Programozás II

gyakorló feladatok

Szenasi.sandor@nik.uni-obuda.hu

Feladat

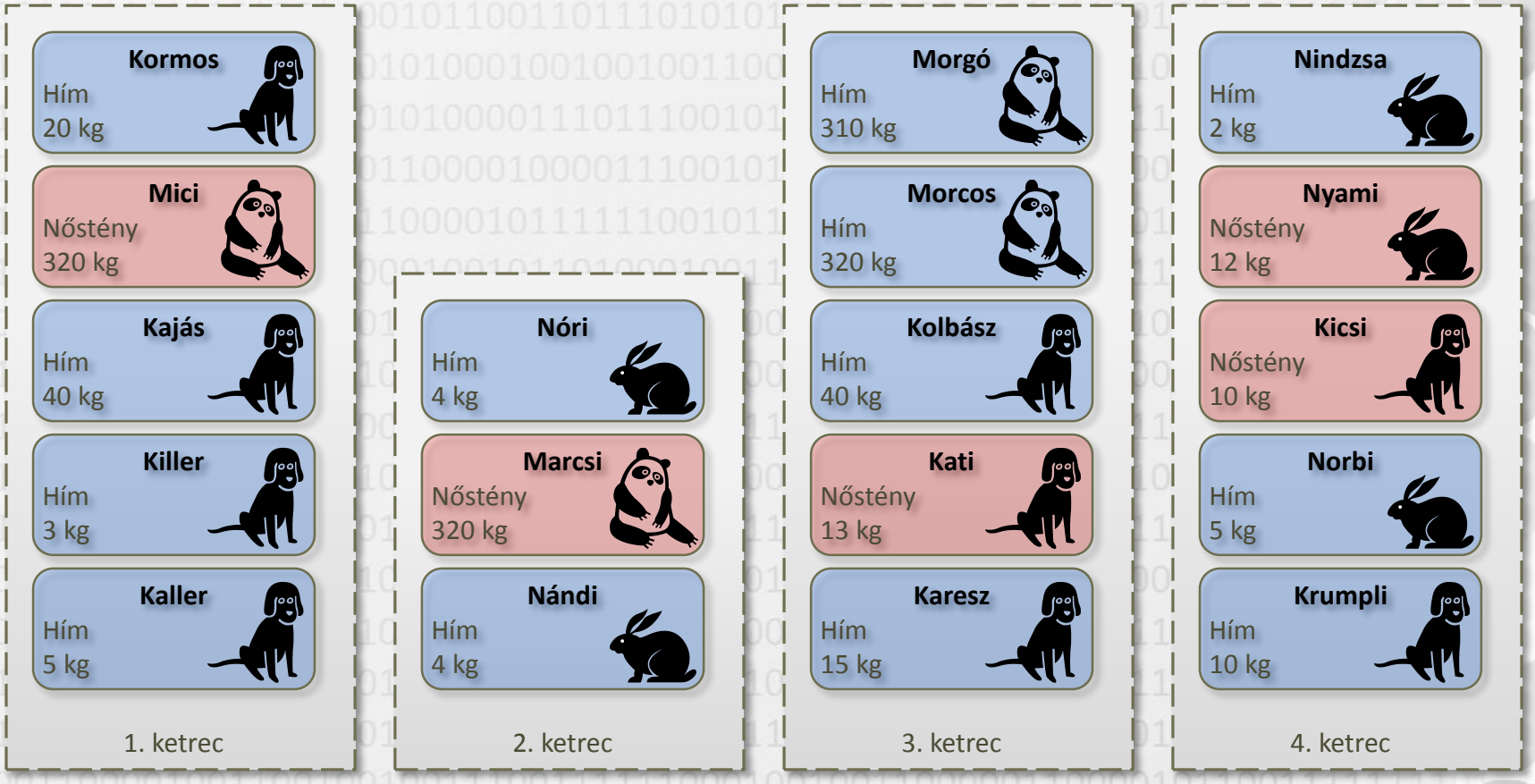
Programozási tételek összeépítése 1.

