Borkereskedő

Dokumentáció a Programozási módszertan elmélete című tárgy beadandó programjához

Magyar Attila
mattila@inf.elte.hu
2002.11.26.

Tartalomjegyzék:

Beadandó feladat

Az ELTE TTK Programtervező matematikus szakán a 7. félévben tanított Programozási Módszertan Elmélete című tárgy követelményei alapján a beadandó feladat egy objektum elvű módon megvalósított „mesterség-szimuláció”, azaz adott foglalkozást űző vállalkozó (például: ács) életét szimuláló alkalmazás.

Bármely objektum-orientált nyelv választható a megvalósítás eszközéül. A programot felhasználói dokumentációval , valamint megfelelő felhasználóbarát környezettel kell ellátni.

Borkereskedő

Ezesetben a megvalósítandó feladat egy borkereskedő szimulációja. Élete során bort termel, és eladja azt az üzletében. A szimuláció során csupán az eladásra összpontosítunk, azaz feltételezzük hogy a borászunknak rendelkezésére áll egy kifogyhatatlan „borforrás”, ez lesz az ő pincéje.

Természetesen borát nem a pincéjében kínálja, ehhez rendelkezik egy üzlettel, ahová a vendégeit várja. Az üzletben igyekezik a lehető legjobb feltételeket biztosítani a borkóstoláshoz és eladáshoz, ezért különféle kellékeket tart raktáron a különféle borok mellett. Ha ezen kellékek fogynak el, ezekből a szakboltban (borászboltban) vásárolhat, természetesen pénzért. Ilyen kellék például a sajt, vagy pohár.

Tehát adott a **borászunk**, aki rendelkezik egy kiinduló tőkével, és kifogyhatatlan mennyiségű borral. Elmegy az üzletébe,visz egy adag bort a pincéjéből, majd a szakboltból kellékeket is hoz. Ezután vendégek érkezésére vár, akik ittlétük alatt, bort kóstolnak (közben sajtot rágcsálnak, eltörik poharukat, vagy leöntik a terítőt), nagyobb mennyiséget rendelnek, és ha kiszolgálták őket kannákba töltött borukkal, fizetnek és távoznak. A borász a kiszolgálások közben szemmel tartja az üzlet raktárkészletét, és ha bármilyen bornak vagy kelléknek hiányát tapasztalja, elmegy a pincébe avagy a szakboltba hogy pótolja amit szükséges. Majd visszatér az üzletbe és várja az újabb vendégeket.

A szimulációban adott számú **vendég** létezik, akik visszatérő fogyasztók. Számuk legfeljebb hat lehet. Szintén rendelkeznek “pénztárcával”, és nekik van alkoholszintjük is. Ha nem az üzletben tartózkodnak, alkoholszintjük csökken, pénzük gyarapszik, közben adott valószínűséggel újra és újra visszatérnek az üzletbe borozgatni, ha az épp nyitva van. Kóstolgatás közben alkoholszintjük növekszik, rendelés után pénzük fogy a

fizetés által. Ha túl részegek vagy nem rendelkeznek elegendő pénzzel a rendelésük teljesítéséhez természetesen a borász nem szolgálja ki és kirúgja őket az üzletből.

Létezik még **ellenőr** is a programban, csak hogy színesítse az eseményeket. Időnként ellátogat az üzletbe és vizsgálódik. Ellenőrzi hogy a raktáron mindenből elegendő mennyiség áll-e rendelkezésre, valamint ellenőrzi hogy van-e részeg ember a helységben. A tapasztaltak alapján jutalmazza, vagy bünteti az üzlet tulajdonosát, de égbekiáltó szabályszegések esetén ki is üríttetheti a borozót. Az ellenőrnek hangulata is van, ami büntetések során javul, jutalmazások során pedig romlik.

A büntetés/jutalmazás mértékének meghatározásakor az ellenőr természetesen figyelembe veszi saját hangulatát is.

Állandók

A program néhány jellemzője előre meghatározott. Ilyenek a vendégek maximális száma, a különböző bor- és kellékfajták, vagy az ellenőr hangulatai. Ezek felsorolása látható alább:

Borfajták:

- Merlot
- Bikavér
- Kékfrankos
- Olaszrizling
- Aszú
- Szamorodni

Kellékek:

- Sajt
- Terítő
- Pohár
- Kanna

Vendégek:

- Borszakértő
- Tokaji borászkolléga
- Külföldi turista
- Környékbeli bácsi
- Vendégmunkás
- Szegedi tanár

Ellenőr hangulatai:

- Jókedvű
- Hétköznapi
- Rosszkedvű

A szimuláció indítása előtt lehetőség van elvégezni néhány kezdeti beállítást, melyeket aztán már futás közben nem módosíthatunk. A borász neve mellett megadhatjuk kiinduló pénzmennyiségét is (0 és 99999 között). A vendégek számát (azaz a borozónk férőhelyét) is állíthatjuk legfeljebb 6-ig, valamint hogy a vendégek mennyi pénzzel rendelkezzenek kezdetben (szintén 0 és 99999 között). Módosítható az ellenőr neve, és kezdeti hangulata is, éppúgy mint az összes épület elnevezése. Szintén 6-ig állítható a borász négy fő műveletének időtartama is (leltározás, kiszolgálás, borhozás, vásárlás), azaz hogy hány körig tartson, egy-egy művelet a felsoroltak közül.

