

Unified Modeling Language

Szoftvertechnológia

Dr. Simon Balázs

BME, IIT

Tartalom

- UML diagrammok:
 - Szekvenciadiagram
 - Kommunikációs diagram
 - Interakciós áttekintő diagram
- Útmutató a házi feladathoz

Hol tartunk?



Use Case diagram

← A funkcionális követelmények magas szintű leírása:
A use case-ek alapvető interakciós sorozatok a rendszer és a felhasználói között

Aktivitásdiagram

← Use case interakciós sorozatok munkafolyamatként ábrázolva

Komponens-diagram

← Áttekintő kép a rendszer architektúrájáról:
A komponensek és kapcsolataik

Hova kell telepíteni a komponenseket

→ Telepítési diagram

Hol tartunk?



Osztály-
diagram

← Egy komponens/rendszer struktúrája
objektumorientált modellként

Csomag-
diagram

← Csomagok és a közöttük lévő függőségek

Objektum-
diagram

← A rendszer részletes állapotának ábrázolása
egy adott időpillanatban

Most következik:
Hogyan tervezzük meg egy komponens belsejét?
(Viselkedés)

Hol tartunk?

Most következik:
Hogyan tervezzük meg egy komponens belsejét?
(Viselkedés)

Strukturális UML diagramok:

Komponens- diagram	Telepítési diagram	Osztálydiagram	Csomagdiagram
Objektumdiagram	Összetett struktúradiagram	Profildíagram	

Viselkedési UML diagramok:

Use case diagram	Aktivitásdiagram	Szekvenciadiagram	Kommunikációs diagram
Állapotdiagram	Időzítődiagram	Interakciós áttekintő diagram	

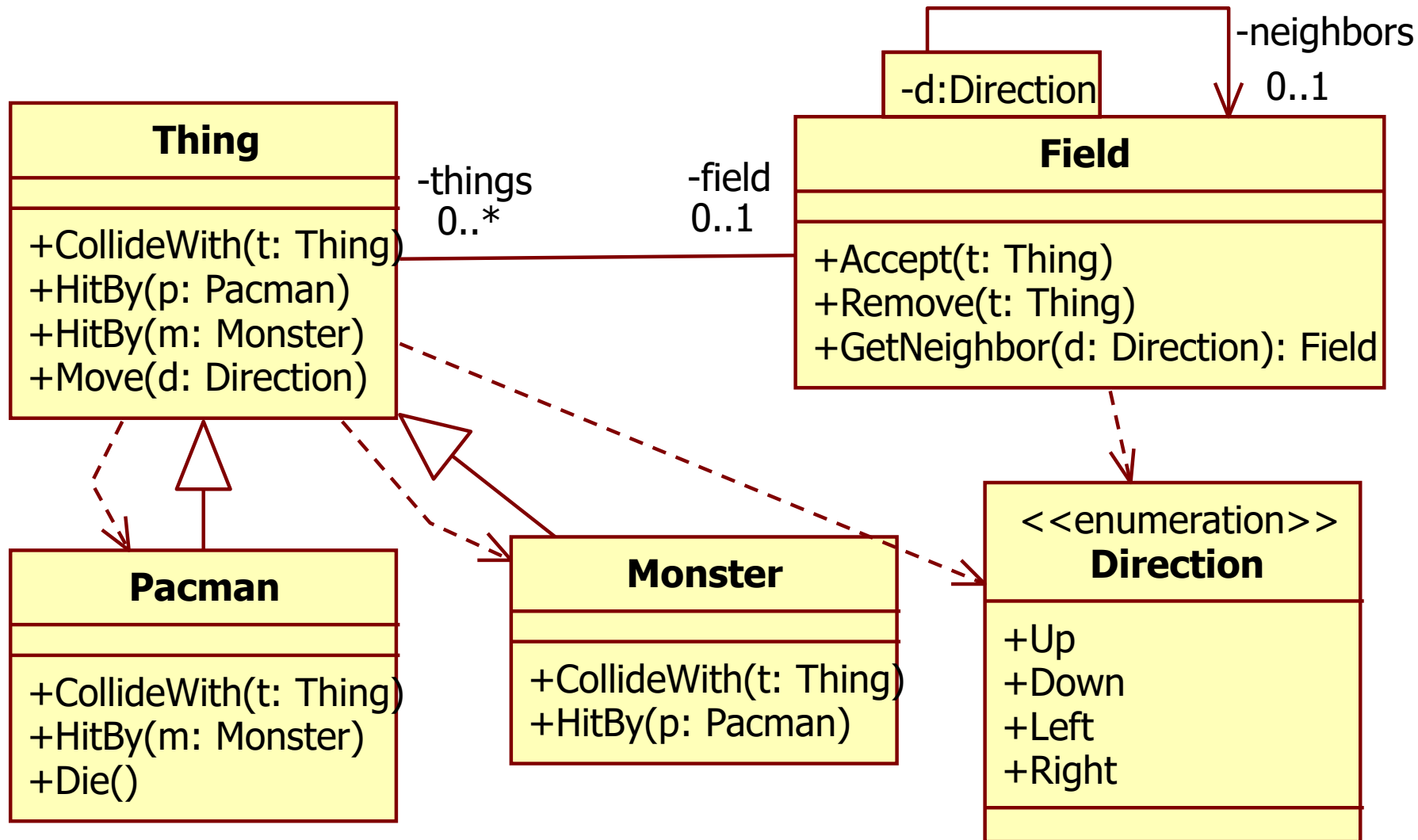
Szekvenciadiagram (Sequence Diagram)

Szekvenciadiagram (Sequence Diagram)

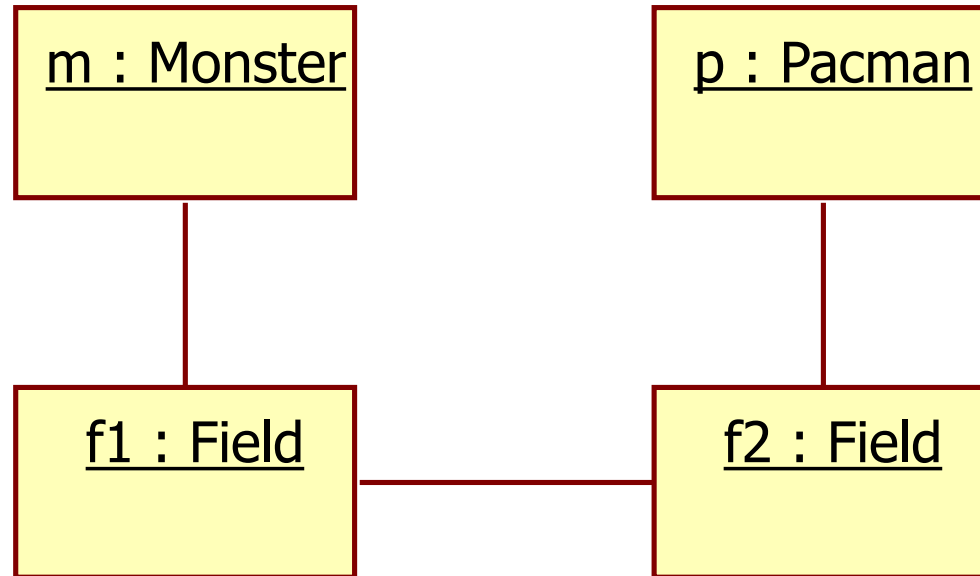


- Interakciók grafikus ábrázolására szolgál
 - a rendszer dinamikus viselkedését mutatja
 - az interakciók a résztvevők közötti információcserére fókuszálnak
- A szekvenciadiagram használható use case forgatókönyvek leírására, metódusok belső logikájának definiálására és egy protokollban történő üzenetváltások ábrázolására
- Egy szekvenciadiagram üzenetváltások egy lehetséges időbeli lefutását mutatja
 - egyszerű futássorozatok vannak ábrázolva
 - le lehet írni helyes és helytelen lefutásokat is
 - a nem ábrázolt lefutásokról nem mindig dönthető el egyértelműen, hogy helyesek vagy helytelenek

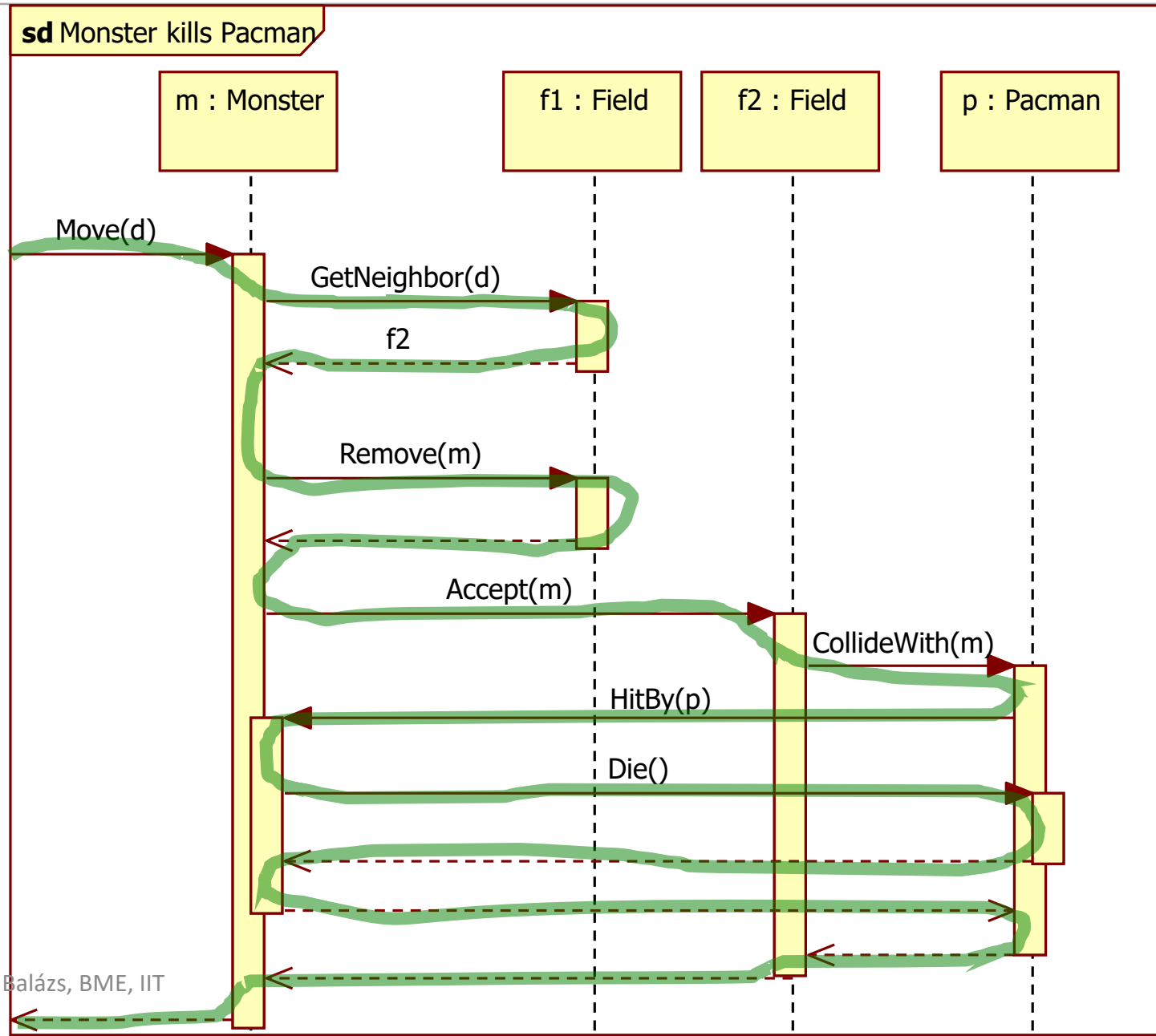
Pacman: osztálydiagram részlet



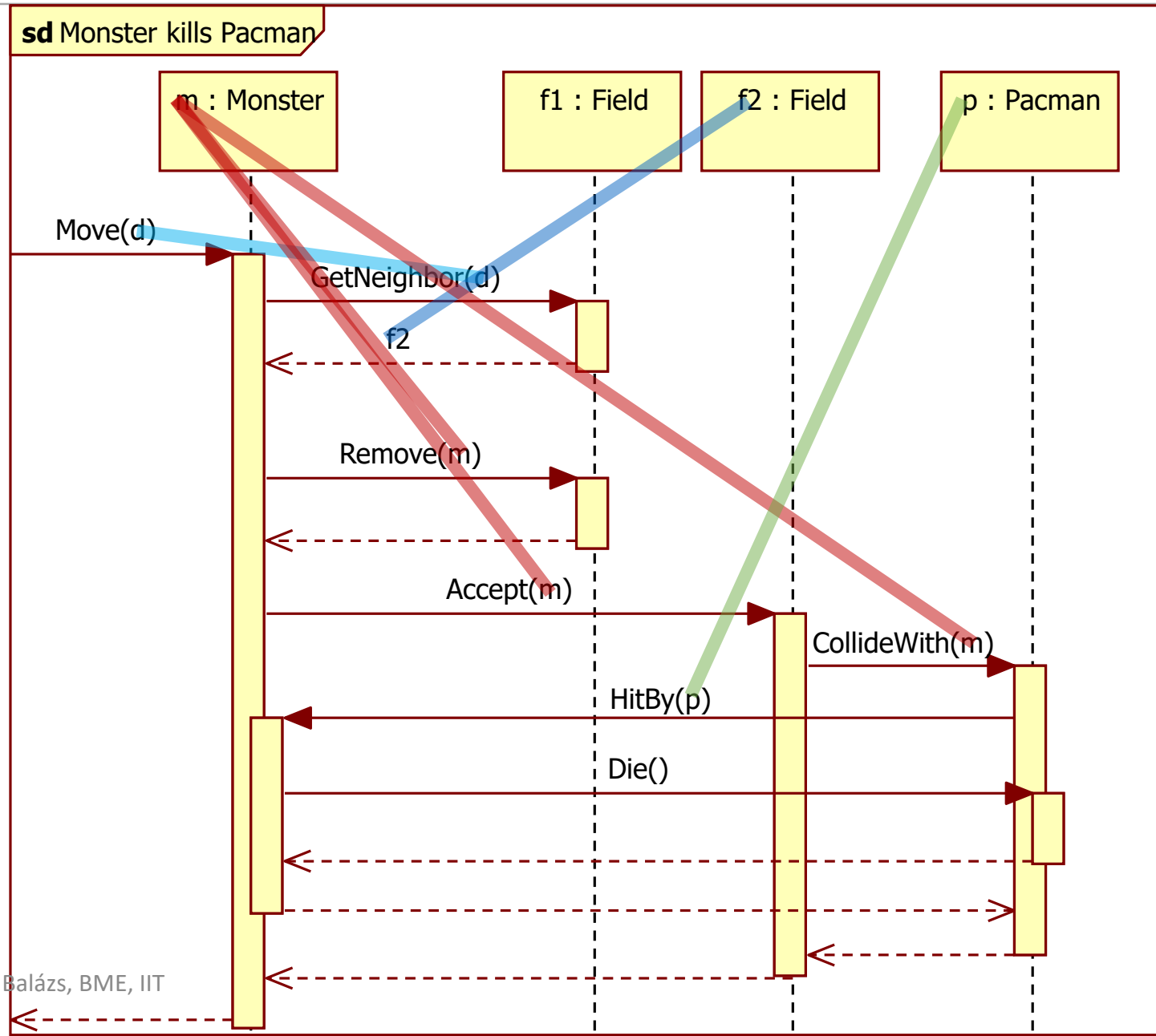
Pacman: objektumdiagram egy lehetséges kezdőállapotra



