

# Eseményvezérelt alkalmazások

1. rész

Benzinkút

Gregorics Tibor

gt@inf.elte.hu

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

# Feladat

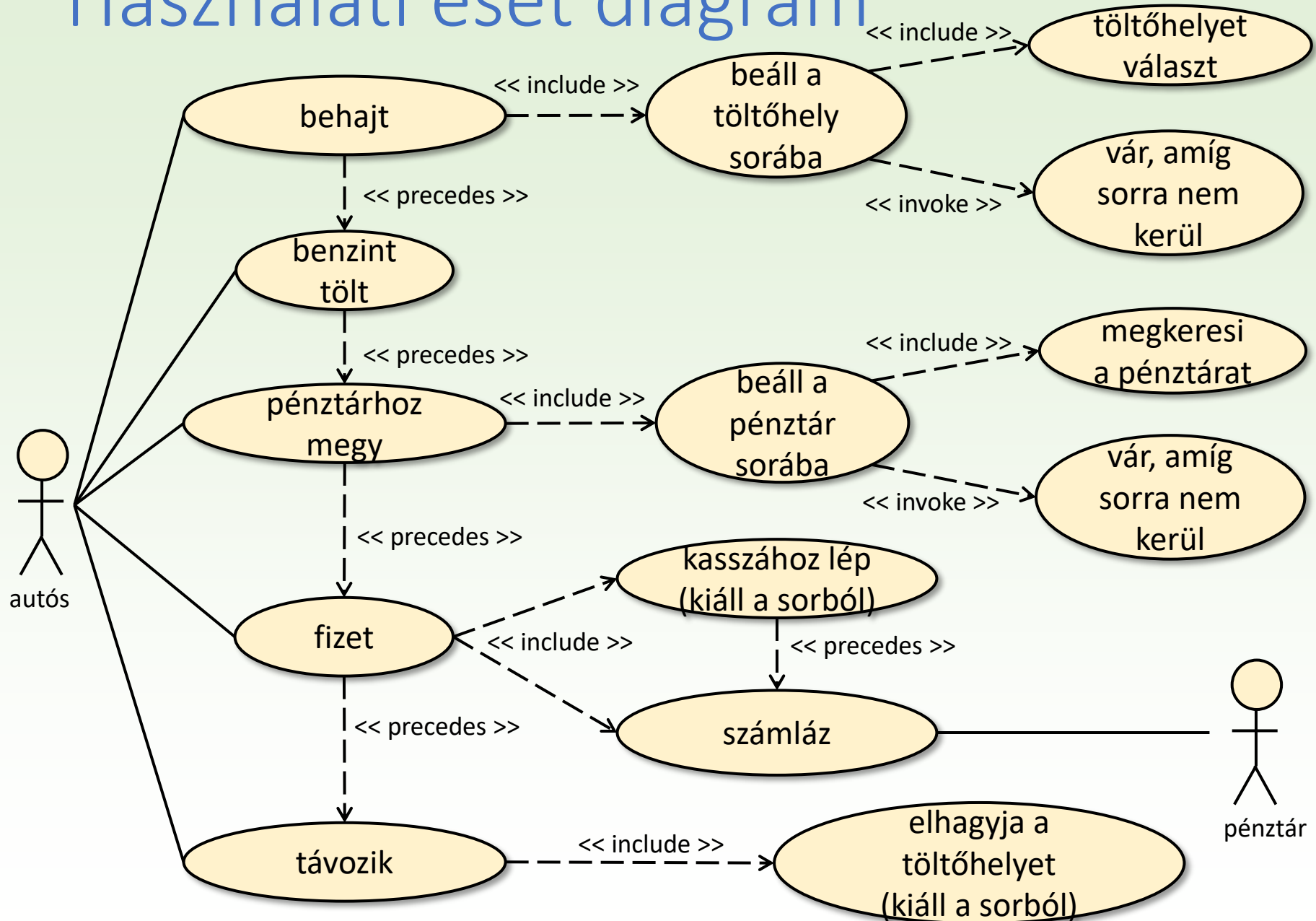
❑ Egy benzinkútnál több töltőhely és egy több kasszából álló pénztár működik.

- Az autósok behajtanak, és beállnak valamelyik töltőhelyhez tankolni.
- Amikor sorra kerülnek, akkor kívánt mennyiségű benzint vesznek fel.
- Ezután elmennek fizetni: beállnak a pénztárhoz várakozók sorába.
- Amint egy kassa szabad lesz, a soron következő autós által fizetendő összeget kiszámolják.
- Fizetés után az autós kihajt a töltőhelyről, és távozik.

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



# Használati eset diagram



# Felhasználói esetek

| eset            |       | leírás   |
|-----------------|-------|--|
| behajt          | GIVEN | létezik a benzinkút töltőhelyekkel   |
|                 | WHEN  | beáll egy töltőhelynél álló sorba  |
|                 | THEN  | várakozik  |
| benzint tölt    | GIVEN | egy töltőhelyen elsőként áll a sorban  |
|                 | WHEN  | benzint tölt   |
|                 | THEN  | a kijelző mutatja a felvett benzint  |
| pénztárhoz megy | GIVEN | létezik a benzinkút pénztárral   |
|                 | WHEN  | beáll a pénztárnál álló sorba  |
|                 | THEN  | várakozik  |
| fizet           | GIVEN | pénztárnál áll a sorban, autója egy töltőhelynél áll                                 |
|                 | WHEN  | egy kassa üres   |
|                 | THEN  | kiáll a sorból, kiszámolják a fizetendő összeget, lenullázzák a töltőhely kijelzőjét |
| távozik         | GIVEN | a benzinkút egyik töltőhelyén áll, kijelző nulla                                     |
|                 | WHEN  | távozik  |
|                 | THEN  | elhagyja a töltőhelyet (kiáll a sorból)  |

| eset         |       | leírás                               |
|--------------|-------|--------------------------------------|
| benzint tölt | GIVEN | az egyik töltőhelyen elsőként áll    |
|              | WHEN  | nulla liter benzint tölt             |
|              | THEN  | figyelmeztetés                       |
| távozik      | GIVEN | töltőhelyen áll, a kijelző nem nulla |
|              | WHEN  | távozik                              |
|              | THEN  | riasztás                             |

# Felhasználói (hibás) esetek

| eset            |       | leírás  |
|-----------------|-------|---|
| behajt          | GIVEN | nem létezik a benzinkút vagy nem létezik a kiválasztott töltőhely |
|                 | WHEN  | beáll egy töltőhelynél álló sorba                                 |
|                 | THEN  | hibajelzés  |
| benzint tölt    | GIVEN | nem áll töltőhelyen   |
|                 | WHEN  | benzint tölt  |
|                 | THEN  | hibajelzés  |
| pénztárhoz megy | GIVEN | létezik a benzinkút, de nem létezik pénztár                       |
|                 | WHEN  | beáll a pénztárnál álló sorba                                     |
|                 | THEN  | hibajelzés  |
| fizet           | GIVEN | pénztárnál áll a sorban, de nem áll töltőhelynél                  |
|                 | WHEN  | egy kassza üres   |
|                 | THEN  | hibajelzés  |
| távozik         | GIVEN | nem áll töltőhelynél  |
|                 | WHEN  | távozik   |
|                 | THEN  | hibajelzés  |

