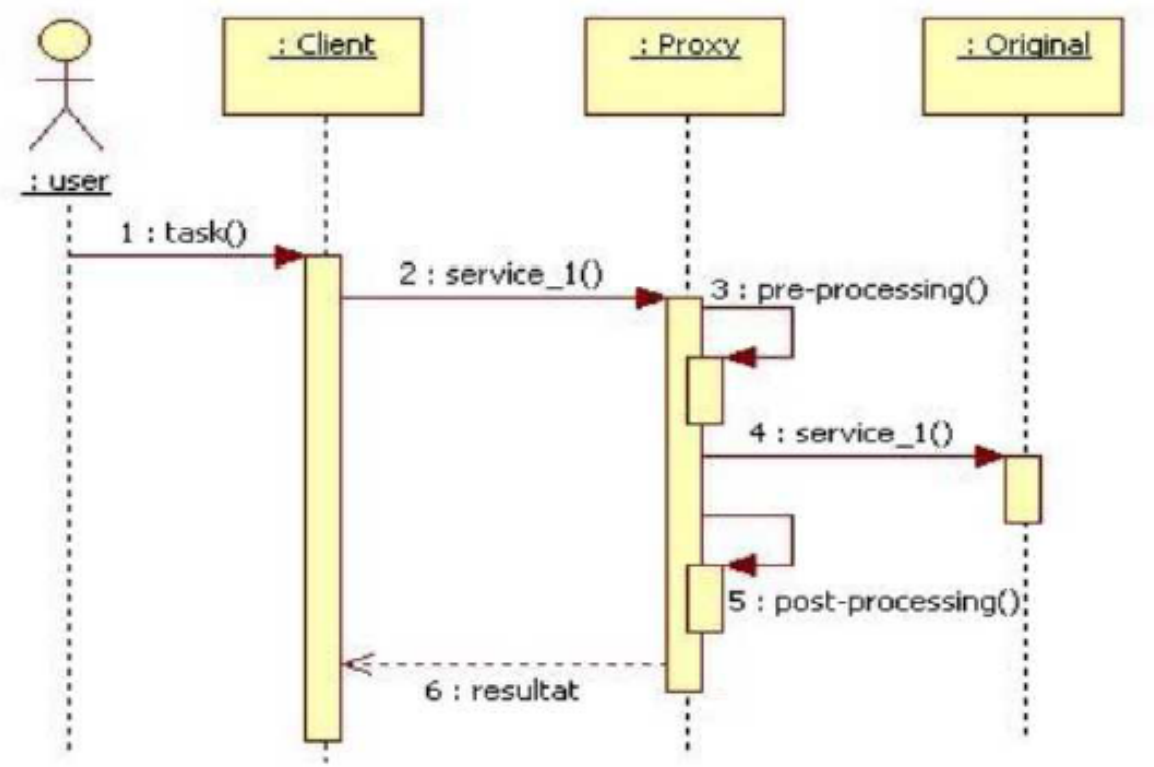
cartoon stock.com

Class Diagram - a proxy server offering 2 services



Patron: Proxy

Sequence Diagram - a use case task requires 1 of the proxy services



Here, the proxy also adds functionality (through pre/post processing)

