BLG 411E – Software Engineering

Recitation Session 3

Test Driven Development

Bilge S. AKKOCA GAZİOĞLU, Müge EREL ÖZÇEVİK, Beyza EKEN

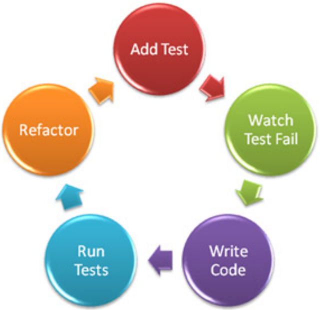
05.12.2017

Outline

1. Test Driven Development
   - TDD
2. Unit Testing
   - JUnit
3. Mock Objects
   - Mockito
4. JMeter
5. Test Automation
   - Selenium
6. References

TDD
Definition

- A software development process.
- Based on text-first concept in Extreme Programming.
- Relies on the repetition of a very short development cycle.
- Also called Red-Green-Refactor



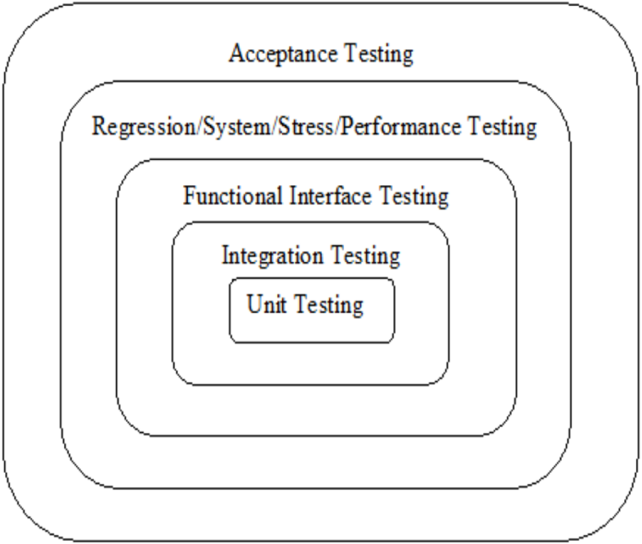
TDD
Life Cycle

1. Write a test
   - Before implementation
2. Run the test
   - Should fail
3. Write the implementation code
   - Just enough to pass the test
   - Not perfect
4. Run all tests
   - All should pass
5. Refactor
   - Optimize code without introducing new features
   - Should not make any tests fail
   - Should not include new tests
6. Repeat

TDD
Features

- All about speed: short unit tests (JUnit), imperfect but fast code.
- Forces us to think before coding.
- Allows us to develop fast without being afraid to break something.
- Allows us to refactor with confidence.
- External dependencies should be mocked (mockito).
- So that we can test without other parts implemented.
- So out tests run faster (For instance, a DB connection).
- This forces us to separate concerns (DB instance can be replaced easily).

Testing



Acceptance Testing

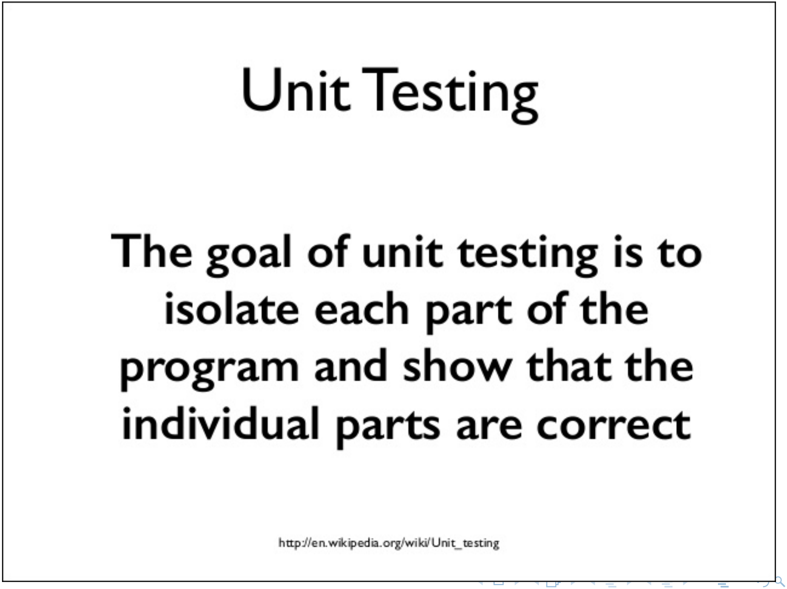
Regression/System/Stress/Performance Testing

Functional Interface Testing

Integration Testing

Unit Testing

Unit Testing



Unit Testing

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct

http://en.wikipedia.org/wiki/Unit_testing

Unit Testing

Definition

- A **white-box** testing method for individual units of code.
- A **unit** is the smallest testable part of an application.
  - Function in procedural programming
  - Interface, class or method in object-oriented programming.
- Should be easy to write and fast to execute.
- Each unit test should be independent of others.
- Helps to identify problems early and to refactor easily.

