Objektumok viselkedése

Gregorics Tibor

gt@inf.elte.hu

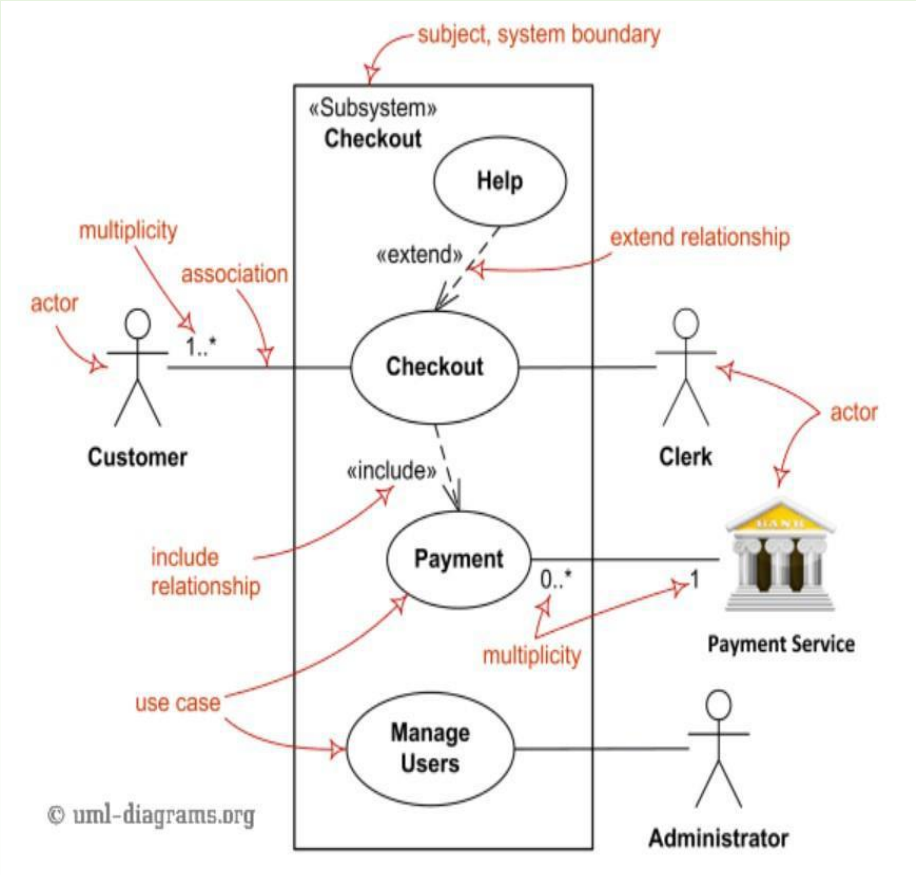
http://people.inf.elte.hu/gt/oep

Viselkedési nézetek

- Az UML az objektumok dinamikus viselkedésének jellemzésére számos nézetet vezetett be. Ezek közül az alábbiakkal ismerkedünk majd meg:
  - Használati eset (*use case*) diagram
  - Kommunikációs (*communication*) diagram
  - Szekvencia (*sequence*) diagram
  - Állapotgép (*state machine*) diagram

Használati eset diagram

- Megmutatja, hogy a tervezett rendszer
  - milyen funkciókat lát el, azaz mire lesz képes,
  - kik számára (aktorok) nyújt szolgáltatásokat
  - milyen követelményeket támaszt a környezetével szemben.



Használati esetek kapcsolatainak specifikátorai

- Használati esetek rákövetkezési sorrendje.
  - precede: felhasználó által közvetlenül kezdeményezhető két tevékenység között biztosítandó sorrendet jelöli
  - invoke: egy felhasználói tevékenység és az azt követő, de közvetlenül nem előidézhető tevékenység közötti kapcsolatot jelöli
- Használati eset kiegészítése.
  - include: egy felhasználó tevékenységnek egy jól elkülöníthető, önállóan is kezdeményezhető része, amely nélkül azonban a tartalmazó tevékenység nem teljes (absztrakt).
  - extend: egy felhasználói tevékenységet opcionálisán kiegészítő másik tevékenység, amely önmagában is teljes (soha nem absztrakt)
- Tevékenységek között is, és az aktorok között is jelölhető származtatási viszony
- Multiplicitás is megadható az aktoroknál.

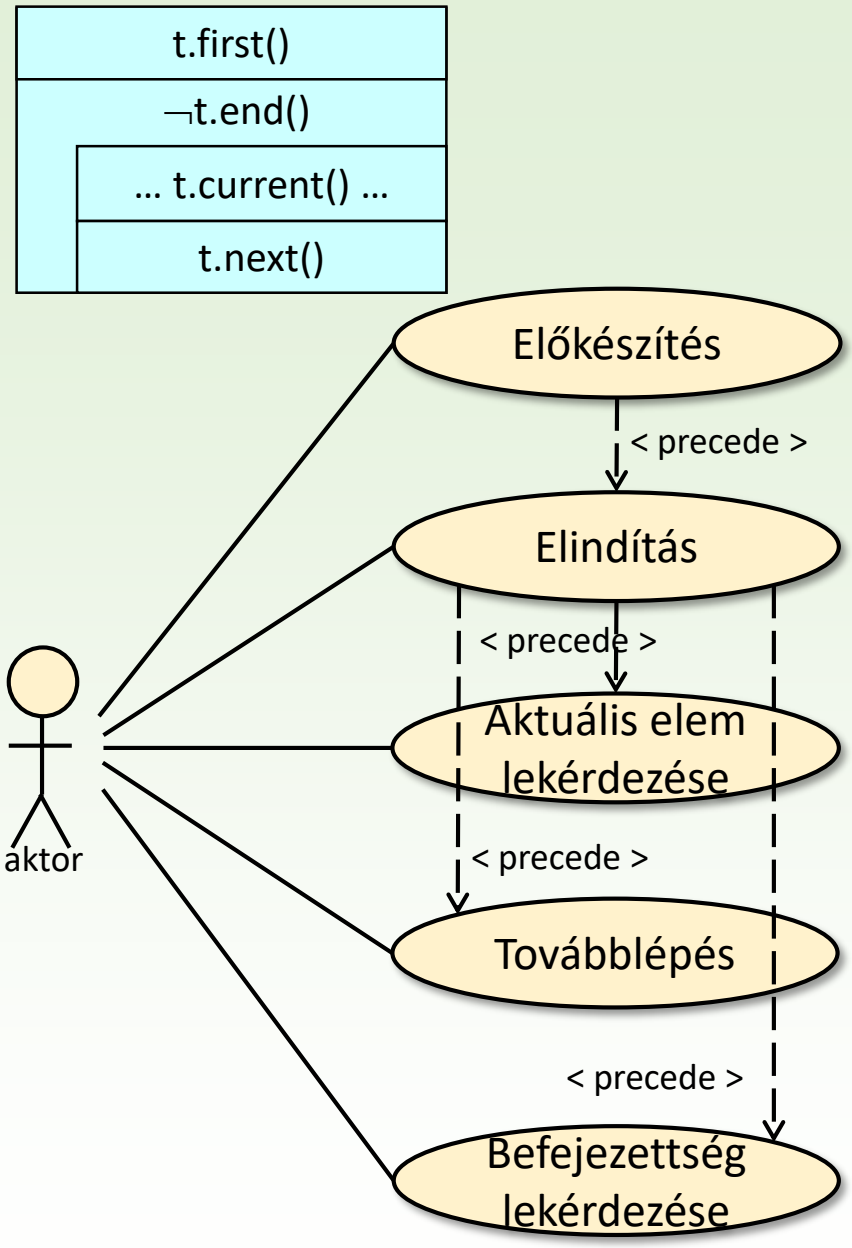
Felhasználói esetek (user story)

- A használati eset diagram önmagában nem ad elégséges képet a megvalósítandó rendszerről, hiszen az egyes tevékenységekről a nevükön kívül nem árul el semmit.
- A felhasználói esetek felhasználói csoportonként („AS a …”) történő táblázatos („user story”) leírásában minden felhasználói tevékenységet részletesen ki kell fejteni:
  - mi a tevékenység neve,
  - milyen előfeltétel meglétét feltételezi (GIVEN)
  - milyen esemény hatására következik be (WHEN)
  - mi a hatása a végrehajtásának, milyen eredményt ad (THEN).

AS a …

eset		leírás
tevékenység neve	GIVEN	tevékenység kiváltásakor feltételezett alaphelyzet
	WHEN	tevékenység kiváltása
	THEN	tevékenység hatása

