Programming Principles in Test Automation

Andreea Drăniceanu
October 24th

Diamond Sponsors:

SIEMENS
Ingenuity for life

Atos

nagarro

Trimble

Raiffeisen BANK

Silver Sponsors:

accenture

endava

Cerner

cegeka

Parteners:

Universitatea Transilvania din Brașov

About me

- Moved to Brașov for the mountain 🙂
- Have tested web apps, desktop apps, APIs
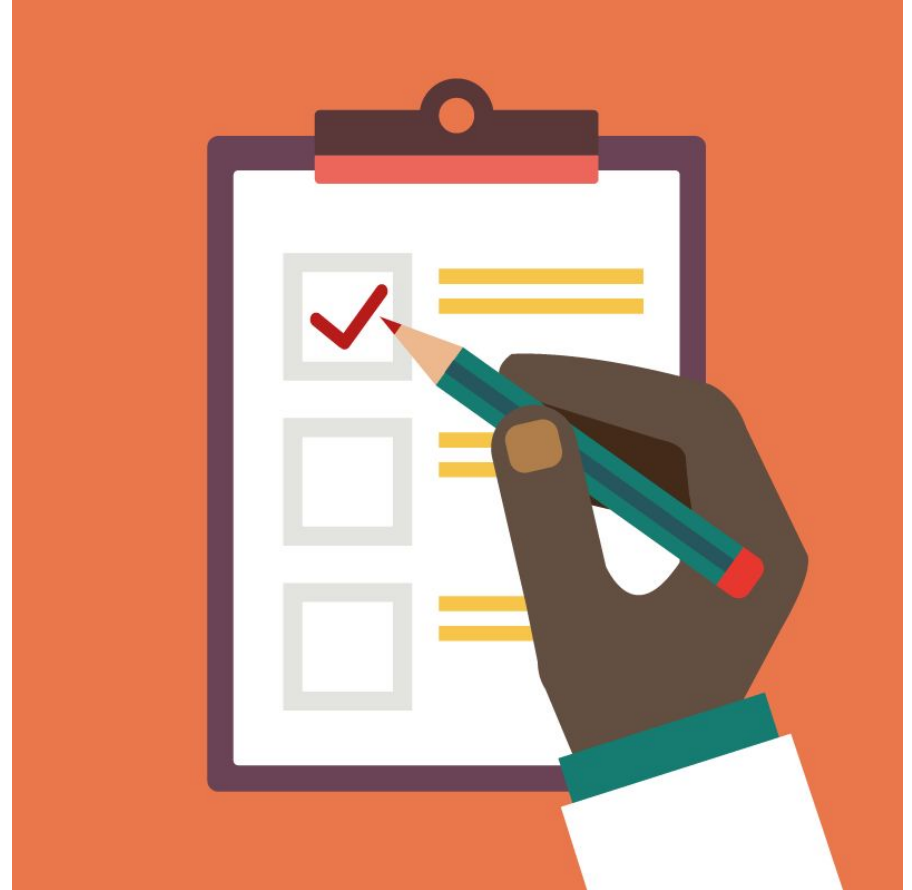- With AgileHub since 2020

Let's get to know each other

- www.menti.com

Agenda

- What is Page Object Model
- Basic OOP principles with examples
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction
- Some good practices in test automation

Communication

- Please turn on the camera if you have one
- Unmute yourself for questions, or comments
- We can also talk on [Slack](#) during and after the workshop
- The presentation slides will be provided after the workshop
- The code will be available on GitHub

# About POM

What is POM (Page-Object Model)

- A common design pattern used in UI automation
- Each application page has its own page object class
- The page object code is separated from the test code
- Page object classes include elements on a specific page and the actions that can be performed on them
- Benefits include: reduces code duplication, is easy to maintain, is easy to read, code can be reused

