Autor:

Kamil Marek

Kamil.Marek.qa@gmail.com

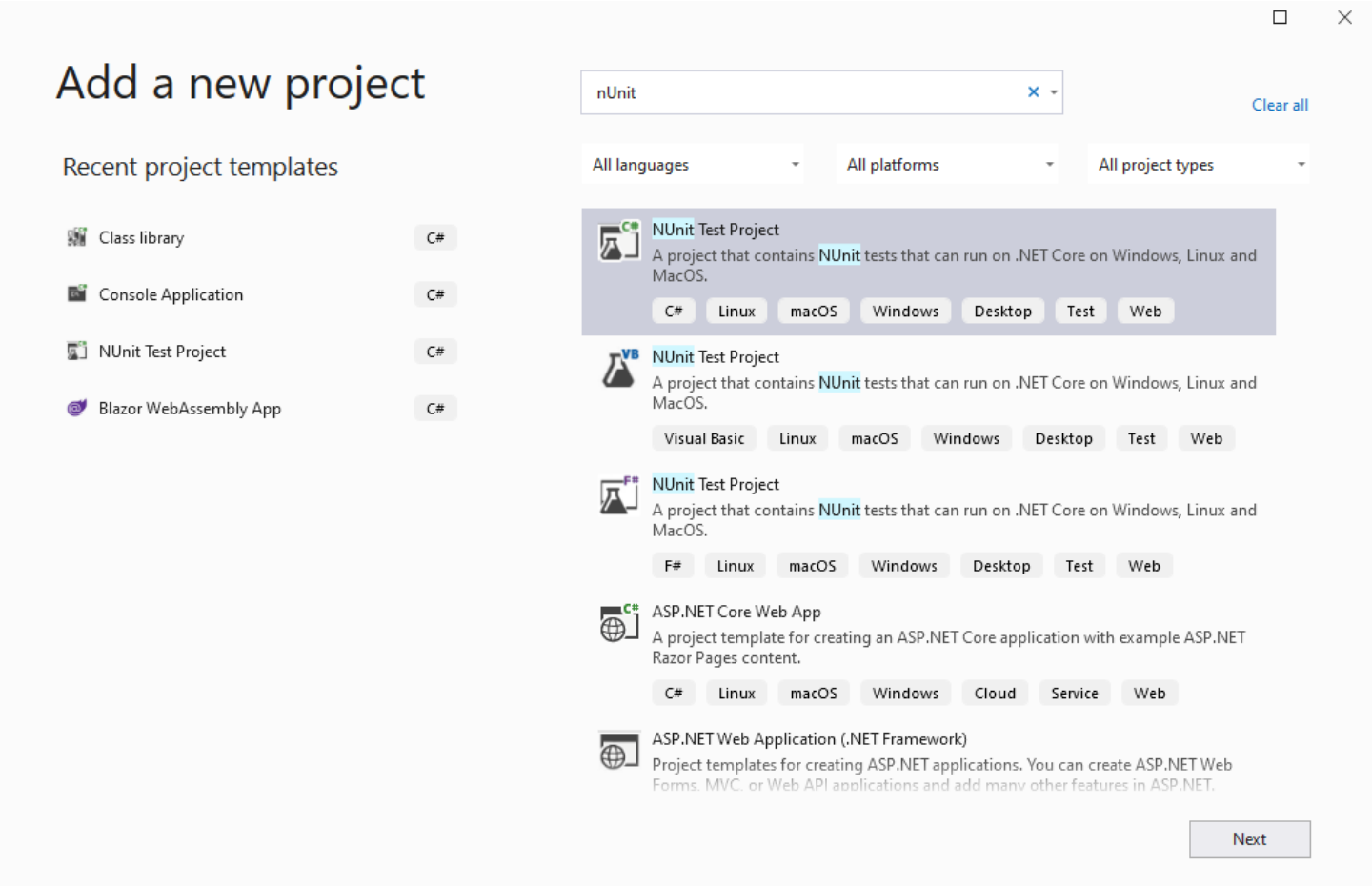
Wersja 1.0.0

Spis Treści

LAB 1 – nUnit

Utworzenie nowego projektu testów

1. Utwórz nową solucję, wybierz szablon o nazwie **nUnit Test Project**.



2. Wpisz **Testerzy.Trainings.CSharp.Tests** jako Project name oraz **Testerzy.Trainings.CSharp jako nazwa solucji**
3. Wybierz .NET 8.0 jako wersję .NET
4. Po utworzeniu projektu, uaktualnij wersję wszystkich zainstalowanych pakietów NuGet

