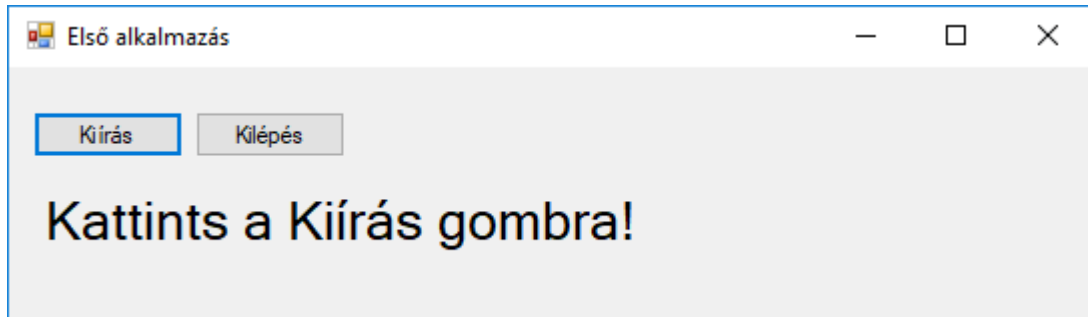
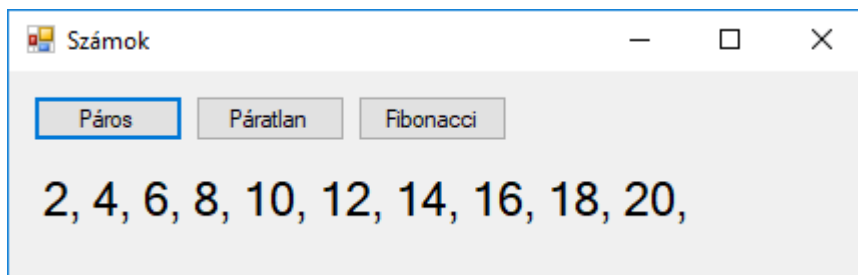


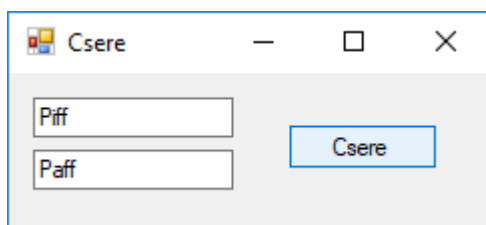
1. Hozzunk létre egy alkalmazást, amelyen két gomb lesz és egy címke. Az egyik gomb megnyomásakor átírja a címke feliratát „Üdvözöllek!”-re, a másik gomb megnyomására kilép a programból.



2. Próbáljunk meg készíteni egy alkalmazást, amelyen három gomb (Páros, Páratlan, Fibonacci) és egy címke szerepel. Az első gomb megnyomásakor a címke feliratát átírja az első 10 páros számra (2, 4, 6, ...), a második megnyomásakor az első 10 páratlan számra (1, 3, 5, ...), a harmadik megnyomásakor kiírja az első 10 Fibonacci számot (0, 1, 1, 2, 3, 5, 8, ... - mindegyik szám az előző kettő összege). A számokat ciklusok segítségével generáljuk ki.

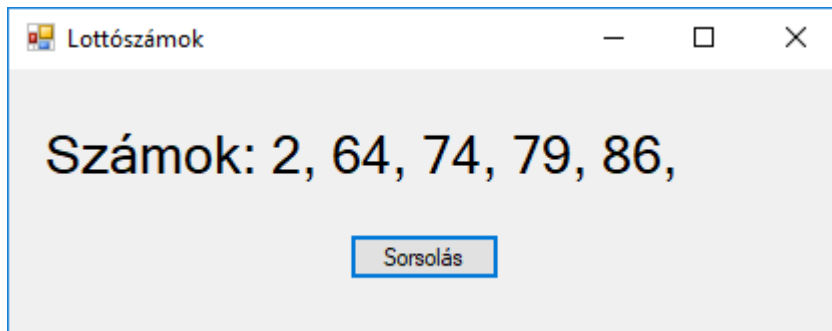


3. Jelenjen meg a képernyőn két beviteli mező és egy Csere feliratú gomb. A gombra kattintáskor a két beviteli mező tartalma cserélődjön meg.