Osztályok és objektumok

A feladatnak megfelelően a programban a következő osztályok szerepelnek:

- *Pince, Üzlet, Szakbolt* osztályok melyek származtathatók egy közös *Hely* osztályból,
- *Borász, Vendég, Ellenőr* osztályok melyek származtathatók egy közös *Ember* osztályból,
- *Kiszolgál, Borthoz, Vásárol* osztályok melyek származtathatók egy közös *Tennivaló* osztályból,
- valamint maguknak a boroknak és kellékeknek lesz egy összevont *Dolog* osztálya.

Pincéből, Üzletből, Szakboltból egy-egy objektum létezik a program futása során, valamint magának a *Hely* osztálynak lesz egy Sehol objektuma, itt tartozkodnak majd a *Vendégek* és az *Ellenőr* amikor nem az *Üzletben* vannak.

Borászból és *Ellenőrből* szintén csak egy-egy objektum szükséges, míg *Vendégből* annyi objektum szükséges ahány visszatérő vendéggel szeretnénk futtatni a szimulációt.

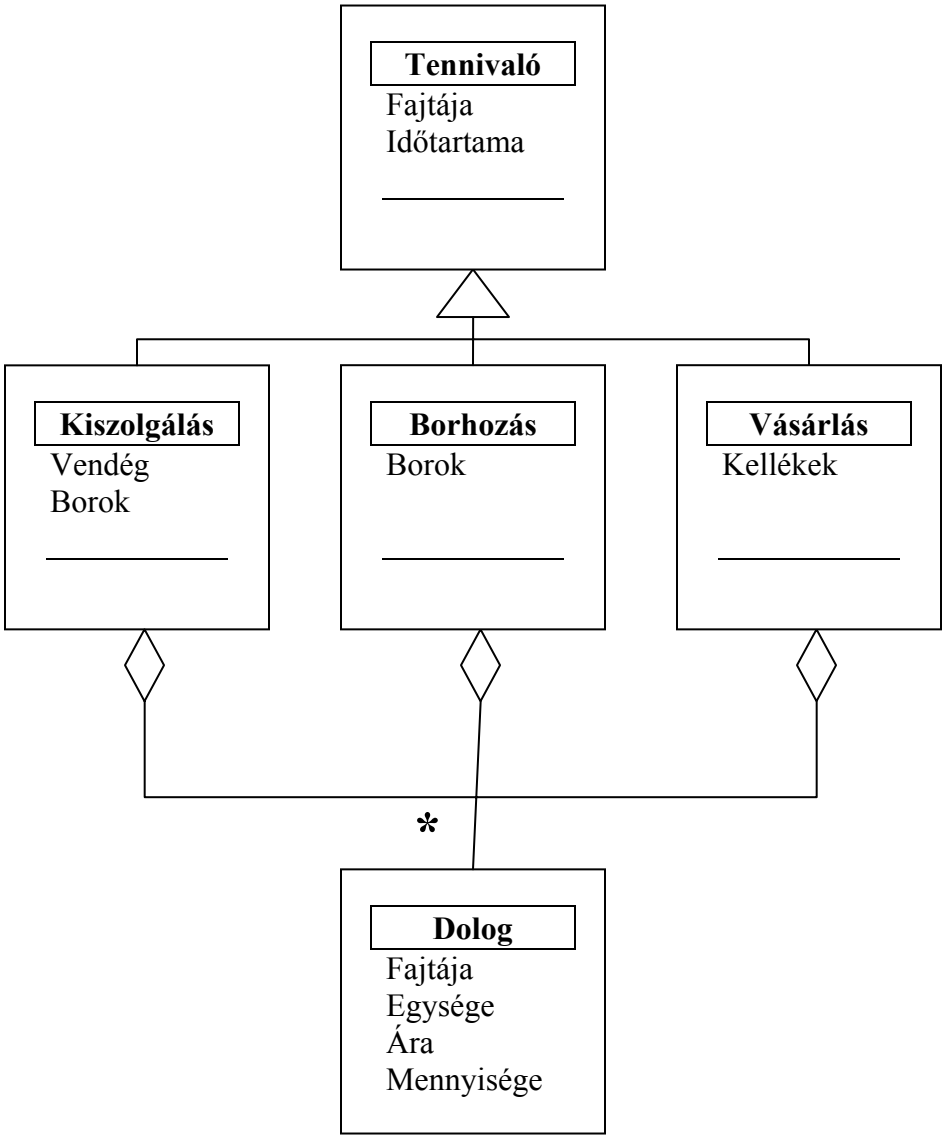
Kiszolgál, Borthoz, Vásárol típusú objektumok mind a borász tennivalói, ezért ezeket majd a *Borász* osztály fogja tartalmazni. *Kiszolgál* típusú objektumból mindig annyi létezik egy időpillanatban, ahány rendelés épp teljesítésre vár. *Borthoz* és *Vásárol* osztályokból egyszerre csak három-három objektum létezhet, ilyen típusú műveletekből a borász nem ad hozzá újabbakat tennivalólistájához, hogyha már elérték a maximális számukat. Egy-egy borhozás vagy vásárlás tennivaló mindig az aktuálisan fogyóban lévő borokra vagy kellékekre vonatkozik, azaz a borász mindig csak azokból hoz amire tényleg szükség is van.