Szenasi.sandor@nik.uni-obuda.hu

Programozási tételek összeépítése

- **Feladat - mintaadatok**

  Állatkertünkben háromféle állat tárolunk különböző méretű ketrecekben az alábbi mintaadatok szerint:



Programozási tételek összeépítése

- **Feladat - kérdések**

  Válaszoljuk meg az alábbi kérdéseket az előző oldalon látható mintaadatokból felépített modell alapján.
  Az összetett feladatok során próbáljuk a már megismert programozási tételek (illetve az előzőleg elkészült részfeladatok) segítségével elkészíteni a megoldást. Tételek összeépítésénél használható mindhárom előadáson megismert összeépítési technika.

- **Egyszerű programozási tételek (N → 1)**
  - Megadott ketrecben hány darab megadott fajú állat található?
    int FajDarab(Allat[ ] A, AllatFaj faj)
  - Megadott ketrecben van-e megadott fajú és nemű állat?
    bool FajEsNemVanE(Allat[ ] A, AllatFaj faj, bool himnemu)
- **Egyszerű programozási tételek (N → N)**
  - Megadott ketrecben melyek a megadott fajú állatok?
    Allat[ ] FajAllatok(Allat[ ] A, AllatFaj faj)

Programozási tételek összeépítése

- **Programozási tételek összeépítése**
  - Megadott ketrecben mennyi a megadott fajú állatok átlagos tömege?
    float AtlagFajTomeg(Allat[ ] A, AllatFaj faj)
  - Megadott ketrecben melyik a legnehezebb megadott fajú állat?
    Allat FajLegnehezebb(Allat[ ] A, AllatFaj faj)
  - Megadott ketrecben hány (a ketrecen belül a saját fajára számított) átlagosnál nehezebb állat van?
    int AtlagnalNehezebbDarab(Allat[ ] A)
  - Melyik ketrecben van a legtöbb (a ketrecen belül a saját fajára számított) átlagosnál nehezebb állat?
    int LegtobbAtlagnalNehezebb(Allat[ ][ ] A)
  - Hány olyan ketrec van, ahol az előzőleg kiszámolt számú átlagosnál nehezebb állat található?
    int LegtobbAtlagnalNehezebbDarab(Allat[ ][ ] A)

Szenasi.sandor@nik.uni-obuda.hu

Programozási tételek összeépítése

- **Programozási tételek összeépítése**
  - Melyik ketrecben található a legtöbb megadott fajú állat?
    int LegtobbFaj(Allat[ ][ ] A, AllatFaj faj)
  - Megadott ketrecben van-e legalább egy azonos fajú, de ellenkező nemű egyedekből álló páros?
    bool AzonosFajEllenkezoNemVanE(Allat[ ] A)
  - Megadott ketrecben tartozik-e mindenkihez legalább egy azonos fajú, de ellenkező nemű állat? (ahhoz nem ragaszkodunk, hogy mindenkihez egy kizárólagos pár tartozzon)
    bool AzonosFajEllenkezoNemMindenkinek(Allat[ ] A)
  - Hány olyan ketrec van, ahol van legalább egy azonos fajú, de ellenkező nemű tagokból álló páros?
    int AzonosFajEllenkezoDarab(Allat[ ][ ] A)
  - Hányas számú ketrecekben nincs egy azonos fajú, de ellenkező nemű egyedből álló pár se?
    int[ ] AzonosFajEllenkezoNemNincs(Allat[ ][ ] A)

Feladat

Programozási tételek összeépítése 2.

Szenasi.sandor@nik.uni-obuda.hu

Összetett feladatok megoldása

- **Feladat - mintaadatok**

Receptkönyv

besamel	tejberizs	kijevi	rostélyos	rántott hús
Liszt 50 g	Rizs 100 g	Vaj 10 g	Vaj 10 g	Hús 100 g
Vaj 50 g	Tej 100 g	Hús 100 g	Hús 100 g	Liszt 10 g
Tej 50 g	Tej 200 g	Vaj 50 g	Liszt 10 g	Tojás 10 g
Tej 100 g	Vaj 10 g	Hagyma 10 g	Hagyma 30 g	

Raktár

- Tej 300 g
- Vaj 300 g
- Liszt 300 g
- Rizs 300 g
- Hús 50 g

Vendégek

Teri	Feri	Mari	
szereti	szereti	szereti	allergiás
Hús	Vaj	Vaj	Tojás
	Liszt	Liszt	
	Rizs	Hús	

V 1.0

Összetett feladatok megoldása

- **Feladat - kérdések**

  Válaszoljuk meg az alábbi kérdéseket az előző oldalon látható mintaadatokból felépített modell alapján (zárójelben megadott osztályban).

