

Selenium WebDriver

Selenium WebDriver

- **Selenium WebDriver** – программная библиотека, позволяющая имитировать действия пользователя в браузере
- <https://www.seleniumhq.org/projects/webdriver/>
- По сравнению с Selenium IDE мы получаем следующие возможности:
 - Можно работать с разными браузерами
 - Можно пользоваться всеми конструкциями, библиотеками и методологиями, применяемыми в программировании
 - Это дает больше возможностей, а также возможность не дублировать общую логику

Selenium WebDriver

- **Selenium WebDriver** доступен для разных языков программирования:
 - Java
 - C#
 - Python
 - Менее популярны: PHP, Ruby, Perl, JavaScript
- В идеале лучше выбирать язык, на котором разработан сам проект – тогда можно привлекать на помощь разработчиков
- Но если вы не знаете этот язык, или его долго изучать, можете выбрать язык, который знаете

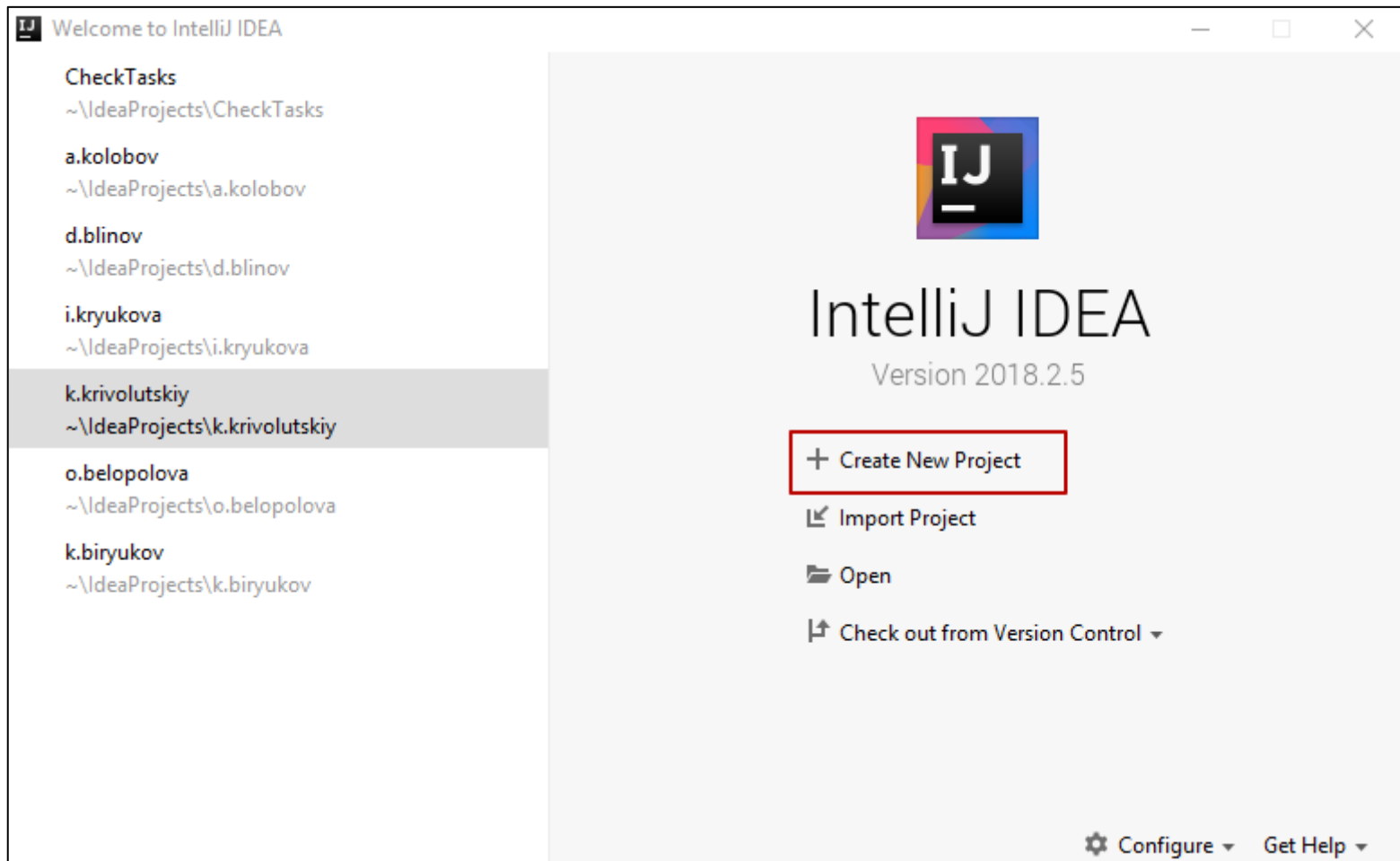
Использование Selenium WebDriver в Java

Установка среды разработки и JDK

- В качестве среды разработки будем использовать **IntelliJ IDEA**
- Сначала скачайте и установите **JDK**:
- <https://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Для 64-битных систем ставьте последнюю версию, для 32-битных скачивайте JDK 1.8 x86 версию
- Скачивайте **Community** версию IDEA:
- <https://www.jetbrains.com/idea/download/>

