**W15**

Test Automation

5/7/2014 3:00:00 PM

# Implementing Testing for Behavior-Driven Development Using Cucumber

**Presented by:**

**Max Saperstone**

**Coveros**

**Brought to you by:**

SOFTWARE QUALITY ENGINEERING THE POWER OF INFORMATION

340 Corporate Way, Suite 300, Orange Park, FL 32073
888-268-8770 · 904-278-0524 · sqeinfo@sqe.com · www.sqe.com

# Max Saperstone

Coveros

For almost a decade, Max Saperstone has been a test engineer focusing on test automation and the continuous integration/continuous delivery process. Max specializes in open source tools-Selenium, JMeter, AutoIT, Cucumber, and Chef. He has led several testing automation efforts, including developing an automated suite focused on web-based software to operate over several applications. Max also headed a major project developing an automated testing structure to run Cucumber tests over multiple test interfaces and environments, while developing a system to keep test data “ageless.” He is currently developing a new testing architecture for SecureCI to allow testing of multiple interfaces, custom reporting, and minimal test upkeep.

# Implementing Effective Testing for Behavior Driven Development using Cucumber-JVM

*STAREAST 2014*

## Max Saperstone

Max Saperstone has been working as a Software and Test Engineer for almost a decade, with a focus on Test Automation and the CI/CD process. He specializes in open source tools, including the Selenium Tool Suite, JMeter, AutoIT, Cucumber, and Chef. Max has lead several testing automation efforts, including developing an automated suite focusing on web-based software to operate over several applications for Kronos Federal. He also headed a project with Delta Dental, developing an automated testing structure to run Cucumber tests over multiple test interfaces and environments, while also developing a system to keep test data 'ageless.' He recently released a new testing architecture for SecureCI™ to allow testing of multiple interfaces, custom reporting, and minimal test upkeep. He is currently engaged in CI/CD work, working to create full automated delivery using open source tools including Jenkins, SonarQube, and Nexus.

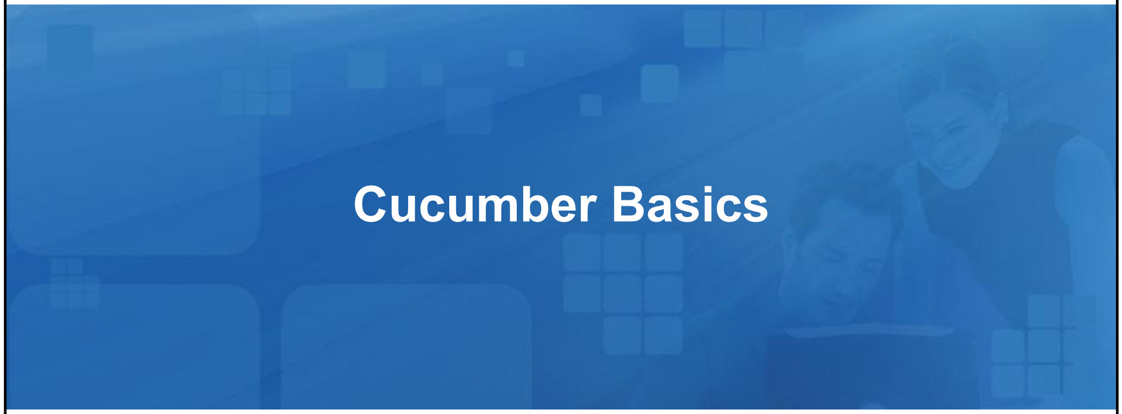
## About Coveros

- Coveros helps organizations accelerate the delivery of secure, reliable software
- Our consulting services:
  - Agile software development
  - Mobile application development
  - Application security
  - Software quality assurance
  - Software process improvement
- Our key markets:
  - Financial services
  - Healthcare
  - Defense
  - Critical Infrastructure

**Corporate Partners**

## Agenda

- Cucumber Overview
  - What is Cucumber
  - Gherkin
  - Glue Code
- Java Implementation
  - Cucumber Structure
  - Recommended Structure
  - Data Models
  - Functionality
- Execution
  - Ant
  - Results



# Cucumber Basics

## Introduction

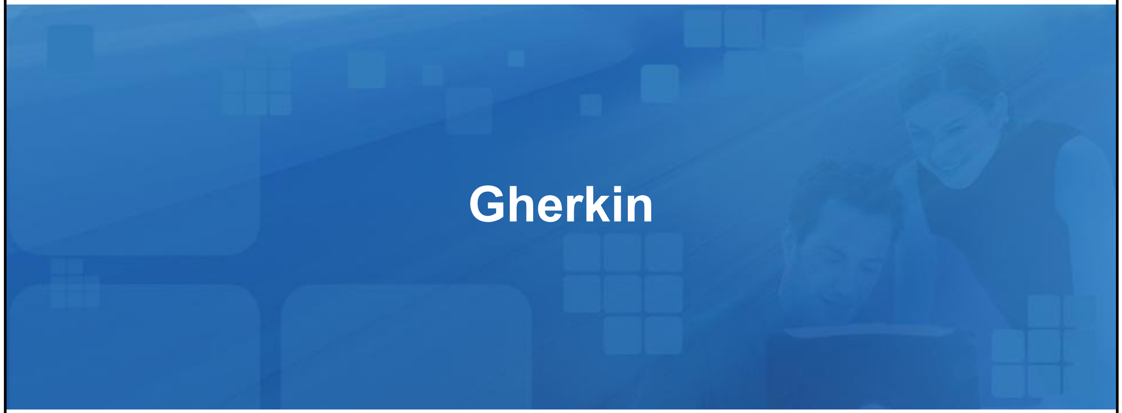
- Cucumber is a tool that supports Behavior Driven Development, BDD.
- Cucumber and BDD is not about testing GUIs. It is about systems behavior.
- To write tests, specify the properties you want a system to have.
- You don’t know, and should not care, about the implementation when you define your features.