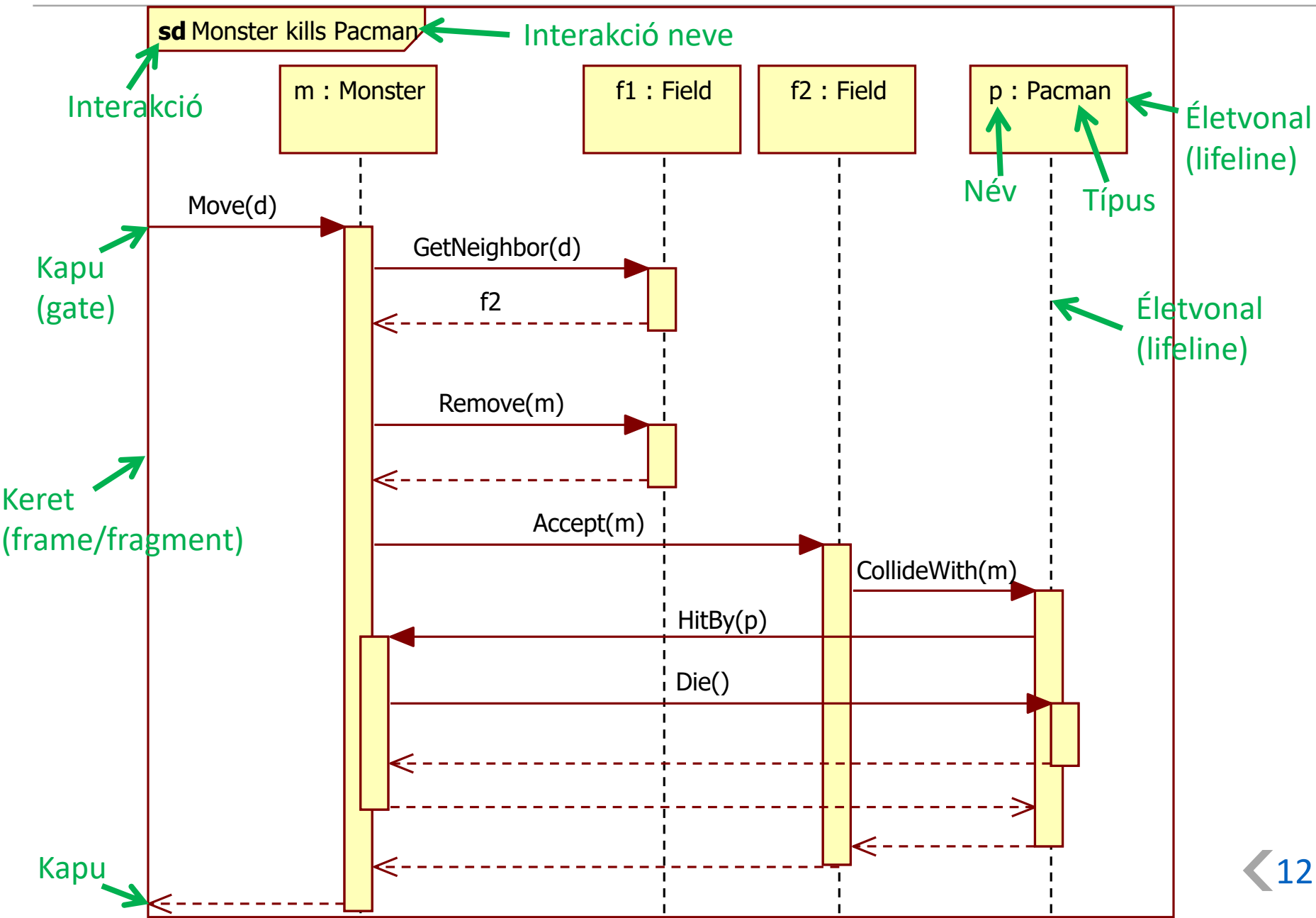
Szekvenciadiagram: a szörny megeszi a Pacmant



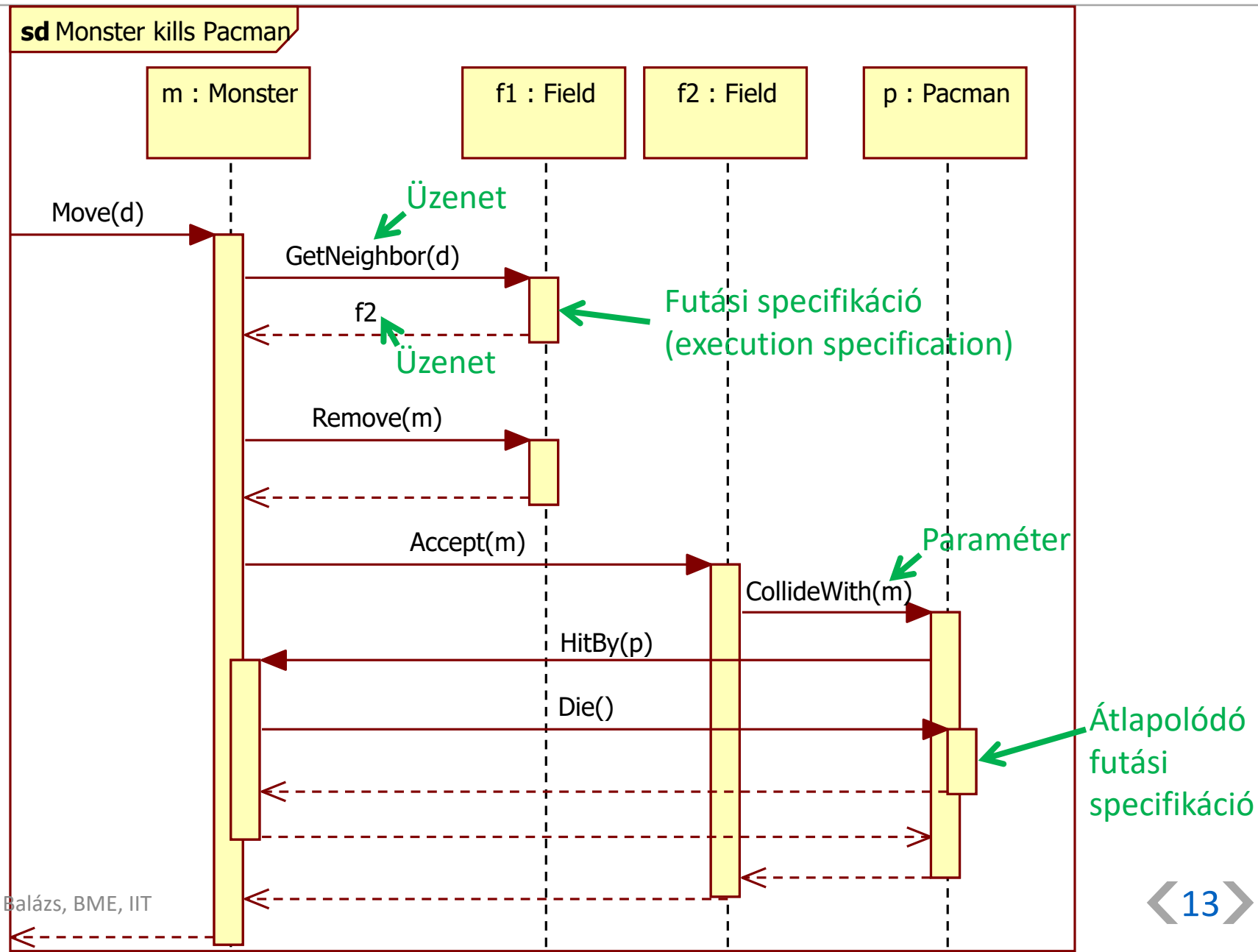
Szekvenciadiagram: a szörny megeszi a Pacmant



Szekvenciadiagram: a szörny megeszi a Pacmant



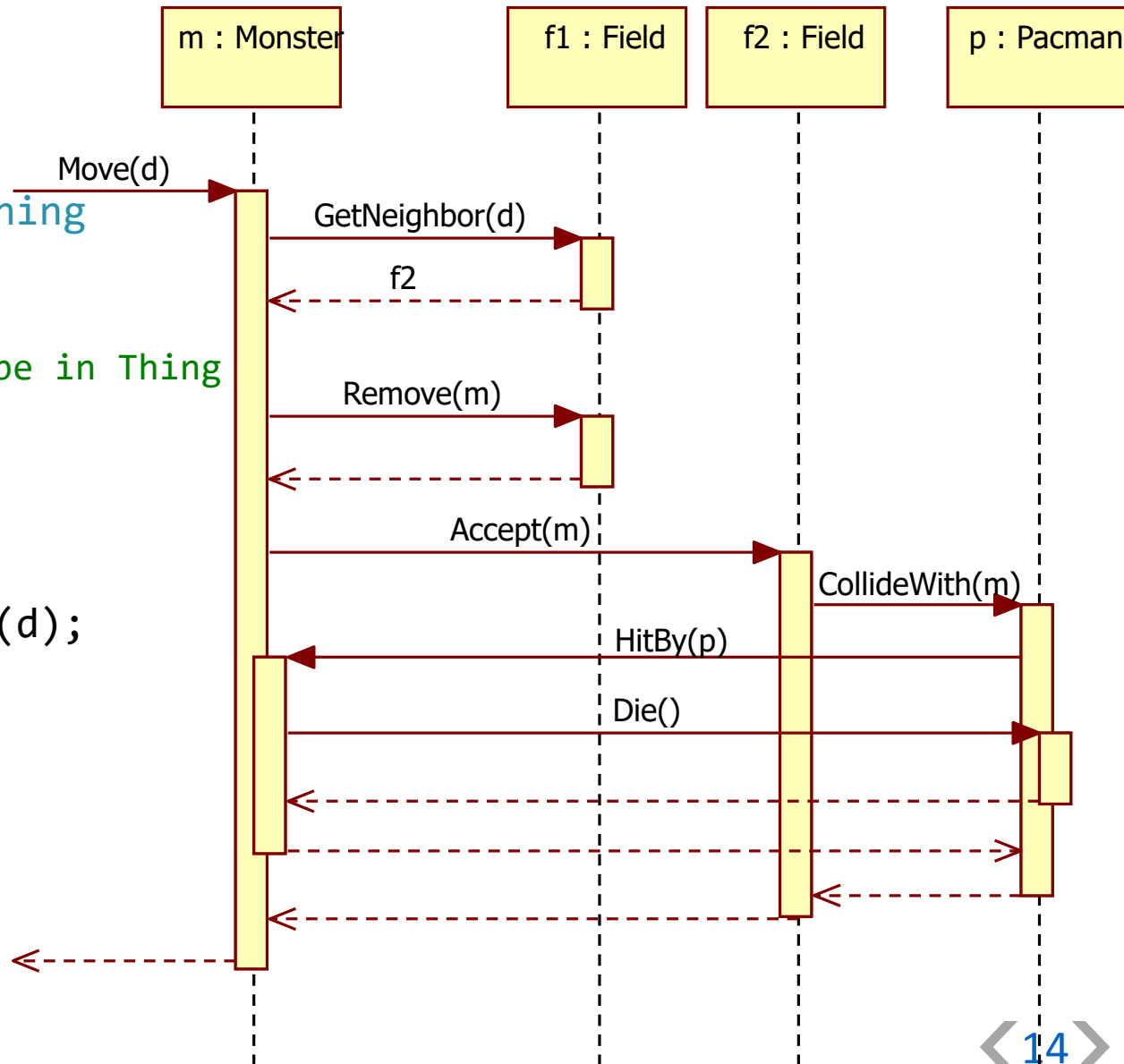
Szekvenciadiagram: a szörny megeszi a Pacmant



Szekvenciadiagram: a szörny megeszi a Pacmant

C++ leképzés:

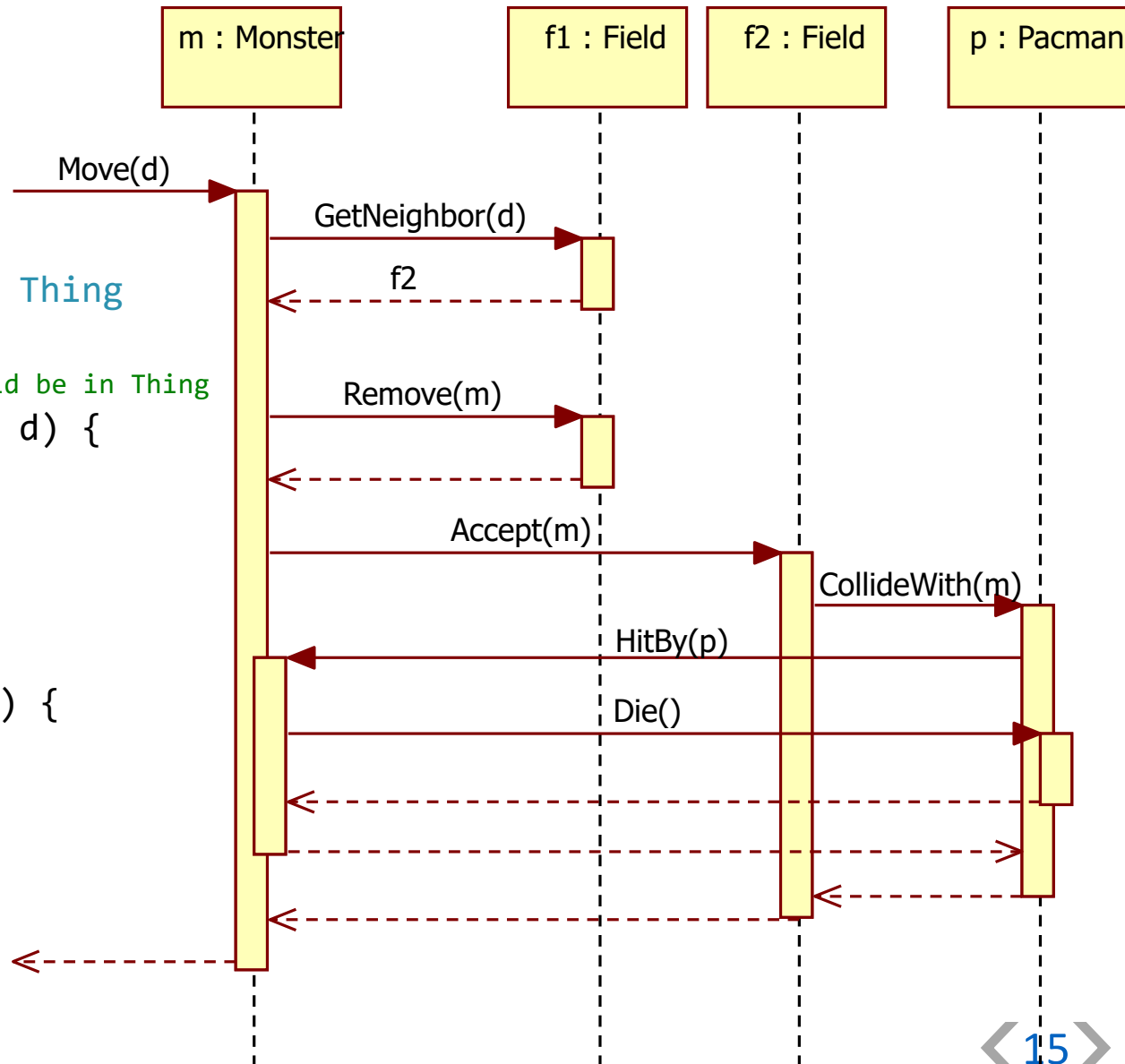
```
class Monster : public Thing
{
private:
    Field* field; //should be in Thing
public:
    void Move(Direction d)
    {
        Field* next =
            field->GetNeighbor(d);
        field->Remove(this);
        next->Accept(this);
    }
    void HitBy(Pacman* p)
    {
        p->Die();
    }
    // ...
}
```



