SELENIUM

ELŻBIETA SĄDEL

GRZEGORZ WITEK

DRY - DON'T REPEAT YOURSELF

- Unikanie wszelkiego rodzaju powtórzeń.
- *"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."* - The Pragmatic Programmer

ŹLE

```
driver.findElement(By.cssSelector("input#wpsc_checkout_form_9")).clear();
driver.findElement(By.cssSelector("input#wpsc_checkout_form_9")).sendKeys(email);
driver.findElement(By.cssSelector("input#wpsc_checkout_form_2")).clear();
driver.findElement(By.cssSelector("input#wpsc_checkout_form_2")).sendKeys(firstName);
driver.findElement(By.cssSelector("input#wpsc_checkout_form_3")).clear();
driver.findElement(By.cssSelector("input#wpsc_checkout_form_3")).sendKeys(lastName);
driver.findElement(By.cssSelector("textarea#wpsc_checkout_form_4")).clear();
driver.findElement(By.cssSelector("textarea#wpsc_checkout_form_4")).sendKeys(address);
```

LEPIEJ

```
provideInput(By.cssSelector("input#wpsc_checkout_form_9"), email);
provideInput(By.cssSelector("input#wpsc_checkout_form_2"), firstName);
provideInput(By.cssSelector("input#wpsc_checkout_form_3"), lastName);
provideInput(By.cssSelector("textarea#wpsc_checkout_form_4"), address);
```

KISS - KEEP IT SIMPLE STUPID

- Reguła powstała w latach 60. XX wieku w środowisku amerykańskich inżynierów wojskowych i przypisywana iżynierowi lotnictwa Kelly'emu Johnsonsowi. Istotą przekazu miało być tworzenie projektowanych samolotów w tak prosty sposób, aby każdy średnio uzdolniony mechanik mógł je naprawić w warunkach polowych i przy użyciu prostych narzędzi.

ŹLE

```
public void addToCard(){
    WebElement buyButton = driver.findElement(By.cssSelector("input.wpsc_buy_button"));
    Actions action = new Actions(driver);
    action.moveToElement(buyButton).click();
    waitForAddedToCardMessage();
}
```

LEPIEJ

```
public void addToCard(){
    WebElement buyButton = driver.findElement(By.cssSelector("input.wpsc_buy_button"));
    buyButton.click();

   wait.until(ExpectedConditions.visibilityOf(driver.findElement(addedToCardMessage)));
}
```

YAGNI - YOU AIN'T GONNA NEED THIS

- Zasada mówiąca o tym, że nie powinniśmy pisać funkcjonalności, które nie są niezbędne.

ŹLE

```
@Test
public void cancelBuyProduct(){
   ClickAllProductsButton();
   addToCard("iPhone 5");

   ClickGoToCardButton();
   ClickRemoveProductButton();

   AssertThatRemoveProductMessageIsCorrect();
}
```

LEPIEJ

```
@Test
public void cancelBuyProduct(){
   driver.findElement(allProductsButton).click();
   addToCard("iPhone 5");

   driver.findElement(goToCardButton).click();
   driver.findElement(removeProductButton).click();

   Assert.assertEquals("Wrong message after cancel remove product",
         "Oops, there is nothing in your cart.", driver.findElement(By.cssSelector("div.entry-
         content")).getText());
}
```

PAGE OBJECT PATTERN

Page Object Pattern – wzorzec w architekturze tworzenia oprogramowania komputerowego (testów), który zakłada, iż każda strona/ekran ma swoje odzwierciedlenie w kodzie jako klasa/obiekt.

Każdy z obiektów/klas składa się z prywatnych pól będących poszczególnymi elementami dostępnymi na stronie oraz z publicznych metod opisujących dostępne akcje, które można wykonywać na tych elementach.

Każda akcja powinna zwracać wynik w postaci następnej klasy/obiektu (strony/ekranu) lub wartości, które są pobierane do asercji.

