# **Java теория**

**3 пастулата ООП:**

1) Инкапсуляция - внутреннее состояние объектов должно быть защищено от возможности изменения извне и должно производится с помощью методов самого объекта

2) Наследование - один класс может наследовать(“extends”) другой класс. Таким образом можно переиспользовать код с класса, от которого наследуются.

3) Полиморфизм - переиспользование кода наследниками (перегрузка и переопределение методов). Как говорится, один интерфейс — множество реализаций. При помощи полиморфизма

можно объединять и использовать разные типы объектов по их общему поведению.

4) Абстракция - абстрагируемся от ненужных полей и методов, создаём только то, что нам необходимо

**Методы класса Object:**

getClass()

hashCode()

equals(Object obj)

clone()

toString()

notify()

notifyAll()

wait(long timeout)

wait(long timeout, int nanos)

wait()

finalize()

**Abstract class vs interface**

1) У абстрактных классов есть конструктор, а у интерфесов нет

2) абстрактный класс может наследоваться от класса, а интерфейс может расширять только другой интерфейс;

3) абстрактный класс может наследоваться только от одного класса, интерфейс может расширять любое количество интерфейсов;

4) абстрактный класс может иметь final, non-final, static, non-shatic переменные, а интерфейс может

иметь только static final variables;

5) абстрактный класс может реализовать интерфейс, а интерфейс не может реализовать абстрактный класс;

6) в абстрактном классе ключевое слово abstract является обязательным для объявления метода как абстрактного, в

интерфейсе abstract необязательно использовать.

**Bubble sort in Java**

public static int[] bubbleSort(int[] array) {

for (int i = 0; i < array.length - 1; i++) {

for (int j = 0; j < array.length - i - 1; j++) {

/\* if a pair of adjacent elements has the wrong order it swaps them \*/

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

return array;

}

# **CI/CD**

**continious integration**

практика разработки программного обеспечения, которая заключается в постоянном слиянии веток в master (до нескольких раз в день) и

выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

**continious delivery**

подход к разработке ПО, при котором ПО производится короткими итерациями, гарантируя, что ПО является стабильным и может быть

передано в эксплуатацию в любое время, а передача его не происходит вручную. Целью является сборка, тестирование и релиз программного обеспечения с большей скоростью и

частотой.

**continious deployment**

Процесс развертывания проверенных фич из промежуточного окружения в промышленное, где они будут готовы к релизу. Тот же Docker создан для неприрывного развёртывания.

DevOps инженеры могут обновлять контейнеры и разворачивать их сразу на продакшене в автоматическом режиме. Такой процесс является ключом к непрерывной доставке, т.к. весь

процесс может занять всего лишь несколько минут.

# **SOLID**

## **S** - SRP, Single Responsibility Principle (Принцип единственной ответственности)

Если класс делает вычисления, что-то куда-то отправляет, выводит, описывает логику логирования – это нарушение SRP и антипаттерн. Такой объект можно назвать «божественный объект» или God object. Класс CalculateDecorationArea содержит код, которым пользуются эти две группы акторов. К каким проблемам это может привести? Например, маляры поймут, что в программе заложен слишком большой коэффициент. Допустим, они захотят его поменять, чтобы при закупке краски не оставалось излишков. Но что если для кафельной плитки такие коэффициенты вполне подходят? Получается, с одной стороны у нас есть маляры, а с другой - укладчики плитки, и обе эти группы пользуются одним и тем же кодом для расчета площади. Если код поменяется с учетом новых требований от маляров, плитки будет закуплено слишком мало. Соответственно, в приложении появится дефект с точки зрения плиточников. Лучше в этом случае разделить вычисления и создать два разных класса.

## **O** - OCP, Open/ Closed Principle (Принцип открытости/ закрытости)

OCP помогает выстроить иерархию компонентов в коде. Наиболее высокоуровневые компоненты и классы должны быть максимально защищены. Это проявляется в том, что они ничего не знают о классах более низкого уровня (не имеют зависимостей на них), а, значит, никак не реагируют на изменения в них. Таким образом, при соблюдении принципа открытости/ закрытости запрос на создание новой функциональности в приложении приведет к тому, что мы будем добавлять новые классы или модули и оставим неизменной логику существующей системы. А это в свою очередь сократит или сведет к нулю появление регрессионных багов.

## **L** - LSP, Liskov Substitution Principle (Принцип подстановки Барбары Лисков)

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом. Подкласс не должен требовать от вызывающего кода больше, чем базовый класс. Подкласс не должен предоставлять вызывающему коду меньше, чем базовый класс.

## **I** - ISP, Interface Segregation Principle (Принцип разделения интерфейсов)

Здесь предполагается разбиение обширных интерфейсов с большим количеством методов на более мелкие, объединяющие методы в соответствии с бизнес-логикой. Проблема «толстых» интерфейсов заключается в том, что классы, которые их имплементируют, вынуждены так или иначе оверрайдить все методы, даже те, которые не имеют в рамках этих классов никакой имплементации. И если такие методы меняются в интерфейсе, программа требует повторной компиляции в языках типа Java. В идеале интерфейсы должны быть настолько гранулярными, чтобы классы, которые их имплементируют, реализовывали также и все методы из этих интерфейсов.

## **D** - DIP, Dependency Inversion Principle (Принцип инверсии зависимостей)

Модули верхних уровней не должны импортировать сущности из модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций. Таким образом, части приложения должны быть достаточно обособленными, самостоятельными и независимыми. Достичь этого проще всего через создание контракта или интерфейса для каждой части. Если говорить просто, DIP - это про прокладывание абстракций между модулями и оперирование абстрактными классами вместо конкретных имплементаций.