Szekvenciadiagram: a szörny megeszi a Pacmant

Java leképzés:

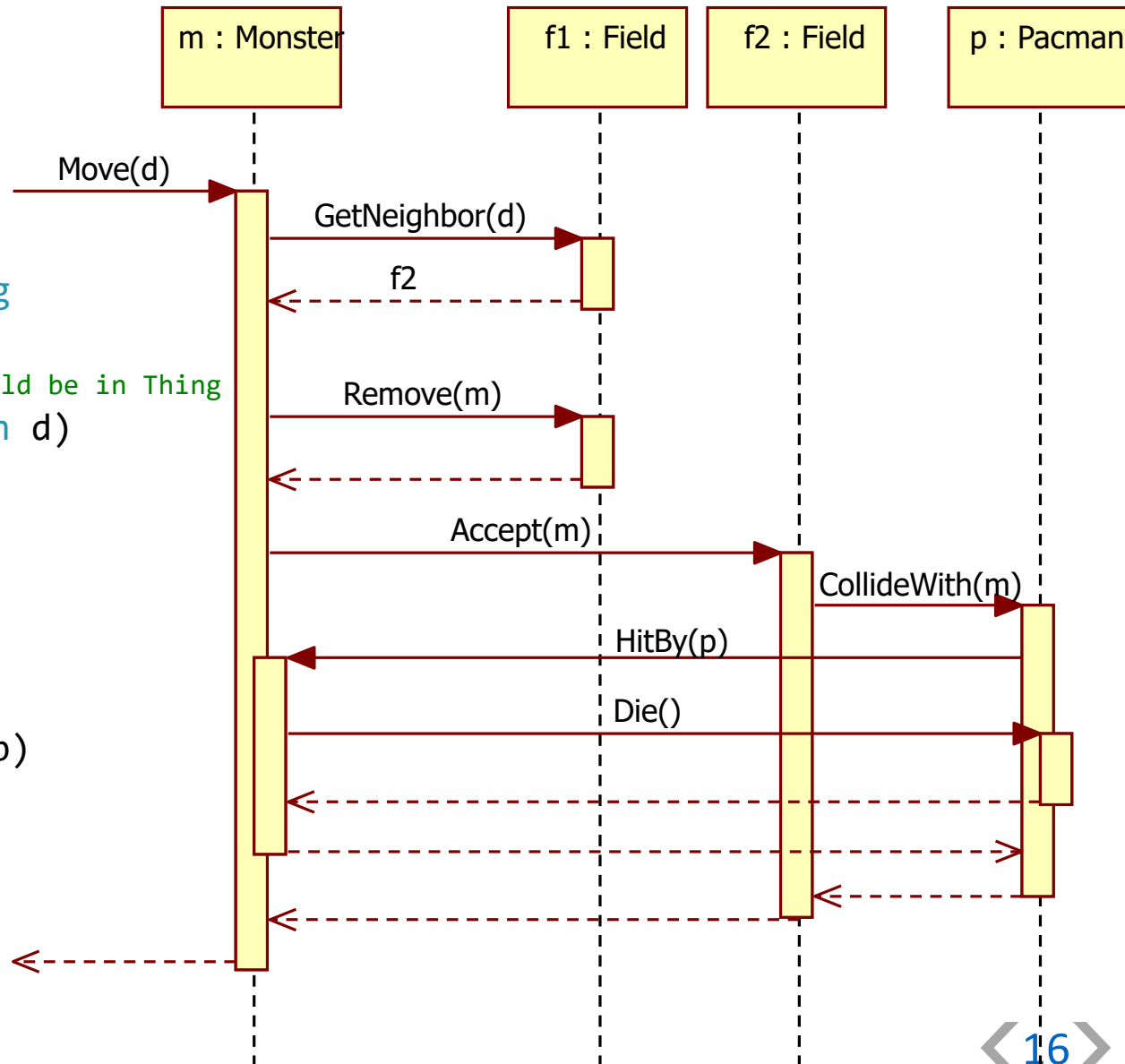
```
public class Monster extends Thing
{
    private Field field; //should be in Thing
    public void Move(Direction d) {
        Field next =
            field.GetNeighbor(d);
        field.Remove(this);
        next.Accept(this);
    }
    public void HitBy(Pacman p) {
        p.Die();
    }
    // ...
}
```



Szekvenciadiagram: a szörny megeszi a Pacmant

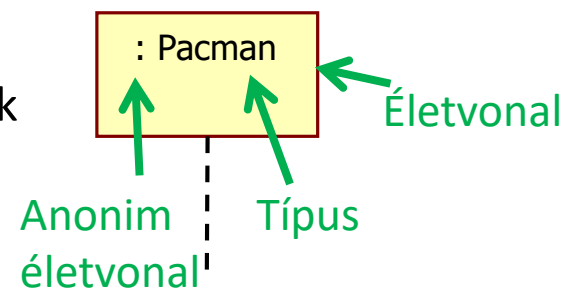
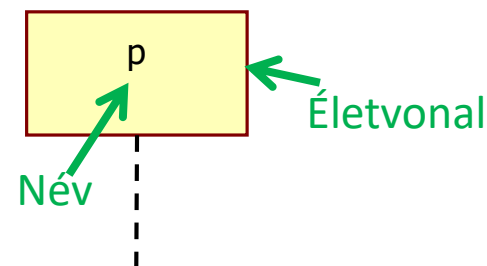
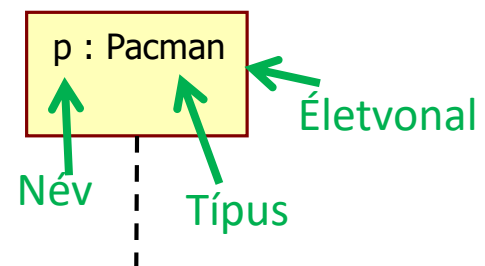
C# leképzés:

```
public class Monster : Thing
{
    private Field field; //should be in Thing
    public void Move(Direction d)
    {
        Field next =
            field.GetNeighbor(d);
        field.Remove(this);
        next.Accept(this);
    }
    public void HitBy(Pacman p)
    {
        p.Die();
    }
    // ...
}
```



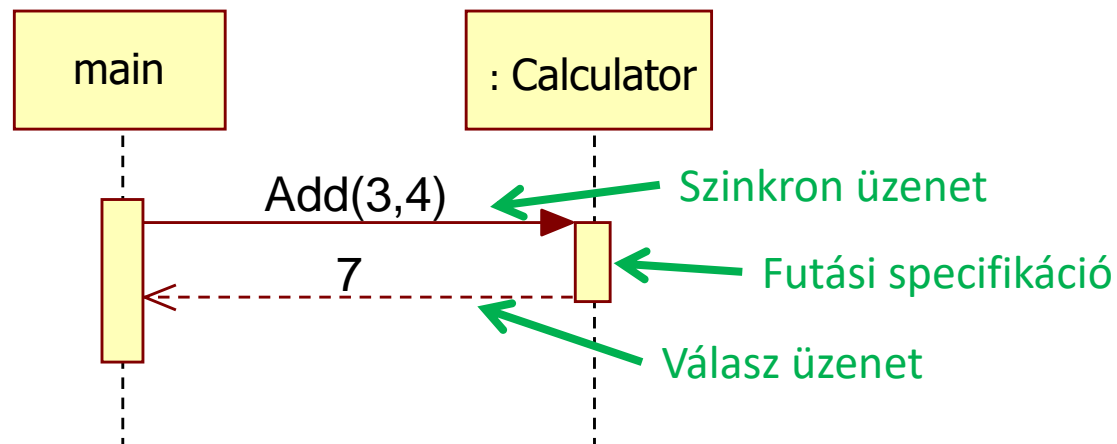
Életvonal (lifeline)

- Egy folyamat idővonalát ábrázolja, ahol az idő fentről lefelé telik
- Az életvonal feje nem objektum!
 - egy objektum (példány specifikáció) az csak egy pillanatkép a dinamikus példányról, amit ő modellez
 - a szekvenciadiagram nem egy pillanatkép, hanem egy időbeli folyamat
 - vagyis: a lifeline fejében *nem kell aláhúzni a szöveget*
 - akár interfész vagy absztrakt osztály is szerepelhet a lifeline fejében
 - de természetesen absztrakt függvények vagy interfészek függvényei csak hívhatók, de mivel nekik nincs implementációjuk, ők nem hívhatnak más függvényeket
- A név és a típus is opcionális, de legalább egyiknek szerepelnie kell



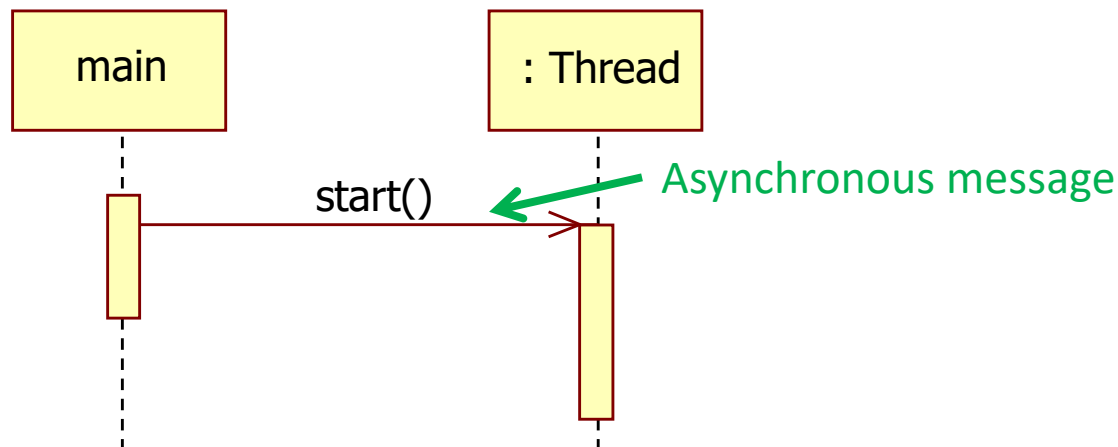
Szekvenciadiagram: szinkron üzenet

- Szinkron üzenet (synchronous message):
 - szinkron függvényhívás
 - az operáció neve és a paraméterek a nyíl felett szerepelnek
 - hatására egy futási specifikáció (execution specification) indul
 - a hívó megvárja, amíg a meghívott függvény véget ér
- Válasz üzenet:
 - a futási specifikáció végétől van rajzolva
 - a visszatérési érték a szaggatott vonal felett szerepel
- Futási specifikáció (execution specification):
 - egy függvényhívás törzsének futását jelöli, ekkor aktív a függvény
 - a programozási nyelvekben ez a stack frame



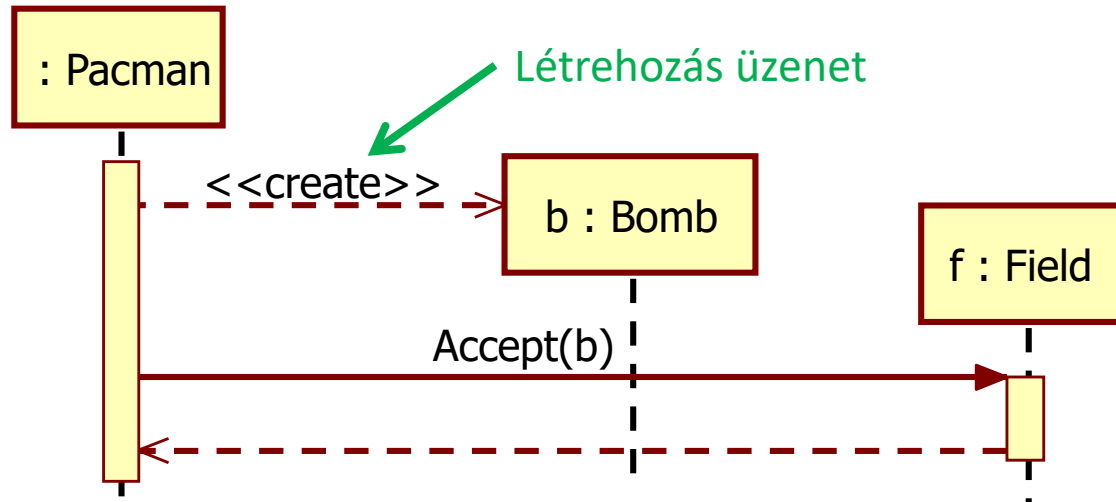
Szekvenciadiagram: aszinkron üzenet

- Aszinkron üzenet (asynchronous message), jelzés (signal):
 - aszinkron függvényhívás
 - a hívó nem várja meg a függvény lefutásának végét
 - egy másik szálát vagy folyamatot indít el
 - az operáció neve és a paraméterek a nyíl felett szerepelnek
 - hatására egy futási specifikáció (execution specification) indul
 - ami nem feltétlenül fér bele a hívó futási specifikációjának időtartamába
 - aszinkron üzenetnek nincs válaszüzenete
 - egy másik aszinkron hívással lehet visszahívást (callback) végezni

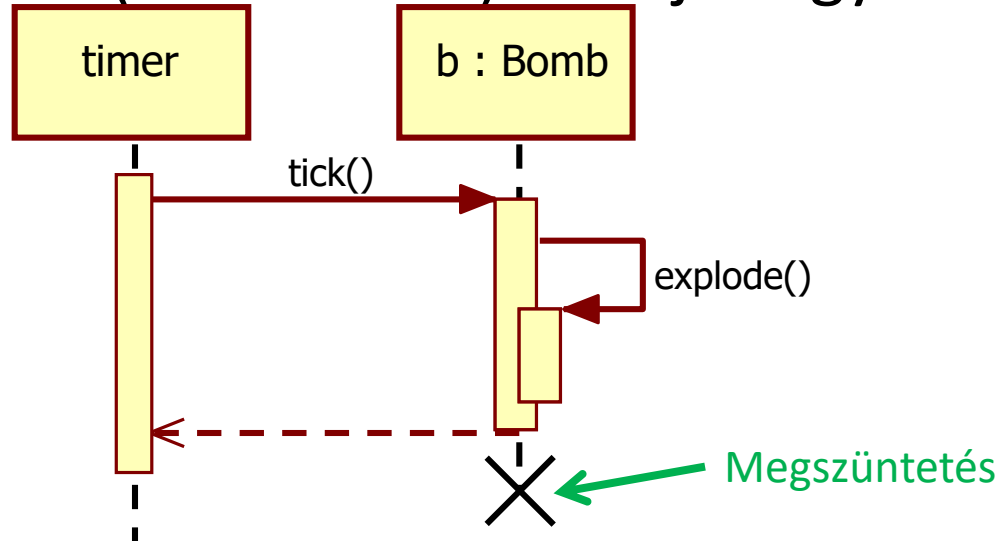


Szekvenciadiagram: létrehozás és megszüntetés

- Létrehozás üzenet (create message): új életvonalat készít



- Megszüntetés (destruction): befejez egy életvonalat



Szekvenciadiagram: alternatívák (alternatives)

