Using Async methods to avoid unnecessary waits

The use case

Say you need to automate a website. And the website has a popup that appears asynchronously after an amount of time. You cannot ignore the popup, since it will sooner or later block your script. So you have one of 2 choices: 1. Wait for the popup, and block the execution of the script 2. Run an `asyncjs` using WebDriver `JavaScriptExecutor`

Since option 2, unfortunately, runs the script asynchronously on the web page ***but still blocks selenium***, we are left with only one choice, really, which is to wait and block other execution until the popup is dismissed.

Selenium is a single thread application, so is javascript

The main constraint is that Selenium WebDriver is a single-thread application. It does not work with multi-thread. So how can we run asynchronous tests? Well, Javascript is a single thread as well. Javascript has an internal mechanism for running on the same thread async actions. Async, not parallel. In order to run async, we need to hook into places in the execution. Luckily, we have `EventFiringWebDriver` exactly for this point.

How does Selenium.Async work?

We hooked into the most basic comment a standard Selenium script does - search for elements (`findElement` and `findElements`). So the basic concept would be: 1. Define an async piece of code. 2. Register it to be executed. 3. Perform a regular selenium script that involves `findElement` and/or `findElements` .

That's it....

The feature is heavily documented, you can check it out. Take a look at the unit tests for gettings started examples

Examples

Example 1 - register the listener with the driver

```
WebDriver driver;

asyncListener = new AsyncListener();
driver = new EventFiringWebDriver(new FirefoxDriver());
((EventFiringWebDriver)driver).register(asyncListener);
```

Example 2 - print the title of the page async to the display

```
AsyncListener listener = new AsyncListener();
listener.setDispatchInterval(2, ChronoUnit.SECONDS);
((EventFiringWebDriver)driver).register(listener);

Observer o = new AsyncTask(driver, new SeleniumTask() {

@Override
public boolean doTask() {
    String pageTitle = driver.getCurrentUrl();
    System.out.println("Time is " + LocalTime.now() + ", page url is " + pageTitle);
```

```
    return false;
}
});

listener.register(o);
```

Example 3 - Wait for a pop up and dismiss it

```
AsyncListener listener = new AsyncListener();
listener.setDispatchInterval(2, ChronoUnit.SECONDS);
((EventFiringWebDriver)driver).register(listener);

Observer o = new AsyncTask(driver, new SeleniumTask() {
    @Override
    public boolean doTask() {
        boolean result = false;

        List<WebElement> elements = driver.findElements(By.id("popup-button-dismiss-locator"));
        if (elements.isEmpty())
            return false;

        try{
            elements.get(0).click();
            result = true;
        }
        catch(Throwable t){
            result = false;
        }

        return result;
    }
});

listener.register(o);
```

Using the Async Feature in the Verisoft Framework - An Example

In the VerisoftDriver object, there is built-in support for the async operation. Here is an example

```
@Test
public void shouldInvokeAsyncOperationOnceAfterFindBy(VerisoftDriver driver)
                                                        throws InterruptedException {
    driver.get(pageTestUrl);

    // Set the async task
    Observer o = new AsyncTask(driver, () -> {
        driver.findElement(By.id("texta")).sendKeys("test1");
        return true;
    });
    driver.async().register(o);

    // Default dispatch is 5 second
    Thread.sleep(5500);
    driver.findElement(By.id("texta"));
    String text = (String) driver.executeScript("return document.getElementById('texta')
```

```
                                                                    .value");

    // Note!! Verisoft Assert
    Asserts.assertEquals("test1", text, "text should have been " + "test1");
}
```