**Serenity with Cucumber:**

**<argLine>-javaagent:serenity/serenity.jar -Xms512m -Xmx1024m ${included.packages} ${included.adapters}</argLine>**

|  |
| --- |
|  |

<properties>

<included.packages>-Dincluded.packages=your.package.name.here</included.packages>

<included.adapters>-Dincluded.adapters=coverage:complexity:dependency</included.adapters>

</properties>

surefire plugin configure --- pom.xml

<properties>

<included.packages>-Dincluded.packages=discover</included.packages>

<included.adapters>-Dincluded.adapters=coverage:complexity:dependency</included.adapters>

</properties>

<build>

...

<plugins>

...

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<forkMode>once</forkMode>

<argLine>-javaagent:serenity/serenity.jar -Xms512m -Xmx1024m ${included.packages} ${included.adapters}</argLine>

</configuration>

</plugin>

...

</plugins>

...

</build>

Cucumber is a popular BDD test automation tool. Cucumber-JVM is the Java implementation of Cucumber. In Cucumber, we express acceptance criteria in a natural, human-readable form.

For example, we could write the "wool scarf" example mentioned above like this:

Given I want to buy a wool scarf

When I search for items containing 'wool'

Then I should only see items related to 'wool'

This format is known as *Gherkin*, and is widely used in Cucumber and other Cucumber-based BDD tools such as SpecFlow (for .NET) and Behave (for Python). Gherkin is a flexible, highly readable format that can be written collaboratively with product owners to ensure that everyone.

The loosely-structured *Given-When-Then* format helps people focus on what they are trying to achieve, and how they will know when they get it.

Sometimes tables can be used to summarize several different examples of the same scenario. In Gherkin, you can use example tables to do this. For instance, the following scenario illustrates how you can search for different types of products made of different materials:

Scenario Outline: Filter by different item types

Given I have searched for items containing '<material>'

When I filter results by type '<type>'

Then I should only see items containing '<material>' of type '<type>'

Examples:

| material | type |

| silk | Handmade |

| bronze | Vintage |

| wool | Craft Supplies |

**Writing executable specifications with Cucumber and Serenity**

In Cucumber, scenarios are stored in *Feature Files*, which contain an overall description of a feature as well as a number of scenarios. The Feature File for the example above is called  search\_by\_keyword.feature, and looks something like this like this:

Feature: Searching by keyword

In order to find items that I would like to purchase

As a potential buyer

I want to be able to search for items containing certain words

Scenario: Should list items related to a specified keyword

Given I want to buy a wool scarf

When I search for items containing 'wool'

Then I should only see items related to 'wool'

These feature files can be placed in different locations, but you can reduce the amount of configuration you need to do with Serenity if you put them in the src/test/resources/features directory.

You typically organize the feature files in sub-directories that reflect the higher-level requirements. In the following directory structure, for example, we have feature definitions for several higher-level features: *search* and *shopping\_cart*:

|----src

| |----test

| | |----resources

| | | |----features

| | | | |----search

| | | | | |----search\_by\_keyword.feature

| | | | |----shopping\_cart

| | | | | |----adding\_items\_to\_the\_shopping\_cart.feature

Another option is to place them in src/test/resources', but underneath the same package name as your scenario runner class (see below). This requires slightly less configuration of the scenario runner class. However in this case, you need to specify the `thucydides.requirements.dir property in your serenity.properties (or thucydides.properties) file to point to the root requirements directory:

thucydides.requirements.dir=src/test/resources/net/serenity\_bdd/samples/etsy/features

**The Scenario Runner**

Cucumber runs the feature files via JUnit, and needs a dedicated test runner class to actually run the feature files. When you run the tests with Serenity, you use the CucumberWithSerenity test runner.

If the feature files are not in the same package as the test runner class, you also need to use the @CucumberOptions class to provide the root directory where the feature files can be found. A simple test runner looks like this:

package net.serenity\_bdd.samples.etsy.features;

import cucumber.api.CucumberOptions;

import net.serenitybdd.cucumber.CucumberWithSerenity;

import org.junit.runner.RunWith;

@RunWith(CucumberWithSerenity.class)

@CucumberOptions(features="src/test/resources/features/search/search\_by\_keyword.feature")

public class SearchByKeyword {}

If your feature files are stored in or under the same package as your scenario runner class (in src/test/resources) as discussed in the previous section, then you don’t need to use @CucumberOptions to provide the root directory.

**Step definitions**

In Cucumber, each line of the Gherkin scenario maps to a method in a Java class, known as a *Step Definition*. These use annotations like @Given, @When and @Then match lines in the scenario to Java methods. You define simple regular expressions to indicate parameters that will be passed into the methods:

public class SearchByKeywordStepDefinitions {

@Steps

BuyerSteps buyer;

@Given("I want to buy (.\*)")

public void buyerWantsToBuy(String article) {

buyer.opens\_etsy\_home\_page();

}

@When("I search for items containing '(.\*)'")

public void searchByKeyword(String keyword) {

buyer.searches\_for\_items\_containing(keyword);

}

@Then("I should only see items related to '(.\*)'")

public void resultsForACategoryAndKeywordInARegion(String keyword) {

buyer.should\_see\_items\_related\_to(keyword);

}

}

These step definitions use Serenity to organize the step definition code into more reusable components. The @Steps annotation tells Serenity that this variable is a Step Library. In Serenity, we use Step Libraries to add a layer of abstraction between the "what" and the "how" of our acceptance tests. The Cucumber step definitions describe "what" the acceptance test is doing, in fairly implementation-neutral, business-friendly terms. So we say "searches for items containing *'wool*", not "enters *'wool*' into the search field and clicks on the search button". This layered approach makes the tests both easier to understand and to maintain, and helps build up a great library of reusable business-level steps that we can use in other tests. Without this kind of layered approach, step definitions tend to become very technical very quickly, which limits reuse and makes them harder to understand and maintain.

Step definition files need to go in or underneath the package containing the scenario runners:

|----src

| |----test

| | |----java

| | | |----net

| | | | |----serenity\_bdd

| | | | | |----samples

| | | | | | |----etsy

| | | | | | | |----features

| | | | | | | | |----AcceptanceTests.java

| | | | | | | | |----steps

| | | | | | | | | |----SearchByKeywordStepDefinitions.java

| | | | | | | | | |----serenity

| | | | | | | | | | |----BuyerSteps.java

|  |  |
| --- | --- |
|  | The scenario runner package |
|  | A scenario runner |
|  | Step definitions for the scenario runners |
|  | Serenity Step Libraries are placed in a different sub-package |

**Serenity Step Libraries and Page Objects**

Both scenario step libraries (annotated with the @Steps annotation) and Page Objects that are declared inside the Cucumber step definition classes will be automatically instantiated.