Создание проекта

- При первом старте жмем **Create New Project**

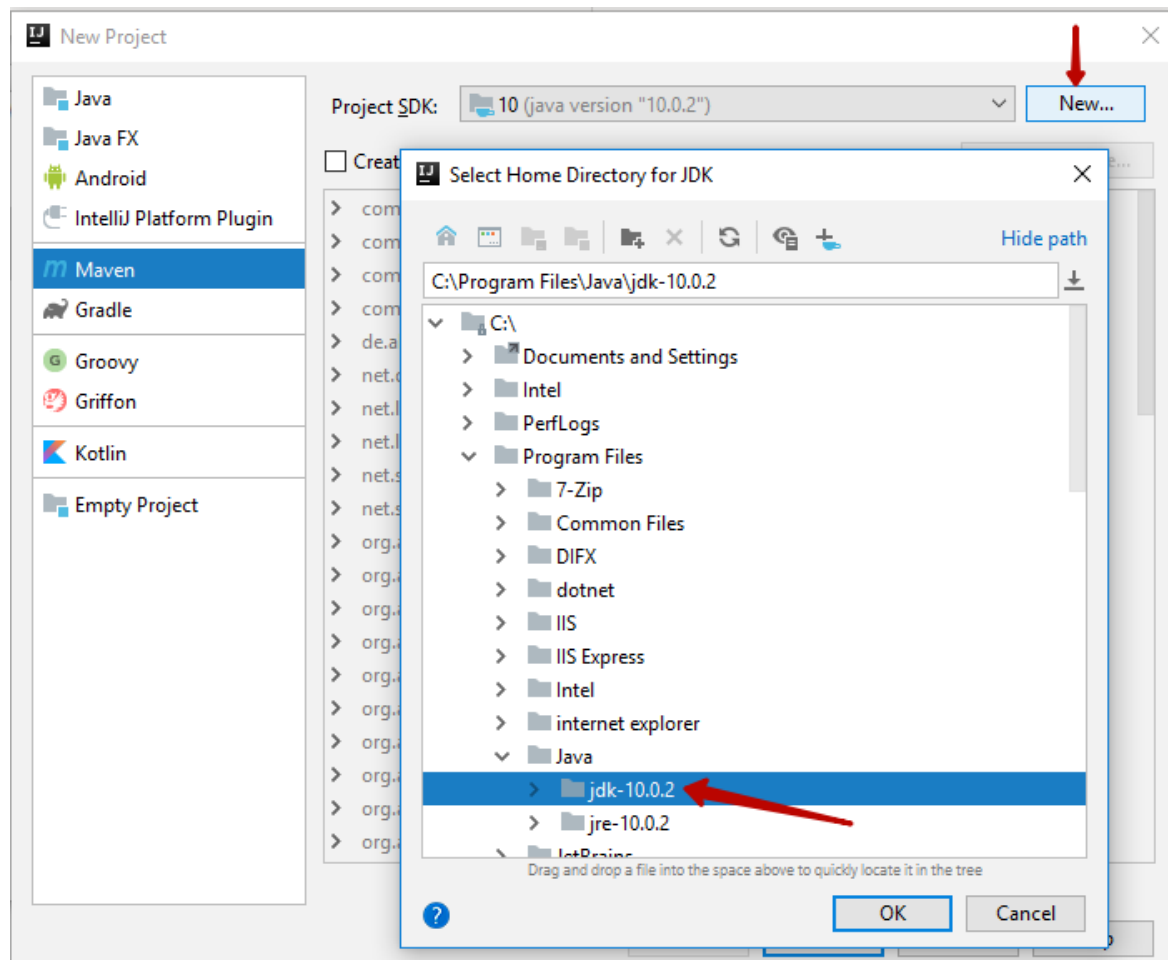


Создание проекта

- При создании первого проекта надо указать путь к JDK
- Если в списке **Project SDK** у вас ничего не выбрано, то нажмите **New...** и укажите путь к папке JDK
- Обычно путь выглядит так:
C:\Program Files\Java\jdk-<номер версии>

