Fundamentals of Software Testing Assignment Part 2 (of 3)

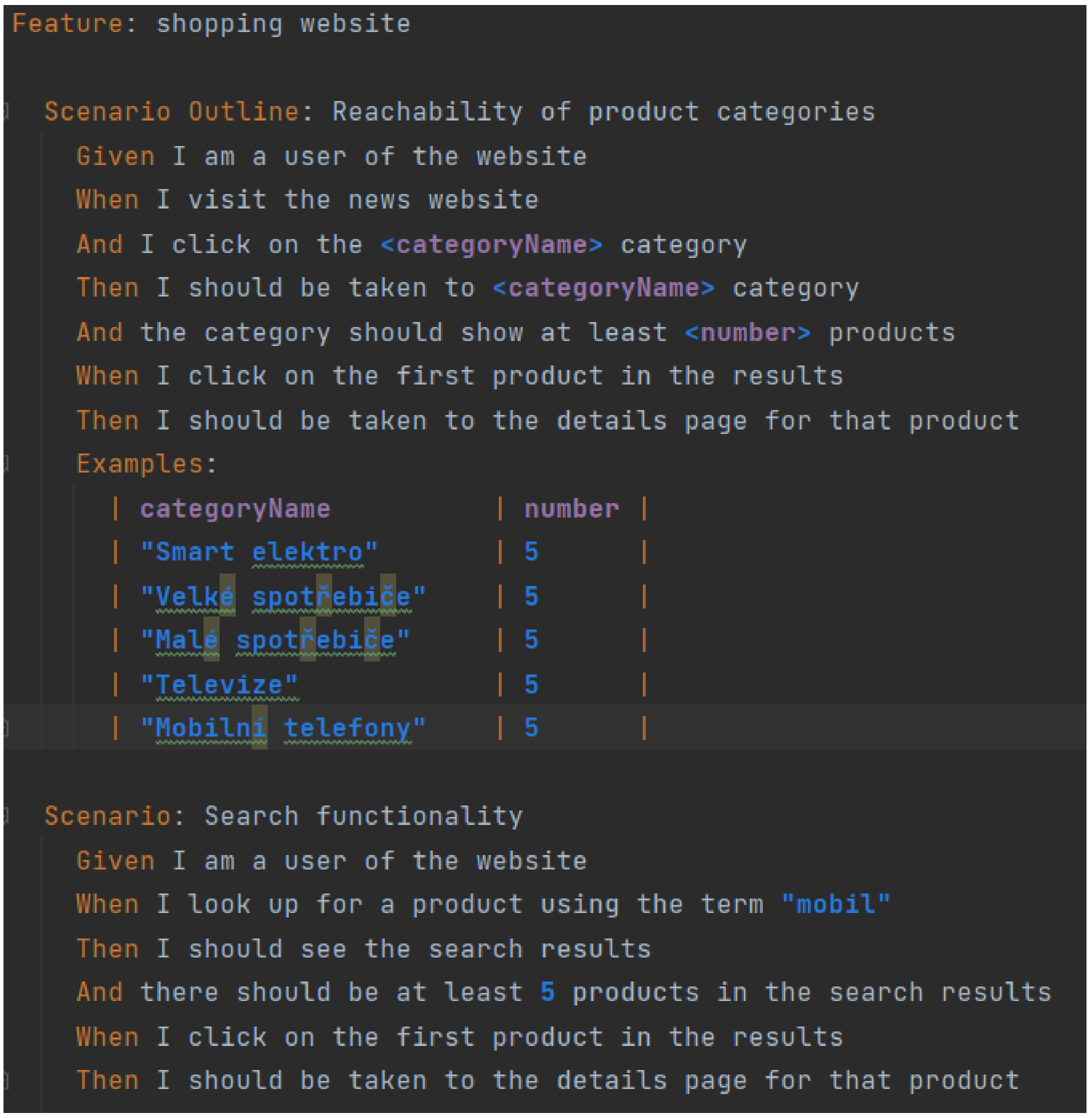
Otmar Nezdařil (2312649)

Git: https://github.com/otheec/WebTesting

Web Test Automation and BDD

Link to the recording: https://www.youtube.com/watch?v=aa_qHOSp4Z8

Implementing scenarios resulted into 6 tests: 5 for checking categories and one for checking search functionality.

