

Eseményvezérelt alkalmazások

1. rész

Benzinkút

Gregorics Tibor

gt@inf.elte.hu

<http://people.inf.elte.hu/gt/oep>

Feladat

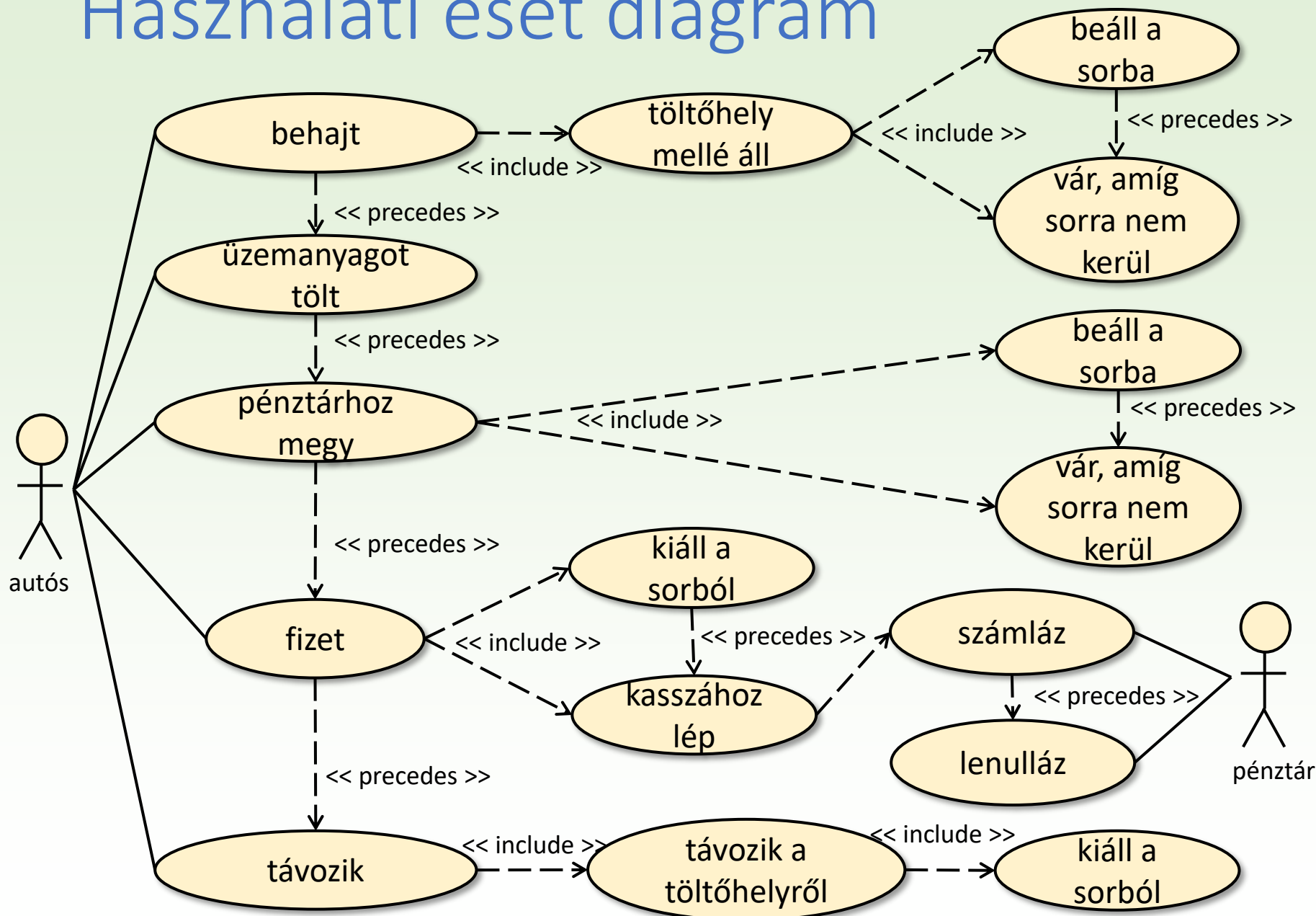
❑ Egy benzinkútnál több töltőhely és egy több kasszából álló pénztár működik.

- Az autósok behajtanak, és beállnak valamelyik töltőhelyhez tankolni.
- Miután sorra kerülnek, üzemanyagot vesznek fel.
- Ezután elmennek fizetni: beállnak a pénztárhoz várakozók sorába.
- Amint egy kassa szabad lesz, a soron következő autós fog fizetni (kilép a sorból, kiszámolják a fizetendő összeget az autós által használt töltőhely kijelzője és a benzinkút egységára alapján, lenullázzák a kijelzőt)
- Fizetés után az autós kihajt a töltőhelyről, és távozik.

❑ Szimuláljuk ezt a folyamatot tetszőleges számú, egymással párhuzamosan tevékenykedő autós esetére.



Használati eset diagram



Felhasználói esetek

eset		leírás
behajt	GIVEN	létezik a választott benzinkút, és annak töltőhelye, az autós nem tankol máshol
	WHEN	behajt egy létező töltőhelyhez
	THEN	besorol a töltőhely melletti sorba
üzemanyagot tölt	GIVEN	az autós a benzinkút egyik töltőhelyén elsőként áll
	WHEN	megadott liter üzemanyag töltése
	THEN	a kijelző mutatja a felvett üzemanyagot
pénztárhoz megy	GIVEN	létezik a pénztár
	WHEN	bemegy a pénztárba
	THEN	beáll a pénztár sorába
fizetés	GIVEN	létezik a pénztár, az autós ugyanazon benzinkút egyik töltőhelyén elsőként áll
	WHEN	sorra kerül (első a pénztár sorában és valamelyik kassa üres)
	THEN	kiáll a sorból, kiszámolják a fizetendő összeget, lenullázzák a töltőhely kijelzőjét
távozik	GIVEN	a benzinkút egyik töltőhelyén áll
	WHEN	távozik
	THEN	kiáll a töltőhely sorából

Felhasználói (hibás) esetek

eset		leírás	eset		leírás
behajt	GIVEN	nem létezik a benzinkút vagy nem létezik a kiválasztott töltőhely	nem tölt	GIVEN	az egyik töltőhelyen első
	WHEN	behajt		WHEN	nulla liter üzemanyag töltése
	THEN	hibajelzés		THEN	figyelmeztetés
üzemanyagot tölt	GIVEN	nem áll töltőhelyen	menekül	GIVEN	töltőhelyen első, a kijelző nem nulla
	WHEN	üzemanyagot tölt		WHEN	távozik
	THEN	hibajelzés		THEN	kiáll a töltőhely sorából, riasztás
pénztárhoz megy	GIVEN	nem létezik pénztár	elmegy	GIVEN	az egyik töltőhelyen áll, de nem elől
	WHEN	bemegy a pénztárba		WHEN	távozik a benzinkútról
	THEN	hibajelzés		THEN	kiáll a töltőhely sorából
fizet	GIVEN	nem áll töltőhelynél	átáll	GIVEN	az egyik töltőhelyen áll, de nem elől
	WHEN	fizet		WHEN	átáll egy másik töltőhelyhez
	THEN	hibajelzés		THEN	kiáll a töltőhely sorából besorol a töltőhely melletti sorba
távozik	GIVEN	nem áll töltőhelynél			
	WHEN	távozik			
	THEN	hibajelzés			

Elemzés

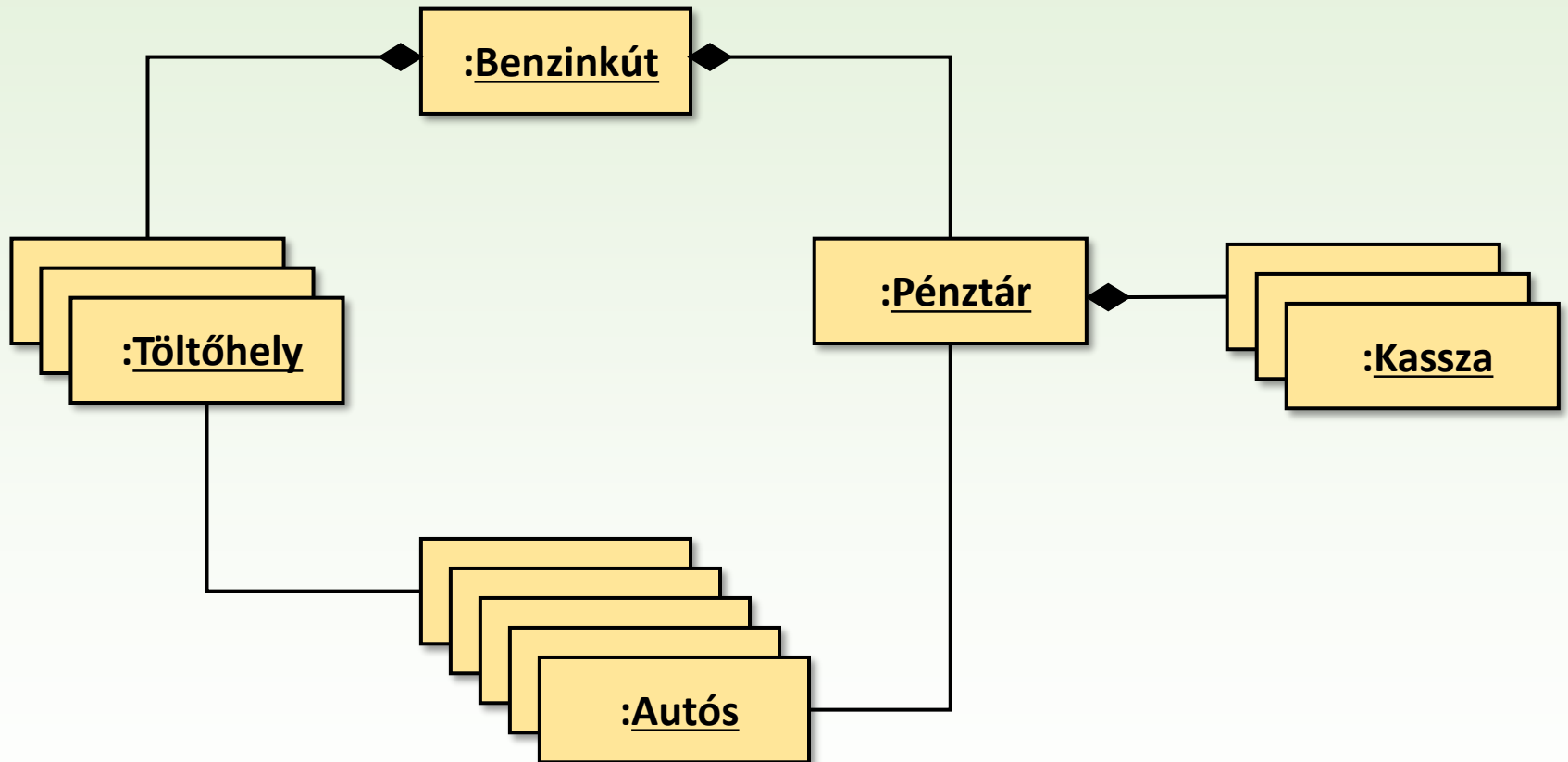
❑ Objektumok és tevékenységeik:

- **autósok** (tankolnak: behajt, tölt, pénztárhoz megy, fizet, távozik)
- **benzinkút** (kezeli a üzemanyag egységárát)
- **töltőhelyek** (amely mellé beáll az autós, ahol várakozik, majd üzemanyagot tölt, végül ahonnan fizetés után kiáll)
- **pénztár több kasszával** (ahol az autós sorba áll, fizet, ami lenullázza a töltőhely számlálóját)

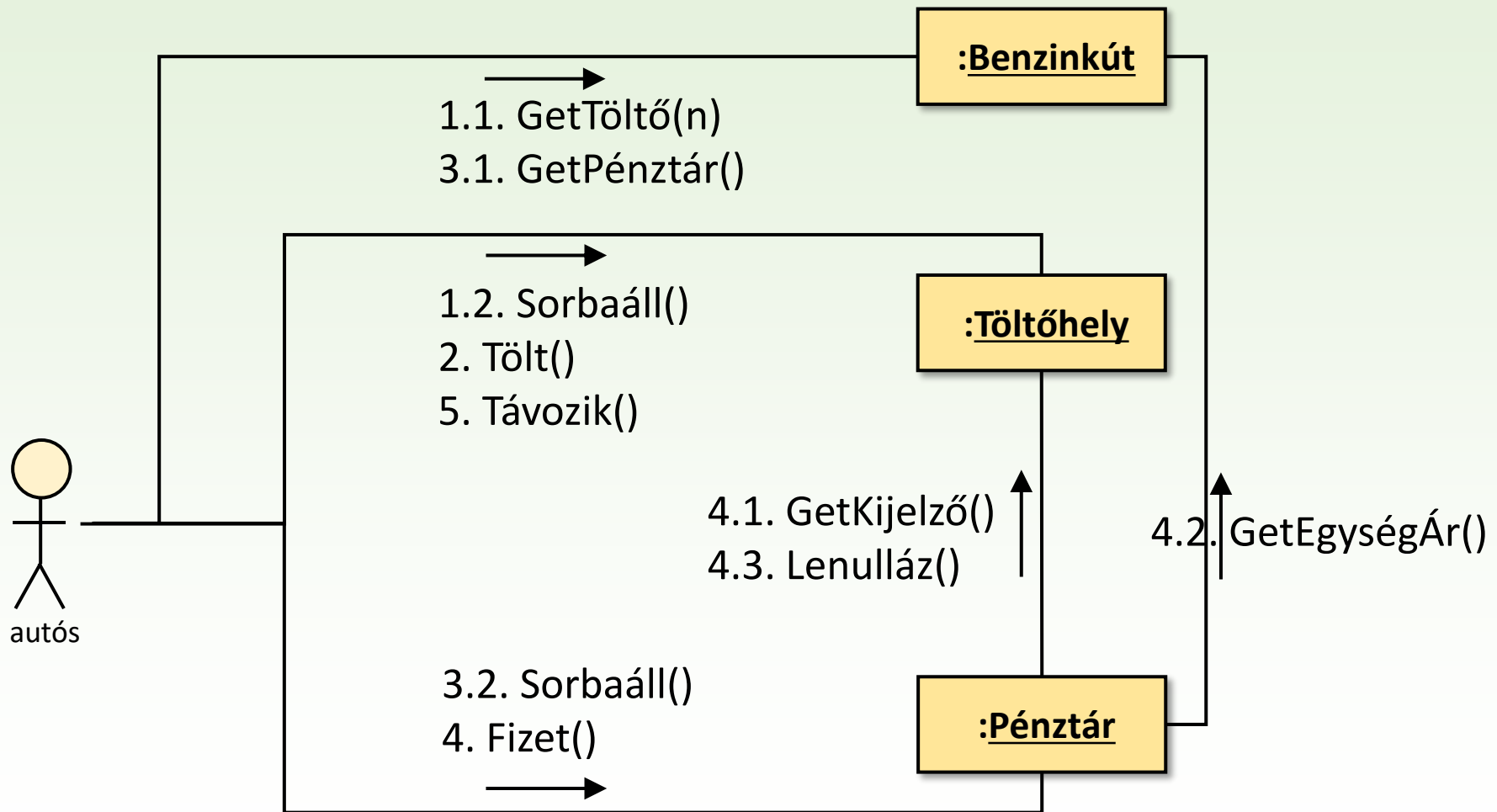
❑ Objektumok közötti kapcsolatok:

- a benzinkútnak **részei** a töltőhelyek és a pénztár
- egy autós egyszerre egy benzinkútnál tankol , amely során annak egy töltőhelyével és a pénztárával kerül közvetlen **kapcsolatba**, amely kétféle lehet: sorban áll (de nem elsőként) vagy tankol, illetve sorban áll vagy fizet

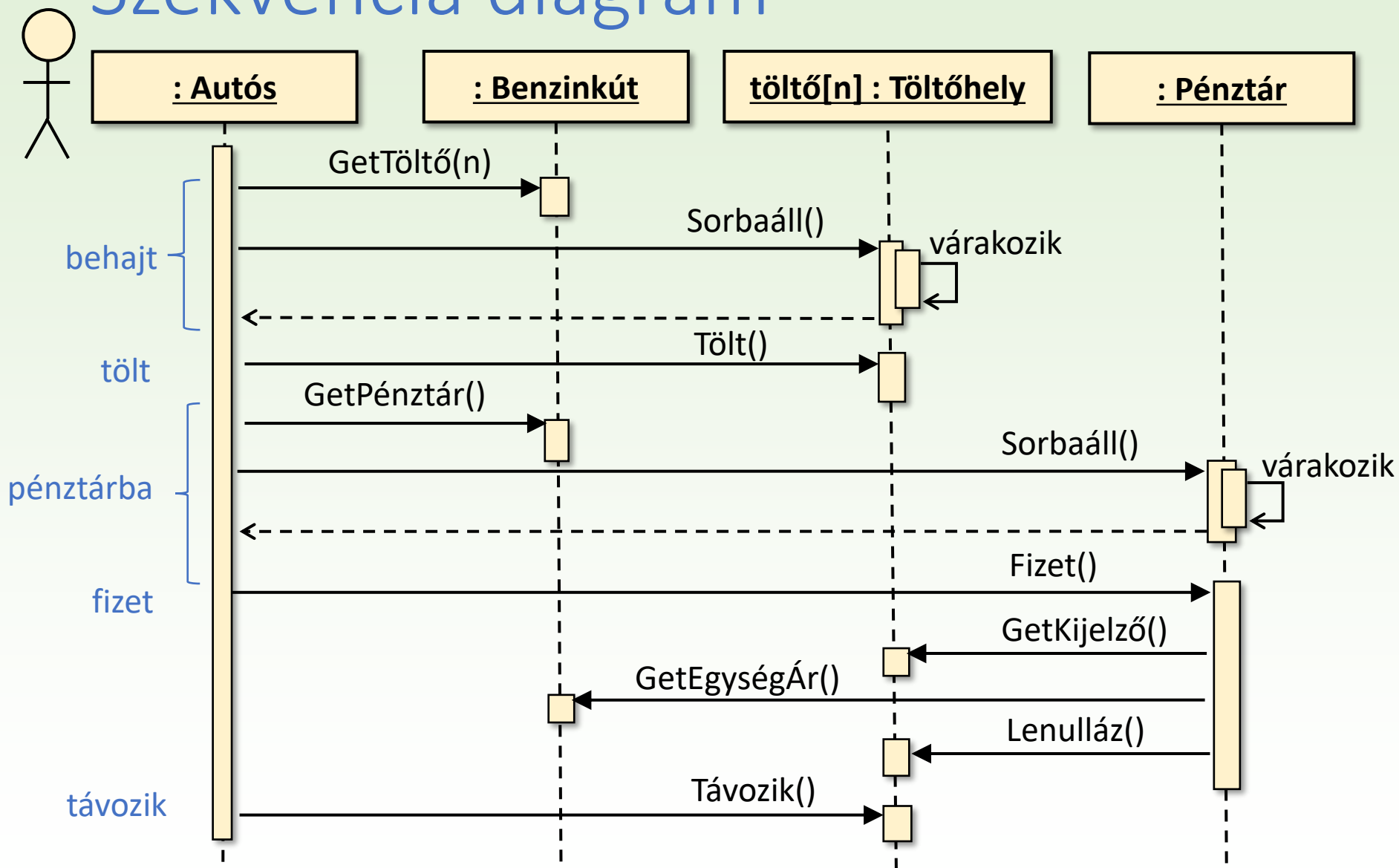
Objektum diagram



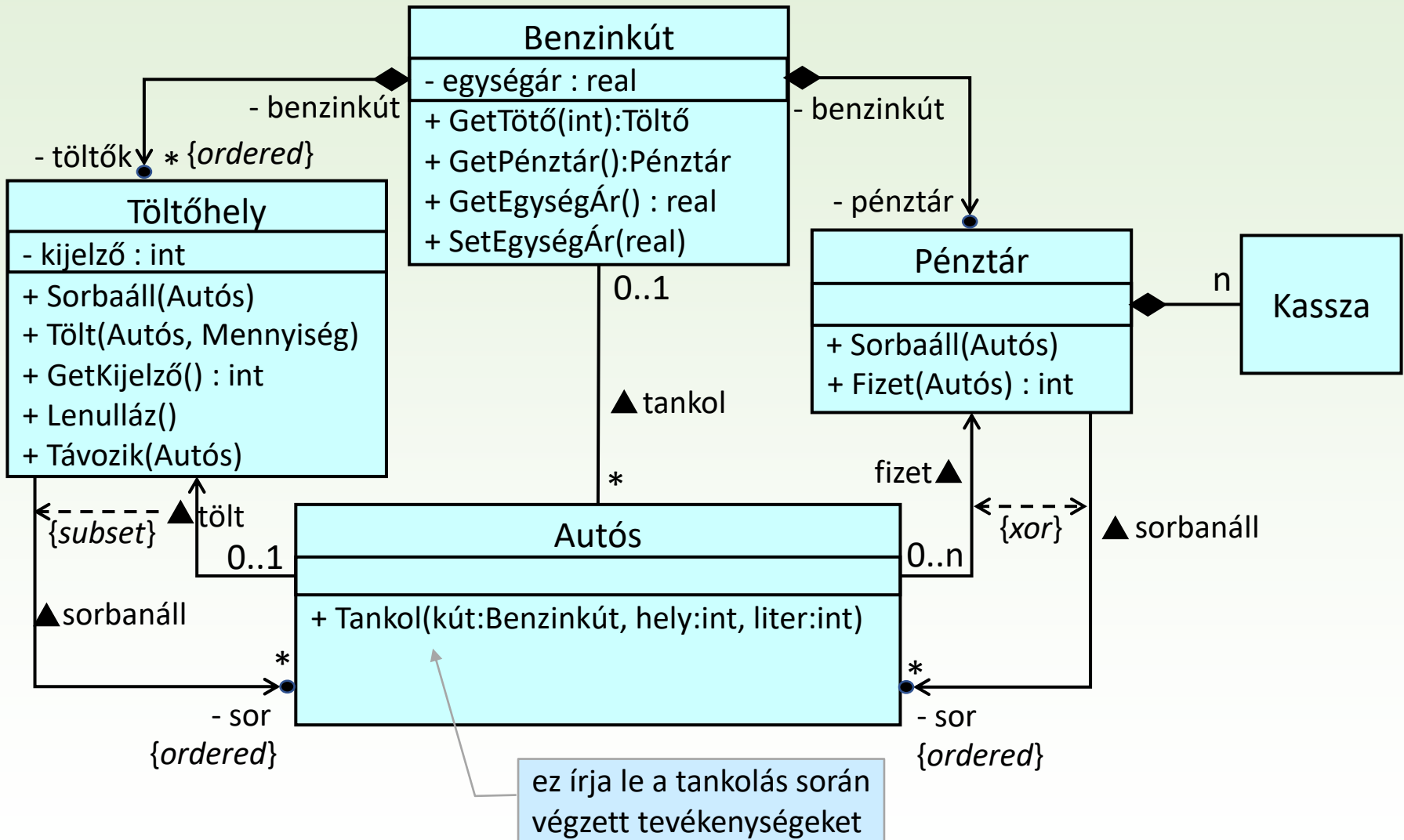
Kommunikációs diagram



Szekvencia diagram



Osztály diagram



Benzinkút osztálya

- A benzinkút getter-eket biztosít a rendszer komponenseinek (egy adott töltőhelynek és a pénztárnak) az elérhetőségéhez, valamint getter-t és setter-t az egységár adathoz.