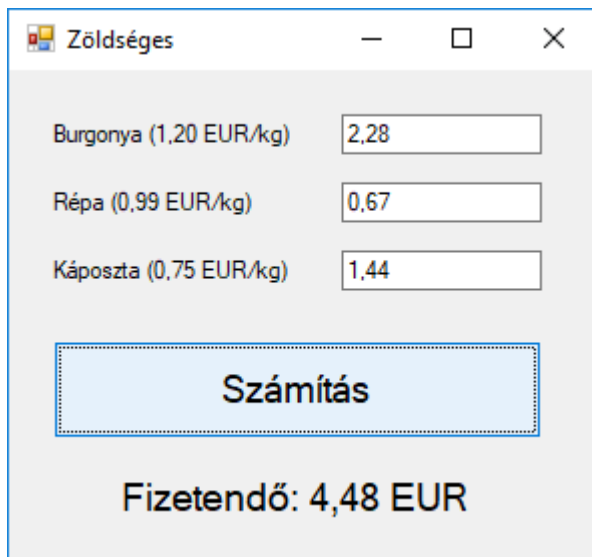


4. Készítsünk programot, amely egy címkét és egy nyomógombot tartalmaz. A gomb megnyomásakor a számítógép a címke feliratába írjon ki öt véletlen lottószámot 1-től 90-ig. A lottószámokat növekvő sorrendbe írjuk ki. Ügyeljünk arra, hogy az öt

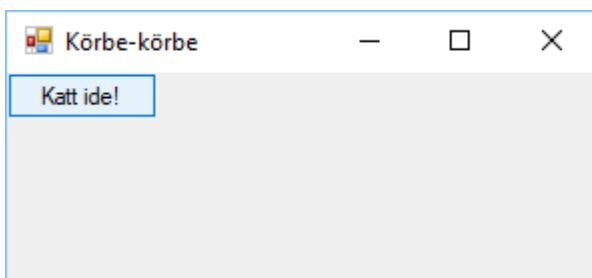
kigenerált lottószám különböző legyen!



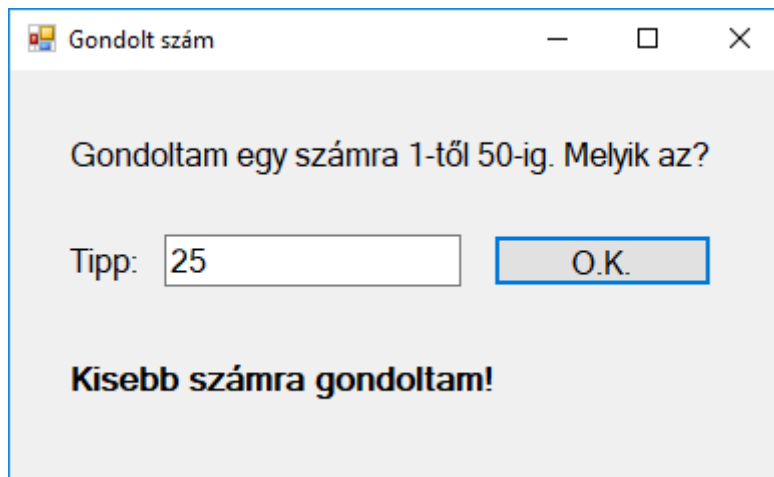
5. Zöldséges standunkon háromféle terméket árulunk: burgonyát, répát és káposztát. Egységárukat egy-egy címke jeleníti meg, a vásárolt mennyiséget egy-egy beviteli mezőbe írjuk. Egy gomb megnyomása után számítsuk ki és jelenítsük meg a fizetendő összeget (a végeredményt ne felejtjük el két tizedesjegyre kerekíteni)!



6. A programablak bal felső sarkában jelenjen meg egy nyomógomb. Ha a felhasználó rákattint, menjen a gomb a jobb felső sarokba, majd a jobb alsó, bal alsó, végül újra a bal felső sarokba, stb.