## How to Write Tests

- The whole point of BDD is that it is vitally important to write each test/feature step
  - one at a time
  - with a domain expert
  - in plain language
- The use of plain language in the feature file is crucial to successful use of Cucumber
- State the result that you wish to obtain while avoiding specifying how you expect to get it

## Basic Setup

- The most basic Cucumber-jvm setup includes 3 files
  - Generic Test Runner
  - Feature File
  - Test Implementation
- The Test Runner is the actual file to execute within your IDE, and by default runs as a JUnit test
- The Feature Files are what contain all of the human readable tests
- The Test Implementation file will contain all of our implementations for our tests



## Gherkin

## Gherkin

- Gherkin is a business readable, domain specific language that lets you describe software's behaviour
- A ***Feature*** is a set of functionality - think Test Suite
- A single ***Feature*** is typically contained in its own file (ending in .feature)
- Features are typically composed of multiple ***Scenarios***
- A ***Scenario*** is a block of statements that describe some desired behavior
- A ***Scenario Outline*** is a block of statements that gets repeated over a set of data
- ***Scenarios*** specify *What* and should avoid answering the question *How*

## How to write Scenarios

- A scenario statement - step - consists of three parts:
- Given – the preconditions of the system under test. The setup of the systems state if you want. For our tests, we indicate (if desired) which interface we want to test.
- When – the actual change of the system. Transforming it from the initial state to the final state.
- Then – the expected final state of the system. The verification that the state change was the desired change.

## Example Feature File

```
Feature: Testing for login page

   Scenario: Login without password

        Given I want to use the browser Firefox
        When I provide username testuser1
        And I login
        Then I see the login error message "Please provide a password."
        And I am on the login page

   Scenario: Login without username

        Given I want to use the browser Firefox
        When I provide testuser1
        And I login
        Then I see the login error message "Please provide a username."
        And I am on the login page
```

## How to Write Tests (example)

- For example, for a login scenario you should write:

```
When I login as USER1 to CosmicComix
```

- And not:

```
When I visit www.cosiccomix.com
And I click on the login button
When I enter USER1 in the username field
And I click the continue button
And I click the login button
```

- You should concern yourself with what has to happen and not how you expect it to happen.

## Feature and Scenario File Tips

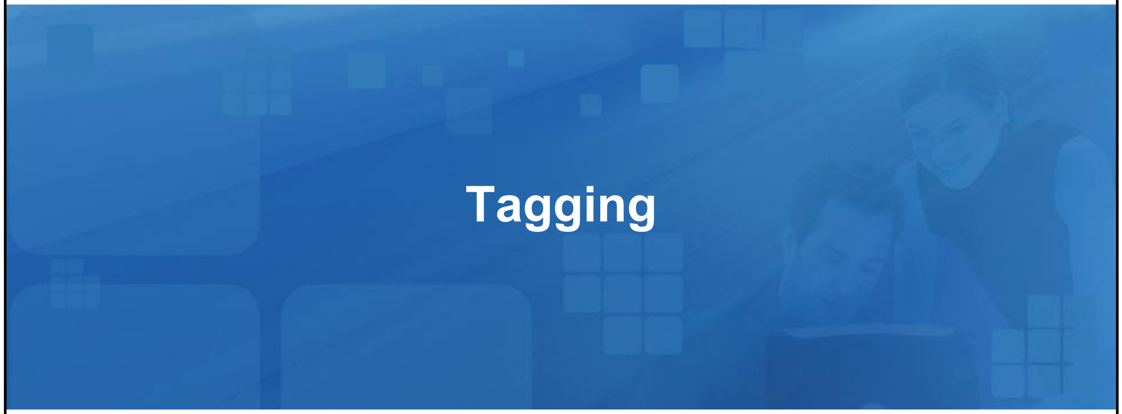
- Scenario Outlines can contain multiple variable types
  - Doc Strings
  - Data Tables
  - Data Table Diffing
  - List of Maps/Scalars/Lists
  - Data Table Transformation
  - String Transformations
- Often times when writing multiple scenarios you see repeated test steps
- Initial similar test steps can be moved out into *Background*
- These steps get executed before every scenario

## What was once...

```
Feature: Testing for login page

   Scenario: Login without password

        Given I want to use the browser Firefox
        When I provide username testuser1
        And I login
        Then I see the login error message "Please provide a password."
        And I am on the login page

   Scenario: Login without username

        Given I want to use the browser Firefox
        When I provide testuser1
        And I login
        Then I see the login error message "Please provide a username."
        And I am on the login page
```

## Can become...

```
Feature: Testing for login page
   Background
        Given I want to use the browser Firefox
   Scenario: Login without password

        When I set the username to testuser1
        And I login to CosmicComix
        Then I see the login error message "Please provide a password."
        And I am on the login page

   Scenario: Login without username

        When I set the password to testuser1
        And I login to CosmicComix
        Then I see the login error message "Please provide a username."
        And I am on the login page
```



## Tagging

## Tagging Basics

- Cucumber provides a simple method to organize features and scenarios by user determined classifications.
- This is implemented using the convention that any space delimited string found in a feature file that is prefaced with the commercial at (@) symbol is considered a tag.
- Any string may be used as a tag and any scenario or entire feature can have multiple tags associated with it.
- Be aware that tags are heritable within Feature files.
  - Scenarios inherit tags from the Feature statement.
  - Examples inherit tags from the Feature and Scenario statements.