Benzinkút
- töltők : Töltőhely[] - pénztár : Pénztár - egységár : real
+ Benzinkút(n : int, m : int) + Keres(autós : Autós) : Töltőhely <<getter>> + GetTöltő(int):Töltőhely + GetPénztár():Pénztár + GetEgységÁr() : real <<setter>> + SetEgységÁr(real)

Benzinkút osztálya (PetrolStation.cs)

```
class PetrolStation
{
    private readonly List<Pump> Pumps = new ();
    public Cash CashDesk { get; }
    public double Unit { get; set; }

    public PetrolStation(int n, int m)
    {
        for (int i = 0; i < n; ++i) Pumps.Add(new Pump(this));
        CashDesk = new Cash(this, m);
    }

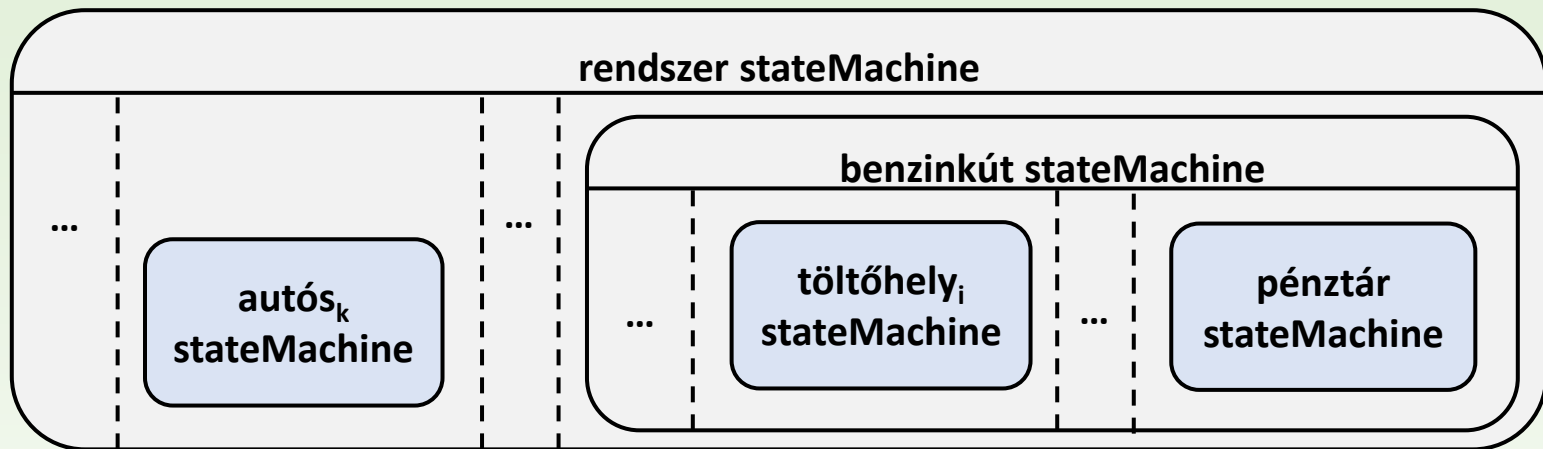
    public Pump GetPump(int number) { return Pumps[number]; }

    public int PumpsCount { get => Pumps.Count; }

    public Pump Search(Car car) { return pumps.Find(p => p.IsFirst(car)); }
}
```

megkeresi azt a töltőhelyet,
ahol az autós tankolt

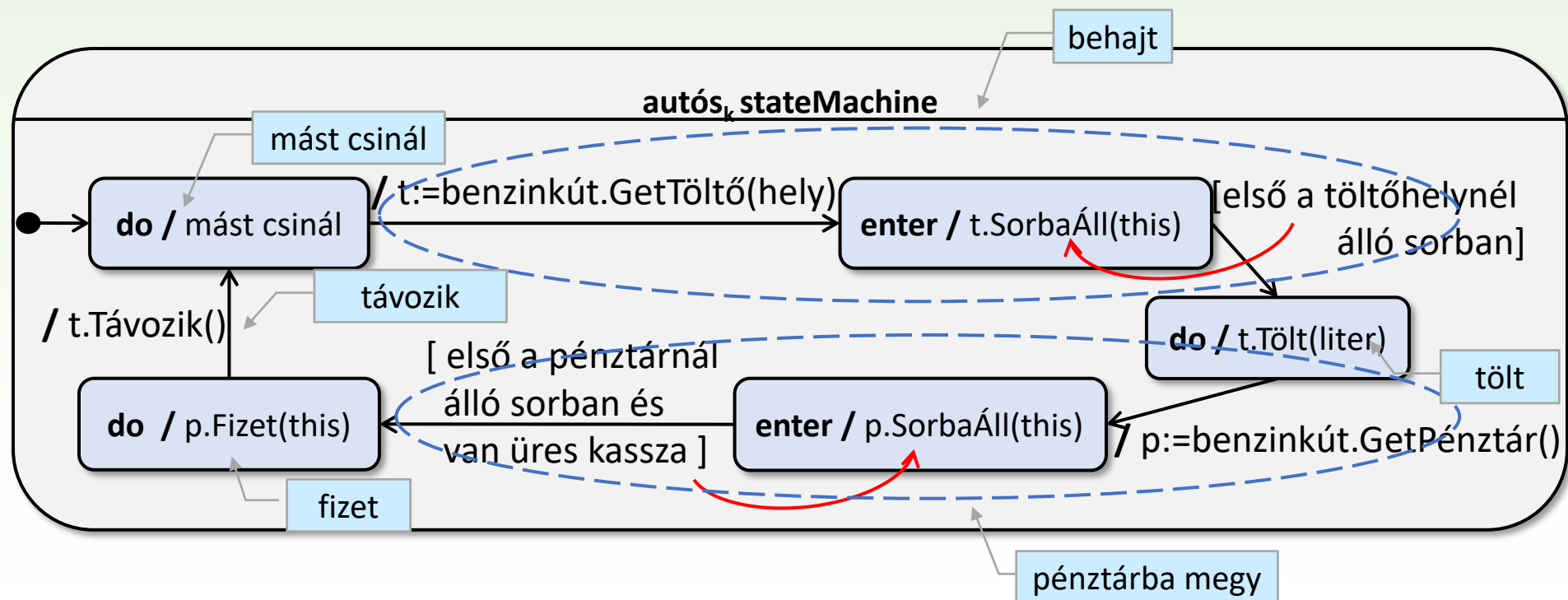
Rendszer állapotgépe



- ❑ A rendszer állapotát az autósok és a benzinkút állapota együtt határozza meg; a benzinkút állapota a töltőhelyek és a pénztár állapotaitól függ.
- ❑ Az autósok állapotgépei egymással párhuzamosan külön szálakon, „önerőből” működnek, nem a külvilág által kapott üzenetek hatására.
- ❑ A töltőhelyek és a pénztár állapotgépei az autósok által küldött szinkron üzenetek hatására működnek. Nem igényelnek külön szálakat.

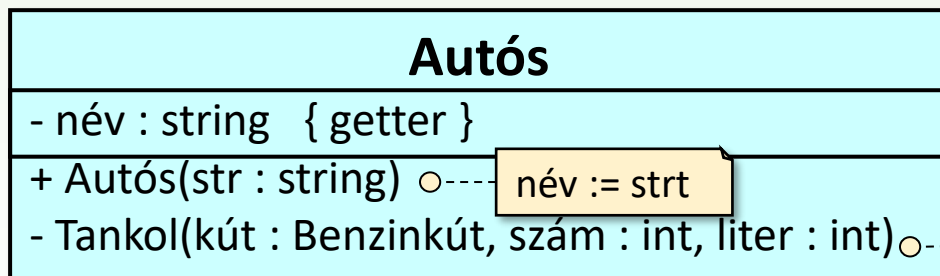
Autós objektum állapotgépe

- Az állapotok hosszabb ideig tartó tevékenységeket jelölnek.
- Az átmeneteket (ezekhez legfeljebb csak rövid idejű tevékenység tartozik) nem üzenetek váltják ki, hanem vagy az átmenet kiinduló állapota belső tevékenységének vége, vagy az átmenet őrfeltételének teljesülése.



Autós osztály

- ❑ A Tankol() metódus írja le tankolás folyamatát, amelyhez a benzinkút komponensei (töltőhely, pénztár) biztosítanak metódusokat.
- ❑ A tankolás folyamatát minden autós esetében külön szálon kell majd elindítani, hogy az autósok egymással párhuzamosan tankolhassanak.
- ❑ Ha a „mást csinál” tevékenységet is implementálnánk, akkor a konstruktornak kellene elindítani az autónak azt az életciklusát, amely felváltva hívná a mást csinál és tankol tevékenységeket.



loop
„mást csinál”
Tankol()
endloop

töltőhely := kút.GetTöltő(szám)
töltőhely.Sorbaáll(**this**)
töltőhely.Várákozik(**this**)
töltőhely.Tölt(liter)
pénztár := kút.GetPénztár()
pénztár.Sorbaáll(**this**)
pénztár.Várákozik(**this**)
pénztára.Fizet(**this**)
hely.Távozik(**this**)