# Elemzés

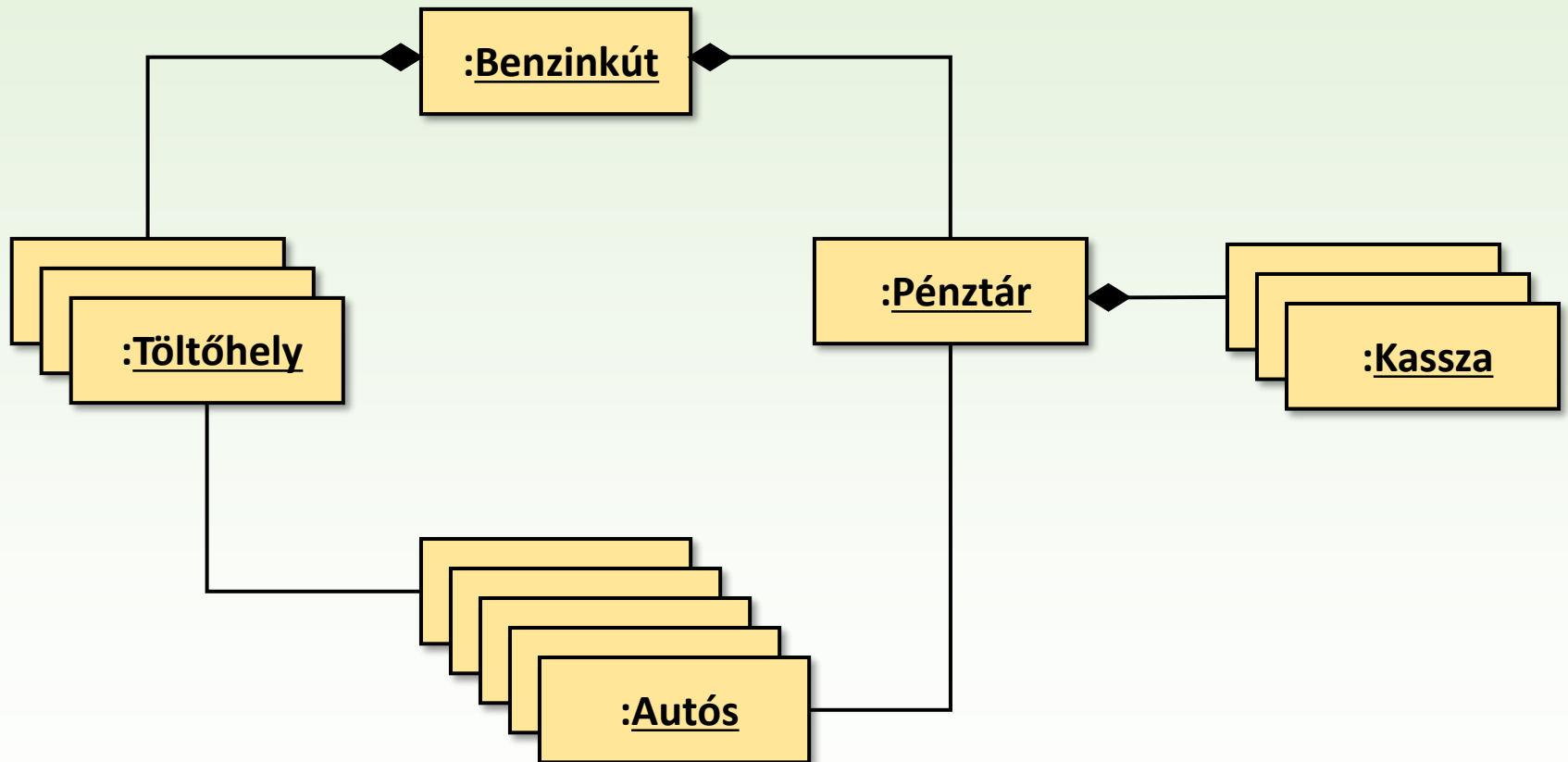
## ❑ Objektumok és tevékenységeik:

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

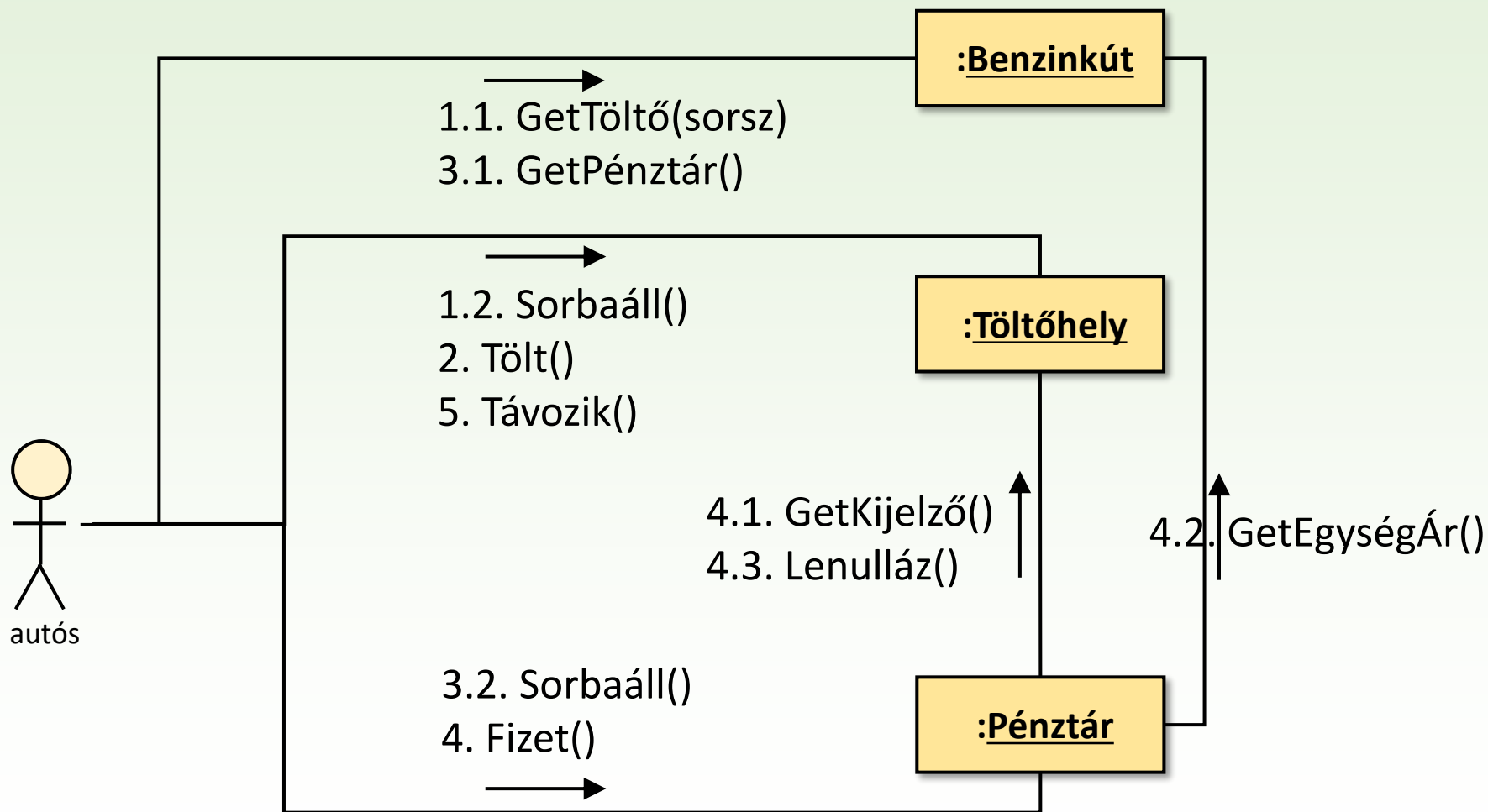
## ❑ Objektumok közötti kapcsolatok:

- a benzinkút részei a töltőhelyek és a pénztár
- egy autós ideiglenesen kapcsolatba kerül egy töltőhellyel és a pénztárral.