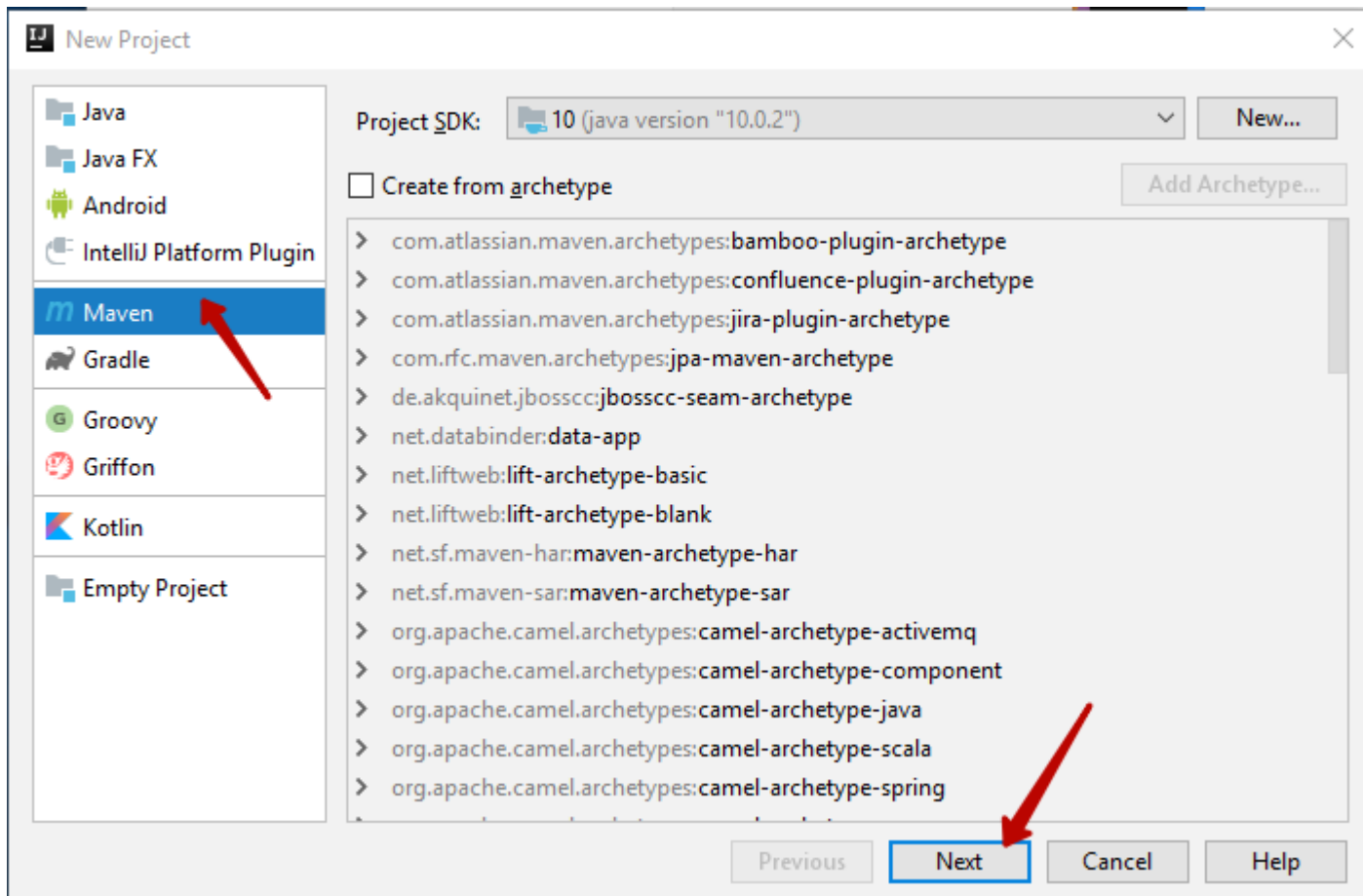
Создание проекта

- Обычно путь выглядит так:
C:\Program Files\Java\jdk-<номер версии>



Создание проекта

- Выбираем **Maven** проект, жмем **Next**

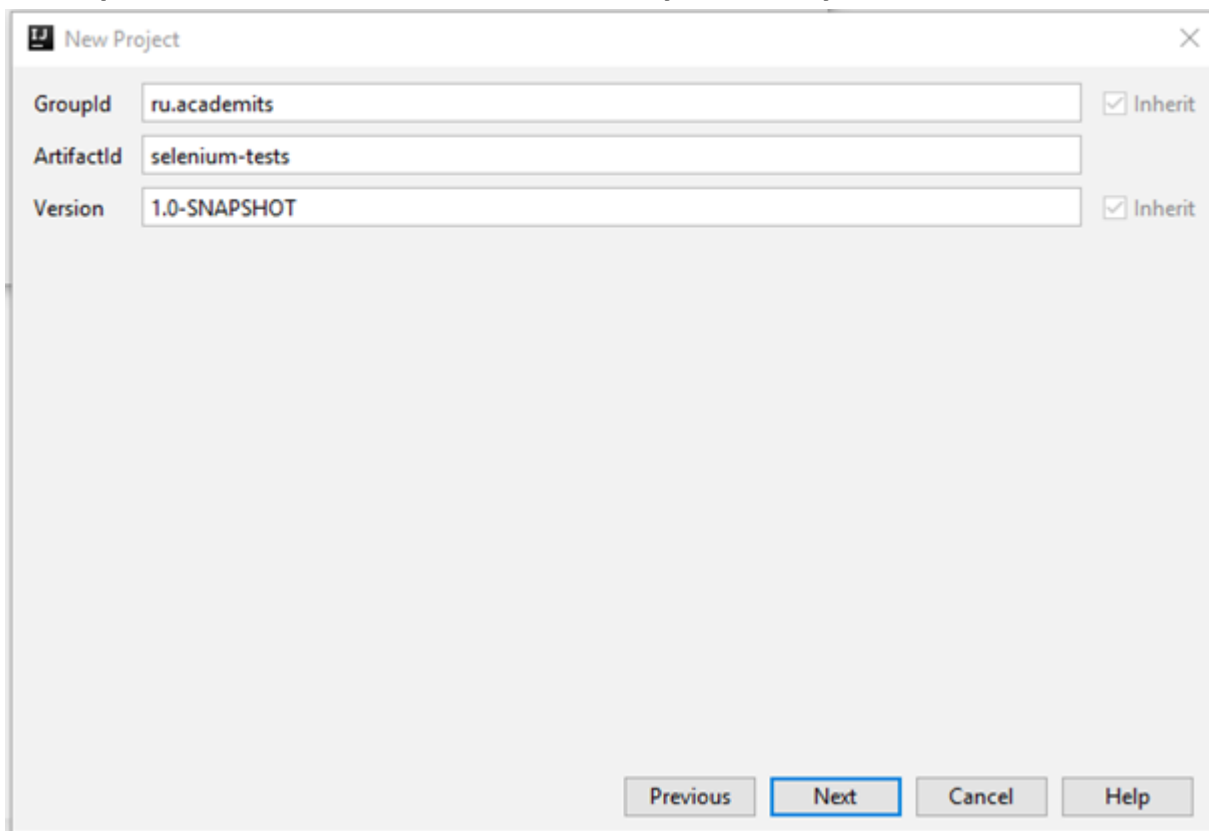


Maven

- **Maven** – это менеджер пакетов для Java проектов
- Он позволит нам не скачивать вручную библиотеку **Selenium WebDriver** (и другие библиотеки тоже)
- Кроме того, Maven накладывает структуру на проект – в каждом проекте предполагается папка **main** для кода и папка **test** для тестов
- Мы будем работать с папкой **main/java** для создание вспомогательных классов (page object'ов и др.) и с папкой **test/java** для размещения самих тестов
- Также **Maven** является средством сборки проекта – он может компилировать проект, автоматически выполнять тесты и создавать дополнительные артефакты

Создание проекта

- **GroupId** – это имя пакета в Java проекте. Обычно это адрес сайта компании наоборот. Можно заполнить, например, ru.academits
- **ArtifactId** – это имя проекта для Maven. Пусть будет selenium-tests
- Затем жмем **Next** и **Finish**



New Project

GroupId ru.academits ☒ Inherit

ArtifactId selenium-tests ☒ Inherit

Version 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help

Добавление зависимостей

- Чтобы скачать и использовать библиотеку **WebDriver**, надо указать **Maven**, что наш код зависит от этой библиотеки
- Для настройки Maven проекта используется файл **pom.xml**
- В этом файле надо создать элемент **dependencies**
- Внутри этого элемента надо добавлять все зависимости в виде элементов **dependency**

Добавление зависимостей

- Для установки WebDriver нам нужно добавить внутрь `dependencies` элемент `dependency` с указанием правильных данных для библиотеки
- В итоге получится так:
- `<dependencies>`

`<dependency>`

`<groupId>org.seleniumhq.selenium</groupId>`

`<artifactId>selenium-java</artifactId>`

`<version>3.14.0</version>`

`</dependency>`

`</dependencies>`

Добавление зависимостей

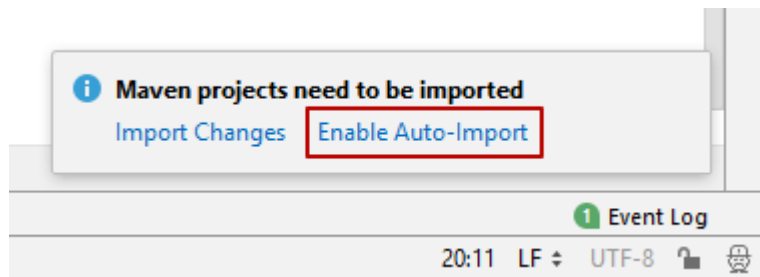
- Какие зависимости нам понадобятся:
- **selenium-java** – Selenium WebDriver для Java
- **junit-jupiter** – тестовый фреймворк Junit 5
- **webdrivermanager** – библиотека для управления драйверами браузером WebDriver Manager
- **selenide** – готовый тестовый фреймворк, разработанный (<https://selenide.org/>)

Как узнать что писать в dependency?

