Alkalmazásfejlesztés-II

WinForms technológia - Heroes alkalmazás

University of Szeged

2022. október 17.– 09:24

A leírás a gyakorlaton bemutatott `Items` entitást nem tartalmazza, hiszen az a `Hero` entitás megértése után már motorikus folyamat eredménye.

Projekt felépítése

Az előző gyakorlati dokumentumban található a feladat részletes leírása. Ennek megfelelően a projekt felépítése a következő:

- Controller - Az üzleti logikáért felel,
- DAO (Data Access Object) - A perzisztenciáért felel. Most az objektumok memóriában kerülnek mentésre, de későbbiekben ez SQL adatbázisra (vagy másra cserélhető),
- Model - A tárolt entitások specifikálása itt történik,
- View - A GUI-t tartalmazza,
- ViewModel - Ha nem ugyanazt az entitást szeretnénk megjeleníteni, mint ami el van tárolva, akkor azt ViewModellek segítségével lehet megtenni. Pl. Hero rendelkezik ID-vel, viszont ezt a felhasználó felé felesleges mutatni, ezért egy HeroViewModel osztályt létrehozunk és kihagyjuk az ID-t).

Model

A feladat leírása alapján a modellben kell lennie egy `Hero` osztálynak. Mappára jobb kattintás és `Add -> Class`. A Visual Studio létrehozza a fájlt és sablonszerűen kitölti.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Heroes.Model {
  class Hero { }
}
```

A feladat kiírása alapján az osztály property-jeit fel kell sorolni a megfelelő adattípusokkal. Megfigyelve a fájl elejét, látható, hogy az összes `using` szürke, azaz nincsenek használva. Ezek törölhető sorok – általános igazság, hogy a nem használt `using`-ok törölendők a fájlokból (kivétel az IDE által generált fájlok).

Egy másik észrevétel, hogy az osztály nem publikus. Ez most még nem okoz problémát, viszont amikor a View-ből kell elérni, akkor gond lesz, hogy egy publikus osztályból el szeretnénk érni egy internal-t, ezt a hibaüzenetet megelőlegezve írjuk a `class Hero` elé a `public` kulcsszót.

```
namespace Heroes.Model {
  public class Hero {
    public int ID { get; set; }
    public string Name { get; set; }
    public string HeroName { get; set; }
    public string Power { get; set; }
    public int Age { get; set; }
  }
}
```

DAO - Data Access Object

Amennyiben az alkalmazásnak tárolnia kell adatokat perzisztensen, akkor a DAO feladata ezt a funkcionalitást biztosítani. A feladat során biztosítja a hozzáadást, a listázást és a szűrést. A bevezetésben már említve lett, hogy most memóriában tárolunk, és szeretnénk lecserélni a későbbiekben adatbázisra. Ez úgy tehető meg, ha az alkalmazás a DAO implementációjától függetlenül tud működni, mely interfész segítségével érhető el.

Az interfész meghatározza, hogy mit kell tudnia a DAO-nak és az üzleti logika az interfészt fogja felhasználni. A DAO mappára jobb kattintás és `Add -> New Item -> Interface`. A fájl neve legyen `IHeroesDao`.

```
namespace Heroes.DAO {
  public interface IHeroesDao {
    bool AddHero(Hero hero);
    bool ModifyHero(Hero hero);

    int HeroesCount();
    Hero GetHero(int heroId);
    IEnumerable<Hero> GetHeroes();
  }
}
```

Az alkalmazásnak lehetősége van hozzáadni, módosítani és lekérni a hősöket, illetve lekérdezni a tárolt hősök számát. A különböző megvalósításoknak ezeket a függvényeket meg kell valósítani.

A megvalósítás hozzáadásához jobb klikk a DAO mappára `Add -> Class`. A fájl neve legyen `HeroesMemoryDao`. A létrejött osztály után írjuk a `: IHeroesDao` kódrészletet, így jelezve, hogy megvalósítja a jelölt interfészt. Miután ez megtörtént, a fejlesztőkörnyezet hibát jelez, ami annak jele, hogy nem történt még meg a megvalósítás. Ebben az esetben a kis égőre kattintva `Implement Interface` menüpontot kiválasztva az IDE legenerálja az összes hiányzó függvényt.

A teljes implementáció megtalálható a `/pub/Alkalmazasfejlesztes-II/gyakorlat` mappában.

Controller

Amit az alkalmazás *tud*, azt a controller biztosítja. Konstruktorában vár egy DAO megvalósítást, a paraméter maga `IHeroesDao` típusú és az itt kapott objektum fogja meghatározni, hogy az alkalmazás hová menti az adatokat.