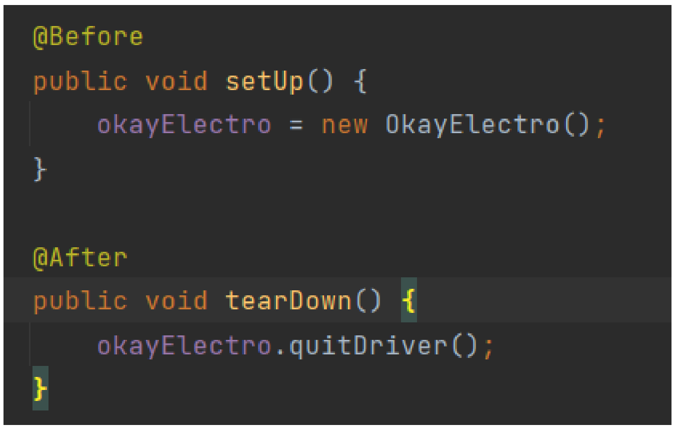

```
Feature: shopping website

  Scenario Outline: Reachability of product categories
    Given I am a user of the website
    When I visit the news website
    And I click on the <categoryName> category
    Then I should be taken to <categoryName> category
    And the category should show at least <number> products
    When I click on the first product in the results
    Then I should be taken to the details page for that product
    Examples:
      | categoryName           | number |
      | "Smart elektro"        | 5      |
      | "Velké spotřebiče"     | 5      |
      | "Malé spotřebiče"      | 5      |
      | "Televize"             | 5      |
      | "Mobilní telefony"     | 5      |

  Scenario: Search functionality
    Given I am a user of the website
    When I look up for a product using the term "mobil"
    Then I should see the search results
    And there should be at least 5 products in the search results
    When I click on the first product in the results
    Then I should be taken to the details page for that product
```

One of the things I've found is that I must put the strings in parentheses for passing variables into scenarios, as how one screenshot above.

I have created the class with desired methods to communicate with the webpage and call methods in step defs class as it seems to be better design pattern.

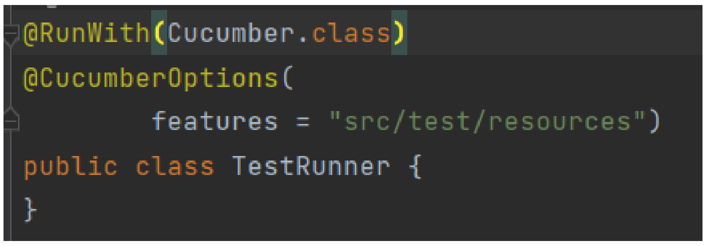
I have found out locating element from my chosen webpage seems a bit challenging (that is why I choose different webpage for the second part of the assignment) and even if it's not directly a matter of testing, I have learnt a lot about webpage component naming etc. for further development of the website and following ease of testability.

I have also added the @Before and @After annotated methods to cerate and quit instance of the tested class with driver.



```
@Before
public void setUp() {
    okayElectro = new OkayElectro();
}

@After
public void tearDown() {
    okayElectro.quitDriver();
}
```

Running the cucumber is implemented in the TestRunner class.

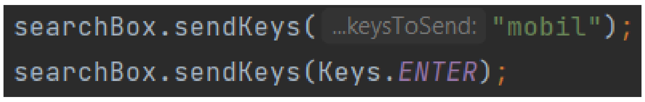


```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "src/test/resources")
public class TestRunner {
}
```

All test passed without any problem.

Interesting design decisions:

- Searching is implemented using SendKeys (text + Enter)