JUnit

- JUnit is a simple, powerful, open source framework to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
- Junit Features include
  - Assertions for testing expected results
  - Test fixtures for sharing common test data
  - Test runners for running tests

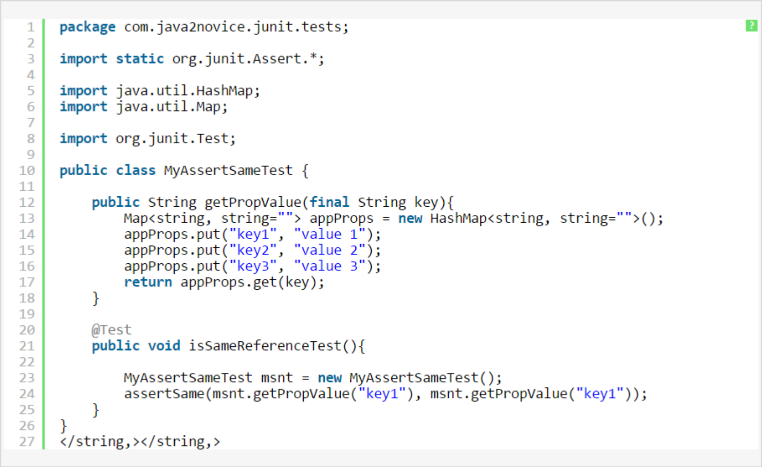
JUnit

- JUnit is a test framework which uses **annotations** to identify methods that specify a test.
  - @Test, @Test (expected = Exception.class), @Test(timeout=100), @Before, @After, @BeforeClass, @AfterClass, @Ignore
- JUnit provides static **methods** to test for certain conditions.
  - Assert class provides a set of assertion methods useful for writing tests.
  - fail, assertTrue, assertFalse, assertEquals, assertNull, assertNotNull, assertSame, assertNotSame

JUnit

Assertion method Assert.assertSame() example

Assert.assertSame() methods checks that two objects refer to the same object. If they are not the same, then an AssertionError will be thrown.

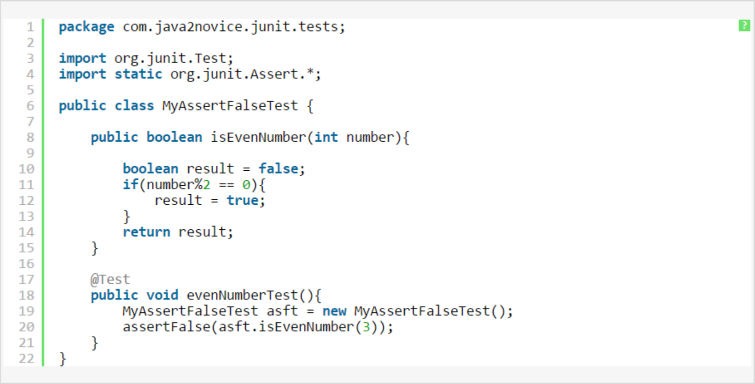


```
package com.java2novice.junit.tests;

import static org.junit.Assert.*;

import java.util.HashMap;
import java.util.Map;

import org.junit.Test;

public class MyAssertSameTest {

    public String getPropValue(final String key){
        Map<string, string=""> appProps = new HashMap<string, string="">();
        appProps.put("key1", "value 1");
        appProps.put("key2", "value 2");
        appProps.put("key3", "value 3");
        return appProps.get(key);
    }

    @Test
    public void isSameReferenceTest(){

        MyAssertSameTest msnt = new MyAssertSameTest();
        assertSame(msnt.getPropValue("key1"), msnt.getPropValue("key1"));
    }
}
</string,></string,>
```

JUnit

Assertion method Assert.assertFalse() example

Assert.assertFalse() methods checks whether the expected value is false or not.



```java
package com.java2novice.junit.tests;

import org.junit.Test;
import static org.junit.Assert.*;

public class MyAssertFalseTest {

    public boolean isEvenNumber(int number){

        boolean result = false;
        if(number%2 == 0){
            result = true;
        }
        return result;
    }

    @Test
    public void evenNumberTest(){
        MyAssertFalseTest asft = new MyAssertFalseTest();
        assertFalse(asft.isEvenNumber(3));
    }
}
```