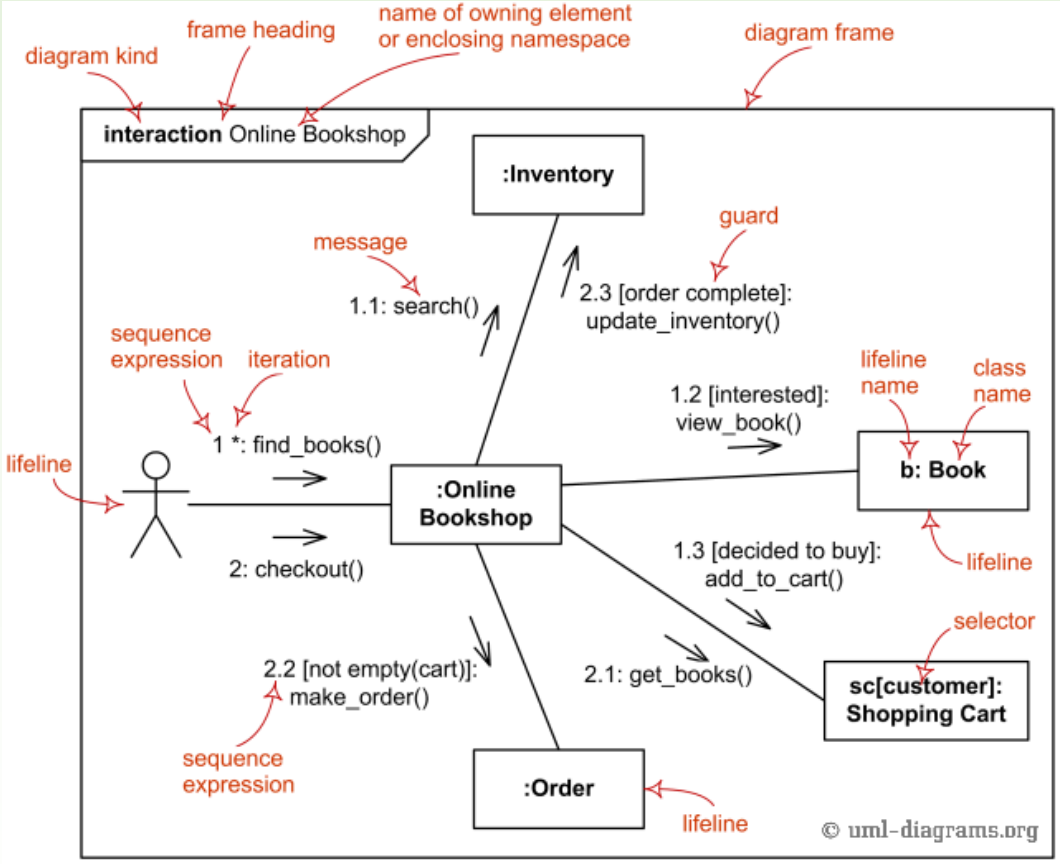
Példa: Felsorolás



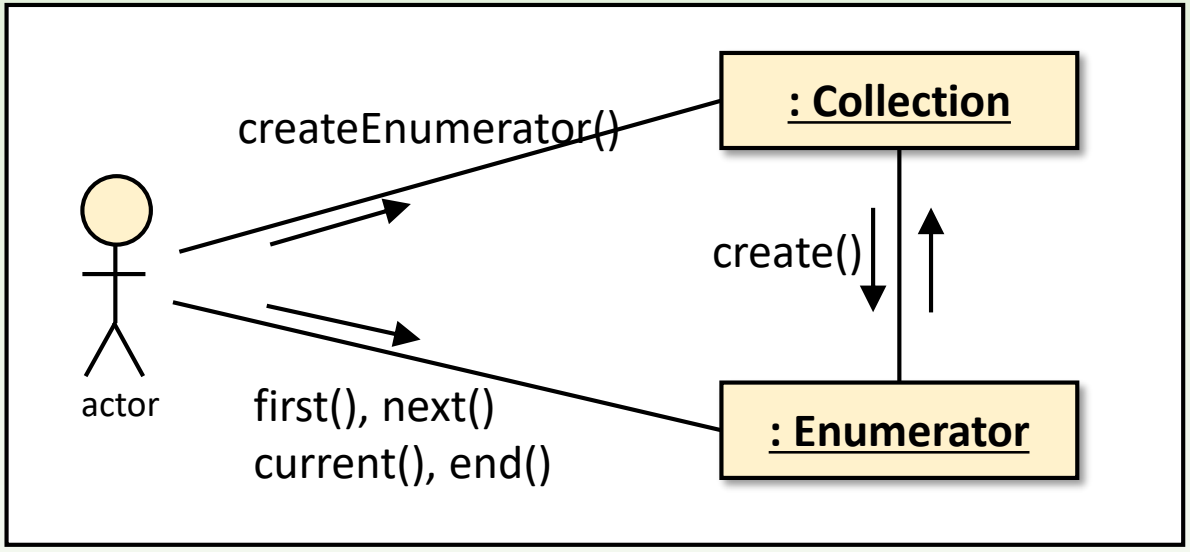
eset		leírás
Előkészítés normális esetben	GIVEN	Adott a felsorolni kívánt gyűjtemény.
	WHEN	Felsoroló példányosítása.
	THEN	Létrejön a felsoroló objektum.
Előkészítés abnormális esetben	GIVEN	Nincs felsorolni kívánt gyűjtemény.
	WHEN	Felsoroló példányosítása.
	THEN	Hiba, felsoroló objektum nem jön létre.
Elindítás normális esetben	GIVEN	Adott egy még el nem indított (pre-start) állapotú felsoroló objektum.
	WHEN	Felsorolás elindítása a **first()** művelettel.
	THEN	A felsoroló folyamatban van (in-process) állapotba kerül.
Elindítás abnormális esetben	GIVEN	Adott egy folyamatban levő (in-process) vagy befejeződött (finished) felsoroló.
	WHEN	Felsorolás elindítása a **first()** művelettel.
	THEN	Hiba, és a felsoroló megőrzi állapotát.
...		

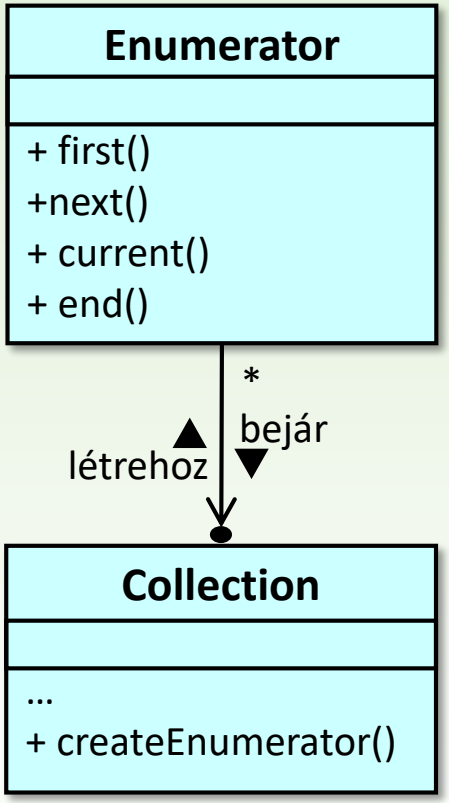
Kommunikációs diagram

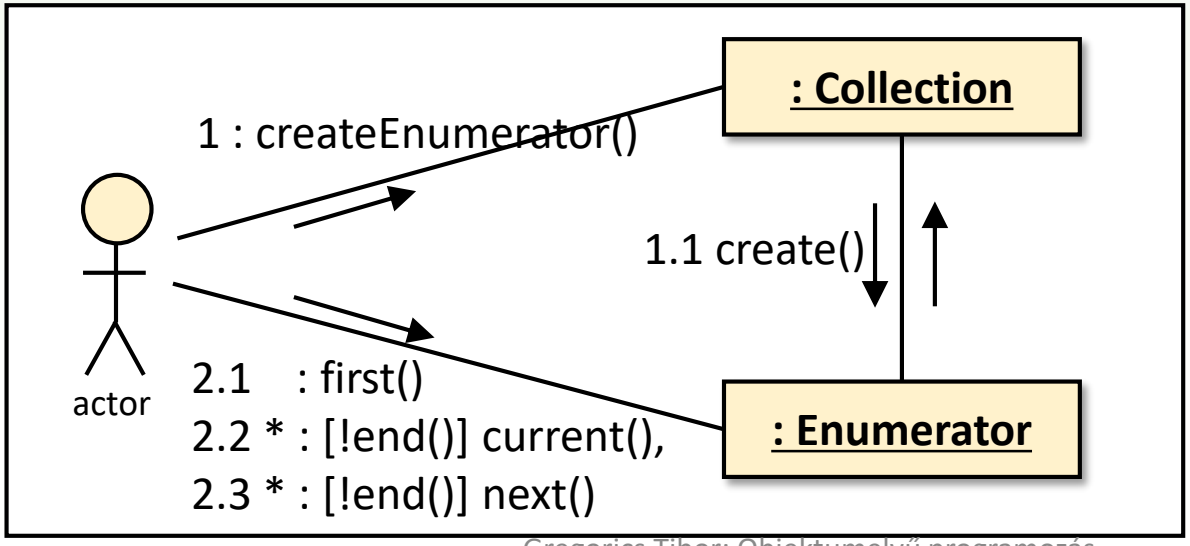
- A kommunikációs diagram azt mutatja meg, hogy az objektumok milyen üzenetekkel (metódus-hívások, szignál-küldések) kommunikálnak egymással.
- Lehetőséget ad az üzenetek sorrendjének kijelölésére (sorszámozással), illetve előfeltételek megadására szögletes zárójelpár között.

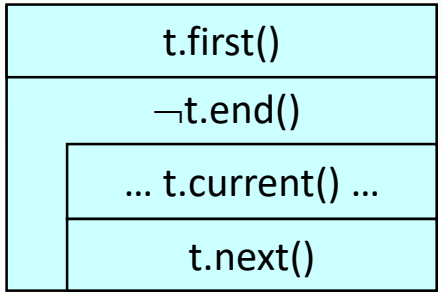


Példa: Felsorolás



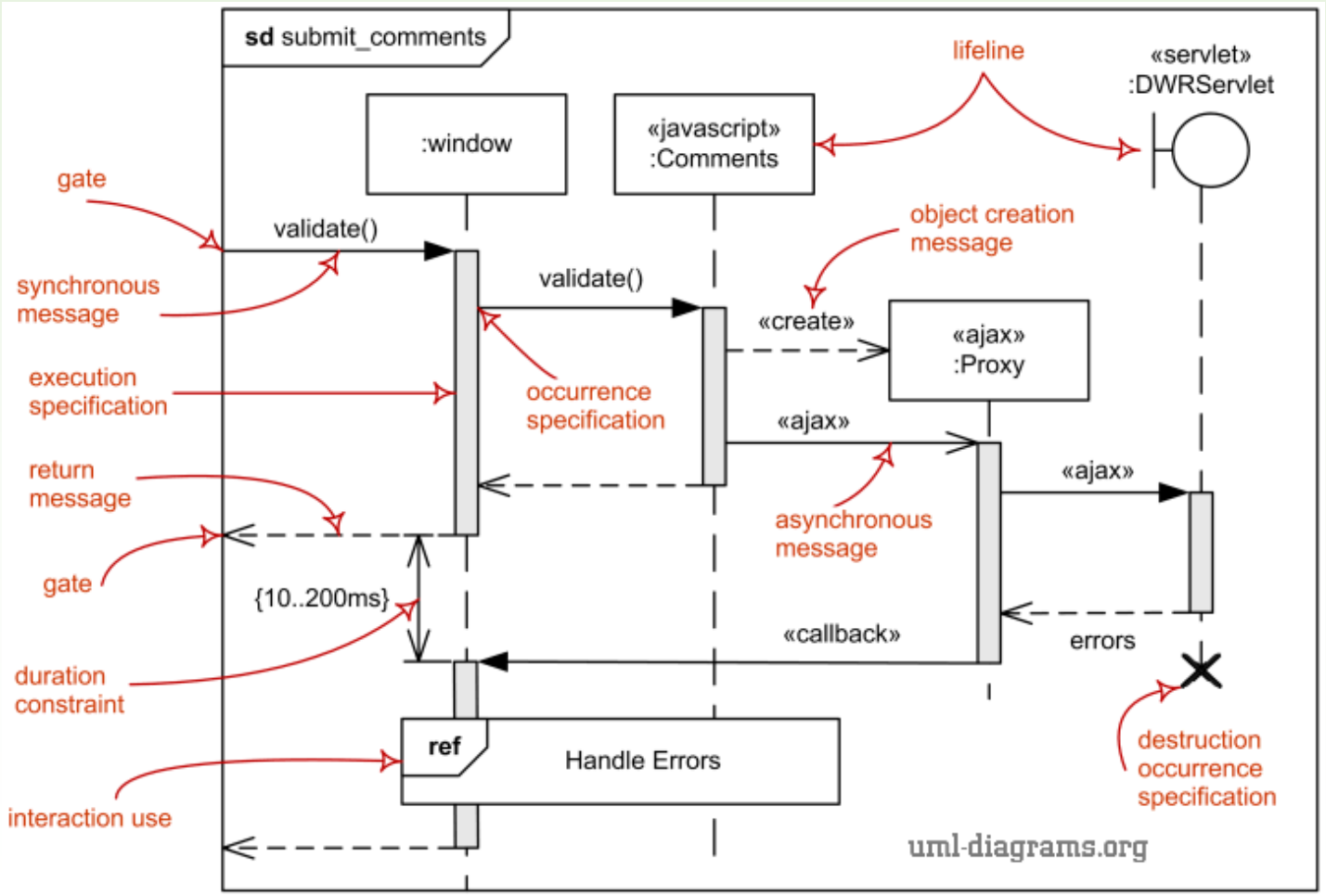






Szekvencia diagram

- A kommunikációban az üzenetváltások időbeli sorrendjét mutatja.

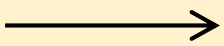
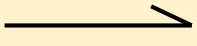


Szekvencia diagram elemei

- objektum életvonala (az objektumból lefelé induló függőleges szaggatott vonal) : az üzenetváltásokban résztvevő objektum élettartamának egy részét jelző idővonal. (Az objektum példányosítása illetve megszüntetése nem feltétlenül része a diagramnak.)
- tevékenység (hosszú elnyújtott téglalap az életvonalon): egy objektum aktivitásának időbeli elnyúlását jelzi az életvonalon
- üzenet (vízszintes nyíl életvonalak között) : egy objektum tevékenysége küldi egy másik objektumnak.
- üzenetek sorrendje: az üzenetek egymás alatti megjelenése időbeli sorrendre utal, de az eltelt időt külön jelzés hiányában nem definiálja.
- fragmentek (dobozba zárt részek): olyan részleteit jelölik ki a diagramnak, mint amely például ciklikusan ismétlődhet (loop), vagy alternatívaként következhet be (alt), vagy máshol van részletesebben kifejtve (ref).