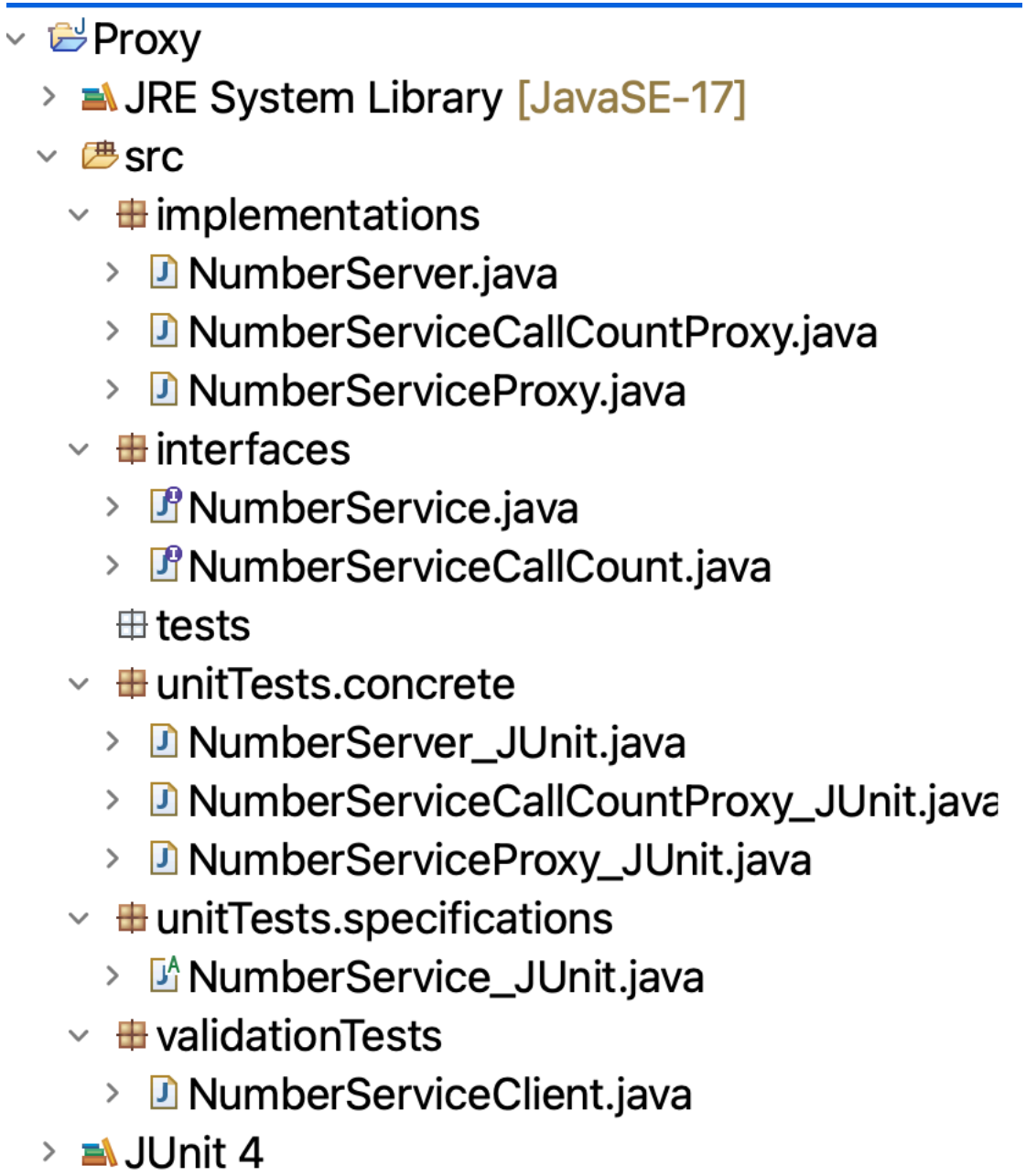
Proxy Problem I - Counter proxy (code provided)

Create a service, as a method of a Java class, that will:

- take an integer and return if it is even,
- take an integer and return if it is non-negative

Write a proxy that will count the number of times a service is called, and provides a method to reset the count

Code can be downloaded as a Java zip archive: Proxy.zip


Proxy
JRE System Library [JavaSE-17]
src
implementations
NumberServer.java
NumberServiceCallCountProxy.java
NumberServiceProxy.java
interfaces
NumberService.java
NumberServiceCallCount.java
tests
unitTests.concrete
NumberServer_JUnit.java
NumberServiceCallCountProxy_JUnit.java
NumberServiceProxy_JUnit.java
unitTests.specifications
NumberService_JUnit.java
validationTests
NumberServiceClient.java
JUnit 4

```java
package interfaces;

public interface NumberService {

  /**
   * check whether a number passed as argument is even.
   * @param num the number to be checked
   * @return true if even, false otherwise.
   */
    boolean isEvenNumber(int num);


   /**
    * check whether a number passed as argument is non-negative.
    * @param num the number to be checked
    * @return true if non-negative, false otherwise.
    */
    boolean isNonNegativeNumber(int num);

}
```