# **XPath**

**Критерии хорошего Xpath:**

1) не более 15 мин на написание

2) должен быть читабельным

3) должен быть надёжным и существовать как можно дольше

/ - спускаемся на один уровень ниже по DOM дереву

// - спускаемся на несколько уровней ниже по DOM дереву

/.. - поднимаемся на уровень выше по DOM дереву

**Поиск по тексту значения атрибута:**

//\*[@class='select-results']//li[contains(.,'Nepal')]

**Поиск любого тега с любым атрибутом**

//\*[@\*=’name’]

**Поиск по названию поля, которое отображено на странице(на случай если id динамические)**

//td[contains(.,"название поля")//option

. - означает поиск по содержанию, контексту

**Поиск по нескольким атрибутам одновременно**

//a[@class="button" and contains(.,"Date")] and - означает логический оператор "и" (существуют ещё or, not, =, !=, >, <, >=, <=)

**Поиск по нескольким атрибутам(один из атрибутов находиться выше по DOM дереву)**

//div[@class='sticker sale']/../../..//i[@class='fa fa-search']

**Получение всех предков, находящихся выше по DOM дереву**

//div[@class='sticker sale']/ancestor::\*

**Получение конкретного предка, находящегося выше по DOM дереву**

//div[@class='sticker sale']/ancestor::li[contains(@class,'product')]//i[@class='fa fa-search']

**Получение последнего вложенного элемента в указанном тэге**

//div[last()]

//div[last() - 1] – предпоследний элемент

**Получение чётных или нечётных элементов**

//tr[position() mod 2 = 0] - чётные элементы

//tr[position() mod 2 != 0] - нечётные элементы

**child::** — содержит множество элементов-потомков (элементов, расположенных на один уровень ниже). Это название сокращается полностью, то есть его можно вовсе опускать.

**descendant::** — содержит полное множество элементов-потомков выбранного элемента(то есть, как ближайших элементов-потомков, так и всех их элементов-потомков).

Выражение /descendant::node()/ можно сокращать до //.

**descendant-or-self::** — содержит полное множество элементов-потомков и текущий элемент. С помощью этой оси, например, можно вторым шагом организовать отбор элементов с

любого узла, а не только с корневого: достаточно первым шагом взять всех потомков корневого. Например, путь //span отберёт все узлы span документа, независимо от их

положения в иерархии, взглянув как на имя корневого, так и на имена всех его дочерних элементов, на всю глубину их вложенности.

**ancestor::** — содержит множество элементов-предков.

**ancestor-or-self::** — содержит множество элементов-предков и текущий элемент.

**parent::** — содержит элемент-предок на один уровень назад. Это обращение можно заменить на ..

**self::** — содержит текущий элемент. Это обращение можно заменить на .

**following::** — содержит множество элементов, расположенных ниже текущего элемента по дереву (на всех уровнях и слоях), исключая собственных потомков.

**following-sibling::** — содержит множество братских элементов того же уровня, следующих за текущим слоем.

**preceding::** — содержит множество элементов, расположенных выше текущего элемента по дереву (на всех уровнях и слоях), исключая множество собственных предков.

**preceding-sibling::** — содержит множество братских элементов того же уровня, предшествующих текущему слою.

**attribute::** — содержит множество атрибутов текущего элемента. Это обращение можно заменить на символ @

**namespace::** — содержит множество элементов, относящихся к тому или иному пространству имён (то есть присутствует атрибут xmlns).

**Функции:**

**node-set node()** Возвращает сам узел. Вместо этой функции часто используют заменитель \*, но, в отличие от звёздочки, функция node() возвращает и текстовые узлы

**string text()** Возвращает узел, если он текстовый

**node-set current()** Возвращает множество из одного элемента, который является текущим. Если мы делаем обработку множества с предикатами, то единственным способом

дотянуться из этого предиката до текущего элемента будет данная функция

**number position()** Возвращает позицию элемента в множестве элементов оси. Корректно работает только в цикле <xsl:for-each/>

**number last()** Возвращает номер последнего элемента в множестве элементов оси. Корректно работает только в цикле <xsl:for-each/>

**number count(node-set)** Возвращает количество элементов в node-set.

**string name(node-set?)** Возвращает полное имя первого тега в множестве

**string namespace-url(node-set?)** Возвращает ссылку на URL, определяющий пространство имён

**string local-name(node-set?)** Возвращает имя первого тега в множестве, без пространства имён

**node-set id(object)** Находит элемент с уникальным идентификатором

**string string(object?)** Возвращает текстовое содержимое элемента. По сути, возвращает объединённое множество текстовых элементов на один уровень ниже

**string concat(string, string, string\*)** Соединяет строки, указанные в аргументах

**number string-length(string?)** Возвращает длину строки

**boolean contains(string, string)** Возвращает true, если первая строка содержит вторую, иначе — false

**string substring(string, number, number?)** Возвращает строку, вырезанную из строки, начиная с указанного номера, и, если указан второй номер, — количество символов

**string substring-before(string, string)** Если найдена вторая строка в первой, возвращает строку до первого вхождения второй строки

**string substring-after(string, string)** Если найдена вторая строка в первой, возвращает строку после первого вхождения второй строки

**boolean starts-with(string, string)** Возвращает true, если вторая строка входит в начало первой, иначе — false

**boolean ends-with(string, string)** Возвращает true, если вторая строка входит в конец первой, иначе — false

**string normalize-space(string?)** Убирает лишние и повторные пробелы, а также управляющие символы, заменяя их пробелами.

Примеры:

//\*[contains(normalize-space(text()), 'Click for JS Confirm')]

string translate(string, string, string) Заменяет символы первой строки, которые встречаются во второй строке, на соответствующие позиции символам из второй строки

символы из третьей строки. Например, translate("bar", "abc", "ABC") вернёт BAr.

# **CSS селекторы**

Удобно искать в консоли dev tools браузера $$("...")

$$("body") поиск по тегу

$$("#div") поиск по id

$$(".class\_name") поиск по class

$$("[role='main']") поиск по атрибуту

$$("h3:nth-child(2)") второй элемент с тегом h3

$$("[role\*='ai']") поиск по атрибуту в названии которого присутствует 'ai' (как contains)

$$("[role^='ma']") поиск по атрибуту название которого начинается с 'ma'

$$("[role$='in']") поиск по атрибуту название которого заканчивается на 'in'

$$("div[role]") все div теги с атрибутом role

$$("class\_name\_1.class\_name\_2.class\_name\_3") поиск элемента у которого есть 3 указанных класса

$$("div:not(.class\_name)") поиск элемента с тегом div, в котором нет класса class\_name

$$("a:not(href^='http')")

$$("div[role='main'] div.class\_name") поиск вложенного div с классом class\_name внутри div c атрибутом role

$$("div[role='main'] > div.class\_name") поиск непосредственного наследника (как child в xpath)

# **Selenium**

## **WebDriverExceptions**

WebDriverException расширяет RuntimeException

ElementNotVisibleException — возникновение этого исключения означает, что хоть элемент и присутствует в DOM, он невидим и поэтому взаимодействовать с ним невозможно.