## Tagging Examples

```
@CCOrg @CCNet
Feature: Testing for login page

   Scenario Outline: Bad login
        Given I want to use the browser Firefox
        When I set the username to [username]
        When I set the password to [password]
        When I login to CosmicComix
        Then I see the error message "[message]"
        And I am on the login page

        @Regression
        Examples:
        |    username        |    password        |        message        |
        |    testuser1       |                    |    Please provide a password.    |
        |                    |    testuser1       |    Please provide a username.    |
        |    testuser        |    testuser        |    That username does not match anything in
        |    testuser1       |    testuser2       |    The password provided does not match the
```

## Step Definitions/Glue Code

## Cucumber Implementation

- Cucumber searches the classpath provided in the runner to find any methods annotated with regular expressions that will match each Given/When/Then part of the feature
- There must only be one method, step, which matches the regular expression in the classpath
- If you have described two different parts of the system with the exact same wording, then you have an issue with ambiguity

## Step Definition Best Practices

- The matcher is not overly verbose.
- The matcher handles both positive and negative (true and false) conditions.
- The matcher has at most two value parameters
- The parameter variables are clearly named
- The body is less than fifteen lines of code
- The body does not call other steps

## Step Definition Basics

- When you put part of a regular expression in parentheses, whatever it matches gets captured for use later.
  - This is known as a "capture group."
- In Cucumber, captured strings become step definition parameters.
  - Typically, if you're using a wildcard, you probably want to capture the matching value for use in your step definition.

```
Given("^I'm logged in as an? (.*)$")
public void ImLoggedInAsA(String role) {
  // log in as the given role
}
```

- If your step is Given *I'm logged in as an admin*, then the step definition gets passed "admin" for role.

## Capturing and not capturing

- Cucumber converts captured strings to the step definition parameter type
- Sometimes, you have to use parentheses to get a regular expression to work, but you don't want to capture the match.
- For example, suppose I want to be able to match both *When I log in as an admin* and *Given I'm logged in as an admin*.
- Both step definitions do the same thing, there is no reason to have duplicated automation code

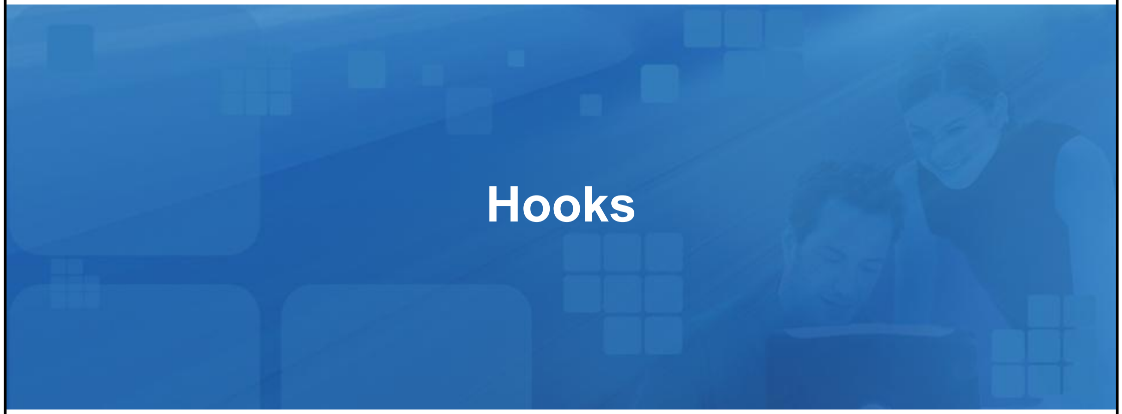
```
When("^(I'm logged|I log) in as an? (.*)$")
public void LogInAs(string role) {
  // log in as the given role
}
```

## Capturing and not capturing (cont.)

- My regular expression is capturing two strings, but my step definition method only takes one.
- I need to designate the first group as non-capturing like this:

```
When("^(?:I'm logged|I log) in as an? (.*)$")]
public void LogInAs(string role) {
  // log in as the given role
}
```

- Now, with the addition of ?: at the beginning of the group, it will perform as expected
- As mentioned previously, a multitude of object types can be provided, and if expected in *(.*)* will be automatically parsed



## Hooks

## Hooks Overview

- Hooks allow us to perform actions at various points in the cucumber test cycle
- Before hooks will be run before the first step of each scenario.
- After hooks will be run after the last step of each scenario, even when there are failing, undefined, pending or skipped steps.
- Scenario Hooks
  - Similar to JUnit *@Before* and *@After* run with each scenario
  - Placing common functionality in these reduces the number of test steps in each scenario

## Hooks Overview

- Tagged Hooks
  - We can also indicate that *@Before* and *@After* only run with scenarios with certain tags