■ Alternatívák: **alt**

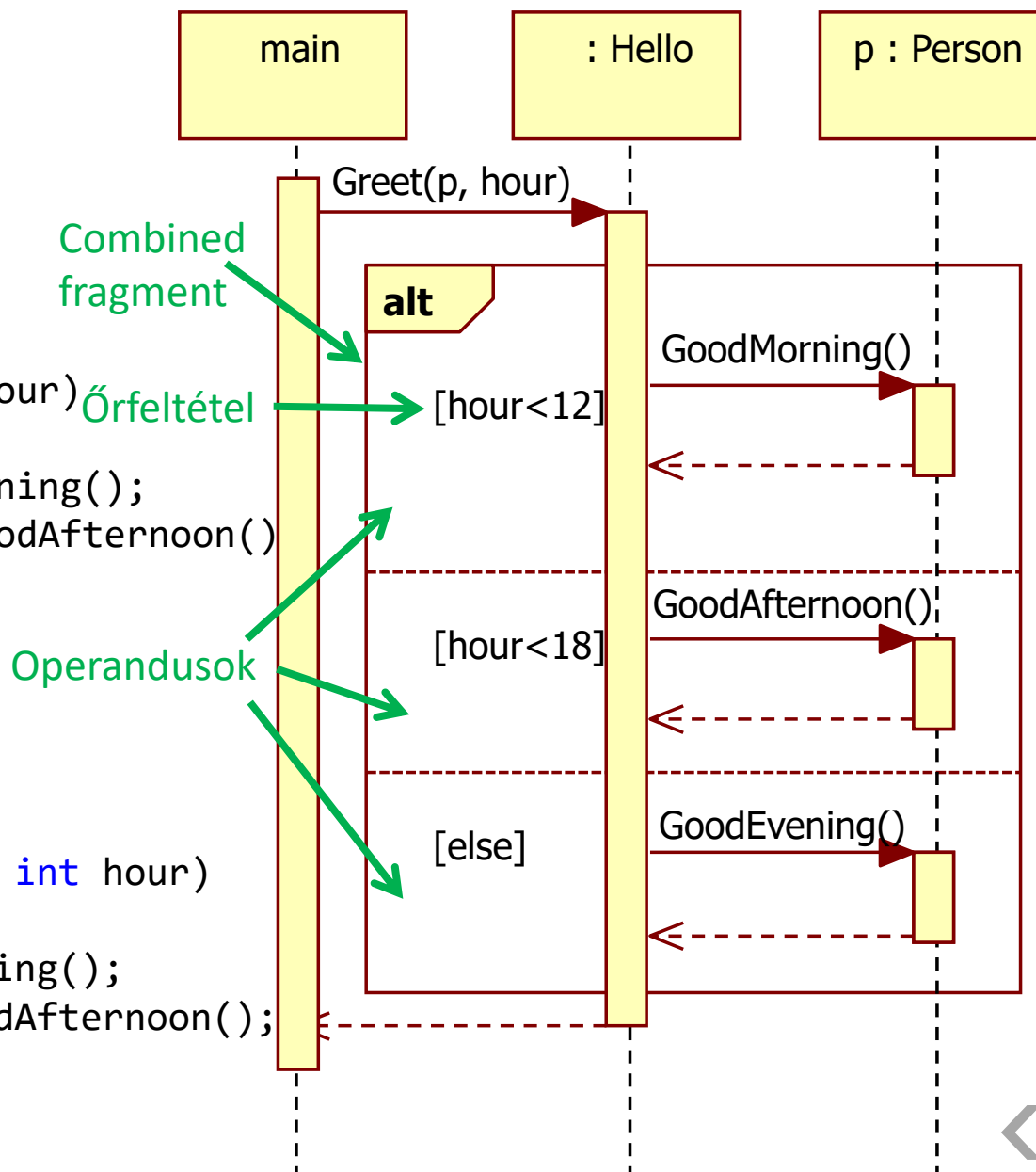
■ if-else ágak

C++ leképezés:

```
class Hello {  
public:  
    void Greet(Person* p, int hour)  
    {  
        if (hour < 12) p->GoodMorning();  
        else if (hour < 18) p->GoodAfternoon();  
        else p->GoodEvening();  
    }  
}
```

Java/C# leképezés:

```
public class Hello {  
    public void Greet(Person p, int hour)  
    {  
        if (hour < 12) p.GoodMorning();  
        else if (hour < 18) p.GoodAfternoon();  
        else p.GoodEvening();  
    }  
}
```



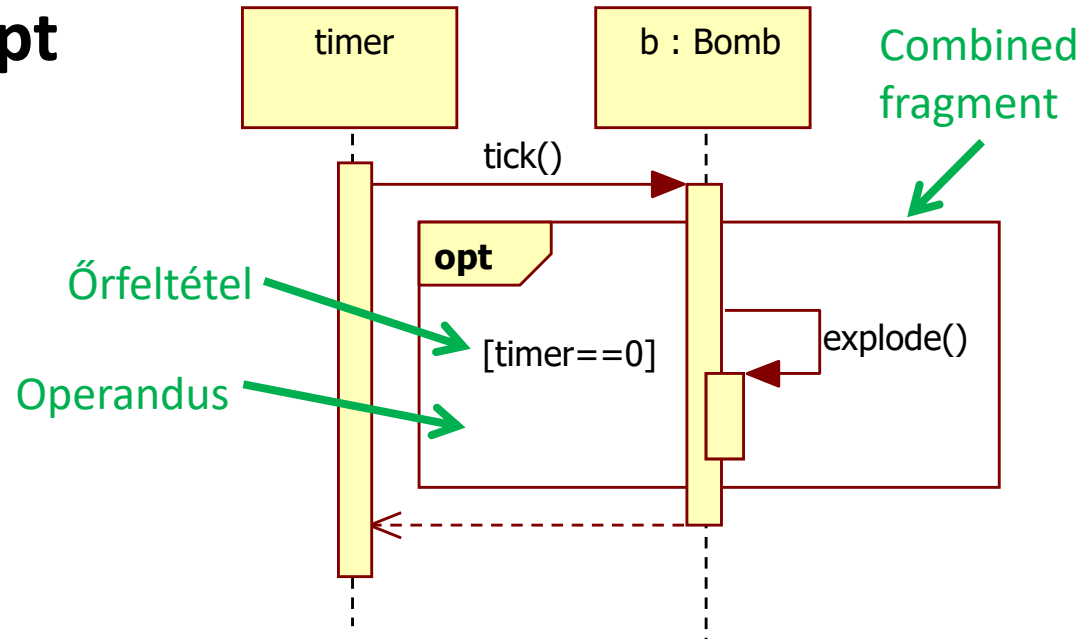
Szekvenciadiagram: opció (option)

■ Feltételes viselkedés: **opt**

- else nélküli if

C++ leképezés:

```
class Bomb {  
private:  
    int timer;  
public:  
    void Tick() {  
        --timer;  
        if (timer == 0) {  
            this.Explode();  
        }  
    }  
    void Explode() { /*...*/ }  
}
```



Java/C# leképezés:

```
public class Bomb {  
    private int timer;  
    public void Tick() {  
        --timer;  
        if (timer == 0) {  
            this.Explode();  
        }  
    }  
    public void Explode() { /*...*/ }  
}
```

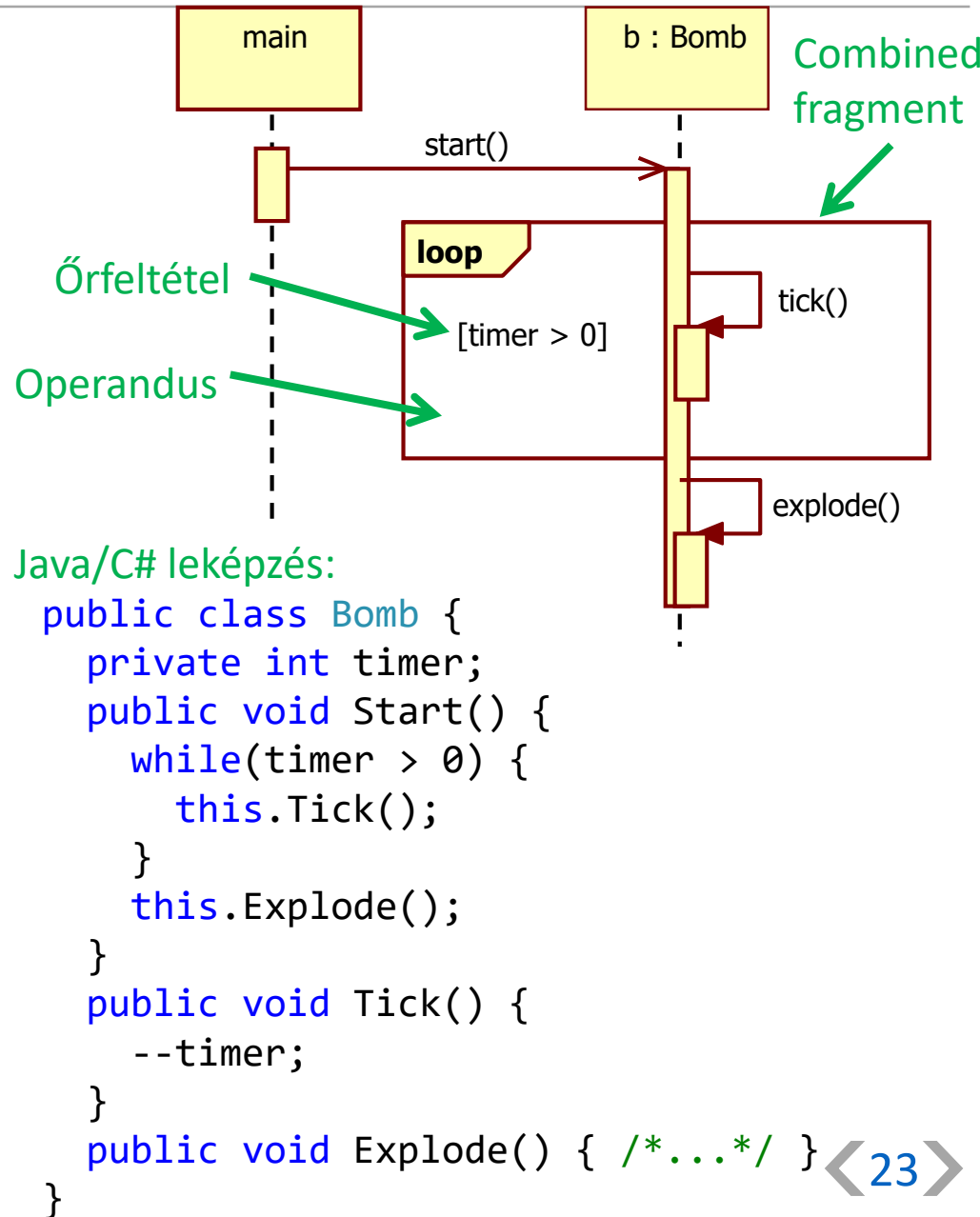
Szekvenciadiagram: ciklus (loop)

■ Ismételt viselkedés: **loop**

C++ leképzés:

```
class Bomb
{
private:
    int timer;
public:
    void Start()
    {
        while(timer > 0)
        {
            this.Tick();
        }
        this.Explode();
    }
    void Tick()
    {
        --timer;
    }
    void Explode() { /*...*/ }
```

Dr Simon Balázs, BME, IIT



Szekvenciadiagram : combined fragments

Rövidítés	Fajta	Jelentés
alt	Alternatívák	Viselkedés kiválasztása feltétel alapján: legfeljebb egy operandus lesz lefuttatva.
opt	Opció	Opcionális viselkedés feltétel alapján: vagy lefut az egyetlen operandus, vagy nem történik semmi.
break	Megszakítás	A tartalmazó fragment futása megszakad, és a maradék rész nem fut le.
par	Párhuzamos	Párhuzamosan futnak le az operandusok.
seq	Gyenge sorrend	Gyenge sorrendet határoz meg az operandusok között: csak az azonos lifeline-on belül kell tartani a sorrendet.
strict	Erős sorrend	Erős sorrendet határoz meg az operandusok között: a függőleges koordináta szigorúan meghatározza a sorrendet.

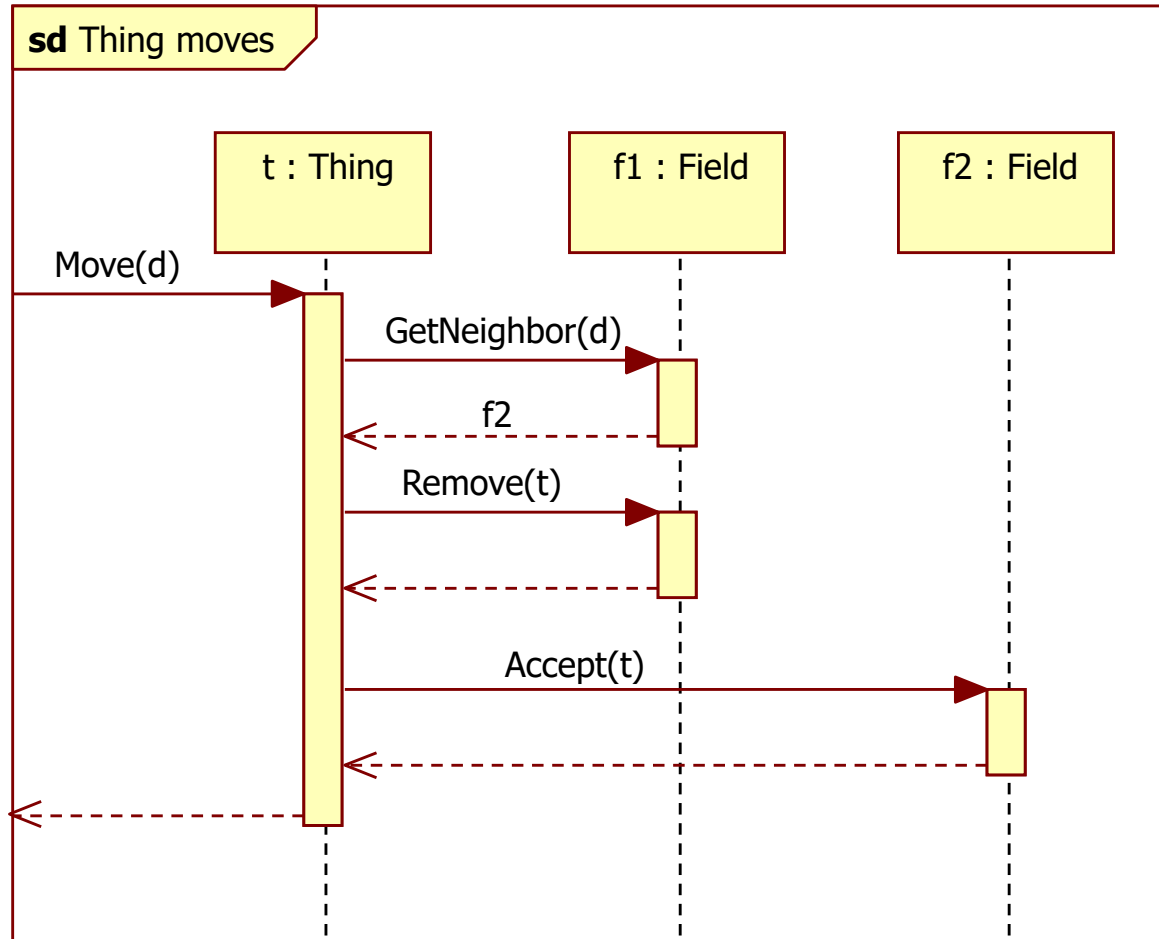
Szekvenciadiagram : combined fragments

Rövidítés	Fajta	Jelentés
neg	Negatív	Helytelen lefutásokat mutat. Minden ezektől eltérő lefutás helyesnek és lehetségesnek számít.
critical	Kritikus szakasz	Atomi műveletnek tekinthető a tartalmazó fragment szempontjából.
ignore	Rejtett üzenetek	Azt jelzi, hogy néhány üzenet nincs megmutatva az adott fragmentben.
consider	Fontos üzenetek	Azt jelzi, hogy néhány üzenet fontos az adott fragmentben, a többi üzenet rejtettnek tekinthető.
assert	Állítás	Egy állítást reprezentál. Csak az operandus szekvenciái a helyes folytatások, minden más folytatás helytelen.
loop	Ciklus	Ciklust reprezentál: az operandus ismételten le lesz futtatva.