Utworzenie pierwszego testu

1. W projekie **Testerzy.Trainings.CSharp.Tests** dodaj katalog Lab1 a w nim publiczną klasę FirstTests
2. W klasie FirstTests dodaj zależność do NUnit.Framework (using)
3. Dodaj metodę testową o nazwie TestMultiplyOperatorOnIntNumbers
4. Zweryfikuj czy działanie 5*7 daje poprawny wynik
5. Uruchom test w Test explorer

LAB 2.A – Klasy, obiekty, metody i pola

1. Dodaj katalog **Lab2A** w nim następujące klasy
    a. Publiczna klasa UserRequest, która zawiera publiczne pole **userId** typu int
    b. Publiczna klasa Client, która zawiera publiczną metodę SendRequest (metoda zwraca int a przyjmuje jeden argument typu UserRequest) – zaimplementuj metodę tak by zwracała 200, jeśli pole **userId** w obiekcie przekazanym do metody jest większe niż 0 a liczbę 404 w przeciwnym wypadku.
    c. Dodaj publiczną klasę Lab2ATests
    d. Dodaj 2 testy które przetestują metodę SendRequest dla oczekiwanego rezultatu 200 i 404

LAB 2.B – Konstruktory

1. Dodaj katalog **Lab2B**
2. W katalogu dodaj nową klasę o nazwie Person
   W klasie Person dodaj 2 publiczne pola typu string o nazwach firstName i lastName
3. Dodaj konstruktor, który umożliwi przypisanie imienia i nazwiska w momencie tworzenia obiektu. Pozostaw możliwość tworzenia instancji bez parametrów.
4. Dodaj klasę Lab2BTests i utwórz w niej 2 testy, w których przetestujesz tworzenie obiektów klasy Person używając różnych konstruktorów (w asercjach sprawdzaj wartości pól po utworzeniu obiektu)

   np. dla konstruktora bez parametrów domyślna wartość pola typu string to null
   możesz to zweryfikować używając asercji `Assert.That(person.firstName, Is.Null);`

LAB 2.C – Dziedziczenie

1. Dodaj katalog **Lab2C**
2. W katalogu dodaj nową klasę o nazwie Person
   W klasie Person dodaj 2 publiczne pola typu string o nazwach **firstName** i **lastName**
3. W katalogu Lab3_4 dodaj nową klasę o nazwie Student, która dziedziczy po klasie Person
   W klasie Student dodaj publiczne pole typu string
4. Dodaj klasę Lab2CTests a w niej test, w którym utwórz obiekt klasy Person oraz obiekt klasy Student. Każdemu obiektowi przypisz dowolne wartości do wszystkich dostępnych pól
5. Zweryfikuj wartości pól

LAB 2.D – Konstruktory a dziedziczenie

1. Dodaj katalog **Lab2D**
2. Dodaj klasę BaseClient
    a. dodaj prywatne (bez public) pole _baseUrl typu string
    b. konstruktor, który przyjmie 1 parametr typu string a następnie przekazaną wartość przypisze do zmiennej _baseUrl - dodaj w konstruktorze wypisanie przekazanej zmiennej na konsole
    c. dodaj metodę GetBaseUrl, która zwróci wartość pola _baseUrl
    d. Dodaj klasę AccountsClient, która dziedziczy po klasie BaseClient
       - dodaj konstruktor, który przyjmie 1 parametr typu string a następnie użyje go w wywołaniu kontruktora klasy bazowej
    e. Dodaj klasę Lab2DTests a w niej metodę testową, który utworzy obiekt klasy AccountsClient z dowolnym urlem. Zweryfikuj wartość url poprzez metodę GetBaseUrl

f. Uruchom test i sprawdź zalogowany url

[Zadanie dodatkowe]

3. Dodaj klasę ProfilesClient, która dziedziczy po klasie BaseClient
    a. dodaj analogiczny konstruktor jak w klasie AccountsClient
    b. dodaj drugi konstruktor, który przyjmie 2 parametry typu string: url i resource a następnie wywoła bazowy kontruktor przekazując sumę tych parametrów (url + response)
    c. W klasie Lab2DTests dodaj nowy test - utwórz obiekt ProfilesClient przekazując 1 parametr o wartości "https://api.example.pl/profiles/". Zweryfikuj wartość url poprzez metodę GetBaseUrl .
    d. Dodaj kolejny test, utwórz obiekt ProfilesClient przekazując 2 parametry o wartości "https://api.example.pl/" oraz "profiles/" Zweryfikuj wartość url poprzez metodę GetBaseUrl
    e. Uruchom testy i sprawdz, że w obu przypadkach w konstruktorze BaseClient zostanie wypisany ten sam url

