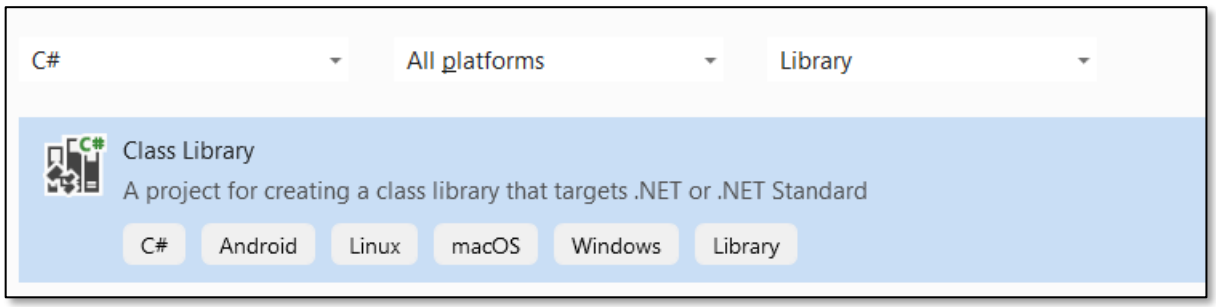
Csokigyár (tutoriál)

Ebben a feladatban csokigyártást kell modellezned. A feladathoz elkészített interfész és az osztályok külön osztály könyvtárba (Class Library) kerüljenek! A feladat elkészítésénél törekedj az **OOP elvek** és a **clean code** szabályok betartására!

A Class Library-t új projektként kell létrehozni:



1. Készíts egy interfészt **`IEtel`** néven, amely egy metódust és egy tulajdonságot deklarál. A `MibolKeszul` metódus nem vár paramétert, és egy string típusú IEnumerable sorozattal tér vissza. A `MegfeleloMinoseg` csak olvasható tulajdonság igaz vagy hamis értékkel tér vissza, esetleg `SilanyMinosegException` kivételt dobhat.

Az interfész láthatósága legyen public. Minimum 2 projekttel fogunk dolgozni, lesz egy osztály könyvtárunk, amiben létrehozzuk az alkalmazás logikájáért felelős interfészt és az osztályokat, és lesz egy külön projektünk, ami a bemenő adatok feldolgozásáért és az adatok megjelenítéséért felel. Utóbbiban is használni fogjuk az IEtel interfészt, ezért nem lesz elég az internal láthatóság.

Az interfészben csak a metódusok és tulajdonságok fejét adjuk meg. Nem adunk nekik public láthatóságot, amelyik osztály implementálja az interfészt, ott az osztályban publikus láthatósággal kell ellátni az interfészben meghatározott tulajdonságokat és függvényeket.

```
5 references
public interface IEtel
{
    1 reference
    IEnumerable<string> MibolKeszul();
    3 references
    bool MegfeleloMinoseg { get; }
}
```

2. Készítsd el a **SilanyMinosegException** osztályt, amely az Exception osztályból származik, és *„Nem igazi csoki!”* szöveggel dob hibaüzenetet!

Minden saját kivétel osztály az Exception osztályból származik. A saját kivétel osztály konstruktorának meg kell hívnia az Exception osztály valamelyik konstruktorát. Az ősosztály konstruktorának paraméterül adjuk a feladat által meghatározott hibaüzenetet.

```
4 references
public class SilanyMinosegException : Exception
{
    2 references
    public SilanyMinosegException() : base("Nem igazi csoki!")
    {
    }
}
```

A Warning üzenetet most figyelmen kívül hagyjuk, nem szeretnénk a serialization pattern-t alkalmazni.

3. Írj egy osztályt **Csoki** néven, mely implementálja az IEtel interfészt!

A Csoki osztályt csak a projekten belül fogjuk felhasználni, elegendő az internal láthatóság. Jobb egér gombbal generáltassuk le az interfészben szereplő metódust és tulajdonságot, hogy a piros aláhúzás eltűnjön. A későbbiekben meg fogjuk majd írni a metódus és a tulajdonság kódját.

```
internal class Csoki : IEtel
```

a. Az osztálynak 3 adattagja legyen: szöveg típusú adatként tároljuk a gyártott csoki fajtáját, egy string típusú tömbben a felhasznált alapanyagokat, egy szám típusú adattagban pedig a kakaó tartalmat.

A legszűkebb láthatóságra tervezünk, és szükség esetén módosítjuk az elérhetőségi szinteket. Automatikus tulajdonságokat hozunk létre get és init részekkel, a tulajdonságok értékeit konstruktorból állítjuk be, más metódus nem módosíthatja az értéküket.

```
2 references
private string Csokifajta { get; init; }
3 references
private string[] Alapanyagok { get; init; }
4 references
private int KakaoTartalom { get; init; }
```

A későbbiekben majd szükség lesz a KakaoTartalom tulajdonság láthatóságának módosítására.

b. Az osztályhoz paraméteres konstruktor is tartozik, mely 3 paramétert vár, és minden adattagot inicializál.