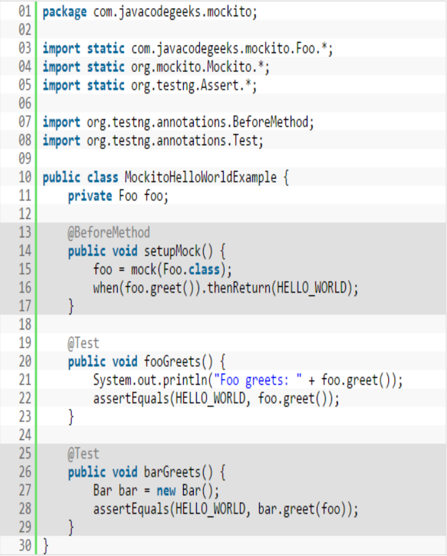
Mockito

- Mockito is a popular mocking framework which can be used in conjunction with JUnit.
- Allows us to create and configure mock objects.
- Using Mockito simplifies the development of tests for classes with external dependencies significantly.
- We can create the mock objects manually or can use the mocking framewors like Mockito, EasyMock. jMock etc.
- Mock frameworks allow us to create mock objects at runtime and define their behavior.
- The classical example for a mock object is a data provider.

Mockito

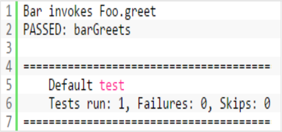
- Behavior can be defined using **when**, **thenReturn**, **thenThrow**, **thenAnswer** methods.
  - myClass myMock = mock(myClass.class);
  - when(myMock.myMethod(5))
    .thenReturn("five");
- Behavior can be verified using **verify** method.
  - verify(myMock).myMethod(12);
  - verify(myMock).myMethod(anyInt());
  - verify(myMock)
    .myMethod(argThat(isValid()));
  - verify(myMock, times(2)).myMethod(any());
  - verify(myMock, atMost(3)).myMethod(5);
  - verifyNoMoreInteractions(myMock);
  - inOrder.verify(myMock, atLeast(2))
    .myMethod(5);
    inOrder.verify(myMock, times(1))
    .myMethod(4);

Mockito Examples



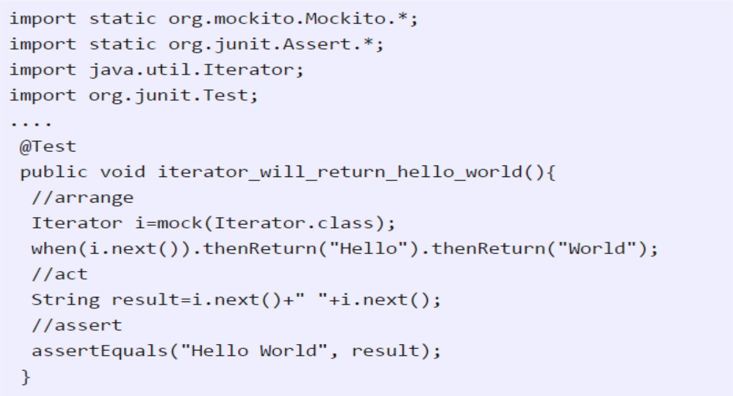
```java
package com.javacodegeeks.mockito;

import static com.javacodegeeks.mockito.Foo.*;
import static org.mockito.Mockito.*;
import static org.testng.Assert.*;

import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class MockitoHelloWorldExample {
    private Foo foo;

    @BeforeMethod
    public void setupMock() {
        foo = mock(Foo.class);
        when(foo.greet()).thenReturn(HELLO_WORLD);
    }

    @Test
    public void fooGreets() {
        System.out.println("Foo greets: " + foo.greet());
        assertEquals(HELLO_WORLD, foo.greet());
    }

    @Test
    public void barGreets() {
        Bar bar = new Bar();
        assertEquals(HELLO_WORLD, bar.greet(foo));
    }
}
```

Output:

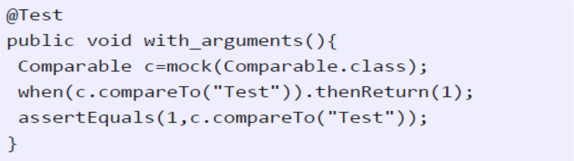


```
Bar invokes Foo.greet
PASSED: barGreets

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================
```

Mockito Examples

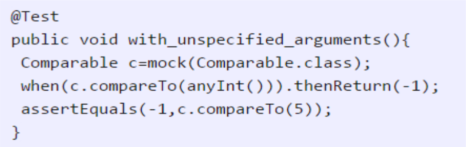


```
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;
import java.util.Iterator;
import org.junit.Test;
....
 @Test
 public void iterator_will_return_hello_world(){
  //arrange
  Iterator i=mock(Iterator.class);
  when(i.next()).thenReturn("Hello").thenReturn("World");
  //act
  String result=i.next()+" "+i.next();
  //assert
  assertEquals("Hello World", result);
 }
```

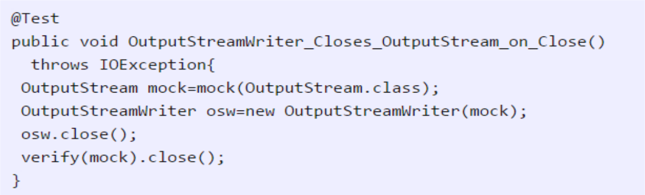


```
@Test
public void with_arguments(){
 Comparable c=mock(Comparable.class);
 when(c.compareTo("Test")).thenReturn(1);
 assertEquals(1,c.compareTo("Test"));
}
```