LAB 2.E – Modyfikatory dostępu

1. Dodaj katalog **Lab2E**
2. W katalogu dodaj nową klasę o nazwie BankAccount
   W klasie BankAccount dodaj prywatne pole typu decimal o nazwie _balance
3. Dodaj publiczną metodę o nazwie Deposit, która przyjmuje jeden argument typu decimal o nazwie amount
   W kodzie metodzie dodaj wartość amount do aktualnego stanu konta (_balance)
4. Dodaj publiczną metodę o nazwie GetBalance, która nie przyjmuje parametrów i zwraca decimal
5. Dodaj klasę **Lab2ETests** a w niej test, w którym utwórz obiekt typu BankAccount,
   pobierz stan konta, wpłać kwotę 100 i sprawdź czy aktualny stan konta się zwiększył.

LAB 2.F – Właściwości

1. Dodaj katalog **Lab2F**
2. W katalogu dodaj nową klasę o nazwie Person
   W klasie Person dodaj publiczną właściwość typu string o nazwie **FirstName**, ustaw setter jako prywatny
   Dodaj konstruktor, który przyjmie jeden argument typu string, który przypisze do właściwości FirstName
    a. Dodaj klasę **Lab2FTests**, w niej test, który utworzy nowy obiekt Person i sprawdzi wartość FirstName
    b. Sprawdź, że nie da się nadpisać FirstName z poziomu metody testowej

[Zadanie dodatkowe]

3. Dodaj nową klasę AccountResponse
   Dodaj właściwości o odpowiednich nazwach i typach, tak by zamodelować odpowiedź API na podstawie przykładowej odpowiedzi w formacie JSON

```
{
"email" : "user@email.com",
"username" : "user12",
"firstName" : "John",
"lastName" : "Doe",
"age" : 56,
"isAdmin" : false

}
```

LAB 2.G – Polimorfizm

1. Dodaj katalog **Lab2G**
    a. Utwórz klasę RestRequest z publiczną właściwością Url typu string
    b. Utwórz klasę RestClient
    c. W klasie RestClient utwórz metodę o nazwie Send, która przyjmuje 1 parametr typu RestRequest a zwraca typ string. Zaimplementuj metodę by zwracała wartość Url z przekazanego argumentu
    d. utwórz kolejną metodę o nazwie Send, która przyjmuje 2 parametry - 1 typu RestRequest a drugi typu string o nazwie method. Metoda zwraca string (połączoną wartość method i Url)
    e. Dodaj klasę **Lab2GTests** i test, w których utwórz obiekt RestRequest a następnie obiekt RestClient i wywołaj na nim obie wersje metody Send z odpowiednimi argumentami.
    f. Zweryfikuj zwracany tekst dla obu metod

LAB 2.H – Kompozycja

2. Dodaj katalog **Lab2H**
3. W katalogu dodaj nową klasę o nazwie **BankAccount**
   W klasie BankAccount dodaj publiczną właściwość typu string o nazwie **AccountNumber**
4. W katalogu dodaj nową klasę o nazwie Person
   W klasie Person dodaj publiczne właściwości:
   string FirstName
   string LastName
   BankAccount BankAccount
5. Dodaj klasę **Lab2HTests** w niej test, w którym utwórz nowy obiekt Person i przypisz mu imię, nazwisko i numer konta a następnie zweryfikuj te wartości na obiekcie

LAB 2.I – Elementy statyczne

1. Dodaj katalog **Lab2I**
2. W katalogu dodaj nową klasę o nazwie BankAccount
   W klasie BankAccount dodaj publiczną właściwość typu string o nazwie AccountNumber
3. Dodaj nową statyczną klasę DataHelper
   Dodaj statyczną metodę o nazwie GetValidBankAccount.
   Zaimplementuj metodę tak by zwracała obiekt typu BankAccount z numerem konta PL12345678901234567890
4. Dodaj klasę o nazwie **Lab2ITests**
   Dodaj test, w którym utwórz zmienną i przypisz do niej obiekt typu BankAccount przy pomocy metody GetValidBankAccount, a następnie zweryfikuj numer konta tego obiektu