    *e.x. @Before('@web')* for tests needing a browser launched
  - Tagged hooks can have multiple tags, and follow similar tagging AND/OR rules that the runner does

    *e.x. @Before('@CCOrg, @CCNet)* would run before scenarios tagged with *@CCOrg* OR *@CCNet*

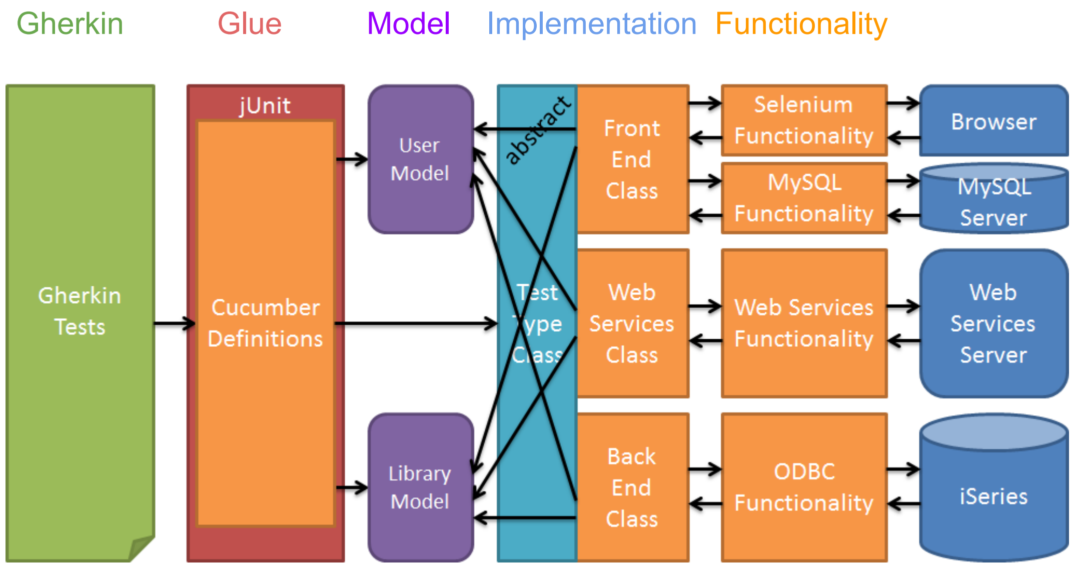
    *e.x.* @Before('@CCOrg', '~@CCNet') would run before scenarios tagged with *@CCOrg* AND NOT *@CCNet*
- Global Hooks
  - Cucumber doesn't support global hooks, however, hacks are possible

## Sample Code

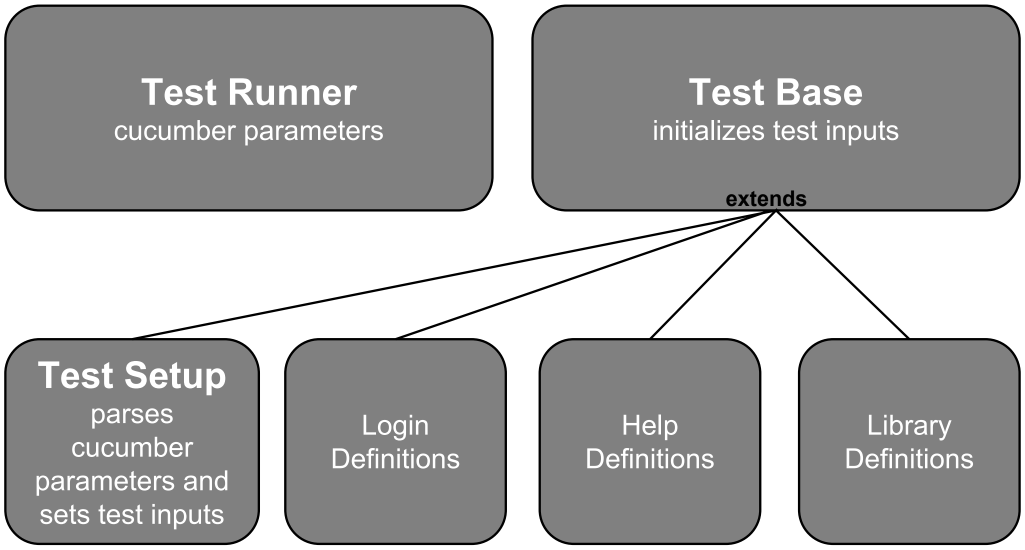
```
@Before   //performed before anything is done
public void setup() {
     if ( webEls == null ) {
          webEls = new LoadElements( xmlFiles );
          //set our database
          if( System.getProperty( "dataenvironment" ) != null ) {
               database = Databases.valueOf( System.getProperty(
                         "dataenvironment" ).substring(1) );
          }
     }
     tests.clear();
     tests.add( database.getValidTestingInterfaces() );
}

@Before('@ExternalWebsite')   //performed before each web scenario
public void externalWebsite() {
     tests.add( new ExternalWebsite( webEls, Browsers.Firefox ) );
}
```

## Java Structure

## Test Architecture



## Definition Structure



# Data Modeling

## Java Implementation

- Drive the initial implementation from the steps. As it looks now I will need
  - Potential Interface selector
  - A place to store user information
    - Username
    - Password
  - A method to submit the information
  - A method to check the page
  - A method to check the message
- Before I start implementing the model, I want to implement the steps that will verify the model.
- Start with an implementation of the steps like this...