- Чтобы узнать эти данные самим, нужно найти библиотеку на сайте **Maven**
- <https://mvnrepository.com/>
- В поиске ищите **selenium-java**
- Заходите в selenium-java, там заходите в нужную версию (обычно нужна последняя стабильная версия)
- И там уже будет нужный элемент dependency, его нужно просто скопировать:
- <https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java/3.14.0>

Применить Auto-Import

- При изменении **pom.xml** IDEA выдаст вам такое окно:



- IDEA предлагает вам варианты:
 - **Import Changes** – применить изменения. Т.к. вы добавили dependency, то Maven скачает зависимость
 - **Enable Auto-Import** – сделать чтобы IDEA вызывала **Import Changes** сама при всех изменениях
 - Это достаточно удобно, поэтому выберем этот вариант

Варианты работы с драйвером

1. Скачивание .exe файла для различных браузеров (chromedriver, geckodriver для Firefox и т.д.)
2. Использование библиотеки WebDriverManager:
<https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager>
3. Использование готовые «обертки» над Selenium'ом, например Selenide:
<https://mvnrepository.com/artifact/com.codeborne/selenide>

Скачивание драйвера браузера

- Далее нам нужно скачать сам драйвер для каждого нужного нам браузера
- Драйвер представляет из себя **.exe** файл
- Здесь есть ссылки на сайты, где можно найти ссылки на скачивание драйверов для нужных браузеров:
- <https://www.seleniumhq.org/download/>

Скачивание драйвера браузера

- Мы скачаем драйвер для **Google Chrome**:
<https://chromedriver.storage.googleapis.com/index.html>
- <https://chromedriver.storage.googleapis.com/index.html?path=98.0.4758.80/>
- Драйвер для **Firefox**:
- <https://github.com/mozilla/geckodriver/releases>

Работа со скаченным WebDriver

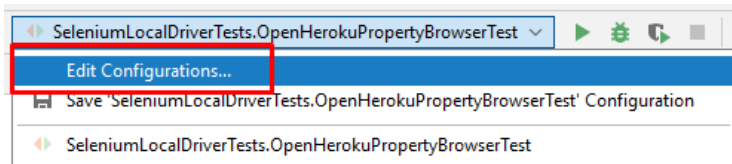
- Чтобы работать с браузером, нам надо создать объект самого драйвера
- `WebDriver webDriver = new ChromeDriver();`
- `WebDriver` – это интерфейс, а `ChromeDriver` – конкретная реализация для Chrome
- Также есть другие классы для работы с драйверами других браузеров
- Чтобы этот код сработал, надо чтобы программа знала путь до `.exe` файла с драйвером
- Если `.exe` файл лежит в каком-либо пути из переменной **PATH**, то Selenium его найдет

Указание пути к драйверу

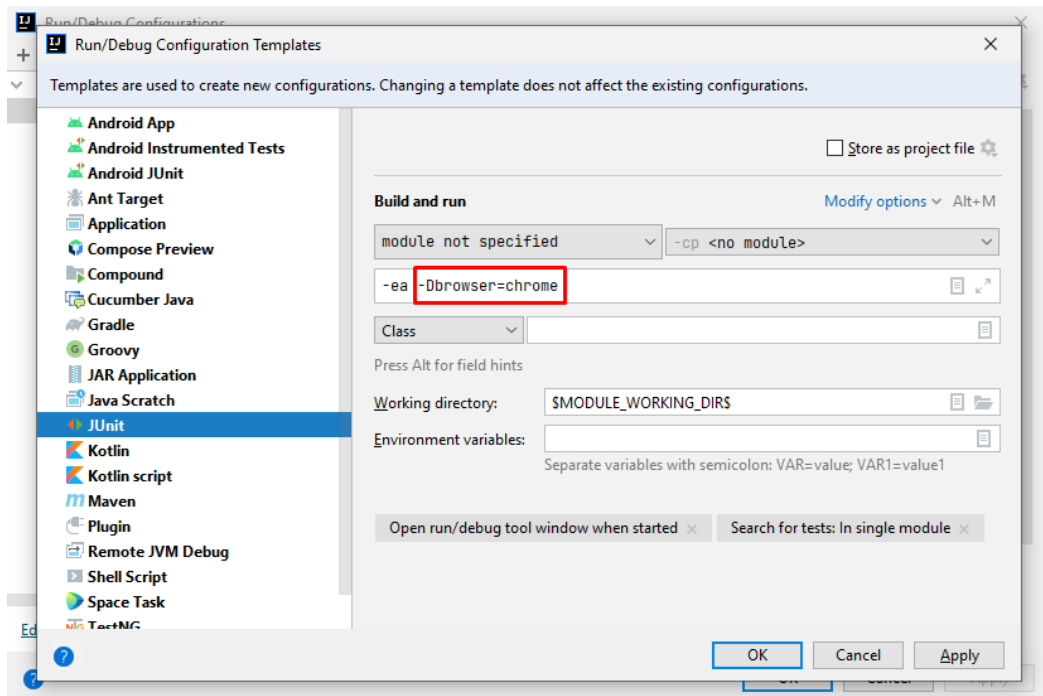
- Можно явно указать путь к файлу драйвера
- Для этого надо вызвать эту команду до создания драйвера:
- `System.setProperty("webdriver.chrome.driver", "путьКФайлу");`
- Например, так:
- `System.setProperty("webdriver.chrome.driver",
"C:\\chromedriver.exe");
WebDriver webDriver = new ChromeDriver();`
- Похожие настройки есть для драйверов других браузеров

