

Szoftverttechnikák önálló gyakorlat

1. Önálló feladat

A modell és a kód kapcsolata

C# alapok

A modell és a kód kapcsolata

Interfész és absztrakt (ős)osztály alkalmazástechnikája

A gyakorlatot kidolgozta: Benedek Zoltán

Utolsó módosítás ideje: 2020.02.26

Tartalom

TARTALOM	2
BEVEZETÉS	2
FEJLESZTŐKÖRNYEZET	2
A KIINDULÁSI KÖRNYEZET LETÖLTÉSE, AZ ELKÉSZÜLT MEGOLDÁS FELTÖLTÉSE	3
GIT	3
GITHUB	4
GITHUB CLASSROOM	4
GIT, GITHUB ÉS GITHUB CLASSROOM A TÁRGY KONTEXTUSÁBAN.....	4
VISUAL STUDIO ÉS A GIT	5
A FELADAT BEADÁSÁNAK LÉPÉSEI	5
<i>GitHub repository klónozása a GitHub webes felületéről indulva.....</i>	<i>6</i>
<i>Alternatíva: GitHub repository klónozása és menedzselése a GitHub extension for Visual Studio segítségével.....</i>	<i>8</i>
<i>Napi Git munka Visual Studio segítségével (Commit, Push)</i>	<i>9</i>
FELADAT 1 – EGY EGYSZERŰ .NET KONZOL ALKALMAZÁS ELKÉSZÍTÉSE	13
FELADAT 2 - AZ UML ÉS A KÓD KAPCSOLATA, INTERFÉSZ ÉS ABSZTRAKT ŐS ALKALMAZÁSTECHNIKÁJA.....	15
TOVÁBBIAK.....	18

Bevezetés

A feladathoz nem kapcsolódik előadás. A feladatok elméleti és gyakorlati háttéréül az „1. A modell és a kód kapcsolata” vezetett laborgyakorlat szolgál:

- Ezt a laborgyakorlatot a hallgatók a gyakorlatvezető útmutatásával, a gyakorlatvezetővel közösen vezetett módon végzik/végezték el.
- A laborgyakorlathoz útmutató tartozik, mely részletekbe menően bemutatja az elméleti háttérrel, valamint lépésenként ismerteti a megoldás elkészítését („Labor anyag (a labor idejére) - 1. A modell és a kód kapcsolata” és „Labor útmutató és megoldás - 1. A modell és a kód kapcsolata” a tárgy honlapján)

Erre építve jelen önálló gyakorlat feladatai a feladatleírást követő rövidebb iránymutatás segítségével elvégezhetők.

Az önálló gyakorlat célja:

- Egy egyszerű .NET alkalmazás elkészítése, C# alapok gyakorlása
- Az UML és a kód kapcsolatának szemléltetése
- Az interfész és az absztrakt őssz osztály alkalmazástechnikája

Fejlesztőkörnyezet

Visual Studio

A félév során a házi feladatok megoldásához a Visual Studio 2019 (vagy 2017) fejlesztőkörnyezetet kell használni, ebből célszerű a legfrissebb verziót feltenni. Ha telepítve van már a gépünkre a Visual Studio 2019, akkor a Start menüből indítsuk el a „Visual Studio Installer”-t. Ez induláskor ellenőrzi, érhető-e el Visual Studio-ból újabb változat online, és ha igen, az Update gombra kattintva indítsuk is el a friss verzió telepítését (jelen jegyzet írásakor a 16.4.5-es a legfrissebb változat).

Visual Studio edition-ök

A Visual Studionak több kiadása létezik. A tárgy teljesítéséhez megfelel az <http://msdn.microsoft.com/en-us/visualstudio/ee421866.aspx>-ről letölthető *Professional* edition, vagy a Microsoft honlapjáról letölthető *Community* edition. Az előbbi némiképpen több kényelmi funkcióval rendelkezik, érdemi különbség nincs a két kiadás között.

Telepítendő komponensek

A tárgy első előadása röviden kitér a .NET különböző változataira (.NET Framework, .NET Core, stb.) . Bizonyos feladatok megoldásához a .NET Framework-öt, másokhoz a .NET Core-t használjuk a félév során. Lényeges, hogy mindkettőhöz tartozó Visual Studio komponenseket telepíteni kell, ezen felül szükség van még „Class diagram” támogatásra. A „Visual Studio Installer”-t indítva a Workload oldalon az alábbi workload mindenképpen legyen kiválasztva és telepítve:

- .NET desktop development

Az „Individual components” oldalon extraként fel kell tenni a Code tools / Class Designer-t.

A kiindulási környezet letöltése, az elkészült megoldás feltöltése

Az első házi feladat kiindulási környezetének publikálása, valamint a megoldás beadása *GitHub*, pontosabban *GitHub Classroom* segítségével történik. A tárgy keretében nem célunk a Git és GitHub, részletes megismerése, csak a legszükségesebb lépésekre szorítkozunk, valamint a legfontosabb parancsokat használjuk ahhoz, hogy minden hallgató a házi feladat(ok) kiindulási programvázát egy a számára dedikált GitHub repository-ból le tudja tölteni, illetve a kész munkát ide fel tudja tölteni.

Git

A Git egy sok szolgáltatással rendelkező, rendkívül népszerű és elterjedt, ingyenesen letölthető és telepíthető, elosztottan is használható verziókezelő rendszer. A központosított rendszerekhez képest (TFS, CVS, SNV) a GIT esetében nem egyetlen központi repository-ba dolgoznak a fejlesztők, hanem mindenki egy saját lokális repository példánnyal rendelkezik. A folyamat legfontosabb lépései - némi egyszerűsítéssel - a következők (feltéve, hogy létezik egy központi repository, ahol a verziókezelő kód adott változata már elérhető):

1. A fejlesztő klónozza (**Clone**) az adott központi repository-t, melynek során egy azzal megegyező helyi repository jön létre a saját számítógépén. Ezt a műveletet elég egyszer elvégezni.

