COMAQA.BY
QA automation community

Out of box Page Object Design Pattern, Java

Anton Semenchenko

About ☺

Founder of communities www.COMAQA.BY and www.CoreHard.by, co-founder of company www.DPI.Solutions, «tricky» manager at EPAM Systems. Almost 15 years of experience in IT, main specialization: Automation, C++ and lower development, management, sales.

Agenda, part 1 (to refresh)

1. Page Object – just to “refresh”
2. State-less or state-full solution - just to “refresh”
3. How to select a proper PO implementation?

Agenda, part 2 (to refresh)

- Hypothetical project context
- Compare complexity
- Compare “tech” limitations
- Compare “business” limitations
- Rationale “business” limitations
- How to find and “update” a balance for your own project?

Agenda, part 3 (solutions)

1. “raw” Selenium Page Objects
2. Page Factory from Selenium library
3. HMTL Element framework from Yandex
4. Selenide
5. JDI framework from EPAM
6. SWD Page Recorder

Agenda, part 4 (take away points)

1. A real-life example
2. “Homework”
3. “Rules” and principles
4. A set of useful links
5. What’s next?
6. Questions

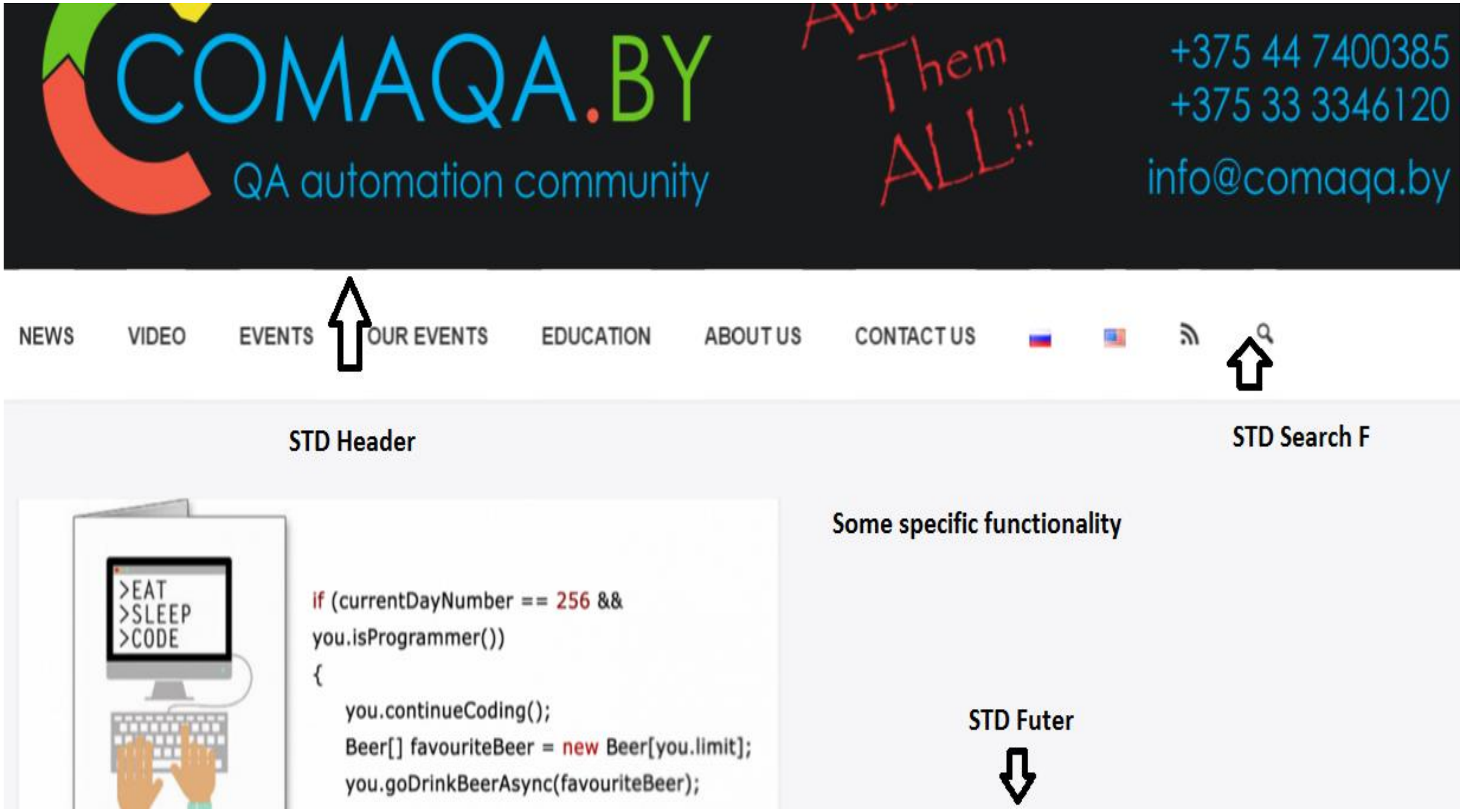
Page Object – just to “refresh”