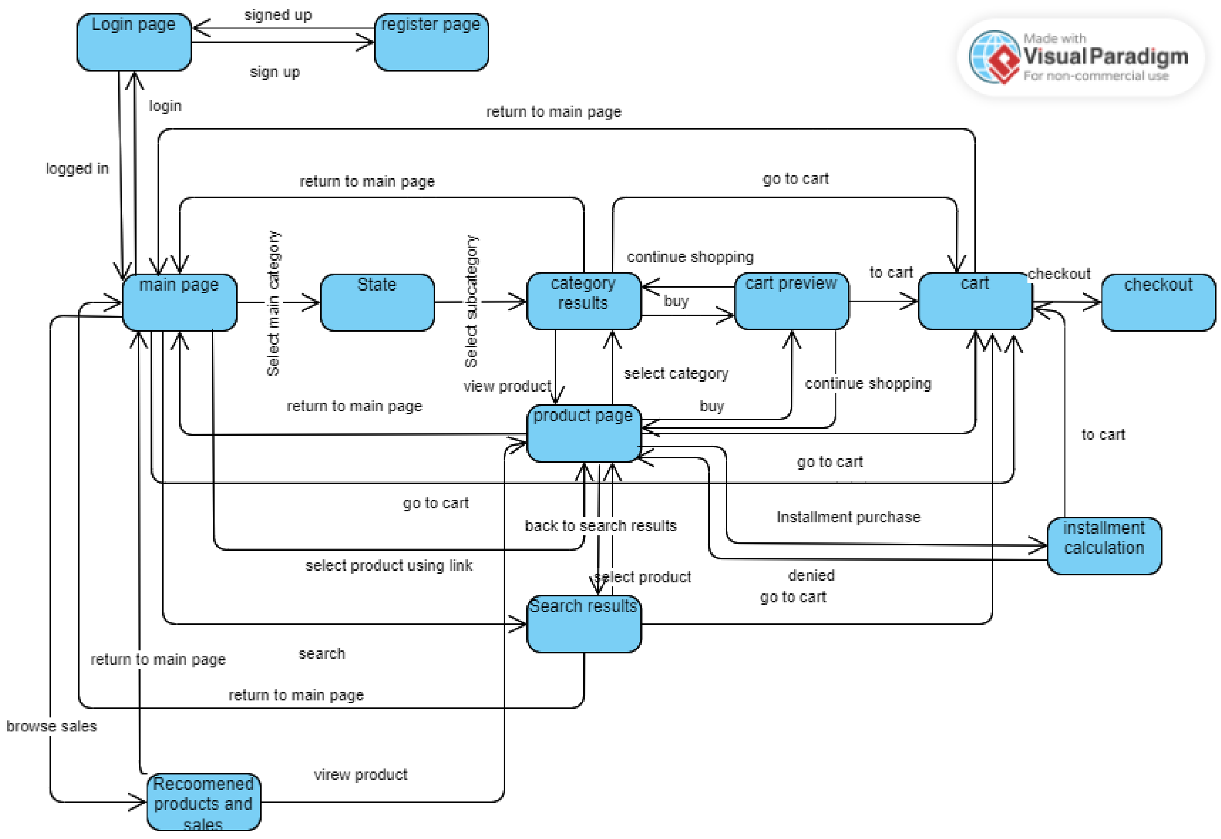
```
searchBox.sendKeys( ...keysToSend: "mobil");
searchBox.sendKeys(Keys.ENTER);
```

- Locating products using divs with common class (wrappers)
- Compared with the webpage used in the second task, I have been able to locate almost every element just by class name and it there was no need to sonsider composition of the HTML elements (inner or outer divs)