2. A fejlesztő a helyi repository-hoz tartozó munkakönyvtárban (working directory) változtatásokat végez a kódon: új fájlokat vesz fel, meglévőket módosít és töröl.
3. Ha elkészül egy érdemi részfeladat, akkor a fejlesztő a változtatásokat **Commit**-olja a számítógépén levő helyi repository-ba. Ennek során **Commit**-ot célszerű egy a változtatások jellegét jól összefoglaló megjegyzéssel ellátni.
4. A helyi repository-ból egy **Push** művelettel a fejlesztő felölti a változásokat a központi repository-ba, ahol így változtatásai mások számára is láthatóvá válnak.

Mivel a legtöbb esetben a fejlesztők csapatban dolgoznak, időnként szükség van arra, hogy mások által a központi repository-ba push-olt változtatásokat a fejlesztők a saját lokális repository-jukba letöltsék és belemerge-eljék: erre szolgál a **Pull** művelet. Fontos szabály, hogy push-olni csak akkor lehet a központi repository-ba (a Git csak akkor engedi), ha előtte mások változtatásait a saját lokális repository-nkba egy pull művelettel előtte belemerge-eltük.

A Szoftvertechnikák tárgy keretében a pull műveletet nem kell használni, mert mindenki önállóan, saját repository-ba dolgozik.

A fentieken túlmenően a GIT számos további szolgáltatást biztosít (pl. teljes verziótörténet megtekintése minden fájlra, commit történet megtekintése, tetszőleges múltbeli verzióra visszaállás, ágak kezelése, stb.).

GitHub

A GitHub egy online elérhető website és szolgáltatás (<https://github.com>), mely teljes körű Git szolgáltatást biztosít. Mindezt ráadásul – legalábbis publikus, vagyis mindenki számára hozzáférhető repository-k vonatkozásában – teljesen ingyenesen. Napjainkra a GitHub vált a közösségi kód (verziókezelt) tárolásának első számú platformjává, a legtöbb open source projekt „otthonává”.

GitHub Classroom

A GitHub Classroom egy ingyenesen elérhető GitHub-bal integrált szolgáltatás, mely többek között oktatási intézmények számára lehetővé teszi önálló tanulói feladatokhoz tartozó tanulónként egyedi GitHub repository-k létrehozását, ezáltal a kiindulási kód tanulók számára történő „kiosztását”, valamint az elkészült feladatok „beszedését”.

Git, GitHub és GitHub Classroom a tárgy kontextusában

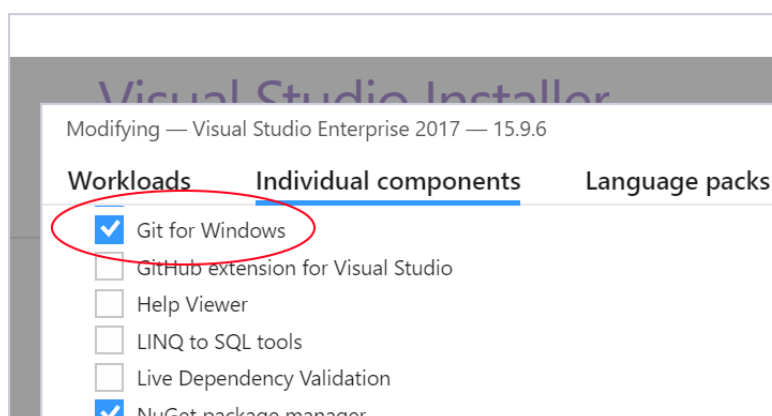
A tárgy keretében a GitHub Classroom segítségével kap minden hallgató az első házi feladatához egy dedikált, a GitHub-on hostolt repository-t, mely a megfelelő kiindulási környezettel (kiinduló Visual Studio solution-ök) inicializálásra kerül. Mindenkinek a számára dedikált repository-t kell a saját gépére Clone-oznia ebbe a változtatásait Commit-álni, és a határidőig az elkészült megoldását Push-olni. A pontos lépésekre rövidesen visszatérünk.

Visual Studio és a Git

A Git egy elosztott verziókezelő rendszer. Ahhoz, hogy a saját gépünkön dolgozni tudjunk vele, a Git-nek telepítve kell lennie. A Git önmagában is telepíthető, és command line is ki tudjuk adni a szükséges clone, push, stb. parancsokat. Az egyszerűség kedvéért a tárgy keretében a Git telepítésére és még inkább a parancsok kiadására egy grafikus felülettel rendelkező eszköz javasolt (de a rutinosak nyugodtan dolgozhatnak parancssorból is). A célnak tökéletesen megfelel a Visual Studio, mely integrált grafikus Git szolgáltatásokat is biztosít, de ha valaki esetleg otthonosan használ már a parancssori vagy grafikus eszközt (pl. GitExtensions, GitHub Desktop), természetesen ezek is tökéletesen megfelelnek a célnak. Jelen útmutató mindenestre a Visual Studio Git szolgáltatásait ismerteti.

Git telepítése Visual Studio-val

Amennyiben a számítógépünkre nincs még Git telepítve, akkor tudjuk telepíteni a Visual Studio segítségével. A Visual Studio Installer indítását követően az „Individual components” oldalra váltva a „Git for Windows” komponenst kell telepíteni.



Megjegyzés: a Git Visual Studio-tól függetlenül innen is telepíthető (Windows operációs rendszerre): <https://git-scm.com/download/win>


A feladat beadásának lépései

GitHub

Az első házi feladatot GitHub segítségével publikáljuk (a további házi feladatok vonatkozásában később hozunk döntést). A GitHub-on publikált házi feladatok kiinduló környezet letöltésének és a megoldás beadásának lépései a következők:

1. [Regisztráld](#) egy GitHub accountot, ha még nem regisztráltál és lépj be vele GitHub-ra.
2. A feladat beadásához tartozó linket nyisd meg. Ez minden feladathoz más lesz, a feladat leírásában találod. Az első házi feladathoz ez tartozik: <https://classroom.github.com/a/VSEj8vPv>

Ha a hivatkozásra kattintva hibát kapsz („There was a problem authenticating with GitHub, please try again.”), copy-paste-tel másold be közvetlenül a böngésző címsorába a címet.