IllegalLocatorException — возникает когда By не может обработать переданные ему аргументы.

StaleElementReferenceException — возникает когда ссылка на элемент, к которому идет обращение, больше не действительна.

MoveTargetOutOfBoundsException — вызывается, если в метод moveToElement(WebElement toElement) передан элемент, который находится за пределами экрана, а так же в некоторых

случаях когда драйвер не может попасть по движущемуся элементу (например, когда страница находится в процессе скролинга (jquery) — стоит дождаться его завершения и только

после этого обращаться к элементам)

SessionNotCreatedException — это исключение означает, что сессия не может быть создана. Генерируется при использовании ‘requiredCapabilities’ для удаленного запуска, если

браузер с «требуемыми» свойствами не найден.

TimeoutException — вызывается, когда команда не завершается в достаточный промежуток времени.

UnhandledAlertException — возникает при появлении неожиданных модальных диалоговых окон.

UnsupportedCommandException — исключение означает, что команда, используемая удаленным вебдрайвером, не поддерживается.

InvalidCookieDomainException — вызывается при попытке добавить cookie для домена, отличающегося от текущего.

UnableToSetCookieException — генерируется когда драйвер не может по каким либо причинам установить cookie.

Исключения группы «Not Found»

NoAlertPresentException — генерируется при попытке передать управление диалоговому окну, которого в данный момент нет.

NoSuchElementException — вызывается методом findElement(By by), если элемент с заданным селектором не найден на странице.

InvalidSelectorException — исключение генерируется методом findElement(By by) когда заданный селектор поиска не возвращает WebElement.

NoSuchFrameException — возникает во время работы методов WebDriver.switchTo().frame(int frameIndex) и WebDriver.switchTo().frame(String frameName), если фрейм с заданным параметром не найден.

NoSuchWindowException — возникает при отсутствии окна с заданным именем (WebDriver.switchTo().window(String windowName)).

## **Implicit wait**

chromeDriver.manage().timeouts().pageLoadTimeout(50, TimeUnit.SECONDS); // Ожидание загрузки всей страницы

chromeDriver.manage().timeouts().setScriptTimeout(50, TimeUnit.SECONDS); // Ожидание загрузки скриптов JavaScript

chromeDriver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS); // Ожидание конкретного элемента

chromeDriver.manage().timeouts().implicitlyWait(Duration.ofSeconds(40)); // Для новых версий selenium

## **Explicit wait**

WebDriverWait wait = new WebDriverWait(driver, 10)

wait.until(ExpectedConditions.presenceOfElementLocated(By.id("bar")));

ExpectedConditions.attributeContains() – ждём элемент, c указанным атрибутом, который содержит указанный текст

ExpectedConditions.attributeToBe() - ждём элемент, c указанным атрибутом и его значением

ExpectedConditions.elementToBeSelected()

ExpectedConditions.elementToBeClickable()

ExpectedConditions.invisibilityOf – убедится, что элемент невидим

ExpectedConditions.invisibilityOfAllElements - убедится, что все элементы невидимы

ExpectedConditions.invisibilityOfElementLocated - убедится, что элемент невидим или отсутствует в DOM

Для visibility те же методы

not(ExpectedConditions. …) отрицание

ExpectedConditions.numberOfElementsToBe – ждём появления указанного количества элементов

ExpectedConditions.numberOfElementsToBeLessThen

ExpectedConditions.numberOfElementsToBeMoreThen

ExpectedConditions.presenceOfAllElementsLocatedBy -ждём пока элементы отобразятся на странице

ExpectedConditions.presenceOfElementLocated – ждём пока элемент отобразится в DOM

ExpectedConditions.titleContains

ExpectedConditions.titleIs

ExpectedConditions.urlContains

ExpectedConditions.urlMatches

ExpectedConditions.urlIs

## **Fluent Wait (кастомное ожидание)**

Wait<WebDriver> wait = new FluentWait(driver)

.withTimeout(Duration.ofSeconds(11))

.pollingEvery(Duration.ofMillis(500))

.ignoring(NoSuchElementException.class)

WebElement el = wait.until(new Function<WebDriver, WebElement>) {

@Override

Public WebElement apply(WebDriver driver) {

Return driver.findElement(By.xpath(“xpath”))

}

}

## **Click**

Можно использовать метод click() или метод submit(), если у элемента в DOM type='submit'

## **Работа с текстовыми полями**

sendKeys("текст")

sendKeys("текст", Key.ENTER)

clear() - очистить текстовое поле

getAttribute("value") - вернёт текст из поля

## **Радиокнопки и чек боксы**

isSelected() к элементу input вернёт true или false

## **Выпадающие списки**

public void checkThatSelectedElementHasSymbols(){

Select select = new Select(driver.findElement(By.id("dropdown")));

WebElement option = select.getFirstSelectedOption();

assertEquals("Please select an option", option.getText());

}

public void checkSelectByValue(){

Select select = new Select(driver.findElement(By.id("dropdown")));

select.selectByValue("1");

WebElement option = select.getFirstSelectedOption();

assertEquals("Option 1", option.getText());

}

public void checkSelectByVisibleText(){

Select select = new Select(driver.findElement(By.id("dropdown")));

select.selectByVisibleText("Option 1");

WebElement option = select.getFirstSelectedOption();

assertEquals("Option 1", option.getText());

}

public void checkSelectByIndex(){

Select select = new Select(driver.findElement(By.id("dropdown")));

select.selectByIndex(2);

WebElement option = select.getFirstSelectedOption();

assertEquals("Option 2", option.getText());

}

## **Actions**

Actions actions = new Actions(driver);