7. Találjuk ki a gép által gondolt egész számot tippeléssel, ha a gép minden tipp után megmondja, hogy a gondolt szám kisebb, nagyobb, vagy egyenlő-e a tippelt számmal! Ha a számot eltaláltuk, a számítógép gondoljon egy új számra és a játék folytatódjon.



Gondoltam egy számra 1-től 50-ig. Melyik az?

Tipp:

**Kisebb számra gondoltam!**

8. Készítsünk programot pizza elektronikus rendeléséhez! A kért összetevőket jelölőnégyzetekkel lehessen megadni. A program ezek alapján automatikusan a jelölés közben jelenítse meg a pizza árát (nyomógomb nélkül)!

Alkalmazásunkhoz keressünk képet az interneten és ezt jelenítsük meg.



**Pizza alapára: 2,50 EUR**

- ☐ Sonka (+0,80 EUR)
- ☐ Szalámi (+0,70 EUR)
- ☐ Kukorica (+0,75 EUR)
- ☐ Gomba (+0,60 EUR)
- ☐ Ananász (+0,45 EUR)

**Pizza ára: 2,50 EUR**



9. Készítsünk szoftvert kávé automatóához! Rádiógombokkal lehessen megadni az italt (kávé, tea, kakaó), jelölőnégyzetekkel a hozzávalókat (citrom, cukor, tej, tejszín). A szoftver számolja ki és jelenítse meg a fizetendő összeget! Teához ne lehessen tejszínt, kávéhoz citromot, kakaóhoz se citromot, se tejszínt kérni!

A számítást a program automatikusan végezze, nyomógomb megnyomása nélkül.

**Kávéautomata**

Ital:

- ☒ Kávé (0,35 EUR)
- ☐ Tea (0,25 EUR)
- ☐ Kakaó (0,45 EUR)

Hozzávalók:

- ☐ Citrom (+0,05 EUR)
- ☐ Cukor (+0,10 EUR)
- ☐ Tej (+0,05 EUR)
- ☐ Tejszín (+0,10 EUR)

**Fizetendő: 0,35 EUR**

10. A képernyőn jelenjen meg három vízszintes görgetősáv, amely segítségével az RGB színmodell három alapszínét lehessen beállítani 0 és 255 között. A kikevert szín egy címke háttérében jelenjen meg!

**Színkeverés**

Piros: 255

Zöld: 238

Kék: 195

11. Készítsünk csúszkás számológépet! A kért számot egy-egy vízszintes görgetősáv tologatásával lehessen bevinni, majd a megfelelő nyomógombra (feliratuk: Összeadás, Kivonás, Szorzás, Osztás) való kattintáskor jelenjen meg egy címkében az eredmény!

**Csúszkás számológép**

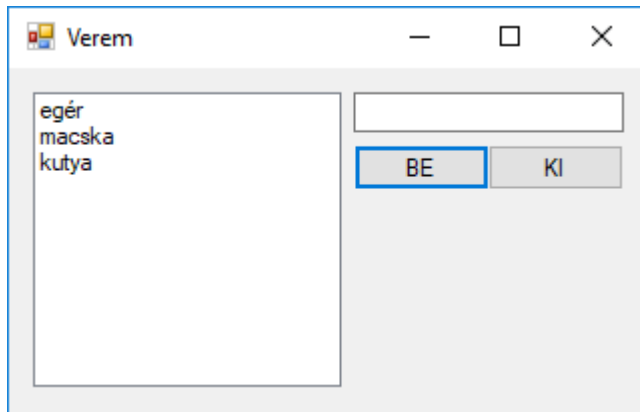
Első szám: 26

Második szám: 50

Összeadás Kivonás Szorzás **Osztás**

**Eredmény: 0,52**

12. Verem demonstrálása: készítsünk egy alkalmazást, amely tartalmaz egy listát és egy beviteli mezőt. A beviteli mező adata a BE gomb hatására kerüljön a lista tetejére, míg a KI gomb hatására a lista felső eleme kerüljön a beviteli mezőbe, és törlődjön a listáról. A lista legfeljebb 10 elemű lehet. Ha a lista tele van vagy üres, akkor a megfelelő gomb hatására kapjunk hibajelzést (üzenetablakot)!