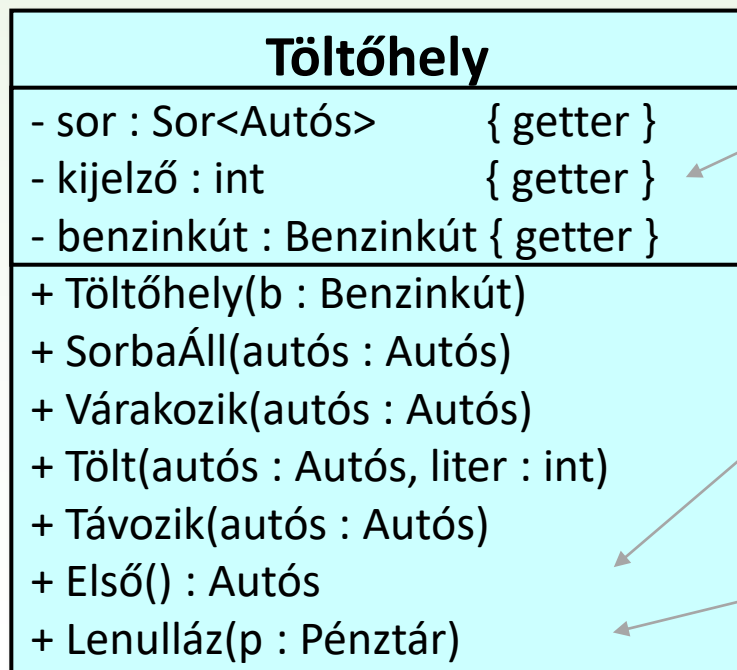
Autós osztály (Car.cs)

```
class Car
{
    public string Name { get; private set; }
    public Car(string str) { name = str; }
    private PetrolStation station;
    private int number;
    private int liter;
    private Thread fuelThread;
    public void Refuel(PetrolStation station, int number, int liter)
    {
        this.station = station; this.number = number; this.liter = liter;
        fuelThread = new Thread(new ThreadStart( Activity ));
        fuelThread.Start();
    }
    public class NoRefuelingException : Exception { }
    private void Activity()
    {
        if (null==station || null==station.CashDesk || number < 0 ||
            number >=station.PumpsCount ) throw new NoRefuelingException();
        station.GetPump(number).JoinQueue(this); // joins the n-th pump
        station.GetPump(number).Fill(this, liter); // refuels petrol
        station.CashDesk.JoinQueue(this); // goes to cashdesk
        int sum = station.CashDesk.Pay(this); // pays
        station.GetPump(number).Leave(this); // leaves the petrolstation
    }
}
```

külön szálon indul az autós tankolási tevékenysége

Töltőhely osztály

- ❑ A töltőhely nyilvántartja a tankolásra várakozó autók **sorát**, amelynek elején áll az éppen tankoló, illetve fizető autós. Fontos adattag a töltőhely **kijelzője**, és a töltőhelyet üzemeltető **benzinkút** is.
- ❑ A Sorbaáll(), Várakozik(), Tölt(), Távozik() metódusokat az autós Tankol() metódusa hívja, a Lenulláz() metódust a pénztár Fizet() metódusa használja, az Első() metódus a sor-első ellenőrzésére szolgál.



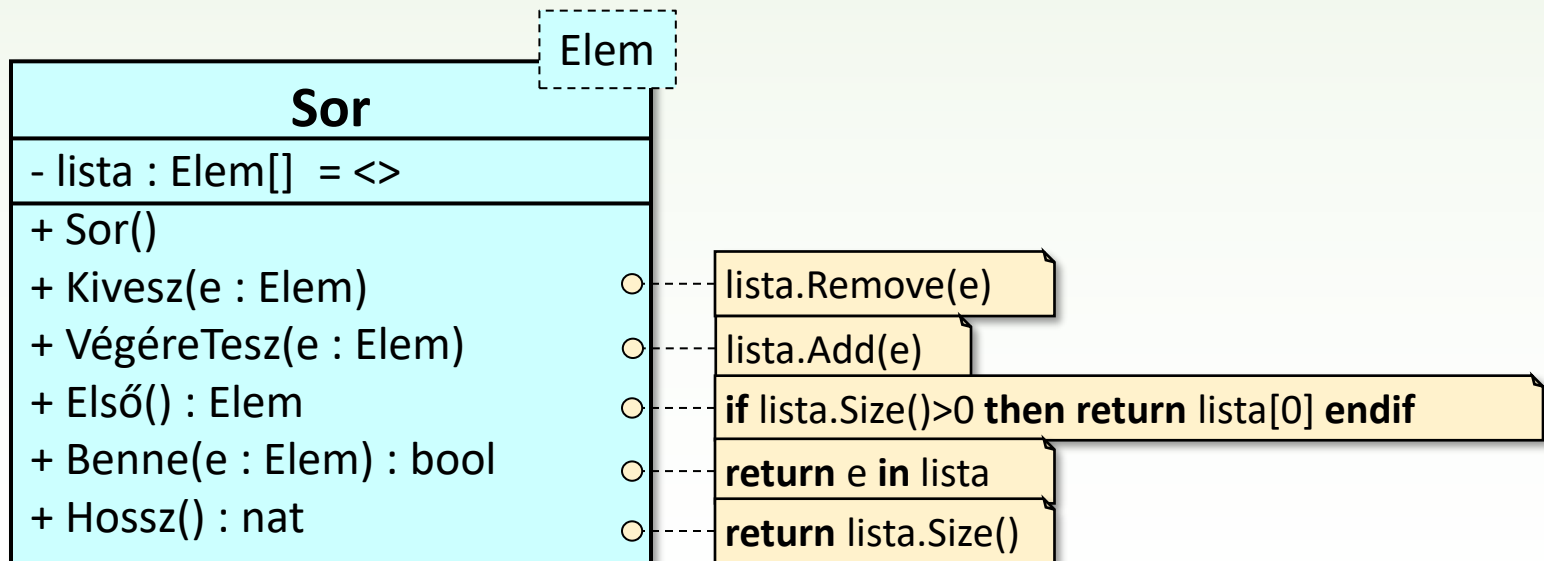
Látni kell a töltőhelyen álló sort, a kijelző értékét, és az üzemeltető benzinkutat.

Sokszor kell lekérdezni, hogy egy autós első-e a töltőhelynél álló sorban.

A töltőhely lenullázását csak annak a pénztárnak engedjük meg, ahol kifizették a felvett üzemanyagot.

Speciális sor típus

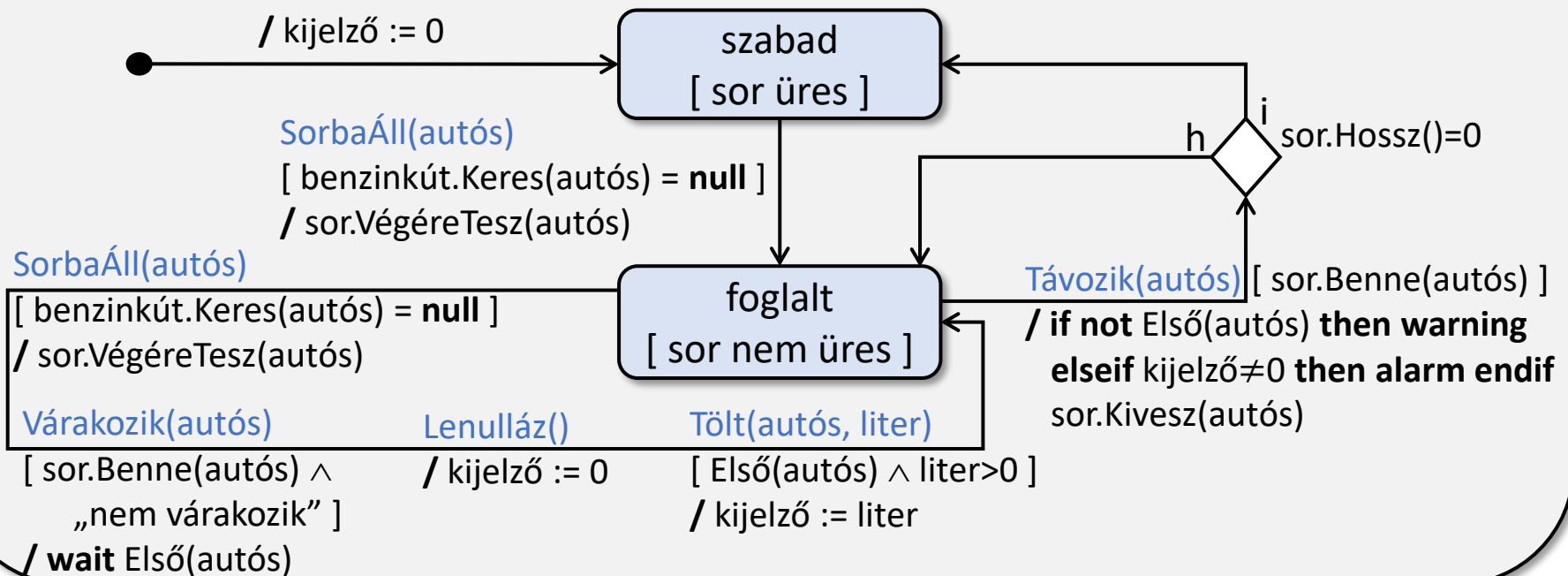
- ❑ Akár egy töltőhelynél, akár a pénztárban álló autók sorának típusa nem a hagyományos sor típus, mert a szokásos „sor végére új elemet tesz” művelet mellett megengedjük, hogy a sor bármelyik elemét ki lehessen venni a sorból, ne csak a sor legelső elemét.
- ❑ Meg lehet továbbá nézni, hogy egy adott elem a sorban van-e, első-e a sorban, illetve hány elemű a sor.



Töltőhely objektum állapotgépe

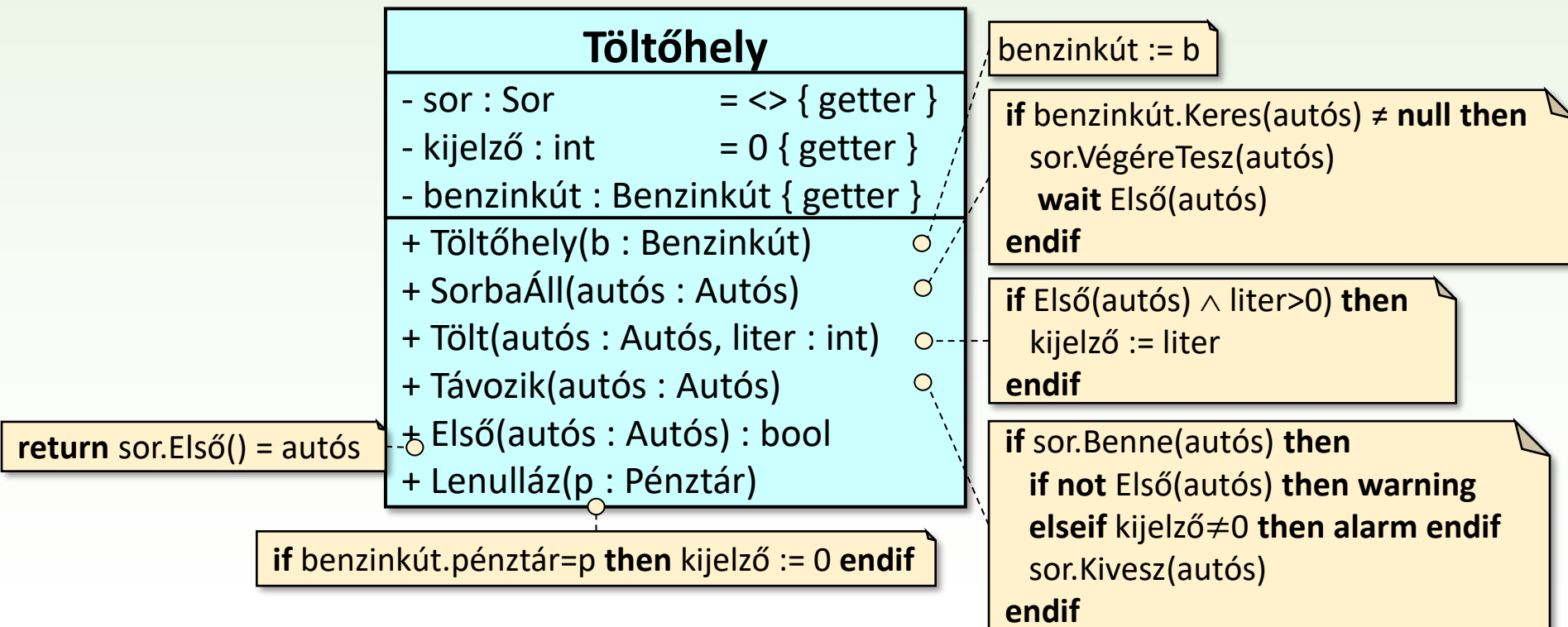
- ❑ Egy töltőhelynek két állapotát különböztetjük meg: **szabad** (senki nem áll ott sorban), **foglalt** (egy vagy több autó áll sorban).
- ❑ A Sorbaáll() és Távozik() metódusok változtatják a **sort**, tehát állapotot válthatnak; a többi metódus nem ilyen.

töltőhely_i stateMachine



Töltőhely osztály metódusai

- A Töltőhely metódusait az autók saját szálain futó Tankol() metódus törzséből hívjuk meg, így azok egymással párhuzamosan is futhatnak (akár ugyanaz a metódus is eltérő autóssal). Mivel közös erőforrásokat (sor, kijelző) használnak, ezért kölcsönösen kizárásos módon kell ezeket futtatni: egyszerre egyidőben legfeljebb csak egyet.