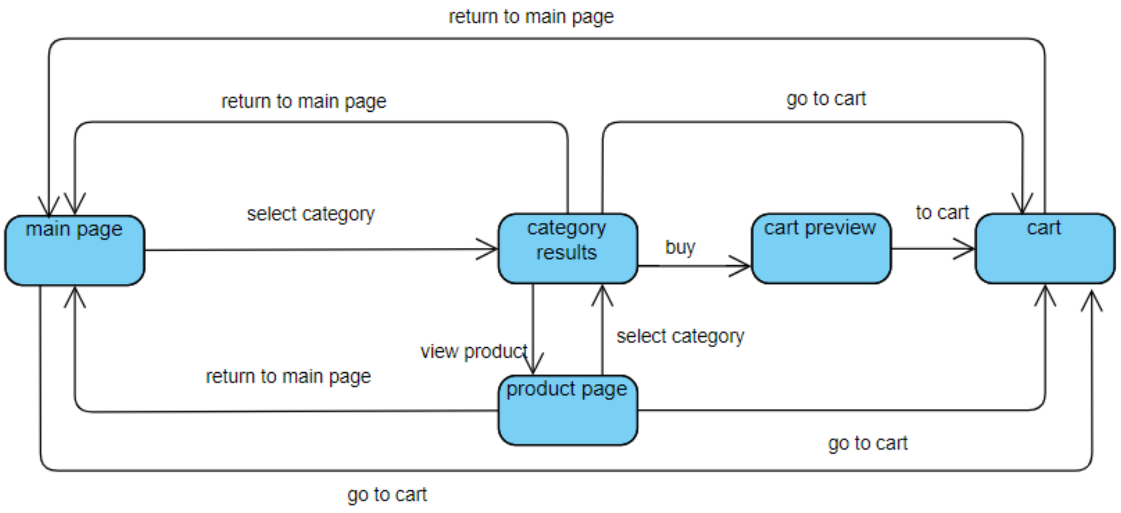
Model Based Testing

Link to the recording: https://www.youtube.com/watch?v=769UqL929kA

For the second task, I have again used online shop with electronics. Here is provided conceptualized model of the webpage interactions by user:



From that, I have reduced it to subset:



Interactions model design decisions:

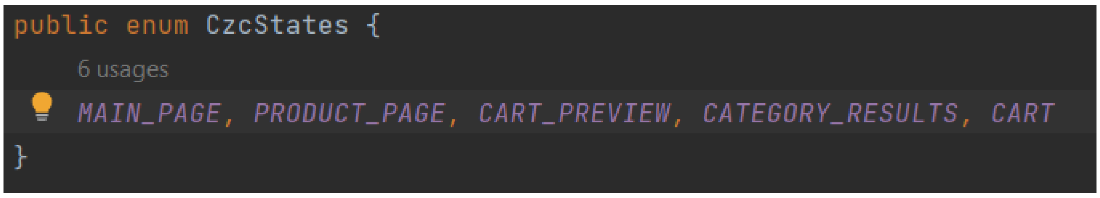
- Since I am locating just the "Buy button" from the page of multiple product list, there is no possibility of product not being available (not in stock) – this would not be good practice, if I would not aim to implement just the reduced subset of the interaction model (example of better maintainable testing approach could be: "not in stock" button and "buy button" would be in same wrapper, based on this I would get element text and find out if product is in stock or not.

- Since product is always added to the cart, there is no counter to verify number of products in the cart.
- Returning from the cart preview to the main page is not possible, because the action is implemented by clicking of the logo on the webpage and the logo is not presented in the cart preview.

Implementation- Model

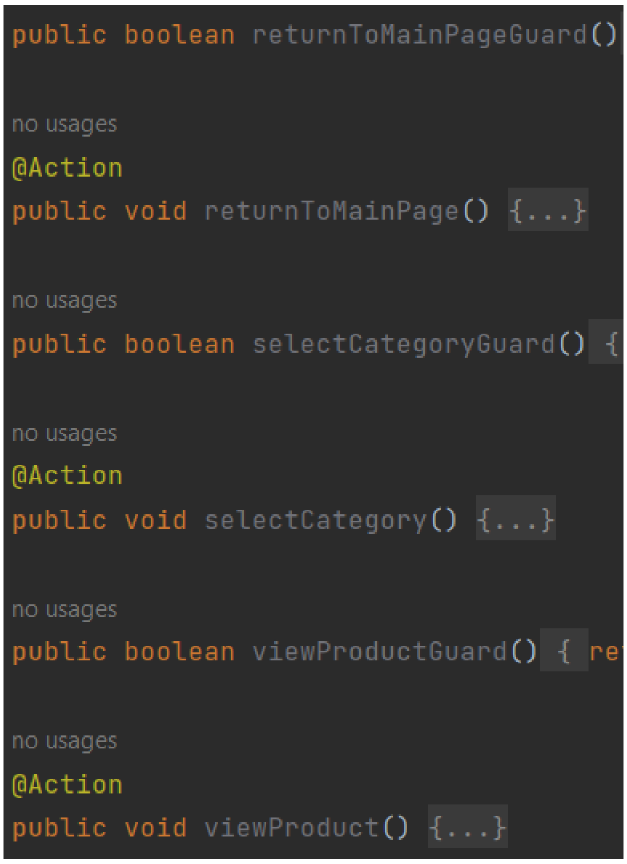
In the same way as in previous part, I have implemented all driver-based operation in method is separate class to call just one method per one action.