# Objektum diagram

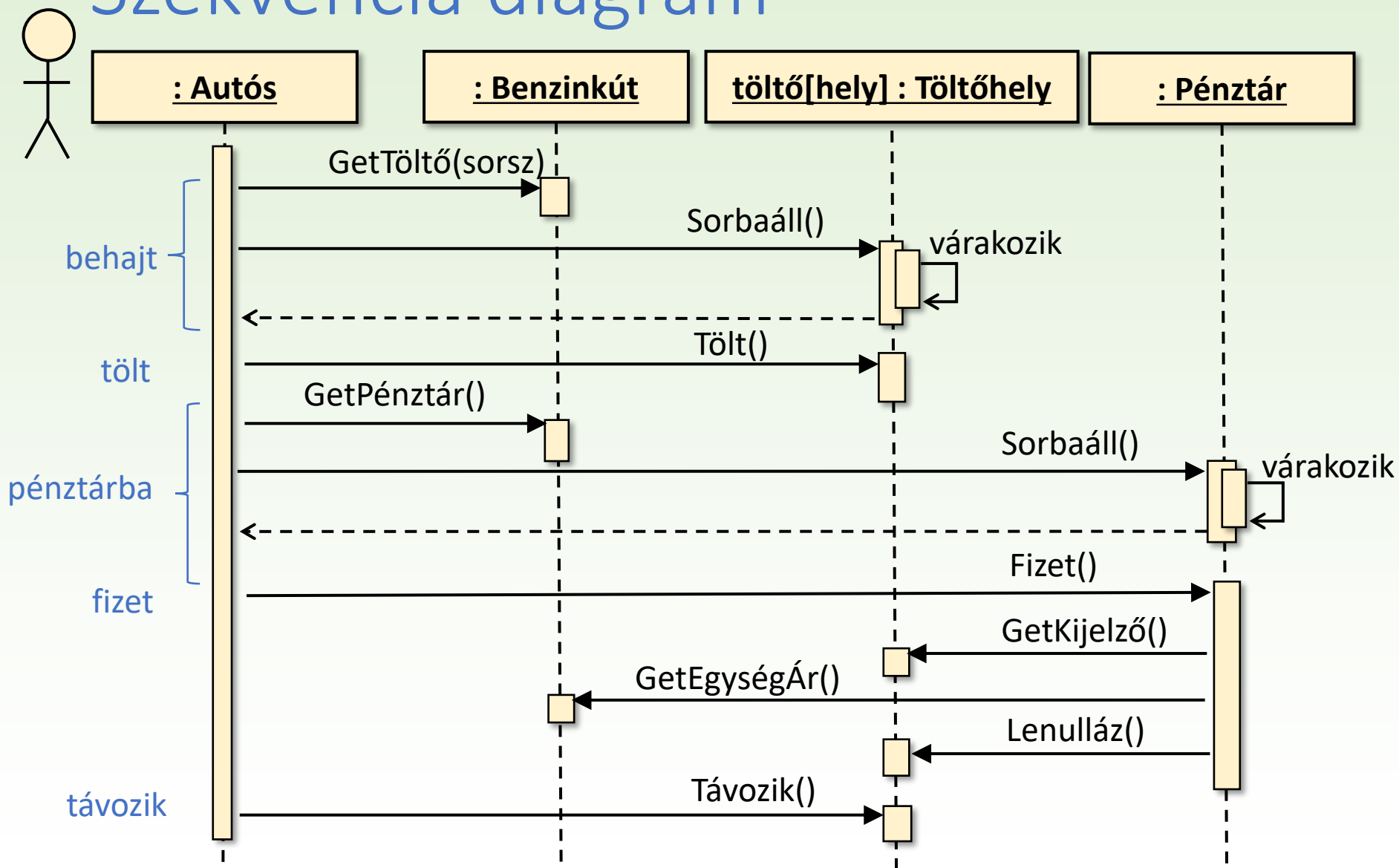


# Kommunikációs diagram

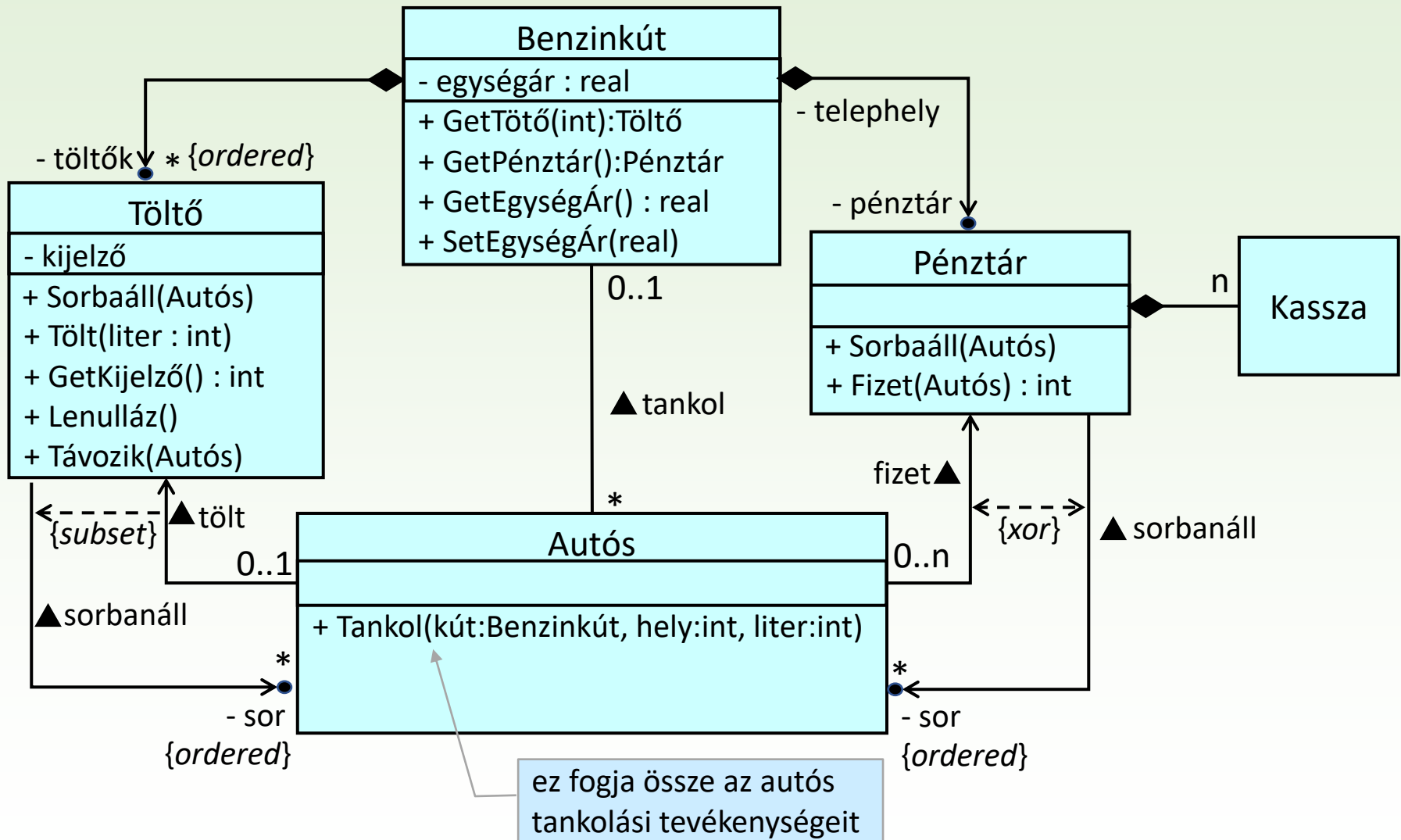




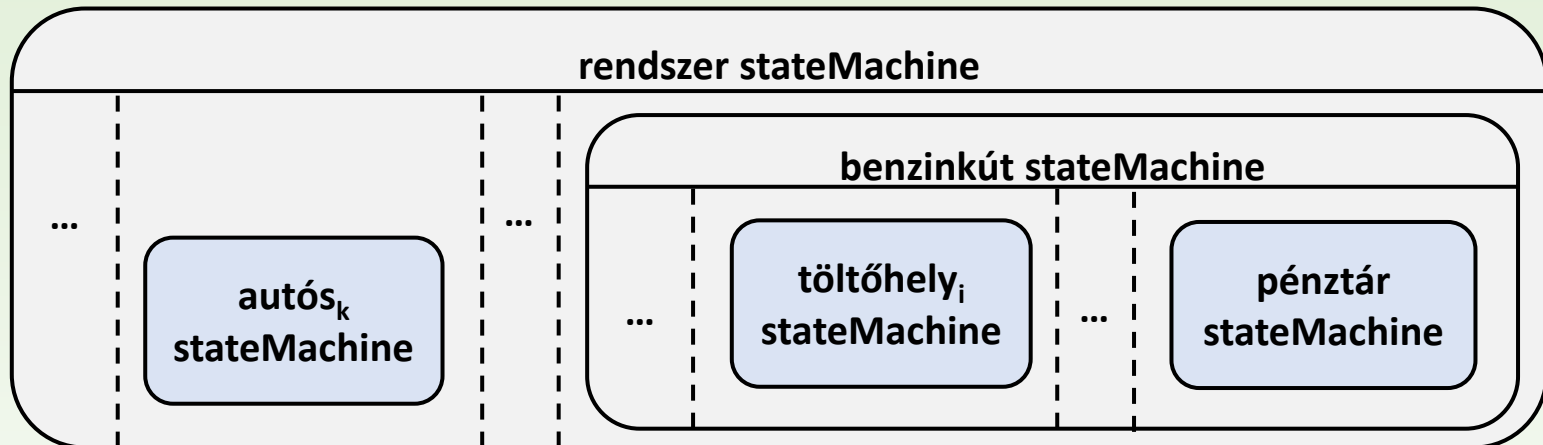
# Szekvencia diagram



# Osztály diagram



# Rendszer állapotgépe

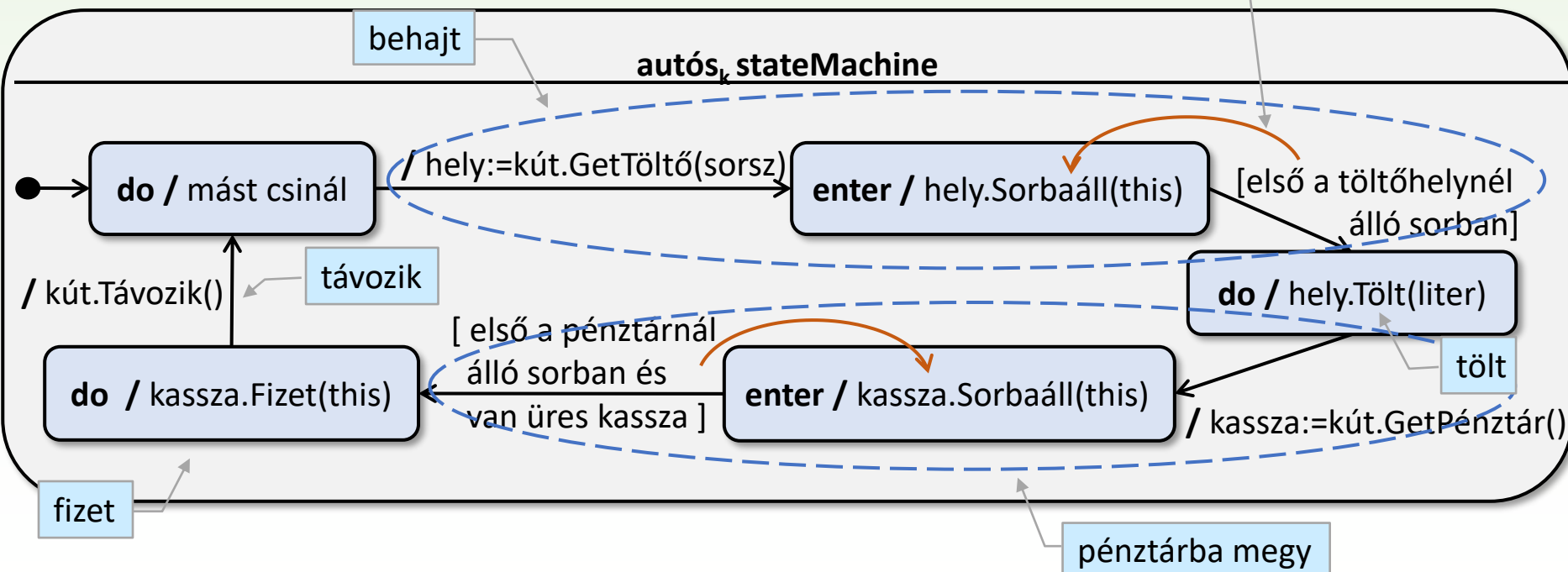


- ❑ A rendszer állapotát az autósok és a benzinkút állapota határozza meg. A benzinkút állapota a töltőhelyek és a pénztár állapotától függ.
- ❑ Az autósok „önerőből” végeznek tevékenységet, így állapotgépeik külön szálakon futnak majd.
- ❑ A benzinkút állapotgépe más objektumok által küldött szinkron üzenetek hatására működik. Nem igényel külön szálát.