3. Ha kéri, adj engedélyt a *GitHub Classroom* alkalmazásnak, hogy használja az account adatait.
4. Látni fogsz egy oldalt, ahol elfogadhatod a feladatot ("Accept the ... assignment"). Kattints a gombra.
5. Várd meg, amíg elkészül a repository. A GitHub ennek végén kiírja az új repository url-jét, amin kattintva a repository-ra lehet navigálni (ehhez hasonló: <https://github.com/bmeviauab00/hazi1-2020-username>). De nem is szükséges az url elmentése, a GitHub nyitóoldalon (<https://github.com/>) baloldalt a saját repository-k közt bármikor meg lehet később is találni.
6. Klónozd le a repository-t (ennek mikéntjére rövidesen visszatérünk). Ebben találni fogsz egy keretet, vagy kiinduló kódot. Ezen dolgozz, ezt változtasd.
7.  *A repository gyökérmappájában található neptun.txt fájlba írd bele a Neptun kódod, csupa nagybetűvel. A fájlban csak ez a hat karakter legyen, semmi más.*
8. Old meg a feladatot. Pushold a határidőig. Akárhány commitod lehet, a legutolsó állapotot fogjuk nézni.

A fenti lépések kapcsán két kérdés fogalmazódhat meg bennünk:

- Hogyan klónozzuk (clone) a repónkat?
- Hogyan commit-áljunk és push-oljunk a Visual Studio segítségével GitHub-ra?

Rövidesen ezekre is kitérünk.

Beadás a tanszéki portálon

 *Valamennyi házi feladatot a megadott határidőig a tanszéki portálra (<https://www.aut.bme.hu/>) is fel kell tölteni egy zip csomagban!*

A tanszéki portálra feltöltendő zip állománynak tartalmaznia kell a megoldás(ok) forráskódját. További szabályok:

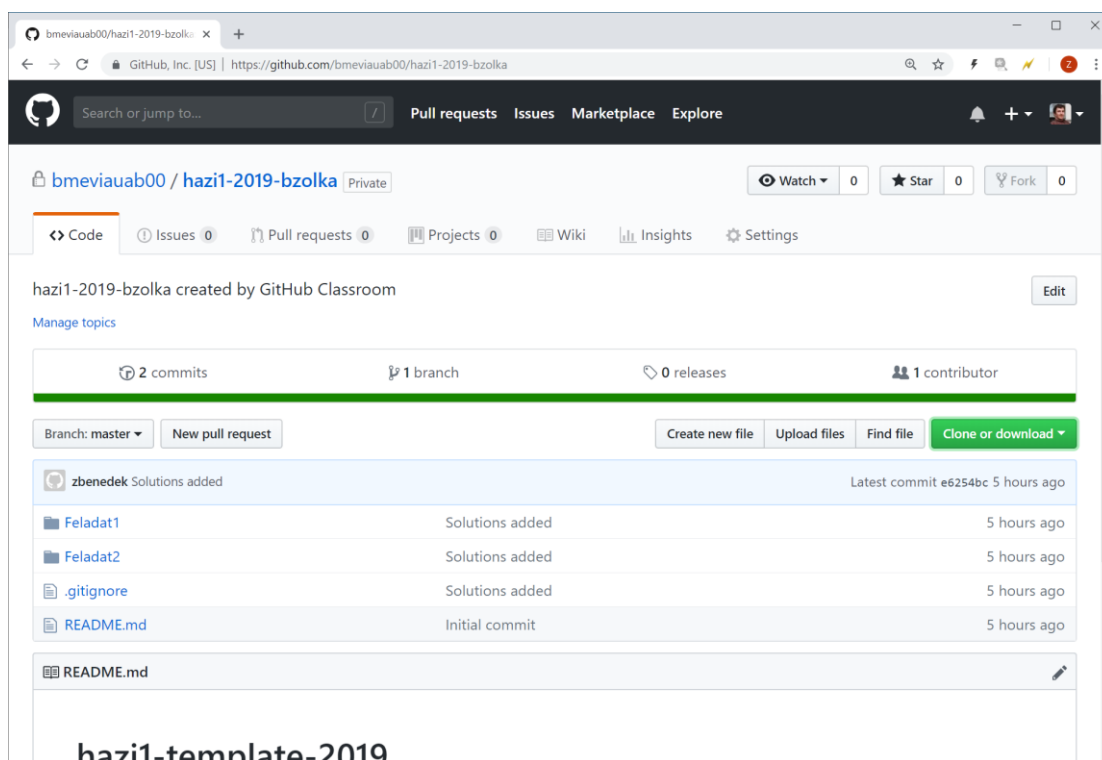
- A zip állomány nem tartalmazhatja a kimeneti („.exe”) és köztes állományokat! Ezen állományok törléséhez a Solution mappából kézzel törölni kell a „bin” és „obj” mappákat teljes tartalmukkal együtt.
- A zip állományba ne tegyük bele a „.git” és „.vs” könyvtárat.

GitHub repository klónozása a GitHub webes felületéről indulva

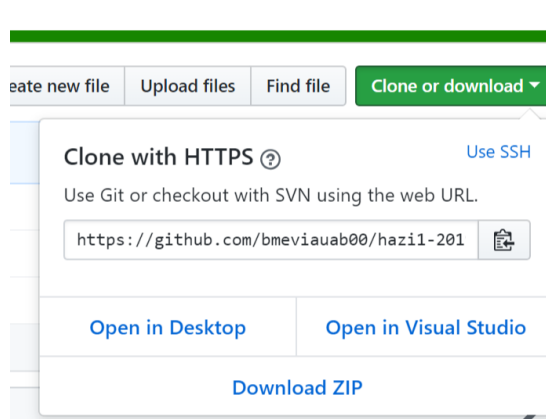
Egy repository klónozásra számos mód van, egy lehetőség a következő. Nyissuk meg az elkészült repository online oldalát, melyre számos módon eljuthatunk. Lehetőségek pl.:

- A repo létrehozásakor megjelenik a GitHub felületen az url, csak kattintani kell rajta
- A GitHub nyitóoldalon (<https://github.com>) - ha be vagyunk lépve - listázódnak baloldalt azon repository-k, melyekhez van hozzáférésünk, csak kattintsunk a megfelelőn
- Amikor elkészül a repónk, e-mail értesítést is kapunk róla, ebben is megtalálható a link.