LAB 3.A – Wyjątki

4. Dodaj katalog **Lab3**
5. Dodaj klasę BaseClient
    a. Dodaj prywatne pole string _baseUrl

b. Dodaj konstruktor, który przyjmie 1 parametr typu string, sprawdzi czy wartość jest null lub pusta (możesz skorzystać z string.IsNullOrEmpty(). Jeśli tak, rzuci odpowiednim wyjątkiem, jeśli nie, przypisze wartość do pola _baseUrl
c. Dodaj klasę **Lab3Tests** oraz testy, które sprawdzą tworzenie obiektu BaseClient dla różnych wartości przekazanych w konstruktorze
d. Uruchom testy i sprawdź statusy

LAB 3.B – Obsługa wyjątków

1. W katalogu **Lab3**
2. W klasie Lab3Tests zmodyfikuj testy, aby przechwycić ewentualne wyjątki przy tworzeniu obiektów
    a. Przed blokiem try zadeklaruj zmienną typu bool exceptionThrown i przypisz jej wartość false.
    b. W bloku catch przypisz wartość true do zmiennej exceptionThrown
    c. Poza blokiem try-catch zweryfikuj wartość zmiennej exceptionThrown

LAB 3.C – Własne wyjątki

1. W katalog **Lab3** dodaj własny wyjątek **ConfigurationException** wraz ze standardowymi konstruktorami
    a. W konstruktorze klasy **BaseClient** dodaj kod, który sprawdzi dodatkowo czy przekazany argument jest poprawnym URLem (możesz skorzystać z metody **Uri.IsWellFormedUriString(baseUrl, UriKind.Absolute)**, która zwraca wartość bool określającą poprawność URL)
    b. W przypadku niepoprawnego URL, rzuć wyjątkiem **ConfigurationException**
    c. Dodaj test(y) w klasie **Lab3Tests**, który zweryfikuje przypadek z niepoprawnym URL przy tworzeniu obiektu BaseClient

LAB 4.A – Typy wyliczeniowe

1. Dodaj katalog **Lab4**
2. W katalogu dodaj nową klasę o nazwie Month
3. W pliku Month.cs zamień deklarację klasy na deklarację typu wyliczeniowego Month (public enum Month **zamiast** public class Month). Dodaj wszystkie 12 miesięcy tzn January, February itp
4. Dodaj klasę **Lab4Tests**
5. Dodaj metodę testową, w której zweryfikujesz czy dowolnie wybrany miesiąc ma przypisaną odpowiednią liczbę np. czy Month.January ma liczbę 0

LAB 4.B – Typy wyliczeniowe

1. W katalogu **Lab4** Dodaj klasę **PriceCalculator**
2. Dodaj publiczną statyczną metodę o nazwie GetPrice, która zaimplementuje poniższe wymagania
   Wejście: enum Month
   Działanie: zwróć cenę wynajmu domku letniskowego (int) w zależności od miesiąca

   Maj, Czerwiec, Wrzesień – 300
   Lipiec, Sierpień – 400
   Pozostałe miesiące – 200

3. W klasie **Lab4Tests** dodaj testy, w których wykonasz metodę GetPrice dla 3 miesięcy z różnych przedziałów cenowych i zweryfikujesz zwróconą cenę

LAB 5.A – nUnit

1. W **Testerzy.Trainings.CSharp.Tests** dodaj katalog **Lab5**
2. Dodaj w nim klasę Calculator
3. Dodaj metodę Add, która przyjmuje 2 argumenty typu decimal a zwraca ich sumę jako decimal
4. W katalogu Lab5 dodaj klasę **CalculatorTests**
5. Korzystając z atrybutu TestCase, dodaj testy, które zweryfikują działanie metody Add dla różnych kombinacji 2 dowolnych liczb z przedziałów (liczby ujemne, liczby dodatnie, zero)
6. Uruchom testy

[Zadanie dodatkowe]

1. W katalogu Lab5 dodaj klasę PasswordValidator z kodem poniżej:

```
using System.Text.RegularExpressions;

namespace Testerzy.Trainings.CSharp.Lab5;

public class PasswordValidator
{
	private const string pattern = @"^(?=.*\d)(?=.*[A-Z])(?=.*[a-z])(?=.*[^\w\d\s:])([^\s]){8,15}$";

	public bool IsPasswordValid(string password)
	{
		return Regex.IsMatch(password, pattern);
	}
}
```