Üzenetek fajtái

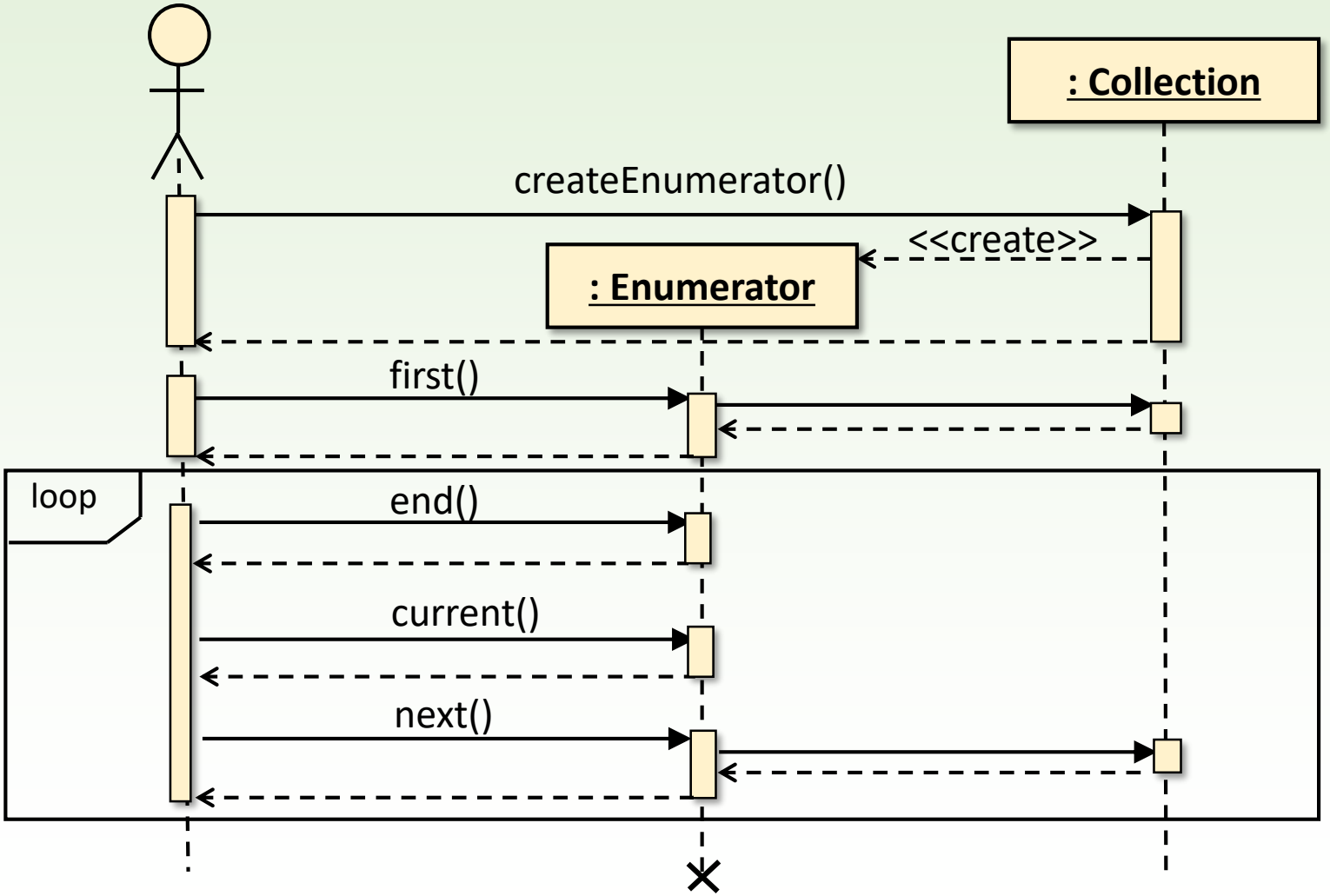
- Szinkron üzenet 
  amikor a küldő objektum átadja a vezérlést a fogadó objektumnak, és a saját tevékenységét blokkolja mindaddig, amíg a fogadó objektum ezt nem oldja fel.
  - fogadó objektum egy metódusának hívása.
  - egy időhöz kötött várakozó üzenet, amely során a küldő objektum megjelölt ideig várakozik arra, hogy a fogadó objektum fogadja az üzenetet.

- Aszinkron üzenet  régebben: 
  a párhuzamosan működő objektumok között fordul elő, amikor a küldő objektum nem szakítja meg a tevékenységét az üzenet elküldése után. Gyakran egy jel (szignál) elküldése (paraméter-értékekkel kiegészítve).

- Speciális üzenetek: 
  - visszatérési üzenet: korábban üzenetet kapó objektum küldi vissza a küldő objektumnak (visszaadja a vezérlést vagy hibát jelez).
  - példányosító (create) üzenet: létrehoz egy objektumot
  - randevú üzenet: a fogadó objektum várakozik a küldő objektum üzenetére.

t.first()
¬t.end()
... t.current() ...
t.next()

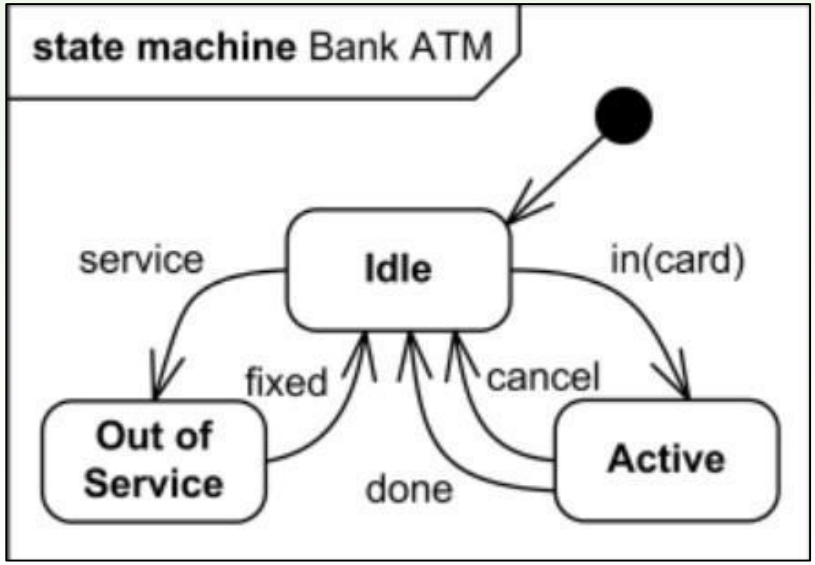
Példa: Felsorolás



Állapotgép

- Az állapotgép diagram egy objektum életciklusát ábrázolja. Megmutatja, hogyan változik az objektum belső (logikai) állapota az objektumnak küldött üzenetek (metódus hívások vagy szignálok) hatására.

- Az állapotgép egy irányított gráf, amelynek csomópontjai a logikai állapotokat, irányított élei pedig az állapotok közötti átmeneteket mutatják.
- Mind az állapotokhoz, mind az átmenetekhez tartozhatnak végrehajtandó tevékenységek.

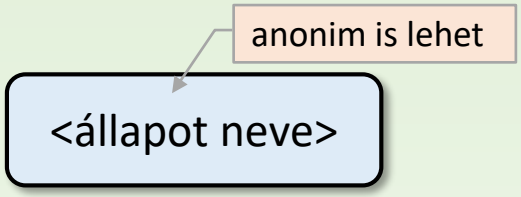


Objektum életciklusa

- Az objektum életciklusa során:
  - létrejön: az objektum speciális műveletével, a konstruktorral,
  - működik: más objektumokkal kommunikál, azaz szinkron vagy aszinkron módon hívják egymás műveleteit, vagy az egymásnak küldött jelzésekre (*signal*) reagálnak aszinkron módon, és ennek során megváltozhatnak az adatai,
  - megsemmisül (egy másik speciális művelettel, a destruktorral).
- Egy objektumnak különféle állapotai (*state*) vannak: egy állapot (fizikai állapot) az objektum adatai által felvett értékek együttese, amely az életciklus során változik.
- A könnyebb áttekinthetőség kedvéért azonban gyakran egy állapotnak az objektum több különböző, de közös tulajdonságú fizikai állapotainak összességét tekintjük (logikai állapot) .

Állapotok

- Az állapotokat lekerekített sarkú téglalap jelöli:



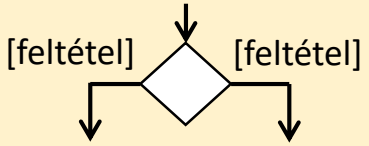
Pszeudo állapotok:

- kezdő állapot
- végállapot

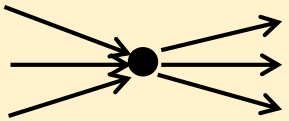
Hierarchikus állapotgépekben:

- belépés (entry)
- kilépés (exit)
- megszüntetés
- shallow history
- deep history

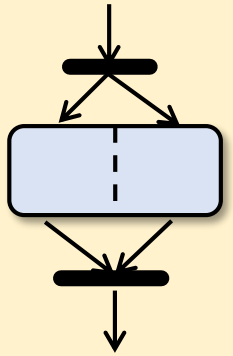
- elágazás (choice)



- csomópont (junction)

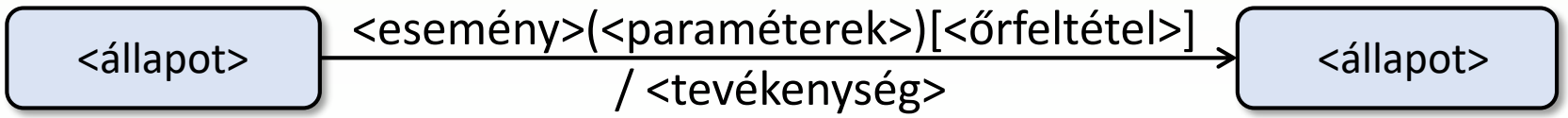


- szétágazás (fork)
- összefutás (join)



Állapot-átmenetek jelölése

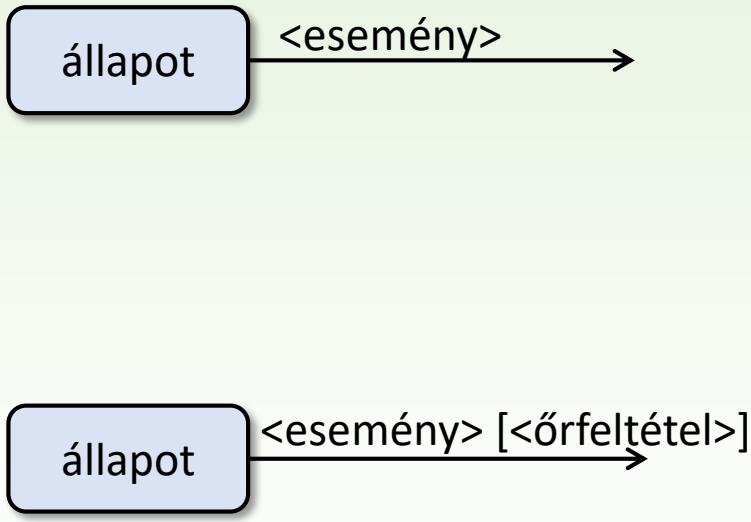
- Az állapot-átmenet jellemzői (bármelyik hiányozhat):
  - átmenetet előidéző esemény (*event, trigger*) a paramétereivel
    - lehet az adott objektum egy metódusának a hívása
    - vagy az objektumnak küldött (aszinkron feldolgozott) szignál
  - átmenetet megengedő őrfeltétel (*guard*), amely
    - vagy az esemény paramétereitől függő logikai állítás (*when*)
    - vagy egy időhöz kötött várakozási feltétel (*after*)
  - átmenethez rendelt tevékenység (az objektum adattagjaival és a kiváltó esemény paramétereivel operáló program)
- Egy átmenet lehet reflexív.



Állapot-átmenetek szemantikája I.

Amikor az átmenetnek van kiváltó eseménye:

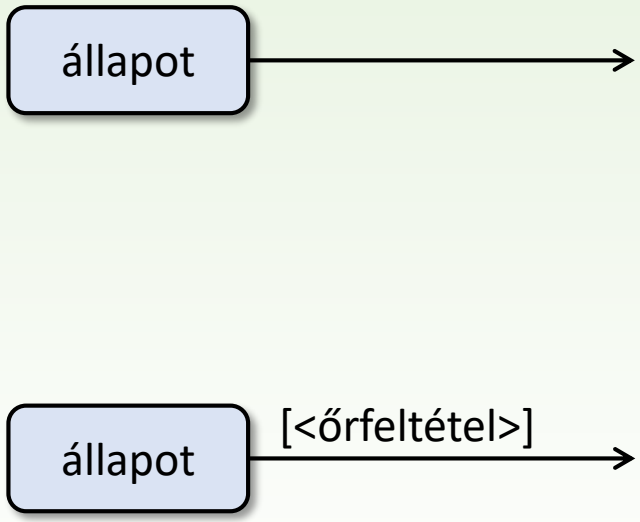
- Őrfeltétel hiányában, ha az esemény bekövetkezik, akkor az állapot belső tevékenysége, ha van, megszakad, és az átmenet megvalósul. Ugyanazon állapotból nem vezethet ki két él ugyanazon eseménnyel.
- Őrfeltétel mellett az átmenet csak akkor valósul meg az esemény bekövetkezésekor, ha az őrfeltétel igaz. Egyébként az esemény hatástalan. (nincs várakozás) Ugyanaz az esemény diszjunkt őrfeltételekkel egy állapotból kivezető több élhez is tartozhat.