# Autós objektum állapotgépe

- Az állapotok a huzamosabb ideig tartó tevékenységeket jelölik:
  - mást csinál, egy töltőhelynél sorba áll, üzemanyagot vesz fel, a pénztárban sorba áll, fizet
- Az átmeneteket nem üzenetek váltják ki, hanem a megelőző állapot tevékenységének befejeződése, vagy őrfeltétel bekövetkezése.

Azért, hogy az átmenetek rövid idejű tevékenységek legyenek, az őrfeltételre várakozásokat beépítjük a sorbaállásokba.



# Autós osztálya

- ❑ Az autósok saját szálukon hívják tankol() metódusukat, amely elindítja az állapotgépet.
- ❑ A „mást csinál” tevékenységet nem implementáljuk, így az állapotgép sem alkot majd egy kört: az állapotgép ciklikusságát tankol() metódus ismételt hívásaival lehetne szimulálni.
- ❑ Eseménysor kezelésére nincs szükség

| Autós  |
|--|
| - név : string   |
| + Autós(str : string)                                      |
| + Tankol(kút : Benzinkút, sorsz : int, liter : int) : void |

```
hely := kút.GetTöltő(sorsz)
hely.Sorbaáll(this)
hely.Tölt(liter)
kassa := kút.GetPénztár()
kassa.Sorbaáll(this)
kassa.Fizet(this)
hely.Távozik(this)
```

# Autós osztály (Car.cs)

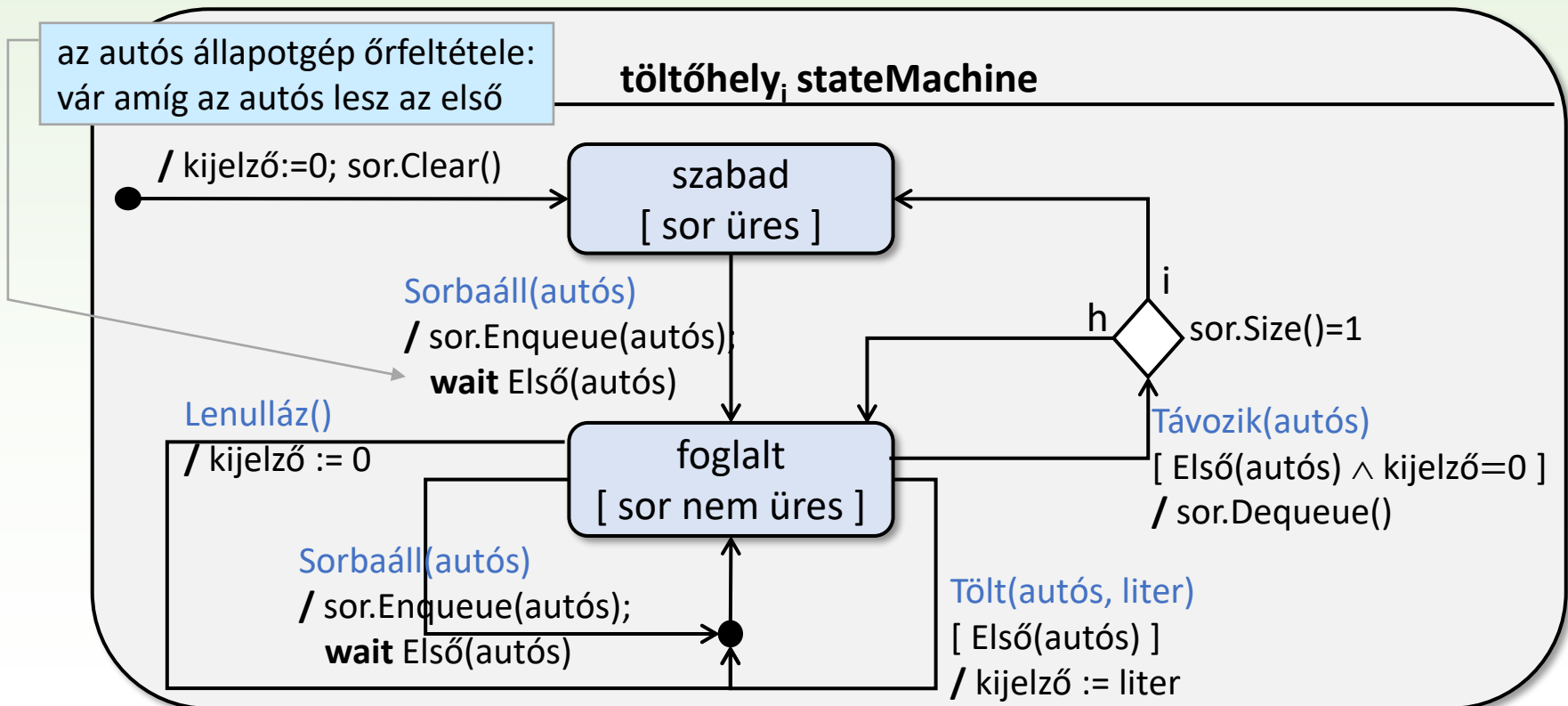
```
class Car
{
    public string Name { get; private set; }
    public Car(string str) { name = str; }
    private PetrolStation station;
    private int number;
    private int liter;
    private Thread fuelThread;

    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 cash
        int sum = station.CashDesk.Pay(this); // pays
        station.GetPump(number).Leave(this); // leaves the petrolstation
    }
}
```