Powyższy kod potraktuj jako **czarną skrzynkę**, która implementuje poniższe wymagania dotyczące hasła:
password must contain at least 1 number (0-9)
password must contain 1 uppercase letters
password must contain 1 lowercase letters
password must contain 1 non-alpha numeric character
password is 8-16 characters with no space

1. Dodaj klasę PasswordValidatorTests
2. **Zaprojektuj** testy (wartość wejściowa – wartość oczekiwana) na podstawie wymagań powyżej i a następnie zautomatyzuj je w klasie PasswordValidatorTests
3. Uruchom testy

LAB 5.B – Setup i Teardown

1. Do klasy **Calculator** dodaj metodę **Subtract**, która przyjmuje 2 argumenty typu decimal a zwraca ich różnicę jako decimal
2. W klasie **CalculatorTests** dodaj testy, które przetestują działanie tej metody

3. Kod, w którym tworzysz obiekt Calculator przenieś do metody oznaczonej [OneTimeSetUp]
4. Uruchom wszystkie testy kalkulatora

LAB 5.C – Ignore / Repeat / Retry

1. Dodaj klasę testową Lab5CTests
2. Dodaj test VerifyIgnore, który dodatkowo oznacz atrybute Ignore
3. Dodaj test VerifyRepeat, który oznacz atrybutem Repeat oraz dowolną liczbą powtórzeń
   W teście dodaj wypisywanie dowolnego tekstu na konsolę
4. Uruchom testy w klasie Lab5Tests i sprawdź w raporcie statusy i Standard Output
5. Dodaj test VerifyRetry wraz z metodą OneTimeSetup i polem count jak poniżej:

```
private static int count;

[OneTimeSetUp]
public void OneTimeSetup()
{
	count = 0;
}

[Test, Retry(1)]

public void VerifyRetry()
{
	Console.WriteLine($"verify retry count={count}");
	count++;
	Assert.That(count, Is.EqualTo(2));
}
```

6. Uruchom test i potwierdź, że status == Failed
7. Zmień liczbę potworzeń na 2 i uruchom jeszcze raz, potwierdź, że teraz status == Passed

LAB 5.D – Property

1. Oznacz dowolne testy z klas **Lab5CTests** i **CalculatorTests** atrybutami **Property.**
   W **Test Explorer** Uruchom wybrane testy na podstawie wybranych filtrów

LAB 5.E – TestContext

1. W klasie **Lab5CTests** dodaj metodę oznaczoną atrybutem [TearDown]
2. Wypisz na konsolę nazwę testu oraz jego status
3. W przypadku statusu Failed, dodatkowo wypisz tekst „Test failed”
4. Uruchom testy i sprawdź w oknie Test Detail Summary Standar Output dla każdego testu

LAB 6.A – Atrybuty

1. Dodaj projekt typu Class Library o nazwie **Testerzy.Trainings.CSharp.Framework**
2. W projekcie **Testerzy.Trainings.CSharp.Tests** dodaj zależność do projektu *.Framework
3. W projekcie *.Framework dodaj katalog **Attributes**
4. W katalogu **Attributes** dodaj nową klasę DescriptionAttribute, która dziedziczy po klasie Attribute (wymagany using System)
5. W klasie DescriptionAttribute dodaj publiczną właściwość Description (typu string) z prywatnym setterem
6. Dodaj konstruktor przyjmujący 1 argument typu string, który przypisze do właściwości Description
7. Ogranicz użycie atrybutu do metod
8. Oznacz dowolny test atrybutem [Description]

LAB 6.B – Atrybuty / Property

1. W projekcie *.Framework dodaj katalog Enums
2. W katalogu Enums dodaj publiczny enum o nazwie TestTypeName wraz z dwoma wartościami: UI i API
3. Do projektu dodaj zależność NUnit poprzez NuGet Package Manager
4. Do pliku projektu *.Framework dodaj element

```
<PropertyGroup>
            <IsTestProject>false</IsTestProject>
</PropertyGroup>
```

5. W katalogu Attributes dodaj klasę TestTypeAttribute
6. Dodaj konstruktor, który przyjmie 1 parametr typu TestType a następnie nazwę filtra „TestType” oraz jego wartość zamienioną na string do konstruktora bazowego
7. Ogranicz użycie atrybutu do metod, pozwól na wielokrotne użycie
8. Oznacz testy w klasie CalculatorTests atrybutem TestType + różnymi wartościami
9. Uruchom testy w Test Explorer na podstawie wartości TestTypeName