Töltőhely osztály (Pump.cs)

```
class Pump
{
    private int quantity;

    public int Quantity()
    {
        int value;
        Monitor.Enter(this);
        value = quantity;
        Monitor.Exit(this);
        return value;
    }
}
```

```
class Queue <Item> where Item : class
{
    private readonly List<Item> list = new();
    public Queue () { }
    public void Exit(Item item) { list.Remove(item); }
    public void Enter(Item item) { list.Add(item); }
    public Item First()
    {
        if (list.Count > 0) return list[0]; else return null;
    }
    public bool In(Item item) { return list.Contains(item); }
    public int Size() { return list.Count; }
}
```

```
private Queue<Car> queue = new ();
```

```
public Pump() { ResetQuantity(); }
```

```
public void ResetQuantity()
{
    Monitor.Enter(this);
    Quantity = 0;
    Monitor.Exit(this);
}
```

```
...
```

az adott töltőhelyhez (this) tartozó kritikus szakasz eleje

az adott töltőhelyhez (this) tartozó kritikus szakasz vége

Töltőhely osztály (Pump.cs)

```
public void JoinQueue(Car car)
{
    Monitor.Enter(this);
    if (!queue.In(car))
    {
        queue.Enter(car);
        while ( !IsFirst(car) ) Monitor.Wait(this);
    }
    Monitor.Exit(this);
}
```

```
public bool IsFirst(Car car)
{
    Monitor.Enter(this);
    bool l = queue.First() == car;
    Monitor.Exit(this);
    return l;
}
```

az autós várakozik (az őt futtató szál felfüggesztődik), amíg a töltőhelynél álló sor elejére nem kerül

```
public void Leave(Car car)
{
    Monitor.Enter(this);
    if (queue.In(car));
    {
        if (!IsFirst(car)) { /* warning */
        else if (quantity > 0) { /* alarm */
            queue.Exit(car);
            Monitor.PulseAll(this);
        }
        Monitor.Exit(this);
    }
}
```

```
public void Fill(Car car, int liter)
{
    Monitor.Enter(this);
    if ( IsFirst(car) && liter > 0)
        quantity = Math.Max(liter,0);
    Thread.Sleep(liter*200);
    // elapsed time of fueling
    Monitor.Exit(this);
}
```

elindítja az összes felfüggesztett szálát:
az összes várakozó autós tevékenységét

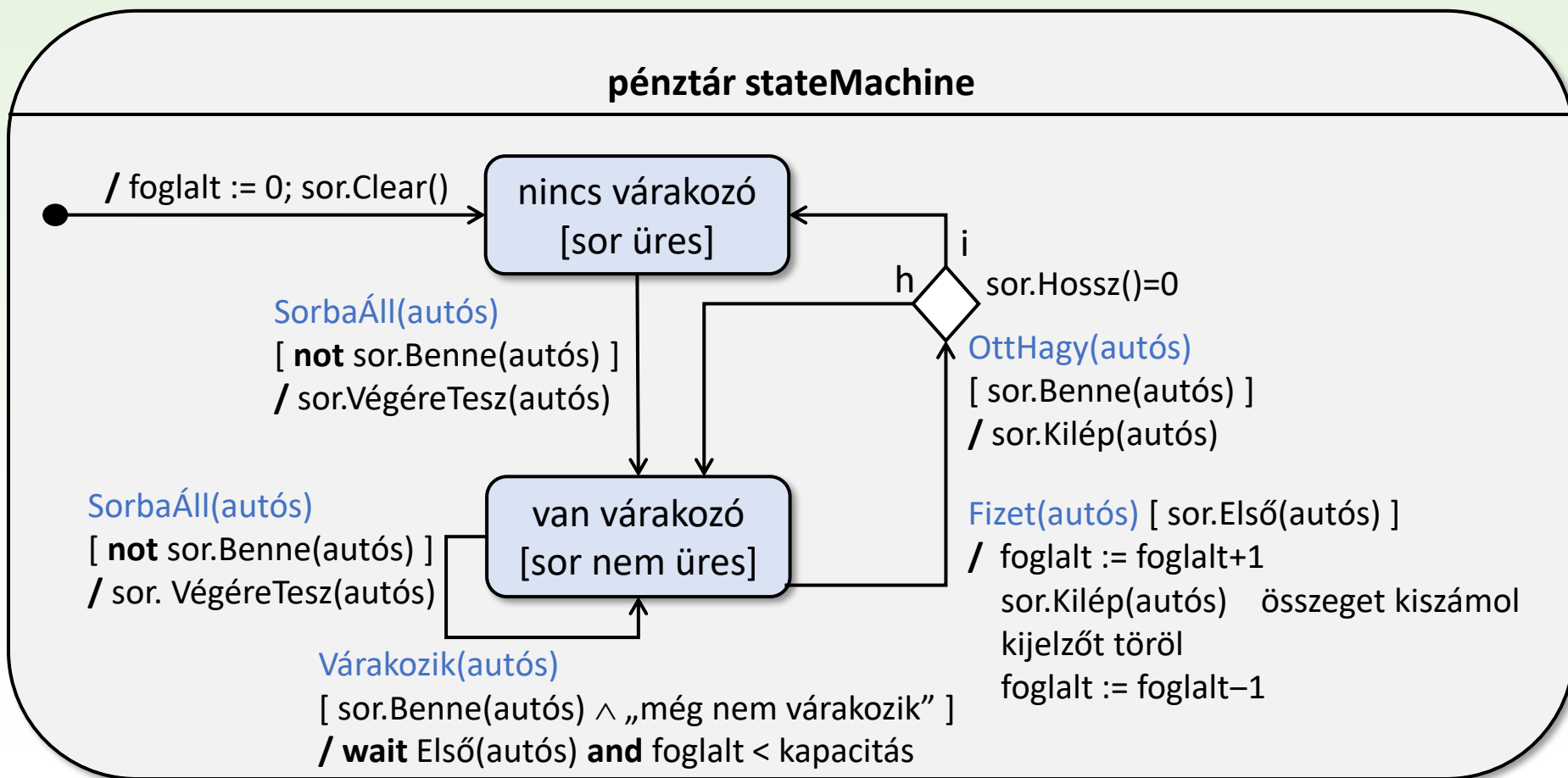
Pénztár osztály

- ❑ A pénztár állapotát a fizetni akaró autósok **sora**, az összes kassa száma (**kapacitás**), és a **foglalt** kasszák (az éppen fizető autósok) száma határozza meg.
- ❑ A **SorbaÁll()** és **Fizet()** metódusokat az autós **Tankol()** metódusa hívja meg. A **Fizet()** metódus kapcsolatban áll adott autós töltőhelyével: annak kijelzője alapján számláz, majd lenullázza a kijelzőt.

Pénztár
- kapacitás : int - foglalt : int - sor : Sor<Autós> - benzinkút : Benzinkút
+ Pénztár(b:Benzinkút, m:int) + SorbaÁll(autós : Autós) + Várakozik(autós : Autós) + Fizet(autós : Autós) : int + OttHagy(autós : Autós)

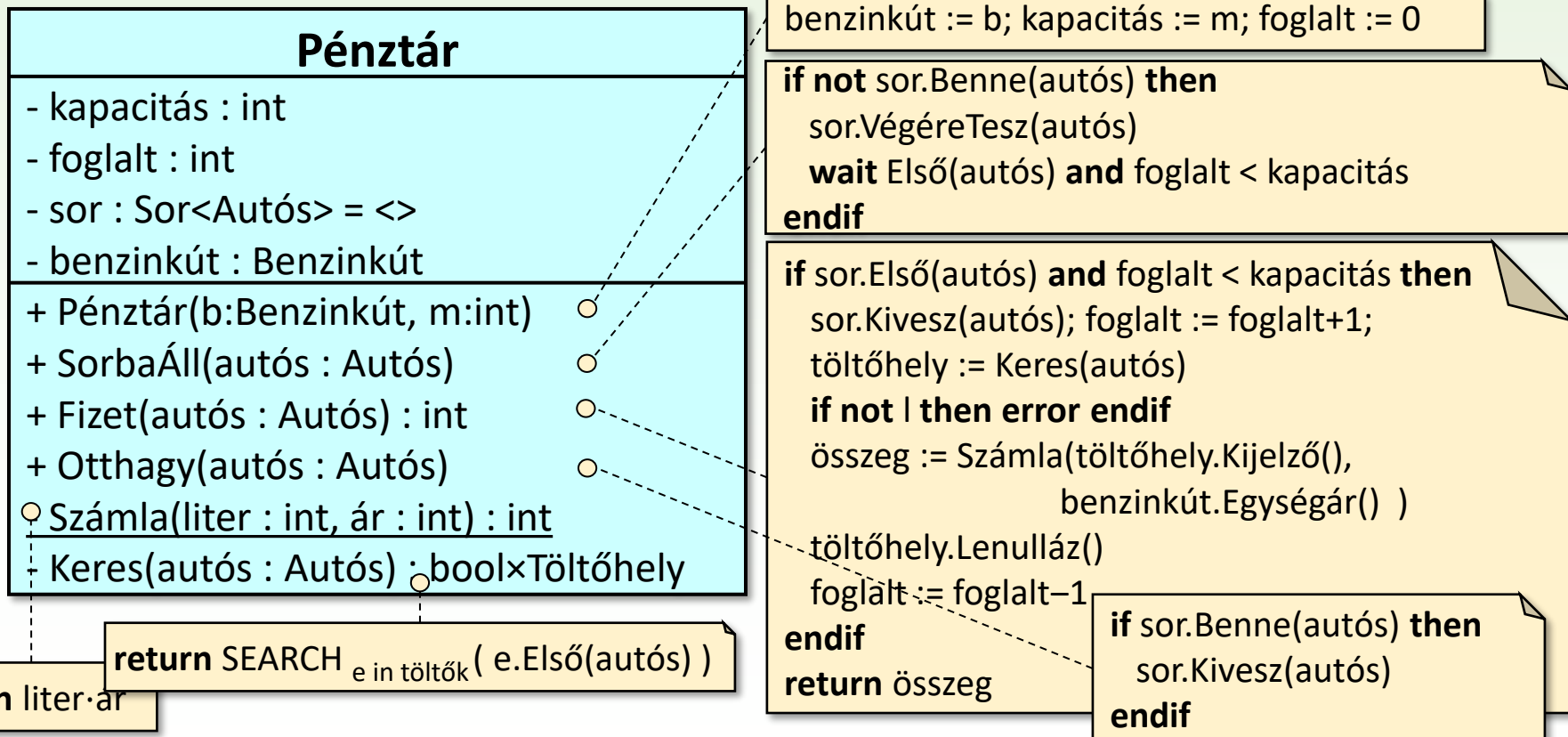
Pénztár objektum állapotgépe

- A pénztár lehet **szabad** vagy **várakozókkal** teli. Az állapot-átmeneteket a pénztár szinkron hívott metódusai (Sorbaáll(), Fizet()) valósítják meg.