```
public class HeroesController {
  private readonly IHeroesDao dao;
  public HeroesController(IHeroesDao heroesDao) {
    dao = heroesDao;
  }

  public IEnumerable<Hero> GetHeroes() {
    return dao.GetHeroes();
  }

  // ...
  public bool ModifyHero(Hero hero) {
    return dao.ModifyHero(hero);
  }
}
```

A kódrészlet nem teljes. A `dao` objektum megfelelő metódusait hívják a függvények és ezek előtt lehet megvalósítani a megfelelő üzleti logikát. Megfigyelhető, hogy a `GetHeroes` metódus nem egy konkrét listával tér vissza, hanem egy interfésszel. Ez azért van, mert különböző DAO megvalósítások különböző listákat adhatnak vissza és ily módon mind működni fog.

View

Hozzáadás

Egy hős hozzáadásánál az `OK` gomb eseménykezelőjét kell megvalósítani, amely ellenőrzi, hogy a feladatban leírt megszorításoknak eleget tett-e a felhasználó (ki vannak-e töltve a beviteli mezők).

```
private void OkButton_Click(object sender, EventArgs e) {
  string name = nameTextBox.Text;
  string heroName = heroNameTextBox.Text;
  string power = powerComboBox.SelectedItem.ToString();
  int age = (int) ageNumericUpDown.Value;

  if (name == string.Empty || heroName == string.Empty)
      return; // + hiba kiírása

  Hero hero = new Hero { /* ... */ };
  controller.AddHero(hero);
}
```

A `controller` objektumot a Form a konstruktorában kapja meg. Az `InitializeComponent` inicializálja a GUI-t az összes elemével együtt, lefutása után használhatók az eszközök. A powerComboBox nevű elemet feltöltjük három értékkel és alapértelmezetten legyen az első (0. indexű) kiválasztva.

```
public AddHeroWindow(HeroesController controller) {
  this.controller = controller;
  InitializeComponent();
  powerComboBox.Items.AddRange(new string[] {
    "Maga által készített",
    "Szerzett",
    "Veleszületett"
  });
  powerComboBox.SelectedIndex = 0;
}
```

Listázás

A listázáshoz egy GridView elem szükséges a Form-on. A menü listázás elemére duplán kattintva létrejön az eseménykezelő. A Hero típus rendelkezik egy int típusú ID-val, amit a felhasználó felé nem szerencsés mutatni, ezért a ViewModel-ben készítünk egy `HeroViewModel`, ami ugyanúgy néz ki, mint a Model-ben lévő párja, csak az ID ki van hagyva belőle. Miután ez megvan, akkor az eseménykezelő a következőképpen alakul:

```
private void MenuItem_Click(object sender, EventArgs e) {
  var heroes = controller.GetHeroes();
  var viewModels = new List<HeroViewModel>();

  foreach (var item in heroes)
    viewModels.Add(new HeroViewModel { /* ... */ });

  heroesGridView.DataSource = null;
  heroesGridView.DataSource = viewModels;
  heroesGridView.Visible = true;
}
```

Módosítás

Módosításhoz a GridView-hoz kell eseménykezelőt kötni, méghozzá a `Mouse/CellMouseClick` eseményhez. Az eseménykezelőnek el kell kérnie a kiválasztott hőst, majd megnyitni a hős-hozzáadás ablakot kissé máshogy, mint az előzőekben.

```
private void EditHero(/* ... */) {
  if (!(heroesGridView.CurrentRow.DataBoundItem is Hero hero))
      return;

  using var window = new AddHeroWindow(controller, hero));
  window.ShowDialog();
}
```

Az if-ben egy típusellenőrzés található, ezen felül a `hero` objektum létrejön. Az `AddHeroWindow`-ot úgy hívjuk, mint a hozzáadásnál, kivéve a konstruktorparamétereket, ahol is átadásra kerül a hős objektum is. Ehhez egy új konstruktort kell létrehozni a megfelelő Form-ban.

```
public AddHeroWindow (HeroesController controller, Hero hero)
    : this(controller) {
  heroId = hero.ID;

  nameTextBox.Text = hero.Name;
  heroNameTextBox.Text = hero.HeroName;
  ageNumericUpDown.Value = hero.Age;
  powerComboBox.SelectedIndex =
    powerComboBox.Items.IndexOf(hero.Power);

  okButton.Text = "Modify";
  IsModification = true;
}
```

A `heroId` egy int típusú változó az osztályban, az `IsModification` pedig egy bool típusú. Az első azért kell, hogy tudjuk melyik hőst kell változtatni, a második pedig az Ok gomb lenyomásakor megtörtént eseményeket befolyásolja.

A konstruktor többi része az eszközök szövegét állítja be a már meglévő hősnek megfelelően. Végül az `Ok` gomb-ot átnevezzük `Modify`-re. A fejléc végén található `: this(controller)` azt jelenti, hogy mielőtt a konstruktor törzse lefutna, a másik konstruktort meghívja a controller paraméterrel. Ez a duplikáció elkerülése miatt szükséges, nélküle át kellene másolni a kódot a már meglévő konstruktorból.

Az Ok gomb eseménykezelőjében is kell módosítani, mégpedig azt, hogy ha módosítás van, akkor más controller függvényt hívjon.

```
if (IsModification) {
  hero.ID = heroId;
  controller.ModifyHero(hero);
}
else {
  controller.AddHero(hero);
}
```