A tulajdonságok létrehozása után generáltassuk le a fejlesztőeszközzel a konstruktort. (Osztály nevén jobb egér gomb, Quick Actions and Refactorings…, ott válasszuk a Generate constructor… parancsot, és legyen kijelölve az összes tulajdonság. A generált konstruktort módosítani fogjuk, ha IEnumerable<string> típusként kapja meg az alapanyagokat, az általánosabb, és elrejti a külvilág felé, hogy milyen típusként tároljuk az osztályban ténylegesen az alapanyagokat. Természetesen ha módosítjuk a konstruktor paraméterének típusát, akkor a konstruktor törzsében gondoskodnunk kell a paraméter átalakításáról.

```
2 references
public Csoki(string csokifajta, IEnumerable<string> alapanyagok,
    int kakaoTartalom)
{
    Csokifajta = csokifajta;
    Alapanyagok = alapanyagok.ToArray();
    KakaoTartalom = kakaoTartalom;
}
```

c. A `MibolKeszul` metódus implementációja adja vissza a felhasznált alapanyagokat!

A metódus az Alapanyagok tulajdonság értékével tér vissza. A láthatósága public, hiszen az IEtel interfészben szerepel. A visszatérési értéke IEnumerable<string> lesz, mert ez általánosabb a tömbnél, és az összes sorozat típus ezt az interfészt megvalósítja. A tömböt nem kell átalakítani másik típussá, mivel interfész típusú változó értéket kaphat egy őt megvalósító osztály példányából.

```
1 reference
public IEnumerable<string> MibolKeszul()
{
    return Alapanyagok;
}
```

d. A `MegfeleloMinoseg` tulajdonságban ellenőrizni kell a kakaó tartalmat. Ha a kakaótartalom nagyobb 50%-nál, akkor térjen vissza igaz értékkel, ha 0% és 50% közötti, akkor hamissal, ha negatív, akkor dobjon hibát a `SilanyMinosegException` kivétel osztály használatával!

A MegfeleloMinoseg tulajdonságot publikus láthatósággal látjuk el, mivel a megvalósítandó interfészben szerepel. A tulajdonság törzsében a switch kifejezést használjuk. Figyeljünk rá, hogy a feltételek megfelelő sorrendben legyenek felsorolva!

```
2 references
public bool MegfeleloMinoseg
    => KakaoTartalom switch
    {
        > 50 => true,
        >= 0 => false,
        _ => throw new SilanyMinosegException()
    };
```

A későbbiekben a tulajdonságot virtuálissá fogjuk tenni, mivel a PremiumCsoki osztályban felül szeretnénk írni a tartalmát.

e. Az osztály `ToString` metódusa adja vissza szövegként a csoki fajtáját, kakaó tartalmát, valamint az alapanyagokat!

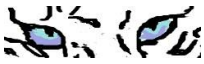
A ToString metódus az Object osztályban van definiálva, amely minden osztálynak az őse. Virtuális metódus, ha szeretnénk a saját osztályunkban felülírni a működését, akkor override kulcsszót kell használni. Ha beírjuk a programkódba azt, hogy override, fel fogja ajánlani, hogy melyik metódust szeretnénk felülírni. Válasszuk ki a ToString-et, és készítsük el a feladat leírásnak megfelelő tartalmat!

```
2 references
public override string ToString()
{
    return $"{Csokifajta} kakaótartalom: {KakaoTartalom}% " +
        $"alapanyagai: {String.Join(", ",Alapanyagok)}";
}
```

4. Készíts egy **`PremiumCsoki`** osztályt, amely a `Csoki` osztály leszármazottja, belőle viszont nem származhat további osztály.

Az osztályból nem származhat további osztály, ezt a sealed kulcsszóval jelezzük. Az osztály a Csoki osztályból származik, ezt az osztály neve után, kettőspont után adjuk meg. A Csoki osztály implementálja az IEtel interfészt, így a PremiumCsoki is implementálni fogja automatikusan, ezért ezt nem kell megadni.

```
internal sealed class PremiumCsoki : Csoki
```



A PremiumCsoki osztály a Csoki osztályból származik. A Csoki osztálynak nincs paraméter nélküli konstruktora, ezért nekünk kell gondoskodnunk arról, hogy a PremiumCsoki osztály meghívja a Csoki osztály paraméteres konstruktorát, a megfelelő paraméterek átadásával. Az ősosztály konstruktorát a base kulcsszó használatával érjük el. Mivel mindhárom paramétert megkapja az ősosztály konstruktora, és az ősosztály konstruktora gondoskodik az adatok értékének inicializálásáról, ezért a PremiumCsoki osztály konstruktorának törzse üresen marad.

```
public PremiumCsoki(string csokifajta,
    IEnumerable<string> alapanyagok, int kakaoTartalom)
    : base(csokifajta, alapanyagok, kakaoTartalom)
{
}
```

a. Ebben az osztályban a megfelelő minőséget vizsgáló tulajdonság csak 80%-ot meghaladó kakaó tartalom esetén adjon vissza igaz értéket!