1. Page Objects – encapsulates the way of identification and logical grouping of widgets.

1. Page Object == Logical Page

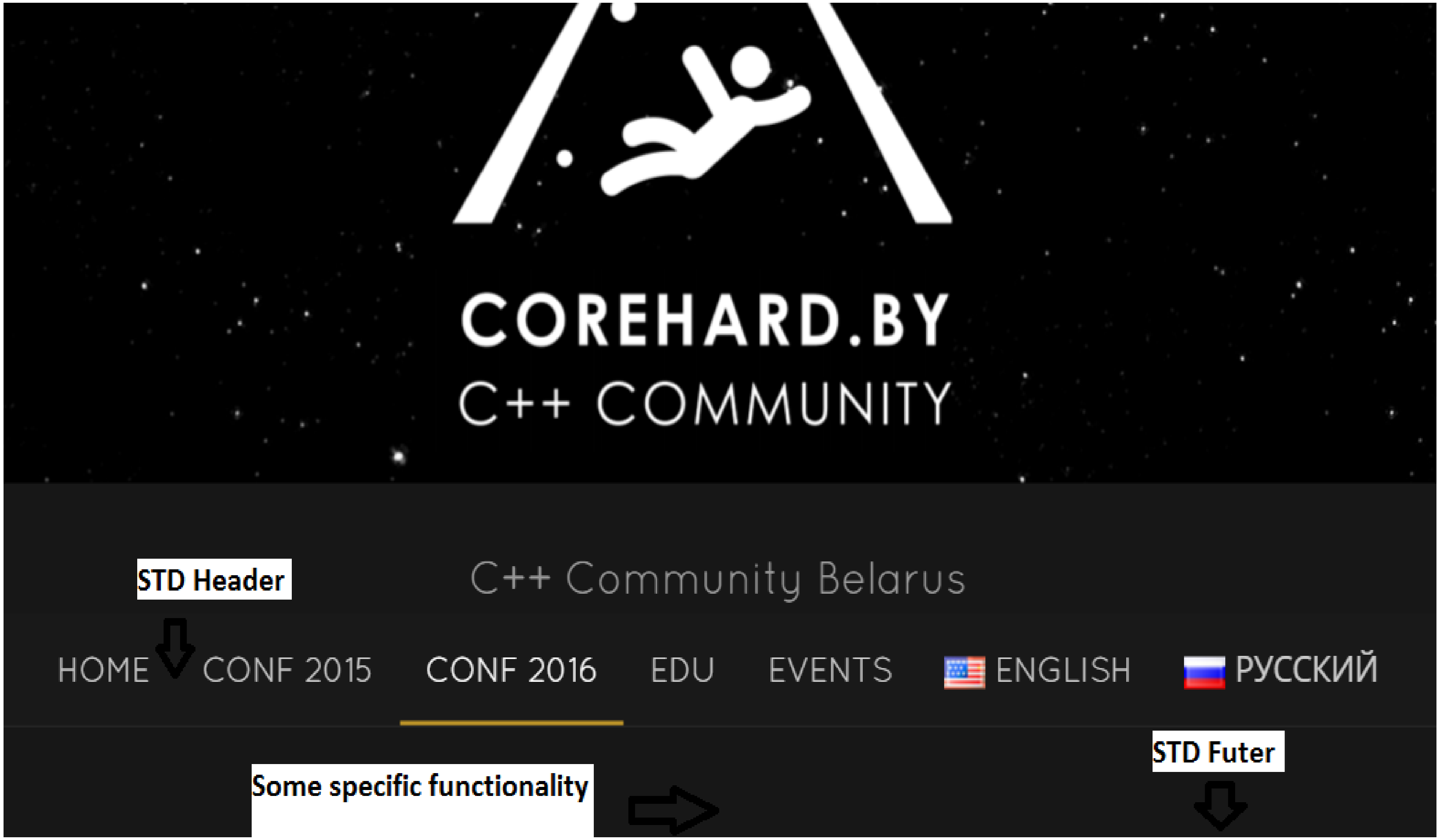
1. Page Object != Physical Page

Page Object – “example” ☺



```
if (currentDayNumber == 256 &&
you.isProgrammer())
{
    you.continueCoding();
    Beer[] favouriteBeer = new Beer[you.limit];
    you.goDrinkBeerAsync(favouriteBeer);
```

Page Object – “example” ☺



State-less or state-full solution?

1. Let’s compare:

   - Photo
     - Share – looks like parallelism (easy parallelism).
   - Video
     - Share – looks like parallelism (not trivial parallelism).

State-less or state-full solution?

1. How easy transform solution from “single” to “multi” threading (to decrease “QA Automation Windows”)

    1. State-less – like share a photo
        1. Just 5 minutes of work.
    2. State-full – like share a video
        1. Not trivial task, could be a night mare.

2. Summary

    1. prefer state-less solutions to state-full solutions in mooooost cases;
    2. before start implementation a state-full solution, please, take a break for a minute, and re-thing everything again, possibly you can find a proper state-less solution.

Object or static class \ State-full or state-less solution?

1. Static class
   - could be implemented as a state-less solution easily
2. Object
   - State-full solution in 99,99% cases
3. Summary
   - prefer static class based solutions (state-less) to object based (state-full) in mooooost cases;
   - before start implementation based on objects, please, take a break for a minute, and re-thing everything again, possibly you can find a proper solution based on static classes.

COMAQA.BY

Page Objects – state-full, general case?

1. Too complicated business logic due to Domain

2. Too complicated business logic due to System size (thousands test-cases)

3. Too many “contexts”

    - Browser versions
    - Environments
    - Customers with a tiny nuances of functionality
    - Platforms (cross-browser Web plus cross-platform Mobile)
    - Combination ☺

1. Combination ☺

Page Objects – state-full, special cases

1. Web UI that behaves like a Wizard

2. Web UI in combination with Mobile in one use case

3. Internet of Things (in most cases)

4. More then 1 page during 1 test (for example several portals or several instances of one portal to implement one “business use case”):

    – Really seldom;
    – Looks like integration tests (in most cases):
        • Std solution- some type of White Box Testing.

5. Many others “special cases”

How to select a proper PO implementation?

Hypothetical project context – to compare

1. 30 Physical Pages

2. Standard Header, footer and search functionality for all physical pages

3. Each Physical Page consists of

   – Header – H
   – Footer – F
   – Search – S
   – Some functionality, specific for the Page – U (unique)

Compare complexity – Static Page Object based

1. 33 Static Page Objects = Logical Pages
2. 0 explicit and 30 implicit Physical Pages
3. Just share 33 Static Page Objects between 30 implicit Physical Pages
   - For example, Header Static Page Object (static class) is used in test cases for all 30 Physical Page Objects

1. **Complexity – just 33 static state-less entities** (plus limitations due to state-less solutions)

Compare complexity – Dynamic Page Object based (option 1)

1. 33 Dynamic Page Objects = Logical Pages

2. It depends on implementation (one of the ways):

   – 0 explicit and 30 implicit Physical Pages
   – implicit Physical Page implements on Action layer (limitations example)
   – Action for Physical Page aggregates all necessary Dynamic Logical Pages
     - Physical Pages are implemented in a next way: create all necessary instances of logical pages, aggregate in some way, use Action layer as an point of aggregation in most cases, free resources

Compare complexity – Dynamic Page Object based (option 1)

1. 120 objects (min), each with some state, dynamically created and frees to implement necessary behavior - to implement 30 implicit Physical Pages

2. **Complexity – 120 dynamic, state-full entities min** (plus some limitations due to state-full solution implementation nuances)

Compare complexity – Dynamic Page Object based (option 2)

1. 33 Dynamic Page Objects = Logical Pages

2. It depends on implementation (another way):

   – 30 explicit Physical Pages
   – Multiple Interface inheritance
   – Combine Page Objects and Actions layer (in most cases)
   – Action-Physical Page (limitations example)
     - Implements all Logical Pages interfaces using aggregation and “delegation”
     - Aggregate all Dynamic Logical Page Objects
     - Create and frees resources (Dynamic Logical Page Objects)

Compare complexity – Dynamic Page Object based (option 2)

1. 150 objects, each with some state, dynamically created and frees to implement necessary behavior - to implement 30 explicit Physical Pages

2. **Complexity – 150 dynamic, state-full not trivial entities with a multiple interface inheritance** (plus some limitations due to state-full solution implementation nuances)

Compare “tech” limitations - Static Page Object based

1. Can be used together with next Design Patterns \ Approaches

   - Action (both static – preferable and dynamic)
   - Key-word driven
   - DSL – external only
   - BDD – partially, it depends on exact BDD engine implementation limitations

2. Can’t be used together with next Design Patterns

   - Factory
   - Flow (Fluent Interface)
   - Navigator (for Web)

Compare "tech" limitations - Dynamic Page Object based

1. No limitations, but …
   - For example, in most cases it's hard to isolate Action and Page Objects layers

Compare “business” limitations - Static

1. Too complicated business logic due to Domain

2. Too complicated business logic due to System size (thousands test-cases)

3. Too many “contexts”

   – Browser versions
   – Environments
   – Customers with a tiny nuances of functionality
   – Platforms (cross-browser Web plus cross-platform Mobile)
   – Combination ☺

1. Combination ☺

Rationale “business” limitations - Static

1. State-less approach - you have a conditional that chooses different behavior depending on …

2. Solution to simplify the project – refactoring “**Replace Conditional with Polymorphism**”

3. Polymorphism = object = State-full approach

Rationale “business” limitations - Static

1. “From refactoring to Patterns”
   - There is a set of specific Design Patterns
2. The trickiest part – find a balance for your project now and update point of balance in time

Compare “business” limitations - Dynamic

1. Relatively simple business logic due to Domain

2. Relatively simple business logic due to System size (hundreds test-cases)

3. Not so many “contexts”

   – Browser versions
   – Environments
   – Customers with a tiny nuances of functionality
   – Platforms (cross-browser Web plus cross-platform Mobile)

Rationale “business” limitations - Dynamic

1. State-full approach - you have a set of objects \ classes, which developed, possibly, using several Design Patterns to implement necessary functionality – to choose different behavior depending on …