Pénztár osztály metódusai

- ❑ A Pénztár osztály metódusait is az autósok hívják párhuzamosan, így itt is kölcsönös kizárással kell a kritikus szakaszokat futtatni.
- ❑ Ilyen kritikus szakasz a SorbaÁll() és Első() metódusok teljes törzse, és a Fizet() metódusban a foglalt adattagra irányuló két értékadás



Pénztár osztály (Cash.cs)

```
class Cash
{
    private readonly int capacity;
    private int engaged = 0;
    private readonly Queue<Car> queue = new ();
    private readonly PetrolStation station;

    public Cash(PetrolStation station, int capacity)
    {
        this.station = station; this.capacity = capacity;
    }

    private static double Invoice(int liter, double price) { return liter*price; }

    ...
}
```

ha az autós nem a legelső a pénztár sorában, vagy nincs szabad kassza, akkor várakozik

```
public void JoinQueue(Car car)
{
    Monitor.Enter(this);
    if (!queue.In(car))
    {
        queue.Enter(car); // joins the queue
        while (!(queue.First(car) && engaged < capacity))
            Monitor.Wait(this);
    }
    Monitor.Exit(this);
}
```

az adott töltőhelyhez (this) tartozó kritikus szakasz eleje

az adott töltőhelyhez (this) tartozó kritikus szakasz vége

Pénztár osztály (Cash.cs)

```
public int Pay(Car car)
{
    Monitor.Enter(this);
    bool l = queue.First(car) && engaged < capacity;
    Monitor.Exit(this);
```

```
    int sum = 0;
    if (l)
    {
        Monitor.Enter(this);
        queue.Exit(); // leaves the queue
        ++engaged;    // steps to a cash desk
        Monitor.Exit(this);
```

```
        Pump pump = station.Search(car);
```

keresi az autós töltőhelyét

```
        if (pump != null)
        {
```

kiszámolja a fizetendő összeget

```
            sum = (int)Invoice(pump.Quantity(), station.Unit);
            pump.ResetQuantity(); // resets the pump
            Thread.Sleep(10000);  // elapsed time of paying
        }
```

a pénztárhoz (this) tartozó kritikus szakasz eleje

```
        Monitor.Enter(this);
        --engaged; // leaves the cash desk
```

```
        Monitor.PulseAll(this);
```

elindítja az összes várakozó autós szálát

```
        Monitor.Exit(this);
```

a pénztárhoz (this) tartozó kritikus szakasz vége

```
    }
    return sum;
}
```

```
public void Leave(Car car)
{
    Monitor.Enter(this);
    if (queue.In(car)) queue.Exit(car);
    Monitor.Exit(this);
}
```

Eseményvezérlés

2. rész

Stopper óra

Gregorics Tibor

gt@inf.elte.hu

<http://people.inf.elte.hu/gt/oep>

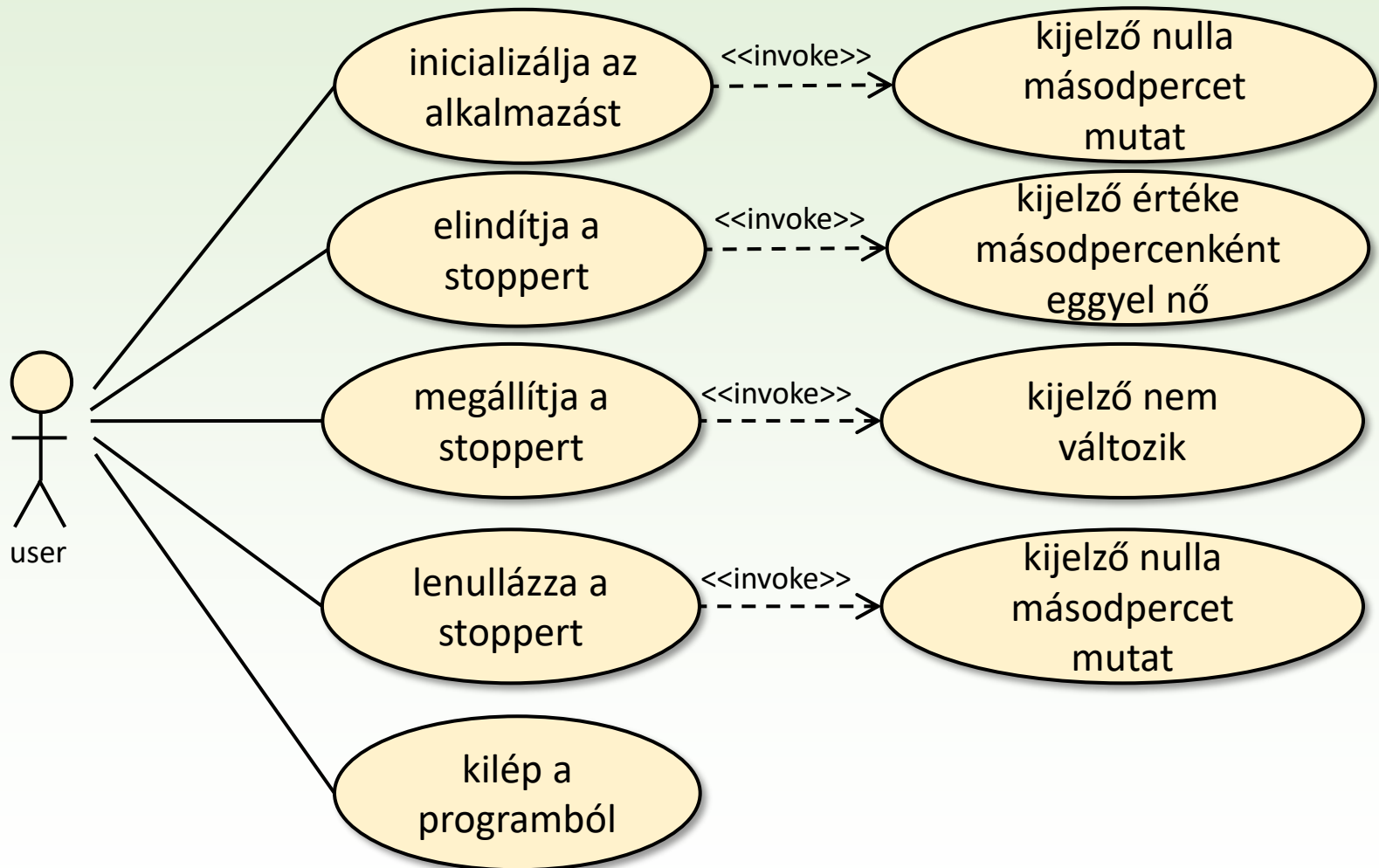
Feladat: Stopper



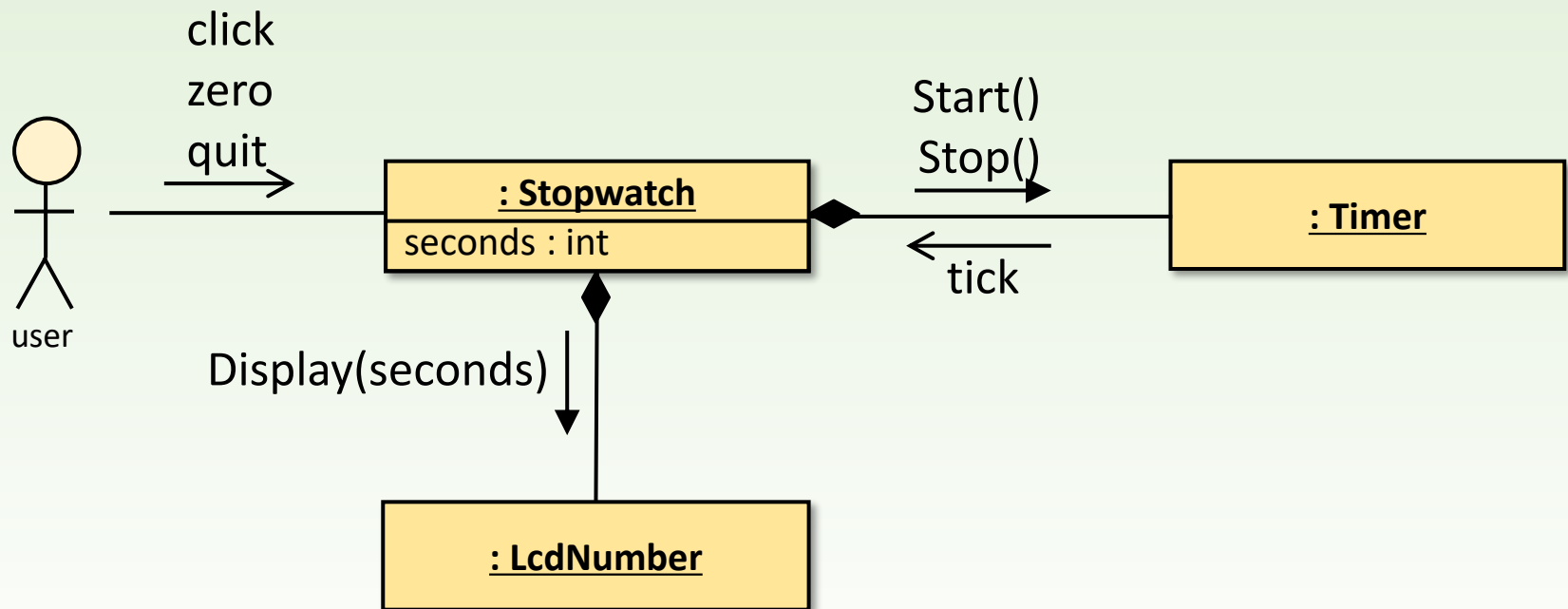
Készítsünk egy stoppert, amely másodpercenként méri a múltó időt.

- A mérés egy jelzés hatására induljon el, majd egy ugyanilyen jelzés hatására álljon le; majd újabb jelzés hatására folytatódjon, és így tovább.
- Legyen lehetőség a kijelzett idő lenullázására, amely a mérést is megállítja.
- Külön jelzés hatására az alkalmazás álljon le.