Állapot-átmenetek szemantikája II.

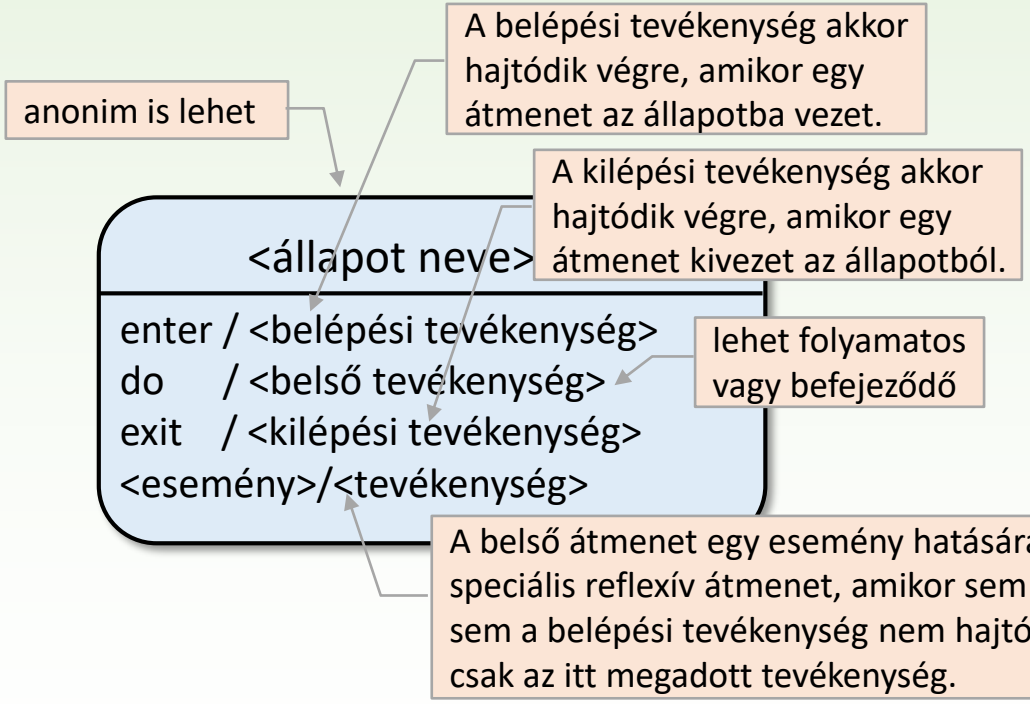
Amikor az átmenetet nem esemény váltja ki:

- Őrfeltétel hiányában az átmenet az állapot belső tevékenységének befejeződésekor valósul meg.
  Ha nincs belső tevékenység, akkor azonnal, ahogy az állapotba jut.
- Őrfeltétel esetén az átmenet bekövetkezése (az esetleges belső tevékenység befejeződése után) addig várakozik, amíg az őrfeltétel igaz lesz, feltéve, hogy várakozás közben egy másik állapot-átmenetre nem kerül sor.



Állapotok jelölése

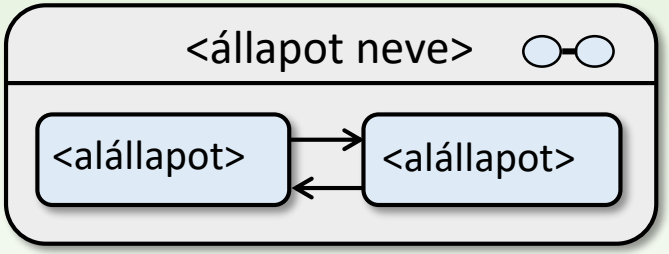
- Egyszerű állapot



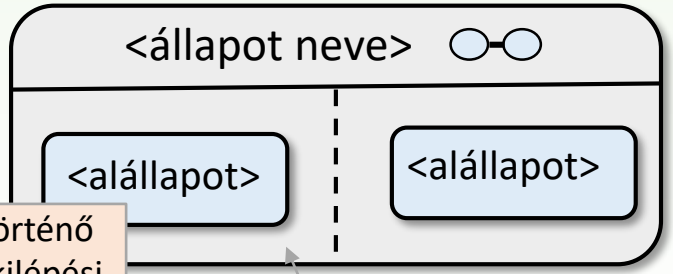
- Hierarchikus állapot

állapotgépet tartalmazó

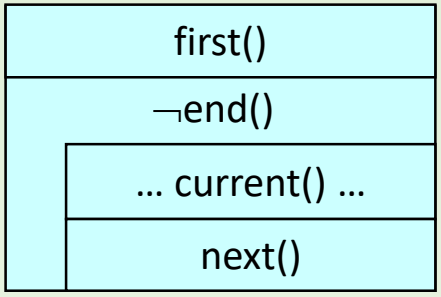
Szekvenciális:



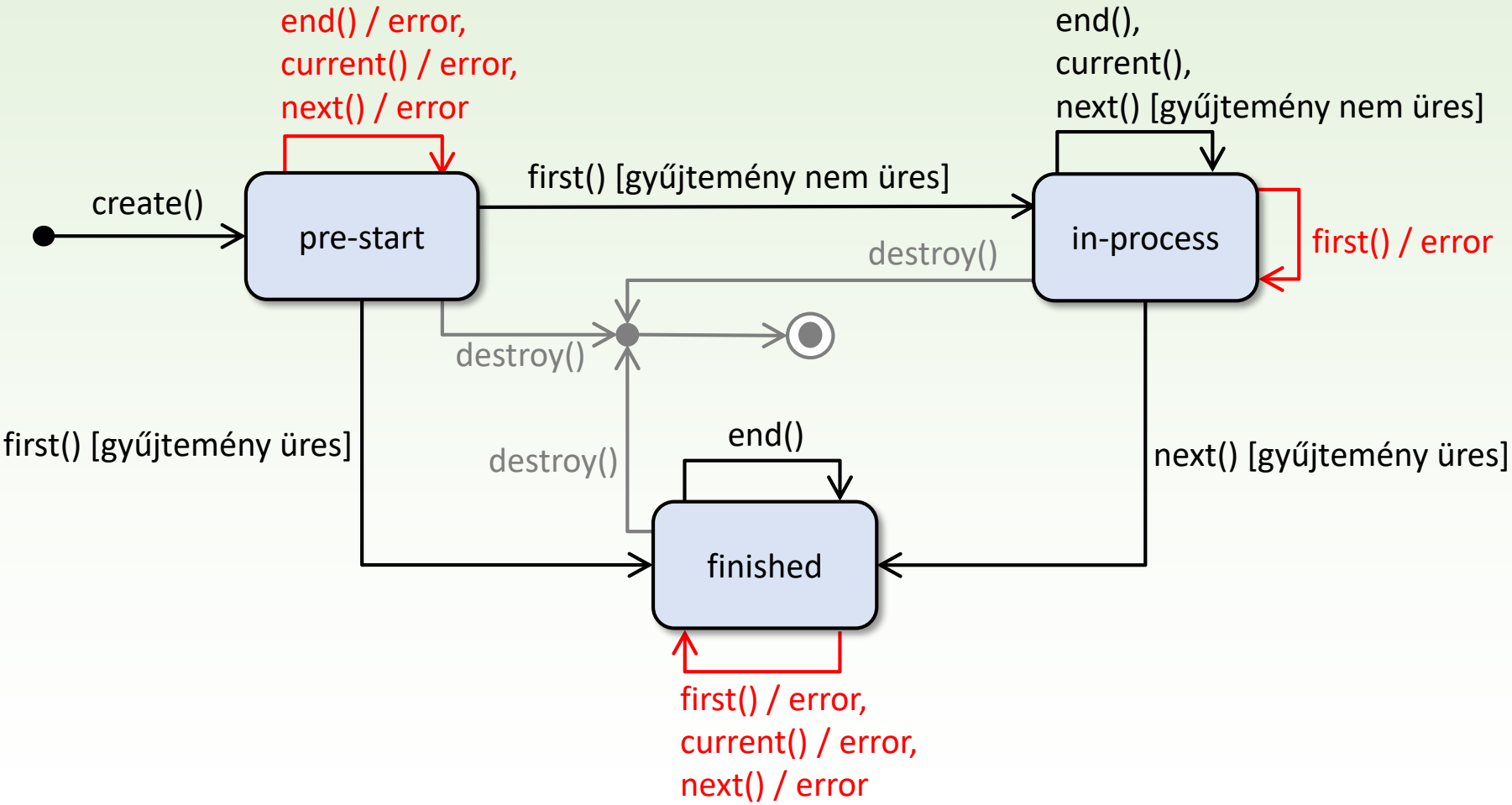
Párhuzamos:



ortogonális régiók



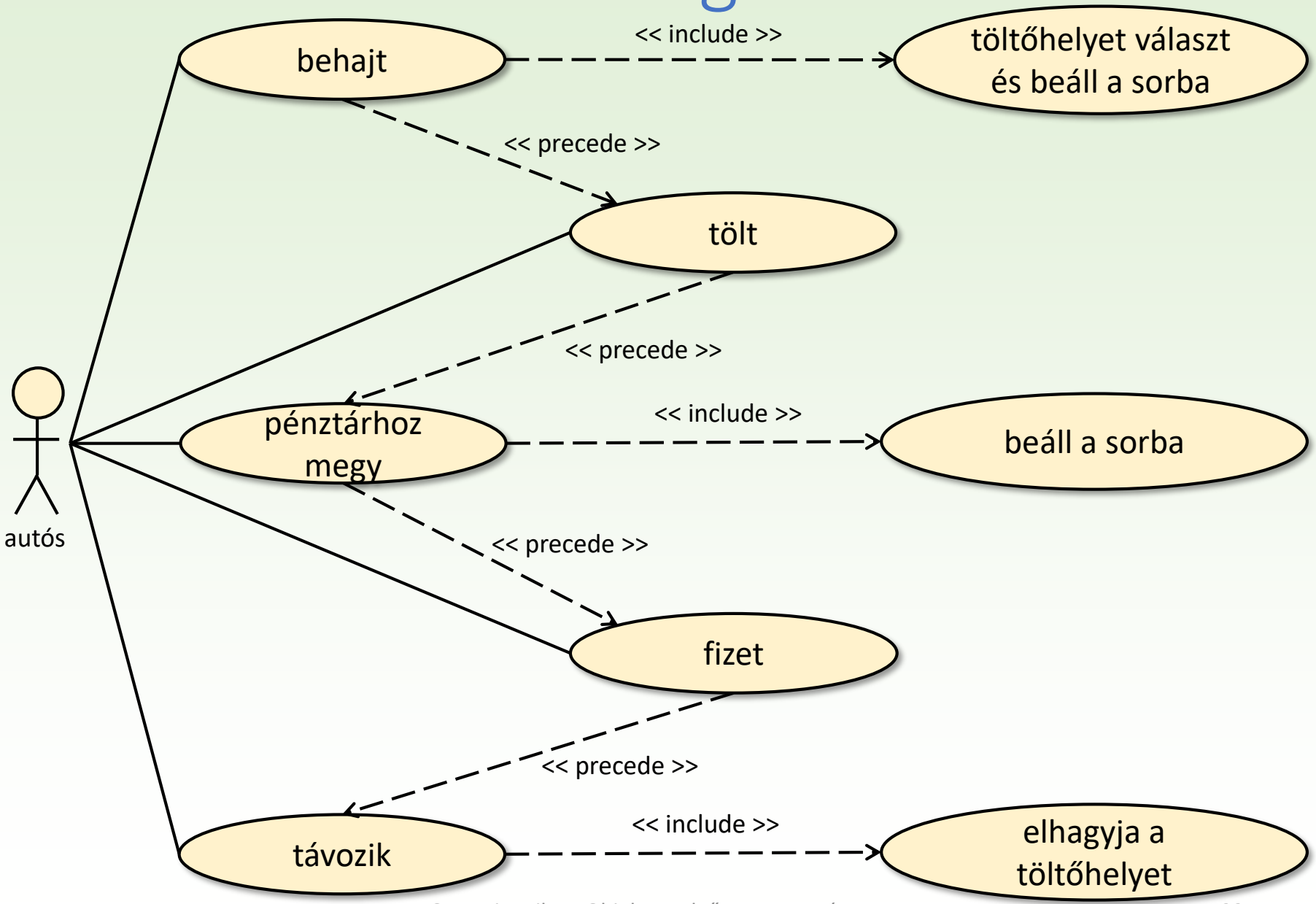
Példa: Felsorolás



Feladat

- Egy benzinkútnál több töltőhely és egy több kasszából álló pénztár működik.
  - Az autósok behajtanak a benzinkúthoz, beállnak valamelyik töltőhelyhez tankolni.
  - Amikor sorra kerülnek, akkor kívánt mennyiségű benzint töltenek a járművük benzintartályába.
  - Ezután elmennek fizetni, és beállnak a pénztárhoz álló sorba.
  - Amint egy kassza szabad lesz, a sorban elől álló autós odalép, ahol kiszámolják a tankolt mennyiség alapján fizetendő összeget.
  - Fizetés után az autós kihajt a töltőhelyről, és távozik.
- Modellezzük ezt a folyamatot tetszőleges számú, egymással párhozamosan tevékenykedő autós esetére.