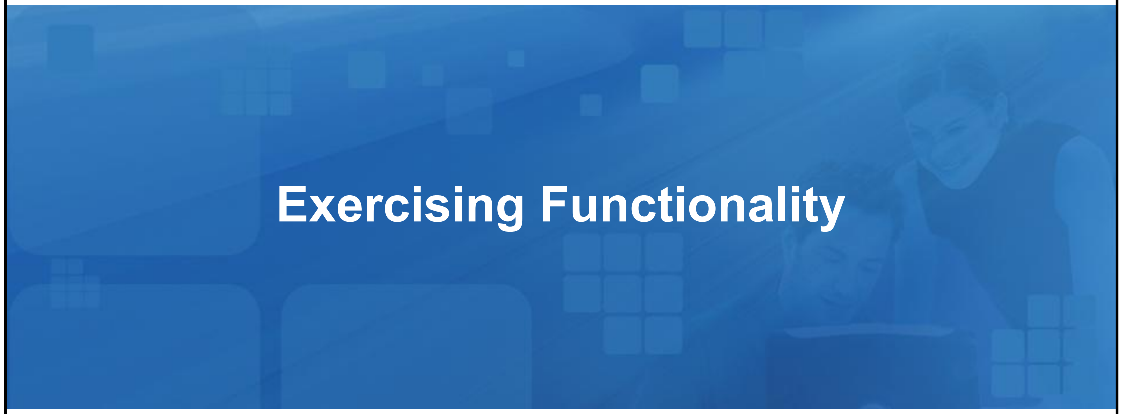
## Java Implementation - Code

```
    //type in our username
    @When("^I set the username to (.*)$")
    public void setUsername(String user) throws Exception {
         user.setUsername( user );
    }
    //type in our password
    @When("^I set the password to (.*)$")
    public void setPassword(String password) throws Exception {
         user.setPassword( password );
    }
    //click the login button
    @When("^I login to CosmicComix$")
    public void login() throws Exception {
         login.login( user );
    }

    //check our message
    @Then("^I see the login error message \"(.*)\"$")
    public void checkLoginErrorMessage(String errorMessage) throws Exception {
         login.checkLoginErrorMessage( errorMessage );
    }
  //check our page
    @Then("^I am on the (.*) page$")
    public void checkPage(Pages page) throws Exception {
         login.checkPage( page );
    }
```

## Data Modeling

- Once all of the test steps have been implemented, data models should be created to encapsulate all needed fields
- Based on our above examples, we'll need a *User* object which will contain:
  - Username
  - Password
  -

```
public class User {
     private String user;
     private String password;
     public void setUser( String user ) {              this.user = user;             }
     public void setPassword( String password ) { this.password = password;     }
     public String getUser() {                          return user;                  }
     public String getPassword() {                 return password;              }
}
```



## Exercising Functionality

## Functionality

- The functionality should be broken down into two distinct ‘layers’
  - Base calls (e.g. Selenium, ODBC, HTTP)
  - Implementation of workflow
- All of the static calls to external services are defined and can remain unmodified as workflow and application functionality changes
- These calls can also automatically log all data being passed to and from the services to provide a more seamless logging

## Implementation

- The workflow of the application (the how) is all contained in the abstract class forming the implementation
- One class should be written to contain all workflows (each as a method) expected from the application
  - Login
  - Help
  - Library
- Some methods can be defined at this level, however any that are interface specific should be left as abstract to be defined by the extending class

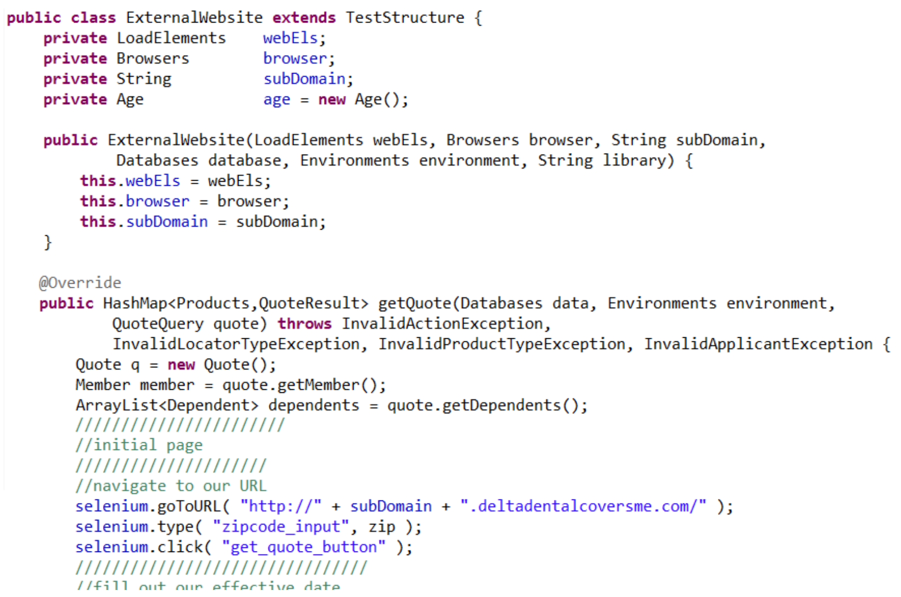
## Base Class

```
public abstract class TestStructure {
     protected WebDriver                  driver;        //this is our selenium webdriver contro
     protected SeleniumWebdriver          selenium;      //this is our selenium instance
     private ArrayList<Object>            call = new ArrayList<Object>();          //this is the 
     private ArrayList<Object>            response = new ArrayList<Object>();      //this is the 
     private ArrayList<Object>            actions = new ArrayList<Object>(); //this is all of the

     protected final String       appURL = "http://cosmiccomix.net"; //the url of our applica

     public abstract void setDriver() throws InvalidBrowserException;

     public abstract void unsetDriver();

     public SeleniumWebdriver getSelenium() {
          return selenium;
     }

     public WebDriver getDriver() {
          return driver;
     }

     public abstract void login(User user);

     public void checkPage(Pages page) {
          if ( selenium == null )
          throw new InvalidTestInterfaceException();
          selenium.getTitle();
     }
}
```