States:

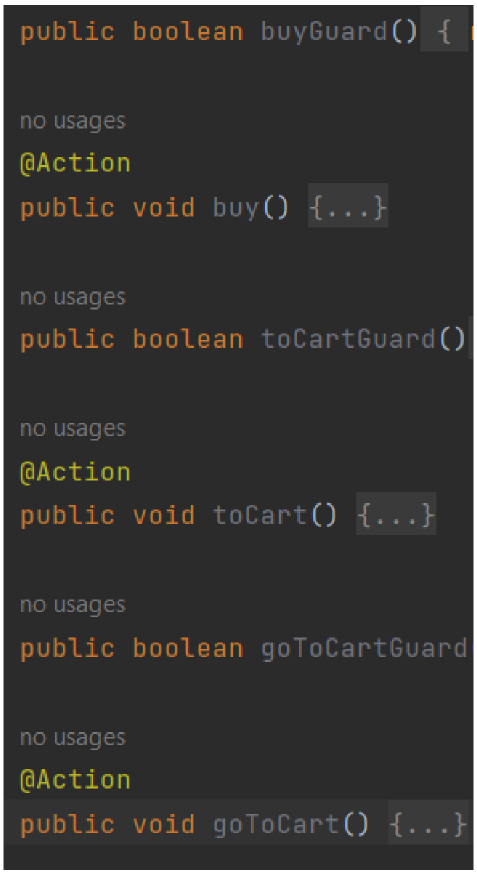


```
public enum CzcStates {
    6 usages
    MAIN_PAGE, PRODUCT_PAGE, CART_PREVIEW, CATEGORY_RESULTS, CART
}
```

Actions:



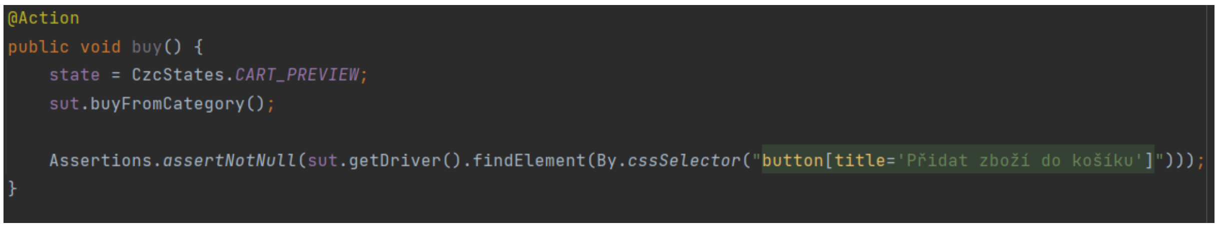
```
public boolean returnToMainPageGuard()

no usages
@Action
public void returnToMainPage() {...}

no usages
public boolean selectCategoryGuard() {

no usages
@Action
public void selectCategory() {...}

no usages
public boolean viewProductGuard() { re

no usages
@Action
public void viewProduct() {...}
```



```
public boolean buyGuard() {

no usages
@Action
public void buy() {...}

no usages
public boolean toCartGuard()

no usages
@Action
public void toCart() {...}

no usages
public boolean goToCartGuard

no usages
@Action
public void goToCart() {...}
```

All actions haves guard to verify previous state.

Each action test by Assertions achieved state.

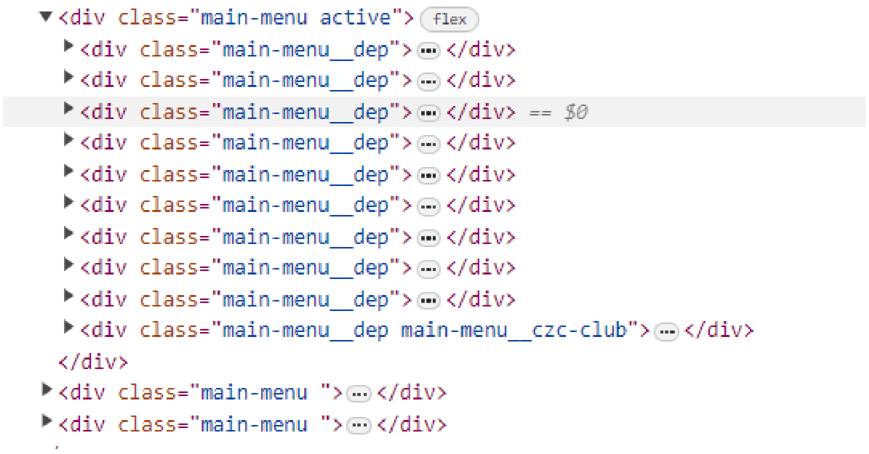


```
@Action
public void buy() {
    state = CzcStates.CART_PREVIEW;
    sut.buyFromCategory();

    Assertions.assertNotNull(sut.getDriver().findElement(By.cssSelector("button[title='Přidat zboží do košíku']")));
}
```