külön szálon indul az autós tevékenysége

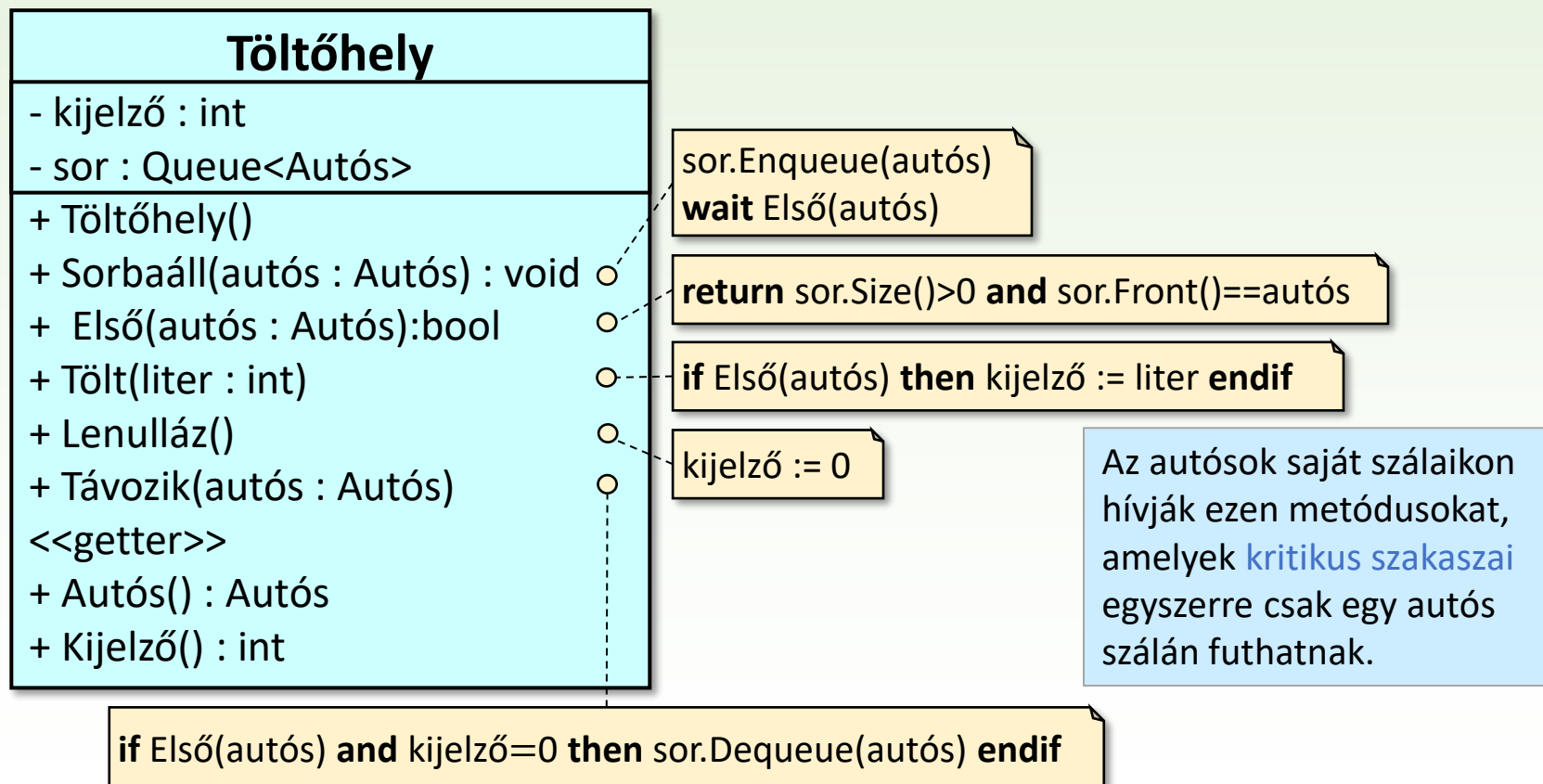
# Töltőhely objektum állapotgépe

- ❑ Egy töltőhely lehet **szabad** vagy **foglalt**.
- ❑ Az állapot-átmeneteket a töltőhely szinkron módon metódus hívásai valósítják meg: a Sorbaáll() és Távozik() a töltőhelynél álló **sorra** van hatással, a Tölt(), Lenulláz() a töltőhely **kijelzőjét** módosítja.



# Töltőhely osztály

- ❑ A töltőhely tulajdonságai közé tartozik a kijelzőn kívül az a sor, amelybe a töltőhelynél várakozó autósok kerülnek. A sor legelején álló autós az, aki tankolhat, illetve fizet.





# Töltőhely osztály (Pump.cs)

```
class Pump
{
    public int Quantity { get; private set; }

    public void ResetQuantity() { Quantity = 0; }

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

    public Pump() { ResetQuantity(); }

    private readonly object criticalSection = new ();

    public bool IsFirst(Car car)
    {
        Monitor.Enter(criticalSection);
        bool l = queue.Count > 0 && queue.Peek() == car;
        Monitor.Exit(criticalSection);
        return l;
    }
    ...
}
```

kölcsönösen kizáró módon működő  
kritikus szakaszok megjelöléséhez

kritikus szakasz eleje

kritikus szakasz vége

# Töltőhely osztály (Pump.cs)

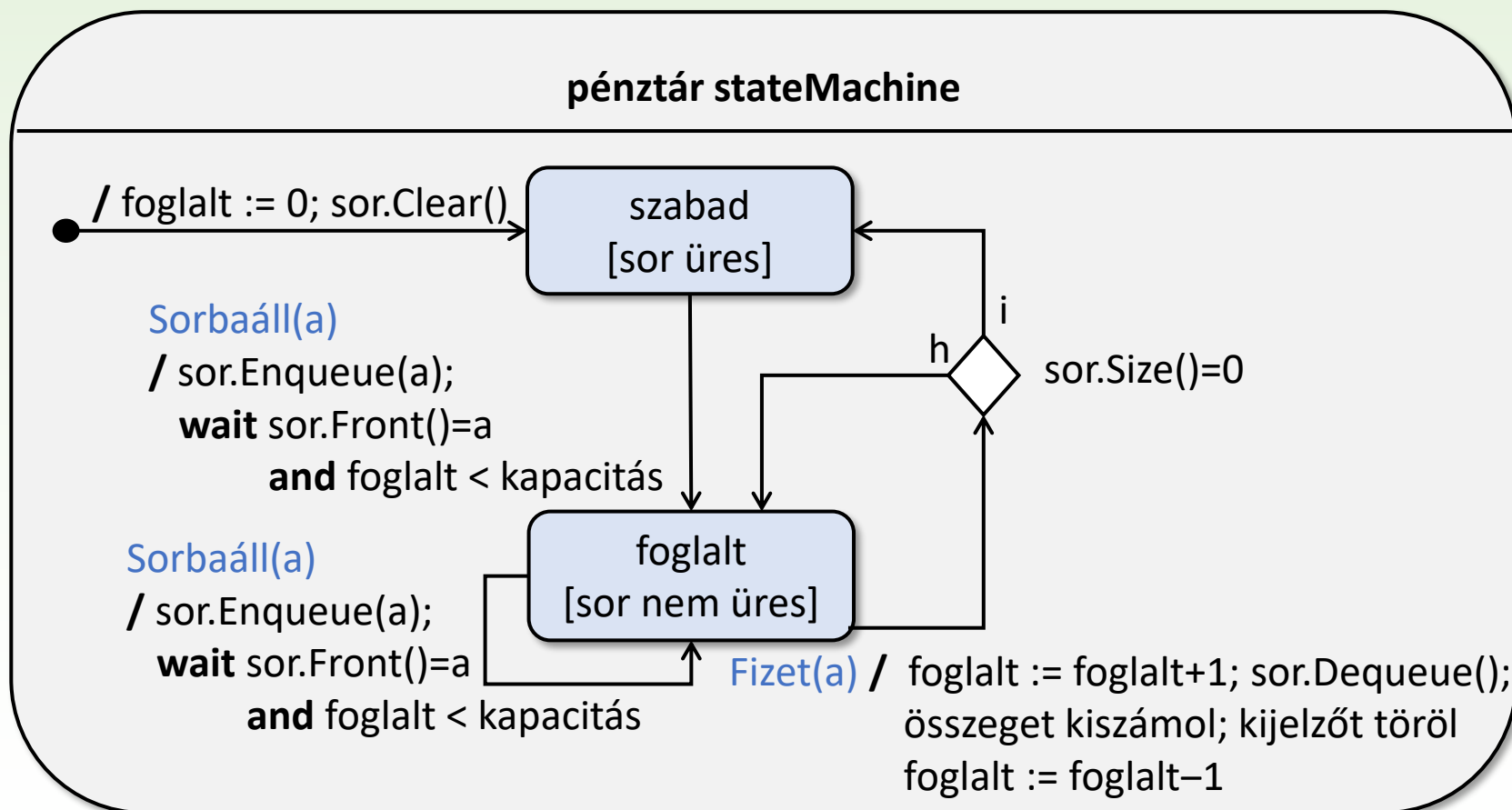
```
...
public void JoinQueue(Car car)
{
    Monitor.Enter(criticalSection);
    queue.Enqueue(car);           // joins the queue
    while ( !IsFirst(car) ) Monitor.Wait(criticalSection);
    Monitor.Exit(criticalSection);
}
public void Fill(Car car, int liter)
{
    Monitor.Enter(criticalSection);
    if ( !IsFirst(car) ) return;
    Quantity = liter;
    Monitor.Exit(criticalSection);
    Thread.Sleep(liter * 100); // time of fueling
}
public void Leave(Car car)
{
    Monitor.Enter(criticalSection);
    if ( IsFirst(car) && 0==Quantity ) queue.Dequeue(); // leaves the queue
    Monitor.PulseAll(criticalSection);
    Monitor.Exit(criticalSection);
}
}
```

aktuális szál végrehajtásának felfüggesztése azért, hogy más szál is kritikus szakaszba lépjen