UML Modellek

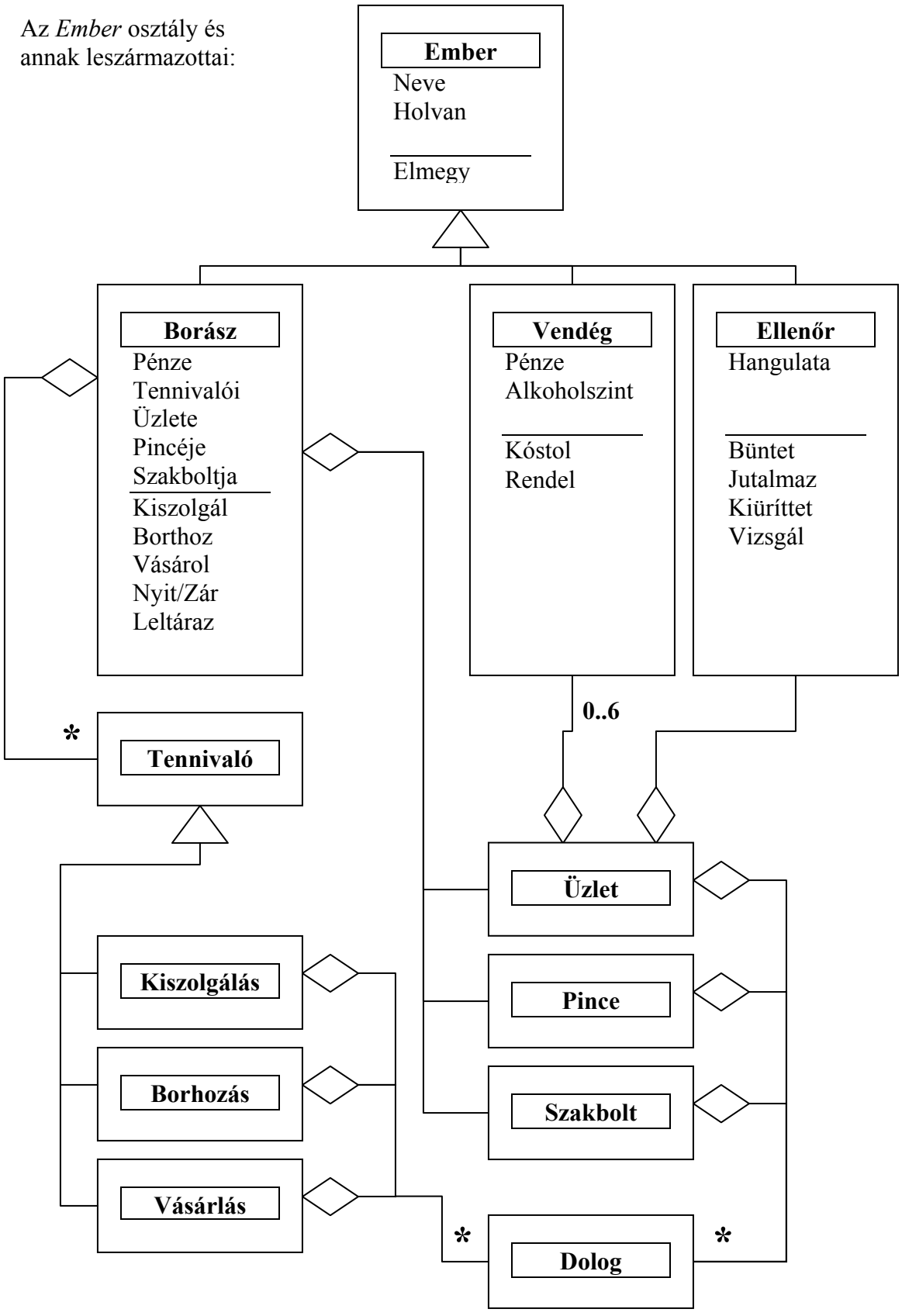
Az előző fejezet alatt felvázolt osztályokat többek közt a következő UML modellekkel és azok diagramjaival írhatjuk le:

Statikus modell

A *Tennivaló* osztály, három leszármazottjával (*Kiszolgálás*, *Borhozás*, *Vásárlás*), melyek tartalmaznak *Dolog* osztálybeli objektumokat (borok és kellékek). A leltár művelet egyszerűsége miatt (végrehajtása azonnali, paraméterei nincsenek) nem kapott külön osztályt, még csak nem is objektuma a *Tennivaló* osztálynak, egyszerű függvényként lett megvalósítva. Az osztálydiagrammok:

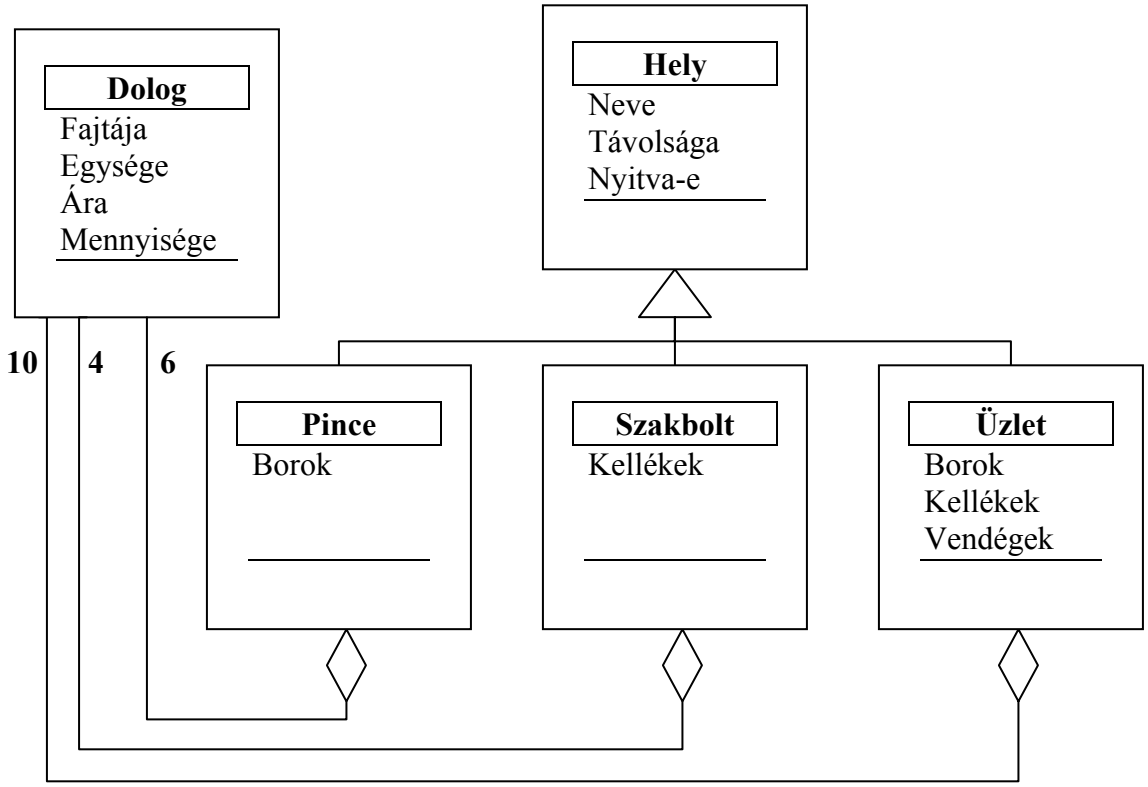


Az *Ember* osztály és annak leszármazottai:



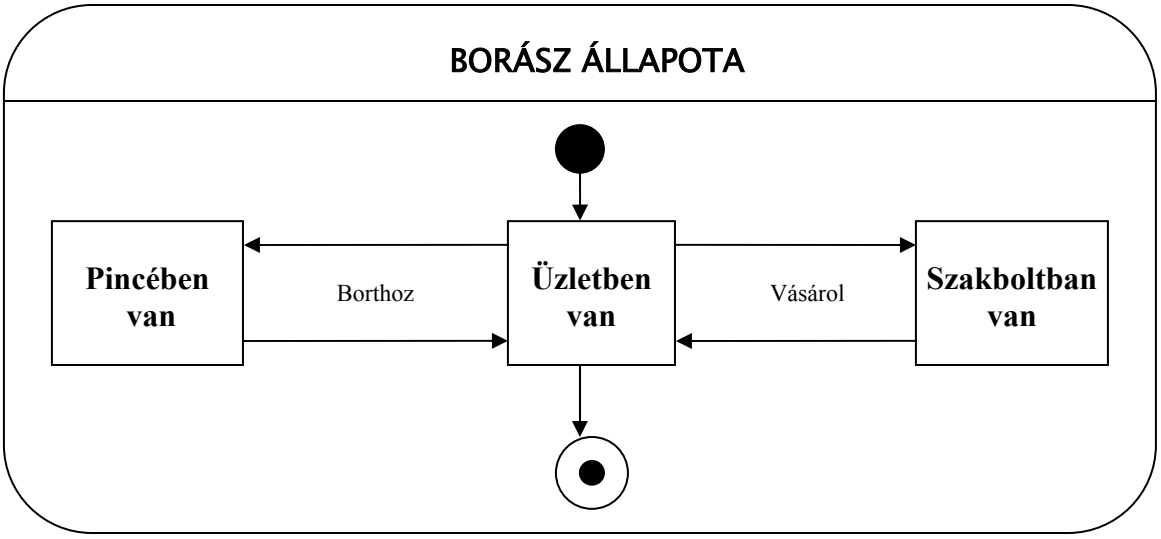
Ebből az összetettebb ábrából leolvasható, miszerint a *Borász*, *Vendég*, *Ellenőr* osztályok közös őse az *Ember*, és hogy egy *Borász* tartalmaz meghatározatlan számú *Tennivaló*t, valamint egy-egy-egy *Üzlet*-, *Pince*-, és *Szakbolt*beli objektumot. Továbbá hogy egy *Üzlet* tartalmaz egy *Ellenőr*t, és több *Vendég*et. Az ábra alsó része a két kisebb diagram egyszerűsített változata.

