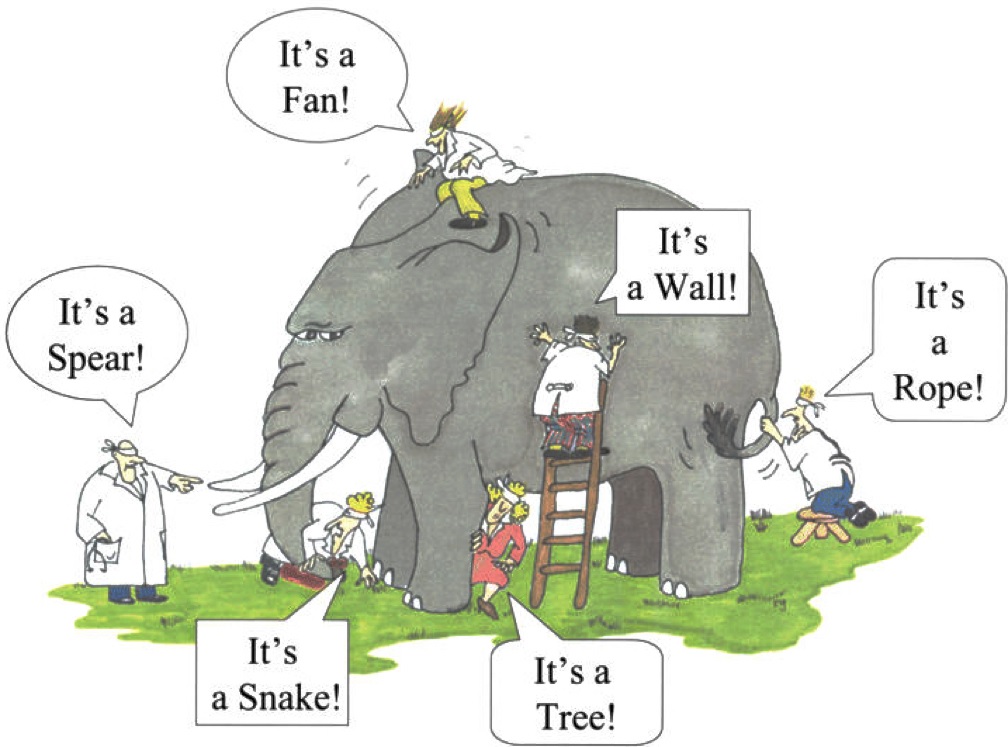
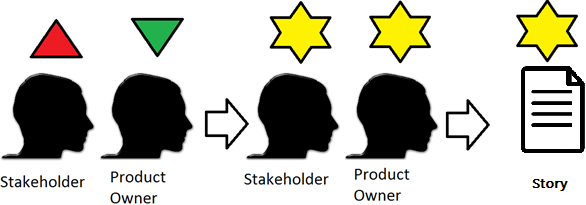
**Chapter 1: STORY**

The stories are written by the Product Owner and Stakeholders (a Stakeholder is any person that has an interest in the product, it could be monetary interest or a person who will be using the product). JBehave consists of two parts, the first part is called the narrative and the second part is the Scenarios which are written using Gherkin. Both of them are written in business language, allowing them to have a common language in order to have better communication and reduce the misinterpretation. 

*Image 1: Illustrates how each person interprets what they are in contact with due that they are blind folded and they aren’t able to see and not willing to move around to see the big picture. This same thing happens when the development team and the stake holder analyze a problem, they can only associate with what they are in contact with. [2]*

The hardest task of developing a software is for all of the people involved to have the same idea of the feature, and be able to write that concept in a story, so that the person that reads it will be able to abstract the same idea. Even though the people who are working on the project might speak the same language for example (English, Spanish or any natural language), more than likely there will be communication problems which are usually caused by the different jargons in language caused by the area in which they work. For example the development team uses a technical jargon to communicate with the stakeholders who generally have a different Jargon which might be more business oriented to communicate among themselves.