Használati eset diagram



Felhasználói esetek

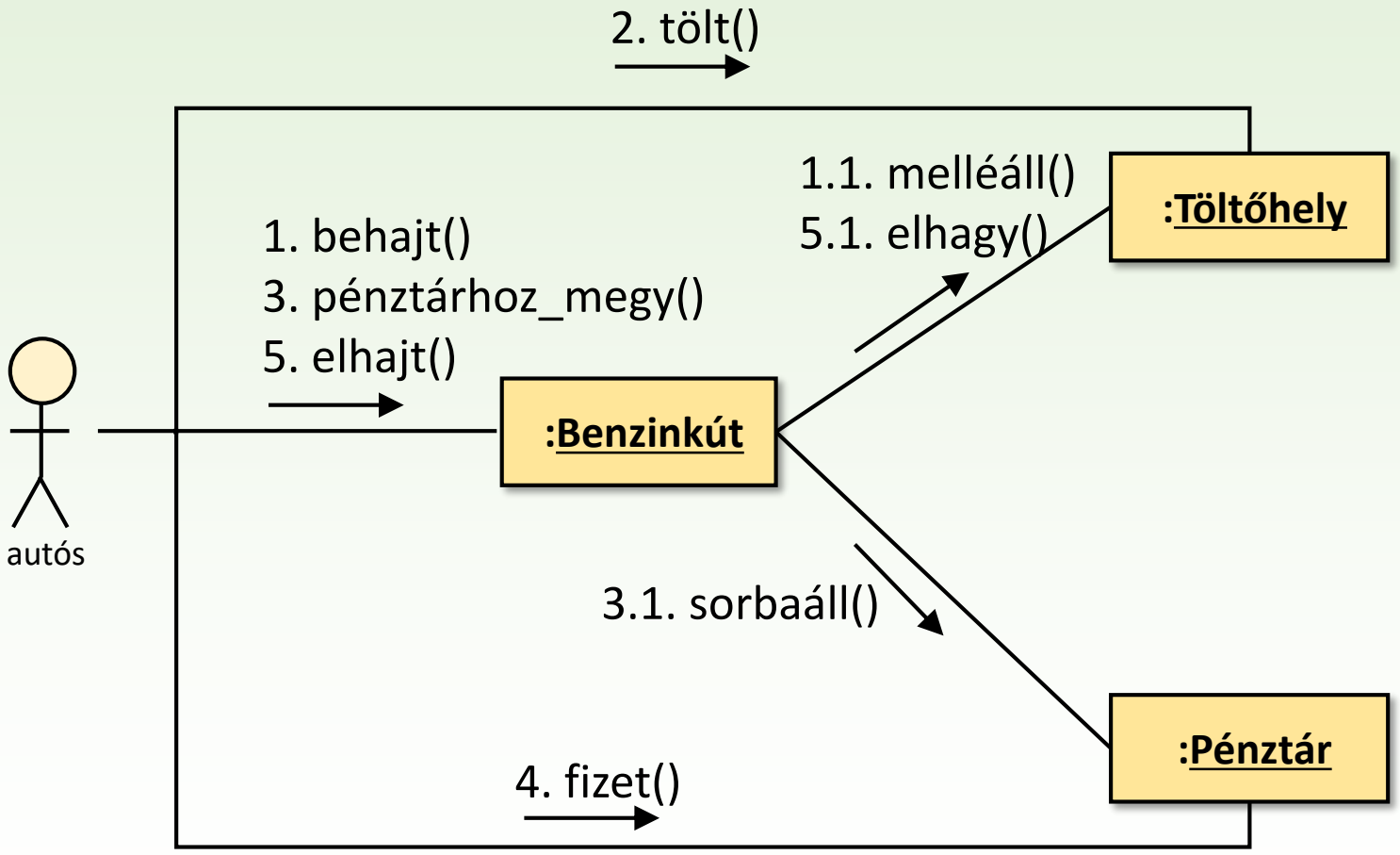
eset		leírás
behajt	GIVEN	létezik a benzinkút töltőhelyekkel
	WHEN	behajt egy létező töltőhelyhez
	THEN	besorol a töltőhely melletti sorba
tölt	GIVEN	az egyik töltőhelyen elsőként áll
	WHEN	megadott liter benzin töltése
	THEN	a kijelző mutatja a felvett benzint
pénztárhoz megy	GIVEN	létezik a benzinkút pénztárral
	WHEN	bemegy a pénztárba
	THEN	beáll a pénztár sorába
fizet	GIVEN	létezik a benzinkút pénztárral, egy töltőhelyen áll, egy kasszánál áll
	WHEN	fizet
	THEN	kiszámoljuk a fizetendő összeget, lenullázzuk a töltőhely kijelzőjét
távozik	GIVEN	a benzinkút egyik töltőhelyén áll
	WHEN	elhajt
	THEN	kiáll a töltőhely sorából

eset		leírás
tölt	GIVEN	az egyik töltőhelyen elsőként áll
	WHEN	nulla liter benzin töltése
	THEN	figyelmeztetés
távozik	GIVEN	töltőhelyen áll, a kijelző nem nulla
	WHEN	elhajt
	THEN	riasztás

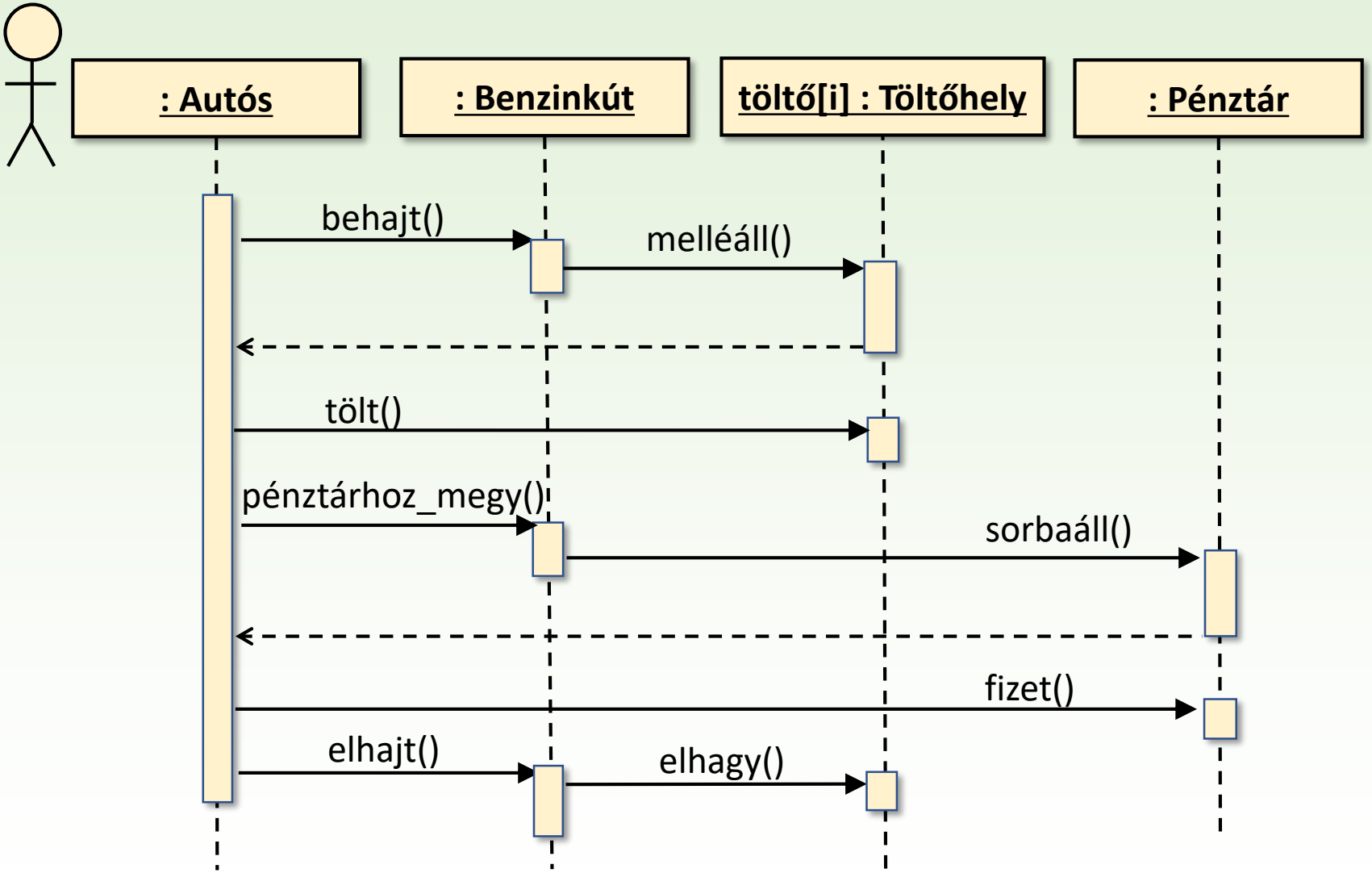
Felhasználói esetek

eset		leírás
behajt	GIVEN	nem létezik a benzinkút vagy nem létezik a kiválasztott töltőhely
	WHEN	behajt
	THEN	hibajelzés
tölt	GIVEN	nem áll töltőhelyen
	WHEN	benzint tölt
	THEN	hibajelzés
pénztárhoz megy	GIVEN	létezik a benzinkút, de nem létezik pénztár
	WHEN	bemegy a pénztárba
	THEN	hibajelzés
fizet	GIVEN	nem áll töltőhelynél
	WHEN	fizet
	THEN	hibajelzés
távozik	GIVEN	nem áll töltőhelynél
	WHEN	elhajt
	THEN	hibajelzés