## External Web Class

```java
public class ExternalWebsite extends TestStructure {
      private LoadElements     webEls;
      private Browsers         browser;
      private String           subDomain;
      public ExternalWebsite(LoadElements webEls, Browsers browser, String subDomain) {
            this.webEls = webEls;
            this.browser = browser;
            this.subDomain = subDomain;
      }

      @Override
      public void setDriver() throws InvalidBrowserException {
            selenium = new SeleniumWebdriver(this, webEls, browser);
            driver = selenium.getDriver();
      }

      @Override
      public void unsetDriver() {
            if ( selenium != null ) {
                  selenium.killDriver();
            }
      }

      @Override
      public HashMap<Products,QuoteResult> login(User user) {
            //navigate to our URL
            selenium.goToURL( "http://" + subDomain + ".cosmiccomix.com/" );
            selenium.type( "username", user.getUser() );
            selenium.type( "password", user.getPassword() );
```

## External Website Class



```java
public class ExternalWebsite extends TestStructure {
    private LoadElements    webEls;
    private Browsers        browser;
    private String          subDomain;
    private Age             age = new Age();

    public ExternalWebsite(LoadElements webEls, Browsers browser, String subDomain,
            Databases database, Environments environment, String library) {
        this.webEls = webEls;
        this.browser = browser;
        this.subDomain = subDomain;
    }

    @Override
    public HashMap<Products,QuoteResult> getQuote(Databases data, Environments environment,
            QuoteQuery quote) throws InvalidActionException,
            InvalidLocatorTypeException, InvalidProductTypeException, InvalidApplicantException {
        Quote q = new Quote();
        Member member = quote.getMember();
        ArrayList<Dependent> dependents = quote.getDependents();
        //////////////////////
        //initial page
        ////////////////////
        //navigate to our URL
        selenium.goToURL( "http://" + subDomain + ".deltadentalcoversme.com/" );
        selenium.type( "zipcode_input", zip );
        selenium.click( "get_quote_button" );
        ////////////////////////////////
        //fill out our effective date
```

## Executing Tests

## Executing Tests

- Cucumber can currently be executed using two different methods.
  - A command line tool
  - A JUnit runner
- Connecting through JUnit with a runner such as Ant can make it a seamless part of a project developed using tests.
- Multiple inputs can be set, which can be overridden by values in the build.xml script

```
package comix.cosmic.definitions;

@RunWith(Cucumber.class)
@Cucumber.Options(
          features = "comix/cosmic/features/",
          glue = "comix.cosmic.definitions",
          format = {"pretty", "html:target/cucumber-html-report", "json-pretty:target/cucumber
          tags = {"@Smoke"}
)
public class TestRunner{
}
```

## Test Inputs (cont.)

- Additionally, Java system environments can also be set from the Ant script to create a more dynamic testing scenario.

```
<java classname="cucumber.api.cli.Main" fork="true" failonerror="false" resultproperty="cucumber.exitstatus">
    <classpath refid="classpath"/>
    <arg value="--format"/>
    <arg value="junit:target/cucumber-junit-report/allcukes.xml"/>
    <arg value="--format"/>
    <arg value="pretty"/>
    <arg value="--format"/>
    <arg value="html:target/cucumber-html-report"/>
    <arg value="--format"/>
    <arg value="json:target/cucumber.json"/>
    <arg value="--glue"/>
    <arg value="dental.delta.definitions"/>
    <arg value="${feature}"/>
    <arg value="--tags"/>
    <arg value="${dataenvironment}"/>
    <arg value="--tags"/>
    <arg value="${testtype}"/>
    <sysproperty key="dataenvironment" value="${dataenvironment}"/>
    <sysproperty key="environment" value="${environment}"/>
    <sysproperty key="testinterfaces" value="${testinterfaces}"/>
</java>
```

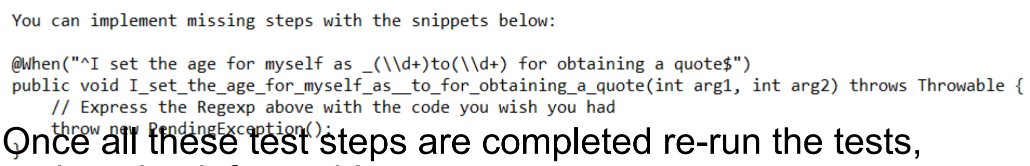
## Interpreting Results

## General Results

- Upon completion, test results will be available in the project directory under a folder labeled target.
- Navigate to the folder labeled *cucumber-html-report* and open the *index.html* file.
- This file will show your results.
- There are four different stylings for the results.
  - Unimplemented
  - Failed
  - Skipped
  - Passed