- **Egyszerű segéd metódusok**
  - Receptkönyv egy megadott nevű receptjének a kiválasztása (ReceptKonyv)
    Recept ReceptKivalasztas(string nev)
  - Recept tartalmaz megadott alapanyagot? (Recept)
    bool TartalmazAlapanyagot(Alapanyag alapanyag)
  - Mennyi egy recept hozzávalóinak összesített mennyisége? (Recept)
    public float ReceptOsszMennyiseg( )
  - Hány recept nem tartalmaz egy megadott alapanyagot? (ReceptKonyv)
    int HanyReceptNemtartalmazAlapanyagot(Alapanyag alapanyag)
  - Melyik receptek tartalmaznak egy megadott alapanyagot? (ReceptKonyv)
    Recept[ ] AlapanyagotTartalmazoReceptek(Alapanyag alapanyag)
  - Megadott vendég allergiás-e egy megadott alapanyagra? (AllergiasVendeg)
    bool AllergiasRa(Alapanyag alapanyag)

Összetett feladatok megoldása

- **Receptek hozzávalóival kapcsolatos kérdések**
  - Recept hányféle egymástól különböző alapanyagot tartalmaz? (Recept)
    int KulonbozoAlapanyagokSzama( )
  - Melyik a legbonyolultabb recept (amelyik a legtöbb különböző alapanyagot tartalmazza)? (ReceptKonyv)
    public Recept LegbonyolultabbRecept( )
  - Az egyes hozzávalókból összesítve mennyire van szükség a recept elkészítéséhez? (Recept)
    public Hozzavalo[ ] HozzavalokOsszesitve( )
  - Megadott hozzávalókból elkészíthető-e a recept? (Recept)
    bool MegvalositathatoHozzavalokbol(Hozzavalo[ ] raktar)
  - Hányféle receptet lehet elkészíteni megadott hozzávalókból? (ReceptKonyv)
    int HanyfeleReceptValositathatoMegHozzavalokbol(Hozzavalo[ ] raktar)
  - Melyik recepteket lehet elkészíteni megadott hozzávalókból? (ReceptKonyv)
    Recept[ ] MelyikReceptekValositathatokMegHozzavalokbol(Hozzavalo[ ] raktar)

Összetett feladatok megoldása

- **Vendégekkel kapcsolatos kérdések**
  - Mennyire jónak értékel egy vendég egy receptet (az általa szeretett hozzávalók mennyiségének az összege)? (Vendeg)
    virtual float Ertekel(Recept recept)
  - Valósítsuk meg az értékelést az allergiás vendégek esetén is (ha allergiás bármelyik hozzávalóra, akkor 0, egyébként a szokásos! (AllergiasVendeg)
    override float Ertekel(Recept recept)
  - Megadott vendégnek melyik receptet ajánljuk? (ReceptKonyv)
    Recept SzemelyreszabottAjanlat(Vendeg vendeg)
  - Mennyire jó egy recept egy társaság számára (a megadott vendégek értékelésének átlaga)? (Recept)
    float MennyireSikeres(Vendeg[ ] vendegek)
  - Megadott társaság számára melyik a legsikeresebb recept? (ReceptKonyv)
    Recept LegsikeresebbRecept(Vendeg[ ] vendegek)

Feladat

Programozási tételek összeépítése 3.