Használati eset diagram

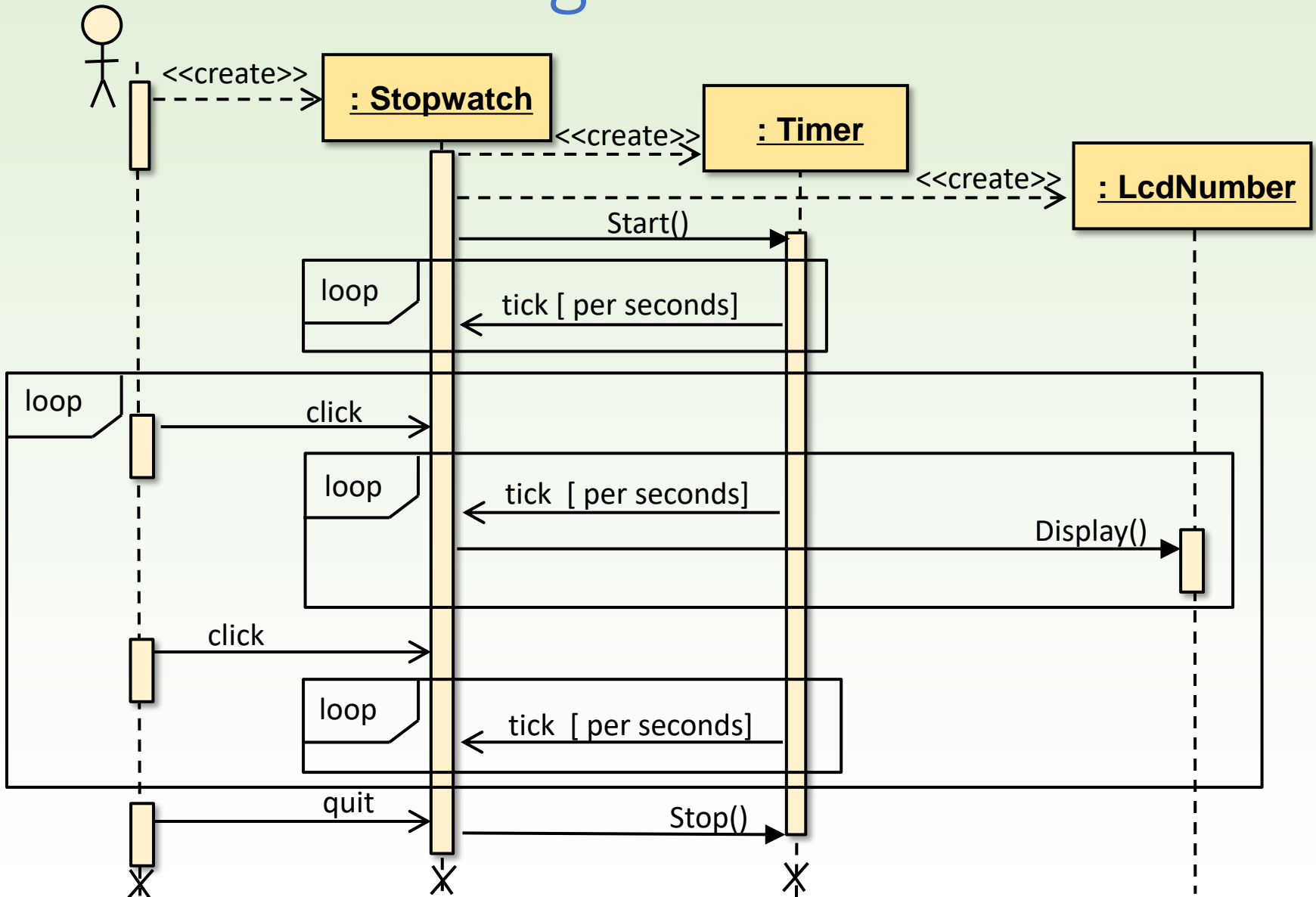


Kommunikációs és objektum diagram

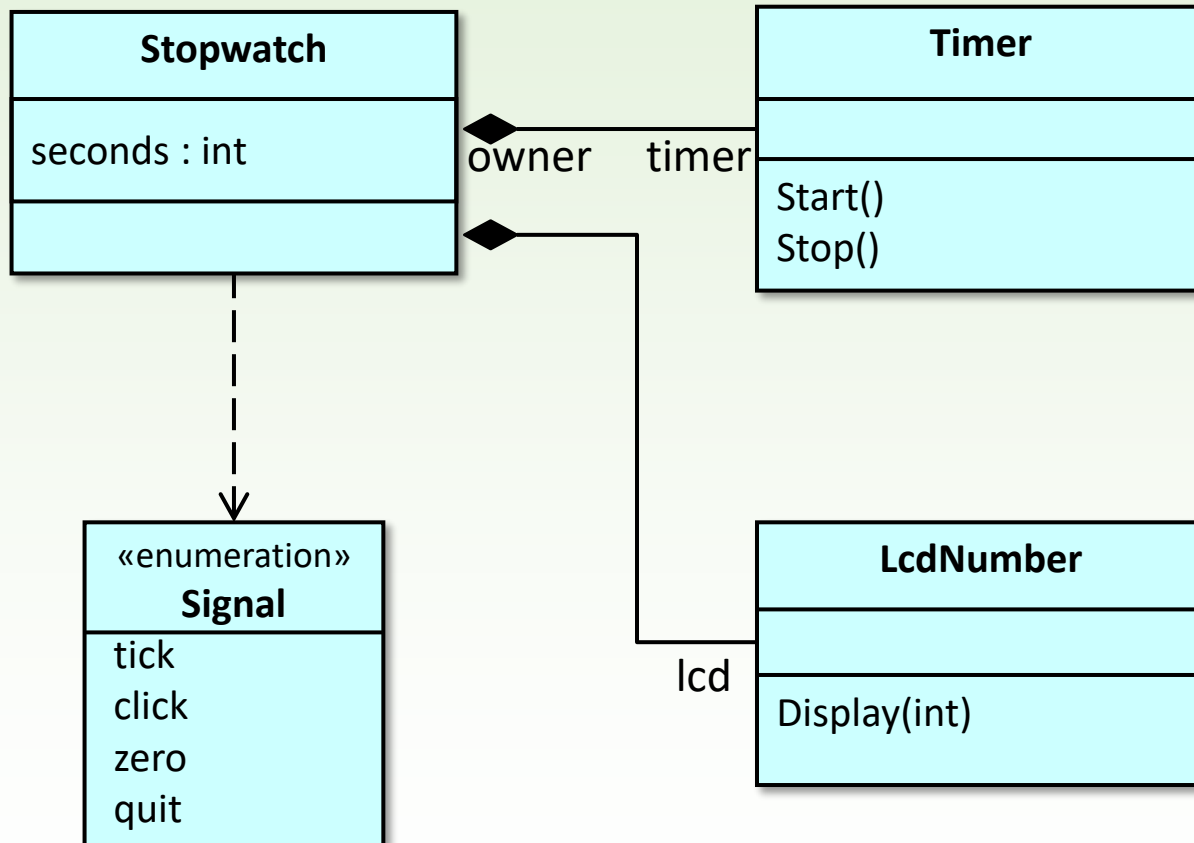


- a felhasználó és az időzítő szignálokat küld a stoppernek
- a stopper a neki küldött szignálokat aszinkron módon dolgozza fel.

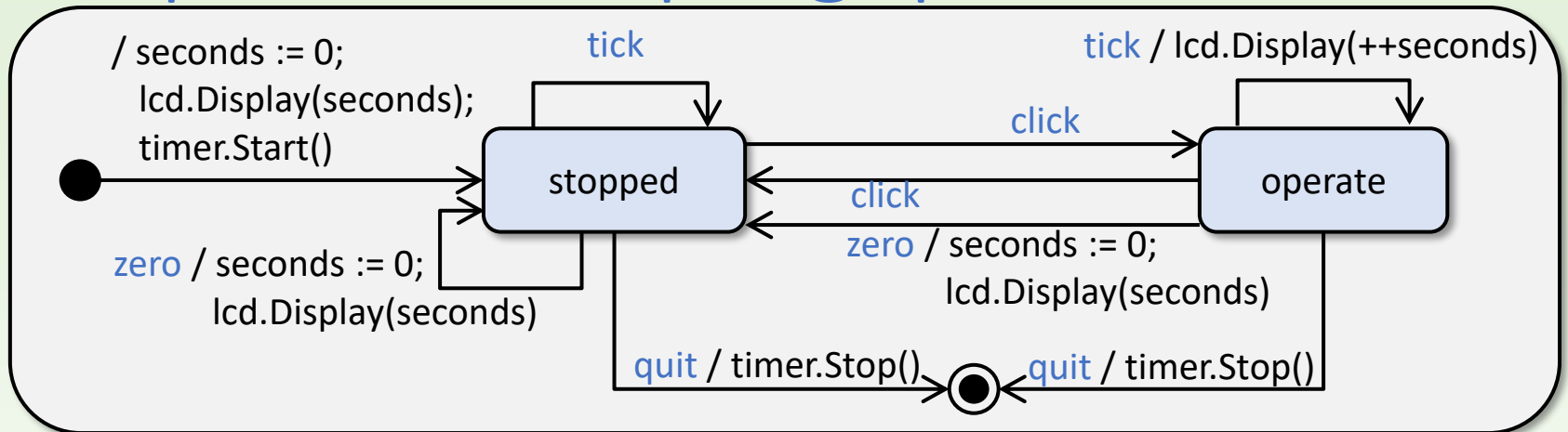
Szekvencia diagram



Osztálydiagram

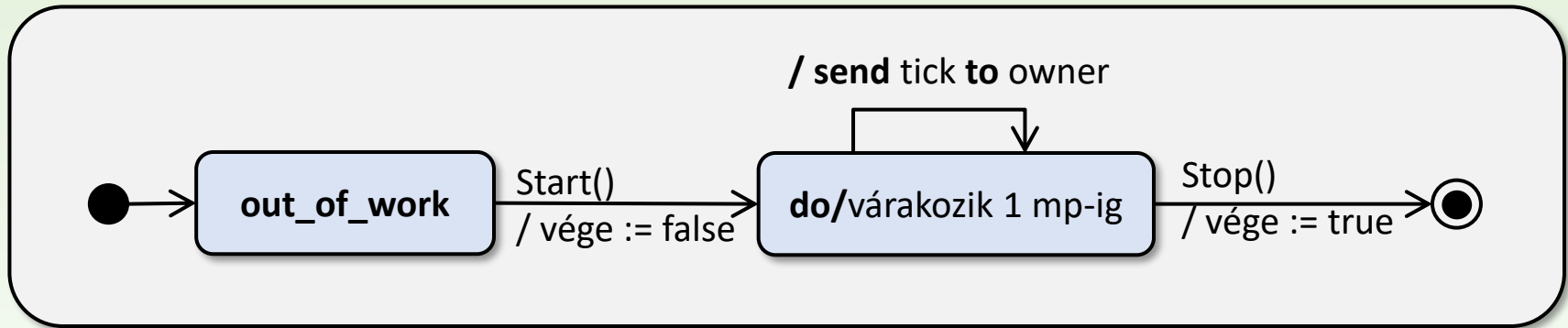


Stopwatch állapotgépe



állapot esemény	stopped	operate
click	operate	stopped
tick	stopped	/ lcd.Display(++seconds) operate
zero	/ second:=0; lcd.Display(seconds) stopped	/ second:=0; lcd.Display(seconds) stopped
quit	/ timer.Stop() final	/ timer.Stop() final