The services code

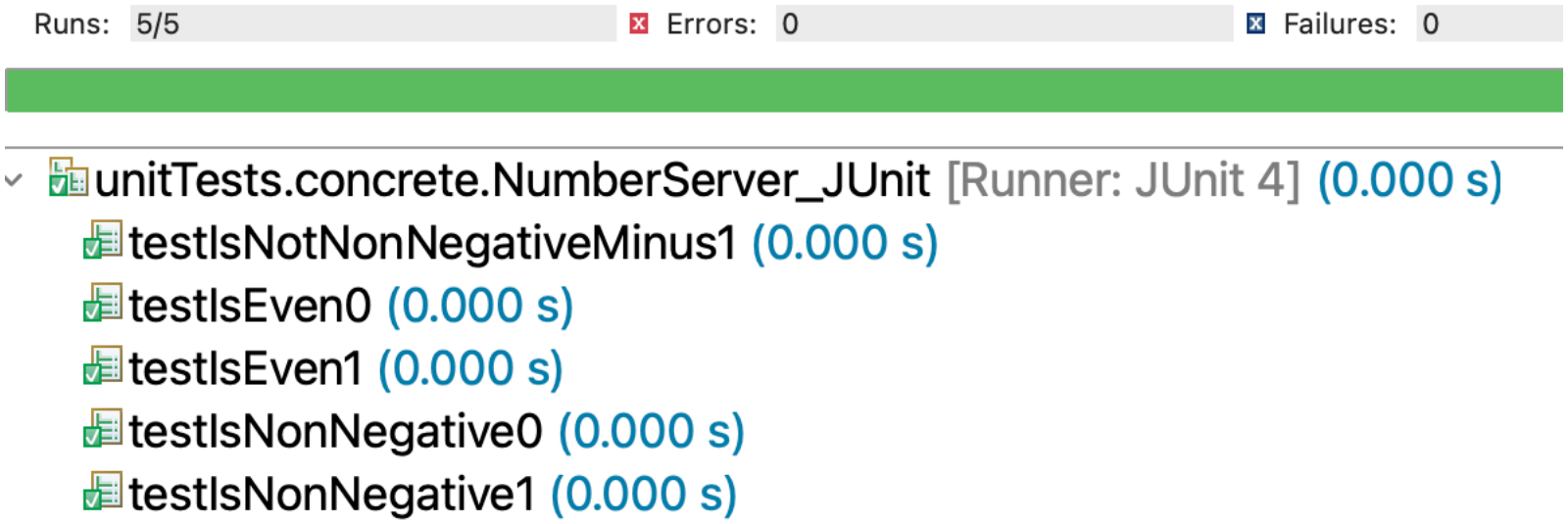
```
public class NumberServer implements NumberService{

  @Override
  public boolean isEvenNumber(int num) {

    return (num%2 == 0);
  }

  @Override
  public boolean isNonNegativeNumber(int num) {

    return (num >=0);
  }

}
```

The unit test code

```
public abstract class NumberService_JUnit {


  protected NumberService numberServiceUnderTest;

  /**
   * Initialise the numberServiceUnderTest object to a concrete instance
   */
  @Before
  public abstract void setUp();

  /**
   * Test1: 0 is an even number
   */
  @Test
  public void testIsEven0() {
    Assert.assertTrue(numberServiceUnderTest.isEvenNumber(0));
  }

…}
```

The unit test for the concrete implementation

```
public class NumberServer_JUnit extends NumberService_JUnit{

  @Override
  public void setUp() {
    numberServiceUnderTest = new NumberServer();

  }

}
```



Unit tests pass

Validation test code

```
public class NumberServiceClient {

  public static void main(String[] args)
  {

  NumberService numberService = new NumberServer();

  System.out.println("numberService.isEvenNumber(0) =
"+numberService.isEvenNumber(0));
  System.out.println("numberService.isEvenNumber(1) =
"+numberService.isEvenNumber(1));

  System.out.println("numberService.isNonNegativeNumber(0) =
"+numberService.isNonNegativeNumber(0));
  System.out.println("numberService.isNonNegativeNumber(1) =
"+numberService.isNonNegativeNumber(1));
  System.out.println("numberService.isNonNegativeNumber(-1) =
"+numberService.isNonNegativeNumber(-1));
  }
}
```

Validation tests in the terminal/console:

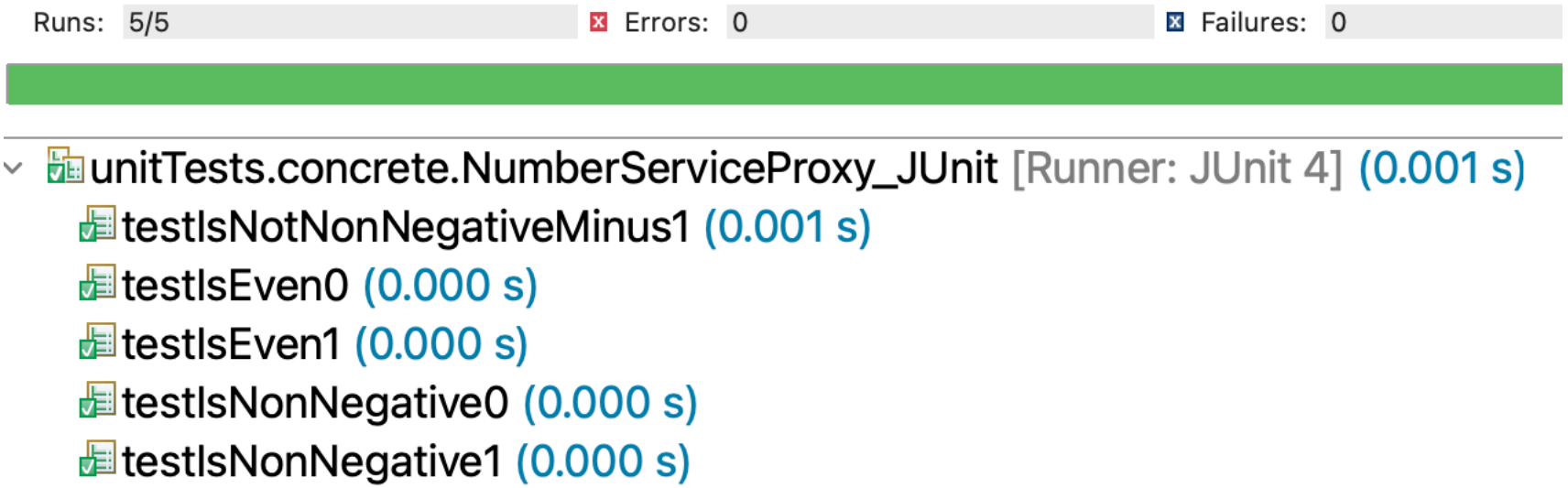
```
numberService.isEvenNumber(0) = true
numberService.isEvenNumber(1) = false
numberService.isNonNegativeNumber(0) = true
numberService.isNonNegativeNumber(1) = true
numberService.isNonNegativeNumber(-1) = false
```

The proxy redirects the call to the concrete service

```
public class NumberServiceProxy implements NumberService{

  NumberService numberService;

  public NumberServiceProxy (NumberService numberService) {
    this.numberService=numberService;
  }

  @Override
  public boolean isEvenNumber(int num) {

    return numberService.isEvenNumber(num);
  }

  @Override
  public boolean isNonNegativeNumber(int num) {

    return numberService.isNonNegativeNumber(num);
  }

}
```

Test the proxy

```
public class NumberServiceProxy_JUnit extends
NumberService_JUnit{

  @Override
  public void setUp() {
    numberServiceUnderTest = new NumberServiceProxy(new
NumberServer());

  }
```