A megfelelő minőséget vizsgáló tulajdonságot felül kell definiálni az osztályban, ezért a Csoki osztályban 2 módosítást is el kell végezni.

Egyrészt a MegfeleloMinoseg tulajdonság virtuális lesz, hogy a leszármazott osztályban a tulajdonságot override kulcsszóval felül tudjuk írni. Így csak futás időben dől el, az objektum dinamikus típusa alapján, hogy az ős vagy a gyerek osztály tulajdonságát kell meghívni.

```
3 references
public virtual bool MegfeleloMinoseg
    => KakaoTartalom switch
    {
        > 50 => true,
        >= 0 => false,
        _ => throw new SilanyMinosegException()
    };
```

A tulajdonság használja a KakaoTartalom tulajdonság értékét, így ezt a tulajdonságot elérhetővé kell tenni a leszármazott osztályban is. Ezért a KakaoTartalom tulajdonság láthatóságát módosítjuk protected-re.

```
protected int KakaoTartalom { get; init; }
```

Most már minden adva van ahhoz, hogy elkészítsük a PremiumCsoki osztály MegfeleloMinoseg tulajdonságát:

```
    3 references
    public override bool MegfeleloMinoseg
=> KakaoTartalom switch
{
    > 80 => true,
    >= 0 => false,
    _ => throw new SilanyMinosegException()
};
```

b. Az osztály `ToString` metódusa adja vissza szövegként a csoki fajtáját, kakaó tartalmát, valamint az alapanyagokat, jelezve, hogy prémium csokoládéról van szó!

Mivel az osztály ToString metódusa csak 1 szóban tér el az ősosztály ToString metódusától, ezért a visszaadott string összeállításánál felhasználjuk az ősosztály ToString metódusát. Az ősosztály publikus metódusai a base kulcsszó segítségével érhetők el.

```
2 references
public override string ToString()
{
    return $"Prémium {base.ToString()}";
}
```

5. Készíts egy **`EtelFactory`** osztályt, amelynek `Factory` metódusa `IEtel` típussal tér vissza!

Mivel ezt az osztályt fogjuk használni arra, hogy az alkalmazás projektjében a megfelelő csoki példányokat előállítsuk, ezért az osztály láthatósága publikus lesz. Konstruktort nem készítünk, csak a Factory metódus lesz az osztály tartalma.

```
public class EtelFactory
```

a. Az 1 string paraméterrel rendelkező metódus az adatfájl 1 adatsorából elkészíti az adatoknak megfelelő csoki példányt. Az adatsor adatai pontosvessző karakterrel vannak egymástól elválasztva. A sorban az első adat a csokoládé típusa, a 2. adat a kakaó tartalom egész számként, utána pedig felsorolva az alapanyagok. A sor végén opcionálisan álló „prémium” szöveg jelzi, ha a csokoládé prémium minőségű.

A metódus láthatósága public, a megfelelő csoki példányok előállítására szeretnénk használni. A paraméterül kapott adatsort ; karakter mentén daraboljuk. Megvizsgáljuk, hogy az utolsó érték „prémium” szó volt-e, mert ebben az esetben prémium csokit kell gyártanunk. Mivel a kapott adatsorban és a csoki osztályok konstruktorában nem ugyanolyan sorrendben vannak a paraméterek, ezért az olvashatóság kedvéért a paraméterek beállítását a paraméternevek megadásával végezzük. Ügyeljünk arra, hogy az alapanyagokhoz ne kerüljön be a „prémium” szó!
Amennyiben az utolsó érték nem „prémium” szó volt, akkor az alapanyagok lista a 2. elemtől kezdve az utolsó elemig tart.

```
1 reference
public IEtel Factory(string adatSor)
{
    string[] adatElemek = adatSor.Split(';');
    if (adatElemek[^1] == "prémium")
    {
        return new PremiumCsoki(
            adatElemek[0],
            kakaoTartalom: int.Parse(adatElemek[1]),
            alapanyagok: adatElemek[2..^1]
            );
    }
    return new Csoki(
        adatElemek[0],
        kakaoTartalom: int.Parse(adatElemek[1]),
        alapanyagok: adatElemek[2..]
        );
}
```

b. Szorgalmi feladatként készíts el többféle `Factory` metódust is, amelyek segítségével a különböző típusú módon felépített fájlok feldolgozását gyakorolhatod!