13. Készítsünk egy **Haromszog** osztályt, amely konstruktorában megadható a háromszög három oldalának hossza (ezeket az osztály privát adatmezőkben tárolja). Az osztály tartalmazzon **GetA()**, **GetB()**, **GetC()** publikus metódusokat, amelyekkel lekérdezhetők az oldalhosszak. Az osztálynak legyenek továbbá a következő publikus metódusai:

- **Egyenloszaru()** – igaz/hamis értéket ad vissza attól függően, hogy a háromszög egyenlő szárú-e.
- **Szabalyos()** – igaz/hamis értéket ad vissza attól függően, hogy a háromszög szabályos-e.
- **Kerulet()** – kiszámolja a háromszög kerületét.
- **Terulet()** – kiszámolja a háromszög területét (Herón-képlet segítségével).

Készítsünk alkalmazást, melyben létrehozunk egy objektumot a Haromszog osztályból, majd kipróbáljuk annak metódusait.

14. Hozzunk létre egy **Fenykepalbum** osztályt. Ennek legyen egy kívülről nem látható adatmezője: **lapokSzama**, melyet egy publikus **GetLapokSzama** metódussal lehessen lekérdezni. Az osztály alapértelmezett (paraméterek nélküli) konstruktora **16** lapból álló albumot hozzon létre. Az osztályhoz tartozzon még egy konstruktor, amely segítségével tetszőleges számú lapból álló album létrehozható.

Az öröklést felhasználva hozzunk létre egy **NagyFenykepalbum** osztályt, amelynek konstruktora **64** lapból álló albumot hoz létre.

Készítsünk alkalmazást, melyben létrehozunk három objektumot: egy albumot az alapértelmezett konstruktor segítségével, egy 24 lapból álló albumot és egy NagyFenykepalbum-ot. Mindhárom album létrehozása után írjuk ki azok oldalszámát.

15. Készítsünk egy **OkosTomb** osztályt, amely egész számokat fog tartalmazni.

- Az osztály konstruktorának paraméterében lehessen megadni az okos tömb méretét (ez alapján hozza létre és állítsa be az elemek tárolásához szükséges privát adatmezőt). A tömb elemeinek értéke kezdetben 0 legyen.
- Az OkosTomb elemeit kizárólag a **GetOkosTomb()** metódus segítségével lehessen lekérdezni, amely visszatérési értéke egy egész szám típusú tömb legyen.
- Az osztály tartalmazzon továbbá egy **Kiiras(TextBox tbox)** metódust is, amely paraméterében megadott TextBox komponensbe kiírja a tömb elemeit vesszővel elválasztva.
- Az osztályban legyen egy **Generalas()** metódus, amely segítségével kigenerálhatók a tömb elemei a  $[0, 100]$  zárt intervallumból.
- Az osztályban legyen egy másik **Generalas(int min, int max)** metódus is, amely segítségével kigenerálhatók a tömb elemei a  $[\text{min}, \text{max}]$  zárt intervallumból.
- Az osztály tartalmazzon még **Min()**, **Max()**, **Osszeg()**, **Atlag()** metódusokat is, amelyek segítségével meghatározható a tömb legkisebb eleme, legnagyobb eleme, elemek összege és az elemek átlaga.

Készítsünk alkalmazást, melyben létrehozunk egy objektumot az OkosTomb osztályból, majd kipróbáljuk annak metódusait.

16. Készítsünk egy **Pont** osztályt!

- Az osztálynak legyen egy **konstruktor**a két paraméterrel, melyekkel megadhatók a pont **x** és **y** koordinátái (double típusúak legyenek).