2. Solution to simplify the project – refactoring “**Replace Polymorphism with Conditional**”

3. Polymorphism ~= object ~= State-full approach

4. No Polymorphism ~= no objects ~= State-less approach

Rationale "business" limitations - Dynamic

1. "From Patterns to refactoring"
   – There is no need to use a set of specific Design Patterns
2. The trickiest part – find a balance for your project now and update point of balance in time

Find and “update” a balance for your own project

Page Factory from Selenium library

Page Factory - definition

1. Page Factory is an **inbuilt page object model concept** for Selenium WebDriver but it is very optimized.
2. We use **initElements method to initialize web elements**

1. Additionally with the help of PageFactory class - use **annotations @FindBy to find WebElement**.
2. **@FindBy can accept as attributes**:
    1. tagName
    2. name
    3. partialLinkText
    4. linkText
    5. Id
    6. Css
    7. className
    8. Xpath

Page Factory - notes

1. If you use the PageFactory, you can assume that the **fields are initialized**. If you don't use the PageFactory, then NullPointerExceptions will be thrown if you make the assumption that the fields are already initialized.

2. List<WebElement> fields are decorated if and only if they have @FindBy or @FindBys annotation. Default search strategy "by id or name" that works for WebElement fields is hardly suitable for lists because it is rare to have several elements with the same id or name on a page.

Page Factory - notes

- **WebElements are evaluated lazily.** That is, if you never use a WebElement field in a PageObject, there will never be a call to "findElement" for it.

- **The functionality works using dynamic proxies**. This means that you shouldn't expect a WebElement to be a particular subclass, even if you know the type of the driver. For example, if you are using the HtmlUnitDriver, you shouldn't expect the WebElement field to be initialized with an instance of HtmlUnitWebElement.

Login Page

```
public class LoginPage {
    private WebDriver driver;
    @FindBy(id = "id_username")
    private WebElement usernameInput;
    @FindBy(id = "id_password")
    private WebElement passwordInput;
    @FindBy(id = "main_action_form_button")
    private WebElement loginButton;
}
```

Login Page

```
public class LoginPage {
    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }
    public void login(String username, String password) {
        usernameInput.sendKeys(username);
        passwordInput.sendKeys(password);
        loginButton.click();
    }
}
```

Home Page

```
public class HomePage {
    private WebDriver driver;
    @FindBy(css = "a[href*=\"login\"]")
    private WebElement loginButton;
    public HomePage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }
}
```

Home Page

```
public class HomePage {
    public void login() {
        login(Creds.username, Creds.password);}
    private LoginPage clickLoginButton() {
        loginButton.click();
        return new LoginPage(driver);}
    private void login(String username, String password) {
        LoginPage loginPage = clickLoginButton();
        loginPage.login(username, password);
```

Login test

```
public class LoginTest {
  private WebDriver driver;
@BeforeMethod
    public void setup() {
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
        driver.get("https://www.pandadoc.com/");}
    @AfterMethod
    public void teardown() {
        driver.close();}}
```

Login test

```java
public class LoginTest {
private void assertSignedUp() {
        new WebDriverWait(driver, 30000).until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".header-avatar")));}
    @Test
    public void loginTest() {
        HomePage homePage = new HomePage(driver);
        homePage.login();
        assertSignedUp();}}
```

Implementation is based on

- Reflection
- Annotation
- Templates

Page Factory from Selenium library - info

Page Object Model and Page Factory

Page Factory detailed desription

Page Object (Page Factory context)

AjaxElementLocatorFactory (obsolete)

Source code example

PageFactory.java (source code)

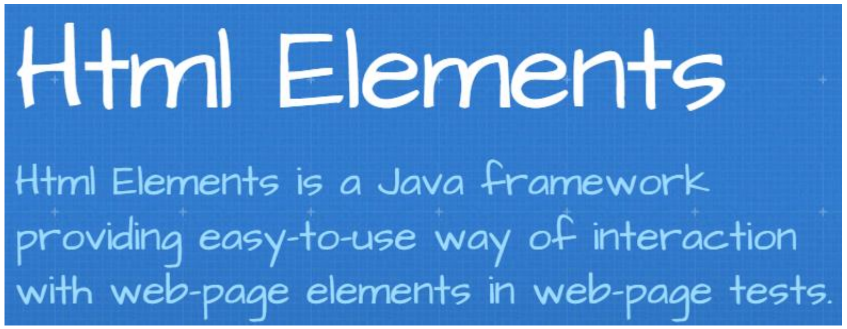
Book with a brief summary of Selenium WebDriver plus Java bindings (Ru only)

HMTL Elements framework from Yandex

- Designed to provide easy-to-use way of interaction with web-page elements in your tests.

- It may be considered as an extension of WebDriver Page Object (PageFactory).

- With the help of Html Elements framework you can group web-page elements into blocks (Logical Pages), encapsulate logic of interaction with them and then easily use created blocks in page objects (Physical Pages).

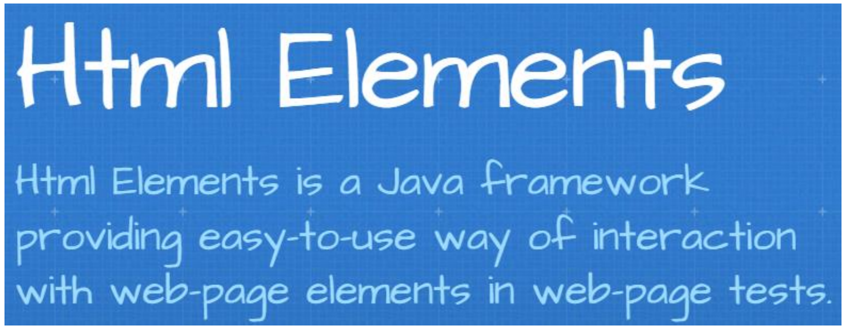


Html Elements

Html Elements is a Java framework providing easy-to-use way of interaction with web-page elements in web-page tests.

HMTL Elements framework from Yandex

- Dividing page on blocks using @Block annotation
- Separating different types of elements (button, checkbox)
- Provides an ability to annotate methods as steps – annotation @Step for work in step-based style
- Provides a set of helpful matchers to use with web-page elements and blocks



Html Elements

Html Elements is a Java framework providing easy-to-use way of interaction with web-page elements in web-page tests.

Login Page

```
public class LoginPage {
    private WebDriver driver;
    @FindBy(id = "id_username")
    private static TextInput usernameInput;
    @FindBy(id = "id_password")
    private static TextInput passwordInput;
    @FindBy(id = "main_action_form_button")
    private static Button loginButton;
}
```

Login Page

```java
public class LoginPage {
    public LoginPage(WebDriver driver) {
        this.driver = driver;
        HtmlElementLoader.populatePageObject(this, driver);
    }
    public void login(String username, String password) {
        usernameInput.sendKeys(username);
        passwordInput.sendKeys(password);
        loginButton.click();
    }
}
```

Home Page

```
public class HomePage {
    private WebDriver driver;
    @FindBy(css = "a[href*=\"login\"]")
    private static Button loginButton;
    public HomePage(WebDriver driver) {
        this.driver = driver;
        HtmlElementLoader.populatePageObject(this, driver);
    }
}
```

Home Page

```
public class HomePage {
    public void login() {
        login(Creds.username, Creds.password); }
    private LoginPage clickLoginButton() {
        loginButton.click();
        return new LoginPage(driver); }
    private void login(String username, String password) {
        LoginPage loginPage = clickLoginButton();
        loginPage.login(username, password);}}
```