Создание конфигурации запуска

- Можно создать параметр для запуска тестов в различных браузерах и прописать его в конфигурации:



- Добавить template для JUnit:



Создание конфигурации запуска

- Использование параметра для запуска тестов в различных браузерах:

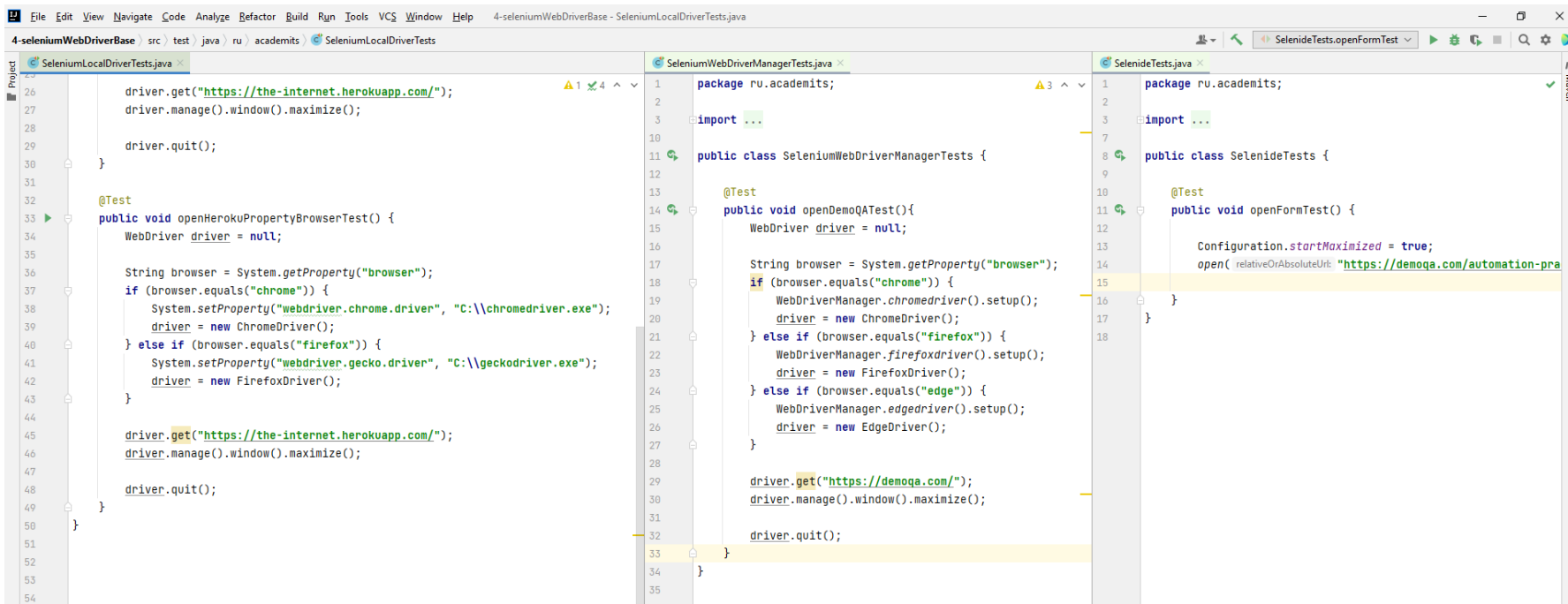
```
String browser = System.getProperty("browser");
if (browser.equals("chrome")) {
    System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");
    driver = new ChromeDriver();
} else if (browser.equals("firefox")) {
    System.setProperty("webdriver.gecko.driver", "C:\\geckodriver.exe");
    driver = new FirefoxDriver();
}
```

Работа с WebDriverManager

- Перед тем, как создавать объект самого драйвера, нам нужно вызвать команду WebDriverManager'a:
`WebDriverManager.chromedriver().setup();`
- Далее также создать объект самого драйвера
- `WebDriver webDriver = new ChromeDriver();`
- Для других браузеров:
- `WebDriverManager.operadriver().setup();`
`driver = new OperaDriver();`
- `WebDriverManager.firefoxdriver().setup();`
`driver = new FirefoxDriver();`

Использование Selenide

- Не нужно скачивать драйвер и даже создавать объект драйвера
- Можно сразу открыть нужный url с помощью команды *open*



```
4-seleniumWebDriverBase - SeleniumLocalDriverTests.java
SeleniumLocalDriverTests.java
26 driver.get("https://the-internet.herokuapp.com/");
27 driver.manage().window().maximize();
28
29 driver.quit();
30 }
31
32 @Test
33 public void openHerokuPropertyBrowserTest() {
34     WebDriver driver = null;
35
36     String browser = System.getProperty("browser");
37     if (browser.equals("chrome")) {
38         System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");
39         driver = new ChromeDriver();
40     } else if (browser.equals("firefox")) {
41         System.setProperty("webdriver.gecko.driver", "C:\\geckodriver.exe");
42         driver = new FirefoxDriver();
43     }
44
45     driver.get("https://the-internet.herokuapp.com/");
46     driver.manage().window().maximize();
47
48     driver.quit();
49 }
50
51 }
52
53
54
SeleniumWebDriverManagerTests.java
1 package ru.academits;
2
3 import ...
4
5
6
7
8
9
10
11 public class SeleniumWebDriverManagerTests {
12
13     @Test
14     public void openDemoQATest() {
15         WebDriver driver = null;
16
17         String browser = System.getProperty("browser");
18         if (browser.equals("chrome")) {
19             WebDriverManager.chromedriver().setup();
20             driver = new ChromeDriver();
21         } else if (browser.equals("firefox")) {
22             WebDriverManager.firefoxdriver().setup();
23             driver = new FirefoxDriver();
24         } else if (browser.equals("edge")) {
25             WebDriverManager.edgedriver().setup();
26             driver = new EdgeDriver();
27         }
28
29         driver.get("https://demoqa.com/");
30         driver.manage().window().maximize();
31
32         driver.quit();
33     }
34 }
35
SelenideTests.java
1 package ru.academits;
2
3 import ...
4
5
6
7
8 public class SelenideTests {
9
10     @Test
11     public void openFormTest() {
12
13         Configuration.startMaximized = true;
14         open(relativeOrAbsoluteUrl: "https://demoqa.com/automation-pra
15     }
16
17
18 }
```