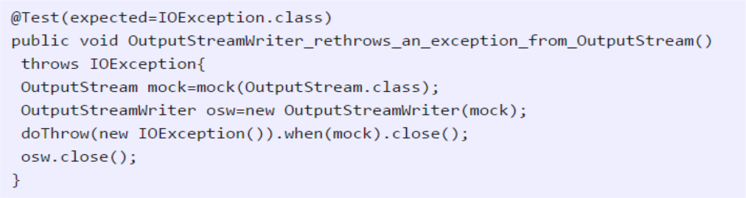
Mockito Examples



```
@Test
public void with_unspecified_arguments(){
 Comparable c=mock(Comparable.class);
 when(c.compareTo(anyInt())).thenReturn(-1);
 assertEquals(-1,c.compareTo(5));
}
```

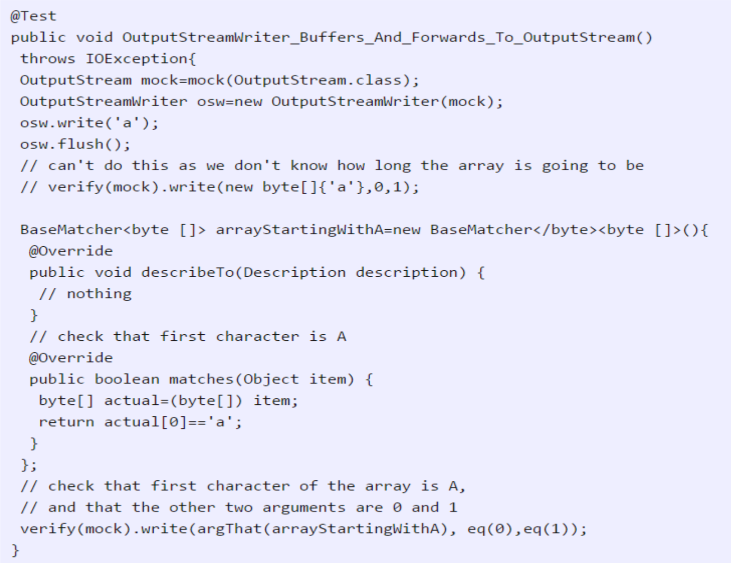


```
@Test
public void OutputStreamWriter_Closes_OutputStream_on_Close()
  throws IOException{
 OutputStream mock=mock(OutputStream.class);
 OutputStreamWriter osw=new OutputStreamWriter(mock);
 osw.close();
 verify(mock).close();
}
```



```
@Test(expected=IOException.class)
public void OutputStreamWriter_rethrows_an_exception_from_OutputStream()
 throws IOException{
 OutputStream mock=mock(OutputStream.class);
 OutputStreamWriter osw=new OutputStreamWriter(mock);
 doThrow(new IOException()).when(mock).close();
 osw.close();
}
```

Mockito Examples

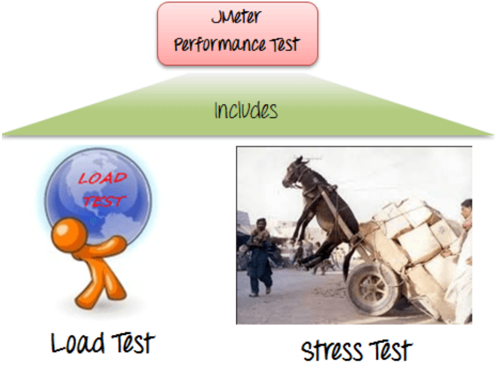


```
@Test
public void OutputStreamWriter_Buffers_And_Forwards_To_OutputStream()
 throws IOException{
 OutputStream mock=mock(OutputStream.class);
 OutputStreamWriter osw=new OutputStreamWriter(mock);
 osw.write('a');
 osw.flush();
 // can't do this as we don't know how long the array is going to be
 // verify(mock).write(new byte[]{'a'},0,1);

 BaseMatcher<byte []> arrayStartingWithA=new BaseMatcher</byte><byte []>(){
  @Override
  public void describeTo(Description description) {
   // nothing
  }
  // check that first character is A
  @Override
  public boolean matches(Object item) {
   byte[] actual=(byte[]) item;
   return actual[0]=='a';
  }
 };
 // check that first character of the array is A,
 // and that the other two arguments are 0 and 1
 verify(mock).write(argThat(arrayStartingWithA), eq(0),eq(1));
}
```

Apache JMeter

- Open Source testing software
- An application for load and performance testing
- Designed to cover categories of tests like load, functional, performance, regression, etc.
- To analyze and measure the performance of web application or variety of services. Performance testing means testing a web application against heavy load, multiple and concurrent user traffic.
- JMeter originally is used for testing Web Application or FTP application. Nowadays, it is used for functional test, database server test etc.