Login test

```
public class LoginTest {
    private WebDriver driver;
    @BeforeMethod
    public void setup() {
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("https://www.pandadoc.com/"); }
    @AfterMethod
    public void teardown() {
        driver.close(); }}
```

Login test

```
public class LoginTest {
    private WebDriver driver;
    private void assertSignedUp() {
        new WebDriverWait(driver, 30000).until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".header-avatar"))); }
    @Test
    public void loginTest() {
        HomePage homePage = new HomePage(driver);
        homePage.login();
        assertSignedUp(); }}
```

Implementation is based on

- Reflection
- Annotation
- Templates

HMTL Elements framework from Yandex - info

- Official site of the tool
- Examples
- Source code example
- Book with a brief summary of Selenium WebDriver plus Java bindings (Ru only)

Selenide

Selenide is a wrapper for Selenium Webdriver, oriented, as Geb, on **fast and laconic automation** but using Java. Just focus on your business logic and let Selenide do the rest! Main advantages of that tool:

- Concise fluent API for tests
- True Page Objects

Selenide

1. Webdriver access
2. jQuery style selectors
3. Ajax support for stable tests
4. Auto-start and browser destructions
5. You don't need to think how to shutdown browser, handle timeouts and StaleElement Exceptions or search for relevant log lines, debugging your tests.

Login Page

```
public class LoginPage {
    private static final By USERNAME_INPUT = By.id("id_username");
    private static final By PASSWORD_INPUT = By.id("id_password");
    private static final By LOGIN_BUTTON = By.id("main_action_form_button");

    public void login(String username, String password) {
        $(USERNAME_INPUT).val(username);
        $(PASSWORD_INPUT).val(password);
        $(LOGIN_BUTTON).click();
    }}
```

Login Page

Source code – where …?

Home Page

```
public class HomePage {
    private static final By LOGIN_BUTTON = By.cssSelector("a[href*=\"login\"]");
    public void login() {
        login(Creds.username, Creds.password); }
    private LoginPage clickLoginButton() {
        $(LOGIN_BUTTON).click();
        return page(LoginPage.class);}
    private void login(String username, String password) {
        LoginPage loginPage = clickLoginButton();
        loginPage.login(username, password); }}
```

Home Page

Source code – where …?

Login test

```
public class LoginTest {
    private HomePage homePage;
    private void assertSignedUp() {
        $(By.cssSelector(".header-avatar")).waitUntil(Condition.visible,
30000); }
    @BeforeMethod
    public void setup() {
        Configuration.timeout = 10000;
        homePage = open("https://www.pandadoc.com/",
HomePage.class); }
    @Test
    public void loginTest() {
        homePage.login();
        assertSignedUp(); } }
```

Login test

Source code – where …?

Implementation is based on

- Reflection
- Annotation
- Templates

Selenide - info

- Official site of the tool
- Examples
- Source code example
- Book with a brief summary of Selenium WebDriver plus Java bindings (Ru only)

GDI – from EPAM Systems

JDI Goals

1. Accumulate best practices of UI automation (open source, page objects, logging, parallelism etc.)
2. Multiplatform framework
3. Intuitive test framework
4. Stable framework
5. Flexible framework

GDI features

- Interfaces based elements
- Complex UI Objects
- Simple + Complex UI elements + standard Forms = 31
- Cascade UI Objects Initialization
- Multiplatform
- Logging for all actions
- Stabilization mechanism
- Wide Customization abilities
- Module-based architecture
- Parallel test runs support

PageObject “infrastructure”

UI Objects

- UI elements for Web (Button, Label, Table…)
- IU elements for other platforms (any UI)
- “Standard” PageObjects
  - Form
  - Search
  - Pagination
  - Login
  - others

Page Objects

- UI elements
- Actions with UI elements

Simple UI Elements – 10

1. Text
2. Button/Label
3. Link
4. Image
5. Checkbox
6. TextField
7. DatePicker
8. FileInput
9. TextArea
10. And others :)

Complex UI Elements – 12

- Selector
- RadioButtons
- Dropdown
- Tabs/Menu
- CheckList
- DropList
- ComboBox
- TextList
- Table
- Tree
- Range
- And others :)

Standard Pages - 9

- Page
- Section
- Form
- Pop
- PopupForm
- Search
- Pagination
- NavigationPane

PageObject std example, 1 (GDI)

```
@JPage(url = "http://www.epam.com")
public class JacketPage extends WebPage {
    @FindBy(css=“someSelector") public Menu                menu;
    @FindBy(css="someSelector") public CheckList           sizes;
    @FindBy(css=" someSelector") public Button             searchButton;
    @FindBy(css=" someSelector") public Table              results;
}
```

Test std example, 1 (GDI)

```
@Test
    public void menuTest() {
        menu.select("Man");
        sizes.check("M", "L");

        searchButton.click();

        Assert.areEquals(results.rows().count(), 4);
        Assert.isNotEmpty(results.row("Best Jacket", column("Title")));
    }
```

PageObject std example, 1 (Selenium)

```
public class JacketPage {
    @FindBy(css="") public List<WebElement> menuElements;
    @FindBy(css="") public WebElement menuIsSelected;

    @FindBy(css="") public List<WebElement> sizesElements;
    @FindBy(css="") public WebElement sizesIsChecked;
    @FindBy(css="") public List<WebElement> sizesLabels;

    @FindBy(css="") public WebElement searchButton;

    @FindBy(css="") public List<WebElement> resultsColumnHeaders;
    @FindBy(css="") public List<WebElement> resultsRowsHeaders;
    @FindBy(css="") public List<WebElement>resultsCellsHeaders;
    @FindBy(css="") public List<WebElement> resultsColumn;
    @FindBy(css="") public List<WebElement> resultsRow;
                …
}
```

PageObject std example, 1 (Selenium)

```
public class JacketPage {
    @FindBy(css="") public List<WebElement> menuElements;
            ...
    @FindBy(css="") public List<WebElement> resultsRow;

            public void selectElementFromMenu(String name) { … }
            public String getSelectedMenuItem() { … }
            public boolean isMenuItemSelected(String name) { … }
            public void selectElementFromMenu(String name) { … }
            …
            public int getResultsCount() { … }
            public boolean isResultPresent(String name) { … }
            public WebElement getSomeAttributeForResult(String name) { … }
            …
            public List<WebElement> findAllResultsMatch(String name) { … }
}
```

PageObject std example, 2 (GDI)

```
@JPage(url = "/login", title = "EPAM - Login page")
public class LoginPage extends PageForm {
    @FindBy(css = “.login”)          private ITextField login;
    @FindBy(css = “.password”)    private ITextField password;
    @FindBy(css = “.loginButton”) private IButton loginButton;
}
```

Test std sexample, 2 (GDI)

```
@Test
public void exampleTest(String searchText) {
    loginPage.login(client);
     searchPage.search(searchText);
     resultsPage.results.eachContains(searchText);
     resultsPage.openFirstResult();
     productPage.check.productHasNoEmptyData();
}
```

Test std example, 3 (GDI)

```
@Test
  public void shopTest() {
      loginPage.open();
      loginForm.login(new User(“user", "password"));
      search.find("best jacket");
      productForm.submit(Products.TestJacket);
      pagination.next();
      pagination.selectPage(5);
  }
```

PageObject std example, 4 (GDI, “model” based)

```
public class LoginForm extends Form<User> {
    @FindBy(css=“someLocator")
    public TextField name;
    @FindBy(css=“someLocator")
    public TextField password;
    @FindBy(css=“someLocator")
    public Button loginButton;
}
```