Интерфейс WebDriver

- Перейти на нужный адрес можно так:
- **webdriver.get("https://academ-it.ru/mantisbt/login_page.php");**
- В интерфейсе **WebDriver** есть много различных методов
- Например, есть методы:
 - **webdriver.getTitle()** – получить title страницы
 - **webdriver.getCurrentUrl()** – получить текущий адрес страницы

Заккрытие браузера

- Чтобы в конце закрыть браузер, нужно использовать команду **quit**:
 - `webdriver.quit();`
- Есть похожая команда **close**, но ее не рекомендуется использовать — она закрывает только текущее окно



- То, что мы рассматривали в лекции до этого – это просто программы с использованием Selenium, а не тесты
- Потому что здесь нет самого главного – проверки соответствия фактического результата ожидаемому
- Чтобы наш код стал именно тестами, лучше всего использовать фреймворк для юнит-тестов
- Например, **JUnit** или **TestNG**

Установка JUnit

- Добавим **JUnit** в **pom.xml** внутрь **dependencies**:
- ```
<dependency>
 <groupId>org.junit.jupiter</groupId>
 <artifactId>junit-jupiter-api</artifactId>
 <version>5.7.1</version>
 <scope>test</scope>
</dependency>
```
- ```
<dependency>  
  <groupId>org.junit.jupiter</groupId>  
  <artifactId>junit-jupiter-engine</artifactId>  
  <version>5.7.1</version>  
  <scope>test</scope>  
</dependency>
```

Создание теста

- Чтобы создать тест, нам нужно создать метод и пометить его аннотацией `@Test`
- ```
public class HerokuTests {
 @Test
 public void openPageTest() {
 // код теста
 }
}
```
- В этом тесте мы откроем сайт heroku и проверим, что URL стал <https://the-internet.herokuapp.com/>

# Создание теста

@Test

```
public void loginTest() {
 WebDriverManager.chromedriver().setup();
 WebDriver webDriver = new ChromeDriver();
 webDriver.get("https://the-internet.herokuapp.com/");

 Assert.assertEquals("https://the-internet.herokuapp.com/",
 webDriver.getCurrentUrl());

 webDriver.quit();
}
```



# Ассерты

- Проверки делаются через **Assert'ы (ассерты)**
- Если проверка прошла успешно, то код теста просто продолжит выполняться
- Если проверка не прошла успешно, то код заваливается с ошибкой, и тест считается заваленным (failed)
- В JUnit входит большое количество стандартных ассертов:
  - **assertEquals(expected, actual)** – проверяет, что ожидаемый результат равен фактическому
  - **assertNotEquals(unexpected, actual)** – проверяет, что результат не равен указанному значению

# Ассерты

- В JUnit входит большое количество стандартных ассертов:
  - **assertArrayEquals(expected, actual)** – проверяет, что массив совпадает с ожидаемым массивом
  - **assertTrue(condition)** – проверка, что условие **true**
  - **assertFalse(condition)** – проверка, что условие **false**
  - **fail()** – позволяет завалить тест
- Также есть проверки равенства ссылок, проверки на **null** и др.
- В некоторых ассертах также можно указать сообщение, тогда оно выведется, если проверка завалится

# Ассерты

- Что будет, если тест завален:

! Tests failed: 1 of 1 test – 6 s 68 ms

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...
```

```
Starting ChromeDriver 2.43.600210 (68dcf5eebde37173d4027fa8635e332711d2874a) on port 22122
```

```
Only local connections are allowed.
```

```
нояб. 23, 2018 5:48:50 ПП org.openqa.selenium.remote.ProtocolHandshake createSession
```

```
INFO: Detected dialect: OSS
```

```
org.junit.ComparisonFailure:
```

```
Expected :https://academ-it.ru/mantisbt/my_view_page.php
```

```
Actual :https://academ-it.rul/mantisbt/my_view_page.php
```

```
<Click to see difference>
```

# Дублирование кода

- Предположим, мы будем писать много тестов для сайта
- Тогда в каждом методе у нас одинаковый будет код создания драйвера, перехода на страницу и, возможно, логина на сайт
- И в конце метода будет код закрытия драйвера и, возможно, разлогинивания
- Т.е. возникает дублирование кода, и этого следует избегать
- В JUnit есть возможность избежать дублирования кода, который должен выполняться перед тестом и после теста

# Аннотации Before/After в JUnit

- В JUnit есть аннотации, которыми можно помечать методы
- И, в зависимости от аннотации, метод будет вызываться до/после каждого теста или до/после всех тестов в классе
- **@BeforeEach** – метод будет выполняться перед каждым тестом
- **@AfterEach** – метод будет выполняться после каждого теста
- **@BeforeAll** – метод будет выполняться перед всеми тестами
- **@AfterAll** – метод будет выполняться после выполнения всех тестов

# Аннотации Before, After

- Вынесем логику инициализации теста в метод **setUp**, а логику завершения теста в метод **tearDown**
- Имена можно дать любые, но обычно они примерно такие
- И сделаем поле, чтобы хранить объект **WebDriver**

- ```
public class HerokuTests {  
    private WebDriver webDriver;
```

```
    @AfterEach
```

```
    public void tearDown() {  
        if (webDriver != null) {  
            webDriver.quit();  
        }  
    }  
}
```

Метод setUp будет
на следующем
слайде

Аннотации Before, After

@BeforeEach