JMeter Performance Test



JMeter Performance Test

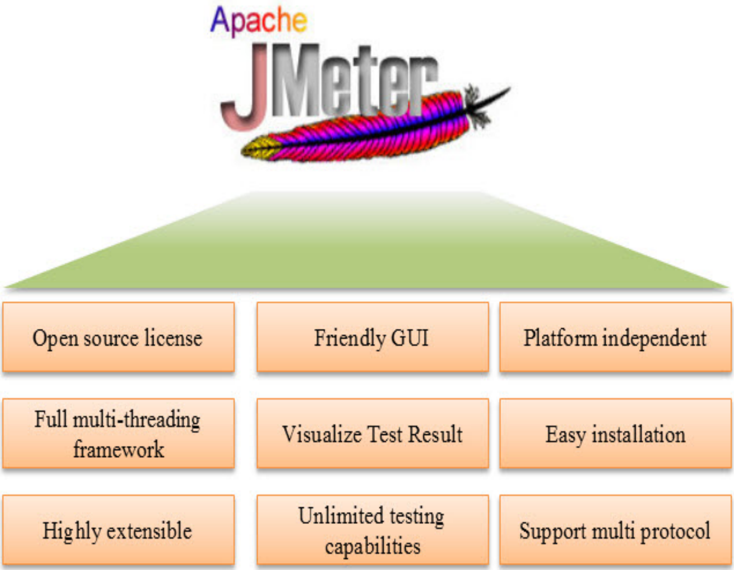
Includes

Load Test

Stress Test

Load Testing: Modelling the expected usage by simulating multiple user access the web services concurrently.
Stress Testing: Finding the maximum load the web server can handle.

JMeter advantages



Open source license

Friendly GUI

Platform independent

Full multi-threading framework

Visualize Test Result

Easy installation

Highly extensible

Unlimited testing capabilities

Support multi protocol

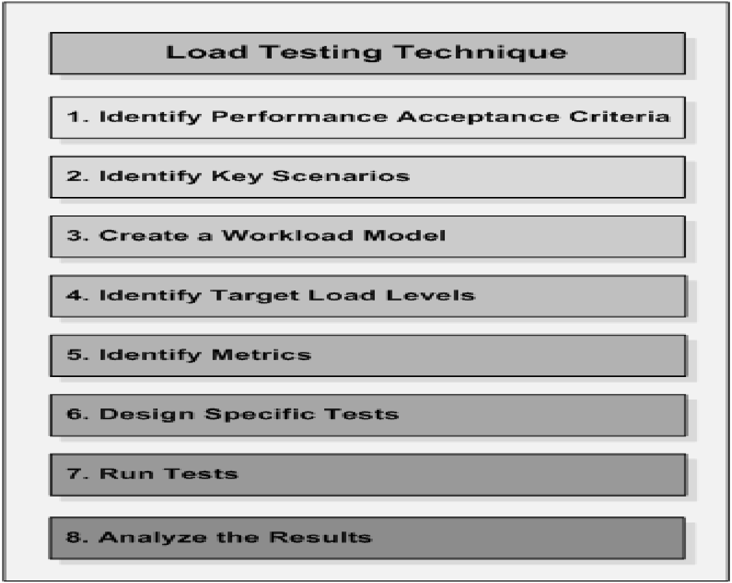
JMeter advantages

JMeter offers following benefit in Performance testing

- JMeter can be used to test performance of both static resources such as JavaScript and HTML, as well as dynamic resources, such as JSP, Servlets, and AJAX.
- JMeter can discover maximum number of concurrent users that your website can handle
- JMeter provides a variety of graphical analyses of performance reports.

JMeter

Load Testing Techniques

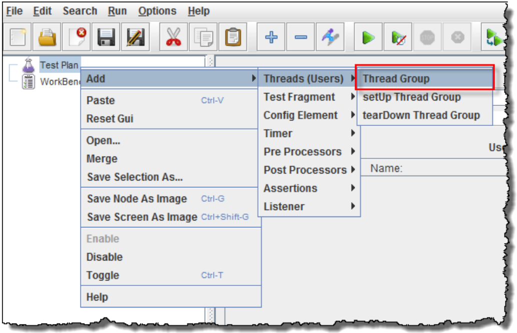


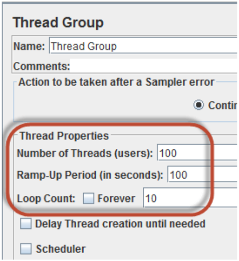
Load Testing Technique

1. Identify Performance Acceptance Criteria
2. Identify Key Scenarios
3. Create a Workload Model
4. Identify Target Load Levels
5. Identify Metrics
6. Design Specific Tests
7. Run Tests
8. Analyze the Results