Szekvenciadiagram : interakció használata (interaction use)

- A szekvenciadiagramok meghivatkozhatnak más szekvenciadiagramokat az interakció használata segítségével
- Előnyök:
 - modularitás: nagy diagramok feldarabolhatók több kisebb diagramra
 - újrahasznosítás: több diagramon előforduló azonos viselkedés kiemelhető és meghivatkozható egy közös diagram alapján
 - olvashatóság: magasabb szintű diagramok finomíthatók alacsonyabb szintű diagramok segítségével

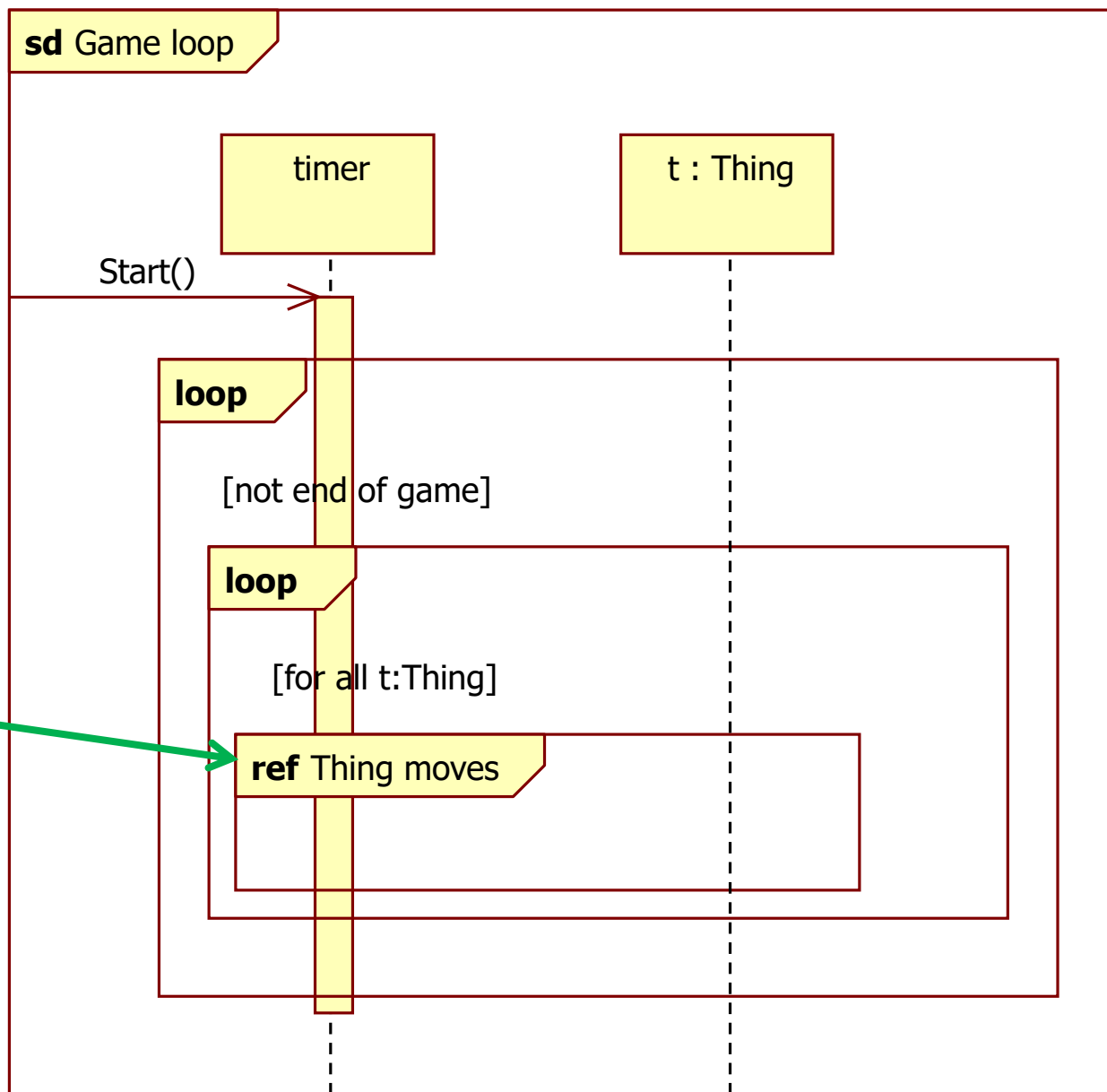
Szekvenciadiagram: Thing moves



Szekvenciadiagram: Interakció használata

Interakció
használata

(meghivatkozza az
előző dián
szereplő
szekvenciát)



Konzisztens és kezelhető szekvenciadiagramok

- Egy diagram pontosan egy viselkedést ábrázoljon
- Azonos típusú, de különböző objektumok külön lifeline-t kapjanak
- A hívónak ismernie kell a cél objektumot
 - egy objektumdiagramon ábrázolhatjuk a kezdeti ismeretségeket
- A hívott függvénynek elérhetőnek kell lennie a használt objektum ismert statikus típusa alapján
- A polimorfikus viselkedés leírására külön diagramokat rajzoljunk
- Ugyanaz a függvény ugyanúgy viselkedjen különböző diagramokon
- Tüntessük fel a paramétereket és a visszatérési értékeket is
- Az operációknak létezniük kell az osztálydiagramon

Informális szekvenciadiagram

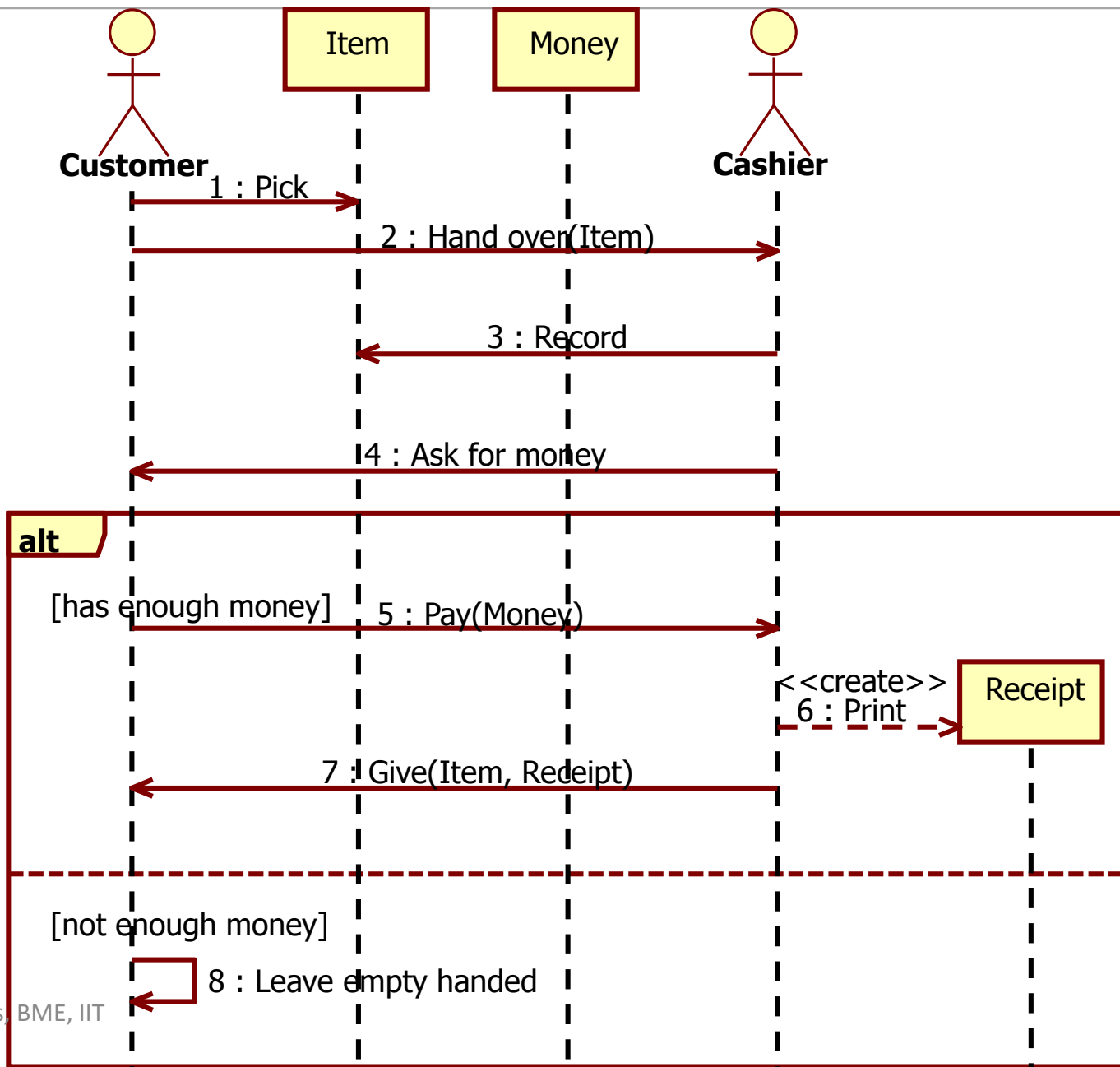
- Informális szekvenciadiagramok:
 - enyhítenek az UML szigorú formalizmusán
 - nem pontosan követik az UML szabványt
 - de kiválóan alkalmasak ötletek felvázolására
- Tipikusan nem rajzoljuk meg a rendszer minden egyes apró részletét formális diagramokon
 - általában ez szükségtelen
 - sokszor időpazarló: a viselkedés leprogramozása egyszerűbb és gyorsabb, mint békés eszközökkel szekvenciadiagramokat rajzolgatni
 - szinkronban tartani őket a forráskóddal is időpazarló, és nehéz
- Általában az informális diagramok elegendő információt adnak a rendszer működéséről
 - ezeket könnyebb használni vázlatként
 - elegendő részletet adnak az áttekintő kép megértéséhez
 - csak akkor kell őket frissíteni a dokumentációban, ha az általuk mutatott viselkedést befolyásoló tervezői döntések megváltoznak
- **Megjegyzés: a házi feladatban és a vizsgán, valamint a „Szoftver projekt labor” c. tárgyban részletes *formális* diagramokat várunk el**
 - a célunk, hogy lássuk, sikerült-e elsajátítani az UML szabványt

Emlékeztető: Vásárlás use case



- **Use case:** Árucikk vásárlása
- **Aktorok:** Vásárló, Pénztáros
- **Főforgatókönyv:**
 - 1. A Vásárló kiválasztja az árucikket
 - 2. A Vásárló átadja az árucikket a Pénztárosnak
 - 3. A Pénztáros lehúzza az árucikket a pénztárgépen
 - 4. A Pénztáros elkéri a pénzt a Vásárlótól
 - 5. A Vásárló kifizeti az összeget a Pénztárosnak
 - 6. A Pénztáros visszaadja az árucikket és a blokkot a Vásárlónak
- **Alternatív forgatókönyv 5.A:**
 - 5.A.1. A Vásárlónak nincs elég pénze
 - 5.A.2. A Vásárló üres kézzel távozik

Informális szekvenciadiagram a vásárlásra



Hol tartunk?

Strukturális UML diagrammok:

Komponens-diagram	Telepítési diagram	Osztálydiagram	Csomagdiagram
Objektumdiagram	Összetett struktúradiagram	Profildiagram	

Viselkedési UML diagrammok:

Use case diagram	Aktivitásdiagram	Szekvenciadiagram	Kommunikációs diagram
Állapotdiagram	Időzítődiagram	Interakciós áttekintő diagram	

Kommunikációs diagram (Communication diagram)