Для выполнения сложных действий, лучше отдельным методом перейти к элементу и сделать build().perform() и только затем выполнять задуманное действие над элементом

action.moveToElement(slider).build().perform();

action.clickAndHold().moveToElement(sliderId).release().build().perform();

actions.moveToElement(titleA).build().perform();

actions.clickAndHold();

actions.moveToElement(titleC).build().perform();

actions.release().build().perform();

Action myAction = actions.moveToElement(titleA).clickAndHold().moveToElement(titleC).release().build();

myAction.perform;

actions.doubleClick()

actions.contextClick()

## **Java script**

### **Алерты**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("alert('Hello world');");

js.executeAsyncScript("alert('Hello world');"); // лучше использовать асинхронный т.к. велика вероятность, что не дождёмся выполнения при синхронном выполнении скрипта

### **Закрыть алерт**

public void mockConfirm() {

executeJavaScript(

"window.confirm = function() {return true;};"

);

}

js.executeScript("arguments[0].scrollIntoView();", webElement);

js.executeScript("arguments[0].click();", webElement);

js.executeScript("window.scrollBy(0, 1000)");

js.executeScript("window.scrollBy(0, -1000)");

js.executeScript("window.scrollTo(0, document.body.scrollHeight)"); // проскролить вниз страницы

js.executeScript("window.scrollTo(0, -document.body.scrollHeight)"); // проскролить вверх страницы

### **Открыть url**

js.executeScript("window.open('https://the-internet.herokuapp.com/windows/new');");

### **Прокрутка страницы**

Прокрутка вниз страницы  
*executeJavaScript*("window.scrollTo(0, document.body.scrollHeight);");

### **Изменение атрибута**

/\*  
 \* JS by xpath selector:  
 document.evaluate("//html",document,null,XPathResult.FIRST\_ORDERED\_NODE\_TYPE,null).singleNodeValue.style.overflow='auto';  
 \* JS by css selector: document.querySelector('html').style.overflow = 'auto';  
 \*/  
public static void setElementProperty(String xpath, String prop, String val) {  
 String js = "var iterator = document.evaluate(\"" + xpath + "\", document, null, XPathResult.UNORDERED\_NODE\_ITERATOR\_TYPE, null);\n" +  
 "try {\n" +  
 " for (var thisNode = iterator.iterateNext(); thisNode;) thisNode.style.setProperty('" + prop + "', '" + val + "', 'important'), thisNode = iterator.iterateNext()\n" +  
 "} catch (e) {}";  
 try {  
 *executeJavaScript*(js);  
 } catch (Exception ignored) {  
 }  
}

### **Установка даты в data picker**

void setDateByName(String name, String date) {

executeJavaScript(

String.format("$('[name=\"%s\"]').val('%s')",

name, date)

);

}

**Закрыть data picker**  
  
executeJavaScript(

"$('.datepicker').hide();"

);

## **Работа с Alert**

driver.switchTo().alert().accept(); - нажать ок в alert

driver.switchTo().alert().dismiss(); - нажать нет в alert

wait.until(ExpectedConditions.*alertIsPresent*()).dismiss();

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("confirm('Hello world');"); - создаём алерт с помощью js

## **Работа с окнами и вкладками браузера**

String mainWindow = driver.getWindowHandle();

for (String windowHandle : driver.getWindowHandles()) {

driver.switchTo().window(windowHandle);

}

## **Работа с frame и iFrame**

Чтобы переключиться на frame, который находится внутри другого frame, необходимо сначала переключится на главный, а затем на вложенный. Чтобы вернуться к основной странице

или к другим frame необходимо переключится на вкладку браузера, а затем на другой frame. С iFrame ситуация аналогичная

chromeDriver.findElement(By.xpath("//a[text()='Nested Frames']")).click();

chromeDriver.switchTo().frame("frame-top");

chromeDriver.switchTo().frame("frame-left");

Assertions.assertEquals(chromeDriver.findElement(By.xpath(".//html/body")).getText(), "LEFT", "frame LEFT не выбран");

Set<String> handles = chromeDriver.getWindowHandles();

for (String handle : handles) {

chromeDriver.switchTo().window(handle);

}

chromeDriver.switchTo().frame("frame-top");

chromeDriver.switchTo().frame("frame-middle");

Assertions.assertEquals(chromeDriver.findElement(By.xpath(".//html/body")).getText(), "MIDDLE", "frame MIDDLE не выбран");

for (String handle : handles) {

chromeDriver.switchTo().window(handle);

}

chromeDriver.switchTo().frame("frame-top");

chromeDriver.switchTo().frame("frame-right");

Assertions.assertEquals(chromeDriver.findElement(By.xpath(".//html/body")).getText(), "RIGHT", "frame RIGHT не выбран");

## **Проверка доступности и наличия элементов**

webElement.isEnabled(); - вернёт true, если, например, на кнопку можно нажать

webElement.isSelected(); - вернёт true, если, например, radio button или check box активирован

webElement.isDisplayed():

• Возвращает TRUE, если элемент существует в DOM И отображается.

• Возвращает FALSE, если элемент существует в DOM и скрыт.

• Выдает исключение, если элемент не существует в DOM

webElement.isPresent(): возвращает TRUE, если элемент существует в DOM иначе возвращает false

List<WebElement> list = driver.findElements(By.xpath(//......)).size(); - можем проверить наличие элемента на странице, если размер листа равен нулю, то элемента нет

## **Имитация нажатия клавиш на клавиатуре**

webElement.sendKeys(Keys.ENTER);

webElement.sendKeys(Keys.chord(Keys.SHIFT, "test"); - пишем капсом

String select = Keys.chord(Keys.CONTROL, "a"); сочетание клавиш для выделения всего текста

String cut = Keys.chord(Keys.CONTROL, "x"); сочетание клавиш для вырезания текста

String paste = Keys.chord(Keys.CONTROL, "v"); сочтание клавиш для вставки текста

webElement.sendKeys("test");

webElement.sendKeys(select); применяем созданное сочетание клавиш

## **Загрузка файлов**

Необходимо найти элемент input с type='file' и применить к нему метод sendKeys()

webElement.sendKeys("C:\\test\\picture.jpg");