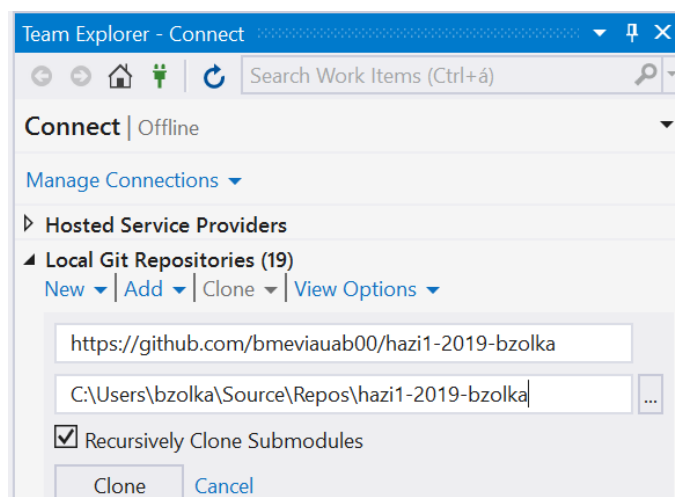
Az oldal képe nagyjából megfelel a következőnek (az mindenképpen különbség, hogy a repó url végén mindenkinél a saját felhasználóneve szerepel):



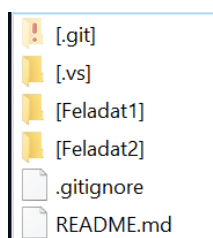
Kattintsunk a zöld színű „Clone or download” gombon, majd a lenyíló menüben az „Open in Visual Studio” linkre:



A böngészőnk ekkor jó eséllyel feldob egy ablakot (pl. a Chrome esetében a címsor alatt) melyben egy külön gombkattintással (Open...) tudjuk indítani a Visual Studio-t. Ha minden jól megy, a Visual Studio elindul, és indulás után láthatóvá válik a Team Explorer nézet, jellemzően a főablak bal vagy jobb oldali sávjában:

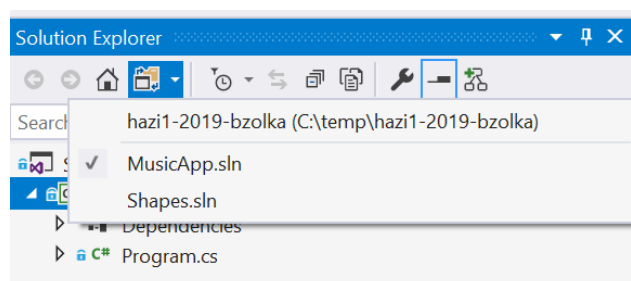


Ebben igény esetén módosítsuk a könyvtárat, ahol a lokális repository-akat tárolni szeretnénk, majd kattintsunk a **Clone** gombra. Ez néhány másodperc alatt leklónozódik a repository a megadott célmappába, pl. Windows Explorer-ben meg tudjuk tekinteni a létrehozott mappákat és fájlokat:



Ebből jól látható, hogy egy Git repository nem más, mint mappák és fájlok gyűjteménye, valamint egy a gyökérben található .git mappa, mely (némi egyszerűsítéssel élve) az egyes fájlok verziótörténetét tartalmazza.

Az első házi feladat két fő részből áll, melyekhez eltérő solution tartozik. Az elsőhöz a Feladat1 mappában található MusicApp.sln fájlt, a másodikhoz a Feladat2-ben található Shapes.sln-t kell megnyitni. A megnyitást megtehetjük Explorerből, az adott .sln fájlra duplán kattintva. De van erre más mód is. Amennyiben Visual Studio-ban a Git gyökérmappát nyitottuk meg (a Clone-t követően is ez a helyzet állt elő) a Solution Explorer nézetre váltva a nézet fejlécében „Solutions and folders” gombot lenyitva tudunk az egyes solution-ök között kényelmesen váltani:

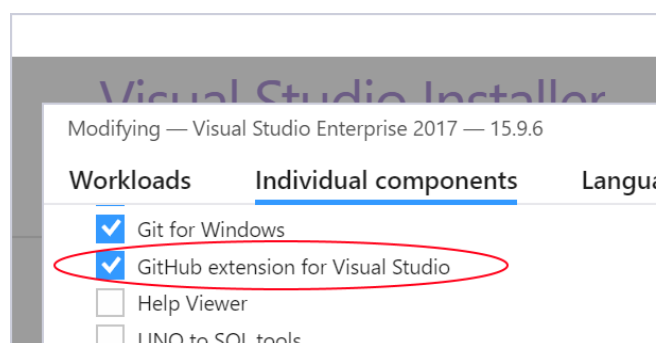


Alternatíva: GitHub repository klónozása és menedzselése a GitHub extension for Visual Studio segítségével

(Megjegyzés: az alábbi lépések a Visual Studio 2017-es verzióra vonatkoznak, újabb verziók esetén lehetnek kisebb eltérések).

A „GitHub extension for Visual Studio” használata egy alternatív lehetőség GitHub repository-k klónozására, valamint GitHub repository-k VS alóli kényelmes menedzselésére. Akkor érdemes kipróbálni, ha ez előző fejezetben ismertetett megközelítés valamiért nem működik, vagy sokat dolgozunk különböző GitHub repókkal.

Ehhez a Visual Studio Installer segítségével előbb telepíteni kell a „**GitHub extension for Visual Studio**” komponenst:



Ezt követően a Visual Studio alatt tudunk kapcsolódni a GitHub-hoz az accountunkkal, listázni a repóinkat, stb. Részletesebb leírást pl. itt található:

<https://social.technet.microsoft.com/wiki/contents/articles/38935.visual-studio-2017-install-and-use-github-extension.aspx>, amin belül ezek érdekesek számunkra:

- Connect:
<https://social.technet.microsoft.com/wiki/contents/articles/38935.visual-studio-2017-install-and-use-github-extension.aspx#Connect>

A lényeg: a Visual Studio Teams Explorerben megadjuk a GitHub login adatainkat. Ezt követően „integráltan” tudunk VS alatt a GitHub repóinkkal dolgozni.


