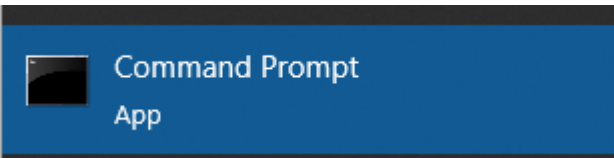


Vježba 5 - Testiranje složenih struktura / Uklanjanje vanjskih ovisnosti

Uvod

Dohvat koda za vježbu 5 iz git repozitorija



```
D:
cd TPP\Study
git fetch --all
git checkout app_study_vj5
git pull origin app_study_vj5
```

ZADATAK 1: Anonimna funkcija

Korištenjem lambda izraza napisati anonimnu funkciju za množenje dva cijela broja.

- Definirati delegat tipa `Func<T1, T2, TResult>` kao svojstvo klase `Util` imena `Multiply` te ga deklarirati lambda izrazom
- Napisati 3 testa za svojstvo `Multiply` u klasi `UtilTests`. Koristiti parametrizirane testove

ZADATAK 2: Testiranje polja

Napisati testove za metodu `GetOddNumbers` u klasi `Util`. Testirati je li:

- metoda vraća polje koje sadrži broj 1 ukoliko je parametar metode `limit` `>= 1`
- metoda vraća ispravan broj elemenata do gornje granice
- metoda vraća polje koje sadrži sve neparne brojeve u bilo kojem redoslijedu do gornje granice
- metoda vraća polje čiji su elementi poredani uzlazno
- metoda vraća polje čiji su elementi jedinstveni

ZADATAK 3: Testiranje tipova

Testirati tip svojstva `CityRepository` na klasi `CityController`

- Napraviti testnu klasu `CityControllerTests` u projektu `Study.UnitTests`
- Napisati testove za svojstvo `CityRepository`:
 - Vrijednost svojstva mora biti tipa `CityRepository`
 - Vrijednost svojstva mora biti instanca osnovnog tipa `Repository`

ZADATAK 4: Testiranje iznimki

Testirati metodu `Enroll` na klasi `Student`

- Testirati je li metoda baca iznimku tipa `ArgumentException` kada razine studenta i tečaja nisu kompatibilne

ZADATAK 5: Testiranje uvjeta

Promijeniti metodu `Introduce` u klasi `Student`:

- Ako je ime studenta dulje od 7 znakova na kraj rezultata dodati rečenicu: "I have a long name."
- Ako je razina studenta veća od 5 na kraj rezultata dodati rečenicu: "My knowledge level is high."

- Napisati testove za sve puteve izvršavanja metode

ZADATAK 6: Testiranje običnih petlji

Napisati testove za metodu `Factorial` u klasi `Util`

Pokriti slučajeve kada se petlja izvršava :

- $n = 0$
- $n = 1$
- $n = 2$
- $2 < n < \text{max} - 1$
- $n = \text{max} - 1$
- $n = \text{max}$
- $n = \text{max} + 1$ puta.

ZADATAK 7: Testiranje ugnježđenih petlji

Napisati testove za metodu `SumMatrix` u klasi `Util`

Pokriti slučajeve kada se unutarna petlja izvršava:

- $n = 0$
- $n = 1$
- $n = 2$
- $2 < n < \text{max} - 1$
- $n = \text{max} - 1$
- $n = \text{max}$
- $n = \text{max} + 1$ puta.
- Pokriti iste slučajeve i za vanjsku petlju

ZADATAK 8: Uklanjanje vanjskih ovisnosti

Testirati klasu `JsonDataAccess`. Klasa `JsonDataAccess` ovisi o klasama `JsonConvert`, `StreamReader` i `File`.

Ekstrahirati kod koji koristi vanjske ovisnosti u novu klasu `JsonUtil`:

- Napraviti metode
 - `List<T> DeserializeJson<T>(string json)` koja će koristiti klasu `JsonConvert`
 - `string SerializeJson(object value)` koja će koristiti klasu `JsonConvert`
 - `string ReadFile(string fileName)` koja će koristiti klasu `StreamReader`
 - `void WriteAllText(string fileName, string content)` koja će koristiti klasu `File`
- Pozive prema vanjskim ovisnostima na `JsonDataAccess` zamijeniti pozivima prema klasi `JsonUtil`
- Izraditi interface `IJsonUtil` na temelju klase `JsonUtil`
- Izmijeniti klasu `JsonDataAccess` da koristi interface umjesto konkretne klase `JsonUtil`
 - Koristiti dependency injection pomoću konstruktora dodavanjem parametra tipa `IJsonUtil`
 - Parametar treba biti neobavezan
 - Ukoliko parametar nije poslan instancirati konkretnu klasu `JsonUtil`
- Testirati klasu `JsonDataAccess`
 - Napraviti klasu `JsonDataAccessTests`
 - Testirati metodu `GetEntities()`
 - Napraviti novu lažnu klasu `FakeJsonUtil` koja implementira `IJsonUtil`
 - Programirati `FakeJsonUtil` tako da:
 - metoda `DeserializeJson` uvijek vraća praznu listu
 - metoda `ReadFile` uvijek vraća prazan string

- metoda `SerializeJson` uvijek vraća prazan string
 - metoda `WriteAllText` može ostati prazna
- Instancirati `JsonDataAccess` slanjem `FakeJsonUtil` instance u konstruktoru
- Pozvati `GetEntities`
- Provjeriti je li rezultat prazna lista
- Testirati metodu `Remove()`
 - Instancirati lažnu klasu koja implementira sučelje `IJsonUtil` pomoću biblioteke `Moq`
 - programirati instancu na isti način kao i `FakeJsonUtil`
 - instancirati `JsonDataAccess` s lažnom klasom
 - Pozvati metodu `Remove()`
 - Pomoću lažne klase provjeriti da je pozvana metoda `SerializeJson`

ZADATAK 9: Interakcijsko testiranje

Testirati klasu `StudentRepository`

Testirati metodu `GetStudentWithCourse(int id)`

- Pomoću biblioteke `Moq` Izraditi lažnu klasu koja implementira interface `IDataAccess`
 - Programirati lažnu klasu tako da:
 - metoda `GetById<Student>()` s bilo kojom vrijednosti identifikatora vraća instancu klase `Student` kojoj je `CourseId = 5`
 - `GetById<Course>(5)` vraća instancu klase `Course` kojoj je `Id = 5`
 - Instancirati `StudentRepository` s lažnom klasom
- Provjeriti je li metoda `GetById` pozvana dva puta:
 - s parametrom tipa `Student`
 - s parametrom tipa `Course`

Slanje vježbi na Moodle