## Unimplemented Steps

- The test step was not properly implemented and therefore could not be run.
- These will appear yellow in the cucumber HTML results
- If run directly as a JUnit test, the output will offer suggestions on how to implement the missing steps



```
You can implement missing steps with the snippets below:

@When("^I set the age for myself as _(\\d+)to(\\d+) for obtaining a quote$")
public void I_set_the_age_for_myself_as__to_for_obtaining_a_quote(int arg1, int arg2) throws Throwable {
    // Express the Regexp above with the code you wish you had
    throw new PendingException();
}
```

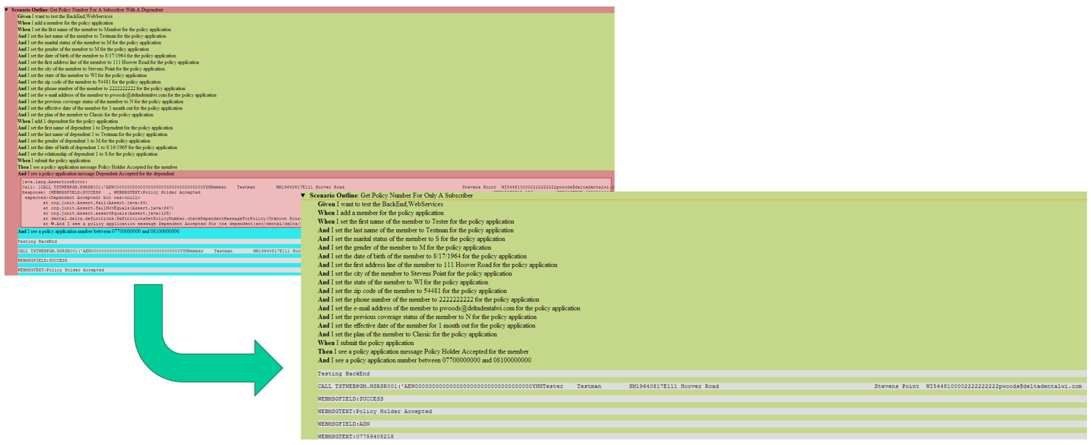
- Once all these test steps are completed re-run the tests, and re-check for problem areas.

## Failing Steps

- These are steps where an exception was thrown (either via asserts, or Java exceptions).
- The next step is to examine these test steps.
- There are also several possibilities for a bad test steps.
  - The test code may not be waiting properly for a return
  - The code may be checking a bad field
  - The internal call may have changed
  - The input data may be bad
  - The expected output may have changed.

## Getting to Green

- After all of these un-implemented and failed test steps are fixed, and the tests are re-run, the skipped (blue) test steps should resolve themselves.



▼ **Scenario Outline**: Get Policy Number For A Subscriber With A Dependent
**Given** I want to test the BackEnd,WebServices
**When** I add a member for the policy application
**When** I set the first name of the member to Member for the policy application
**And** I set the last name of the member to Testman for the policy application
**And** I set the marital status of the member to M for the policy application
**And** I set the gender of the member to M for the policy application
**And** I set the date of birth of the member to 8/17/1964 for the policy application
**And** I set the first address line of the member to 111 Hoover Road for the policy application
**And** I set the city of the member to Stevens Point for the policy application
**And** I set the state of the member to WI for the policy application
**And** I set the zip code of the member to 54481 for the policy application
**And** I set the phone number of the member to 2222222222 for the policy application
**And** I set the e-mail address of the member to pwoods@deltadentalwi.com for the policy application
**And** I set the previous coverage status of the member to N for the policy application
**And** I set the effective date of the member for 1 month out for the policy application
**And** I set the plan of the member to Classic for the policy application
**When** I add 1 dependent for the policy application
**And** I set the first name of dependent 1 to Dependent for the policy application
**And** I set the last name of dependent 1 to Testman for the policy application
**And** I set the gender of dependent 1 to M for the policy application
**And** I set the date of birth of dependent 1 to 8/16/1965 for the policy application
**And** I set the relationship of dependent 1 to S for the policy application
**When** I submit the policy application
**Then** I see a policy application message Policy Holder Accepted for the member
**And** I see a policy application message Dependent Accepted for the dependent