elindítja az összes várakozó autós szálát

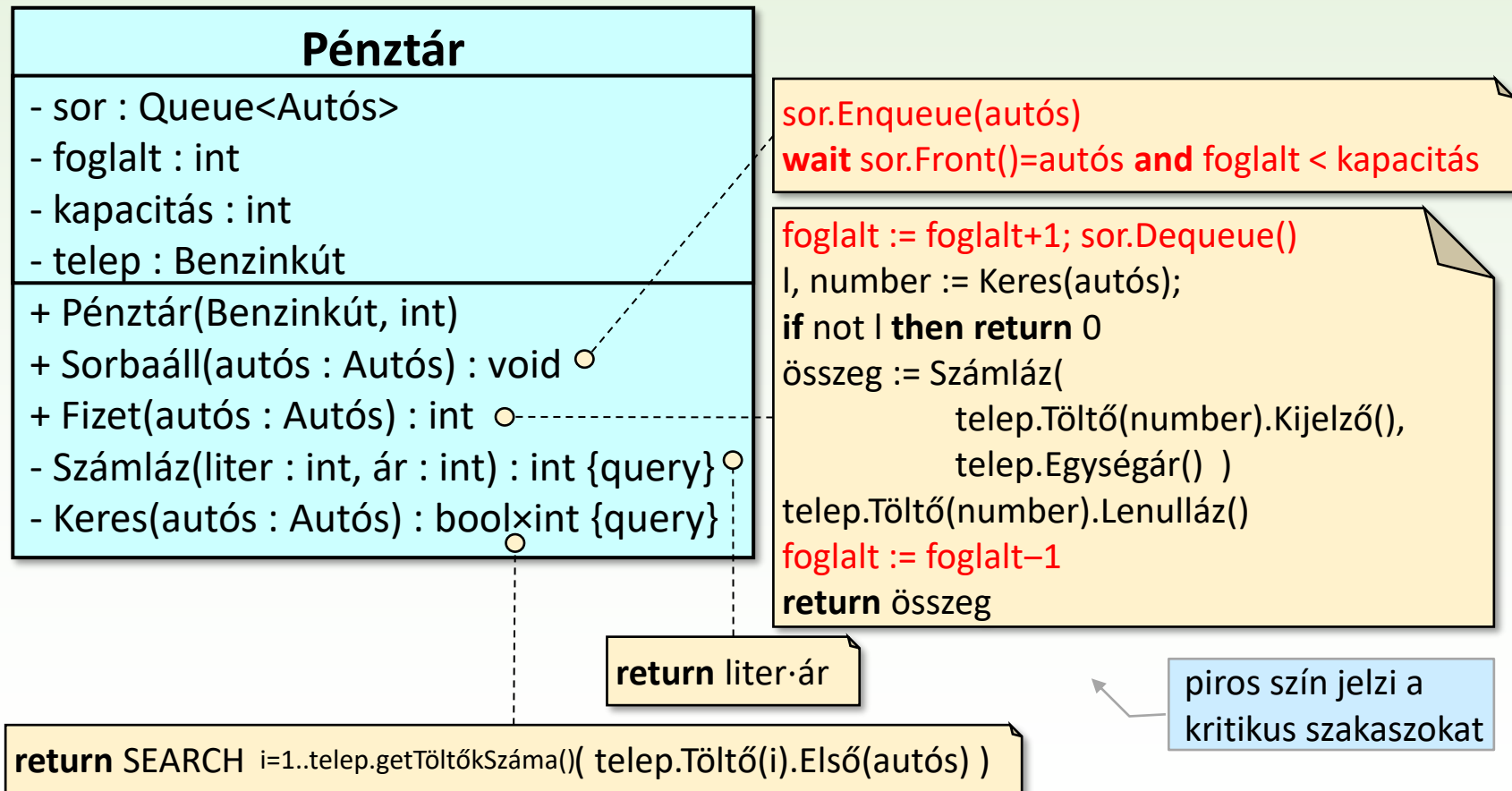
# Pénztár objektum állapotgépe

- A pénztár lehet **szabad** vagy **foglalt**. Az állapot-átmeneteket a pénztár szinkron hívott metódusai valósítják meg.



# Pénztár osztály

- Ismert a kasszák száma (**kapacitás**), a fogalt kasszák száma (**foglalt**), és az autósoknak a pénztárhoz álló sora, amelyből az első akkor lép ki, amikor felszabadul egy kassza, ahol fizetni lehet.



# Pénztár osztály (Cash.cs)

```
class Cash
{
    private readonly PetrolStation station;
    private readonly int capacity;
    public Cash(PetrolStation station, int m)
    {
        this.station = station; capacity = m;
    }

    private int engaged;
    private readonly Queue<Car> queue = new ();

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

    private bool Search(Car car, out int i)
    {
        for (i = 0; i < station.PumpsCount; ++i)
        {
            if (station.GetPump(i).IsFirst(car) ) return true;
        }
        return false;
    }
    ...
}
```

megkeresi azt a töltőhelyet,  
ahol az autós tankolt

# Pénztár osztály (Cash.cs)

```
...
private readonly private object criticalSection = new ();
public void JoinQueue(Car car)
{
    Monitor.Enter(criticalSection);
    queue.Enqueue(car);    // joins the queue
    while ( !(queue.Count > 0 && queue.Peek() == car) || engaged == capacity )
        Monitor.Wait(criticalSection);
    Monitor.Exit(criticalSection);
}
public int Pay(Car car)
{
    Monitor.Enter(criticalSection);
    ++engaged;                // steps to a cash desk
    queue.Dequeue();          // leaves the queue
    Monitor.Exit(criticalSection);
    if (!Search(car, out int i)) return 0;
    int sum = Invoice( station.GetPump(i).Quantity, station.Unit );
    station.GetPump(i).ResetQuantity(); // resets the pump
    Thread.Sleep(1000);        // elapsed time of paying
    Monitor.Enter(criticalSection);
    --engaged;                // leaves the cash desk
    Monitor.PulseAll(criticalSection);
    Monitor.Exit(criticalSection);
    return sum;
}
}
```

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

elindítja az összes  
várakozó autós szálát

# Benzinkút osztálya

- A benzinkút getter-eket szolgáltat a rendszer komponenseinek (töltőhelyek, pénztár) eléréséhez, valamint getter-t és setter-t az egységár adathoz.

| Benzinkút   |
|---|
| - töltők : Töltőhely[*]<br>- pénztár : Pénztár<br>- egységár : real   |
| + Benzinkút(n : int, m : int)<br><<getter>><br>+ GetTöltő(n : int) : Pump<br>+ GetPénztár() : Pénztár<br>+ GetEgységÁr() : double<br><<setter>><br>+ SetEgységÁr(e:real) : void |