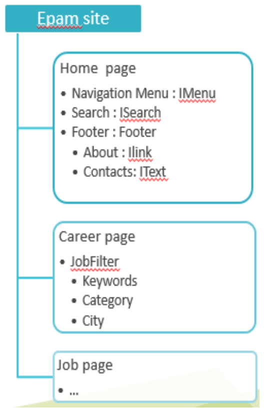
PageObject std example, 4 (GDI, “model” based)

```
public class User {
    public String name;
    public String password;
    public User(String name,
                                          String password) {
        this.name = name;
        this.password = password;
    }
}
```

PageObject std example, 5 (GDI)

```
@JSite(domain = "https://www.epam.com")
public class EpamSite extends WebSite {
        @JPage(url = "/")
        public static HomePage homePage;
        @JPage(url = "/careers", title = "Careers")
        public static CareerPage careerPage;
        @JPage(url = "/careers/job-listings", title = "Job Listings",
                urlCheckType = CONTAIN, titleCheckType = CONTAIN)
        public static JobPage jobPage;
...
}
```

```
WebSite.init(EpamSite.class);
```



“Forms” Page Object (GDI)

```
public class AddCVForm extends Form<Attendee> {
        @FindBy(css = “.first-name“)        private ITextField name;
        @FindBy(css = “.last-name“)         private ITextField lastName;
        @FindBy(css = “.email“)             private ITextField email;
        @FindBy(css = “.file-upload“)       private RFileInput cv;
        @FindBy(css = “.comment-input“)       private ITextArea comment;

        @FindBy(xpath = “//*[.='Submit']“)              private IButton submit;
        @FindBy(xpath = “//*[.='Cancel']“)              private IButton cancel;
}
```

Entities driven testing (GDI)

```
 @Test(dataProvider = “attendees”)
   public void menuTest(Attendee attendee) {
              searchFilter.fill(attendee);
              checkSuggestionsContains(attendee);
              searchFilter.search(attendee);
              assertEquals(results.rows().count(), 1);
              results.row(attendee.name, column(“Name”))
        addCVForm.submit(attendee.cv);
              checkCVInDB(attendee.cv);
              previewForm.verify(attendee);
}
```

Java examples

```
public class HomePage extends WebPage {
    @FindBy(css="div.tabs-ui[data-path*=
       '/content/epam/en/jcr:content/content_container/
       section_4/section-par/tabs']")
    public EpamCoreSection epamCoreSection;
}
```

Java examples

```
public class EpamCoreSection extends Section {
    @FindBy(css="div.tab-1>div.text>div.text-ui
>p>span.font-size-26")
    public Text whoWeAreText;
    @FindBy(css="div.tab-2>div.text>div.text-ui
>p>span.font-size-26")
    public Text whatWeDoText;
    @FindBy(css="div.tab-3>div.text>div.text-ui
>p>span.font-size-26")
    public Text whoWeServeText;
    @FindBy(xpath="//div[.='Who we serve']")
    public Button whoWeServe;
    @FindBy(xpath="//div[.='Who we are']")
    public Button whoWeAre;
    @FindBy(xpath="//div[.='What we do']")
    public Button whatWeDo;
}
```

Java examples

```java
@Test
public void presentationSimpleTest() {
    homePage.checkOpened();
    homePage.epamCoreSection.whoWeAre.highlight();
    homePage.epamCoreSection.whoWeAreText.highlight();
    new Check("Tab text").areEquals("" +
      "We are more than developers.
       We are the experts that
       will take your business into the digital future.",

homePage.epamCoreSection.whoWeAreText.getText());
    homePage.epamCoreSection.highlight();
    homePage.epamCoreSection.whoWeServe.highlight();
    homePage.epamCoreSection.whoWeServe.click();
}
```

Java examples

```
@Test
public void presentationComplexTest() {
    homePage.checkOpened();

textToTest(HomeTabs.WHATWEDO,homePage.epamCoreSection);
textToTest(HomeTabs.WHOWEARE,homePage.epamCoreSection);
textToTest(HomeTabs.WHOWESERVE,homePage.epamCoreSection);
}
```

Java examples

```
public void textToTest(HomeTabs tab, EpamCoreSection
epamCoreSection){
    switch (tab) {
        case WHOWEARE:
            chooseTab(epamCoreSection.whoWeAre,
epamCoreSection.whoWeAreText,
HomeTabsTextToCheck.WHOWEARE);
            break;
        case WHATWEDO:
            chooseTab(epamCoreSection.whatWeDo,
epamCoreSection.whatWeDoText,
HomeTabsTextToCheck.WHATWEDO);
            break;
        case WHOWESERVE:
            chooseTab(epamCoreSection.whoWeServe,
epamCoreSection.whoWeServeText,
HomeTabsTextToCheck.WHOWESERVE);
            break;
    }
}
```

Java examples

```
public void chooseTab(Button tab, Text text, String
textToCheck){
    tab.highlight();
    tab.click();
    text.highlight();
    new Check("Tab text").areEquals(textToCheck,
            text.getText());
}
```

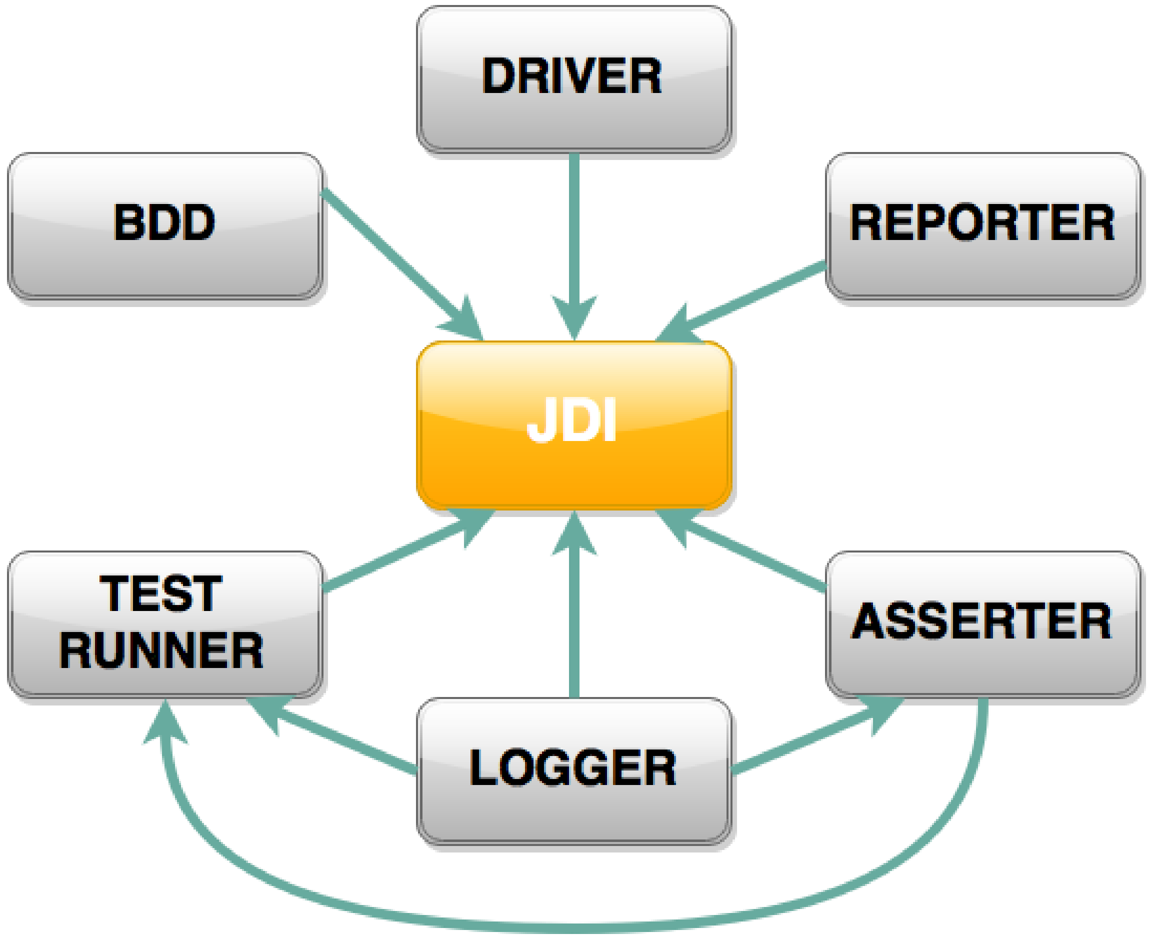
Java examples

```
public static class HomeTabsTextToCheck {
    public static final  String WHOWEARE="We are more than developers." +
        " We are the experts that will take your business into the digital future.";
    public final static String WHATWEDO="We transform businesses through the art of
digitization." + " Our expertise spans multiple disciplines," +
            " providing our clients with software solutions that dramatically drive
results and outcomes.";
    public final static String WHOWESERVE="The right technology translates business
strategies into results." + " We deliver domain-specific, transformative software
solutions that reshape the way you do business.";
}
```