## **Скачивание файлов**

String link = chromeDriver.findElement(By.xpath("//a[text()='200x200.png']")).getAttribute("href");

//Set file to save

File fileToSave = new File("C:\\Users\\drkuznetsov\\IdeaProjects\\bi\_telecom\\200x200.png");

//Download file using default org.apache.http client

CloseableHttpClient httpClient = HttpClients.createDefault();

HttpGet httpGet = new HttpGet(link);

HttpResponse response = null;

try {

response = httpClient.execute(httpGet, new BasicHttpContext());

} catch (IOException e) {

e.printStackTrace();

}

//Save file on disk

try {

copyInputStreamToFile(response.getEntity().getContent(), fileToSave);

} catch (IOException e) {

e.printStackTrace();

}

## **Создание скриншота**

Date date = new Date();

SimpleDateFormat format = new SimpleDateFormat("hh-mm-ss");

String fileName = format.format(date) + ".png";

File screen = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

try {

FileUtils.copyFile(screen, new File("C:\\Screenshots\\" + fileName));

} catch (IOException e) {

e.printStackTrace();

}

## **Shadow DOM**

Библиотека для работы с Shadow Dom для Selenium: https://github.com/sukgu/shadow-automation-selenium

Иногда в DOM дереве нельзя посмотреть вложенные атрибуты у атрибута т.к. они скрыты.

Для просмотра скрытых атрибутов можно обратиться к отображаемому атрибуту

driver.findElement(By.xpath("visibleAttr")).getAttribute("innerHTML")

## **Cookies**

Set<Cookie> cookies = driver.menage().getCookies(); // получаем куки  
String cookie = (String) js.executeScript(“return document.cookie”);  
  
Cookie cookie = driver.menage().getCookieNamed(“auth”); // получаем конкретные куки  
String val = cookie.getValue();  
  
driver.manage().addCookie(new Cookie(“name”, “value”)); // добавить куки  
  
driver.manage().deleteCookie(Cookie cookie);  
driver.manage().deleteAllCookies();

## **Навигация (вперёд, назад)**

driver.navigate().back();  
driver.navigate().forward();  
driver.navigate().refresh();

# **Selenide**

## **Отключение всплывающих окон браузера**

ChromeOptions options = new ChromeOptions();

options.addArguments("--disable-notifications");

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(ChromeOptions.CAPABILITY, options);

Configuration.browserCapabilities = capabilities;

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("autoAcceptAlerts", true); // для принятия всплывающих уведомлений

capabilities.setCapability("autoGrantPermissions", true); // для выдачи прав

## **Click**

click(x, y) , где x - смещение влево(-20) или вправо(20), y - смещение вверх или вниз

Если одна кнопка или элемент частично закрывают нужную нам, то можно использовать настройку Configuration.clickViaJs = true , которая позволяет с помощью JS нажать на

нужный элемент(даже если он полностью перекрыт другим элементом)

Начиная с версии 5.15 можно использовать click(ClickOptions.usingJavaScript()) эффект будет тот же

или click(ClickOptions.usingDefaultMethod().offset(-20, 0)) чтобы использовать стандартный click() со смещением

$("#page").click(usingJavaScript());

$("#page").click(usingJavaScript().offset(123, 222));

$("#page").click(usingJavaScript().offsetY(222));

$("#page").click(usingDefaultMethod());

## **Проверка доступности и наличия элементов**

isDisplayed() //возвращает false, если элемент либо невидимый, либо его нет в DOM

exists() //возвращает true, если элемент есть в DOM, иначе - false

## **Скачивание файлов**

В методе download() можно использовать параметры:

download(6000) - таймаут в милисекундах

download(6000, FileFilters.withName("имя файла")) - таймаут и имя файла, который хотим скачать

download(using(PROXY).withTimeout(6000)withFilter(FileFilters.withName("имя файла")))

download(withExtension("pdf"))

1) Если нет атрибута href c ссылкой на файл (не работает для safari)

добавить зависимость browserup-proxy-core v 2.1.1 (если версия selenide 5.12.0 т.к. в более ранних версиях библиотека с прокси встроена)

Сonfiguration.proxyEnabled=true;

Configuration.fileDownload=FileDownloadMode.PROXY;

$("#external-download").download();

2) Если файл формируется с помощью JS, то selenide v 5.13.0 может справиться и с этим (не работает для safari)

добавить зависимость browserup-proxy-core v 2.1.1 (если версия selenide 5.12.0 т.к. в более ранних версиях библиотека с прокси встроена)

Сonfiguration.proxyEnabled=true;

Configuration.fileDownload=FileDownloadMode.FOLDER; //Отличия тут!

$("#external-download").download();

3) Скачивание через Proxy при использовании selenoid

добавить зависимость browserup-proxy-core v 2.1.1 (если версия selenide 5.12.0 т.к. в более ранних версиях библиотека с прокси встроена)

Configuration.remote="адрес selenoid"

MutableCapabilities caps = new MutableCapabilities();

caps.setCapability("enableVNC", true);

Configuration.browserCapabilities = caps;

Configuration.proxyHost="адрес хоста где гонятся тесты"

Сonfiguration.proxyEnabled=true;

Configuration.fileDownload=FileDownloadMode.PROXY;

$("#external-download").download();

## **Создание скриншота**

screenshot("file path")

String base64 = screenshot(OutputType.BASE64);

byte[] bytes = screenshot(OutputType.BYTES);

String base64\_coverted = Base64.getEncoder().encodeToString(bytes);

Assertions.assertEquals(base64, base64\_coverted);

Assertions.assertEquals(base64.length(), base64\_coverted.length());

## **BaseClass**

public abstract class BaseTest {  
  
 @BeforeTest  
 public void setUp() {  
 Configuration.*pageLoadTimeout* = 20000;  
 Configuration.*timeout* = 15000;  
 Configuration.*browser* = "chrome";  
 Configuration.*browserSize* = "1920x1080";  
// Configuration.fastSetValue = true; //оптимизация мeтода setValue(), если перестанет работать, то можно  
 // использовать sendKeys()  
 Configuration.*headless* = false; // при true - браузер не будет отображаться при запуске тестов  
  