Implementation – Driver

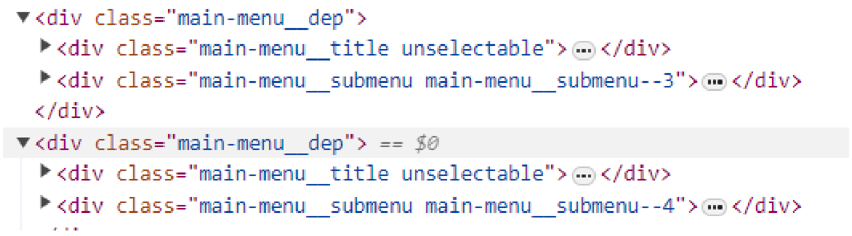
Locating HTML elements was more difficult compared with previous task and there was need to more use structure of the page (locate inner elements). For example, locating one part of the menu:

All divs have same class.



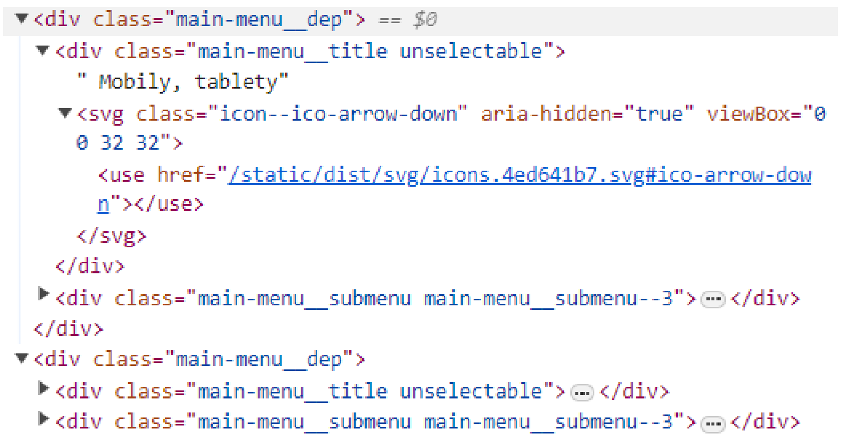
```
▼<div class="main-menu active"> flex
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div> == $0
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep"> ··· </div>
  ▶<div class="main-menu__dep main-menu__czc-club"> ··· </div>
 </div>
▶<div class="main-menu "> ··· </div>
▶<div class="main-menu "> ··· </div>
```

All inner elements have same structure again.



```
▼<div class="main-menu__dep">
  ▶<div class="main-menu__title unselectable"> ··· </div>
  ▶<div class="main-menu__submenu main-menu__submenu--3"> ··· </div>
 </div>
▼<div class="main-menu__dep"> == $0
  ▶<div class="main-menu__title unselectable"> ··· </div>
  ▶<div class="main-menu__submenu main-menu__submenu--4"> ··· </div>
```

And the difference comes at very inner elements level.



```
▼<div class="main-menu__dep"> == $0
  ▼<div class="main-menu__title unselectable">
     " Mobily, tablety"
    ▼<svg class="icon--ico-arrow-down" aria-hidden="true" viewBox="0
     0 32 32">
       <use href="/static/dist/svg/icons.4ed641b7.svg#ico-arrow-dow
       n"></use>
     </svg>
   </div>
  ▶<div class="main-menu__submenu main-menu__submenu--3"> ··· </div>
 </div>
▼<div class="main-menu__dep">
  ▶<div class="main-menu__title unselectable"> ··· </div>
  ▶<div class="main-menu__submenu main-menu__submenu--3"> ··· </div>
```

So, the code co select element look like this:

```
WebElement categoryButton =
driver.findElement(By.xpath("//div[contains(@class, 'main-
menu__dep') " + "and .//div[contains(@class, 'main-menu__title
unselectable') " + "and contains(text(), 'Mobily,
tablety')]]"));
```

Test

I have used the same configuration as mentioned during the recordings. In the provided video, transition-pair coverage 100% is not achieved, but could be achieved by circa 300 test length parameter.