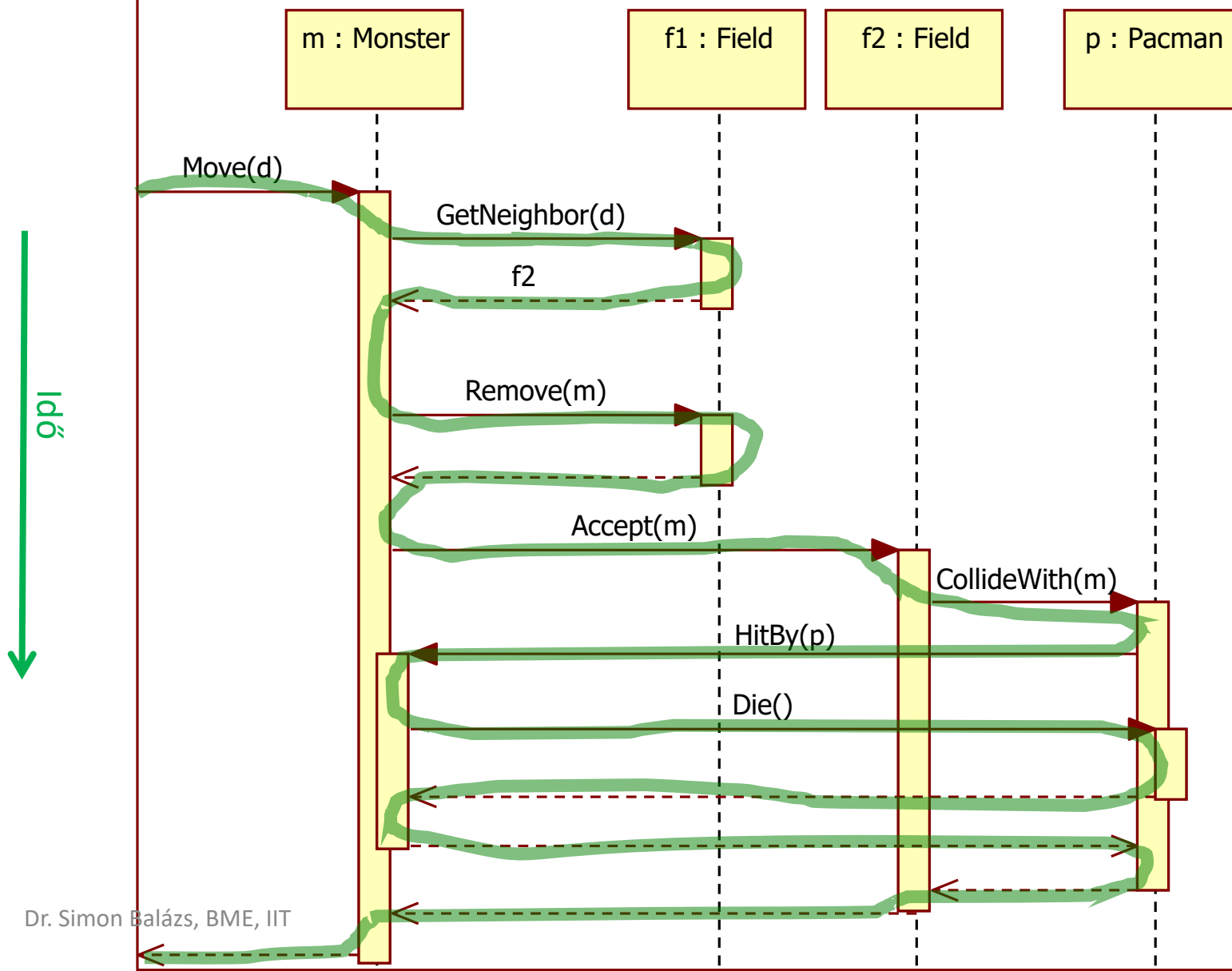
Kommunikációs diagram



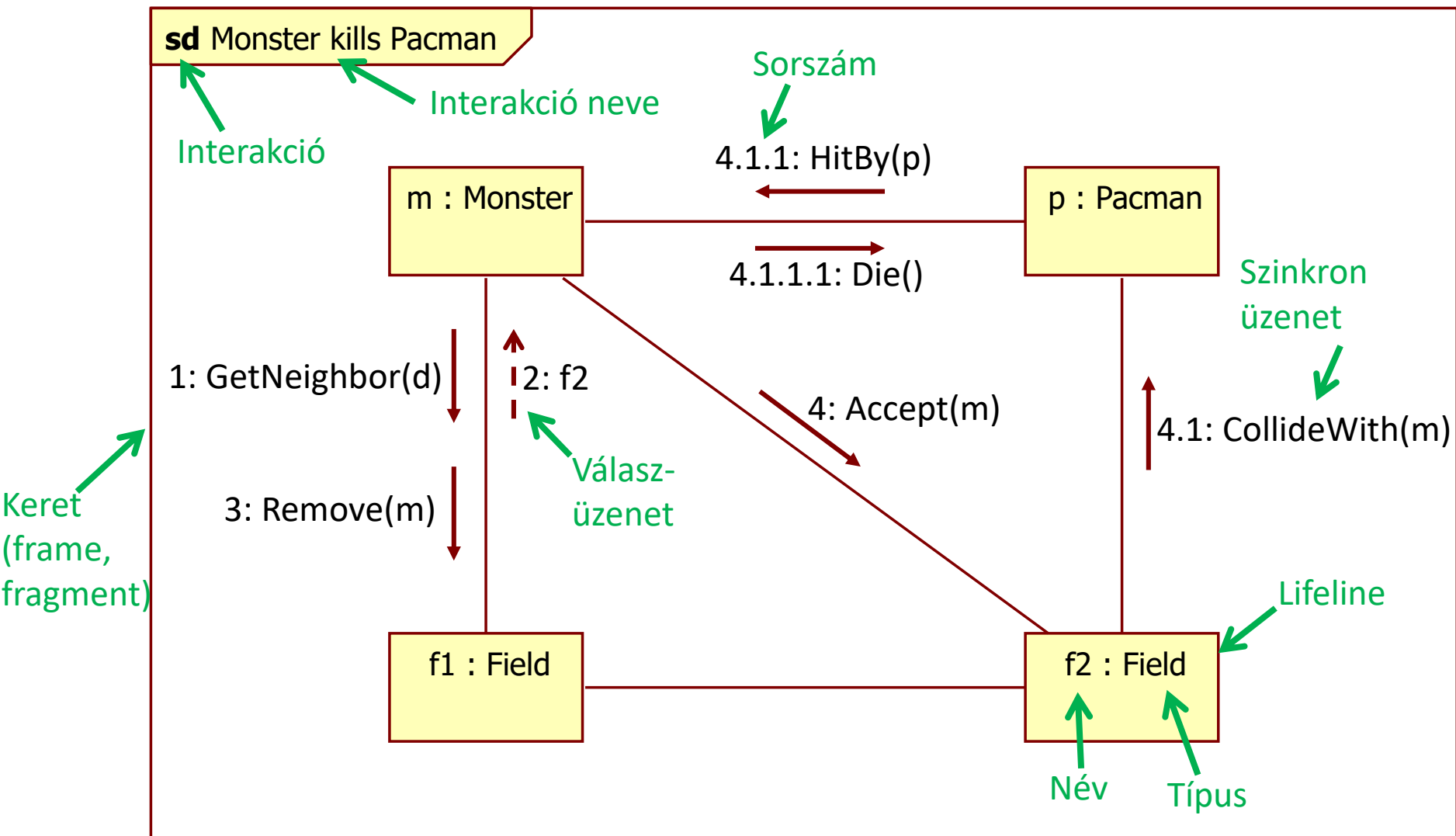
- Interakciók grafikus ábrázolására szolgál
 - a rendszer dinamikus viselkedését mutatja
 - az interakciók a résztvevők közötti információcserére fókuszálnak
 - az üzenetek sorrendjét hierarchikus számozás határozza meg
- Egy kommunikációs diagram olyan szekvenciadiagramnak felel meg, amelyen nincs strukturális jelölés (interakció használata, combined fragment)
 - a szekvenciadiagramok erőssége a logikai sorrend mutatása, az áttekintő képet azonban nehéz látni
 - a kommunikációs diagramok nagyon jó áttekintő képet adnak (szereplők és kapcsolataik), de nehéz követni a logikai sorrendet
 - a kommunikációs diagram olyan, mintha a szekvenciadiagramot felülről néznénk

Emlékeztető: a szörny megeszi a Pacmant

sd Monster kills Pacman



Kommunikációs diagram: a szörny megeszi a Pacmant



Hol tartunk?

Strukturális UML diagrammok:

Komponens-diagram	Telepítési diagram	Osztálydiagram	Csomagdiagram
Objektumdiagram	Összetett struktúradiagram	Profildiagram	

Viselkedési UML diagrammok:

Use case diagram	Aktivitásdiagram	Szekvenciadiagram	Kommunikációs diagram
Állapotdiagram	Időzítődiagram	Interakciós áttekintő diagram	

Interakció áttekintő diagram (Interaction Overview Diagram)

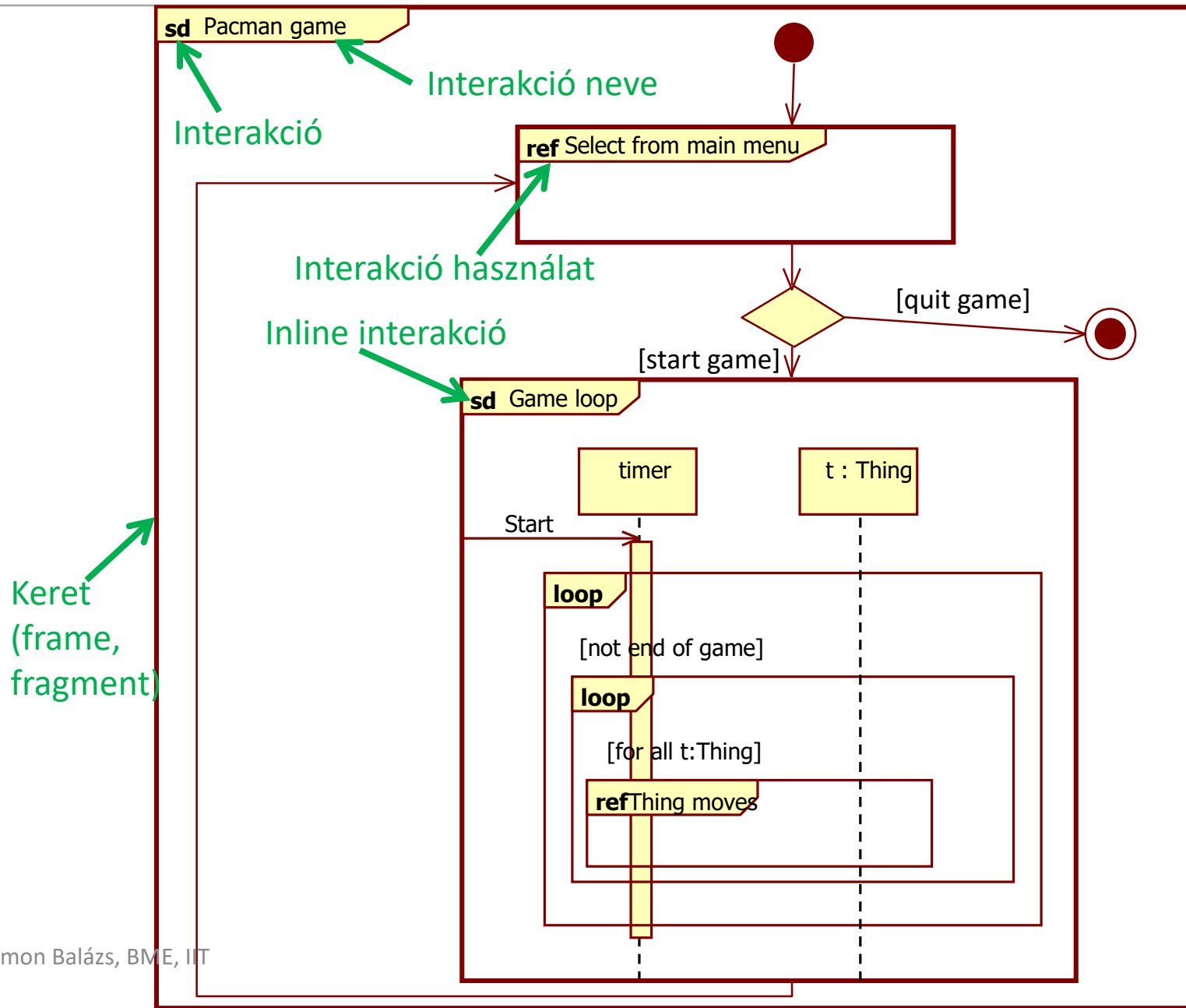
Interakciós áttekintő diagram



- Az interakciós áttekintő diagramok az aktivitásdiagramok egyfajta változatai, amelyek az interakciók egymásutániságáról adnak áttekintő képet
- A különbség az aktivitásdiagramhoz képest: a csomópontok nem akciók, hanem interakciók és interakció használatok
 - Interakciók: inline szekvenciadiagram, kollaborációs diagram, vagy interakciós áttekintő diagram
 - Interakció használatok: referencia egy szekvenciadiagramra, kollaborációs diagramra vagy interakciós áttekintő diagramra

- További különbségek az aktivitásdiagramokhoz képest:
 - csomópontok: interakciók vagy interakció használatok
 - feltételes combined fragment-ek megfelelője a döntési-merge csomópontpárok
 - a párhuzamos combined fragment-ek megfelelője a fork-join csomópontpárok
 - a ciklus combined fragment-ek megfelelője a gráfban létrehozott körök
 - a feltételes és párhuzamos részeknek hierarchikusan egymásba ágyazottnak kell lennie
 - az interakciós áttekintő diagramok is ugyanolyan interakciós keretben vannak, mint a szekvenciadiagramok és a kollaborációs diagramok

Interakciós áttekintő diagram példa: Pacman játék



Hol tartunk?

Strukturális UML diagramok:

Komponens-diagram	Telepítési diagram	Osztálydiagram	Csomagdiagram
Objektumdiagram	Összetett struktúradiagram	Profildíagram	

Viselkedési UML diagramok:

Use case diagram	Aktivitásdiagram	Szekvenciadiagram	Kommunikációs diagram
Állapotdiagram	Időzítődiagram	Interakciós áttekintő diagram	

Összefoglalás

- Interakciós grafikus reprezentációja
- A fókusz a résztvevők közötti információcserén van
- UML diagramok:
 - **Szekvenciadiagram:** a logikai sorrendet mutatja, az áttekintő képet azonban nehéz látni
 - **Kommunikációs diagram:** nagyon jó áttekintő képet ad (szereplők és kapcsolataik), de nehéz követni a logikai sorrendet
 - **Interakciós áttekintő diagram:** az aktivitásdiagramok egyfajta változata, amely az interakciók egymásutániságáról ad áttekintő képet

Hol tartunk?

Strukturális UML diagramok:

Komponens-diagram	Telepítési diagram	Osztálydiagram	Csomagdiagram
Objektumdiagram	Összetett struktúradiagram	Profildíagram	

Viselkedési UML diagramok:

Use case diagram	Aktivitásdiagram	Szekvenciadiagram	Kommunikációs diagram
Állapotdiagram	Időzítődiagram	Interakciós áttekintő diagram	

Útmutató a házi feladathoz

Javasolt eszközök

- Osztálydiagramokhoz és egyéb diagramokhoz:
 - StarUML: <http://staruml.io/>
 - WhiteStarUML: <https://sourceforge.net/projects/whitestaruml/>
- Szekvenciadiagramokhoz:
 - Web sequence diagrams:
<https://www.websequencediagrams.com/>
- Más eszköz is használható, ha az kényelmesebbnek tűnik
- Ha az eszköz nem követi a szabványos jelölést, akkor azt jelezni kell a beadott dokumentációban

Módszer

- Legfontosabb tanács: *Gondolkodjunk!*
- Meg lehet beszélni másokkal is az ötletet *általánosságban*
- De a diagramokat egyedül kell megrajzolni!
- Ne másoljunk le egymástól diagramrészeket!
- Ne rajzoljuk együtt a diagramokat!
- A plagizált megoldásokat nem fogadjuk el!