Kommunikációs diagram



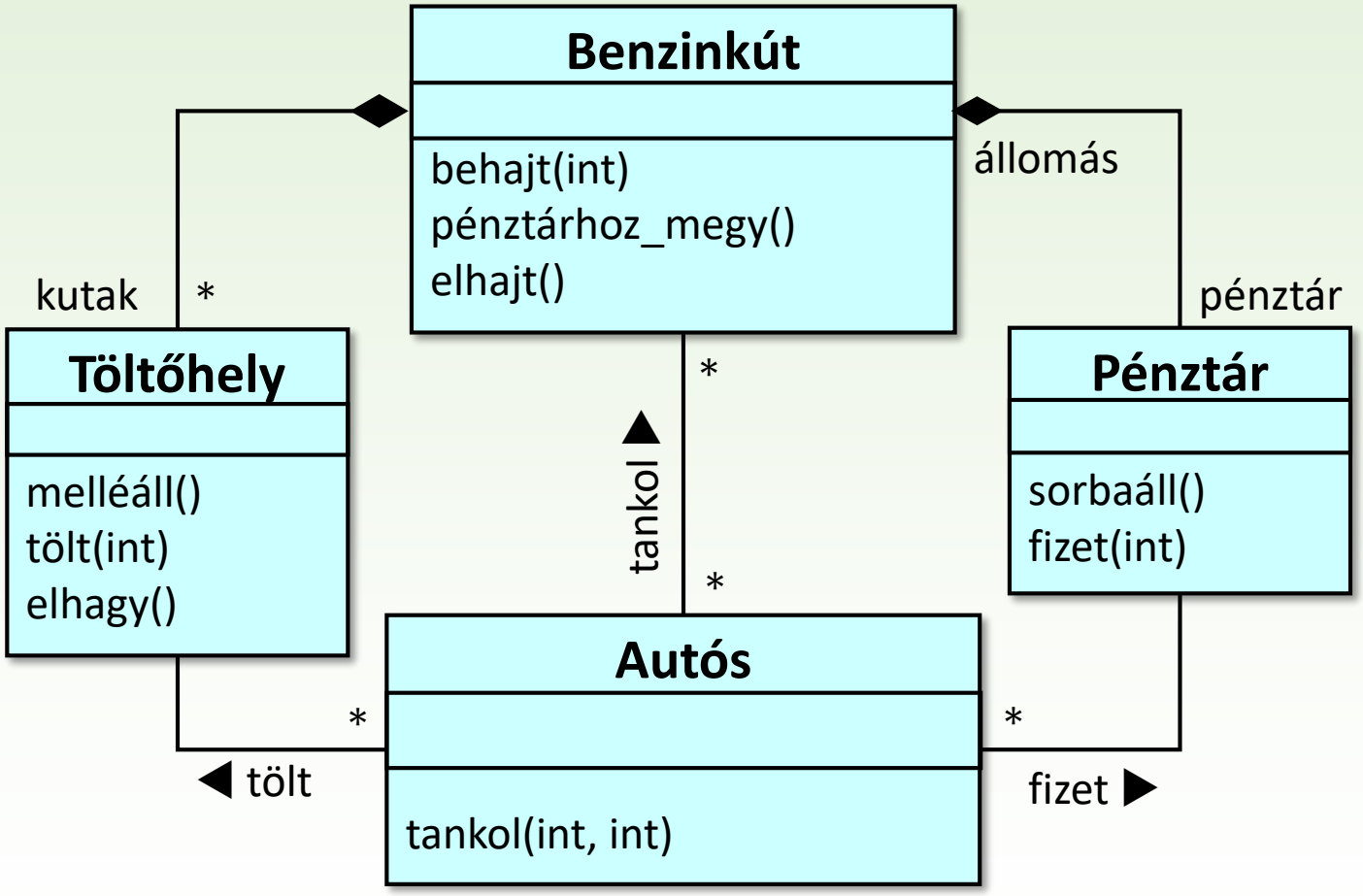
Szekvencia diagram



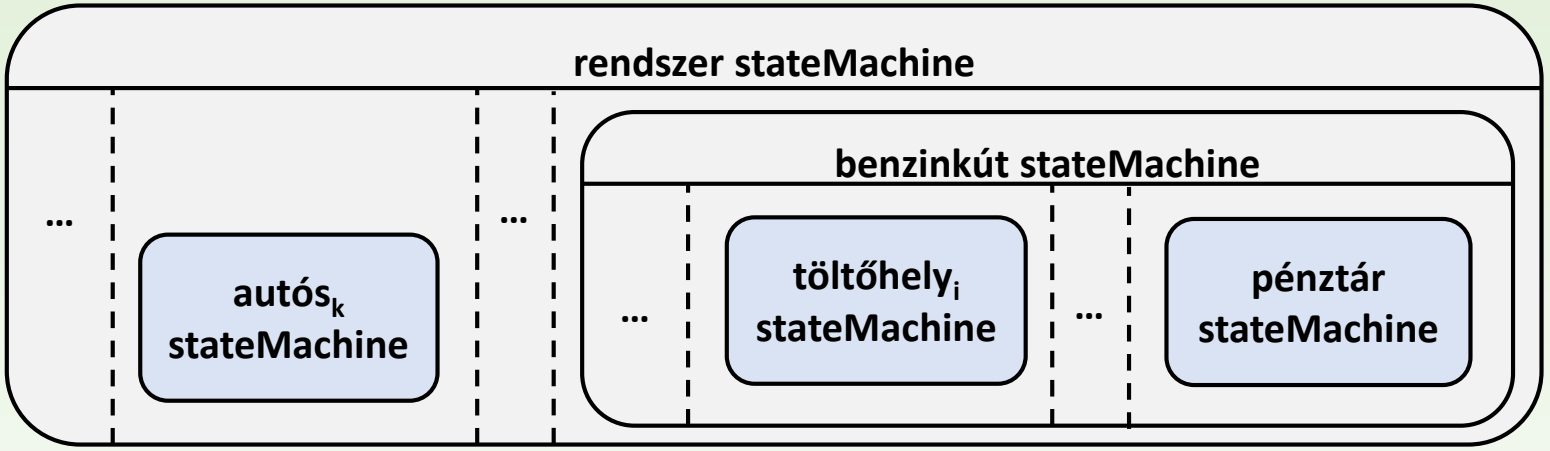
Elemzés eredménye

- Objektumok és tevékenységeik:
  - autósok (tankolnak)
  - benzinkút (ahová az autósok behajtanak, ahol tankolnak, pénztárhoz mennek, fizetnek, ahonnan elhajtanak)
  - töltőhelyek (amely mellé beáll az autós, ahol várakozik, majd benzint tölt, végül fizetés után kiáll)
  - pénztár több kasszával (ahol az autós sorba áll, ahol várakozik, majd fizet)
- Objektumok közötti kapcsolatok:
  - a benzinkút részei a töltőhelyek és a pénztár
  - egy autós ideiglenesen kapcsolatba kerül egy töltőhellyel és a pénztárral.

Osztály diagram



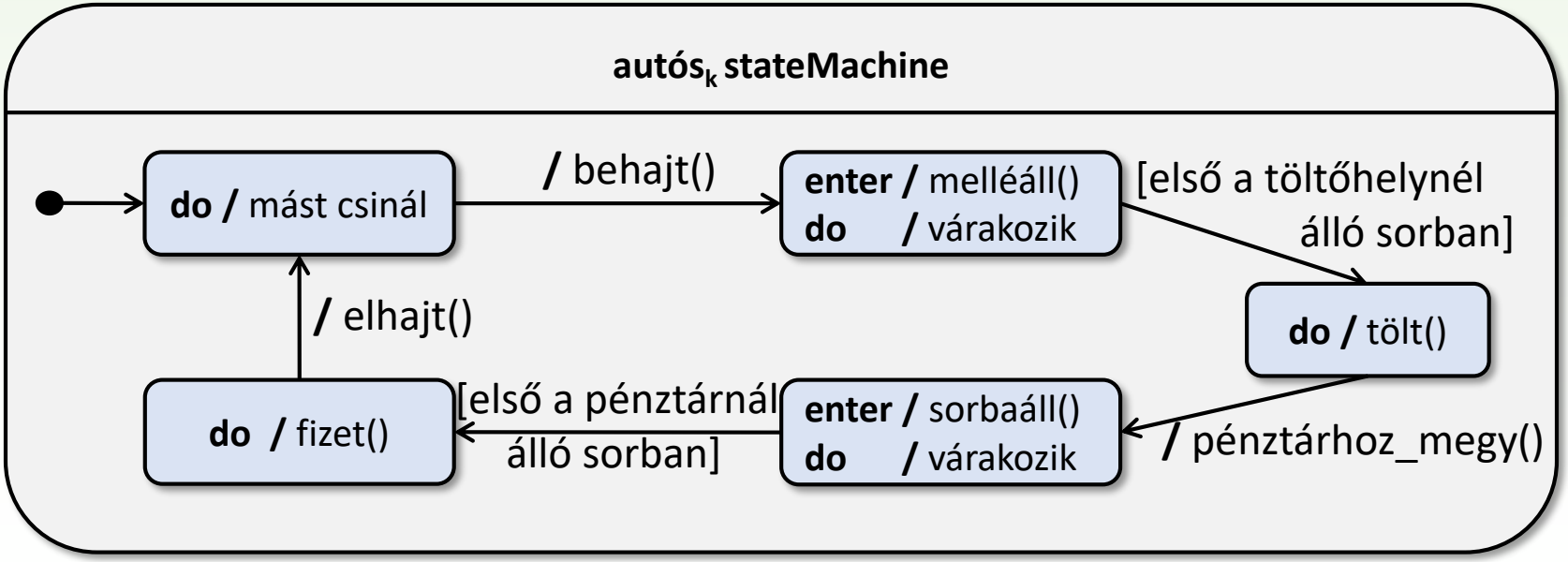
Rendszer állapotgépe



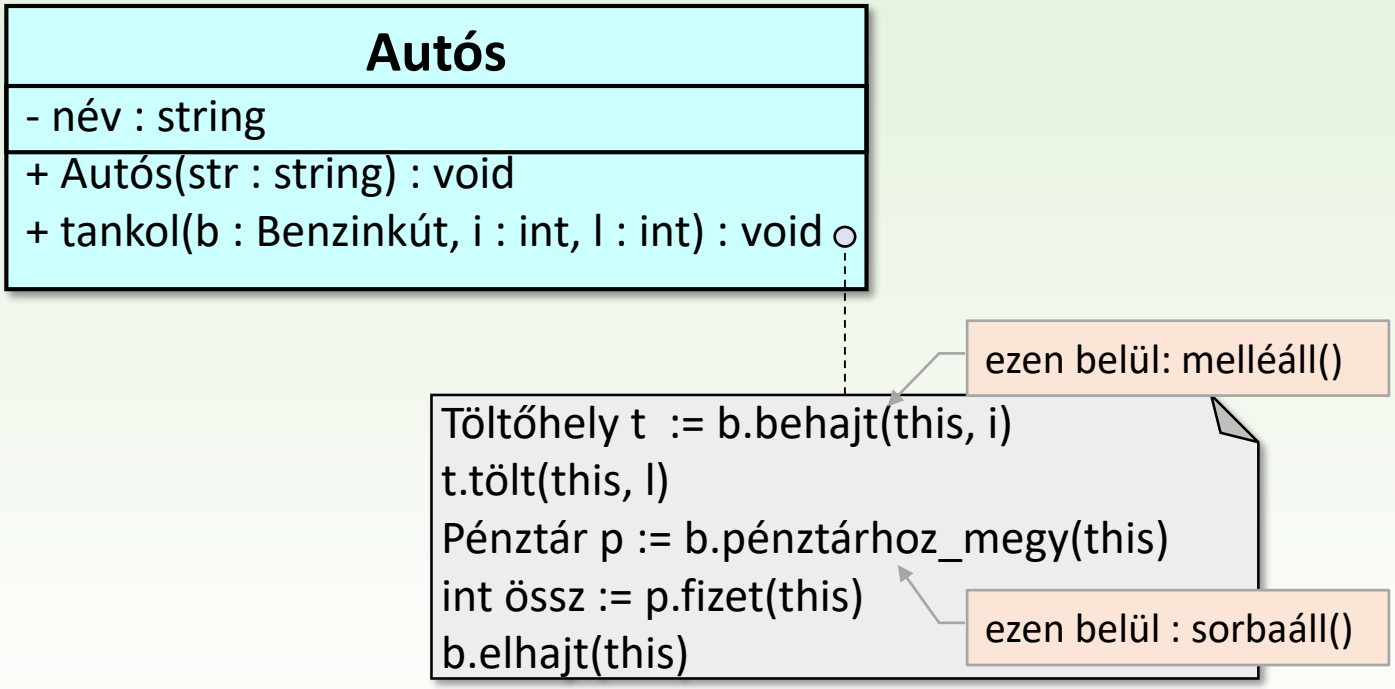
- A rendszer állapotát az autósok és a benzinkút állapota határozza meg. A benzinkút állapota a töltőhelyek és a pénztár állapotától függ.
- Az autósok ún. aktív objektumok: párhuzamosan végeznek tevékenységet, így állapotgépeik külön szálakon futnak majd.
- A benzinkút passzív objektum: állapotgépe más objektumok állapotgépével szinkron módon (metódusainak hívása által) működik. Nem igényel külön szálat.

Autósok állapotgépe

- Egy autós ötféle állapotban lehet, amelyek akár ciklikusan is változhatnak a benzinkút metódusainak hatására:
  - mást csinál, behajt a benzinkút egy kútjához és sorba áll, üzemanyagot vesz fel, pénztárban sorba áll, fizet és elhajt
- Az autósok ún. aktív objektumok (tevékenységüket önmaguk szabályozzák), és egymással párhuzamosan működnek.