 SelenideLogger.*addListener*("AllureSelenide", new AllureSelenide().screenshots(true).savePageSource(false));  
  
 ChromeOptions options = new ChromeOptions();  
 options.addArguments("--disable-extensions");  
  
 DesiredCapabilities capabilities = new DesiredCapabilities();  
 capabilities.setCapability(ChromeOptions.*CAPABILITY*, options);  
 Configuration.*browserCapabilities* = capabilities;  
 }  
  
 @AfterTest  
 public void tearDown() {  
 *closeWebDriver*();  
 }  
  
}

## **Soft Assertions**

Configuration.assertionMode = AssertionMode.SOFT; // в методе с тестом или в @BeforeEach

## **Одновременная работа в нескольких браузерах**



## **Получение логов из консоли разработчика**

getWebDriver().manage().logs().get(LogType.BROWSER).getAll()

## **Scroll**

$x(“…”)scrollIntoView(0,0); // скролл к верхней части страницы

executeJavaScript("window.scrollTo(0,document.body.scrollHeight)") // скролл в нижнюю часть страницы

public static void scrollUntilElementExists(SelenideElement element) {

while (!element.exists()) {

executeJavaScript("window.scrollTo(0,document.body.scrollHeight);");

}

}

## **Изменить разрешение экрана с помощью JS**

public static void setResolution(String res) {  
 String[] dimension = res.split("x");  
 int width = Integer.*parseInt*(dimension[0]);  
 int height = Integer.*parseInt*(dimension[1]);  
 *executeJavaScript*("window.resizeTo(" + width + "," + height + ")");  
}

## **Получить логи браузера**

private static String getBrowserLogs() {  
 StringBuilder result = new StringBuilder();  
 for (String logEntry : *getWebDriverLogs*(LogType.*BROWSER*, Level.*ALL*)) {  
 result.append(logEntry).append("\\").append(System.*lineSeparator*());  
 }  
 return result.toString();  
}

## **SetWebDriver и WebDriverProvider**

Вариант 1:

Configuration.browser = MyWebdriverProvider.class.getName();

Этот вариант предпочтительнее в большинстве случаев. Он хорош тем, что ваш класс MyWebdriverProvider отвечает только за то, КАК открыть браузер (какие ключики ему передать, где найти бинарник, вот это всё). Но не отвечает за то, КОГДА открыть и закрыть браузер. Об этом заботится селенид.

Вариант 2:

@Before

public void setUp() {

var yandexBrowser = new YandexDriver(...);

WebDriverRunner.setWebDriver(yandexBrowser);

}

Этот вариант хуже тем, что вам придётся самому не только открывать, но и закрывать браузер. Это часто вызывает недопонимание. Селенид не может его закрыть, ведь он не знает, где ещё вы можете его использовать. Сам открыл - сам закрывай.

## **Shadow DOM**

$(shadowCss("#anyButton", "#shadow-host")).click();

$(shadowCss("p", "#shadow-host")).shouldHave(text("Inside Shadow-DOM"));

$(shadowCss("p", "#shadow-host", "#inner-shadow-host")).shouldHave(text("The Shadow-DOM inside another shadow tree"));

## **LocalStorage и SessionStorage**

// Очистить всё содержимое:

localStorage().clear();

// Добавить значение:

localStorage().setItem("username", "john");

// Проверить значение:

assertThat(localStorage().getItem("username")).isEqualTo("john");

// Проверить количество элементов:

assertThat(localStorage().size()).isEqualTo(1);

// удалить значение:

localStorage().removeItem("username");

assertThat(localStorage().getItem("username")).isNull();

localStorage().shouldHave(item("cat”));

localStorage().shouldHave(itemWithValue("mouse", "Jerry”));

Кстати, заодно появился метод localStorage().getItems(), возвращающий всё содержимое в виде мапы.

Аналогичные методы и для sessionStorage

## **Cookie**

WebDriverRunner.getWebDriver().manage().addCookie(new Cookie("name", "value”));

## **Проверки для URL**

String url1 = WebDriverRunner.url();

String url2 = WebDriverRunner.currentFrameUrl();

webdriver().shouldHave(url("https://auth.google.com"));

webdriver().shouldHave(url("https://mastercard.ee"), Duration.ofSeconds(42));

webdriver().shouldNotHave(url("http://yandex.ru");

webdriver().shouldNotHave(urlStartingWith("ftp://"));

webdriver().shouldHave(currentFrameUrl(baseUrl + "/login.html"));

webdriver().shouldHave(currentFrameUrlStartingWith(baseUrl + "/logout.html"));

## **Открыть новую, пустую вкладку браузера**

Selenide.switchTo().newWindow(WindowType.TAB);

## **Работа с pdf файлами**

File file = $("#report").download(); // нужна зависимость `com.codeborne:selenide`

PDF pdf = new PDF(file); // нужна зависимость `com.codeborne:pdf-test`

assertThat(pdf).containsText("Hello");

## **Получение запросов и ответов в браузере через прокси**

Как вы знаете, в селениде уже есть встроенный прокси-сервер, надо его всего лишь включить:

**Configuration.**proxyEnabled **=** **true;**

И ещё нужно сказать прокси-серверу, чтобы он начал отслеживать запросы:

**BrowserMobProxy** bmp **=** **WebDriverRunner.**getSelenideProxy**().**getProxy**();**

*// запоминать тело запросов (по умолчанию тело не запоминается, ибо может быть большим)*

bmp**.**setHarCaptureTypes**(CaptureType.**getAllContentCaptureTypes**());**

*// запоминать как запросы, так и ответы*

bmp**.**enableHarCaptureTypes**(CaptureType.**REQUEST\_CONTENT**,** **CaptureType.**RESPONSE\_CONTENT**);**

*// начинай запись!*

bmp**.**newHar**(**"pofig"**);**

Теперь нужно получить HAR и анализировать все записи в нём:

**List<HarEntry>** requests **=** bmp**.**getHar**().**getLog**().**getEntries**();**

HAR (HTTP Archive) - это типа такой “архив” со всеми сетевыми запросами и ответами (“entries”).