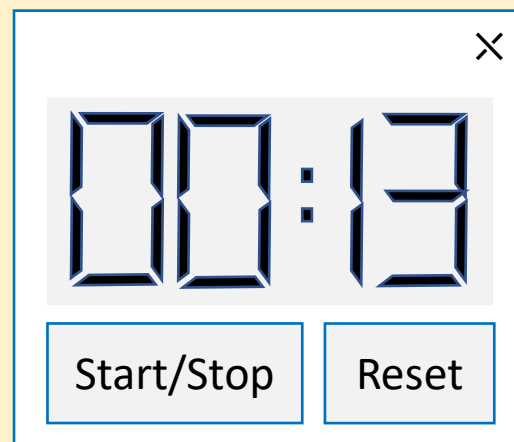
Timer állapotgépe



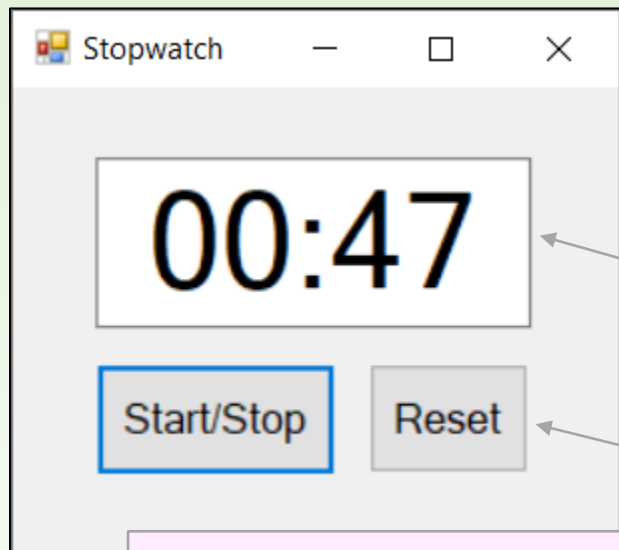
```
vége := false
while not vége loop
    wait 1000
    send tick to owner
endloop
```

Megvalósítás

- ❑ Eddig konzolos alkalmazásokat készítettünk, de szebb lenne grafikus megjelenést használni. Jó lenne, ha a felületet vizuális lehetne megtervezni, és a megjelenítés kódja automatikusan generálóna.
- ❑ Gyorsabb és egyszerűbb lenne a fejlesztés, ha a rendszeresen alkalmazott kódelemek (mint a száakezelést, a szálbiztos eseménysor kezelése, őrfeltételeket megvalósító várakozó utasítások) „automatikusan” kerülhetnének bele a kódba.
- ❑ Olyan fejlesztő környezetet keresünk, amely
 - hozzáad az alkalmazáshoz egy aszinkron eseménykezelő mechanizmust
 - lehetőséget ad egyedi tulajdonságokkal kiegészíthető standardizált grafikus megjelenésű objektumok (nyomógomb, ablak, stb.) létrehozására
 - vizuális tervezőt biztosít a grafikus megjelenítés kialakításához
 - speciális objektumok (pl. időzítő) használatát teszi lehetővé



Stopwatch .net alatt fejlesztve



Stopwatch : Form egy ablakszerű vezérlő objektum osztálya, amely objektum más vezérlőket (időzítő, kijelző, nyomógomb) is tartalmazhat. Az ablak bezárása kiváltja a quit szignált.

TextBox osztály példánya a kijelző, amelynek csak a display metódusát kell megírni.

Button típusú objektumok, amelyek a click illetve a zero szignált váltják ki.

a háttérben lesz egy **Timer** típusú objektum, amely tick szignált vált ki

Application osztály statikus metódusai gondoskodnak arról, hogy az események a megfelelő vezérlőkhöz jussanak el, hogy ott szignálokat váltsanak ki.

```
static class Program
{
    [STAThread]
    static void Main(){
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Stopwatch());
    }
}
```

program.cs

Stopwatch osztály, mint .net Form


```
public class Stopwatch : Form
{
    enum State { stopped, operate };
    State currentState;
    DateTime seconds = new (0);

    private System.Windows.Forms.Timer timer;
    private System.Windows.Forms.Button clickButton;
    private System.Windows.Forms.Button resetButton;
    private System.Windows.Forms.TextBox lcd;

    public Stopwatch()
    {
        InitializeComponent();
        currentState = State.stopped;
        display();
        timer.Start();
    }

    private void display() { ... }

    private void timer_Tick(object sender, EventArgs e) { ... }
    private void clickButton_Click(object sender, EventArgs e) { ... }
    private void resetButton_Click(object sender, EventArgs e) { ... }
    private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
    { timer.Stop(); }
}
```



szignálok eseménykezelői
(lásd állapot-átmenet tábla
megfelelő sorait)

Stopwatch.cs

Visual Studio-val tervezett kód

```
public InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.clickButton = new System.Windows.Forms.Button();
    this.resetButton = new System.Windows.Forms.Button();
    this.lcd = new System.Windows.Forms.TextBox();
    this.timer = new System.Windows.Forms.Timer(this.components);
    ...
    this.Text = "Stopwatch";
    this.clickButton.Text = "Start/Stop";
    this.resetButton.Text = "Reset";
    this.lcd.Text = "00:00";
    this.timer.Interval = 1000;
    ...
    this.Controls.Add(this.lcd);
    this.Controls.Add(this.clickButton);
    ...
    this.timer.Tick += new System.EventHandler(this.timer_Tick);
    this.zeroButton.Click += new System.EventHandler(this.resetButton_Click);
    this.clickButton.Click += new System.EventHandler(this.clickButton_Click);
    this.FormClosed += new System.Windows.Forms.
        FormClosedEventHandler(this.MainForm_FormClosed);
}
```

a kód ezen része automatikusan generálható a vizuális kódolással

szignálok és kezelőik egymáshoz rendelése

Stopwatch.cs

Stopwatch Java-ban



Stopwatch extends **JFrame** egy ablakszerű vezérlő objektum osztálya, amely objektum más vezérlőket (időzítő, kijelző, nyomógomb) is tartalmazhat. Az ablak bezárása kiváltja a quit szignált.

LCDNumber osztály példánya a kijelző, amely `display` metódussal rendelkezik.

JButton típusú objektumok, amelyek a click illetve a zero szignált váltják ki.

a háttérben lesz egy **Timer** típusú objektum, amely tick szignált vált ki

```
public class Stopwatch extends JFrame
{
    ...
    public static void main(String[] args) {
        new Stopwatch();
    }
}
```

Stopwatch.java

Stopwatch osztály, mint JFrame


```
public class Stopwatch extends JFrame
{
    enum State { operate, stopped }
    private State currentState;
    private int seconds = 0;

    private final static int SECOND = 1000 /* milliseconds */;
    private Timer timer = new Timer(SECOND, null);
    private LcdNumber lcd = new LcdNumber("00:00");
    private JButton clickButton = new JButton("Start/Stop");
    private JButton resetButton = new JButton("Reset");
    private JPanel buttonPanel = new JPanel();

    public Stopwatch() { ... }

    void click() { ... }
    void reset() { ... }
    void tick() { ... }
    protected void finalize() throws Throwable { ... timer.stop() ... }

    public static void main(String[] args) {
        new Stopwatch();
    }
}
```



szignálok eseménykezelői
(lásd állapot-átmenet tábla
megfelelő sorait)

Stopwatch.java

Java-s Stopwatch konstruktora

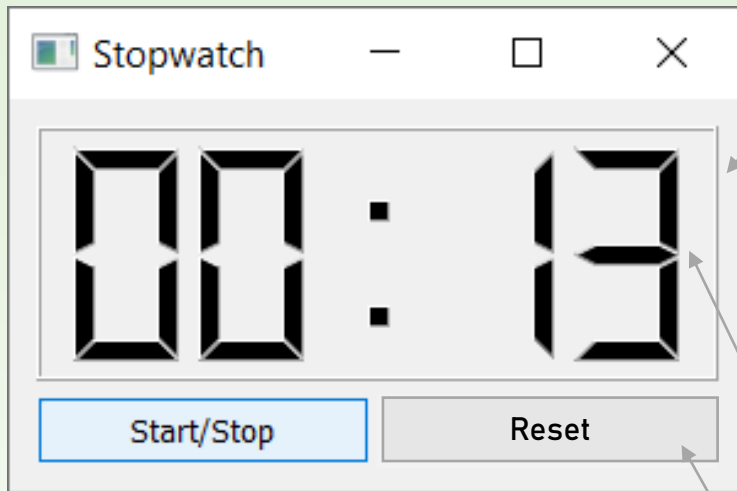
```
public Stopwatch() {  
    super("Stopwatch");  
    setBounds(250, 250, 300, 200);  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    buttonPanel.setBackground(Color.WHITE);  
    buttonPanel.add(clickButton);  
    buttonPanel.add(resetButton);  
    add(lcd);  
    add(buttonPanel, "South");  
  
    clickButton.addActionListener(new ActionListener() {  
        @Override public void actionPerformed(ActionEvent e){ click(); }  
    });  
    zeroButton.addActionListener(new ActionListener() {  
        @Override public void actionPerformed(ActionEvent e){ reset(); }  
    });  
    timer.addActionListener(new ActionListener() {  
        @Override public void actionPerformed(ActionEvent e){ tick(); }  
    });  
    currentState = State.stopped;  
    lcd.display(seconds);  
    timer.start();  
    setVisible(true);  
}
```

quit

események és kezelésük egymáshoz rendelése

Stopwatch.java

Stopwatch Qt-val fejlesztve



Stopwatch : **public QWidget** egy ablakszerű vezérlő objektum osztálya, amely objektum más vezérlőket (időzítő, kijelző, nyomógomb) is tartalmazhat. Az ablak bezárása kiváltja a quit szignált.

QLCDNumber osztály példánya a kijelző, amely display metódussal rendelkezik.

QPushButton típusú objektumok, amelyek a click illetve a zero szignált váltják ki.

a háttérben lesz egy **QTimer** típusú objektum, amely tick szignált vált ki

QApplication típusú objektum többek között gondoskodik arról, hogy az események a megfelelő vezérlőkhöz jussanak el, hogy ott szignálokat váltsanak ki.

```
#include <QApplication>
#include "stopwatch.h"
int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    Stopwatch *stopwatch = new Stopwatch;
    stopwatch ->show();
    return app.exec();
}
```

main.cpp

Stopwatch osztály, mint QWidget

```
#include <QWidget>
enum State {stopped, operate};

class Stopwatch: public QWidget
{
    Q_OBJECT
private:
    QTimer          *_timer;
    QLCDNumber      *_lcd;
    QPushButton     *_clickButton;
    QPushButton     *_resetButton;
    State _currentState;
    int _seconds;
    QString Stopwatch::format(int n) const;
public:
    Stopwatch(QWidget *parent=0);
private slots:
    void oneSecondPass();
    void clickButtonPressed();
    void resetButtonPressed();
protected:
    void closeEvent(QCloseEvent * event) { _timer->stop();
};
```

tick, click, zero szignálok
eseménykezelői
(lásd állapot-átmenet tábla
megfelelő sorait)

quit szignál eseménykezelője

stopwatch.h

Qt-s Stopwatch konstruktora

```
Stopwatch::Stopwatch(QWidget *parent) : QWidget(parent)
{
```

```
    setWindowTitle(tr("Stopwatch"));
    resize(150, 60);
```

```
    _timer = new QTimer;
    _lcd = new QLCDNumber;
    _clickButton = new QPushButton("Start/Stop");
    _resetButton = new QPushButton("Reset");
```

```
    ...
```

```
    connect(_timer, SIGNAL(timeout()), this, SLOT(oneSecondPass()));
    connect(_clickButton, SIGNAL(clicked()), this,
        SLOT(clickButtonPressed()));
    connect(_resetButton, SIGNAL(clicked()), this,
        SLOT(resetButtonPressed()));
```

```
    _currentState = stopped;
    _seconds = 0;
    _lcd->display(_seconds);
    _timer->start(1000);
```

```
}
```

itt kerül sor a grafikus vezérlőknek az ablakban való elrendezésére és egyéb tulajdonságainak megadására. Ez automatikusan is generálható, ha vizuális tervezőt (QtDesigner) használunk

szignálok és kezelők egymáshoz rendelése:

stopwatch.cpp