Autós osztály



Autós osztály

megvárja, míg a külön indított szál befejeződik

car.h

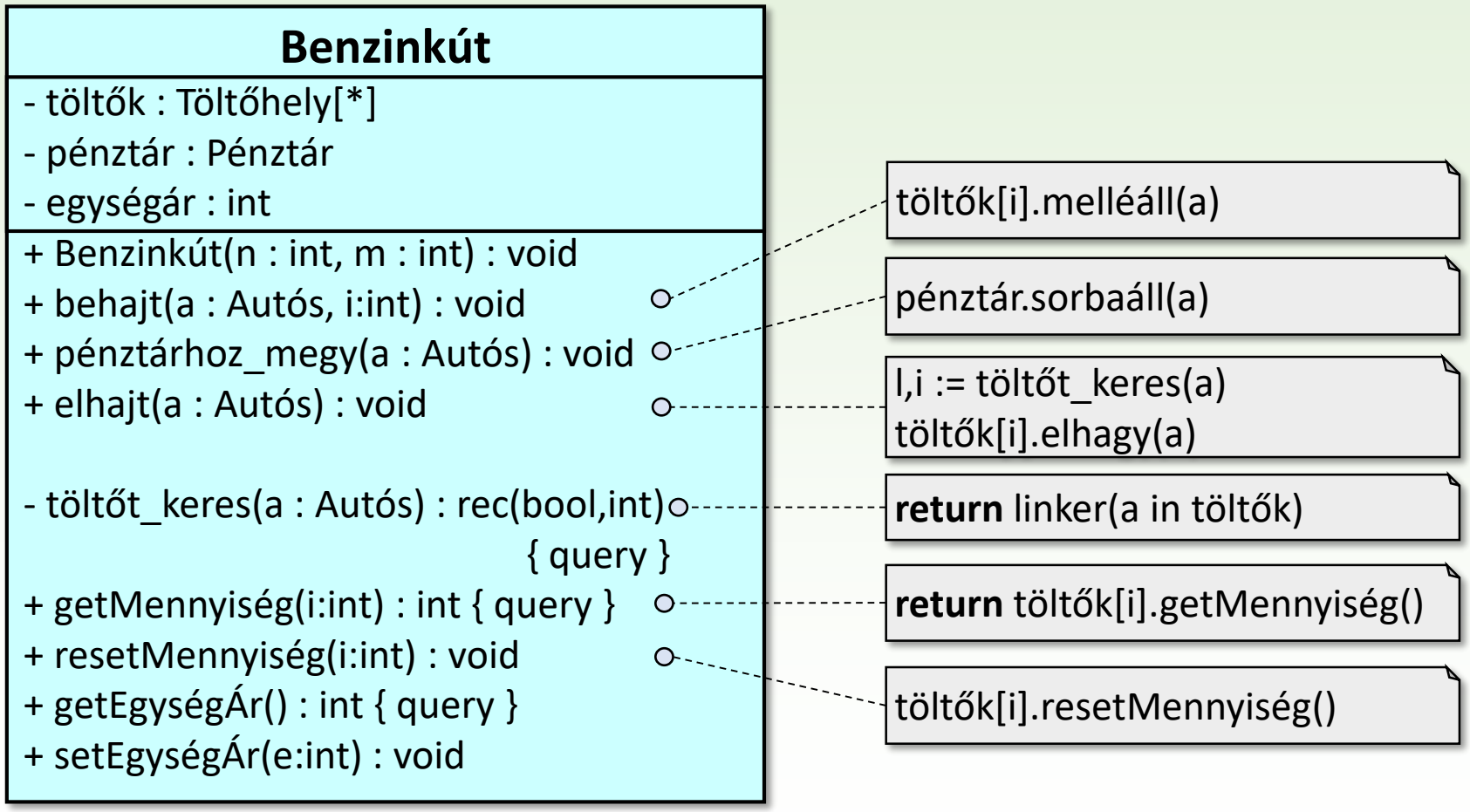
```
class Car {
public:
        Car(const std::string &str) : _name(str) {}
        ~Car() { _fuel.join(); }
        …
        void refuel(PetrolStation* petrol, unsigned int i, int l) {
                _fuel = new std::thread(activity, this, petrol, i, l);
        }
private:
        std::string _name;
        std::thread _fuel;
        void activity(PetrolStation* petrol, unsigned int i, int l);
};
```

külön szálon indul el
#include <thread>

```
void Car::activity(PetrolStation* petrol, unsigned int i, int l)
{
        if ( nullptr==petrol ) return;                  // ha nincs benzinkút
        Pump *pump = petrol ->driveIn(this, i);// behajt, és beáll az i-dik töltőhöz
        if ( nullptr==pump ) return;                    // ha nincs i-dik töltőhely

        pump->fill(this, l);                            // tankol l liter benzint

        Cash *cash = petrol ->goToCash(this);   // pénztárhoz megy
        if ( nullptr==cash ) return;                    // ha nincs pénztár

        int n = cash->pay(this);                        // fizet
        petrol ->driveOff(this);                        // távozik a benzinkúttól
}
```

car.cpp

Benzinkút osztálya



Benzinkút osztálya

```
class PetrolStation {
public:
        PetrolStation(int n, int m) {
                for(int i = 0; i<n; ++i) _pumps.push_back( new Pump() );
                 _cash = new Cash(this, m);
        }
        ~PetrolStation() { for( Pump *p : _pumps ) delete p;  delete _cash; }

        bool driveIn(Car* car, unsigned int i);
        void goToCash(Car* car);
        bool driveOff(Car* car);

        int getUnit() const { return _unit; }
        void setUnit(int u) { _unit = u; }
        void resetQuantity(unsigned int i) { _pumps[i]->resetQuantity(); }
        int getQuantity(unsigned int i) const { return _pumps[i]->getQuantity(); }
private:
        std::vector<Pump*> _pumps;
        Cash *_cash;
        int _unit;

        bool pump_search(Car* car, unsigned int &ind) const;
};
```

petrol.h

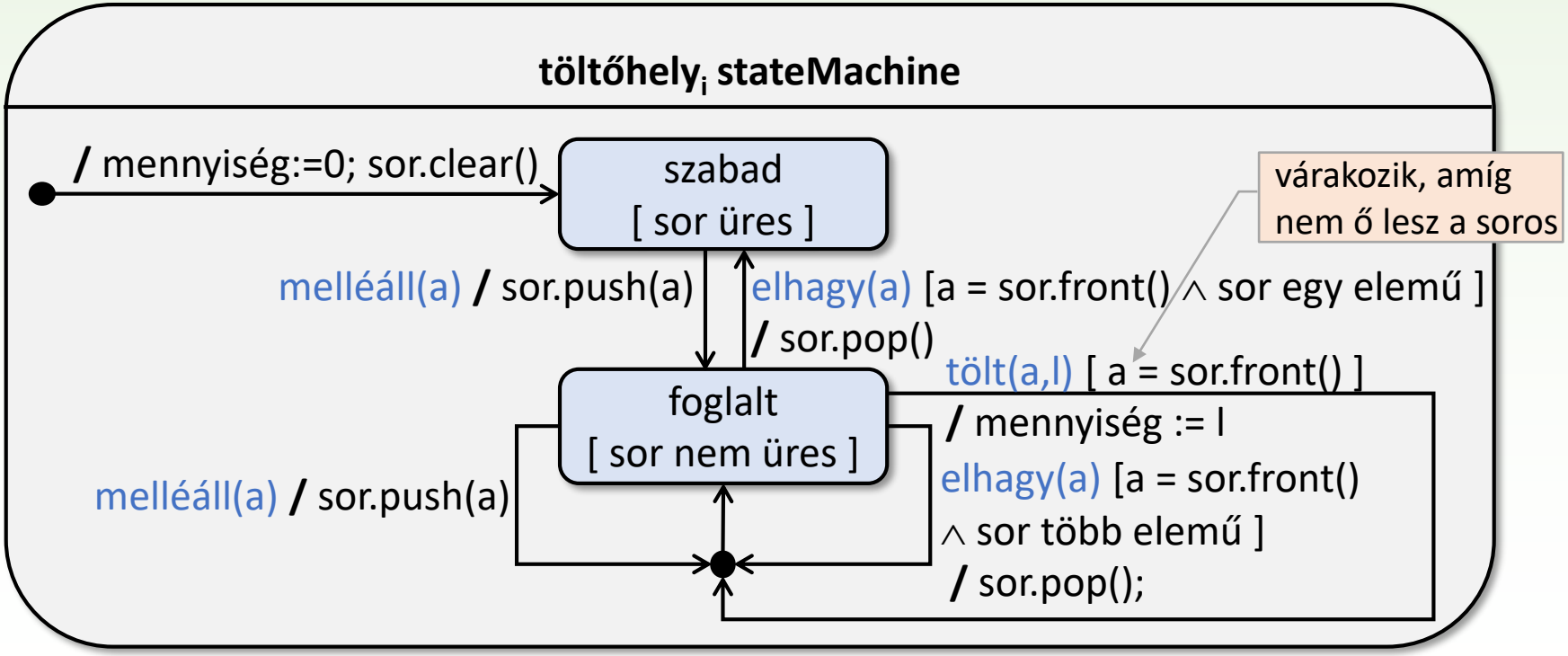
Benzinkút metódusai

```
Pump* PetrolStation::driveIn(Car* car, unsigned int i){
        if ( i>=_pumps.size() ) return nullptr;
        _pumps[i]->standNextTo(car);
        return _pumps[i];
}
Cash* PetrolStation::goToCash(Car* car){
        if (nullptr==_cash ) return nullptr;
        _cash->joinQueue(car);
        return _cash;
}
bool PetrolStation::driveOff(Car* car){
        unsigned int i;
        if ( !search(car, i) ) return false;
        _pumps[i]->leave();
        return true;
}
bool PetrolStation::pump_search(Car* car, unsigned int &i) const {
        bool l = false;
        for ( i = 0; i<_pumps.size(); ++i) {
                if ( (l=_pumps[i]->getCurrent() == car) ) break;
        }
        return l;
}
```

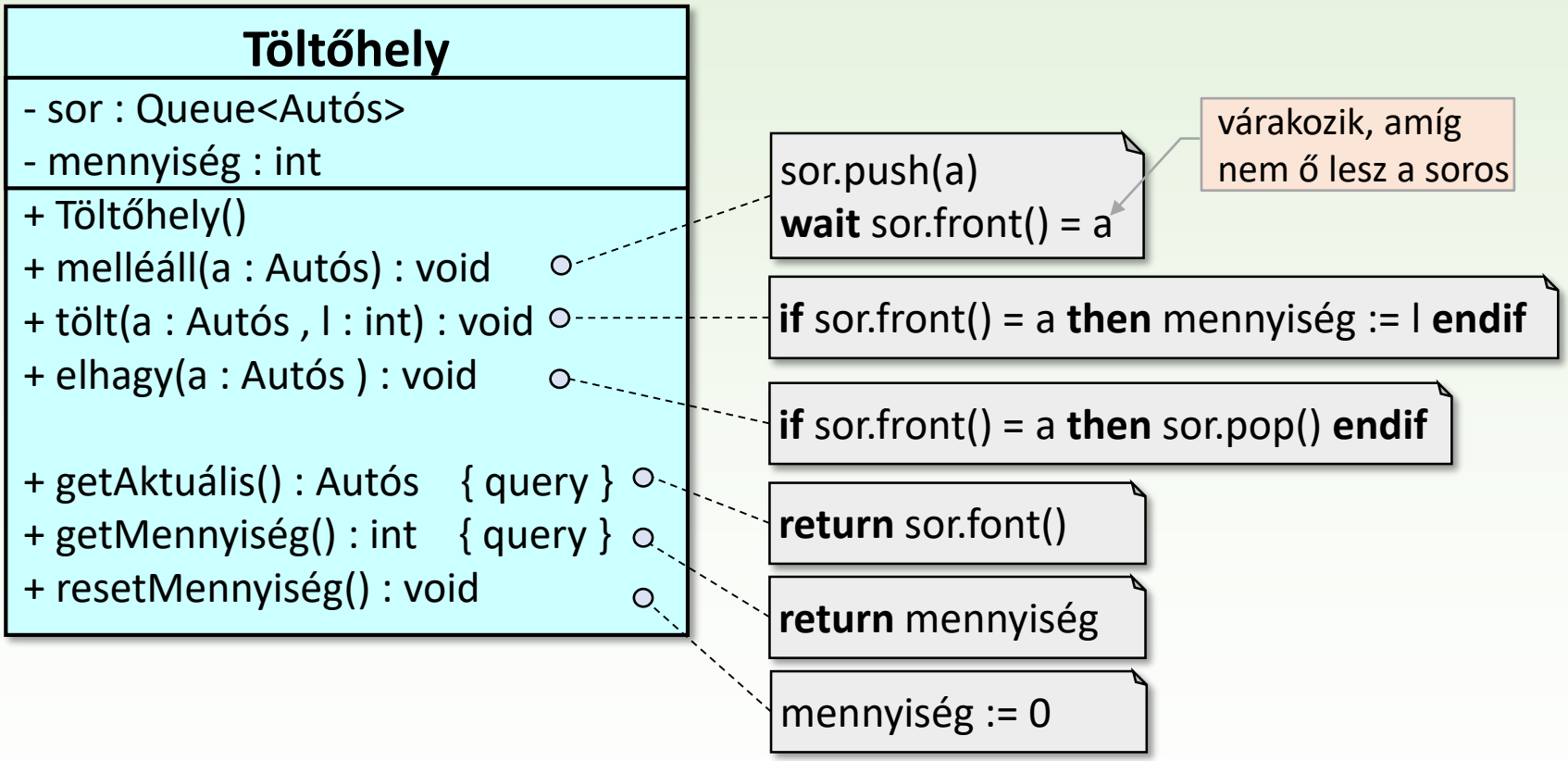
petrol.cpp

Töltőhely állapotgépe

- Egy töltőhely lehet szabad vagy foglalt.
- A melléáll() és elhagy() a töltőhelynél álló sorra van hatással.
- A tölt() csak a foglalt állapotban hajtható végre, amikor az autós az első a sorban. Ekkor megadott mennyiségű benzint tölt az autójába, amely a töltőhely kijelzőjén is megjelenik.