Szenasi.sandor@nik.uni-obuda.hu

Programozási tételek – házi feladat

- **Házi Feladat**

  Egy 364 napos tömbben tároljuk az egy év (52 hét) alatt mért napi átlaghőmérsékleteket. Ezek ismeretében válaszoljunk az alábbi kérdésekre:

  - Hány napon érte el a hőmérséklet az éves maximum 90%-át?
  - Melyik volt az év legmelegebb hete (maximális heti átlaghőmérséklet)?
  - Hány olyan hét volt, amikor legalább egyszer fagyott?
  - Hány olyan hét volt, amikor minden nap fagyott?
  - Melyik napon volt a legnagyobb lehülés az előző naphoz képest?
  - Hány olyan nap volt, amikor előző és következő nap fagyott, de aznap nem?
  - Mikor volt a leghosszabb időszak, amikor folyamatosan esett a hőmérséklet?
  - Hányszor volt az évben kánikula? (legalább 3 napig 35°C feletti hőmérséklet)
  - Volt-e olyan min. 5 napos időszak, amely értékei megismétlődtek később?
  - Milyen hosszú volt az a leghosszabb időszak, amikor egyszer se fagyott?
  - Milyen hosszú volt az a leghosszabb időszak, amelyen belül egyszer se fagyott egymást követő 5 napon át?

Feladat

Programozási tételek összeépítése 4.

Szenasi.sandor@nik.uni-obuda.hu

Összetett feladatok megoldása – házi feladat

- **Házi Feladat**

  Tervezzük meg és implementáljuk egy egyszerű szerszámkölcsönző cég rendszerét, amely megvalósítja az alábbi funkciókat:

  – Tárolja az aktuális időpontot (az év hányadik napja), ami menüből léptethető
  – Tárolja a kölcsönözhető eszközök adatait (típus {porszívó, fúrógép, lángvágó}, állapot {0..1}, alapdíj mértéke 10 napra, késedelmi díj mértéke ezt követően naponta)
  – Tárolja a kölcsönző személyek adatait (név, aktuális kölcsönzési adatok – eszköz, kölcsönzés ideje {max. 5 db}, előjegyzési lista {max. 5 db})
  – Kölcsönzéskor a megadott típus alapján automatikusan válasszuk ki a legjobb állapotú, legdrágább eszközt és rögzítsük a kölcsönzést. Legyen lehetőség listázni az összes kölcsönzést
  – Ha nincs elérhető eszköz, vegyünk fel előjegyzést a legkorábban lejáró eszközre. Legyen lehetőség listázni az összes előjegyzést
  – Legyen lehetőség visszaadni egy eszközt, a program számolja ki az alapdíj és a késedelmi díj alapján a fizetendő összeget (illetve csökkentse az eszköz állapotát nap/100-al). Ha volt előjegyzés az eszközre, akkor automatikusan kerüljön át az új személyhez.
  – Legyen lehetőség meghosszabbítani egy kölcsönzést, de csak akkor, ha arra az eszközre még nincs előjegyzés
  – Selejtezéskor töröljük az eszközt és helyettesítsük a rá vonatkozó előjegyzéseket egy hasonlóval
  – Listázzuk eszköztípusonként, hogy aktuális napon mennyi bevételt hoznak a cégnek

  Töltsük fel a rendszert adatokkal, és menüvezérelt módon legyen lehetőség elérni a fenti funkciókat!

Feladat

Öröklés példa 1.

Öröklődés feladat (alap osztályok)

- **Feladat - Készítsük el az alább felsorolt osztályokat**
- **Tulajdonos osztály**
  - Kívülről írható/olvasható formában tárolja el a tulajdonos nevét
  - Biztonsági okokból ne lehessen belőle származtatni
- **BankiSzolgáltatás osztály**
  - A konstruktorban lehessen megadni a tulajdonost, ez a későbbiekben csak olvasható legyen
  - Ebből az osztályból ne lehessen közvetlenül példányosítani
- **Számla osztály**
  - Legyen a *BankiSzolgáltatás* osztály leszármazottja
  - Konstruktorában lehessen megadni a tulajdonost
  - Kívülről csak olvasható formában tárolja el az aktuális egyenleget
  - Egy *Befizet(összeg)* metódussal lehessen növelni az egyenleget
  - Legyen egy hasonló paraméterű, de nem implementált *Kivesz(összeg)* metódusa is, aminek a visszatérési értéke egy logikai érték

Öröklődés feladat (számlák)