OOP Principles

OOP principles: Encapsulation

- Hiding the internal state and functions of an object a
- It's used to restrict the direct access to some of the object's components
- The fields of the class can be made read-only or write-only
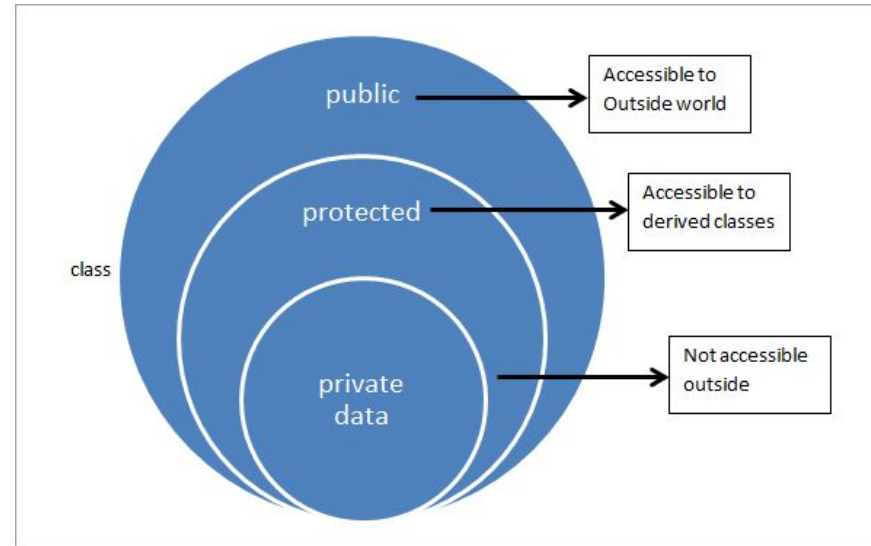
Encapsulation example

```
class Dog
{
    private string name;

    2 references
    public String Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

```
Dog myDog = new Dog
{
    Name = "Rudie"
};
Console.WriteLine(myDog.Name);
```

Encapsulation in POM - encapsulated class

```
1 reference
private IWebElement NameInput => driver.FindElement(By.Id("txtUsername"));
1 reference
private IWebElement PasswordInput => driver.FindElement(By.Id("txtPassword"));
1 reference
private IWebElement LoginButton => driver.FindElement(By.Id("btnLogin"));
```

```
public BasePage Login(string username, string password)
{
    NameInput.SendKeys(username);
    PasswordInput.SendKeys(password);
    LoginButton.Click();
    return new BasePage(driver);
}
```

Encapsulation in POM - accessing the methods

```
public void NavigateToDashboardPage()
{
    LoginPage loginPage = new LoginPage(driver);
    AdminPage adminPage = new AdminPage(driver);
    loginPage.OpenLoginPage();
    loginPage.Login("admin", "admin123");
    Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"));
    adminPage.OpenDashboard();
}
```

Encapsulation Q&A

OOP principles: Inheritance

- One object acquires (**inherits**) the properties and behaviors of the parent object
- The public and protected members of the parent class can be reused without having to define them again
- This means the code is reusable and requires less maintainnance

Inheritance example

```
// Parent class
2 references
public class Animal
{
    0 references
    public void Eat() => Console.WriteLine("Eating");

    0 references
    public void Sleep() => Console.WriteLine("Sleeping");
}
```

Inheritance example

```csharp
// Child classes
1 reference
public class Dog : Animal
{
    0 references
    public void Bark() => Console.WriteLine("Barking");
}


1 reference
public class Cat : Animal
{
    0 references
    public void Meow() => Console.WriteLine("Meowing");
}
```

Inheritance example

```
static void Main(string[] args)
{
    Dog dog = new Dog();
    Cat cat = new Cat();

    dog.Bark();
    dog.Eat();
    dog.Sleep();

    cat.Meow();
    cat.Eat();
    cat.Sleep();
}
```

Inheritance in POM

- Some pages can have common elements (e.g. header, side menus etc.):

Inheritance in POM

- We can use a parent class (e.g. a base page) inherited by the other pages
- The methods inside the class can be used from the derived classes

Inheritance in POM - base page

```csharp
public class BasePage
{
    protected readonly IWebDriver driver;

    3 references
    public BasePage(IWebDriver driver)
    {
        this.driver = driver;
    }

    1 reference
    private IWebElement DashboardMenuLink => driver.FindElement(By.Id("menu_dashboard_index"));

    1 reference | ⊘ 1/1 passing
    public DashboardPage OpenDashboard()
    {
        DashboardMenuLink.Click();
        return new DashboardPage(driver);
    }
}
```

Inheritance in POM - derived page

```csharp
class AdminPage : BasePage
{
    1 reference | ⊘ 1/1 passing
    public AdminPage(IWebDriver driver) : base(driver) {  }

    1 reference
    private IWebElement LoggedUser => driver.FindElement(By.Id("welcome"));

    1 reference | ⊘ 1/1 passing
    public bool IsUserLoggedIn(string username)
    {
        return LoggedUser.Text.Contains(username);
    }
}
```

Inheritance in POM - test