JMeter

Adding Thread Group





Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

● Contir

Thread Properties

Number of Threads (users): 100

Ramp-Up Period (in seconds): 100

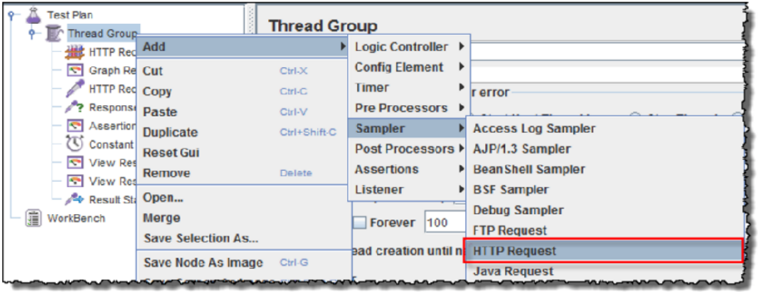
Loop Count: □ Forever 10

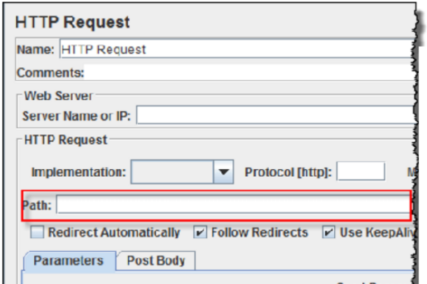
□ Delay Thread creation until needed

□ Scheduler

JMeter

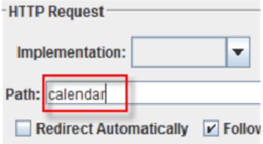
HTTP Request




HTTP Request

Name: HTTP Request

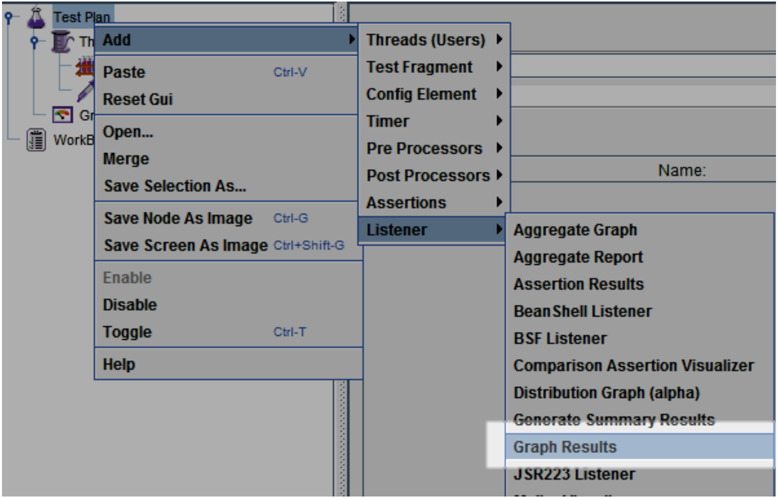
Comments:



Path: calendar

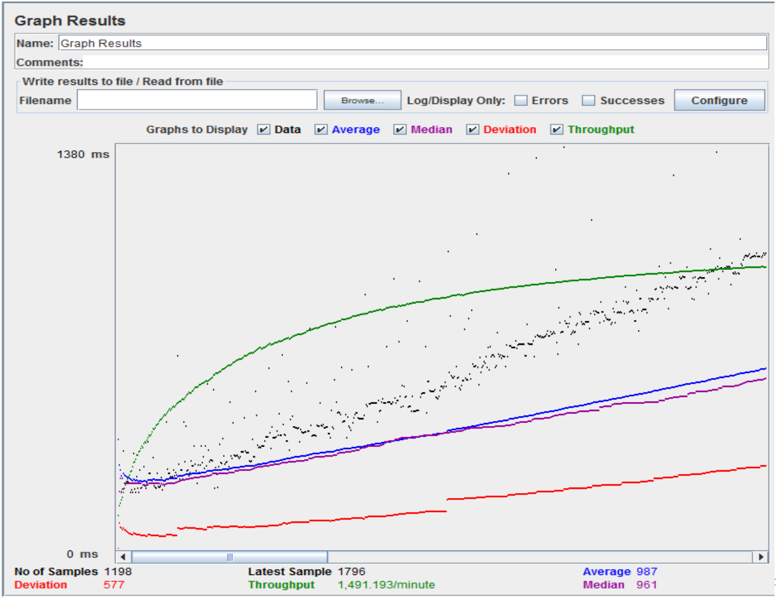
JMeter

Adding Graph Result



JMeter

The result of test



Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

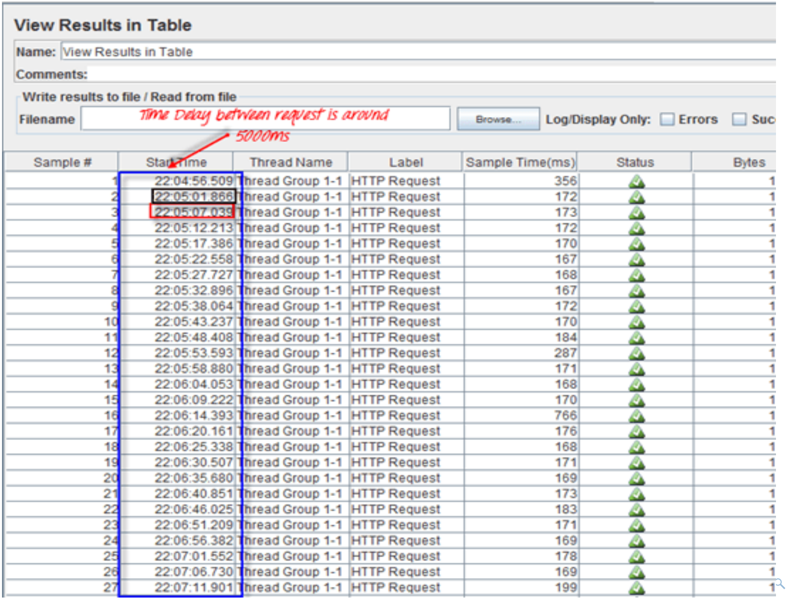
Graphs to Display ☑ Data ☑ Average ☑ Median ☑ Deviation ☑ Throughput

No of Samples 1198
Deviation 577

Latest Sample 1796
Throughput 1,491.193/minute

Average 987
Median 961

JMeter
The result of test



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

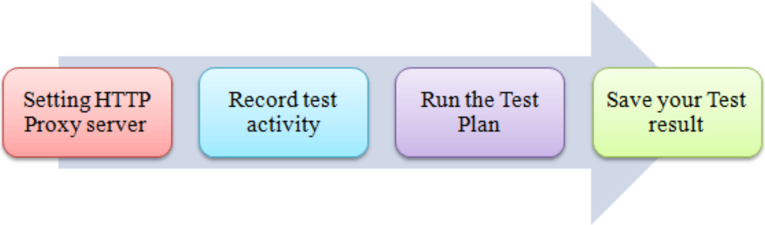
Filename

Time Delay between request is around 5000ms

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	22:04:56.509	Thread Group 1-1	HTTP Request	356	✓	1
2	22:05:01.866	Thread Group 1-1	HTTP Request	172	✓	1
3	22:05:07.039	Thread Group 1-1	HTTP Request	173	✓	1
4	22:05:12.213	Thread Group 1-1	HTTP Request	172	✓	1
5	22:05:17.386	Thread Group 1-1	HTTP Request	170	✓	1
6	22:05:22.558	Thread Group 1-1	HTTP Request	167	✓	1
7	22:05:27.727	Thread Group 1-1	HTTP Request	168	✓	1
8	22:05:32.896	Thread Group 1-1	HTTP Request	167	✓	1
9	22:05:38.064	Thread Group 1-1	HTTP Request	172	✓	1
10	22:05:43.237	Thread Group 1-1	HTTP Request	170	✓	1
11	22:05:48.408	Thread Group 1-1	HTTP Request	184	✓	1
12	22:05:53.593	Thread Group 1-1	HTTP Request	287	✓	1
13	22:05:58.880	Thread Group 1-1	HTTP Request	171	✓	1
14	22:06:04.053	Thread Group 1-1	HTTP Request	168	✓	1
15	22:06:09.222	Thread Group 1-1	HTTP Request	170	✓	1
16	22:06:14.393	Thread Group 1-1	HTTP Request	766	✓	1
17	22:06:20.161	Thread Group 1-1	HTTP Request	176	✓	1
18	22:06:25.338	Thread Group 1-1	HTTP Request	168	✓	1
19	22:06:30.507	Thread Group 1-1	HTTP Request	171	✓	1
20	22:06:35.680	Thread Group 1-1	HTTP Request	169	✓	1
21	22:06:40.851	Thread Group 1-1	HTTP Request	173	✓	1
22	22:06:46.025	Thread Group 1-1	HTTP Request	183	✓	1
23	22:06:51.209	Thread Group 1-1	HTTP Request	171	✓	1
24	22:06:56.382	Thread Group 1-1	HTTP Request	169	✓	1
25	22:07:01.552	Thread Group 1-1	HTTP Request	178	✓	1
26	22:07:06.730	Thread Group 1-1	HTTP Request	169	✓	1
27	22:07:11.901	Thread Group 1-1	HTTP Request	199	✓	1