- **HitelSzámla osztály**
  - Legyen a *Számla* osztály leszármazottja
  - A konstruktorban lehessen megadni a tulajdonos mellett a hitelkeret összegét, a későbbiekben ez csak olvasható legyen
  - Valósítsa meg úgy a *Kivesz(összeg)* metódust, hogy csak a hitelkeret mértékéig engedjen negatív számla egyenleget. Ellenkező esetben ne csökkentse az egyenleget és *hamis* visszatérési értékkel jelezze, hogy nem sikerült a kivétel
- **MegtakarításiSzámla osztály**
  - Legyen a *Számla* osztály leszármazottja
  - Kívülről írható/olvasható formában tárolja el a kamat mértékét
  - Az osztály egy statikus mezőjében tárolja el az alapértelmezett kamatot. Egy új megtakarítási számla létrehozásakor ez legyen a kamat kezdőértéke
  - A *Kivesz(összeg)* metódus ne engedje 0 alá csökkenni az egyenleget, visszatérési értéke jelezze, hogy sikerült-e a kivét
  - Legyen egy *Kamatjóváírás()* metódusa, ami jóváírja az esedékes kamatot

Öröklődés feladat (kártyák)

- **Kártya osztály**
  - Legyen a *BankiSzolgáltatás* osztály leszármazottja
  - A konstruktorban lehessen megadni a tulajdonos mellett a hozzá tartozó mögöttes számlát, illetve a kártya számát
  - A kártyaszám legyen kívülről olvasható, a mögöttes számla nem módosítható
  - Készítsen egy *Vásárlás(összeg)* metódust, ami a paraméterként megadott összeggel megpróbálja csökkenteni a mögöttes számla egyenlegét, és visszatérési értéke legyen ennek sikeressége
- **Számla osztály kiegészítése**
  - Egészítse ki a *Számla* osztály egy *ÚjKártya(kártyaszám)* metódussal, amely a leendő kártyaszámot várja paraméterként
  - A metódus hozzon létre egy új kártyát (az aktuális számlát és annak tulajdonosát adva meg a kártya adataiként) és legyen ez a metódus visszatérési értéke

Öröklődés feladat (bank)

- **Bank osztály**
  - Tároljon el tetszőleges számú számlát, ezek maximális számát a *Bank* konstruktorában lehessen megadni
  - Legyen egy *Számlanyitás(tulajdonos, hitelkeret)* metódusa, amelynek paraméterei egy Tulajdonos objektum és egy hitelkeret összeg. A hitelkeret összegének megfelelően hozzon létre hitel vagy megtakarítási számlát, ezt tárolja el, és ez legyen a metódus visszatérési értéke is
  - Legyen egy *Összegyenleg(Tulajdonos)* metódusa, amely visszaadja a paraméterként átadott tulajdonos számláinak összegyenlegét
  - Legyen egy *LegnagyobbEgyenlegűSzámla(Tulajdonos)* metódusa, amely visszaadja a megadott tulajdonos legnagyobb egyenlegű számláját
  - Legyen egy *Összhitelkeret()* metódusa, amely visszaadja a bank által az összes ügyfélnek adott hitelkeretek összegét
- **A fenti osztályok implementálását követően hozzon létre példa Tulajdonos és Bank objektumokat, majd próbálja ki a fenti funkciók működését**

Feladat

Visszalépéses keresés 1.

Visszalépéses keresés (1)

- **6. Feladat**

  Készítsünk egy visszalépéses keresésen alapuló Sudoku megoldó programot, amely a tábla üres helyeit kitölti az alábbi szabályok szerint:

  - Minden üres helyre egy szám írható 1..9 között
  - Egy sorban, illetve egy oszlopban nem szerepelhet kétszer ugyanaz a szám
  - A teljes tábla 3x3-as blokkokra oszlik, egy blokkon belül nem szerepelhet kétszer ugyanaz a szám (a tábla mérete 9x9 tehát összesen 9 blokkot tartalmaz)

- **Visszalépéses keresés használatához javasolt átalakítások**
  - Kétdimenziós tábla adatainak átalakítása részfeladatok sorozatává
  - Fixen megadott számok és a kitöltendő üres helyek szétválogatása