- Clone:
<https://social.technet.microsoft.com/wiki/contents/articles/38935.visual-studio-2017-install-and-use-github-extension.aspx#Clone>

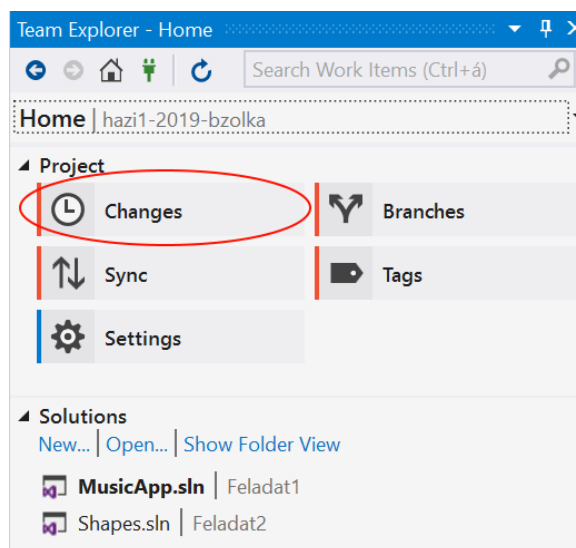
Napi Git munka Visual Studio segítségével (Commit, Push)

Miután leklónoztuk az adott házi feladathoz tartozó GitHub repository-t a számítógépünkre, és ennek során létrejött a lokális Git repository-nk, a benne levő .sln fájlokat Visual Studioban megnyitva pont úgy dolgozunk – veszünk fel új fájlokat, módosítunk/törölünk meglévőket – mintha a fájlok nem is tartoznának semmiféle Git repóhoz. Ugyanakkor, legkésőbb a feladat beadásakor a változtatásainkat commitálni kell, majd push-olni GitHub-ra. A munka során akárhányszor commitálhatjuk/pusholhatjuk az előző commit óta eszközölt módosításainkat: a házi feladat ellenőrzésekor a határidő pillanatában a GitHub-on található állapot kerül elbírálásra, teljesen mindegy, hány commit tartozik hozzá.

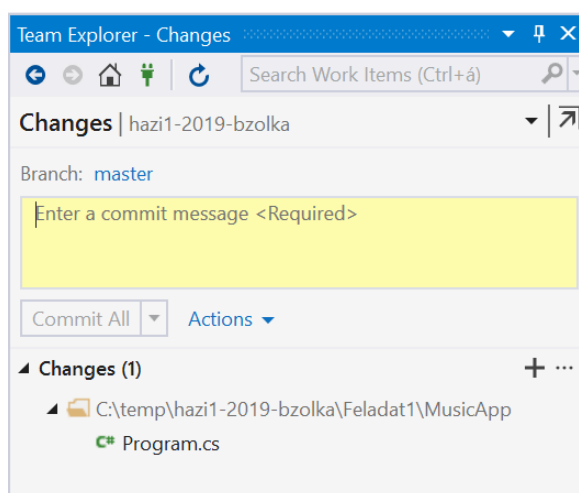
A commit és push műveletek végrehajtásához a Visual Studio Team Explorer nézetét használjuk.

Commit

Az előző commit óta eszközölt változtatások megtekintéséhez kattintsunk a Team Explorer fejlécében a home ikonra  majd a **Changes** gombra:




Ekkor a Team Explorer a **Changes** nézetre vált át, melyben alul megtekinthető a változott fájlok listája:

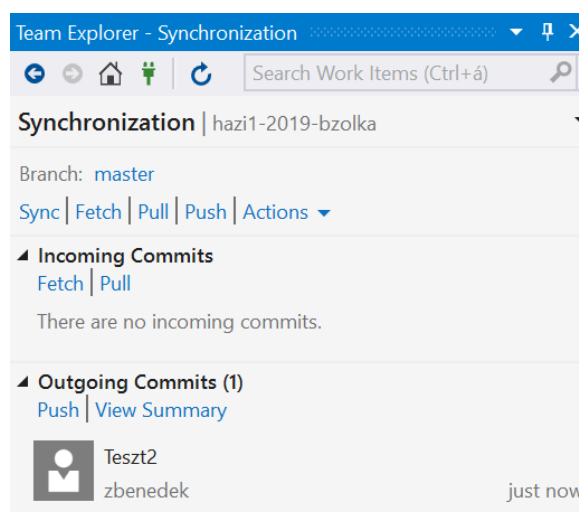


A változtatások commit-álásához írunk a szövegmezőbe egy a változtatásokra jellemző egy-két soros leírást (pl. „Végső megoldás”, „Az xyz hiba javítása”, stb.), majd kattintsunk a „Commit All” gombra. Ha a változtatásainkat egyből publikálni is szeretnénk a GitHub-on levő központi repóba, akkor a gombot lenyitva használjuk bátran a „Commit All and Push” parancsot. A házi feladatok tekintetében célszerű ezt használni, mert ekkor nincs szükség a commit-ot követően külön push műveletre.

Push

A commit művelet csak a helyi repository-ban „érvényesíti” a változtatásokat. Ezt követően a változtatásokat a GitHub központi repository-nkba fel kell tölteni a push művelettel. Erre a lépésre csak akkor van szükség, ha a commit során nem használtuk a „Commit All and Push” vagy „Commit All and Sync” parancsokat.

Lépések: Team Explorer fejlécében kattintsunk a home ikonra  majd a Sync gombra, mely a következő nézethez vezet:



A megjelenő nézet két szekcióval rendelkezik:

- Incoming commits: Megmutatja, hogy a központi repository-ban milyen mások által pusholt, hozzánk még le nem töltött commitok vannak. Ezeket a **Pull** művelettel tudjuk a helyi repónkba merge-elni. A házi feladat vonatkozásában ezt nem használjuk, hiszen mindenki saját dedikált központi repositoryval rendelkezik, melyben egyedül dolgozik.
- Outgoing commits: Megmutatja, hogy milyen a lokális repository-nkba már létező, de a központi repóba még nem push-olt commitok vannak. Ezeket a **Push** művelettel tudjuk feltölteni.