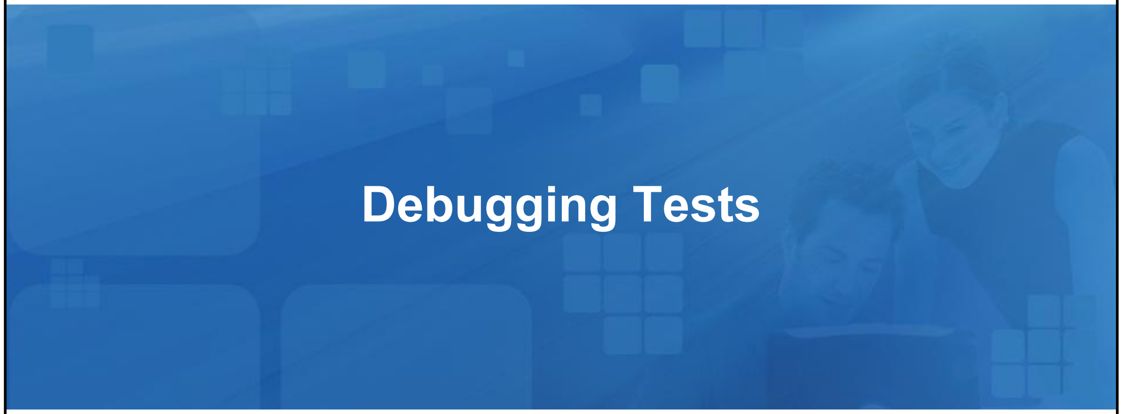
```
java.lang.AssertionError:
Call: [CALL TSTWEBPGM.NSRSR001('AEW000000000000000000000000000000000000YYNMember    Testman        MM19640817E111 Hoover Road                                          Stevens Point  WI5448100002222222222pwoods@deltadentalwi.c
Response: [WEBMSGFIELD:SUCCESS   , WEBMSGTEXT:Policy Holder Accepted
 expected:<Dependent Accepted> but was:<null>
        at org.junit.Assert.fail(Assert.java:93)
        at org.junit.Assert.failNotEquals(Assert.java:647)
        at org.junit.Assert.assertEquals(Assert.java:128)
        at dental.delta.definitions.DefinitionsGetPolicyNumber.checkDependentMessageForPolicy(Unknown Sourc
        at ✲.And I see a policy application message Dependent Accepted for the dependent(src/dental/delta/:
```

**And** I see a policy application number between 07700000000 and 08100000000

```
Testing BackEnd
CALL TSTWEBPGM.NSRSR001('AEW000000000000000000000000000000000000YYNMember    Testman        MM19640817E111 Hoo
WEBMSGFIELD:SUCCESS
WEBMSGTEXT:Policy Holder Accepted
```

▼ **Scenario Outline**: Get Policy Number For Only A Subscriber
**Given** I want to test the BackEnd,WebServices
**When** I add a member for the policy application
**When** I set the first name of the member to Tester for the policy application
**And** I set the last name of the member to Testman for the policy application
**And** I set the marital status of the member to S for the policy application
**And** I set the gender of the member to M for the policy application
**And** I set the date of birth of the member to 8/17/1964 for the policy application
**And** I set the first address line of the member to 111 Hoover Road for the policy application
**And** I set the city of the member to Stevens Point for the policy application
**And** I set the state of the member to WI for the policy application
**And** I set the zip code of the member to 54481 for the policy application
**And** I set the phone number of the member to 2222222222 for the policy application
**And** I set the e-mail address of the member to pwoods@deltadentalwi.com for the policy application
**And** I set the previous coverage status of the member to N for the policy application
**And** I set the effective date of the member for 1 month out for the policy application
**And** I set the plan of the member to Classic for the policy application
**When** I submit the policy application
**Then** I see a policy application message Policy Holder Accepted for the member
**And** I see a policy application number between 07700000000 and 08100000000

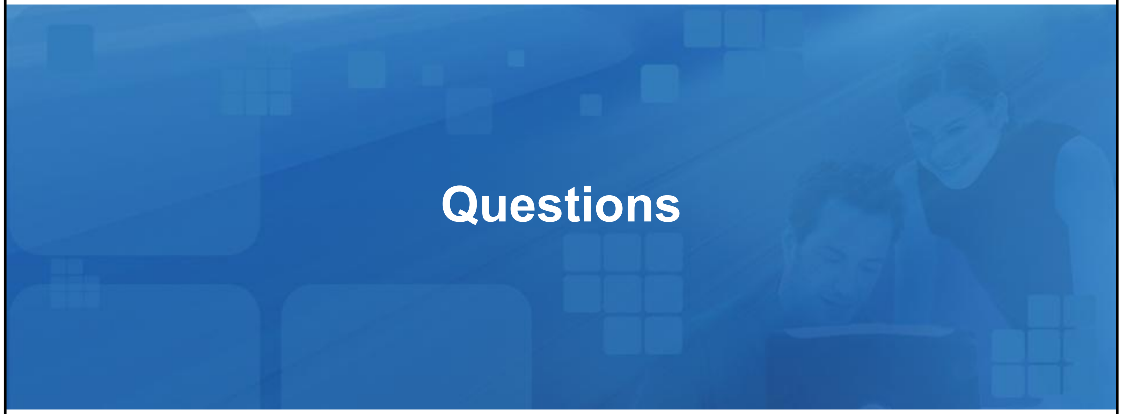
```
Testing BackEnd
CALL TSTWEBPGM.NSRSR001('AEW000000000000000000000000000000000000YNNTester    Testman        SM19640817E111 Hoover Road                                          Stevens Point  WI5448100002222222222pwoods@deltadentalwi.com
WEBMSGFIELD:SUCCESS
WEBMSGTEXT:Policy Holder Accepted
WEBMSGFIELD:ADN
WEBMSGTEXT:07759405218
```



## Debugging Tests

## Debugging Options

- Missing Locators
  - Selenium IDE
  - xPather
  - Firebug
  - WebDriver Element Locator
- Failing Test Steps
  - Input/Output Validation
  - Change of workflow
  - Change of design
- Thrown Exceptions
  - Eclipse debugging mode
    - Breakpoints
    - System output


Questions