```
public void setUp() {  
    WebDriverManager.chromedriver().setup();  
    WebDriver webDriver = new ChromeDriver();  
    webDriver.get("https://the-internet.herokuapp.com/");  
}
```

- В метод setUp также можно вынести авторизацию (логин)
- Теперь не нужно заново создавать драйвер и логиниться в каждом тесте
- И закрывать драйвер после каждого теста

Практика 1

- Реализовать тест на открытие страницы <https://the-internet.herokuapp.com/>
- Проверить, что title страницы – «The Internet»

Работа с элементами на веб странице

Поиск элементов на странице

- Чтобы делать что-то полезное мы сначала должны выбрать нужный нам HTML элемент на странице
- Для представления HTML элементов в Selenium используется интерфейс **WebElement**
- Искать элементы можно при помощи **локаторов**, например, атрибут **id**, **XPath**, **CSS селектор** и др.
- Чтобы задать локатор используется класс **By**
- Например, **By.className("login-logo")** – это будет локатор поиска по CSS классу

Поиск элементов на странице

- Для поиска есть 2 метода:
 - `List<WebElement> findElements(By by)` – выдает список всех элементов, удовлетворяющих локатору
 - `WebElement findElement(By by)` – выдает только первый элемент, удовлетворяющий локатору
- Пример поиска одного элемента:
 - `WebElement pageTitle`
 `= webDriver.findElement(By.className("heading"));`
 - `// получили элемент заголовка страницы`

Варианты поиска

| Тип поиска | Метод класса By |
|---|---|
| CSS селектор | <code>By.cssSelector(".login-logo > img")</code> |
| Атрибут name | <code>By.name("username")</code> |
| Атрибут id | <code>By.id("login-form")</code> |
| CSS класс | <code>By.className("login-logo")</code> |
| Имя тега | <code>By.tagName("img")</code> |
| XPath | <code>By.xpath("//*[@id=\"username\"]")</code> |
| Поиск ссылки по полному совпадению текста | <code>By.linkText("зарегистрировать новую учетную запись")</code> |
| Поиск ссылки по подстроке из текста | <code>By.partialLinkText("учетную")</code> |



Варианты поиска

- Самые удобные варианты поиска – по **name** и **id**
- Потому что **id** – вообще уникален по странице
- **name** обычно уникален внутри формы
- Если есть возможность, нужно чтобы у всех ключевых элементов были заданы атрибуты **id** или **name** – можно просить об этом разработчиков
- Но это не всегда возможно, т.к. некоторые элементы могут повторяться (например, в таблице может быть много строк)
- И тогда удобно использовать CSS селекторы или XPath
- Поиск по тексту ссылок не очень, т.к. зависит от локализации и текста

Поиск внутри элемента

- У самих элементов есть те же самые методы для поиска
- Только они уже будут искать не по всему документу, а только внутри элемента
- `// сначала нашли форму`
`WebElement form = webDriver.findElement(By.id("login-form"));`
- `// потом получили все инпуты из формы`
`List<WebElement> inputs = form.findElements(By.tagName("input"));`
- `// печатаем все элементы в консоль`
`for (WebElement input : inputs) {`
 `System.out.println(input.getAttribute("outerHTML"));`
`}`

Основные методы работы с элементами

- `// получение текста элемента`
`WebElement input = webDriver.findElement(By.id("login"));`
`String text = input.getText();`
- `element.click();` `// клик по элементу`
- `element.clear();` `// очистка текста в элементе`
- `element.isDisplayed();` `// проверка отображен ли элемент`
- `element.isSelected();` `// проверка выбран ли элемент`
- `element.getAttribute();` `// получить значение атрибута`
- `element.getSize();` `// получить размер элемента`

Ввод данных, отправка формы

- Ввести текст в поле ввода можно при помощи команды **sendKeys**:
- `webdriver.findElement(By.id("username")).sendKeys("selenium");`
- Далее нам нужно отправить форму
- Это можно сделать разными способами:
 - Нажать **Enter**, находясь в поле ввода:
 - `webdriver.findElement(By.id("username")).sendKeys(Keys.RETURN);`
 - Найти и нажать кнопку при помощи метода **click**
 - `webdriver.findElement(By.cssSelector("input[type='submit']")).click();`

Работа с чекбоксами и радио-баттонами

- Чтобы выбрать чекбокс или радио-баттон, нужно кликнуть по нему
- Давайте попробуем выбрать радио-баттон «Тестирование»
- https://academ-it-school.ru/payment?course=java_begin
- У радио-баттонов из одной группы атрибут **name** одинаковый. А атрибута **id** часто может не быть
- Поэтому мы будем ориентироваться на атрибут **value** – он должен быть свой у каждого radio button'a
- Узнаем нужное значение **value** через Dev Tools – testing
- `webDriver.findElement(By.cssSelector("input[name='Course Type'][value='Testing']")).click();`

Работа со списками

- Чтобы работать с выпадающим списком есть класс `Select`
- Давайте здесь в списке выберем нужное значение:
 - <http://htmlbook.ru/html/select>
- Объект `Select` можно создать через конструктор, в него надо передать `WebElement`
- `WebElement elem = webDriver.findElement(By.name("select2"));`
`Select select = new Select(elem);`

Работа со списками

- В классе `Select` есть много методов для работы с опциями списка:
 - `void selectByIndex(int index)` – выбрать опцию по индексу (отсчитывается от 0)
 - `void selectByValue(String value)` – выбрать опцию по значению атрибута **value**
 - `void selectByVisibleText(String text)` – выбрать опцию по тексту
 - `WebElement getFirstSelectedOption()` – получить первую выбранную опцию
 - `List<WebElement> getAllSelectedOptions()` – получить все выбранные опции
 - `List<WebElement> getOptions()` – получить все опции

Работа со списками

- В классе `Select` есть много методов для работы с опциями списка:
 - `void deselectByIndex(int index)` – снять выбор опции по индексу (отсчитывается от 0)
 - `void deselectByValue(String value)` – снять выбор опции по значению атрибута **value**
 - `void deselectByVisibleText(String text)` – снять выбор опции по тексту
 - `void deselectAll()` – очистить выбор
 - `boolean isMultiple()` – выдает `true`, если список с множественным выбором

Практика 2

- Открыть страницу
<https://the-internet.herokuapp.com/>
- Перейти в раздел «Dropdown»
- Выбрать «Option 2»
- Проверить, значение успешно выбрано

Ожидания

Необходимость ожиданий

- Давайте попробуем зайти на страницу и оставить комментарий:
- <https://news.s7.ru/news?id=13441>
- Заполняем поле «Ваше имя», его id = "author"
- ```
WebDriver webDriver = new ChromeDriver();
webDriver.get("https://news.s7.ru/news?id=13441");

WebElement author = webDriver.findElement(By.id("author"));
author.sendKeys("name");
```
- Если запустить этот код, то программа упадет с ошибкой

# Необходимость ожиданий

- Смысл ошибки – не удалось найти элемент с id = “author”