Каждая запись - это и есть запрос от тестируемого приложения к серверу.  
Внутри есть все данные: URL, request, response, их http status и body.

## **Clipboard(буфер)**

Clipboard clipboard = Selenide.clipboard();

String foo = clipboard().getText();

clipboard().setText("bar");

clipboard().shouldHave(content("Hello fast World"));

clipboard().shouldHave(content("Hello slow World"), Duration.ofMillis(1500));

# **Junit**

@BeforeClass или @BeforeAll - выполняет указанный метод до выполнения всех методов

@AfterClass или @BeforeAll - выполняет указанный метод после выполнения всех методов

@Before или @BeforeEach - выполняет указанный метод перед всеми методами

@Test - указывает на тестовый метод

@Ignore - игнорирует указанный тест

@Disabled("text") - отключает тест

@Displayname

@RepeatedTest(name = "Repeated test", value = 5) при использовании не надо ставить @Test т.к. количество прогонов будет больше на 1

**Параметризированные тесты:**

[Туториал по JUnit 5 - Аннотация @ParameterizedTest / Хабр (habr.com)](https://habr.com/ru/post/591007/)

boolean[] booleans The boolean values to use as sources of arguments; must not be empty.

byte[] bytes The byte values to use as sources of arguments; must not be empty.

char[] chars The char values to use as sources of arguments; must not be empty.

Class<?>[] classes The Class values to use as sources of arguments; must not be empty.

double[] doubles The double values to use as sources of arguments; must not be empty.

float[] floats The float values to use as sources of arguments; must not be empty.

int[] ints The int values to use as sources of arguments; must not be empty.

long[] longs The long values to use as sources of arguments; must not be empty.

short[] shorts The short values to use as sources of arguments; must not be empty.

String[] strings The String values to use as sources of arguments; must not be empty.

@ParameterizedTest

@ValueSource(strings = {"a", "b", "c"})

void test(String arg) {

assertTrue(arg.length() > 0);

}

@ParameterizedTest

@CsvSource(values = {"A,a", "B,b", "C,c"})

void test(String arg, String low) {

assertEquals(low, arg.toLowerCase());

}

@ParameterizedTest(name = "Строка в верхнем регистре {0}, строка в нижнем регистре {2}")

@CsvFileSource(resources = "myfile.csv") // лучше исп notepad++ для создания файла (шаблон val1,val2)

void test(String arg, String low) {

assertEquals(low, arg.toLowerCase());

}

--------------------------------------------------

public enum Pet {

CAT,

DOG,

TIGER,

}

--------------------------------------------------

class EnumSourceDemo {

@DisplayName("Passing enum values")

@ParameterizedTest(name = "{index} => pet = ''{0}''")

@EnumSource(Pet.class) // Используем все значения из enum

void testEnumValues(Pet pet) {

assertNotNull(pet);

}

@DisplayName("Passing enum values")

@ParameterizedTest(name = "{index} => pet = ''{0}''")

@EnumSource(value = Pet.class, names = {"CAT", "DOG"}) // Используем выборочные значения

void testEnumValues(Pet pet) {

assertNotNull(pet);

}

}

@ParameterizedTest

@MethodSource("argsProviderFactory")

**void** **testWithMethodSource**(String argument) {

assertNotNull(argument);

}

**static** Stream<String> **argsProviderFactory**() {

**return** Stream.of("alex", "brian");

}

--------------------------------------------------

**Запуск выборочных тестов с помощью @Tag**

--------------------------------------------------

package intermediate.lecture\_04

class TagTest {

@Test

@DisplayName("CaseA: prod & dev")

@Tag("production")

@Tag("development")

void testCaseA(TestInfo testInfo) {}

}

--------------------------------------------------

package intermediate.lecture\_04

@RunWith(umPlatform.class)

@SelectPackages("intermediate.lecture\_04")

@IcludeTags("production")

class ProductionTest {}

--------------------------------------------------

package intermediate.lecture\_04

@RunWith(JUnitPlatform.class)

@SelectPackages("intermediate.lecture\_04")

@IcludeTags("development")

class ProductionTest {}

**Запуск тестов в определённом порядке:**

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class Test {

@Test

@Order(3)

void test1() {}

@Test

@Order(1)

void test2() {}

@Test

@Order(2)

void test3() {}

}

## **Ассерты в Junit**

Assertions.assertTrue(1 + 1 == 2, "error");

Assertions.assertFalse(1 + 1 == 10, "error");

Assertions.null();

Assertions.notNull();

Assertions.equals(10, 5 + 5);

Assertions.notEquals(12, 5 + 5);

Assertions.assertIterableEquals(коллекция\_1, коллекция\_2);

Assertions.assertAll(

() -> assertThat("www.google.com", endsWith("com")),

() -> assertThat("12345", isA(String.class)),

() -> assertThat("They differ!", "10", is("10"))

);

// Soft проверка ассертов, т.е. в любом случае будут проверены все ассерты и тест не упадёт на первом проваленом ассерте

@ExtendWith(SoftAssertsExtension.class)

public class tests() {

@Test

void test() {

Configuration.assertionMode = AssertionMode.SOFT;

open("url")

$$("locator").shouldHave(Condition);

$$("locator").shouldHave(Condition);

$$("locator").shouldHave(Condition);

}

}

# **TestNG**

@BeforeTest

@AfterTest

@AfterClass

@BeforeClass

@BeforeSuite

@AfterSuite

@BeforeGroups

@AfterGroups

@BeforeMethod

@AfterMethod

@Test

@Test(description = "описание теста")

@Test(priority = 1) //чем меньше цифра, тем выше приоритет

@Test(testName = “имя теста”) //имя, которое будет отображаться в результатах

**Скачать плагин для idea "Create TestNG XML" затем нажать правой кнопкой на проекте и выбрать "Create testNG XML"**

**Запуск тестов по классам или пакетам:**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="All Test Suite">

<test name="Test1">

<classes>

<class name="smoke.Test1"></class>

</classes>

</test>

<test name="Test2">

<classes>

<class name="smoke.Test2"></class>

</classes>

</test>

<test name="Smoke tests">

<packages>

<package name="smoke"></package>

</packages>

</test>

</suite>

**Запуск тестов по группам:**

@Test(groups = {"SmokeTest", "Regress"})

public void test1() {

System.out.println("Test 1");