Töltőhely osztálya



Töltőhely osztálya

```
class Car;

class Pump {
public:
	Pump() : _quantity(0) { }

	void standNextTo(Car * car);
	void fill(Car * car, int l);
	void leave(Car * car);

	Car* getCurrent() const { return _queue.front(); }
	int getQuantity()  const { return _quantity; }
	void resetQuantity()      { _quantity = 0; }
private:
	int _quantity;
	std::queue<Car*> _queue;

	std::mutex _mu;
	std::condition_variable _cond;
};
```

A különböző szálakon fut az autósok tevékenysége, amelyet szinkronizálni kell

```
#include  <mutex>
#include <condition>
```

pump.h

Töltőhely metódusai

```
void Pump::standNextTo(Car* car)
{
      std::unique_lock<std::mutex> lock(_mu);
      _queue.push(car);
      while ( _queue.front() != car ) _cond.wait(lock);
}

void Pump::fill(Car * car, int l)
{
      std::unique_lock<std::mutex> lock(_mu);
      if ( _queue.front() != car ) return;
      _quantity = l;
}

void Pump::leave(Car * car)
{
      std::unique_lock<std::mutex> lock(_mu);
      if( car ==_queue.front() ) _queue.pop();
      _cond.notify_all();
}
```

egyszerre csak egy autós dolgozhasson a _queue-n

várakozik a szál

egyszerre csak egy autós dolgozhasson a _queue-n

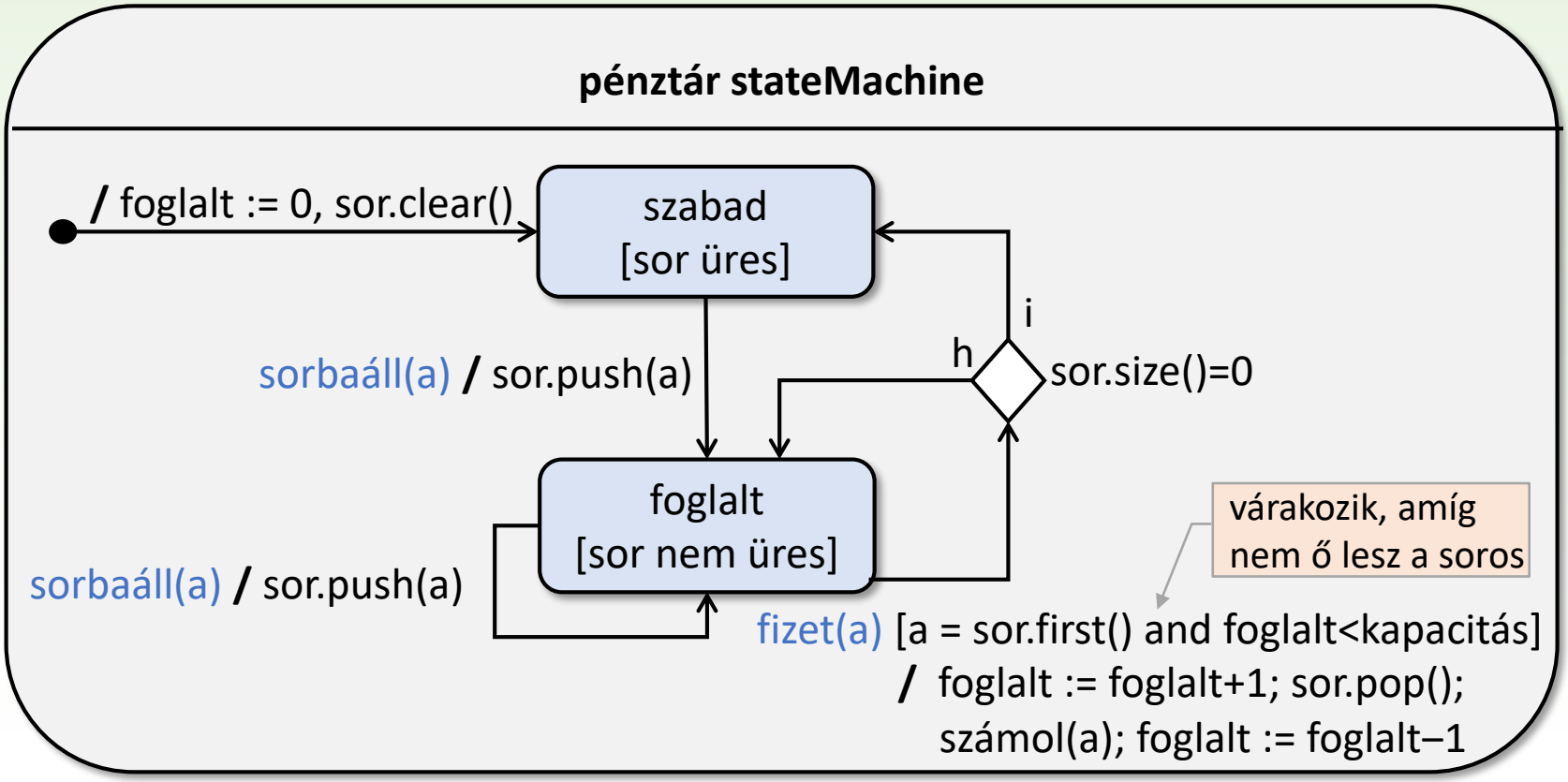
egyszerre csak egy autós dolgozhasson a _queue-n

elindítja az összes cond-nál várakozó szálat

pump.cpp

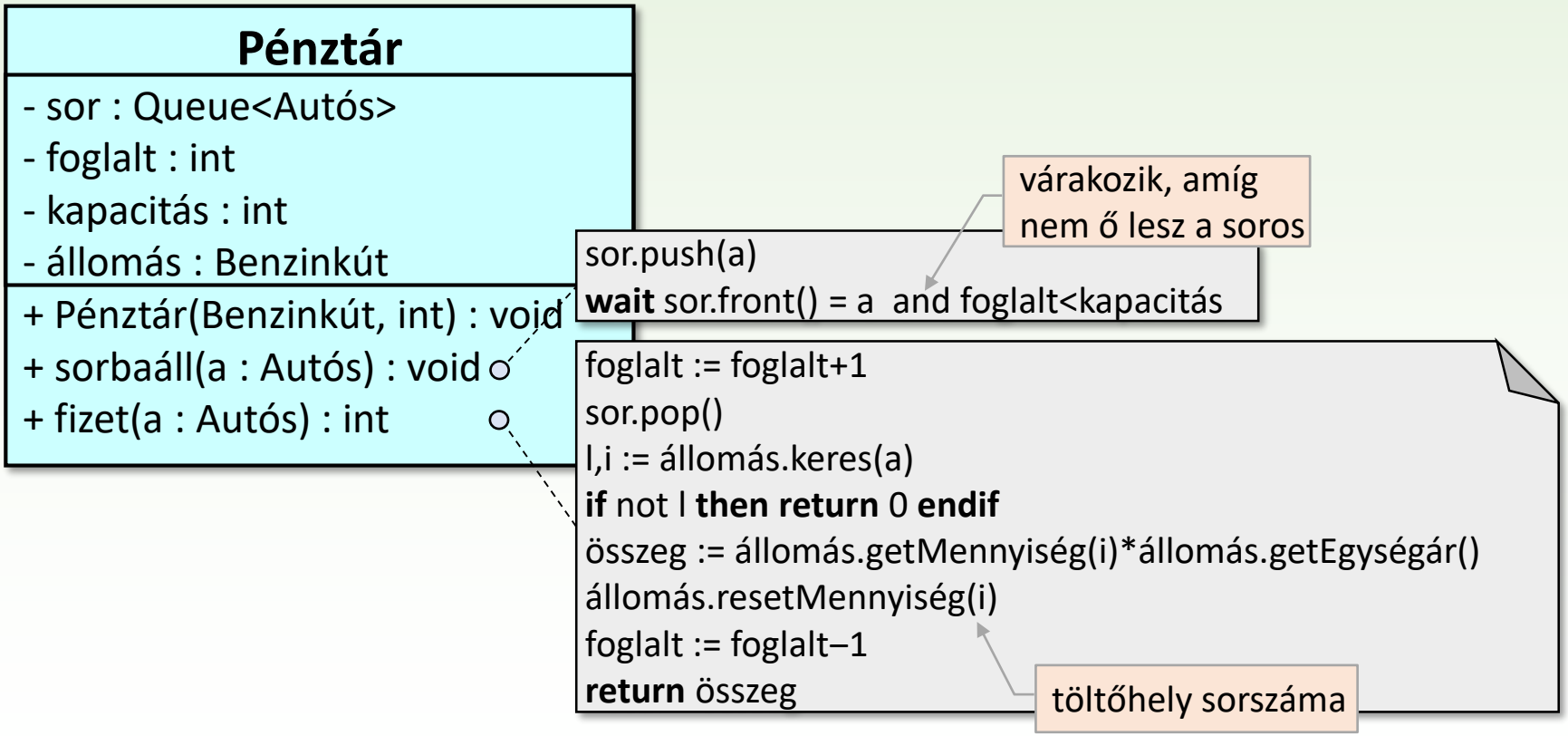
Pénztár állapotgépe

- A pénztárban az autósok sorban állnak.
- Az állapotokra a bemegy() és a fizet() metódusok vannak hatással. Csak a sor elején álló autós fizethet akkor, ha van szabad kassza.



Pénztár

- A pénztár tulajdonságai közé tartozik a kasszák száma (kapacitás), a fogalt kasszák száma (foglalt), és a pénztárban álló sor.



Pénztár osztálya

```
class PetrolStation;
class Car;

class Cash {
public:
	Cash(PetrolStation * ps, int m): _ station(ps), _capacity(m) {}
	void joinQueue(Car *car);
	int pay(Car *car);
private:
	PetrolStation *_station;

	std::atomic_int _engaged;
	int _capacity;
	std::queue<Car*> _cashQueue;

	std::mutex _mu;
	std::condition_variable _cond;
};
```

cash.h

Pénztár metódusai

```
void Cash::joinQueue(Car* car)
{
	std::unique_lock<std::mutex> lock(_mu);
	cashQueue.push(car);
	while ( _cashQueue.front()!=car || _engaged==_capacity ) _cond.wait(lock);
}
int Cash::pay(Car* car)
{
	std::unique_lock<std::mutex> lock(_mu);
	if ( _cashQueue.front()!=car || _engaged==_capacity ) return 0;
	_cashQueue.pop();  --_engaged;    // kilép a sorból és odalép egy kasszához
	_mu.unlock();

	unsigned int i;
	if ( !_station->search(car, i) ) return 0;
	int amount = _station->getQuantity(i) * _station->getUnit();
	_station->resetQuantity(i);     // lenullázza az i-dik töltõ kijelzõjét

	--_engaged;                     // elhagyja a kasszát
	_cond.notify_all();
	return amount;
}
```

várakozik a szál

elindítja a cond-nál várakozó szálakat (autósokat)

cash.cpp