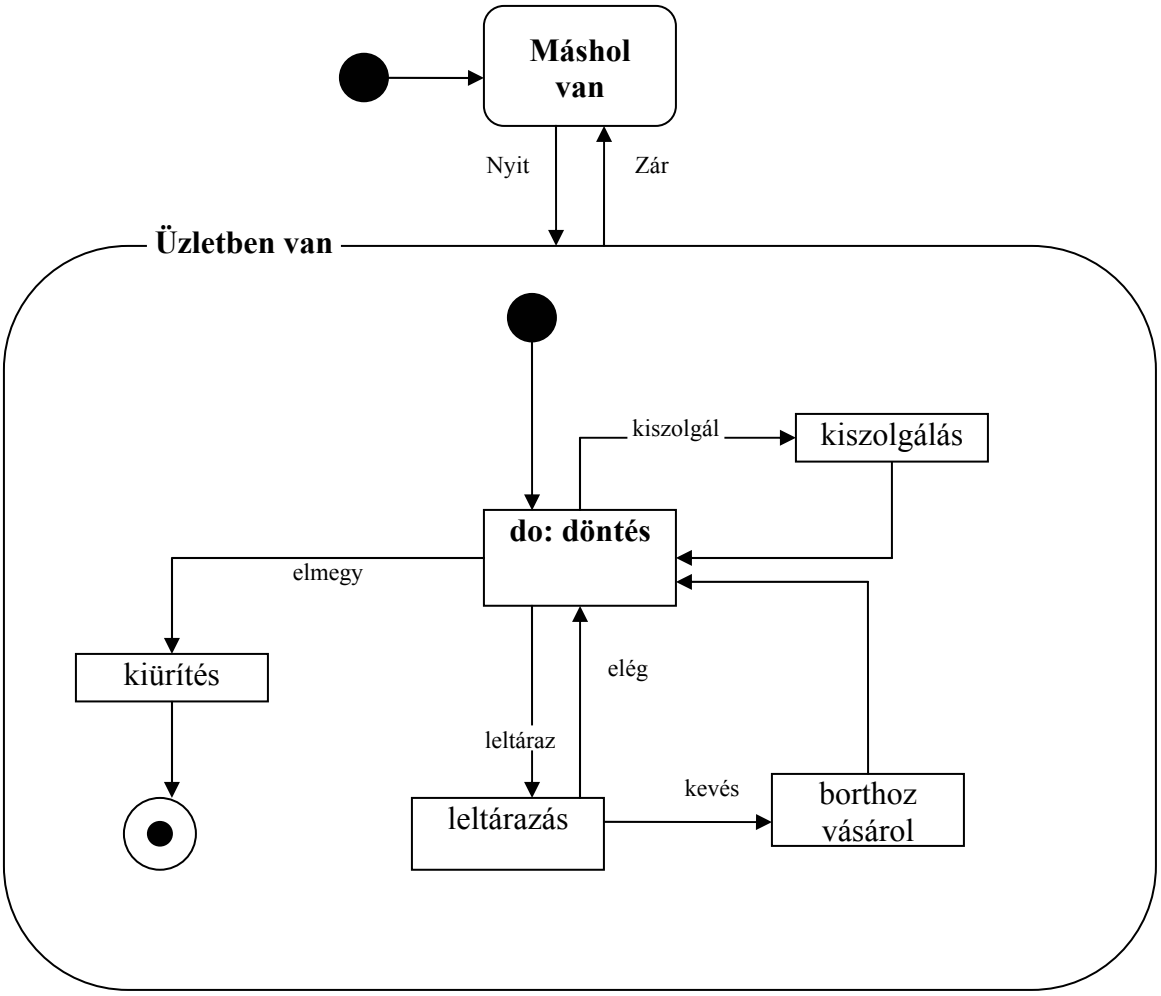
A *Hely* osztály diagramja hasonló ahhoz amit már a *Tennivaló* osztálynál láthattunk:



Dinamikus modell

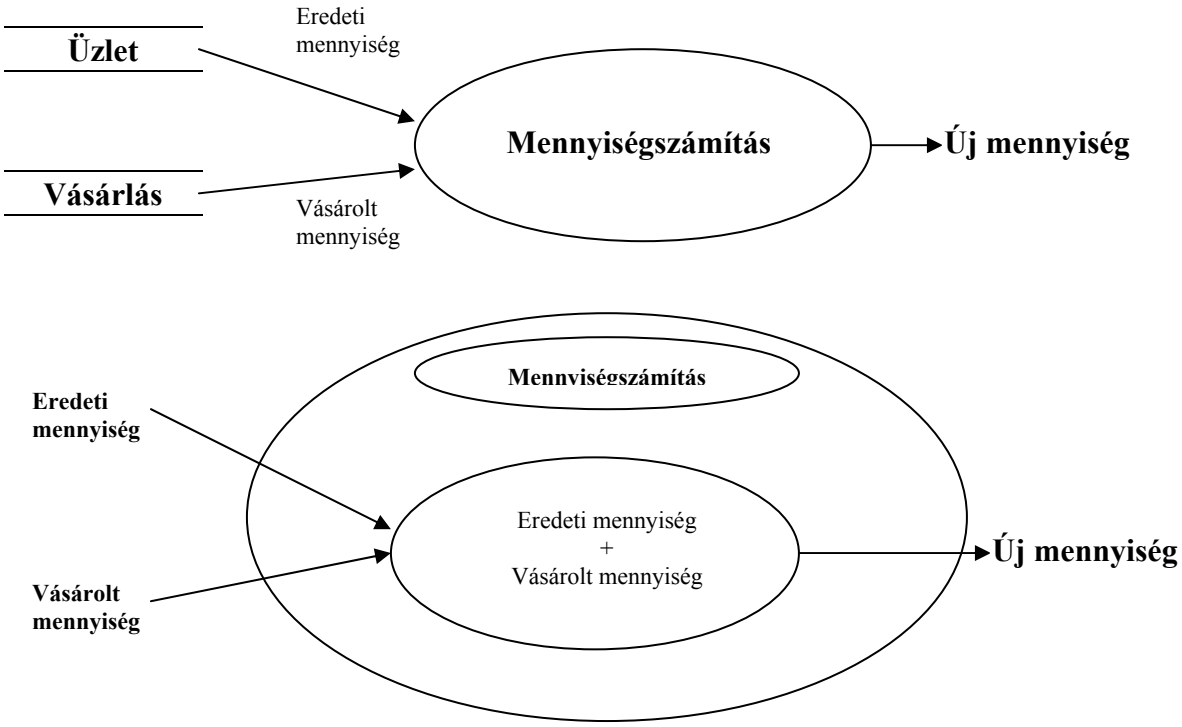
A program osztályhiearchiájának legösszetettebb elemét a *Borász*t kiemelve, megadom annak állapotdiagramját, részletezve azt az állapotot amikor az üzletében tartózkodik:


BORÁSZ ÁLLAPOTA
Pincében van
Borthoz
Üzletben van
Vásárol
Szakboltban van


Máshol van
Nyit
Zár
Üzletben van
kiszolgál
kiszolgálás
do: döntés
elmegy
kiürítés
elég
leltáraz
kevés
leltárazás
borthoz vásárol

Funkcionális modell

A funkcionális modellhez a mennyiségszámítás adatfolyam diagramját szemléltetem:



Környezet és megvalósítás

A feladat megvalósításához a Borland Delphi környezetét választottam, azaz a programot Object Pascal nyelven írom. Egyetlen *exe* fájlból áll majd, mely minden 32 bites Microsoft Windows® környezetben futtatható.

Indítás után a főablakot láthatjuk, melyről minden fontos információt leolvashatunk majd a szimuláció futása során. Három vezérlő gomb látható középen, *Elindít*, *Beállítások*, *Kilép* feliratokkal, melyekkel értelemszerűen a szimuláció indítását, kezdeti beállításait, és a programból való kilépést végezhetjük. Egy szimuláció mindig a beállítások gomb alatt megadott jellemzőknek megfelelő tulajdonságokkal indul. Futás közben a gombok megváltoznak, indítás és beállítás helyett egy *Léptet* gomb lesz látható, valamint kilép helyett egy *Leállít*. Értelemszerűen a szimuláció leállításával a főablak eredeti állapotához léphetünk vissza, míg a léptet gomb nyomogatásával léptethetjük a programot előre az időben. Közben a főablakban nyomon követhetjük az eseményeket, egyrészt a gombok feletti kiírólistán, másrészt a gombok alatt a borászunk és üzletének tulajdonságainak változásait figyelemmel követve. Ezen tulajdonságok között a különféle listákra kattintva, további információkat tartalmazó ablakokat nyithatunk meg, valamint

ellenőr érkezésekor automatikusan felbukkan egy vele kapcsolatos ablak, amely távozásakor el is tűnik.