Megjegyzés: A Pull és Push parancsok helyett bátran használhatjuk a **Sync** műveletet, mely a kettőt kombinálja (előbb Pull majd Push).

Javaslat

Amíg nem vagy rutinos a Visual Studio Git szolgáltatásainak használatában, a push-t követően (legkésőbb akkor, amikor a házi feladatot beadottnak tekintjük) célszerű ellenőrizni a GitHub webes felületén a repository-ban a fájlokra való rápillantással, hogy valóban minden változtatást feltöltöttél-e.

Egyéb irányelvek

A Git commit és push során megfigyelhetjük, hogy a solution-jeink köztes és kimeneti állományai (.dll, .exe, stb. fájlok) nem kerülnek bele a commitba és így nem kerülnek fel GitHubra sem. Ez így is van jól, ezen állományok bármikor reprodukálhatók, a verziókezelő rendszernek nem feladata ezek tárolása, csak felesleges és zavaró helyfoglalók lennének. Felmerül a kérdés, honnan tudja a Git, hogy mely állományokat szükséges figyelmen kívül hagyni a commit során. Erre szolgál a repository-ban (tipikusan annak gyökérmappájában) található .gitignore fájl, mely felsorolja azon mappákat, fájlkiterjesztéseket, illetve egyedi fájlokat, melyeket a commit során figyelmen kívül szeretnénk hagyni. A .gitignore fájl tartalma teljes egészében a kezünk alá tartozik, szabadon szerkeszthető/commitálható/pusholható.

i A tárgy keretében minden kiinduló repónak része egy .gitignore fájl, ne változtassuk a tartalmát! Így a commit/push során a kimeneti állományok a házi feladatok esetében sem kerülnek fel GitHub-ra, és egy így is van rendjén.

A félévben a feladatok megoldása során az egyes osztályok, interfészek, stb. forráskódját külön fájlba kell tenni, vagyis egy C# forrásfájlban egy osztály/interfész/stb. definíciója legyen.

Feladat 1 – Egy egyszerű .NET konzol alkalmazás elkészítése

A kiindulási környezet a „Feladat1” mappában található, az ebben levő MusicApp.sln fájlt nyissuk meg Visual Studioban és ebben a solutionben dolgozzunk.

Feladatkiíráshoz tartozó melléklet

A „Feladat1\Input” mappában található egy music.txt fájl, mely a feladat bemeneteként használandó.

Feladat

Egy szövegfájlban zeneszerzők/előadók/együttesek számainak címeit tároljuk a következő formátumban. Minden szerzőhöz külön sor tartozik. Minden sorban először a szerző neve szerepel, majd „;”-t követve „;”-vel elválasztva számok címei. A fájl tartalma érvényesnek tekintendő, ha üres, vagy csak whitespace (space, tab) karaktereket tartalmazó sorok is vannak.

A mellékelt music.txt fájl tartalma a következő:

```
Adele; Hello; Rolling in the Deep; Skyfall  
Ennio Morricone; A Fistful Of Dollars; Man with a Harmonica  
AC/DC; Thunderstruck; T.N.T
```

Olvassuk be a fájlt `Song` osztálybeli objektumok listájába. Egy `Song` objektum egy dal adatait tárolja (szerző és cím). A beolvasást követően írjuk ki formázott módon az objektumok adatait a szabványos kimenetre az alábbi formában:

```
szerző1: szerző1_dalcím1  
szerző1: szerző1_dalcím2  
...  
szerző2: szerző2_dalcím1  
...  
stb.
```

A mintafájlunk esetében a következő (a fájl tartalmának függvényében lehet eltérés) kimenetet szeretnénk látni:

A megvalósítás lépései

Vegyünk fel egy `Song` nevű osztályt a projektbe (jobb katt a *Solution Explorer*ben a projekten, a menüben *Add/Class*).

Vegyünk fel a szükséges tagokat és egy ezekhez passzoló konstruktort:

```
class Song
{
    public readonly string Artist;
    public readonly string Title;

    public Song(string artist, string title)
    {
        Artist = artist;
        Title = title;
    }
}
```

A tagváltozókat `readonly`-ként vettük fel, mert nem akartuk, hogy utólag ezek a konstruktor lefutását követően megváltoztathatók legyenek.

A következőkben a `Song` osztályunkban definiáljuk felül az implicit `System.Object` ősből örökölt `ToString` műveletet, hogy az az előírt formában adja vissza objektum adatait:

```
public override string ToString()
{
    return string.Format("{0}: {1}", Artist, Title);
}
```

Szövegfájl feldolgozására legkényelmesebben a `System.IO` névtérben levő `StreamReader` osztályt tudjuk használni. Leírása az útmutató írásának pillanatában itt található (idővel változhat, de bármely internetes keresőben könnyen megtaláljuk):

[https://msdn.microsoft.com/en-us/library/system.io.streamreader\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.io.streamreader(v=vs.110).aspx)

A `Main` függvényünkben olvassuk fel soronként a fájlt, hozzuk létre a `Song` objektumokat és tegyük be egy `List<Song>` dinamikusan nyújtózkodó tömbbe. Figyeljünk arra, hogy a fájlban a „;”-vel elválasztott elemek előtt/után whitespace karakterek (space, tab) lehetnek, ezektől szabaduljunk meg!

