Contents

[1. Sequence activities followed in TDD 2](#_Toc11063199)

[1.1. Add a test 2](#_Toc11063200)

[1.2. Run all tests and see if the new one fails 2](#_Toc11063201)

[1.3. Write some code 2](#_Toc11063202)

[1.4. Run tests 2](#_Toc11063203)

[1.5. Refactor code 2](#_Toc11063204)

[2. Requirement 2](#_Toc11063205)

[2.1. method can take array of url and return their respective level 2](#_Toc11063206)

[2.2 method can take null value and throw Custom validation Exception 3](#_Toc11063207)

[2.3 method can take empty array of url and throw Custom validation Exception 3](#_Toc11063208)

[2.4 method can take array of url with garbage data and throw Custom validation Exception 4](#_Toc11063209)

[2.5 method can take array of url and handling custom Exception for Runtime Errors 4](#_Toc11063210)

[3. Security consideration 4](#_Toc11063211)

[Class Access 4](#_Toc11063212)

[Initialisation 4](#_Toc11063213)

[Finality 4](#_Toc11063214)

[Inner Classes 4](#_Toc11063215)

[Hard Coding 5](#_Toc11063216)

[Scope 5](#_Toc11063217)

[Cloneability 5](#_Toc11063218)

[Serialization/Deserialization 5](#_Toc11063219)

# Sequence activities followed in TDD

# Add a test

# Run all tests and see if the new one fails

# Write some code

# Run tests

# Refactor code

# Requirement

The problem to be solved is to compute the number of URLs in Ls at a given level. A level is counted by the number of slashes (i.e. b /b  characters) in the URL after stripping the common domain prefix. For example, [http://www.fancysite.com](http://www.fancysite.com/)  is at level 0, <http://www.fancysite.com/xyz/xyz.html>  is at level 2, and so on. You may assume that the URLs in Ls are unique (i.e. Ls has no duplicates)

### method can take array of url and return their respective level

Input : Method receives the array valid url

output : Array level each url respectively

Description : This is positive test case for validating level of each of item and size of the array

test case code :

@Test

**public** **void** testCountLevelOfUrls() **throws** URLException {

List<String> urls1 = Arrays.*asList*("http://www.fancysite.com", "http://www.fancysite.com/xyz/xyz.html",

"http://www.fancysite.com/abc/xyz/pqr.html");

*assertEquals*(3, TestMain.*countLevelOfUrls*(urls1).length);

}

### 2.2 method can take null value and throw Custom validation Exception

Input : Method receives the null array

output : handling the custom Exception

Description : This is negative test case for validating response for null input

test case code :

@Test

**public** **final** **void** whenNullThrowsThenRuntimeExceptionIsThrown() {

**try** {

TestMain.*countLevelOfUrls*(**null**);

~~Assert~~.~~fail~~();

} **catch** (URLException e) {

~~Assert~~.~~assertEquals~~("Null Value passed", e.getMessage());

}

}

### 2.3 method can take empty array of url and throw Custom validation Exception

Input : Method receives the empty array

output : handling the custom Exception

Description : This is negative test case for validating response for empty input

test case code :

@Test(expected = URLException.**class**)

**public** **final** **void** whenEmptyStringIsUsedThenReturnValueIs0() **throws** URLException {

List<String> urls2 = Arrays.*asList*(" ");

*assertEquals*(0, TestMain.*countLevelOfUrls*(urls2));

}

@Test

**public** **final** **void** verifyEmptyArray() {

List<String> urls1 = Arrays.*asList*("-1");

**try** {

TestMain.*countLevelOfUrls*(**new** ArrayList<String>());

} **catch** (URLException e) {

~~Assert~~.~~assertEquals~~("No Url passed", e.getMessage());

}

}

### 2.4 method can take array of url with garbage data and throw Custom validation Exception

Input : Method receives the array of URL with garbage data

output : handling the custom Exception

Description : This is negative test case for validating response for garbage data input

test case code :

@Test

**public** **final** **void** checkGarbageValidation() {

List<String> urls1 = Arrays.*asList*("-1");

**try** {

TestMain.*countLevelOfUrls*(urls1);

} **catch** (URLException e) {

~~Assert~~.~~assertEquals~~("Invalid URL", e.getMessage());

}

}

### 2.5 method can take array of url and handling custom Exception for Runtime Errors

Input : Method receives the array of URL

output : handling the custom Exception for any runtime errors

Description : This is test case for runtime exception handling for runtime errors

test case code :

@Test

**public** **final** **void** checkRunTimeError() {

List<String> urls1 = Arrays.*asList*("http:// dasdas www.fancysite.com");

**try** {

TestMain.*countLevelOfUrls*(urls1);

} **catch** (URLException e) {

~~Assert~~.~~assertEquals~~("Run time Error", e.getMessage());

}

}

# Security consideration

## Class Access

Verified access levels of methods, class and fields. Made public what is required

### Initialisation

Marked variables private and used setter , getter where ever necessary

Make static variables private, this prevents access to uninitialized variables.

### Finality

Non-Final classes let an attacker extend a class in a malicious manne

### Inner Classes

Use inner class instead separate for security aspect, use private access modifier.

### Hard Coding

No hard coding used

### Scope

Package scope is really used so there are no naming conflicts for an application, especially when reusing classes from another framework.

### Cloneability

Restrict the clonability unless it is required

### Serialization/Deserialization

If processing restricted one JVM it is suggested to restrict the seriliazation