```
[Test]
⊘ | 0 references
public void NavigateToDashboardPage()
{
    LoginPage loginPage = new LoginPage(driver);
    AdminPage adminPage = new AdminPage(driver);
    loginPage.OpenLoginPage();
    loginPage.Login("admin", "admin123");
    Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"));
    adminPage.OpenDashboard();
}
```

Inheritance Q&A

OOP principles: Polymorphism

- Polymorphism = that takes many forms
- In C#, there are two types:
  - Compile Time Polymorphism (aka **overloading**)
  - Run Time Polymorphism (aka **overriding**)

Overloading

- Allows us to have multiple definitions for the same method
- The methods must have different signatures

Overloading Example

```csharp
1 reference
public int CalculateSum(int a, int b)
{
    return a + b;
}

1 reference
public double CalculateSum(double a, double b)
{
    return a + b;
}

1 reference
public int CalculateSum(int a, int b, int c)
{
    return a + b + c;
}
```

```csharp
Polymorphism p = new Polymorphism();

Console.WriteLine(p.CalculateSum(1, 2));
Console.WriteLine(p.CalculateSum(1.1, 2.2));
Console.WriteLine(p.CalculateSum(1, 2, 3));
```

```
3
3.3000000000000003
6
```

Overloading Example #2: Assert class in NUnit

```
//
// Summary:
//     Asserts that a condition is true. If the condition is false the method throws
//     an NUnit.Framework.AssertionException.
//
// Parameters:
//   condition:
//     The evaluated condition
//
//   message:
//     The message to display in case of failure
//
//   args:
//     Array of objects to be used in formatting the message
public static void IsTrue(bool condition, string message, params object[] args);
public static void IsTrue(bool? condition);
public static void IsTrue(bool condition);
public static void IsTrue(bool? condition, string message, params object[] args);
```

```
Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"));
Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"), "The user is not logegd in");
```

Overloading in POM

```csharp
1 reference | ◐ 0/1 passing
public BasePage Login(string username, string password)
{
    NameInput.SendKeys(username);
    PasswordInput.SendKeys(password);
    LoginButton.Click();
    return new BasePage(driver);
}


1 reference | ✔ 1/1 passing
public LoginPage Login(string username)
{
    NameInput.SendKeys(username);
    LoginButton.Click();
    Console.WriteLine(ErrorMessage.Text);
    return this;
}
```

```csharp
[Test]
◐ | 0 references
public void DashboardPageTest()
{
    LoginPage loginPage = new LoginPage(driver);
    AdminPage adminPage = new AdminPage(driver);
    loginPage.OpenLoginPage();
    loginPage.Login("admin", "admin123");
    Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"));
    Assert.IsTrue(adminPage.IsUserLoggedIn("Paul"), "The user is not logegd in");
    adminPage.OpenDashboard();
}


[Test]
✔ | 0 references
public void InvalidLoginTest()
{
    LoginPage loginPage = new LoginPage(driver);
    loginPage.OpenLoginPage();
    loginPage.Login("admin");
    Assert.IsTrue(loginPage.IsMessageDisplayed("Password cannot be empty"));
}
```

Overriding

- Allows us to rewrite a method (or properties, events, indexers) from the base class in a derived class, with a different implementation
- In the base class, the method must be **virtual**
- You cannot override a non-virtual method

Overriding Example

```csharp
public class Animal
{
    5 references
    public virtual void Eat() => Console.WriteLine("Eating");

    2 references
    public void Sleep() => Console.WriteLine("Sleeping");
}
```

```csharp
public class Dog : Animal
{
    1 reference
    public void Bark() => Console.WriteLine("Barking");
    5 references
    public override void Eat() => Console.WriteLine("Eating chicken");
}


1 reference
public class Cat : Animal
{
    1 reference
    public void Meow() => Console.WriteLine("Meowing");
    5 references
    public override void Eat() => Console.WriteLine("Eating salmon");
}
```

Overriding in POM

```csharp
// Method in base page
2 references | ✓ 1/1 passing
public virtual bool IsTitleCorrect()
{
    return driver.Title.Equals(title);
}
```

```csharp
// Method in derived page
2 references
public override bool IsTitleCorrect()
{
    return driver.Title.Equals("OrangeHRM");
}
```

Polymorphism Q&A

OOP principles: Abstraction

- The process of moving the focus from the concrete implementation of things, to the types of things (i.e. classes), the operations available (i.e. methods)
- Can be achieved through abstract classes and interfaces
- Abstract classes allow concrete implementation, while interfaces do not

Abstract Class Example