# 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());
        }
        CashDesk = new Cash(this, m);
    }

    public Pump GetPump( int n) { return pumps[n]; }

    public int PumpsCount { get => pumps.Count; }
}
```



# Eseményvezérelt alkalmazások

2. rész

Stopper óra

Gregorics Tibor

gt@inf.elte.hu

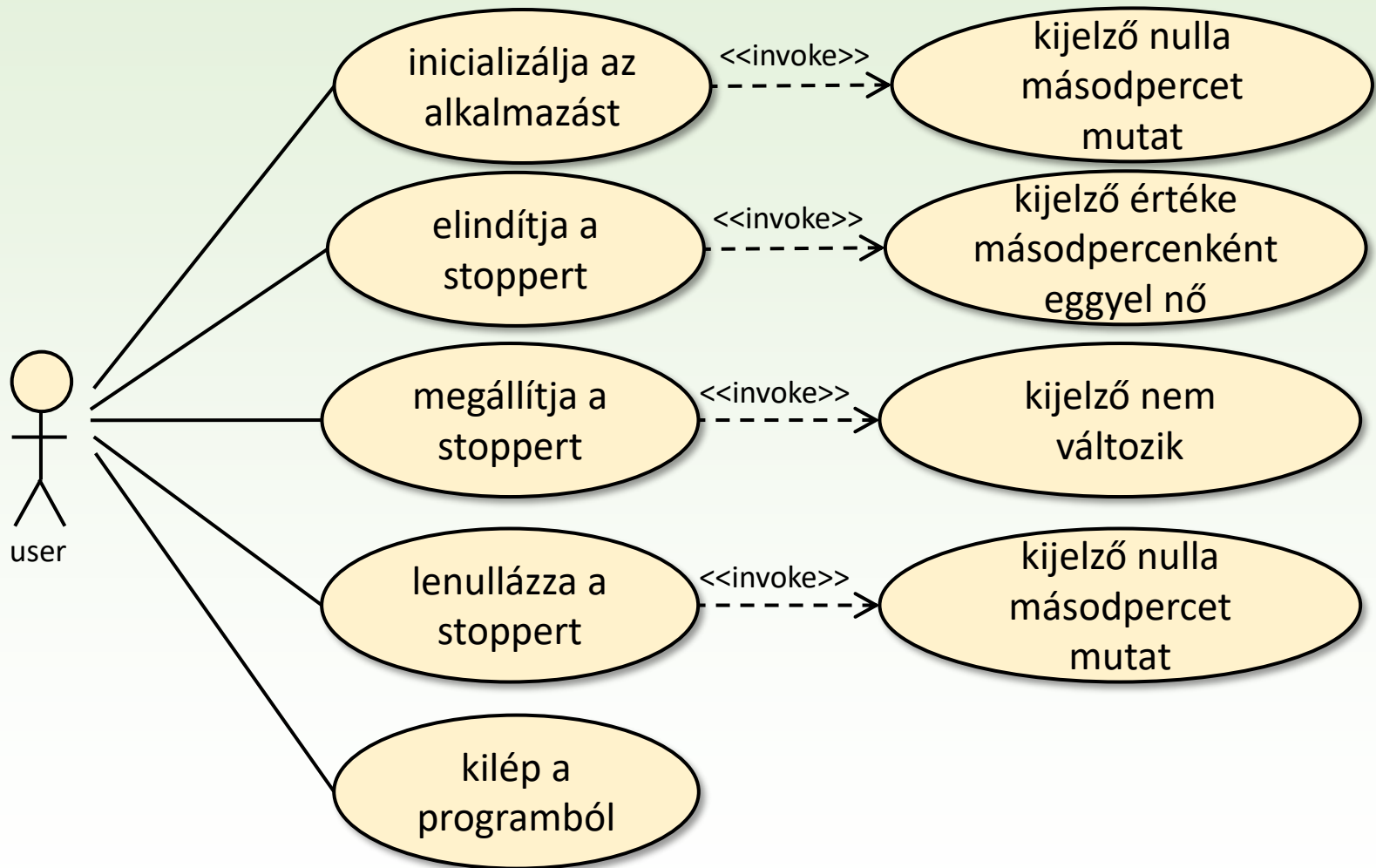
<http://people.inf.elte.hu/gt/oep>

# Feladat: Stopper

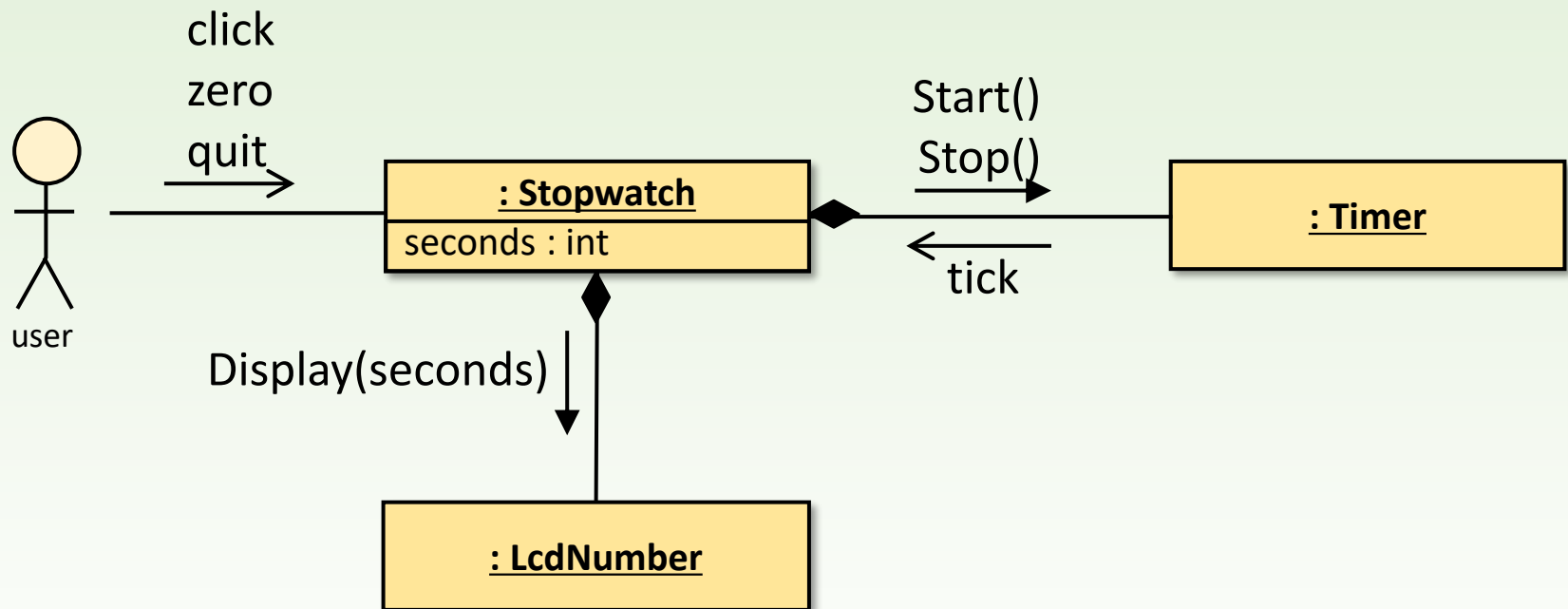
Készítsünk egy stoppert, amely másodpercenként méri a múltó időt.

- 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

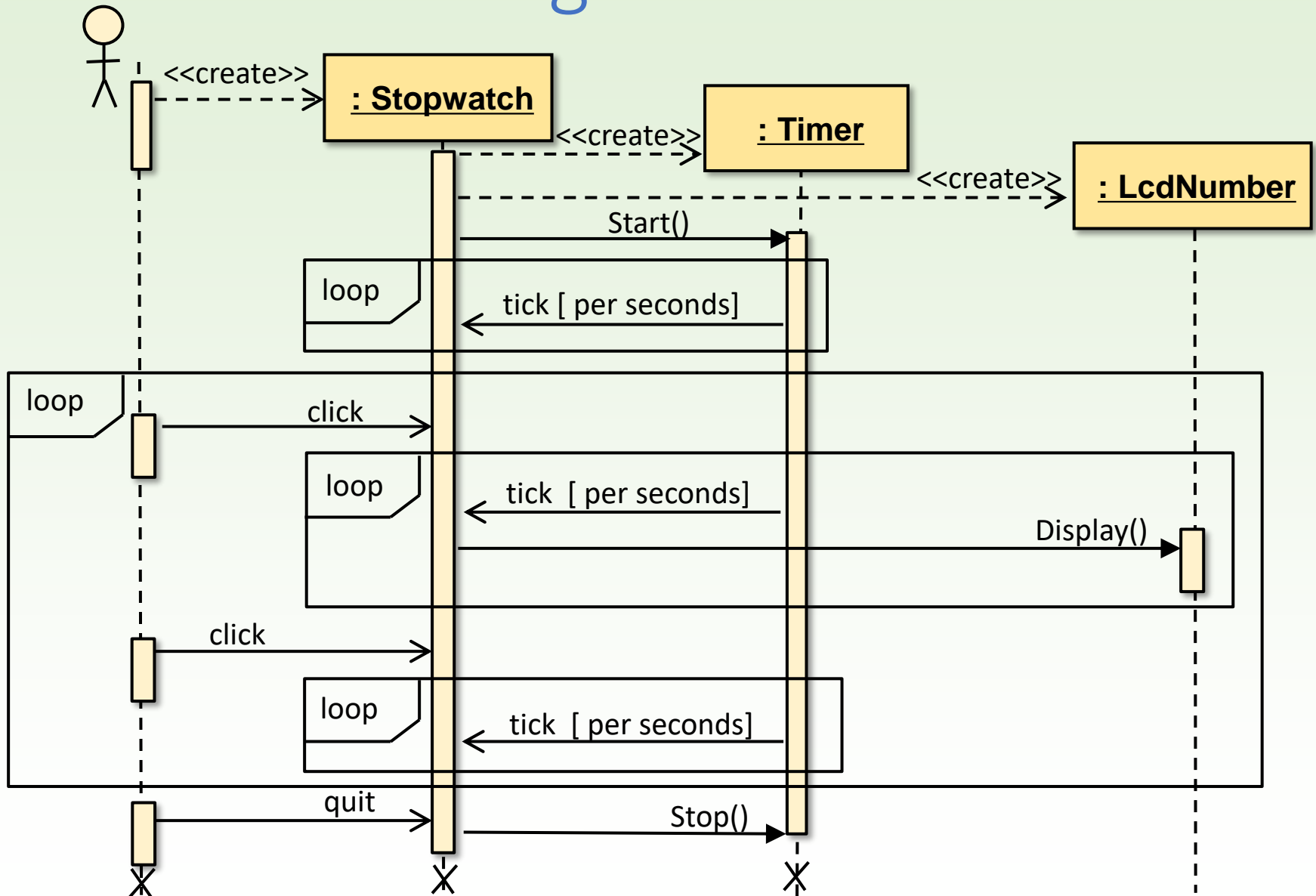


# Kommunikációs és objektum diagram

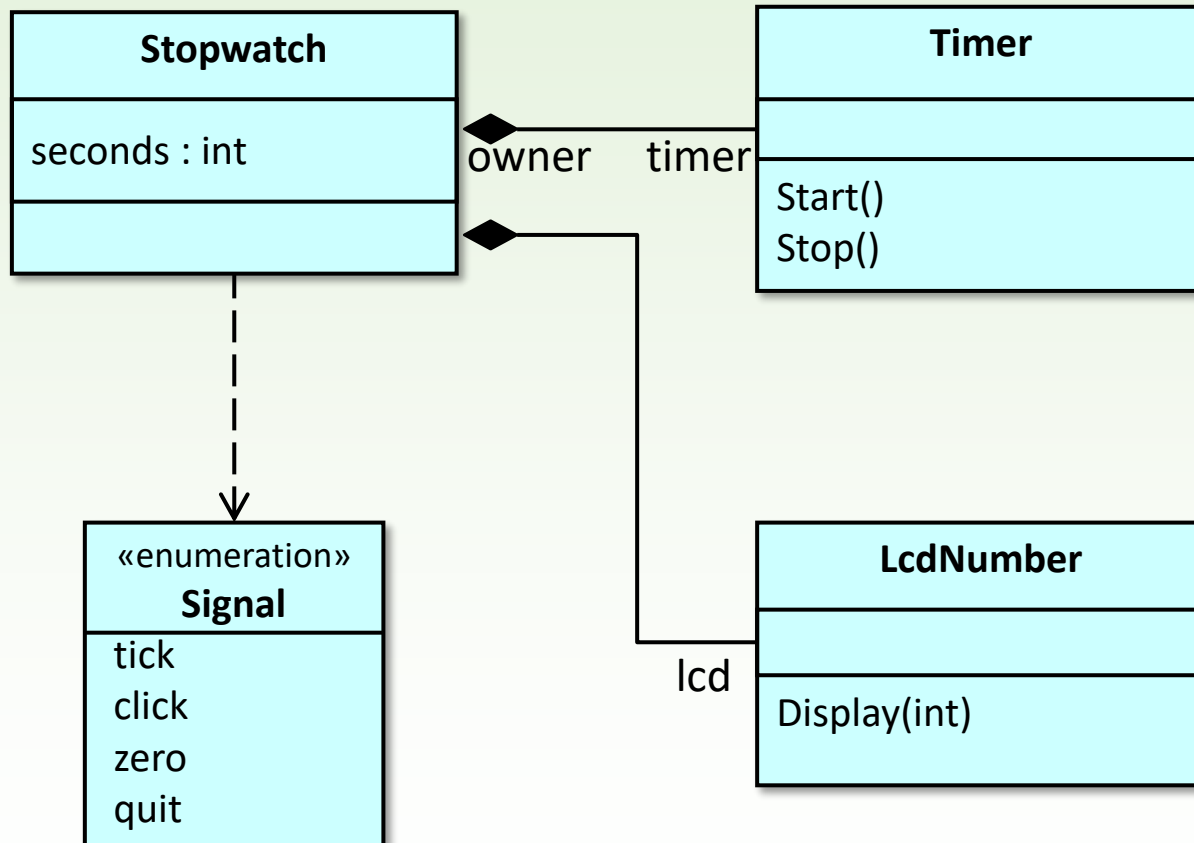


- a felhasználó és az időzítő szignálokat küld a stoppernek
- a stopper a neki küldött szignálokat aszinkron módon dolgozza fel.

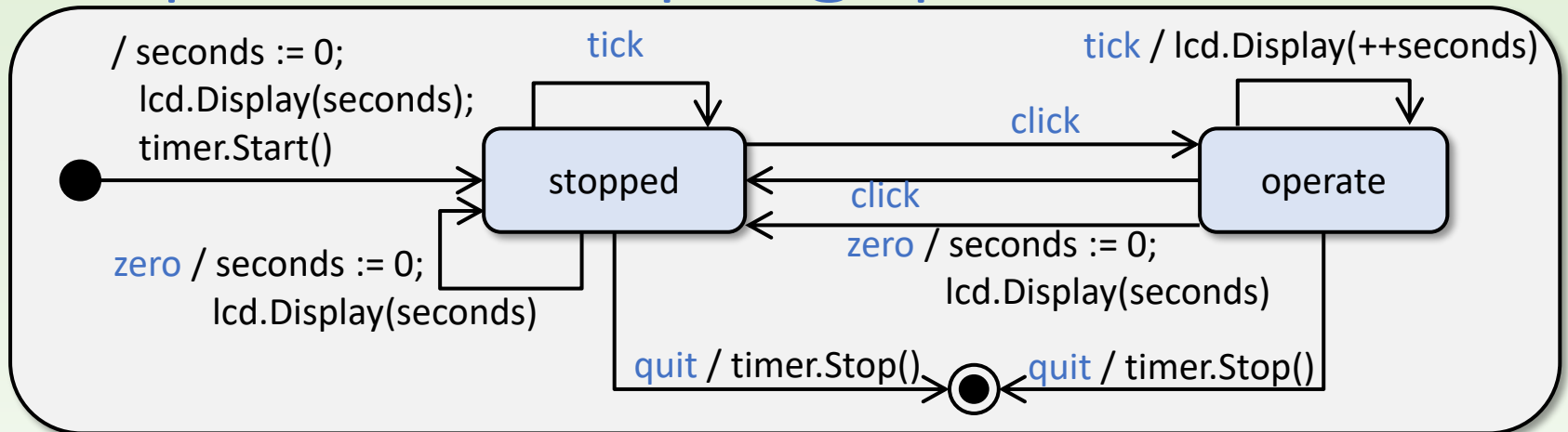
# Szekvencia diagram



# Osztálydiagram

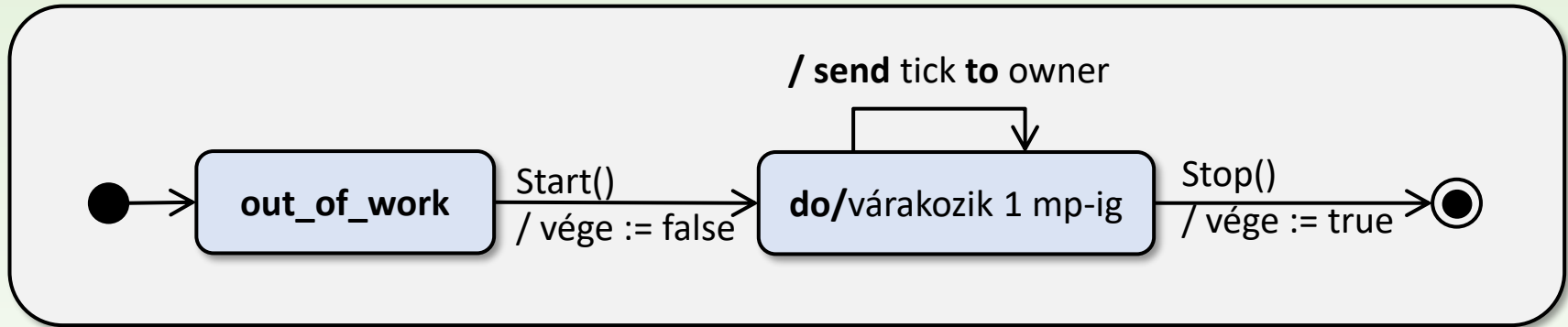


# Stopwatch állapotgépe



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

# Timer állapotgépe



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

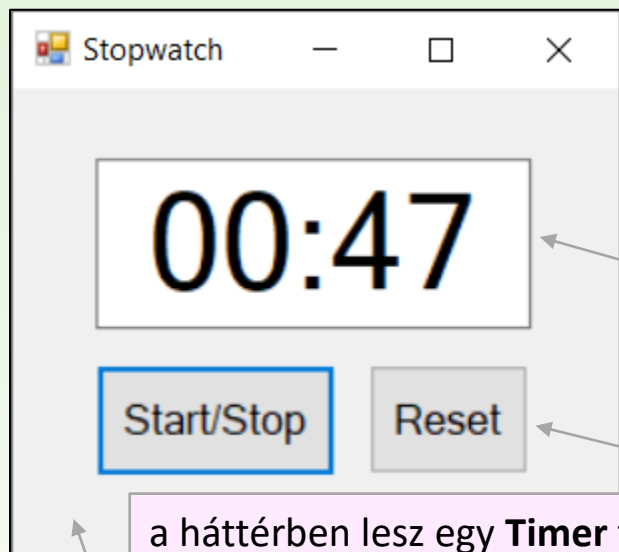


# Megvalósítás

- ❑ Implementálhatnánk a megoldást az eddig látott módon: a száakezelést, a szálbiztos eseménysorral támogatott eseménykezelést, az őrfeltételek várakozó utasítását magunk építjük be a kódba, pedig mennyivel gyorsabb és egyszerűbb lenne a fejlesztés, ha mindez „automatikusan” kerülne bele alkalmazásba.
- ❑ Eddig konzolos alkalmazásokat készítettünk, de szebb lenne grafikus megjelenést használni.
- ❑ Olyan fejlesztő környezetet keresünk, amely