- Az osztálynak legyen egy **konstruktor**a négy paraméterrel is: **xMin**, **xMax**, **yMin**, **yMax** értékek. A pont koordinátái ebben az esetben olyan **x**, **y** véletlen valós számok legyenek (double típus), melyekre igaz: **xMin** <= **x** <= **xMax** és **yMin** <= **y** <= **yMax**.
- Az osztály tartalmazzon még **GetX()**, **GetY()** metódusokat is, melyek visszaadják a pont **x** ill. **y** koordinátáit.
- Az osztálynak definiáljuk felül a **ToString()** metódusát, amely visszaadja a pont koordinátáit "(x,y)" alakban.
- Az osztálynak legyen egy **TavolsagOrigotol()** metódusa is, mely visszaadja a pont távolságát az origótól.

Teszteljük le a létrehozott osztályt! Készítsünk alkalmazást, melyben:

- Hozzunk létre egy tömböt, amely 20 darab **Pont** objektumot tartalmaz. A pontok **x** koordinátája (-100, +100) közötti véletlen szám, **y** koordinátája (-50, +50) közötti véletlen szám legyen. Írjuk ki a létrehozott pontok koordinátáit és az origótól való távolságukat.
- Számítsuk ki és írjuk ki a pontok átlagtávolságát az origótól és a legtávolabbi pont adatait (koordinátáit és távolságát).

17. Készítsünk **Kör** osztályt, melynek konstruktorában megadhatjuk a kör középpontjának koordinátáit (**x**, **y**) és a kör **sugarát** (mindhárom double típusú legyen). Az osztály tartalmazzon még egy további metódust is, amely megadja egy másik körrel való kölcsönös helyzetét:

```
public string KolcsonosHelyzet(Kor k)
{
}
}
```

A metódus az alábbi mondatok valamelyikét adja vissza:

- „A két körnek nincsenek közös pontjaik.”
- „A két kör kívülről érinti egymást.”
- „A két kör metszi egymást.”

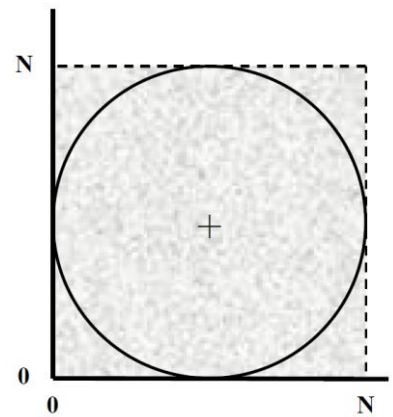
- „Az egyik kör belülről érinti a másikat.”
- „Az egyik kör a másik belsejében található.”

Teszteljük le a létrehozott osztályt: hozzunk létre 10 véletlenszerűen kigenerált kört ( $x$ ,  $y$  és sugár 0 és 20 között legyen). Írjuk ki, hogy ezek a körök milyen viszonyban állnak az  $(5, 5)$  középpontú és 5 sugarú körrel.

18. Bővítsük az előző feladatban létrehozott **Kör** osztályunkat, felhasználva a 16. feladatban létrehozott **Pont** osztályt is. Hozzunk létre a **Kör** osztályban egy **ElemekKörnek(Pont p)** metódust, amely igaz vagy hamis értéket ad vissza attól függően, hogy a paraméterében megadott pont eleme-e a körnek.

Készítsünk alkalmazást, melyben felhasználjuk a létrehozott **Kör** és **Pont** osztályokat:

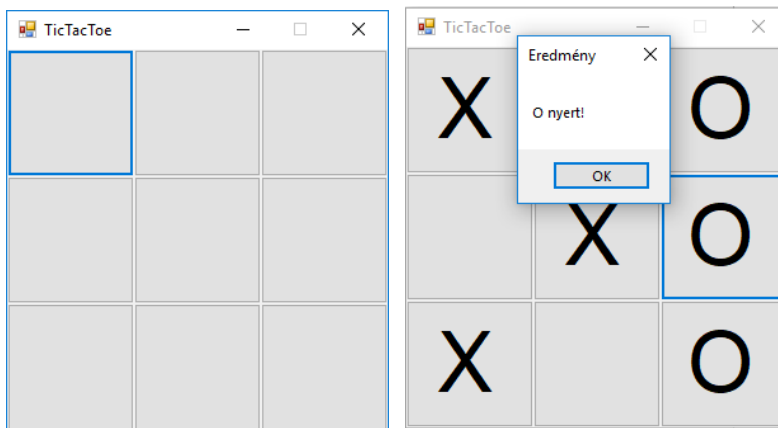
- Az alkalmazásban adjunk meg egy **N** és egy **osszPontokSzama** egész számot (**TextBox**-ok segítségével).
- Hozzunk létre a **Kör** osztályból egy olyan objektumot, melynél a kör középpontjának az  $(N/2, N/2)$  pontot adjuk meg és a kör sugarának hossza  $N/2$  (lásd ábra)
- Hozzunk létre egymás után **osszPontokSzama** darab véletlen pontot (**Pont** objektumot) melyek a  $(0, 0, N, N)$  koordinátákkal megadott négyzetbe esnek (ábrán a szürke négyzet). A létrehozott pontokat ne tároljuk tömbben, elég, ha egyszerre csak egy **Pont** objektumunk van.
- Mindegyik ponthoz határozzuk meg a kör objektumunk **ElemekKörnek** metódusának segítségével, hogy eleme-e a körnek. Határozzuk meg a körbe eső pontok darabszámát.
- Mennyi az értéke a  $4 * \text{korbeEsoPontokSzama} / \text{osszPontokSzama}$  kifejezésnek, ha  $N = 100$  és  $\text{osszPontokSzama} = 10\,000\,000$  ?
- Vajon miért ennyi ez az érték?





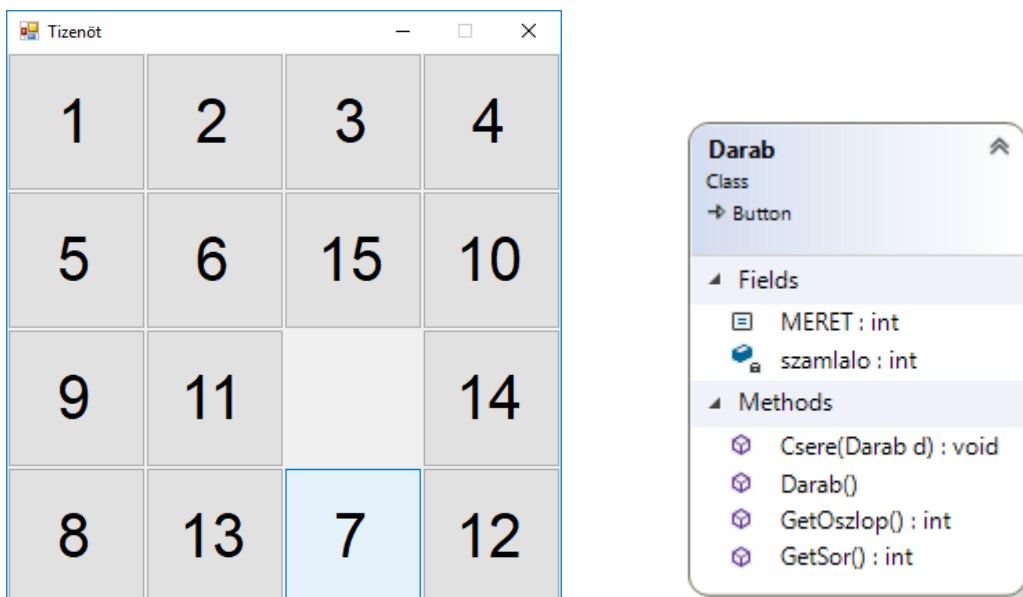
## 19. Készítsünk „TIC-TAC-TOE” játékot!

A játéktáblán hozzunk létre kilenc nyomógomb (Button) futási időben, az ablak Load eseményének bekövetkezésekor. Állítsuk be a gombok méretét, helyét és betűméretét. A játékot két játékos játssza, tehát a gombokra kattintva felváltva rakjuk ki az X és O jelet (természetesen csak akkor, ha még nincs rajta jel). Gondoljuk át, hogyan tudjuk a lehető legegyszerűbben ellenőrizni, hogy nincs-e valamelyik jeltől három darab egymás mellett, alatt, vagy átlósan. Ha vége a játéknak, írjunk ki egy üzenetablakot az eredménnyel (O nyert, X nyert, döntetlen).