*Image 2: Illustrates the ideal situation of each person having different interpretation of a story and after talking about it thoughtfully they both end up with the same concept of the story.*

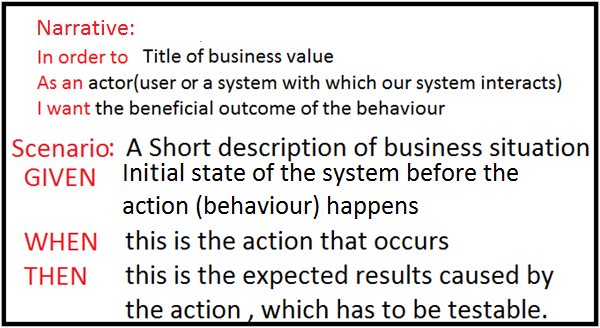
This issue has been solved with the implementation of Gherkin. Gherkin is a new way to write test cases and it can be written in 60 different natural languages. It uses the following Basic Syntax:

Scenario: Given When Then

Depending on which testing tool you will use, additional tags to the basic ones might be used. For example if you use Cucumber you will need to use the feature tag. In addition to this tag you will have to save your test case with an extension “.feature”, and they are called features.

JBehave uses Gherkin and also follows the basic syntax when you write the test cases. This test cases are called stories and they are saved with an extension “.story”. A story is used to capture the behavior of a module of the software being developed. It is an important tool allowing non-engineers to make essential contribution on the testing without even knowing how to program. At the same time it allows the people who interact with the story to have the same concept of the behavior we want to test. The people who will interact with the story are: stakeholder, product owner, QA, Scrum Master, Development team to define it in a way that reading it you can get the concept of the behavior being tested.

A Story written in JBehave follows the following syntax:



*Image 3: Example of the syntax of a JBehave story, and brief description of what each tag does.*

When you are planning on writing a test case you always have to take under consideration the Single Responsibility Principle (this principle states that a piece of code like a function or class can only do one thing and one thing only). SRP allows you to have a very maintainable code since it has high cohesion and low coupling in other words allowing you to only change the code in one place instead of analyzing the entire code and seeing the interdependencies with other modules and having to change all of the modules where a dependency exists.

When you’re writing a test case using Gherkin and JBehave there are a few keywords used to write any story. These are:

* Scenario: This is the title that you will give to the test case, which will provide a person that reads this test case the intention of what this test case test; in other words it will provide in a natural language a title to this business logic module.
* Given:Is a step, of the precondition of the system, it is like a stage. Only in this precondition, the action can happen. The use of several givens is not scene as a good practice, but you could do it.
* When: is a step in which the action happens or you can think of it as in this step is where the behavior of the test case happens. It is strongly recommended the use of a single “When” in a scenario, because otherwise it looks like you are testing more than one behavior, and that you might have to brake this scenario into two Scenarios.
* Then: is a step, in which you can see the post condition of the action generated in the previous step. It is strongly recommended the use of a single “Then” because otherwise it looks like you are testing more than one behavior, and that you might have to brake this scenario into two Scenarios.
* And: is a step in which you can use when you have a series of Givens or Whens. The rule is the same for Given and when tags but for this example we will use for a several Given steps. There is a series of Given steps so what you do is to leave the first Given step (Given header tag) with the Given tag. The And tag is a “polymorphic” tag because it takes the representation of the Header Tag so the next Given Steps you can replace the Given tag for an And tag, but when you run the story the Ands will represent a Given tag. This allows you in a Scenario to have several tags of Givens, Whens, and know where the different tags begin allowing our scenario to be more readable. To settle this concepts we will use the following test case:

Example of usage of “And” step when defining a story with a sequence of “When” steps: You are in the login page and the Log in button is disabled by default. In order for the login button to be enabled first the user must enter the username and the password, after filling them out you must click on the checkbox to verify for any suspicious data like a SQL injection or a Denial of Service Attack. After the input data is verified it enables the Sign In button.