ZALETY POP

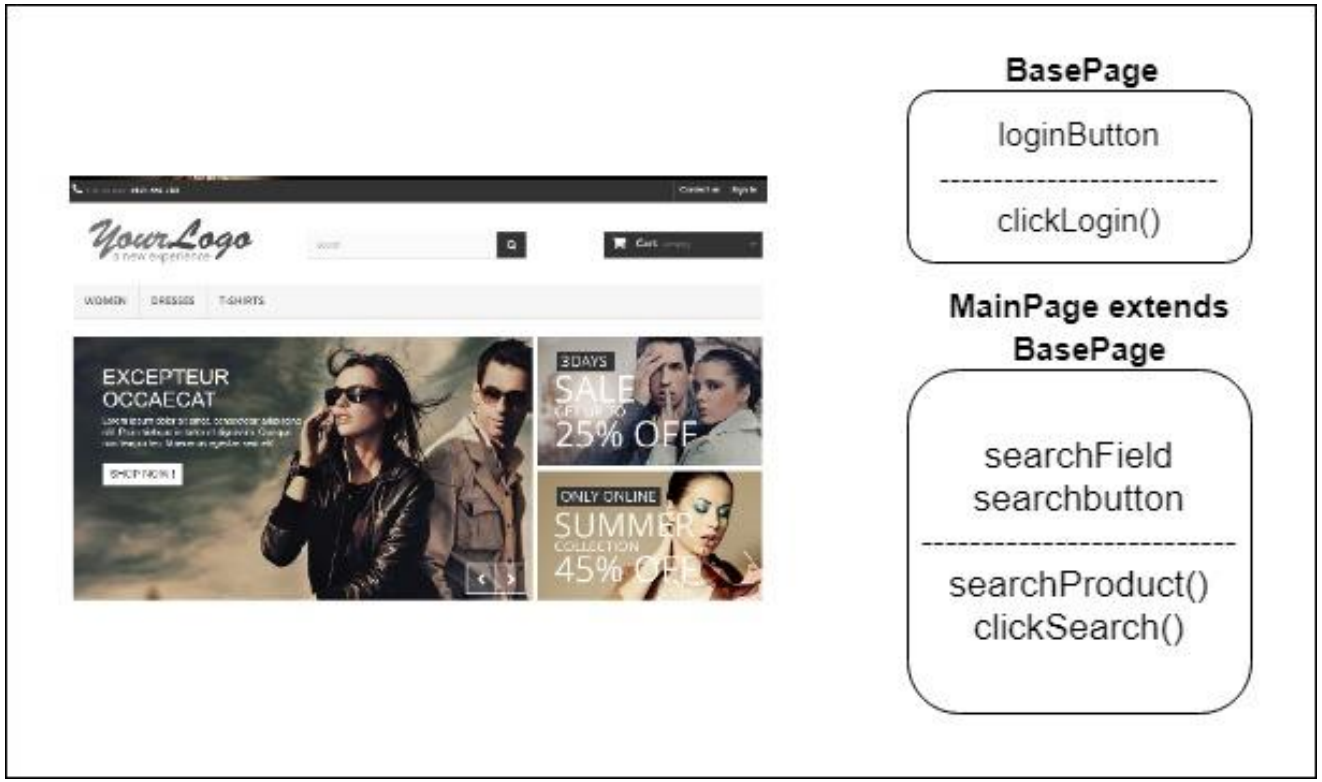
- Pozwala oddzielić logikę metod robiących akcje na stronie od reprezentacji elementów na stronie w postaci obiektów WebElement
- Pozwala łatwiej utrzymywać testy, jeżeli zmienia się lokator (selektor) na stronie np. poprzez zmianę układu strony to lokator (selektor) do elementu zmieniamy tylko w jednym miejscu
- Pozwala w łatwiejszy sposób zrozumieć abstrakcje związaną z pisaniem testów poprzez wykorzystywanie metod, których nazwy mówią jaka akcja jest wykonywana na stronie
- Scenariusze testowe są krótsze, mogą wykorzystywać już istniejące metody i elementy co przekłada się na przyspieszenie automatyzacji w dalszej perspektywie

DOBRE PRAKTYKI

- Klasy Page nie powinny posiadać żadnych asercji.
- W klasie Page powinny być tylko utrzymywane znaczące elementy, nie cała strona. Tak samo z metodami/akcjami.
- Podczas nawigacji do innej strony, metoda powinna zwracać obiekt odpowiadający docelowej stronie.