LAB 7.A – Słowniki

1. W projekcie **Testerzy.Trainings.CSharp.Tests** dodaj katalog **Lab7A**
    a. W katalogu dodaj nową klasę DictionaryTests
    b. Dodaj metodę testową, w której zadeklaruj zmienną o nazwie dictionary typu słownik o kluczach typu **string** i wartościach typu **string**
    c. Dodaj element do słownika - klucz: **Jan Kowalski**, wartość: **jan@kowalski.pl**
    d. Dodaj kolejny element do słownika - klucz: **Maria Nowak**, wartość: **maria@nowak.pl**
    e. Dodaj kod, który wypisze klucze i wartości wszystkich elementów słownika
    f. Dodaj asercję sprawdzającą liczbę elementów w słowniku

2. W katalogu **Lab7A** dodaj klasę **ErrorRegistry**
    a. W klasie dodaj prywatne pole typu słownik o kluczach typu int i wartościach typu string
    b. Dodaj **konstruktor,** w którym zaincjalizujesz słownik oraz dodasz 2 elementy - o kluczu 1012 i wartości Invalid token oraz o kluczu 1013 i wartości Invalid user
    c. Dodaj publiczną metodę GetError, która ma 1 parametr typu int oraz zwraca string
    d. Zaimplementuj metodę tak by zwracała wartość ze słownika na podstawie przekazanego klucza Jeśli klucz nie istnieje, zwróć pusty tekst
    e. Dodaj testy, które sprawdzą działanie metody GetError dla poprawnej i niepoprawnej wartości

LAB 7.B – Listy

1. W projekcie **Testerzy.Trainings.CSharp.Tests** dodaj katalog **Lab7B**
    a. W katalogu dodaj nową klasę ListTests
    b. Dodaj metodę testową w której utwórz zmienną typu lista liczb całkowitych i przypisz listę z elementami 1,2,3,4
    c. Dodaj nowy element o dowolnej wartości
    d. Usuń z listy element o wartości 3
    e. Dodaj asercję sprawdzającą długość listy

    2. Dodaj klasy o nazwach **Settings**, **Environment** i **User**
        a. Zaimplementuj w nich publiczne właściwości, które modelują JSONa

```
{
    "environment": {
            "name": "dev"
    },
    "users": [
            {
                    "email": "user@app.com",
                    "isAdmin": false
            },
            {
                    "email": "admin@app.com",
                    "isAdmin": true
            }
    ]
}
```

b. W klasie ListTests dodaj metodę testową, w której przypisz powyższego JSONa do zmiennej typu string
c. Zdeserializuj tekst do obiektu typu Settings

wymaga `using System.Text.Json;`

```
Settings settings = JsonSerializer.Deserialize<Settings>(json, new
JsonSerializerOptions() { PropertyNameCaseInsensitive = true });
```

gdzie **json** to zmienna typu `string`

a. Zweryfikuj, że emaile pierwszego i drugiego usera zostały poprawnie wczytane do obiektu

LAB 10.A – Metody rozszerzające

1. W projekcie **Testerzy.Trainings.CSharp.Tests** dodaj katalog **Lab10A**
   a. W katalogu dodaj klasę **StringExtensions** a w niej metodę o nazwie ToInt, która rozszerzy typ string
   b. Zaimplementuj metodę, aby konwertowała tekst na int i zwracała skonwertowaną wartość
   c. Dodaj klasę ExtensionTests wraz z testem, który zweryfikuje działanie metody ToInt
2. W klasie StringExtensions dodaj metodę rozszerzającą dla typu string,
   a. Zaimplementuj metodę tak, by zwróciła liczbę słów w danym ciągu tekstowym. Za słowo uznajemy ciąg znaków oddzielony spacjami.

   Wskazówka: Możesz użyć metody Split() do podzielenia tekstu na słowa.
   b. Dodaj test sprawdzający działanie tej metody

3. W klasie StringExtensions dodaj metodę rozszerzającą dla typu string, która sprawdzi czy dany tekst jest palindromem (ignoruj wielkość liter, spacje, znaki zapytania i przecinki).
   a. Metoda ma zwraca wartość typu bool. Dodaj testy sprawdzające kilka przypadków.

   Przykłady poprawnych palindromów:
   Anna
   Popija rum As, samuraj i pop
   Ada raportuje, że jutro parada
   Może jeż łka jak łże jeżom
   O, ty z Katowic, Iwo? Tak, Zyto
   Łapał za kran, a kanarka złapał.
   Elf układał kufle.