A *Leállít* gombot bármikot megnyomva, leállíthatjuk az aktuális szimulációt. Újra előkerül a *Kilép* és a *Beállítások* gomb. Ezekután vagy módosítunk a paramétereken és új szimulációt indítunk, vagy a *Kilép* gombbal bezárjuk a programot.

Delphi forrás

Az osztályok adattagjainak és metódusainak (és azok típusainak) pontos megadásához mellékelem jelenlegi definíciójukat Object Pascalban:

```
INTERFACE

type                                    // DOLOG
   TDolog = class
   private
   Fajta : String;
   Egyseg : String;
   Mennyiseg : Integer;
   Ar : Integer;
   public
   constructor Init (f,e : String; m,a : Integer);
   function GetFajta : String;
   function GetEgyseg : String;
   function GetMennyiseg : Integer;
   function GetAr : Integer;
   procedure SetFajta (f : String);
   procedure SetEgyseg (e : String);
   procedure SetMennyiseg (m : Integer);
   procedure SetAr (a : Integer);
end;

type                                    // HELY
   THely = class
   private
   Nev : String;
   Nyitva : Boolean;
   public
   constructor Init (n: String; ny : Boolean); virtual;
   function GetNev : String;
   function GetNyitva : Boolean;
   procedure SetNev(n : String);
   procedure SetNyitva (ny : Boolean);
end;

type                                    // EMBER
   TEmber = class
   private
   Nev : String;
   Holvan : THely;
   public
   constructor Init (n : String); virtual;
   function GetNev : String;
```

```
   function GetHolvan : THely;
   procedure SetNev (n : String);
   procedure SetHolvan (h : THely);
   function Elmegy (h : THely) : Integer; virtual;
   function Kiir : String; virtual;              // csak a dinamikus kotes miatt
end;

type                                    // TENNIVALO
   TTennivalo = class
   private
   Fajta : String;
   Ido : Integer;
   public
   constructor Init (i : Integer); virtual;
   function GetIdo : Integer;
   procedure SetIdo (i : Integer);
   function GetFajta : String;
   procedure SetFajta (f : String);
end;

type                                    // SZAKBOLT
   TSzakbolt = class (THely)
   private
   Kellekek : TList;
   public
   constructor Init (n : String; ny : Boolean); override;
   function GetCountKellekek : Integer;
   function GetItemKellekek (i : Integer) : Pointer;
   function IsKellek(d : TDolog) : Boolean;
   function SearchKellek (d : TDolog) : Integer;overload;
   function SearchKellek (s : String) : Pointer;overload;
   function GetCountKanna : Integer;
   function GetCountSajt : Integer;
   function GetCountTerito : Integer;
   function GetCountPohar : Integer;
   function GetFirstKanna : TDolog;
   function GetFirstSajt : TDolog;
   function GetFirstTerito : TDolog;
   function GetFirstPohar : TDolog;
   procedure AddKellek (f,e : String; m,a : Integer);
   procedure DelKellek (i : Integer);
end;

type                                    // PINCE
   TPince = class (THely)
   private
   Borok : TList;
   public
   constructor Init (n : String; ny : Boolean); override;
   function GetCountBorok : Integer;
   function GetItemBorok (i : Integer) : Pointer;
   function SearchBor (b : String) : TDolog;
   procedure AddBor (f,e : String; m,a : Integer);
   procedure DelBor (i : Integer);
end;

type                                    // VENDEG
   TVendeg = class (TEmber)
   private
   Penz : Integer;
   AlkoholSzint : Integer;
   Rendelt : Integer;
   public
```

```
    constructor Init (n : String); override;
    function GetPenz : Integer;
    function GetAlkoholSzint : Integer;
    function GetRendelt : Integer;
    procedure SetRendelt (i : Integer);
    procedure SetPenz (p : Integer);
    procedure SetAlkoholSzint (a : Integer);
    function Kiir : String; override;          //csak a dinamikus kotes miatt

    function Kostol (b : TDolog) : Integer;
    function Rendel (b : TEmber) : TTennivalo;
    function Elmegy (h : THely) : Integer; override;
end;

type                                    // UZLET
    TUzlet = class (THely)
    private
    Kellekek : TList;
    Borok : TList;
    Vendegek : TList;
    VanEllenor : Boolean;
    public
    constructor Init (n : String; ny : Boolean); override;
    function GetCountKellekek : Integer;
    function GetItemKellekek(i : Integer) : Pointer;
    function IsKellek(d : TDolog) : Boolean;
    function SearchKellek (d : TDolog) : Integer; overload;
    function SearchKellek (s : String) : Pointer; overload;
    function GetCountKanna : Integer;
    function GetCountSajt : Integer;
    function GetCountTerito : Integer;
    function GetCountPohar : Integer;
    function GetFirstKanna : TDolog;
    function GetFirstSajt : TDolog;