## 20. Készítsünk „TIZENÖT” játékot!

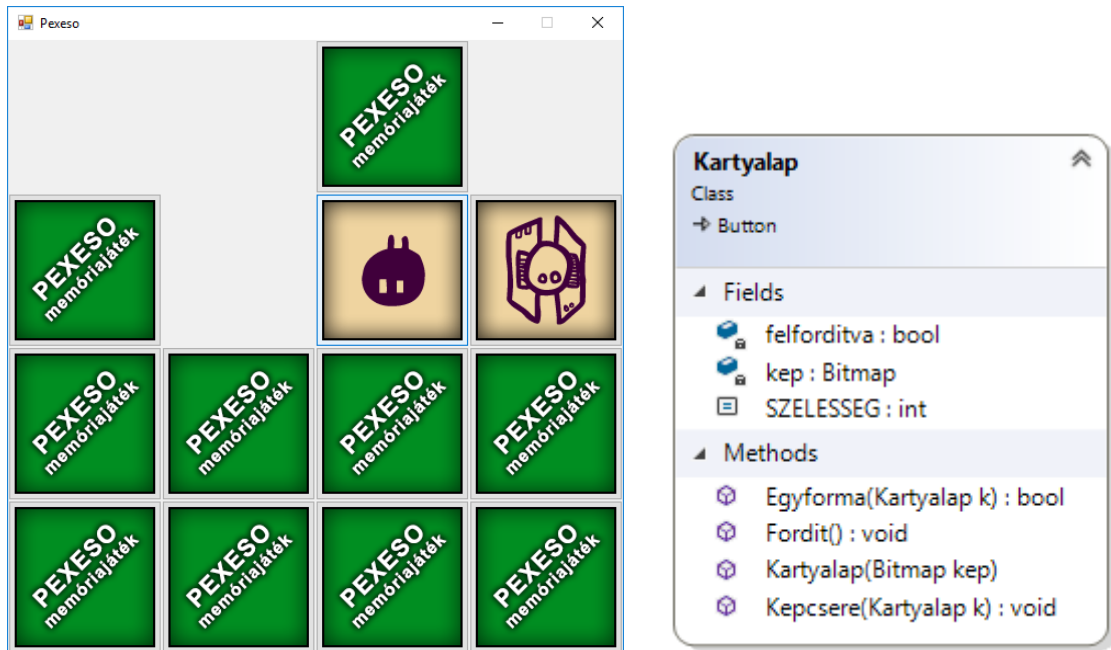
A játéktáblán található 16 négyzetet („Darab” objektumot) a Button osztályból származtassuk. Gondoljuk át, milyen új adatmezőket és metódusokat tartalmazzanak a játékdarabjaink (egy lehetséges megoldás a jobb oldali ábrán látható).



## 21. Készítsünk 4x4-es „PEXESO” memóriajátékot!

Ehhez felhasználhatjuk a „pexeso\_kepek.zip” állományban található képeket.

A memóriajáték kártyalapjait a `Button` osztályból származtassuk. Gondoljuk át, milyen új adatmezőket és metódusokat tartalmazzanak a kártyalapjaink (egy lehetséges megoldás a jobb oldali ábrán látható).



## 22. Készítsünk **MertaniTest** absztrakt osztályt, amely tartalmaz egy absztrakt **Terfogat** metódust. Az osztály **ToString** metódusa adja vissza hogy milyen nevű leszármaztatott osztályból lett létrehozva az objektum (**GetType().Name**) és a mértani test térfogatát.

Az absztrakt **MertaniTest** osztály valósítsa meg az **IComparable** interfészt. Egy mértani test akkor legyen nagyobb egy másik mértani testnél, ha annak a térfogata nagyobb.

Az absztrakt **MertaniTest** osztályból származtassunk le **Teglatest**, **Gomb** és **Henger** osztályokat. Mindegyikhez készítsünk saját konstruktor, melyek paramétereiben megadhatjuk a test méreteit. Írjuk felül az absztrakt osztályban definiált **Terfogat** metódust.

