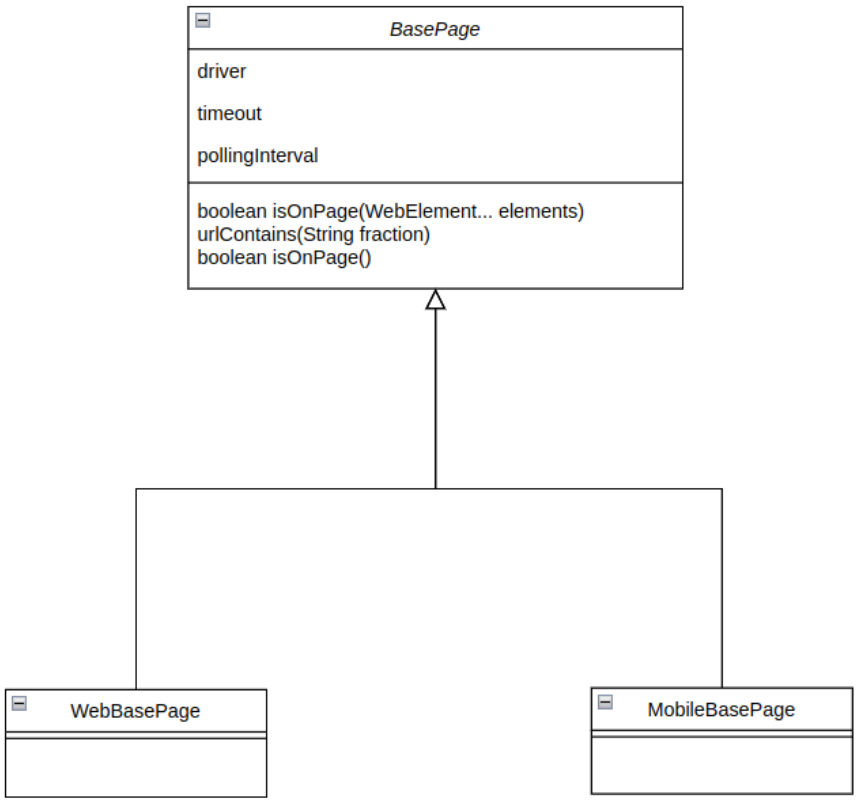
Working with page objects

In this page we will describe the basics of Verisoft Framework support of page objects. We assume you are familiar with the page object concept, and read at least one of the basic UI testing pages - either web UI basic test or mobile UI basic test.

Verisoft framework defines the following hyrarchy:



BasePage is an abstract class. It holds the WebDriver object, and some time related fields. It means each page object in the system should implement the method isOnPage(). In addition, the BasePage is in charge of PageFactory initialization for both web and mobile, so you don't have to worry about that. For WebBasePage objects - that's it. The rest is designed as a regular page object.

For MobileBasePage there are a few additional methods - swipes and scrolls.

In order to use the page object, all you need to to do is to extend the relevant base page. for web UI tests it looks like `public class WikipediaMainPage extends WebBasePage` and for mobile UI tests it looks like `public class MobileChromeWelcomePage extends MobilebBasePage`.

An example for a typical mobile page object will look like this:

```
1  public class MobileChromeWelcomePage extends MobileBasePage {
2
3      @AndroidFindBy(id = "com.android.chrome:id/terms_accept")
4      @iOSXCUITFindBy(id = "SOME IOS ID")
5      private WebElement accept;
6
7
8      public MobileChromeWelcomePage(WebDriver driver) {
9          super(driver);
10     }
11
```

```
12
13      @Override
14      public boolean isOnPage() {
15
16          // Verisoft wait object
17          return Waits.visibilityOf(driver, timeOut, accept).isDisplayed();
18      }
19
20
21      public MobileChromeTurnOnSyncPage accept(){
22          this.accept.click();
23          return new MobileChromeTurnOnSyncPage(driver);
24      }
25  }
```

Note that when implementing isOnPage method, we use Verisoft wait object. I simply provides a shorter way to define waits, with the exact same functionality as the Selenium wait object.

A typical web UI page object will look like this:

```
1   public class WikipediaMainPage extends WebBasePage {
2
3       @FindBy(id = "searchInput")
4       private WebElement searchBar;
5
6       public static final String pageUrl = "https://www.wikipedia.org/";
7
8
9       /**
10       * C-tor. Initializes generic properties such as timeOut and pollingInterval
11       *
12       * @param driver a WebDriver object to store and use
13       */
14      public WikipediaMainPage(WebDriver driver) {
15          super(driver);
16      }
17
18
19      @Override
20      public boolean isOnPage() {
21          return searchBar.isDisplayed();
22      }
23
24
25      public WikipediaResultPage searchForTerm(String term){
26          searchBar.sendKeys(term);
27          new Actions(driver).sendKeys(Keys.ENTER).build().perform();
28          return new WikipediaResultPage(driver);
29      }
30
31
32      public WikipediaMainPage gotoPage(){
33          driver.get(pageUrl);
34          return this;
35      }
36  }
```

Note that we are implementing the page flow mechanism, which means that once on the main Wikipedia page for instance, if we search and hit the enter button, the expected result should be that a Wikipedia result page will appear, so we create a new WikipediaResultPage and

return it to the user. The test code will look like this:

```
WikipediaMainPage wikipediaMainPage = new WikipediaMainPage(driver);

WikipediaResultPage resultPage = wikipediaMainPage.gotoPage().
                                    searchForTerm(phraseToSearch);
```

If you are not familiar with the page object design pattern, it is worth it to take some time and read about it. You may also find it useful to read our page about screenplay design pattern.