    function GetFirstTerito : TDolog;
    function GetFirstPohar : TDolog;
    procedure SetVanEllenor(b : Boolean);
    function GetVanEllenor : Boolean;
    procedure AddKellek (f,e : String; m,a : Integer);
    procedure DelKellek (i : Integer);

    function GetCountBorok : Integer;
    function GetItemBorok(i : Integer) : Pointer;
    function SearchBor (b : String) : TDolog;
    procedure AddBor (f,e : String; m,a : Integer);
    procedure DelBor (i : Integer);

    procedure SetItemVendegek (i : Integer; p : Pointer);
    procedure VendegekPack;
    function IsVendeg (v : TVendeg) : Boolean;
    function GetCountVendegek : Integer;
    function SearchVendeg (v : TVendeg) : Integer; overload;
    function SearchVendeg (s : String) : Integer; overload;
    function GetItemVendegek(i : Integer) : Pointer;
    procedure AddVendeg(v : TVendeg);
    procedure DelVendeg(i : Integer);
end;

type                                    // BORTHOZ
    TBorthoz = class (TTennivalo)
    private
    Borok : TList;
    public
```

```
   constructor Init (i : Integer); override;
   function GetCountBorok : Integer;
   function GetItemBorok (i : Integer) : Pointer;
   function SearchBor (b : String) : TDolog;
   procedure AddBor (f,e : String; m,a : Integer);
   procedure DelBor (i : Integer);
end;

type                                    // VASAROL
   TVasarol = class (TTennivalo)
   private
   Kellekek : TList;
   public
   constructor Init (i : Integer); override;
   function GetCountKellekek : Integer;
   function GetItemKellekek(i : Integer) : Pointer;
   function SearchKellek(k : String) : TDolog;
   procedure AddKellek (f,e : String; m,a : Integer);
   procedure DelKellek (i : Integer);
end;

type                                    // KISZOLGAL
   TKiszolgal = class (TTennivalo)
   private
   Vendeg : TVendeg;
   Borok : TList;
   public
   constructor Init (i: Integer); override;
   function GetVendeg : TVendeg;
   procedure SetVendeg (v : TVendeg);
   function GetCountBorok : Integer;
   function GetItemBorok (i : Integer) : Pointer;
   function SearchBor (b : String) : TDolog;
   procedure AddBor (f,e : String; m,a : Integer);
   procedure DelBor (i : Integer);
end;

type                                    // BORASZ
   TBorasz = class (TEmber)
   private
   Penz : Integer;
   Tennivalo : TList;
   Uzlet : TUzlet;
   Pince : TPince;
   Szakbolt : TSzakbolt;
   Ido : Integer;
   MitCsinal : Integer;
   public
   constructor Init (n : String); override;
   function GetPenz : Integer;
   procedure SetPenz (p : Integer);
   function GetUzlet : TUzlet;
   procedure SetUzlet (u : TUzlet);
   function GetPince : TPince;
   procedure SetPince (p : TPince);
   function GetSzakbolt : TSzakbolt;
   procedure SetSzakbolt (sz : TSzakbolt);
   procedure SetIdo (i : Integer);
   function GetIdo : Integer;
   procedure AddIdo (i : Integer);
   procedure DecIdo (i : Integer);
   procedure SetMitCsinal (i : Integer);
   function GetMitCsinal : Integer;
```

```
   function Kiir : String; override;        // csak a dinamikus kotes miatt

   function GetCountTennivalo : Integer;
   function GetItemTennivalo (i : Integer) : Pointer;
   function IsTennivalo (t : TTennivalo) : Boolean;
   function SearchTennivalo (t : TTennivalo) : Integer; overload;
   function SearchTennivalo (s : String) : Integer; overload;
   procedure AddTennivalo (t : TTennivalo);
   procedure DelTennivalo (i : Integer);
   procedure TennivaloPack;
   function GetCountVasarlas : Integer;
   function GetCountBorhozas : Integer;
   function GetCountKiszolgalas : Integer;
   function GetFirstBorhozas : TBorthoz;
   function GetFirstVasarlas : TVasarol;
   function GetFirstKiszolgalas : TKiszolgal;
   procedure ClearKiszolgal;

   function Vasarol : Integer;
   function Borthoz : Integer;
   procedure Nyit;
   procedure Zar;
   function Kiszolgal : Integer;
   procedure Leltaraz;
   function Elmegy (h : THely) : Integer; override;
end;

type                                    // ELLENOR
   TEllenor = class (TEmber)
   private
   Hangulat : String;
   Lepes : Integer;
   Mehet : Boolean;
   public
   constructor Init (n : String); override;
   function GetHangulat : String;
   procedure SetHangulat (h : String);
   function GetMehet : Boolean;
   procedure SetMehet(b : Boolean);
   procedure Buntet (b : TBorasz; o : Integer);
   procedure Jutalmaz (b : TBorasz; o : Integer);
   procedure Kiurittet (b : TBorasz);
   function Vizsgal (b : TBorasz) :  Integer;
   function Elmegy (h : THely) : Integer; override;
   function Kiir : String; override;              //csak a dinamikus kotes miatt
end;
```