- Objektumorientált tervezés
 - a felelősségeket (metódusokat) egyenletesen osszuk szét az osztályok között
 - a játék világot modellezzük, gondoljunk arra, hogyan működik a valóság
 - a modellnek könnyen bővíthetőnek és karbantarthatónak kell lennie, ha esetleg új követelmények jönnek
- *Csak* a játék logikájára koncentráljunk
 - a modellnek legyenek olyan függvényei, amelyeket a kontroller meghívhat
 - a kontroller akkor hív függvényt, ha lenyomnak egy billentyűt, használják az egeret vagy az időzítő szól
 - ezekből a függvényekből az összes többi függvénynek elérhetőnek kell lennie a szekvenciadiagramokon keresztül
 - a kontrollert és a nézetet (grafikát) *teljesen* hagyjuk ki a modellből
 - nincs többszálúság

Elvárt megoldás

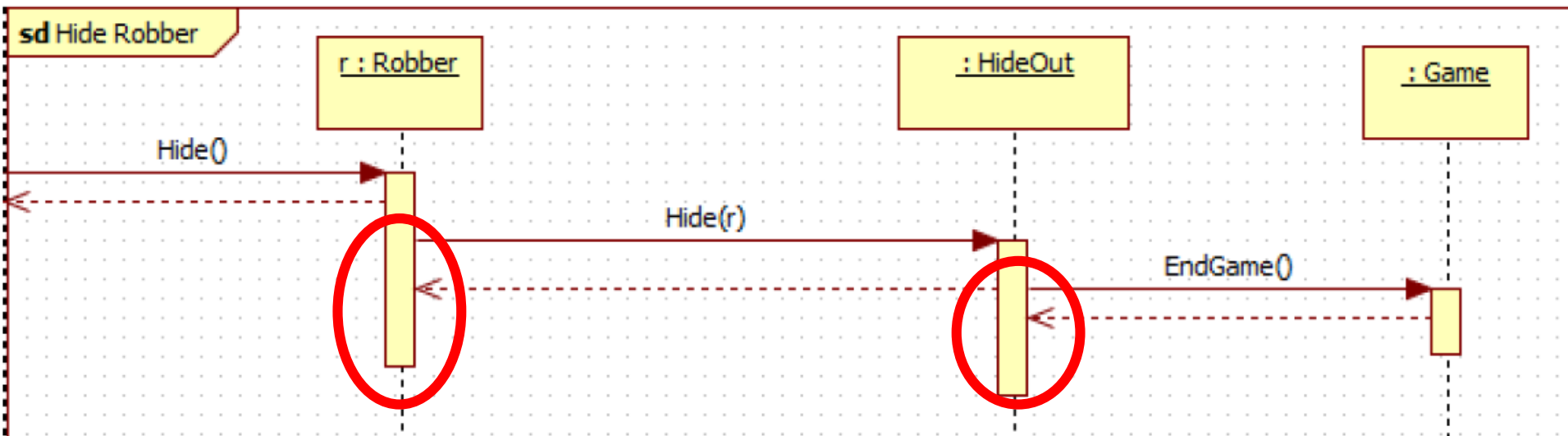
- A modellnek van egy elvárt *magja*: a legfontosabb osztályok és ezek legfontosabb kapcsolatai (függőség, asszociáció, öröklődés)
 - ügyeljünk arra, hogy ezt megtaláljuk és korrektül megoldjuk
- A diagramok legyenek konzisztensek egymással
 - a szekvenciadiagramon előforduló függvényeknek létezniük kell az osztálydiagramon
 - minden osztálydiagramon szereplő függvénynek legalább egyszer elő kell fordulnia egy szekvenciadiagramon
- Ha az osztálydiagram túl nagy, daraboljuk szét több oldalra
- Minden szekvenciadiagramnak olvashatóan el kell férnie egy oldalon
 - a szekvenciadiagramok modularizálhatók:
 - egy függvény egy másik diagramon kifejtve
 - egy másik szekvenciadiagram meghívatkozása (interakció használat)

Az osztálydiagram halálfejes hibái

- Az elvárt mag osztályai vagy azok kapcsolatai hiányoznak
- Az osztálydiagram nem alkot összefüggő gráfot
 - ha az osztályok nem ismerik egymást, a modell nem működik
- Nem objektumorientált megoldás
 - a felelősségek nincsenek egyenletesen szétosztva
 - mindent egyetlen osztály csinál
 - sok olyan osztály van, amelyeknek csak attribútumai és getter-setter-ei vannak
 - típusellenőrzés:
 - típuskasztlás, instanceof, is, stb.
 - típus int-ként, enum-ként, stb. kódolva
 - képességlekérdező függvények: is...(), can...()
- Konkrét osztálynak üres leszármazottai vannak
- Szintaktikailag hibás diagram
 - pl. hibás öröklődés, függőség, asszociáció, kompozíció, stb.
- Kontroller, szálak, grafika, koordináták, stb. a modellben

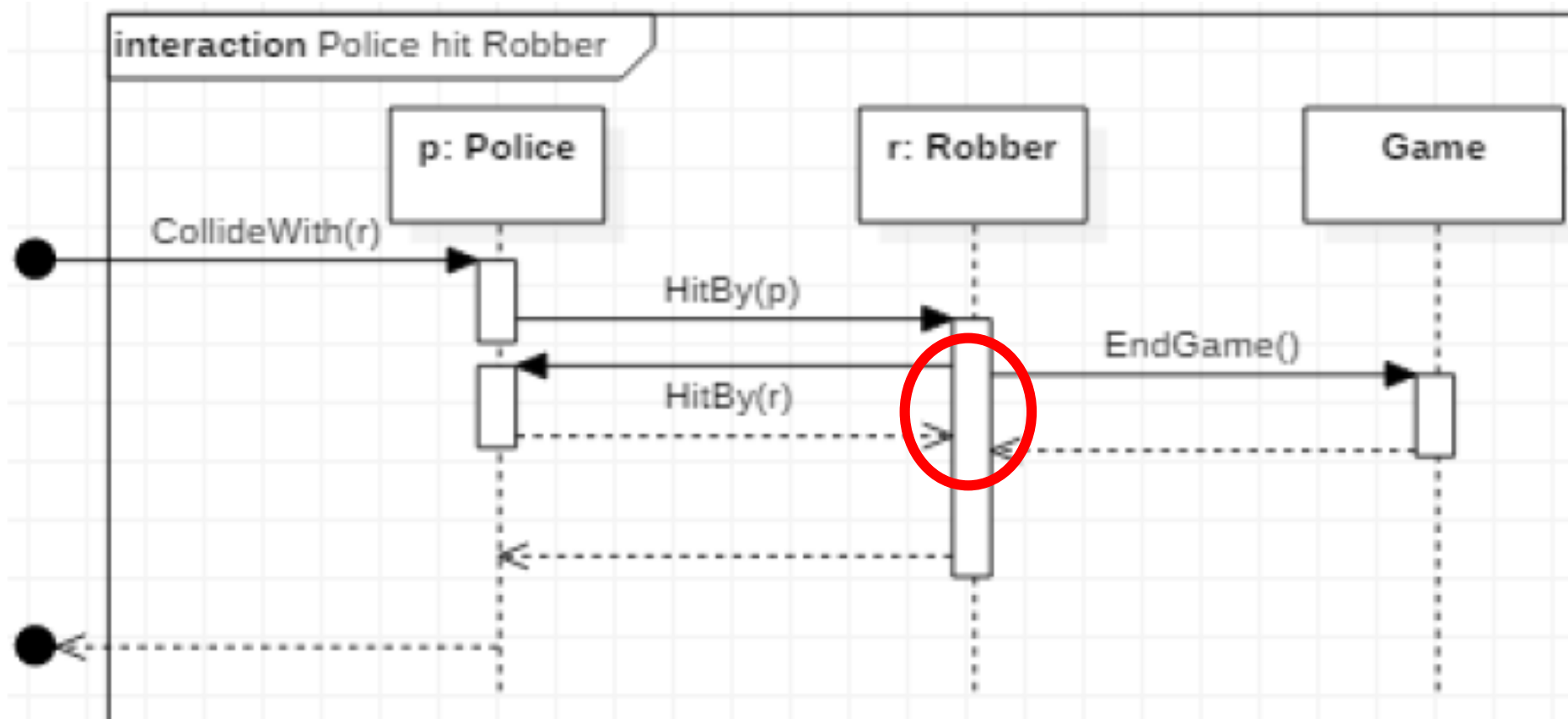
A szekvenciadiagram halálfejes hibái

Q1. Probléma I.



Hiba: befejezett függvény hív új függvényt

Q1. Probléma II.



Hiba: a függvény két másik függvényt hív egyszerre

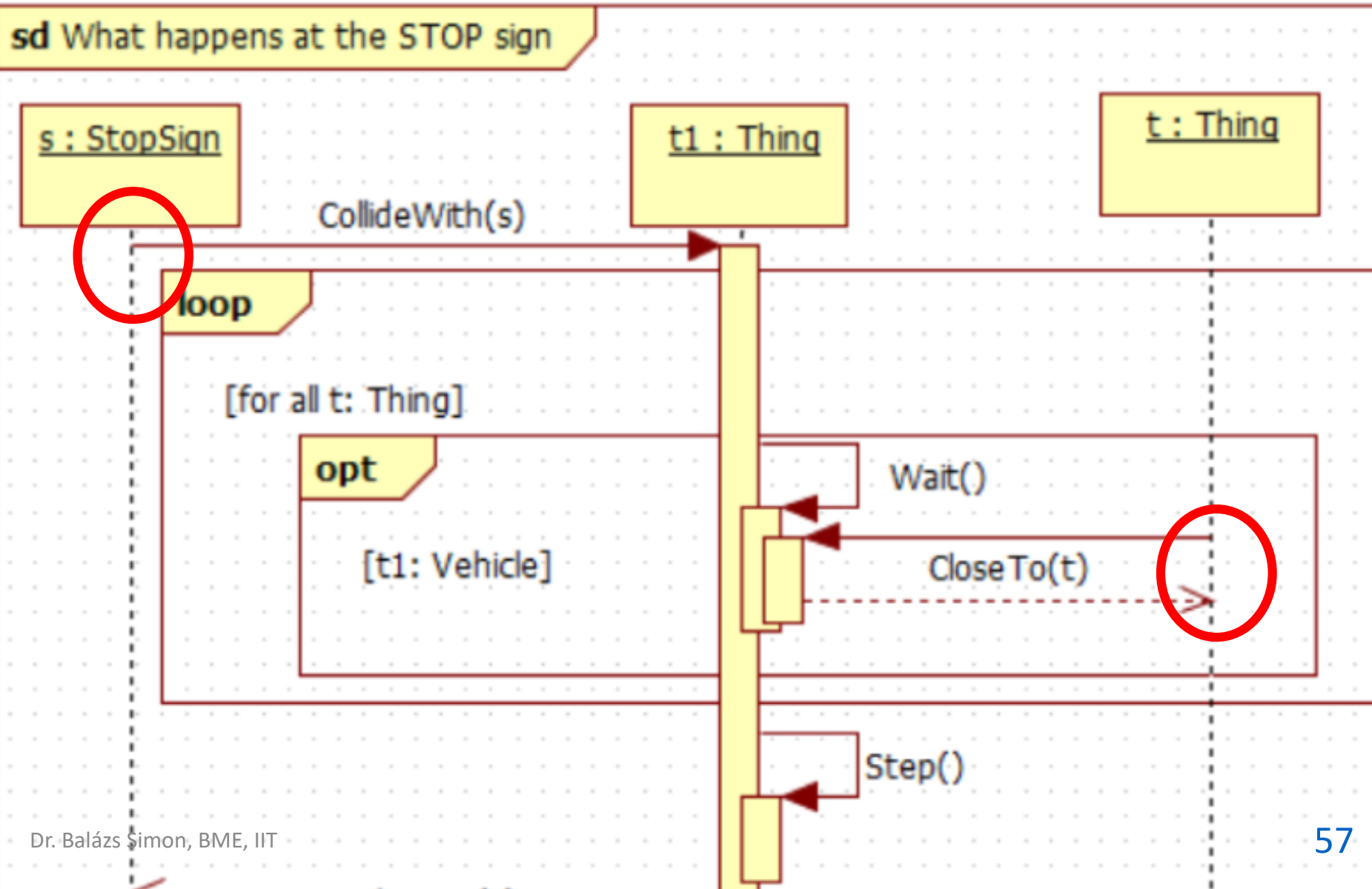
Q1. Szinkron függvényhívások egymásba ágyazása

- Probléma ha megszegjük:
 - inkonzisztens diagramok
 - a függvényeket nem lehet implementálni
- Szabály:
 - a szinkron függvényhívásokat egymásba kell ágyazni
 - az egymásba ágyazás a hívási vermet (call stack) követi
- Kivétel:
 - többszálú működés ábrázolása
 - de a hívásoknak *szálanként* továbbra is megfelelően egymásba kell ágyazódniuk

Q2. Probléma

Hibák:

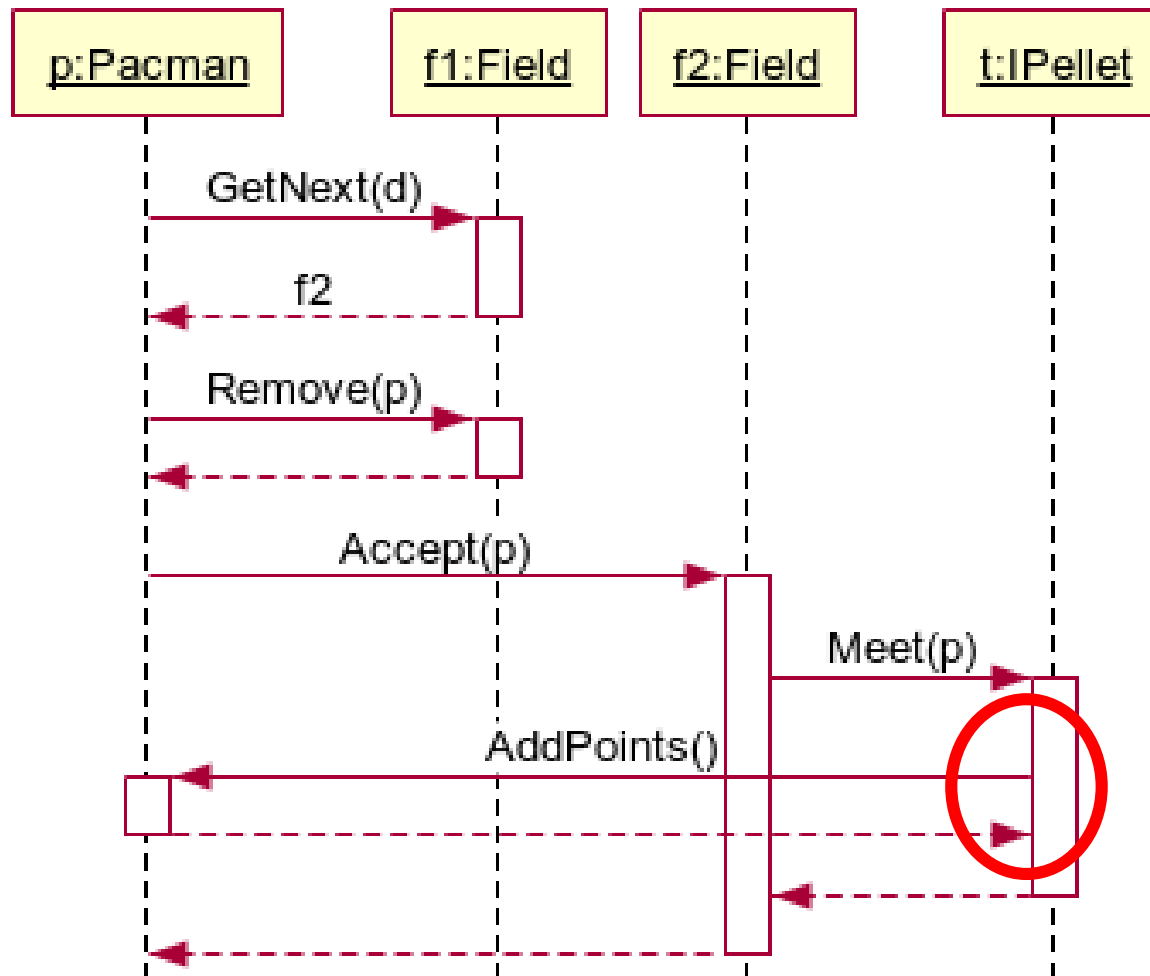
- inaktív lifeline kezdeményez hívást
- a szekvencia két külön pontból is elindul



Q2. Csak aktív lifeline kezdeményez hívást

- Probléma ha megszegjük:
 - inkonzisztens diagramok
 - a függvényeket nem lehet implementálni
- Szabály:
 - csak futó függvény kezdeményezhet hívást
 - a szekvenciadiagramok pontosan egy belépési ponttal rendelkeznek
 - kivéve, ha több szál van
- Kivétel:
 - az UML eszköz lehet, hogy nem támogatja az execution specification jelölést:
 - az első lifeline-ra
 - egy másik szálra
 - (de a szabály továbbra is él, még ha az execution specification a diagramon nem is látható)