```
Main x
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" ...
Starting ChromeDriver 2.43.600210 (68dcf5eebde37173d4027fa8635e332711d2874a) on port 9316
Only local connections are allowed.
нояб. 22, 2018 9:18:20 ММ org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
Exception in thread "main" org.openqa.selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"id","selector":"author"}
(Session info: chrome=70.0.3538.102)
(Driver info: chromedriver=2.43.600210 (68dcf5eebde37173d4027fa8635e332711d2874a),platform=Windows NT 10.0.17134 x86_64) (WARNING: The server did not provide any stacktrace information)
Command duration or timeout: 0 milliseconds
For documentation on this error, please visit: http://seleniumhq.org/exceptions/no_such_element.html
Build info: version: '3.14.0', revision: 'aacc0ce0', time: '2018-08-02T20:19:58.91Z'
System info: host: 'PAVEL-PC', ip: '192.168.1.68', os.name: 'Windows 10', os.arch: 'amd64', os.version: '10.0', java.version: '10.0.2'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Capabilities {acceptInsecureCerts: false, acceptSslCerts: false, applicationCacheEnabled: false, browserConnectionEnabled: false, browserName: chrome, chrome: {chromedriverVersion: 2.43.600210 (68dcf5eebde37173d4027fa8635e332711d2874a), nativePlatformVersion: ''}, platformName: 'Windows NT 10.0.17134 x86_64', setWindowRect: true, strictFileInteractability: false, timeouts: {default: 300000, implicit: 0}, unhandledPromptBehavior: 'dismiss-and-warn'}
Session ID: 5bd7f1754cd63c34e878229142e984b2
*** Element info: {Using=id, value=author}
at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:488)
at org.openqa.selenium.remote.ErrorHandler.createThrowable(ErrorHandler.java:214)
at org.openqa.selenium.remote.ErrorHandler.throwIfResponseFailed(ErrorHandler.java:166)
at org.openqa.selenium.remote.http.JsonHttpResponseCodec.reconstructValue(JsonHttpResponseCodec.java:40)
at org.openqa.selenium.remote.http.AbstractHttpResponseCodec.decode(AbstractHttpResponseCodec.java:80)
at org.openqa.selenium.remote.http.AbstractHttpResponseCodec.decode(AbstractHttpResponseCodec.java:44)
at org.openqa.selenium.remote.HttpCommandExecutor.execute(HttpCommandExecutor.java:158)
at org.openqa.selenium.remote.service.DriverCommandExecutor.execute(DriverCommandExecutor.java:83)
at org.openqa.selenium.remote.RemoteWebDriver.execute(RemoteWebDriver.java:548)
at org.openqa.selenium.remote.RemoteWebDriver.findElement(RemoteWebDriver.java:322)
at org.openqa.selenium.remote.RemoteWebDriver.findElementById(RemoteWebDriver.java:368)
at org.openqa.selenium.By$ById.findElement(By.java:188)
at org.openqa.selenium.remote.RemoteWebDriver.findElement(RemoteWebDriver.java:314)
at ru.academits.Main.main(Main.java:28)

Process finished with exit code 1
```



# Необходимость ожиданий

- Дело в том, что некоторые страницы используют **AJAX**
- При загрузке страницы грузится только основная разметка и код на JavaScript
- А затем уже этот код делает запросы на сервер, и с помощью полученных данных отрисовывает HTML страницы
- С точки зрения браузера (и селениума) страница считается загруженной когда загрузилось изначальное содержимое
- Поэтому наша программа продолжает работать дальше со страницей, и пытается обратиться к элементу, которого еще нет – JavaScript просто не успел еще его создать
- И возникает ошибка

# Ожидания (waits)

- Чтобы все отработало хорошо, нам нужно дождаться момента, когда все отрисуется
- Для этого в Selenium есть ожидания (**waits**)
- <https://www.selenium.dev/documentation/webdriver/waits/>

# Виды ожиданий

Implicit wait (неявное)	Explicit wait (явное)
Указывается 1 раз после установки драйвера	Указывается отдельно для каждого места ожидания
<p>Используется для:</p> <ul style="list-style-type: none"><li>• ожидание полной загрузки страницы <code>pageLoadTimeout()</code></li><li>• ожидание элементов на странице – <code>implicitlyWait()</code></li><li>• ожидание выполнения асинхронного запроса – <code>setScriptTimeout()</code></li></ul>	<p>Использует <code>ExpectedConditions</code>:</p> <ul style="list-style-type: none"><li>• <code>presenceOfElementLocated()</code>;</li><li>• <code>visibilityOfElementLocated()</code>;</li><li>• <code>invisibilityOfElementLocated()</code>;</li><li>• <code>elementToBeClickable()</code>;</li><li>• <code>elementToBeSelected()</code>;</li><li>• и др.</li></ul>
При проверке на отсутствие элемента будет задерживать тест	Срабатывает только 1 раз

# Ожидания (waits)

- <https://news.s7.ru/news?id=13441>
- Здесь мы создаем экземпляр wait, и говорим ждать пока не появится элемент с классом `comments-block-wrapper`
- `WebDriverWait wait = new WebDriverWait(webDriver, 30, 500);  
wait.until(ExpectedConditions.presenceOfElementLocated(  
By.className("comments-block-wrapper")));`
- Добавляем этот код перед обращением к полю ввода, и после этого все работает

# Ожидания (waits)

- `WebDriverWait wait = new WebDriverWait(webDriver, 30, 500);`
- В конструктор мы передаем драйвер и 2 числа:
  - 30 – число секунд, в течение которых нужно ждать
  - 500 – число миллисекунд, интервал между проверками выполнилось ли условие ожидания
- `wait.until(ExpectedConditions.presenceOfElementLocated(By.className("comments-block-wrapper")));`
- Здесь мы указали условие, что мы ждем пока на странице не появится элемент, соответствующий указанному локатору

# Задание на дом «Selenium WebDriver»

- В ДЗ используем WebDriver Manager
- Реализовать логику с параметром запуска – минимум 3 вида браузеров
- Структурировать код с помощью аннотаций @Before / @After
- Написать автотест для формы на странице:  
<https://demoqa.com/automation-practice-form>
- Заполните все поля, нажать кнопку Submit, в открывшемся окне проверить, что данные заполнены верно
- При необходимости использовать wait