A következő kód egy lehetséges megoldást mutat, a megoldás részleteit a kódkommentek magyarázzák. A félév során ez az első önálló feladat, valamint a hallgatók többségének ez első `.Net/C#` alkalmazása, így itt még adunk mintamegoldást, de a rutinosabb hallgatók önállóan is próbálkozhatnak:

```
static void Main(string[] args)
{
    // Ebben tároljuk a dal objektumokat
    List<Song> songs = new List<Song>();

    // Fájl beolvasása soronként, songs lista feltöltése
    StreamReader sr = null;
    try
    {
        // A @ jelentése a string konstans előtt: kikapcsolja
        // a string escape-elést, így nem kell a '\' helyett '\\'-t írni.
        sr = new StreamReader(@"c:\temp\music.txt");
        string line;
        while ((line = sr.ReadLine()) != null)
        {

```

```

        // A line változóban benne van az egész sor,
        // a Split-tel a ;-k mentén feldaraboljuk
        string[] lineItems = line.Split(';');

        // Ha üres volt a sor
        if (lineItems.Length == 0)
            continue;

        // Első elem, amiben az szerző nevét várjuk
        // A Trim eltávolítja a vezető és záró whitespace karaktereket
        string artist = lineItems[0].Trim();

        // Menjünk végig a dalokon, és vegyük fel a listába
        for (int i = 1; i < lineItems.Length; i++)
        {
            Song song = new Song(artist, lineItems[i].Trim());
            songs.Add(song);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("A fájl feldolgozása sikertelen.");
        // Az e.Message csak a kivétel szövegét tartalmazza.
        // Ha minden kivétel információt ki szeretnénk írni (pl. stack trace),
        // akkor az e.ToString()-et írjuk ki.
        Console.WriteLine(e.Message);
    }
    finally
    {
        // Lényeges, hogy finally blokkban zárjuk le a fájlt,
        // hogy egy esetleges kivétel esetén se maradjon mögöttünk
        // lezáratlan állomány.
        // try-finally helyett használhattunk volna using blokkot,
        // azt egyelőre nem kell tudni (a félév derekán tanuljuk).
        if (sr != null)
            sr.Close();
    }

    // A songs lista elemeinek kiírása a konzolra
    foreach (Song song in songs)
        Console.WriteLine( song.ToString() );
}

```

A c:\temp mappába másoljuk ki a music.txt fájlt, és futtassuk az alkalmazást. A megvalósítás során az egyszerűsége törekedve mindent beleöntöttünk a `main` függvénybe, „éles” környezetben mindenképp célszerű a kódot egy külön feldolgozó osztályba kifaktorálni.

Feladat 2 - Az UML és a kód kapcsolata, interfész és absztrakt ős alkalmazástechnikája

A kiindulási környezet a „Feladat2” mappában található, az ebben levő `Shapes.sln` fájlt nyissuk meg Visual Studioban és ebben a solutionben dolgozzunk.

Feladatkiíráshoz tartozó melléklet

A „Feladat2\Shapes” mappában található egy `Controls.dll` fájl, ezt a feladat megoldása során kell majd felhasználni.

Beadandó (a forráskódon túlmenően)

Fél-egy oldalban a „Feladat2” megoldás során hozott tervezői döntések, a megoldás legfontosabb alapelveinek rövid szöveges összefoglalása, indoklása. Ezt a Feladat2 mappában található „Megoldás bemutatása.txt” szövegfájlba kell beleírni.

Feladat

Egy síkbeli vektorgrafikus alakzatokat kezelni képes CAD tervezőalkalmazás első változatának kifejlesztésével bízunk meg bennünket. Bővebben:

- Különböző típusú alakzatokat kell tudni kezelni. Kezdetben a `Square` (négyzet), `Circle` (kör) és `TextArea` típusú alakzatokat kell támogatni, de a kód legyen könnyen bővíthető új típusokkal. Az `TextArea` egy szerkeszthető szövegdoboz.
- Az alakzatokhoz tartozó adatok: x és y koordináta, valamint olyan adatok, melyek a megjelenítéshez és az alakzatok területének kiszámításához szükségesek. Pl. négyzet esetében oldalhosszúság, textarea esetében szélesség és magasság, kör esetében a sugár.
- Minden alakzatnak biztosítania kell műveletek típusának, koordinátái és területének lekérdezéséhez. A típus lekérdező művelet stringgel térjen vissza, illetve a beépített `Type` osztály `GetType` művelete nem használható a megvalósítás során.
- Listázni kell tudni a memóriában nyilvántartott alakzatokat a szabványos kimenetre (konzolra). Ennek során a következő adatokat írjuk ki: alakzat típusa (pl. négyzet esetén „Square”, stb.), a két koordináta, alakzat területe. (beépített `Type` osztály `GetType` művelete nem használható a típus kiírás során).
- Az `TextArea` osztálynak osztálynak kötelezően a jelen feladathoz tartozó `Controls.dll` osztálykönyvtár `Textbox` osztályából kell származnia. A `Controls.dll` egy .NET szerelvény, lefordított formában tartalmaz osztályokat.
- A megvalósítás során törekedjen egységbezárásra: pl. az alakzatok menedzselése legyen egy **erre dedikált osztály** feladata: az **nem elfogadható**, ha a `Main` függvényben egy helyben létrehozott egyszerű listába kerülnek az alakzatok tárolásra!
- A megvalósítás során kerülje ez a kód duplikációt (tagváltozók, műveletek, konstruktorok esetében egyaránt).
- A `Main` függvényben mutasson példát az osztályok használatára.
- Legkésőbb a megvalósítás végére készítsen a Visual Studio solutionben egy osztálydiagramot, melyen a solution osztályait jól áttekinthető formában rendezze el. Az asszociációs kapcsolatokat asszociáció formájában jelenítse meg, ne tagváltozóként (*Show as Association* ill. *Show as Collection Association*, lásd 1. gyakorlat útmutatója).

Megjegyzés: A Visual Studio 2017 – és jó eséllyel a később megjelenő változatok - nem teszik fel minden esetben a Class Designer komponenst a telepítés során. Ha nem lehet Class Diagram-ot felvenni a Visual Studio projektbe (mert a Class Diagram nem szerepel a listában az Add New Item parancs során megjelenő ablak listájában), akkor a Class Diagram

komponenst utólag kell telepíteni. Erről bővebben jelen útmutató „Telepítendő komponensek” fejezetében lehet olvasni.