  - hozzáad az alkalmazáshoz egy aszinkron eseménykezelő mechanizmust
  - lehetőséget ad grafikus megjelenéssel rendelkező objektumok létrehozására úgy, hogy azok egyedi tulajdonságait meg lehessen megadni
  - vizuális tervezőt biztosít a grafikus megjelenítés kialakításához
  - speciális objektumok (pl. időzítő) létrehozására ad kényelmes lehetőséget



# Stopwatch .net alatt fejlesztve



**Stopwatch** : Form egy ablakszerű vezérlő objektum osztálya, amely más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz. Az ablak bezárása váltja ki a quit szignált.

**TextBox** osztály példánya a kijelző, amelyhez egy display metódust kell írni.

**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 generálható az  
automatikusan vizuális tervezéssel

szignálok és kezelőik  
egymáshoz rendelése

Stopwatch.cs

# Stopwatch Java-ban



**Stopwatch extends JFrame** egy ablakszerű vezérlő objektum osztálya, amely más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz. Az ablak bezárása váltja ki 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

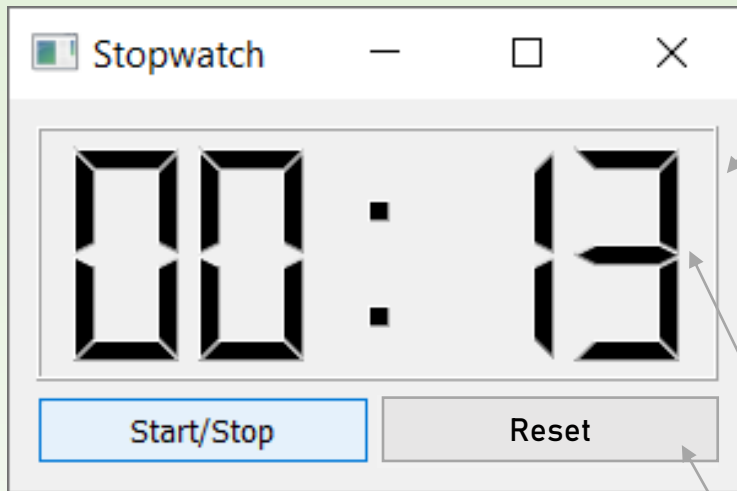
```
public Stopwatch() {  
    super("Stopwatch");  
    setBounds(250, 250, 300, 200);  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    buttonPanel.setBackground(Color.WHITE);  
    buttonPanel.add(clickButton);  
    buttonPanel.add(resetButton);  
    add(lcd);  
    add(buttonPanel, "South");  
  
    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 más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz. Az ablak bezárása váltja ki 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 kezelőik egymáshoz rendelése:

stopwatch.cpp