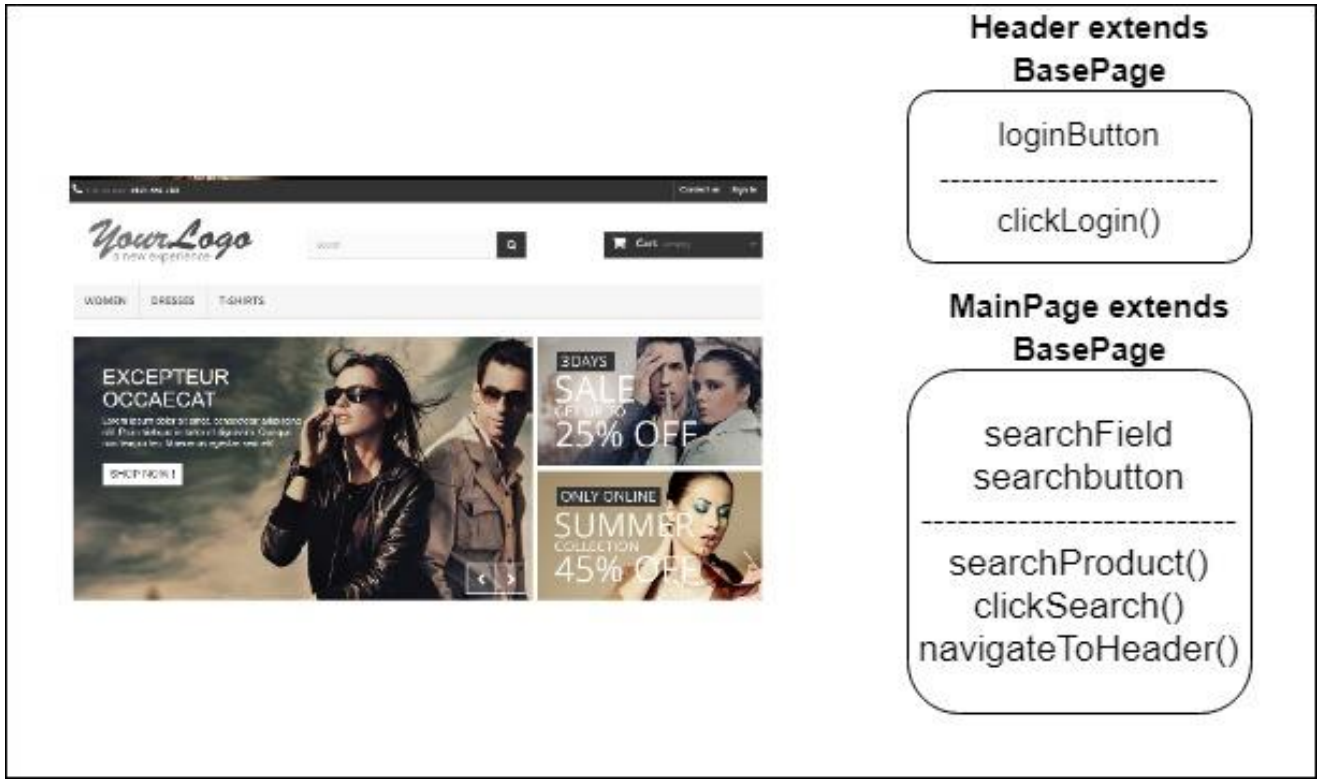
JAK NAWIGOWAĆ?

- **Dziedziczenie** – jest to mechanizm współdzielenia pomiędzy klasami, polegający na tym iż klasa dziedzicząca uzyskuje wszystkie udostępnione metody i pola klasy po ktorej dziedziczy. W przypadku POP klasa nadrzędna przedstawia wszystkie elementy i akcje dostępna na każdej stronie (footer/header).