A megvalósítás során jelentős egyszerűsítéssel élünk:

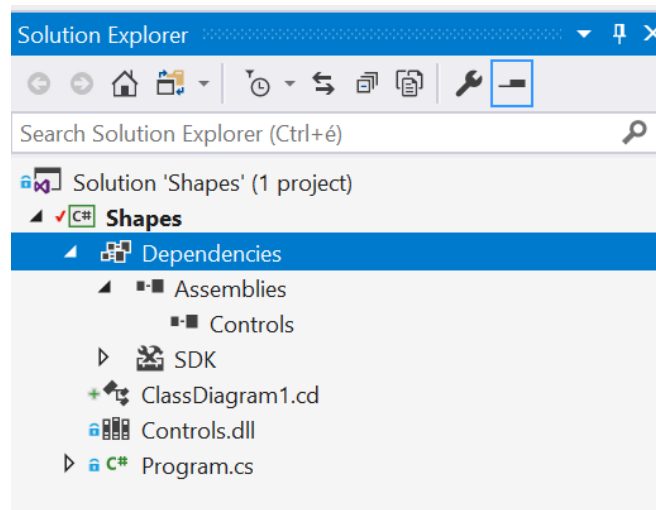
- Az alkatatok kirajzolását nem valósítjuk meg (az ehhez szükséges ismeretek a félév során később szerepelnek).
- Az alkatokat csak a memóriában kell nyilvántartani.

Osztálykönyvtárak használata

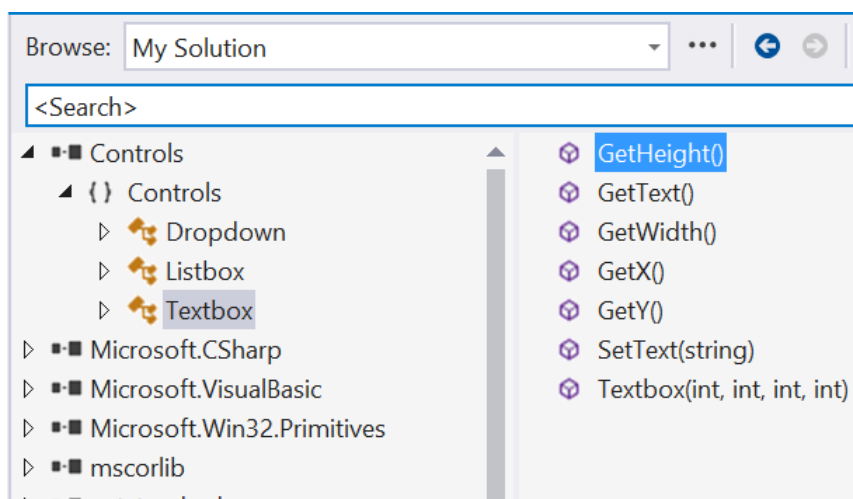
A megoldás az „1. gyakorlat - A modell és a kód kapcsolata” laborgyakorlat - melyhez részletes, lépésenkénti útmutató tartozik - mintájára kidolgozható. Jelen feladat egy lényeges részletében különbözik tőle: míg abban csak szóban kötöttük ki, hogy a `DisplayBase` őszosztály forráskódja nem megváltoztató, jelen esetben a `Textbox` őszosztályunk esetében ez adott, hiszen csak egy lefodított dll formájában áll rendelkezésre. A következőkben nézzük meg, milyen lépésekben lehet egy ilyen dll-ben levő osztályokat kódunkban felhasználni:

1. A Visual Studio *Solution Explorer* ablakában jobb gombbal kattintsunk a *Dependencies* elemen, és válasszuk az *Add Reference*-t.
2. A megjelenő ablak bal oldalán válasszuk ki a *Browse* elemet,
 - a. Ha az ablak közepén a listában megjelenik a *Controls.dll*, pipáljuk ki az elemet.
 - b. Ha nem jelenik meg, akkor kattintsunk az ablakunk jobb alsó részében levő „Browse...” gombon.
 - i. A megjelenő fájlböngésző ablakban navigáljunk el a *Controls.dll* fájlhoz, és kattintsunk rajta duplán, ami bezárja az ablakot.
 - ii. A *Reference Manager* ablakunk középső részén a *Controls.dll* látható kipipálva, az OK gombbal zárjuk be az ablakot.
3. OK gombbal zárjuk be az ablakot.

Ezzel a projektünkben felvettünk egy referenciát a *Controls.dll*-re, így a benne levő osztályok úgy használhatók (pl. lehet példányosítani őket, vagy lehet belőlük származtatni), mintha a forráskódjuk a rendelkezésünkre állna. A *Solution Explorer*-ben a *Dependencies* majd *Assemblies* csomópontot lenyitva a *Controls* megjelenik:



A `Textbox` osztály, melyből az `TextArea` osztályunkat származtatni kell, a `Controls` névtérben található. A `TextBox` osztálynak egy konstruktora van, melynek négy paramétere van, az `x` és `y` koordináták, valamint a szélesség és a magasság. Amennyiben szükség lenne rá, a többi művelet felderítésében az `Object Browser` segít. Az `Object Browser` a `View` menüből az `Object Browser` menü kiválasztásával nyitható meg. Az `Object Browser` egy új tabfülön jelenik meg. A `Controls` komponenst lenyitogatva az egyes csomópontokat kiválasztva (névtér, osztály) az adott csomópont jellemzői jelennek meg: pl. az osztály nevén állva az osztály tagjait látjuk.



Most már minden információ rendelkezésünkre áll a feladat megvalósításához.

Továbbiak

Pár jótanács, ismétlésképpen (a többségük a későbbi házi feladatokra is vonatkozik):

- A repository gyökérmappájában található `neptun.txt` fájlba írd bele a Neptun kódod, csupa nagybetűvel. A fájlban csak ez a hat karakter legyen, semmi más.
- A GitHub-ról leöltött kiinduló solutionben/projektekben kell dolgozni, nem újonnan létrehozottban.
- A 2. feladat során ne felejtsd el a „„Megoldás bemutatása.txt”-ben a megoldásod bemutatni.
- Amíg nem vagy rutinos a Visual Studio Git szolgáltatásainak használatában, a push-t követően (legkésőbb akkor, amikor a házi feladatot beadottnak tekintjük) célszerű ellenőrizni a GitHub webes felületén a repository-ban a fájlokra való rápillantással, hogy valóban minden változtatást feltöltöttél-e.
- Lényeges, hogy a feladatok csak akkor kerülnek elfogadásra, ha teljesen elkészülnek, és minden tekintetben teljesítik a követelményeket. Nem forduló kód, illetve részleges megoldás elfogadásában nem érdemes bízni.
- Természetesen saját munkát kell beadni (hiszen értékelésre kerül).