The count requirements for the proxy interface

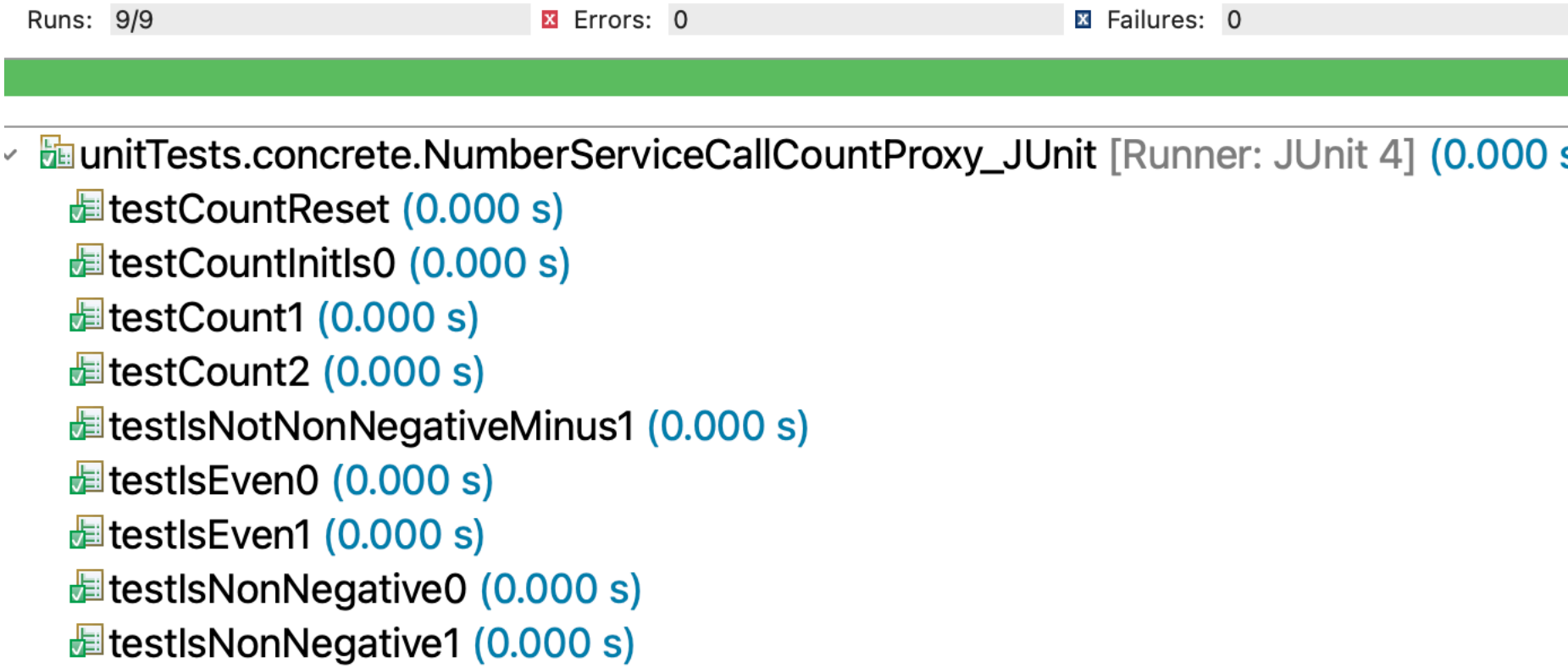
```
package interfaces;

public interface NumberServiceCallCount extends NumberService{

  public void resetCallCount();

  public int callsSinceLastReset();


}
```

Code to count

```
public class NumberServiceCallCountProxy   extends NumberServiceProxy implements
NumberServiceCallCount{

private int callCount;

  public NumberServiceCallCountProxy(NumberService numberService) {
    super(numberService);
    callCount=0;}

  @Override
  public boolean isEvenNumber(int num) {
    callCount++;
    return super.isEvenNumber(num);}

  @Override
  public boolean isNonNegativeNumber(int num) {
    callCount++;
    return super.isNonNegativeNumber(num);}

  @Override
  public void resetCallCount() {
    callCount=0;}

  @Override
  public int callsSinceLastReset() {
    return callCount;}
}
```

Add new count tests

```
public class NumberServiceCallCountProxy_JUnit extends NumberServer_JUnit{

  @Override
  public void setUp() {
    numberServiceUnderTest = new NumberServiceCallCountProxy(new NumberServer());

  }

  /**
   * Test Initial Count is 0
   */
  @Test
  public void testCountInitIs0() {
    Assert.assertEquals(0,
((NumberServiceCallCountProxy)numberServiceUnderTest).callsSinceLastReset());
  }

  /**
   * Test count after 1 call
   */
  @Test
  public void testCount1() {
  numberServiceUnderTest.isEvenNumber(1);
    Assert.assertEquals(1,
((NumberServiceCallCountProxy)numberServiceUnderTest).callsSinceLastReset());
  }
…}
```



Passes all unit tests for proxy behaviour and call count

Proxy Problem II - Password Proxy - TODO

Write a proxy for the service that will ask for a password before the service is executed

(just ‘hide’ the necessary password as a constant in the code… not secure, but easy to implement)

the password must be sent by a sendPassword(String) method call from the client before every service call; i.e there is no session management.

Also write suitable tests

Problem III - Double proxy - TODO

Implement a double proxy:

Passwords and counting - what are the requirements/design decisions

Also write suitable tests