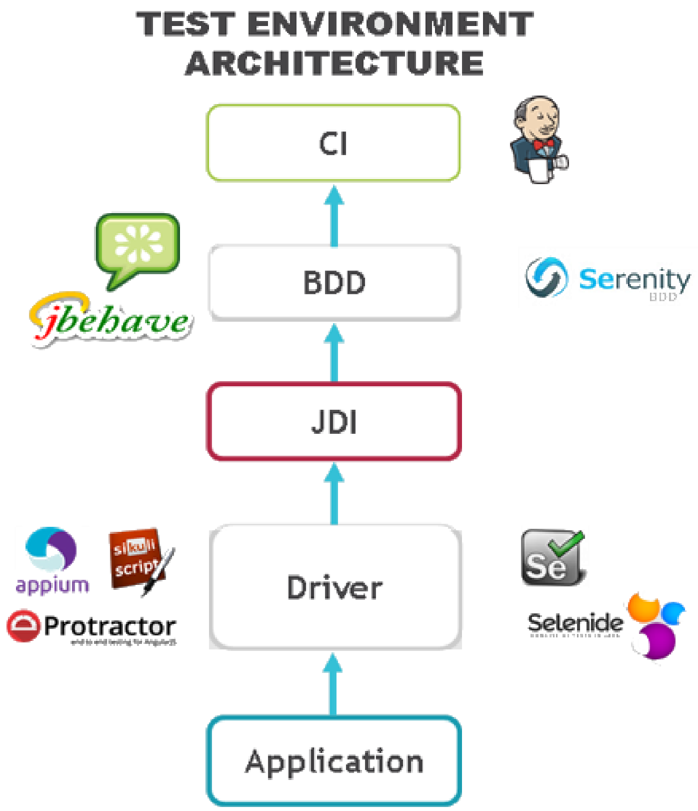
Customization

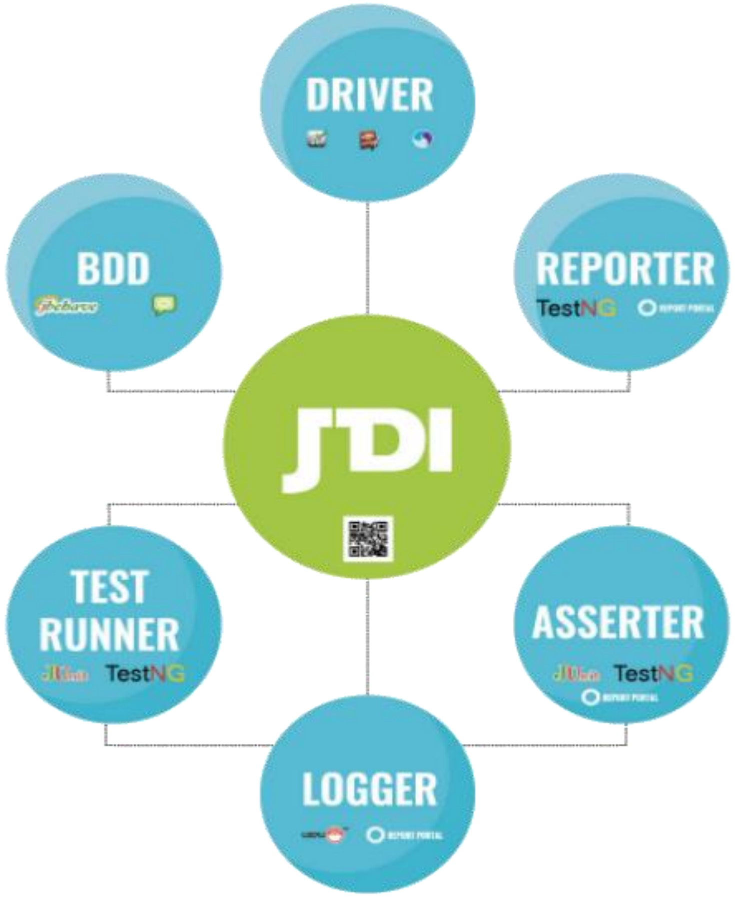
1. Customize method's behavior
2. Customize element's behavior
3. Customize all objects behavior
4. Develop new UI element
5. Modules
   - Logger
   - Asserter
   - Driver
   - etc.

Architecture \ Modules structure



Architecture \ Modules structure





Implementation is based on

Implementation is based on … what?

Let’s:

- download source code
- familiarize
- investigate
- develop a tiny prototype
- try to use
- try to improve (*for example, add Decorator DP based features*)
- **welcome to QA Automation tools development**
- **welcome to Open Source**

JDI - info

- http://jdi.epam.com/
- http://jdi.epam.com/download
- https://github.com/epam/JDI/tree/master/Java/Tests
- https://github.com/epam/JDI/tree/master/Java
- https://github.com/epam/JDI/tree/master/C%23.Net/Tests
- https://github.com/epam/JDI/tree/master/C%23.Net

- IT-Saturday video (Ru only)

- “JDI - EPAM Framework (IT-Subbotnik, with listings).pptx” (RU)
- “JDI - QA Conf 2016.pptx” (Ru)
- “JDI – Secon 2016.pptx” (EN)

Serenity (ex. Thucydides)

1. Serenity (Thucydides) – is an open-source tool, oriented on effective acceptance testing and detailed documentation and reports of the project.

2. It works with Junit and BDD tools like JBehave and Cucumber-JVM, and gives a wide API for automated testing with integrity with Selenium Webdriver.

3. It is made for the following tasks:
   - Creating more flexible tests, which are easier to be supported
   - Getting illustrated story-based reports
   - Clear context of connection between tests and requirements
   - Measurement of requirements coverage.