I

*Image 4: Illustrates the initial state of the System.*



*Image 5: State of the System after the action of the user.*

If we have to write the Story(Behaviour) for the above example using the basic syntax,

Scenario: User fills in user, name and checks the checkbox for validation. After validation the log in button will be enabled

Given user is on the login homepage Given user fills user ‘user’ and password field ‘password’ When user checks the checkbox Then the Log in button will be enabled

If we replace the series of Givens it will look like the following:

Scenario: User fills in user, name and checks the checkbox for validation. After validation the log in button will be enabled

Given user is on the login homepage And user fills user ‘user’ and password field ‘password’ When user checks the checkbox Then the Log in button will be enabled

* But this step is a conditional step
* Example is used when you have an input in the given and want to use different values for the input; instead of writing the scenario several times and putting the different inputs for each of the scenarios. You use your Sceario as a template and write in the lower part example with the name of the variable and the different possible values you want for it to have.

Scenario: User fills in user, name and checks the checkbox for validation. After validation the log in button will be enabled

Given user is on the login homepage And user fills user ‘user’ and password field ‘password’ When user checks the checkbox Then the Log in button will be enabled

Example: | user | password | | ”user1” | ”password2345” | | ”user2” | “pass2345” |

There is a saying that says “We need to be sure to deliver the right software and delivered in the right way”. Right software means to define well the requirements but as technical persons we only focus on “delivering the right way”, this means as technical people we focus on clean code, design patterns including (SOLID principles), architectural patterns we focus just on the product.

I think it is important to know the difference between and error, defect and failure. An error is caused by a compiler or by human interaction with that system, causing it to generate an incorrect result. Saying it in other words it is when the software doesn’t behaves how it is supposed to behave. . A bug is the result of the error when the software is running. A defect is an error found by a tester. A failure is the result of a defect, you can actually notice the failure because the software isn’t doing what it is supposed to do or it is slowing.

In order to motivate you how important defining the requirements are:

* 41-56% of errors discovered are caused by not being able to dig deep in the requirements.
* Out of the top 8 reasons why projects fail, 5 of them are related to poor requirements.
* 80% of all defects present at the moment of development are add when requirements are defined.
* The cost fixing errors spend from 28% to a little above 45% of the projects budget.
* Errors cause increase of schedule time by 22%.

**Chapter2: FINDING ELEMENTS IN A WEBSITE**

When you are testing the User Interface of a website with Behavior Driven Development, you should be able to find any elements on a webpage so that you can be able to work with them in order to be able to comply with the respective story. Serenity offers you the @FindBy, which allows you to mark an element (node) and retrieve it. It retrieves using PageFactory which will store in a list of WebElements which you can later assign and create to a PageObject. PageFactory finds and retrieves WebElements using either XPath or CSS selector.

As a common practice it is a better practice to try to find and element by using a CSS selector than trying to find a node using’ XPath. You can retrieve elements of a website with the @FindBy and then you store it to a WebElementFacade. Preferably when you search for an element it is strongly recommended for the object that you are looking for to have a unique identifier.

If you are using @FindBy and want to get an element using CSS locaters and the element has preferably a unique CSS locator you can retrieve elements by its’:

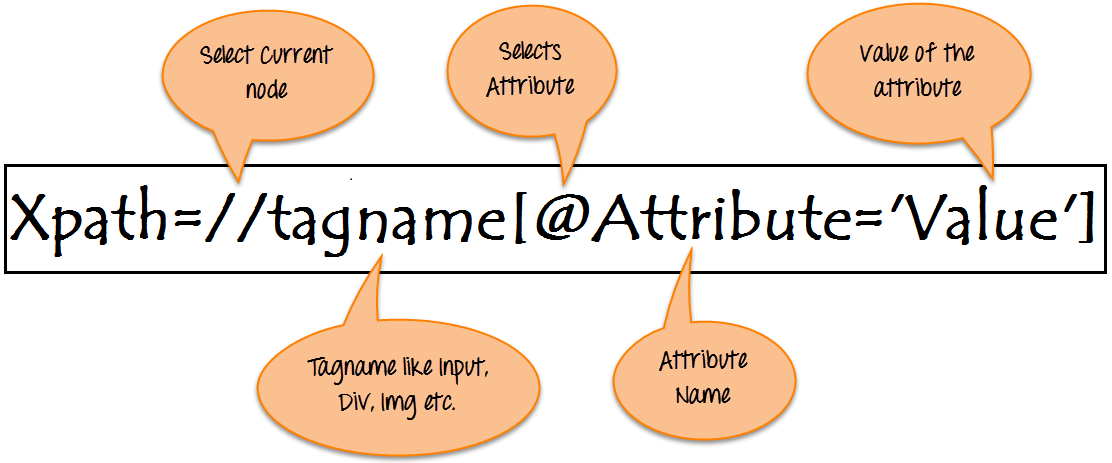
* Id - @FindBy(id=”<id’s name>”)
* tagName - @FindBy(tagName=”<tag>”)
* class - @FindBy( name = “<class name>”)

