Basic Mobile UI Test

I can imagine the first thing you will want to do is to write a very simple and basic test - to heat up the engines. This page will guide you through the code:

```
@Execution(ExecutionMode.CONCURRENT)
```

Click here to expand...

This line tells the JUnit engine that the tests in this class were designed to run one after the other, without paralleling. Note! In order use JUnit parallel mechanism, there is a property file named junit-platform.properties located at src/test/resources folder

```
public class BasicMobileExampleTests extends BaseTest {
```

Click here to expand...

We extend the class BaseTest. BaseTest is a class that was created in the examples repo. It mainly holds the annotations and keeps this code clean.

```
@DriverUrl
private URL url = new URL("http://127.0.0.1:4723/wd/hub/");
```

Click here to expand...

There are 3 main annotations you will need to be familiar with in order to get your driver up without any issues: `@DriverCapabilities,` `@DriverUrl`, and `@DriverCommandExecutor`.

The `@DriverUrl` tells the VeriSoftDriver where to find the url to use when starting up. We use dependency injection for injecting the WebDriver directly into the test. More on our dependency injection mechanism for the driver can be found here. Note the /wd/hub at the end.

```
@DriverCapabilities
private DesiredCapabilities capabilities = new DesiredCapabilities();
```

Click here to expand...

The `@DriverCapabilities` tells the VeriSoftDriver where to find the DesiredCapabilities to use when starting up. We use dependency injection for injecting the WebDriver directly into the test.

```
@Test
@DisplayName("Click On Element")
```

Click here to expand...

Both lines are JUnit lines. The `@DisplayName` line is very important since if you use this annotation, then most of the report mechanisms will use it as the test name, rather than the method name.

```
    public void clickOnElement(VerisoftMobileDriver driver) {
```

Click here to expand...

We use a dependency injection mechanism, which takes the `@DriverCapabilities, @DriverUrl`, and `@DriverCommandExecutor` (if present) and creates an instance of `VerisoftDriver`. The `VerisoftDriver` object is an instance of `WebDriver` and follows its interfaces and more. More about the VeriSoftDriver and VerisoftMobileDriver can be found here.

```
// Note!! Verisoft Assert
String phrase = "No thanks";
```

```java
Asserts.assertTrue(e.getText().contains(phrase), "Page should contain the pharase: " + phrase);
```

Click here to expand...

Asserts in the Verisoft framework do pretty much the same as regular asserts. The difference here is that we log the message to the various logging and reporting mechanisms that are listening to out Observer Report mechanism. More about the Report in the following line. You may choose to use a different Assert mechanism (and there are plenty of those in Java, but you will lose this nice logging feature.

```java
Report.info("Got to this point - We are operating Google chrome on mobile deviceS");
```

Click here to expand...

There are many places we want to write reports to - logging, report mechanism, perhaps a global logging system like Splunk, Jira,, etc. If we would write a line of code for each of the mechanisms we support, we would have more logging lines of code than actual code. So, we use the Observer pattern and use one single Report object to write to all of them. More about the Report object here.

Well, that's it! We made it through a basic web UI test. Here is the complete code, and you can also view it on Bitbucket

```java
@Execution(ExecutionMode.SAME_THREAD)
public class BasicMobileExampleTests extends BaseTest {

    @DriverUrl
    private URL url = new URL("http://127.0.0.1:4723/wd/hub/");

    @DriverCapabilities
    private DesiredCapabilities capabilities = new DesiredCapabilities();

    {
        capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");
        capabilities.setCapability("appPackage", "com.android.chrome");
        capabilities.setCapability("appActivity",
                                   "com.google.android.apps.chrome.Main");
        capabilities.setCapability("automationName", "UIAutomator2");
    }


    public BasicMobileExampleTests() throws MalformedURLException {
    }


    @Test
    @DisplayName("Click On Element")
    public void clickOnElement(VerisoftMobileDriver driver) {

        driver.findElement(By.id("com.android.chrome:id/terms_accept")).click();
        WebElement e =
              Waits.visibilityOfElementLocated(driver, 30,
                       By.id("com.android.chrome:id/negative_button"));

        // Note!! Verisoft Assert
        String phrase = "No thanks";
        Asserts.assertTrue(e.getText().contains(phrase),
                                 "Page should contain the pharase: " + phrase);

        Report.info("Got to this point - We are operating Google chrome on mobile");
    }
}
```