Serenity (ex. Thucydides) workflow

- Step 1: **Defining requirement and acceptance criteria**

It starts from requirements, which are need to be tested. For every requirement there is an acceptance criteria which describing the requirement better. Thucydides automate acceptance criteria

- Step 2: **Modelling the requirements**

With the help of Thucydides you create a simple model of requirements made on Java language. There are some ways of modelling requirements, including simple Java class, using convention and structure of directories of integrating with other tools like Jira. That approach allows the developer what requirement is tested each of the test, and give Thucydides the ability to track testing features and requirements

Serenity (ex. Thucydides) workflow

- Step 3: **Automation of acceptance testing**

Next is describing acceptance criteria by business language is following, and automation engineers implement them using BDD such as jBehave or Cucumber-JVM, or using Java and Junit, in order for Thucydides to run it with “pending” status (body is not implemented)

- Step 4: **Test implementation**

Automation Engineers now can implement acceptance criteria in a form of real-time AUT test. Tests can be divided into steps for better readability and easiness of support. Webdriver is used for testing.

Serenity (ex. Thucydides) workflow

- Step 5: **Test results report**

Thucydides allow you to create detailed test reports about test runs, including:

History for ach test

Screenshot for each test step

Result of test run including time and error messages

- Step 6: **Requirements coverage report**

Besides test run reports, Thucideds also provide information about:

Number of tested requirements

Number of passed requirements

Number of requirements which need to be run

Serenity (ex. Thucydides) workflow

- Step 7: **Project progress report**

  Thucydides provide information about project history progress:

  Change in number of features implemented in time

  Change in quantity of implemented and tested features in time

  Change of failed tests in time

- **Summary**

As we can see, Thucydides is a rather complex tool, build around BDD concept and acceptance testing, using Webdriver for testing web applications.

- Info

Official website

Official documentation

Page Object

And what about Page Object supports …?

Page Object “default” implementation

Framework implements Page Object pattern and let’s you decrease code duplication another one type of classes between tests and pages, called “steps”. Let’s look at “steps” with page object class:

```
public class StepsinBook extends ScenarioSteps {
    public StepsinBook(Pages pages) {
        super(pages);
    }
    public BookPage getPageBook()
    {
        return getPages().currentPageAt(BookPage.class);
    }
}
```

Page Object “default” implementation

```
public class StepsinBook extends ScenarioSteps {
   @Step public void getMain(String url)
    {
        getPageBook().getMainPage(url);
    }
    @Step public void AllBooks()
    {
        getPageBook().allBooks();
    }
    @Step public void search(){
        getPageBook().search(“Book");
    }
    @Step public void catalog(){
        getPageBook().catalog();
    }}
```

Page Object “default” implementation

```
public class BookPage extends PageObject {
    @FindBy(linkText = “All books")
    WebElement allbooksButton;

    @FindBy(linkText = “Search")
    WebElement searchButton;

    @FindBy(name = "query")
    WebElement searchField;

    @FindBy(css = "button")
    WebElement searchBegin;
}
```

Page Object “default” implementation

```
public class BookPage extends PageObject {
    public BookPage(WebDriver driver) {
        super(driver);
    }
    public void getMainPage(String url) {
        getDriver().get(url);
    }
    public void allBooks() {
         allbooksButton.click();
    }
    public void search(String searchWord) {
        searchButton.click();
        searchField.sendKeys(searchWord);
        searchBegin.click();
    }
}
```

Implementation is based on

- Reflection
- Annotation
- Templates

Geb – general info

1. Tool for browser automation, made on Groovy (JVM-based) and based on Selenium Webdriver.

1. It using:
   1. Selenium for browser automation
   2. jQuery selectors for locating elements
   3. Page object pattern
   .
2. In the bounds of testing it can be easily integrated with different test frameworks like:
   1. Junit
   2. TestNG
   3. Spock

Geb (pronounced "jeb")
very groovy browser automation... web testing, screen scraping and more

Geb – general info

If to compare with Webdriver API, Geb provides more comfortable interface in the following areas:

- Working with Webdriver instance (creating, configuration, moves, destruction)
- Locating elements (jQuery locators)
- Page object pattern
- Waiters
- Page interactions
- Work with AJAX elements
- Integration with build-tools (maven, gradle, grails)
- Integration with cloud services (Sauce labs, Browser Stack)

Official website
Official documentation

Geb - example

```
import geb.Browser
 Browser.drive {
    go "http://myapp.com/login"
    assert $("h1").text() == "Please Login"
    $("form.login").with {
        username = "admin"
        password = "password"
        login().click()
    }
    assert $("h1").text() == "Admin Section"
}
```

SWD Page Recorder

1. SWD Page Recorder helps you to create locators of web-page, debug them in the application and generate PageObject code for classes on C#, Java, Python, Ruby, Perl for using them in Selenium Webdriver tests.

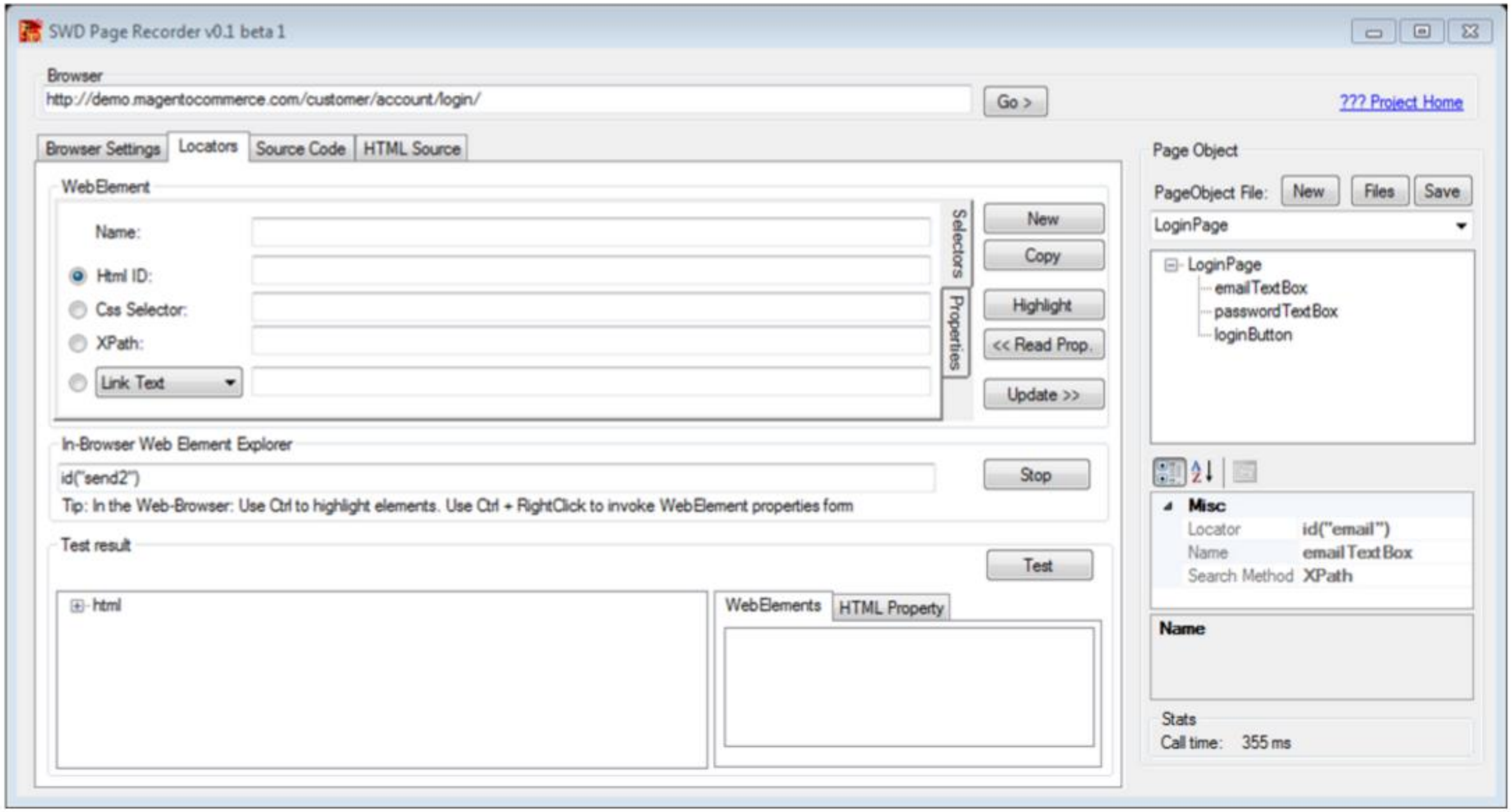
2. That tool not only lets you find locator needed, but also optimize it and generate all the code needed for further element declaration in the code.