}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="All Test Suite">

<test name="Test1">

<groups>

<run>

<include name="SmokeTest"/>

</run>

</groups>

<classes>

<class name="smoke.Test1"></class>

<class name="smoke.Test2"></class>

</classes>

</test>

</suite>

**Параметризированные тесты:**

@Parameters({"first"})

@Test(groups = {"SmokeTest", "Regress"})

public void test1(String parameter) {

System.out.println("Test 1: " + parameter);

}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="All Test Suite">

<test name="Test1">

<parameter name="first" value="Dan"></parameter>

<classes>

<class name="smoke.Test1"></class>

</classes>

</test>

</suite>

**Data provider:**

@DataProvider(name = "db")

public static Object[][] dbData() {

return new Object[][] {

{"SQL", new Integer(1)},

{"NOSQL", new Integer(2)}

};

}

@Test(dataProvider = "db")

public void test1111(String name, Integer value) {

System.out.println("Test 1111: " + name + " with value = " + value);

}

@DataProvider(name="users")  
public static Object[][] users() {  
 return new Object[][] {  
 {"Банковская карта", *user*(*m1941*)},  
 {"СБЕР PAY", *user*(*m1942*)},  
 {"Эвотор счёт", *user*(*m1943*)}  
 };  
}  
  
@Test(description = "Покупка и продление подписки на приложение по скидке МСП", priority = 1, dataProvider = "users")  
public void test\_1(String paymentMethod, User user) {}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="All Test Suite">

<test name="Test1">

<classes>

<class name="smoke.Test1"></class>

</classes>

</test>

</suite>

# **Allure**

@Epic("...") - название для группы фич

@Feature(value = "Test for task 6") - название фичи

@Story(value = "Test for login with valid credentials")

@Severity("...") - степень влияния фичи

@Description("...") - описание теста

@Step(value = "Fill in login with {имя аргумента метода}")

@Step(value = "Fill in login with {0} {1} {2}") - {0} - порядковый номер аргумента

allure serve allure-results – запуск отчёта из командной строки

allure.attach.file(file\_path, name=name, extension="docx")

@Attachment(value = "Page source что там ", type = "text/plain")

public static byte[] pageSource() {

return getWebDriver().getPageSource().getBytes(StandardCharsets.UTF\_8);

}

# **Git**

1) Настраиваем имя в папке с проектом

git config --global user.name "Daniil Kuznetsov"

2) Указываем email

git config --global user.email "ldan93@yandex.ru"

3) Создаём репозиторий

git init

4) Контроль состояния текущего репозитория

git status

5) Указываем гиту файл за изменениями которого он должен следить

git add имя\_файла.расширение

6) Указываем все файлы в папке для отслеживания гитом

git add \* //добавляет все файлы кроме игнорируемых

git add . //дополнительно добавляет файл .gitignore

7) Удаляем файл из отслеживаемой группы

git rm --cashed имя\_файла.расширение

git rm -f имя\_файла.расширение //удалем из папки проекта тоже

8) Создать файл gitignore

touch .gitignore

9) Делаем коммит

git commit -m "сообщение"

git commit --amend - добавить последние изменения к последнему комиту не создавая новый комит

10) После изменения файла в папке необходимо его снова добавить в отслеживаемые гитом и только потом сделать коммит

git add .

git add имя\_файла.расширение

11) Информация о сделанных коммитах

git log

12) Посмотреть изменения в файлах по сравнению с последним коммитом

git diff

13) Связывает локальный репозиторий с удалённым

git remote add origin https://gitlab.com/atcourse/1.git

14) Список удалённых репозиториев с которыми связан текущий локальный репозиторий

git remote

15) Отправить локальный репозиторий на удалённый

git push -u имя\_репозитория имя\_ветки

git push -u origin master

16) Создание дополнительной ветки

git branch название\_ветки

17) Переключение на другую ветку

git checkout название\_ветки

Ветка пустая, в ней также нужно сделать git add

18) Соединение веток

1 Переключаемся на мастер ветку

git checkout master

2 Делаем слияние

git merge имя\_ветки\_которую\_хотим\_присоединить\_к\_master

19) Загрузка удалённого репозитория

git clone url\_репозитория

git remote -v show покажет список репозиториев;

git remote remove <имя> удалит указанный репозиторий(отвяжет локальный от удалённого).

Git pull загружает данные с сервера, и только самые последние. Таким образом, вы не запутаетесь в коммитах и будете уверены, что скопировали только самые свежие и

актуальные из них. Итак, вы просто делаете: git pull origin <branch\_name>, т. Е. вы загружаете последние изменения из ветки удаленного репозитория на сервере. В примере

мы загружаем данные из основной ветки.

$ git pull origin main

20) Удаление последнего(их) комитов

git reset --hard HEAD~n , где n - количество последних комитов с изменениями, которые хотим удалить

git reset --soft HEAD~n , где n - количество последних комитов, которые хотим удалить (изменения при этом останутся)

Если изменения запушены на удалённый репозиторий:

удаляем изменения локально и затем

git revert hash\_комита а затем пушим и в итоге из комита удаляются наши изменения не затрагивая другой код

# **Rest Assured**

@Data

@Builder

@Builder.Default

@JsonIgnoreProperties(ignoreUnknown = true)

@AllArgsConstructor

@NoArgsConstructor

@Accessors(chain = true)

@JsonProperty(value = "isUserActivePlan")

@Getter

@JsonFormat(pattern = "yyyy-MM-dd hh:mm:ss", timezone = "Europe/Moscow")

@JsonNaming(PropertyNamingStrategy.SnakeCaseStrategy.class)

## **Достать JSON из файла в виде Map**

ObjectMapper mapper = new ObjectMapper();

Map<?, ?> map = mapper.readValue(Paths.get("nameOfJson.json").toFile(), Map.class);

## **Сравнение с JSON схемой**

.body(matchesJsonSchemaInClasspath("json/orders/getOrders.json"));