1		2	3	
3	5			2
	4		8	1
2		4	1	8
	3	1		3

Fix mezők: (0,0) (0,2) (0,3) (1,0) …

Üres mezők: (0,1) (0,4) (1,2) (1,3) …

Visszalépéses keresés (2)

- **Megvalósítandó függvények**
  - ft(szint, szám) függvény
    Visszatérési értéke igaz, ha az előre fixen beírt számok egyike sem zárja ki, hogy a **szint**edik részeredményhez tartozó mezőbe beírjuk a **szám** értéket
    bool ft(int szint, int szam)
  - fk(szint, szám, k, kszám) függvény
    Visszatérési igaz, ha a **k**. részeredményként választott **kszám** érték nem zárja ki, hogy a **szint**edik részeredményhez tartozó mezőbe beírjuk a **szám** értéket
    fk(int szint, int szam, int k, int kszam)
  - BackTrack(szint, címszerint VAN, E)
    Az előadáson megismert visszalépéses keresés algoritmus implementációja
    void BackTrack(int szint, ref bool VAN, int[ ] E)
  - Rekurziót indító metódus
    A fenti metódusok célszerűen nem publikusak, ezért készítsünk egy egyszerű, az algoritmus működésének (és bemenő paramétereinek) ismeretét nem feltételező indító metódust
    public bool MegoldasKereses()

Visszalépéses keresés (3)

- **Már megvalósított segéd metódusok**
  - A Pozicio osztály lenti metódusa segítségével egyszerűen eldönthető, hogy a paraméterként átadott két mező kizáró kapcsolatban áll-e egymással (nem tartalmazhatják ugyanazt a számot). Visszatérési értéke csak akkor igaz, ha a két mező egy sorban, egy oszlopban vagy egy blokkban van:
    static int Kizaroak(Pozicio p1, Pozicio p2)
  - A Sudoku osztály alábbi metódusai elvégzik a tábla szétbontását fix és kitöltendő mezők listájára, illetve az eredmény betöltését az eredeti táblába:
    void MezoSzetvalogatas( )
    void MezoOsszefuzesMegoldassal(int[ ] E)
  - A Program osztály rendelkezik egy statikus metódussal, amely segítségével ki lehet listázni egy tábla tartalmát a képernyőre:
    static void TablaKirajzolas(int[,] tabla)
  - A Program osztály rendelkezik egyéb statikus metódusokkal, amelyek visszaadnak egy-egy Sudoku feladvány táblát (a 0. mintatábla csak tesztelési célokat szolgál, ugyanis nem a játék szabályainak megfelelő méretű):
    static int[,] MintaTablaFeltoltes_?( )

Feladat

Visszalépéses keresés 2.

Rekurzió – házi feladat

- **Feladat**

  Különböző tárgyakat (Tárgy objektumok az alábbi tulajdonságokkal: Súly, Méret, Hasznosság) szeretnénk elhelyezni egy véges méretű hátizsákban (mérete: ZsákMéret). A magunkkal cipelni kívánt tárgyakat egy tömbben tároljuk el. Készítsünk egy visszalépéses keresésen alapuló algoritmust, ami választ ad az alábbi kérdésekre:

  – Hányféle elhelyezésre van lehetőségünk?
  – Hogyan tudjuk a lehető legtöbb tárgyat elhelyezni a zsákban?
  – Hogyan tudjuk a lehető leghasznosabb tárgyakkal telepakolni a zsákot?
  – Hogyan tudjuk a lehető legnagyobbnak tűnő, de mégis legkönnyebb zsákot összeállítani?

Feladat

Kivételkezelés 1.

Szenasi.sandor@nik.uni-obuda.hu

Saját kivételek készítése (1)

- **9. Feladat**

  Készítsük el egy egyszerű bank modelljét, ahol az ügyfelek számlái mellett lehetőség van csoportos beszedési megbízások kezelésére is

- **Sor osztály**

  Készítsük egy egyszerű sor adatszerkezetet az alábbi metódusokkal:

  – Legyen egy konstruktora, ahol paraméterként meg lehet adni az adatszerkezet méretét
  public Sor(int meret)
  – Egy metódus segítségével lehessen új objektumot berakni a sorba (Object típust, vagy annak leszármazottait). Amennyiben a fix méretű tömbbe már nem fér el több új elem, dobjon egy SorMegteltKivetelt
  public void Betesz(object elem)
  – Legyen lehetőség kivenni az elsőként berakott elemeket a sorból. Ha üres sornak hívják meg ezt a metódusát, dobjon egy SorUresKivetelt
  public Object Kivesz()
  – Egy tulajdonság mutassa meg, hogy üres-e a sor
  public bool Ures

Saját kivételek készítése (2)

- **BankSzamla osztály**

  Egy bankszámla objektum az alábbi adatokat tárolja: számla azonosítója, számlatulajdonos neve, aktuális egyenleg. Rendelkezzen a mezőket kiolvasó tulajdonságokkal, illetve az alábbi műveletekkel:

  – Egy metódus segítségével legyen lehetőség megadott összeggel megterhelni a számlát. Amennyiben az aktuális egyenleg ezt megengedi, akkor csökkentse annak értékét a paraméterként átadott összeggel, ellenkező esetben pedig dobjon egy SzamlanNincsFedezetKivetelt
  public void Terhel(int osszeg)

  – A SzamlanNincsFedezetKivetel objektum mindig tartalmazza, hogy melyik számlát nem sikerült megterhelni, illetve ki lehessen belőle olvasni a sikertelenül terhelni próbált összeget is
  public BankSzamla Szamla
  public int Terheles

- **Beszedes osztály**

  Készítsünk egy Beszedes nevű osztályt, ami tárolja egy beszedés adatait: szolgáltató neve (aki a terhelést indítja), ügyfél neve (akitől le akarják vonni a pénzt), terhelendő összeg (a szolgáltatás havi díja)

Saját kivételek készítése (3)

- **Bank osztály**

  Készítsük el a bankot modellező osztályt is, ami tetszőleges adatszerkezetekben tárolja az alábbi adatokat:

  - számlák: Szamla objektumok (a bank által vezetett számlák listája)
  - engedélyek: Megbízás objektumok (melyik ügyfél, melyik szolgáltatónak, milyen maximális összeggel adott automatikus beszedési lehetőséget)
  - beszedésiSor: Megbízásokat tartalmazó sor, amelyik tartalmazza a cégektől folyamatosan érkező terhelési kéréseket (a feldolgozás során olyan sorrendben kell majd őket feldolgozni, ahogyan érkeztek)

  – Ügyfél nyithasson új számlát egy név és egy nyitóösszeg megadásával (számlaszám automatikusan generálódjon)
  public void UjSzamlaNyitas(string tulaj, int nyitoosszeg)

  – Ügyfél tudjon engedélyt adni automatikus megbízás teljesítésre az ehhez szükséges adatok meghatározásával
  public void UjMegbizas(string szolgaltato, string ugyfel, int maxOsszeg)

  – Szolgáltató jelezhessen egy új beszedési igényt
  public void BeszedesFelvetele(string szolgaltato, string ugyfel, int osszeg)

Saját kivételek készítése (4)

- **Bank osztály**
  - Egy metódus meghívásával legyen lehetőség a sorban időközben összegyűlt beszedések feldolgozására. A bank ellenőrizze minden beszedés esetén annak jogosságát (engedélyek listája), illetve próbálja meg az ügyfél számláját megterhelni a kívánt összeggel (amennyiben az fedezet hiány miatt kivételt dobna, automatikusan próbálkozzon egy másik számlájával)
    public void BeszedesiSorFeldolgozasa()
  - A fenti metódus dobjon kivételeket az alábbi esetekben:
    - Amennyiben nincs a beszedés adatainak megfelelő (szolgáltató-ügyfél-legalább ekkora összeg) engedély az ügyféltől, akkor dobjon egy BeszedesNemEngedelyezettKivetelt, amely tartalmazza a problémát okozó beszedés objektumot
    - Abban az esetben, ha talált megfelelő engedélyt, de az ügyfélnek nincs számlája, vagy egyik számláján sincs elegendő fedezet, dobjon egy BeszedesNemTeljesithetoKivetelt, ami a beszedés objektum mellett tartalmazza a pontos okot is (nincs számla, nincs egyenleg)