3. SWD Page Recorder is a unified tool for working with locators in every browser which are supported by Selenium.

4. Important moment that Page Recorder tests the selectors using Webdriver – so you’re not going to have cases when locator find other way will not work in Webdriver.

SWD Page Recorder

[Sources, documentation and so on]



How to select an appropriate solution?

A real-life example

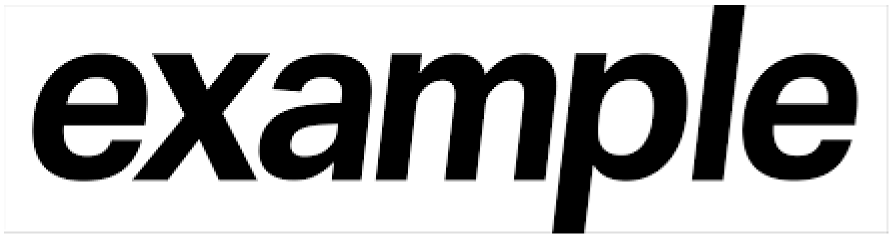
1. Business context
2. Tech context
   - Challenge
   - Solution
   - Technology Stack
3. QA Automation process context
4. Source code example
5. Summary

example

Example

1. Example and “home work”

   – How to setting up environment
      - This is task 0 ☺
   – Where to download (https://github.com/comaqaby/Patterns/tree/master/C%23)
   – How to build
   – How to configure
   – How to run


example

“Homework”, part 1 ☺

1. Download example’ source code (for each “solution”)
2. “Investigate”
3. Develop a TODO list with a set of improvements
4. Setting up environment
5. Compile and Run
6. “Play”
7. Improve \ Update TODO list with a set of improvements

“Homework”, part 2 ☺

1. Read recommended books and articles
2. Improve \ Update TODO list with a set of improvements
3. Provide me via email with an intermediate and final list of improvements
4. Develop at least 1 more auto-test (for each “solution”)
5. Develop a set of metrics to “compare” solutions
6. Develop an algorithm “How to select a proper solution”
7. Provide me via email with a set of metrics + algorithm
8. Next iteration ☺

“Homework”, part 3 ☺

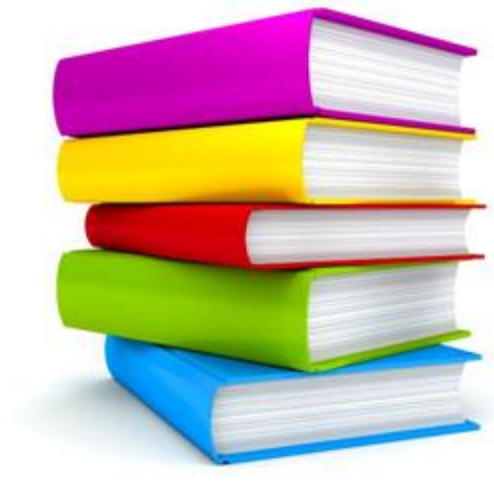
1. Play with GDI “attentively”
2. Join QA Automation tools development
3. Join Open Source “world”

“Rules” and principles

1. Could you please “refresh” your theoretical knowledge, slides **1-31**, thanks

Useful links

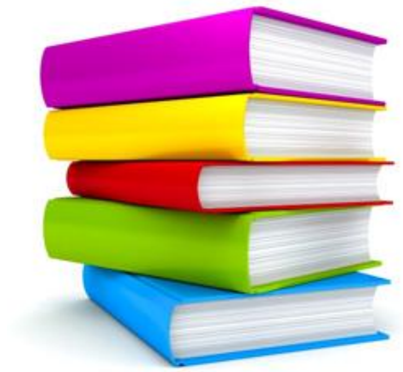
1. Martin Fowler “Refactoring”
   - http://martinfowler.com/books/refactoring.html
   - http://refactoring.com/
   - http://refactoring.com/catalog/
   - http://refactoring.com/catalog/replaceConditionalWithPolymorphism.html



Useful links

1. Refactoring to Patterns and vice versa

   – https://industriallogic.com/xp/refactoring/
   – https://industriallogic.com/xp/refactoring/catalog.html
   – https://industriallogic.com/xp/refactoring/conditionDispatcherWithCommand.html
   – https://industriallogic.com/xp/refactoring/conditionalWithStrategy.html
   – http://martinfowler.com/books/r2p.html



Useful links

1. https://sourcemaking.com/
   - https://sourcemaking.com/refactoring
   - https://sourcemaking.com/refactoring/replace-conditional-with-polymorphism
   - https://sourcemaking.com/design_patterns
   - https://sourcemaking.com/antipatterns
     - https://sourcemaking.com/antipatterns/software-development-antipatterns
     - https://sourcemaking.com/antipatterns/software-architecture-antipatterns
   - https://sourcemaking.com/uml

Tech “basement”

- Grady **Butch «Object oriented analysis and design with examples of apps on C++**

  Notes: IMHO No need to be afraid of C++, 95% of the material is conceptual, there is no strict connection to chosen language. From my point of view is one of the best books for true getting to know with OOP.

- Martin **Fowler «Refactoring»**

  Notes: IMHO strongly recommend to read from cover to cover, twice, in order to have contents of the book – you active professional luggage.

- **Gang of four “Design patterns”**

  Notes: IMHO strongly recommend to read from cover to cover, twice, in order to have contents of the book – you active professional luggage.

Tech “basement”

- D. **Thomas**, Andrew **Hunt** “**Pragmatic Programmer**, The: From Journeyman to Master”

Notes: IMHO Amazing book that consists of a ton of advices. IMHO strongly recommend to read from cover to cover, twice, in order to have contents of the book – you active professional luggage. And then look through different chapters before talking to a customer.

- Steve **McConnel** “**Code complete**”

Notes: IMHO No need to be afraid of the size of the book ... it should be read or before “going to bed”, or from any place, of separate chapters, just to fresh things in the memory in the chosen f ield of problem.

What's next?!

- "Out of box Page Object Design Pattern, Java"
    - Dynamic solutions
    - Let's compare with a static one
- "Out of box Page Object Design Pattern, .Net C#"
    - Dynamic solutions
    - Let's compare with a static one
- "Variants of implementation Page Object Design Pattern from the scratches, without being bind to any programming language"
    - Static solutions
    - Dynamic solutions
    - Let's compare static and dynamic
    - Answer to all our questions

Anton_Semenchenko@epam.com

Skype - semenchenko_anton_v

+375 33 33 46 120
+375 44 74 00 385

https://www.linkedin.com/in/anton-semenchenko-612a926b
https://www.facebook.com/semenchenko.anton.v
https://twitter.com/comaqa

www.comaqa.by
www.corehard.by