Az alkalmazásunk indításakor (Load esemény) hozzunk létre egy tömböt, melyet véletlenszerűen töltünk fel **Teglatest**, **Gomb** és **Henger** osztályokból létrehozott objektumokkal.

Egy nyomógomb megnyomásakor rendezzük a tömböt az `Array.Sort` metódussal, majd írjuk ki a tömb elemeit egy `TextBox`-ba.

23. Az előző alkalmazásunk **MertaniTest** osztályát egészítsük ki absztrakt **Felszin** metódussal. Az osztály **ToString** metódusa írja ki a test felszínét is.

Ne felejtsük el a **Felszin** metódus megvalósításával kibővíteni a **Teglatest**, **Gomb** és **Henger** osztályokat is!

Az absztrakt **MertaniTest** osztály tartalmazzon egy privát belső **FelszinKomparer** osztályt, amely megvalósítja az **IComparer** interfészt, továbbá egy publikus statikus **FelszinSzerintiRendezes** metódust, amely visszaad egy **FelszinKomparer** objektumot.

Az alkalmazásunkban egy másik nyomógomb megnyomásakor rendezzük a tömböt az `Array.Sort` metódussal a testek felszíne szerint növekvő sorrendbe, majd írjuk ki a tömb elemeit a `TextBox`-ba.

24. Készítsünk **Konyv** osztályt, amely három adatmezőt tartalmaz: azonosítószám, szerző, cím. Természetes sorrendként (`Comparable<Konyv>` generikus interfészt használva) definiáljuk az azonosítószám szerinti rendezést. Definiáljunk két komparert is (`IComparer<Konyv>` generikus interfészt használva) a szerző, ill. a cím szerinti rendezéshez!

Ne felejtsük el megírni a **Konyv** osztály konstruktorát és felülírni a `ToString()` metódust!

Az alkalmazásunkban hozzunk létre könyveket tartalmazó tömböt, majd rendezzük a tömb elemeit azonosítószám, szerző, ill. cím szerinti sorrendbe. Mindegyik esetben írjuk ki a rendezett könyveket!

25. Készítsünk **Negyzetszamok** osztályt, amely megvalósítja az `IEnumerable` interfészt (továbbá egy **NegyzetszamokEnumerator** osztályt, amely megvalósítja az `IEnumerator` interfészt). Mindezt úgy készítsük el, hogy a **Negyzetszamok** osztály az alábbi módon legyen használható az alkalmazásunkban:

```

Negyzetszamok nsz = new Negyzetszamok(6);
textBox1.Clear();
foreach(int sz in nsz)
{
    textBox1.AppendText(sz + ", ");
}

```

Ebben az esetben a program kimenete az első 6 drb. négyzetszám:

**1, 4, 9, 16, 25, 36,**

26. Az előző feladathoz hasonlóan készítsünk **FibonacciSzamok** osztályt, amely megvalósítja az `IEnumerable` interfészt (továbbá egy **FibonacciSzamokEnumerator** osztályt, amely megvalósítja az `IEnumerator` interfészt). Mindezt úgy készítsük el, hogy a **FibonacciSzamok** osztály az alábbi módon legyen használható az alkalmazásunkban:

```

FibonacciSzamok fibsz = new FibonacciSzamok(7);
textBox1.Clear();
foreach(int sz in fibsz)
{
    textBox1.AppendText(sz + ", ");
}

```

Ebben az esetben a program kimenete az első 7 drb. Fibonacci szám:

**0, 1, 1, 2, 3, 5, 8,**

27. Készítsünk programot, amely végrehajtja az alábbi műveleteket és leméri azok idejét **List** és **LinkedList** objektumoknál. A lemért adatok alapján döntsük el mikor érdemes használni az egyiket és mikor a másikat.

- 150000-szer egy új elem hozzáadása a lista elejére;
- 150000-szer egy új elem hozzáadása a lista végére.

Az időméréshez a **System.Diagnostics** névtérben található **Stopwatch** osztály metódusait (**Start()**, **Restart()**, **Stop()**) és tulajdonságait (**Elapsed**) használjuk (online dokumentációban nézzünk utána a használatának).