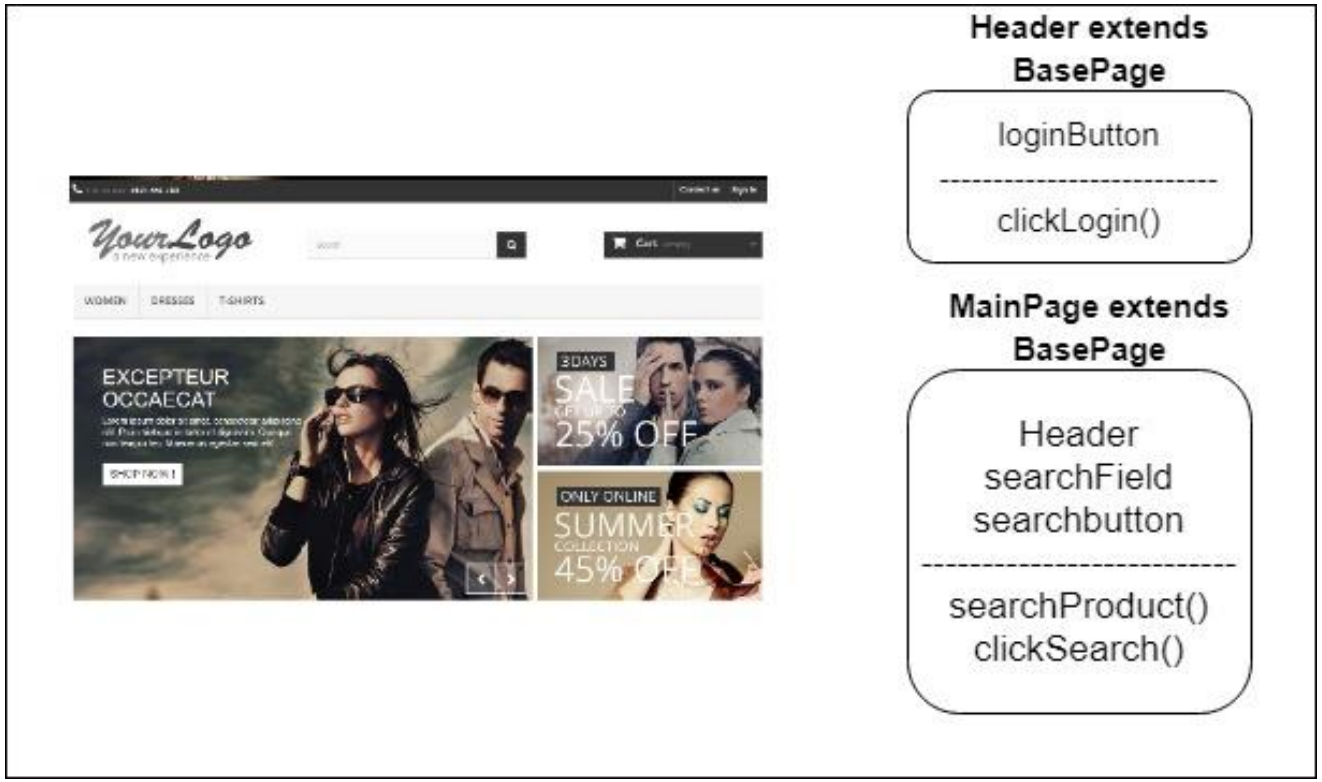
JAK NAWIGOWAĆ?

- **Metody** – tworzenie metod, których zadaniem jest zwrócenie odpowiedniego obiektu. Metody nawigacyjne nic więcej nie robią, tylko zwracają odpowiedni obiekt, na którym można wykonywać kolejne akcje.



JAK NAWIGOWAĆ?

- **Pola** – zmienne dostępne w całej klasie. W przypadku POP można ich użyć jako podstrony/podekrany danej strony/ekranu. Zachowuje to naturalną hierarchię i układ.



PAGE FACTORY

- Posiada te same zalety co POP (bazuje na nim)
- Wprowadza dodatkowe atrybuty, takie jak FindBy, FindBys i CacheLookup

```
@FindBy(name = "q")
private WebElement searchBox;
```

- Inicjalizacja elementów jest leniwa (lazy)
- Problem ze sprawdzeniem czy element **nie** istnieje
- Wymaga inicjalizowania strony

```
// and initialise any WebElement fields in it.
GoogleSearchPage page = PageFactory.initElements(driver, GoogleSearchPage.class);
```

DZIĘKUJEMY