```
abstract class Animal
{
    4 references
    public abstract void Eat();

    4 references
    public abstract void Talk();
}
```

```
class Dog : Animal
{
    4 references
    public override void Eat() => Console.WriteLine("Eating chicken");
    4 references
    public override void Talk() => Console.WriteLine("Barking");
}
```

Abstract Class Example

```csharp
static void Main(string[] args)
{
    Dog dog = new Dog();
    Cat cat = new Cat();

    dog.Talk();
    dog.Eat();

    cat.Talk();
    cat.Eat();
}
```

Interface Example

```
interface IAnimal
{
    4 references
    public void Eat();

    4 references
    public void Talk();
}
```

```
class Dog : IAnimal
{
    4 references
    public void Eat() => Console.WriteLine("Eating chicken");
    4 references
    public void Talk() => Console.WriteLine("Barking");
}
```

Interface Example

```csharp
static void Main(string[] args)
{
    Dog dog = new Dog();
    Cat cat = new Cat();

    dog.Talk();
    dog.Eat();

    cat.Talk();
    cat.Eat();
}
```

Abstraction in POM

```
3 references | ◉ 0/1 passing
public abstract bool IsTitleCorrect();
```

```
public override bool IsTitleCorrect()
{
    return driver.Title.Equals("OrangeHRM");
}
```
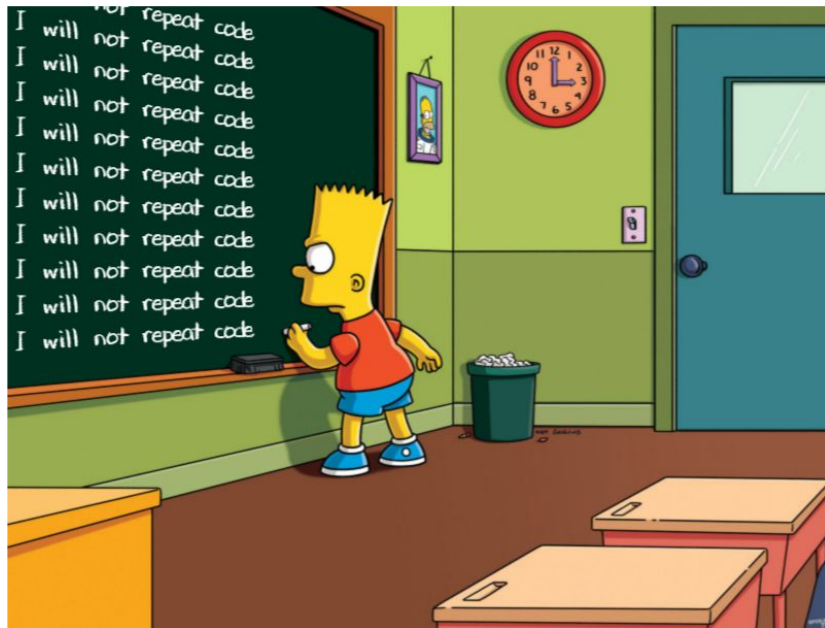
Abstraction Q&A

Good Practices in Test Automation

DRY (Don't Repeat Yourself)

- Makes code easy to maintain
- The code is (usually) easier to read
- The code can be reused

YAGNI (You Ain't Gonna Need It)

- Part of the XP philosophy
- Saves time
- Decreases maintenance work

KISS (Keep It Simple Stupid)

- Make code simple
- Avoid unnecessary complexity
- Use readable names, that express what the purpose of the classes and their members are

Single responsibility principle

- The S in SOLID
- Classes should have one responsibility, therefore only one reason to change
- In tests, this will help isolate the cause of a test failure

Other good practices

- Respect naming conventions of the used programming language
- Keep your tests (as) independent (as possible)
- Keep classes and methods small
- If performing UI automation, choose the right locators

Other good practices - continued

- Limit the number of assertions (there should be 1-2 per test)
- Avoid using Thread.Sleep()
- If working with Selenium, avoid mixing Implicit and Explicit waits
- Keep assertions inside the tests, not the page object classes

What are some of your good practices that we didn't discuss?

Q&A

Workshop Feedback

http://bit.ly/peakit004-feedback

Completați acum

Durează 2-3 minute

Feedback anonim - pentru formator și AgileHub

Thank you!