Another way of identify an element is using XPath. XPath is used as a last resort. XPath is the acronym for XML path and it allows you the find the path of an element in a web page by using the HTML DOM structure. For Xpath any element is a node and since it stores in an arrays.

This example is very simple but will help you understand the difference between Xpath Absolute and relative

To find and element there is 2 type of how to write an xpath:

* Absolute path: is the direct route to an element following the HTML DOM. Using simpler words but a little less technical terms it is the complete path from the root(/) to the node of the element we are looking for. The disadvantage of using the full path until the element is tike an instruction list of how to reach a point, but as you might know webpages change constantly so if one of those points is missing or has changed then there will be an error in that test.
* Relative path: Relative path allows to indicate in which node does the path starts and the node of the element we are looking, which in some cases is the same. You start writing a relative path using (//) The advantages are: the path will be shorter, there is a low percentage for an element in the path to be changed and if they changed other elements in the page it won’t affect our test.



*Image 6: This is the syntax used in Relative XPath and a brief description of what each part does. [1]*

In order to settle the concepts we will use an example based on image 7 where we have been asked by a person on America Av. outside AssureSoft building. The person tells us that they are out of town and want directions to IC Norte. The origin (root) is where AssureSoft is located (America Av. and Bernardo Monteagudo St.). Since we only have to go west to arrive to our destination we will only take on consideration street names we have to cross in order to reach our destination.



*Image 7: Our origin will be AssureSoft Bolivia (locate in America Av. and Bernardo Monteagudo St.) our destination will be IC Norte (located in America Av. and Pando St.)*

We have been asked the full path (absolute path) and the intersection (relative path) they would look something like this.

Absolute path: /Bernardo Monteagudo St./Miguel De Aguirre St./Melchor Urquidi Av/Pando Av.

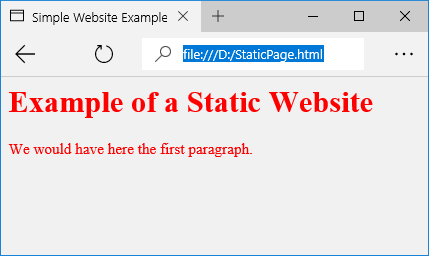
The path means that our origin is Bernardo Monteagudo St, the next Street is Miguel de Aguirre St, the one that follows is Melchor Urquidi Av. and finally we have reached our destination which is Pando Av.

Relative Path: //Pando Av.

The path means the person need to walk until he reaches Pando Av.

But lets say that the Mayor renamed Miguel De Aguirre Street to Cochabamba St., the person that is going with the absolute path will reach what is supposed to be Miguel De Aguirre St and sees Cochabamba St. he will think there is a mistake, and the same time happens when you are testing and will return an error. But if the street has been renamed to Cochabamba St., that won’t affect our relative path.

With the knowledge acquired, it is time to practice on a Website,.



*Image 7: This is a simple web page we are going to use as an example*

<html> <head> <title>Simple Website Example</title> </head> <body bgcolor="#f1f1f1" text="Red"> <h1 id="title">Example of a Static Website</h1> <p>We would have here the first paragraph.<p> </body> </html>

*Image 8: This is the HTML code of the web page for image 7*

Practice Exercise:

Use @FindBy to find some elements using CSS locator and by its Xpath, in the case of the Xpath write the absolute path and the relative Path.

