Selenium TestNG Java Web Automation Framework

- Author: Tyler Lee
- Version: 1.0

Simple Instructions

Writing Tests

- copy the template java file (`TestExample.java`) and write Selenium tests as desired using TestNG for assertions
  - common functions for Selenium (web navigation)

```
driver.get("<URL>");
// navigates to desired url
driver.findElement(By.className("button")).click();
// clicks on the warning button
driver.findElement(By.id("username")).sendKeys("example-user");
// finds login text box and types "example-user"
WebElement myInput = driver.findElement(By.cssSelector("input"));
// finds an element based on its CSS selector and copies a reference
to a local variable for future interaction with this element
```

  - common functions for TestNG (testing assertions)

```
Assert.assertEquals(driver.getTitle(), "Google");
// asserts the web site title is "Google"
Assert.assertEquals(myInput.getAttribute("value"), "4200");
// asserts the myInput value is "4200"
```

Utilities

Screenshots

- to take screenshots of a page use the `WAFScreenShotter` utility class

```
package util;

new WAFScreenShotter(<WEBDRIVER>, "<PICTURE NAME>");
// takes picture of current page and puts it current directory
```

- do not add a file extension to the picture name argument, a `.jpg` is automatically added

- if you already have a `WAFScreenShotter` object you can simply use the `.takeScreenshot()` method to take another screenshot

```
myWAFScreenShotter.takeScreenshot(driver, "example");
// takes picture and places "example.jpg" in current directory
```

File Downloads

- to download all files on a page matching a CSS selector pattern, use the `WAFFileDownloader` utility class

```
package util;

new WAFFileDownloader(<WEBDRIVER>, "<FOLDER NAME>", "<CSS SELECTOR>");
// downloads all files on current page based on selector into the user
specified folder
```

- be sure to include "Files" (capitalization matters) somewhere in your folder name if you want the folder deleted when `make clean` is run
- almost all of your CSS selectors should end in `a` since you should be clicking links to download files
- if you already have a `WAFFileDownloader` object you can simply use the `.downloadFiles()` method to download additional files

```
myWAFFileDownloader.downloadFiles(driver, "my-Files-Folder", "td > a");
// downloads all files on the page that are table element links
```

Compiling and Running Tests

Linux without CLI Arguments

- run the following command in the terminal

```
make
```

Linux with CLI Arguments

- run the following commands in the terminal

```
make compile
java -cp .:res/* Main <OPTIONAL ARGUMENTS>
```

- see **Optional Arguments** below for arguments

Windows with/without CLI Arguments

- run the following commands in the terminal

```
javac -cp "res/*;." util/*.java
javac -cp "res/*;." *.java
java -cp "res/*;." Main <OPTIONAL ARGUMENTS>
```

- see **Optional Arguments** below for arguments

Optional Arguments

`<OPTIONAL ARGUMENTS>` is replaced by your command line arguments

- `-b` or `--browser` followed by browser name (`chrome`, `firefox`, `edge`, `ie`) for specific browser
  - default is `chrome`
- `-w` or `--width` followed by port number for specific browser width
  - default is `1024`
- `-h` or `--height` followed by port number for specific browser height
  - default is `768`
- `-p` or `--port` followed by port number to run on a specific port number
  - parallelization on single port not possible at the moment so don't specify a port if you are running multiple tests
  - default is `0` (web driver will automatically choose ports)
- `-u` or `--url` followed by ip address for specific ip/site to be used
  - default is ``
  - usually specify this on a test by test basis, largely meant to quickly test the same functionality on two copies of the same site from the command line
- `-t` or `--tag` followed by a non-spaced comma-separated value of test class names
  - `Example1, Example2` to run TestExample1 and TestExample2 test classes for example
  - default is just `Example`

Output

- after the tests run, a folder called `test-output` will be generated
  - within this folder, another folder `html` exists and within that is the `index.html` file containing the reportNG report of how the tests went
  - the `testng-results.xml` is an `xml` report that can be used to display results within an environment like Jenkins or Zephyr

Cleaning/Resetting (Linux Only)

- if you want to start fresh and remove compiled files simply run

```
make clean
```

- if you want to remove everything but the bare essentials run

```
make superclean
```

- this removes the `build.xml` (recreated by `Main.java` each run) and the pdf copy of this README

Detailed Explanation

Overview

This framework works in the following manner:

1. All java files are compiled
2. Main is run
    1. Parses optional arguments
    2. Creates `build.xml` TestNG
    3. Uses `build.xml` as a template to run tests
        1. Initializes web driver
        2. Performs actions using Selenium
        3. TestNG test assertions are checked
3. TestNG framework creates output `.html` and `.xml` files to view test results

Making (Linux Only)

- `make` runs the following commands:

```
javac -cp .:res/* util/*.java
javac -cp .:res/* *.java
java -cp .:res/* Main
```

- `make compile` runs the following commands:

```
javac -cp .:res/* util/*.java
javac -cp .:res/* *.java
```

- `make clean` runs the following command:

```
rm -rf test-output *Files* *.class util/*.class *.zip *.png *.jpg
```

- `make superclean` runs the following command:

```
rm -rf test-output *Files* *.class util/*.class *.zip *.png *.jpg *.pdf
docs/*.pdf *.xml
```

Developers

Changelog

- **1.X**
  - demonstration of working selenium/testng functionality
  - portability assurance with chrome/linux
  - local report generation
  - make file for ease of compilation
  - cross platform/browser compatibility with windows/linux
  - user defined test parameters using advanced options parser
    - browser, resolution, port, url
  - parallel same browser tests
  - vastly improved documentation
  - abstracted screen-shotting, file downloading, and login processes
  - utility packaging improvements
  - web driver abstraction for better parallelization support
- **2.X** (future features)
  - catch test class doesn't exist exception
  - improve command line tag parsing
    - `5.1-7.1` would run every test that exists in that range
    - `5.1.1` would run just that specific step (wrapped in login and quit)
  - add test categorization and reversals in conjunction with one another
    - `smoke` would run just tests labeled with smoke
    - `not debug` would run all tests that aren't debug
    - combined these two tags would run smoke tests that aren't debug
  - make multiple tests with a specified port run each additional web driver in successive ports
    - for example, if the user has 3 tests to run and specifies port 4000 then web drivers are spun up on 4000, 4001, and 4002
  - auto fix the web drivers to whatever they need to be
    - right now if the defaults don't work you have to look at the web browser version and manually download the correct one
    - web requests to download new/old drivers?

Trouble Shooting

- If you get an error that says "port already in use" run the following
  - **Windows**: open task manager (right click taskbar) and find chromedriver/geckodriver/mswebdriver
    - right click and kill process
  - **Linux**:

        - open a terminal and run `killall chromedriver` or `killall geckodriver`
- Incompatible web drivers
    - download the correct ones and replace in the `drivers` folder
        - find your browser version (usually in settings > about) and google it for matching web driver version
- If compilation throws warnings on Windows try recompiling (rerunning the same steps) to remove these warnings

Known Bugs

- A port cannot be specified if multiple tests are run as a port collision will occur
- Windows compilation occasionally throws warnings that can be fixed by recompiling
- Internet Explorer web driver not working 😃