6. Készítsd el a futtatható osztályt!

A futtatható osztályt külön projektben készítjük. A Solution Explorer-ben, a projekt Dependencies részén jobb egér gombbal kattintva válasszuk ki az Add Project Reference… parancsot! Kattintsuk be asz osztályokat tartalmazó project-et!
Futtatásnál majd figyeljünk arra, hogy ez a projekt induljon el. A Solution Explorer vastagított betűvel jelzi, hogy melyik projekt az alkalmazás belépési pontja. Szükség esetén a projekt nevén jobb egér gombot nyomva, Set as Startup Project -re kattintva átállíthatjuk, hogy melyik projekt induljon el futtatáskor.

a. Az osztály nyisson meg és olvasson be egy „input.txt” nevű állományt! A fájl első sora tartalmazza, hogy hány további sor van a fájlban. Dolgozd fel a sorokat, és hozd létre a megadott számú `Csoki` vagy `PremiumCsoki` objektumokat!

A projektet top-level statements sablon használatával készítjük. Figyeljünk arra, hogy a megírt metódusok a fájl végére kerüljenek, és ne felejtsük el a metódusokat meghívni! A projektfájl mellé másoljuk oda az input.txt állományt, és a Solution Explorer-ben az input.txt állományon jobb egér gombot nyomva, a Properties parancsra kattintva állítsuk be, hogy a fájl szükség esetén az exe állomány mellé legyen másolva. Ehhez a Copy to Output Directory résznél állítsuk át a fájl tulajdonságát Copy if newer-re.

Készítsünk Feltoltes néven metódust, amely paraméterül kapja a megnyitandó fájl nevét, és visszatér egy IEnumerable<IEtel> típusú sorozattal, a fájlban található csokik példányaival. Mivel az IEtel interfészt a Csoki és a PremiumCsoki osztály is implementálja, ezért a visszaadott sorozat típusa lehet IEtel.

Amennyiben a metódus paramétereként kapott fájl nem létezik, akkor FileNotFoundException kivételt váltunk ki a megfelelő hibaüzenettel, és a Feltoltes metódus futása megszakad.

A példányok elkészítéséhez az EtelFactory osztály egy példányát használjuk.

A fájl tartalmát beolvassuk a memóriába, majd az első sort kihagyva, minden sorra meghívjuk a Factory metódust, a sorozat elemeit egyenként visszaadva.

```
1 reference
IEnumerable<IEtel> Feltoltes(string fajlNev)
{
    if (!File.Exists(fajlNev))
    {
        throw new FileNotFoundException($"A(z) {fajlNev} állomány nem található!");
    }
    EtelFactory etelFactory = new EtelFactory();
    foreach (var item in File.ReadAllLines(fajlNev).Skip(1))
    {
        yield return etelFactory.Factory(item);
    }
}
```

b. Az egyes objektumok `ToString` metódusának használatával jelenítsd meg a csokik adatait, valamint a `MegfeleloMinoseg` tulajdonság segítségével azt, hogy jó vagy rossz minőségű csokit gyártanak-e! Hibadobás esetén jelenítsd meg a hibaüzenetet!

Készítsünk CsokiEllenorzes néven metódust, amely IEnumerable<IEtel> típusú paraméterként megkapja a létrehozott csoki objektumokat. Mivel a MegfeleloMinoseg tulajdonság kivételt dobhat, így a kiírást try … catch blokkba tesszük, és szükség esetén kezeljük a SilanyMinosegException kivételt a kapott hibaüzenet kiírásával.

```
1 reference
void CsokiEllenorzes(IEnumerable<IEtel> csokik)
{
    foreach (var item in csokik)
    {
        try
        {
            Console.WriteLine($"{item} - " +
                $"{(item.MegfeleloMinoseg?"jó":"rossz")} minőségű");
        }
        catch (SilanyMinosegException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

A 2 metódus megírása után már csak a meghívásuk van hátra. Mivel mindkét metódus kiválthat kivételt, ezért a metódusok meghívását try … catch blokkba tesszük. A SilanyMinosegException-t a CsokiEllenorzes metódus kezeli, viszont a FileNotFoundException kivételt itt kell kezelni.
Ne felejtsük el, hogy ezt a programkódot a fájlban a metódusok előtt kell elhelyezni!

```
try
{
    List<IEtel> csokik = Feltoltes("input.txt").ToList();
    CsokiEllenorzes(csokik);
}
catch (FileNotFoundException ex)
{
    Console.WriteLine(ex.Message);
}
```

c. Egészítsd ki a futtatható programodat úgy, hogy felkészíted a többi hibalehetőség kezelésére is!

7. Szorgalmi feladatként készíts az alkalmazáshoz unit teszteket!