1. Find the element with the content: “Example of a Static Website”
2. Find the element with the content: “We would have here the first paragraph”

Please fill in your answer.

|  |  |
| --- | --- |
| **CSS LOCATOR** | |
| **1.** | @FindBy() |
| **2.** | @FindBy( ) |

|  |  |
| --- | --- |
| **XPATH: ABSOLUTE PATH** | |
| **1.** | @FindBy( ) |
| **2.** | @FindBy( ) |

|  |  |
| --- | --- |
| XPATH: RELATIVE PATH | |
| **1.** | @FindBy( ) |
| **2.** | @FindBy( ) |

The

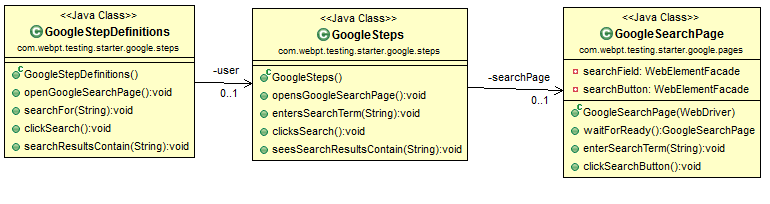
|  |  |
| --- | --- |
| **CSS LOCATOR** | |
| **1.** | @FindBy() |
| **2.** | @FindBy( ) |

|  |  |
| --- | --- |
| **XPATH: ABSOLUTE PATH** | |
| **1.** | @FindBy( ) |
| **2.** | @FindBy( ) |

|  |  |
| --- | --- |
| XPATH: RELATIVE PATH | |
| **1.** | @FindBy( ) |
| **2.** | @FindBy( ) |

**Chapter 3: STRUCTURE OF A PROJECT**

The Project is organized in 3 different paths, and there is a dependency between three classes, which in this tutorial they are known as standard classes. As a common practice you should name these classes by relating the name of the story followed by the name of each stand class name. In other words you should put the <story\_name> followed by the name of a standard class. These standard classes are: “StepDefinition”, “Steps” and “Page”. So if our story is GoogleSearch, would name your standard classes like in Image 6.



*Image 7: Class diagram and dependency between the classes.*

You can see in Image 6 you can see that GoogleStepDefinitions depends on the steps in GoogleSteps class; and GoogleSteps class depends on GoogleSearchPage.

 src/test/resources

In this folder you can store all the test cases. Starting in this folder and all of its’ child folders you need to add the Narrative.txt. A narrative is a short description of the intention of what will be tested in the folder. When you write a story, the narrative of the story needs to be the same as the narrative inside the Narrative.txt file. You need to define each scenario in a different folder.

 src/test/java

In this folder you need to add a class for each scenario. In this class you need to define where your story is located and its respective step definition.

 src/main/java

In this folder for each story you will have create a folder with the name relating it to the story and inside this folder you’ll have to add two packages:

* steps
* pages

Starting with the steps package you’ll need to create a class without the main and name it StepDefinition. In the StepDefinition you will be able to associate the steps of its’ story to a functional code. The next thing you need to do is associate a new class which will provide and manage the functionality to each Step; this class you’ll name it <name\_of\_story>followed by Step. For example if the storyname was “GoogleSearch.story” you could name it “GoogleStep.java” or “GoogleSearchStep”. In the StepDefinition you add an “@” symbol preceding each Gherkin keyword in each step and followed with open and close parenthesis. Inside the parenthesis you need to write the rest of the step inside double quotes.

In this packages you will have strong cohesion and low coupling. In the steps package you will have a main class called step definition class you will have a step definition

JBEHAVE AND SERENITY will help you with testing automation, allowing to make acceptance testing as well as regression testing.

SERENITY

**Bibliography**

1. <http://www.guru99.com/xpath-selenium.html>
2. http://www.agilebuddha.com/agile/user-stories-lack-of-big-picture-leads-to-blind-man-product/