JMeter

HTTP Proxy Server Testing



http://www.guru99.com/how-to-use-jmeter-for-http-proxy-server-testin
html

Test Automation

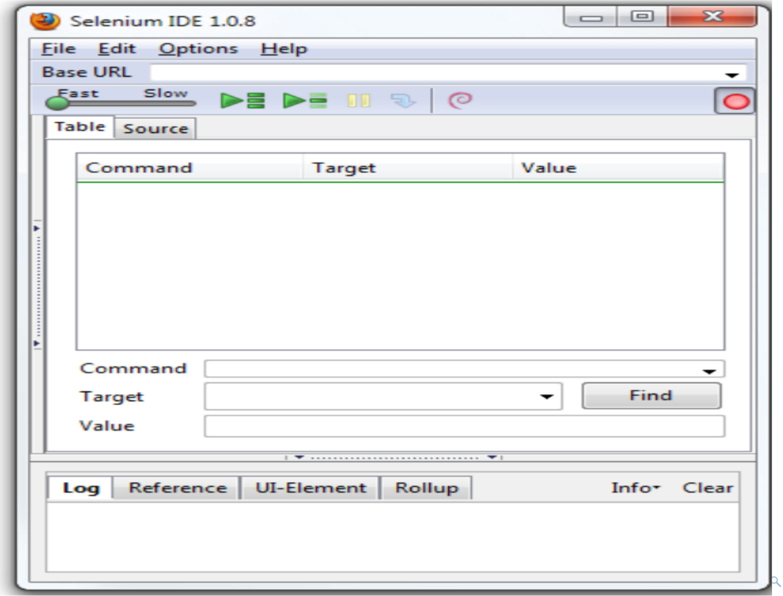
Test automation has specific advantages for improving the long-term efficiency of a software team testing processes. Test automation supports:

- Frequent regression testing
- Rapid feedback to developers
- Virtually unlimited iterations of test case execution
- Support for Agile and extreme development methodologies
- Disciplined documentation of test cases
- Customized defect reporting
- Finding defects missed by manual testing

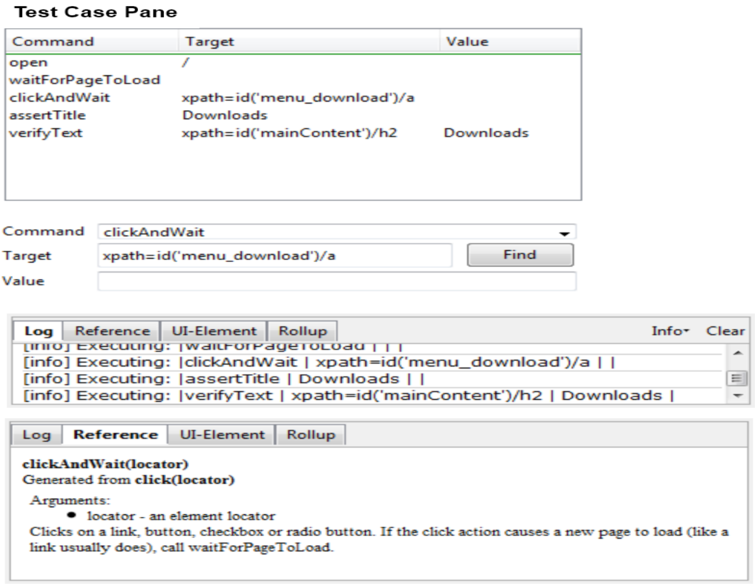
Selenium Test Automation

- Selenium is a set of different software tools each with a different approach to supporting test automation.
- Key feature of Selenium is the support for executing tests on multiple browser platforms.
- Selenium is composed of multiple software tools. Each has a specific role.
  - Web Driver
  - Selenium RC
  - Selenium IDE
  - Selenium-Grid

Selenium IDE



Selenium IDE



Test Case Pane

Command	Target	Value
open	/	
waitForPageToLoad		
clickAndWait	xpath=id('menu_download')/a	
assertTitle	Downloads	
verifyText	xpath=id('mainContent')/h2	Downloads

Command: clickAndWait

Target: xpath=id('menu_download')/a

Value:

```
[info] Executing: |waitForPageToLoad | | |
[info] Executing: |clickAndWait | xpath=id('menu_download')/a | |
[info] Executing: |assertTitle | Downloads | |
[info] Executing: |verifyText | xpath=id('mainContent')/h2 | Downloads |
```

clickAndWait(locator)
Generated from **click(locator)**

Arguments:

- locator - an element locator

Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.

References and Further Reading

- http://technologyconversations.com/2014/09/30/test-driven-development-tdd/
- http://technologyconversations.com/2013/12/20/test-driven-development-tdd-example-walkthrough/
- Meszaros, Gerard (2007). *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley. ISBN 978-0-13-149505-0.
- http://msdn.microsoft.com/en-us/magazine/cc163358.aspx
- http://junit.org/
- https://code.google.com/p/mockito/
- http://jmeter.apache.org/
- http://www.guru99.com/introduction-to-jmeter.html
- http://www.seleniumhq.org/