Q3. Probléma

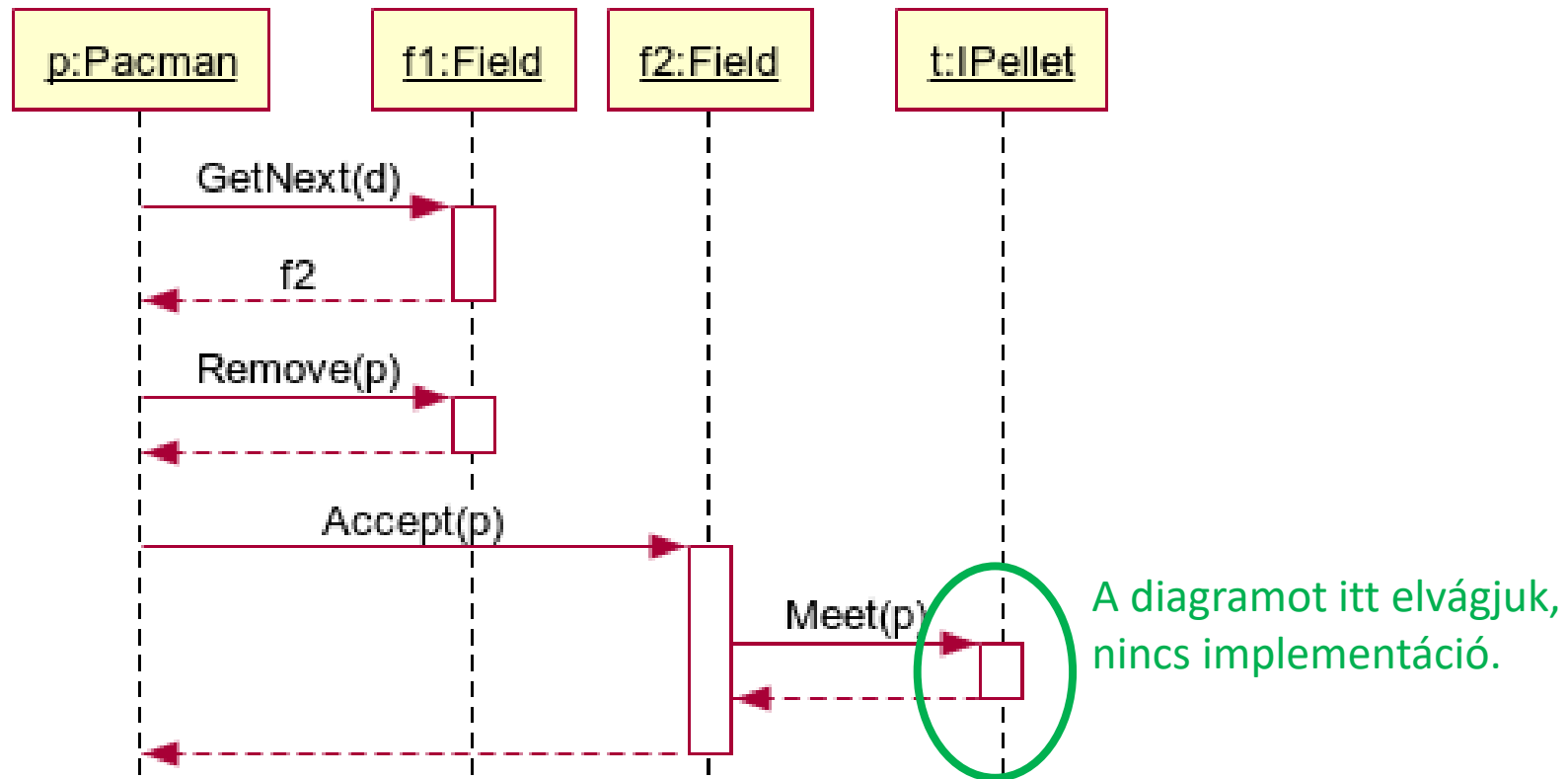


Hiba: interfész kezdeményez hívást

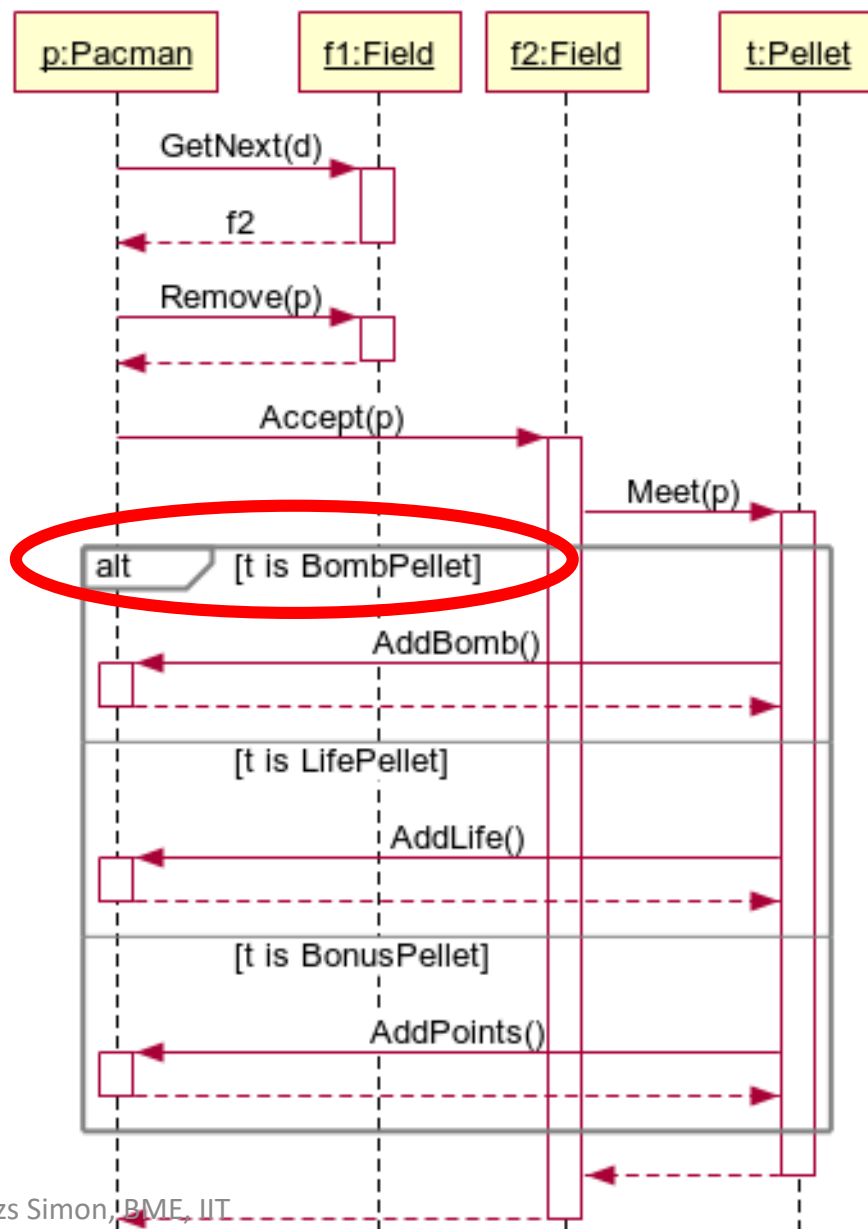
Q3. Csak konkrét függvények kezdeményezhetnek hívást

- Probléma ha megszegjük:
 - a diagram absztrakt függvénynek vagy interfész egy függvényének működését mutatja
- Szabály:
 - interfészek és absztrakt függvények nem kezdeményezhetnek hívást
 - csak konkrét függvény kezdeményezhet hívást
 - a polimorfikus viselkedést ne az interfészen vagy absztrakt függvényen belül mutassuk be

Q3. Megoldás



Q4. Probléma



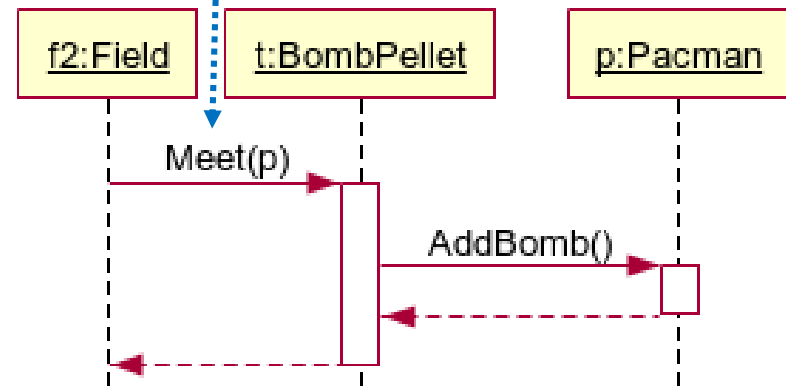
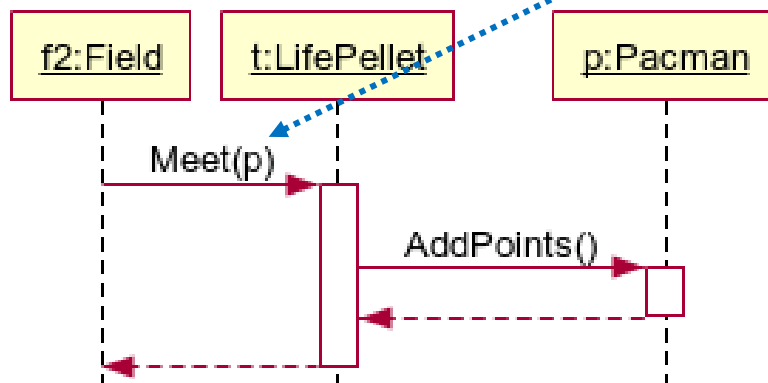
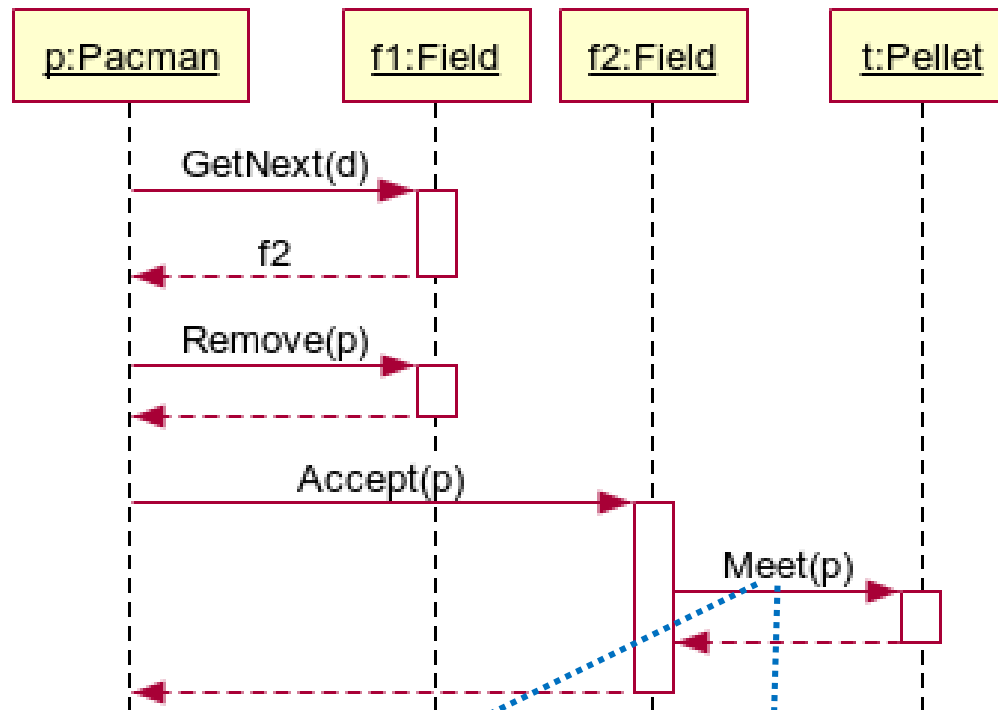
Hibák:

- típusellenőrzés
- polimorfikus viselkedés alt-ként jelölve

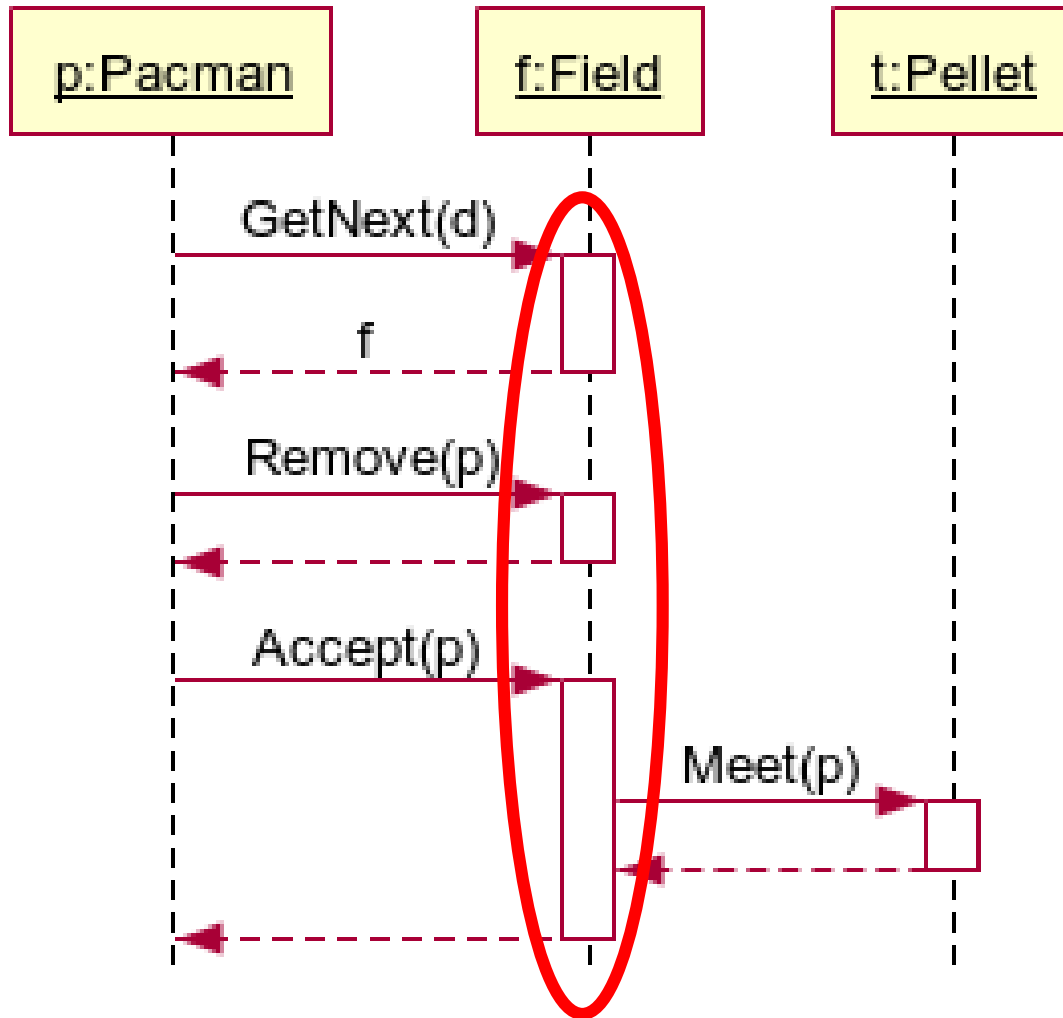
Q4. A polimorfikus viselkedést külön diagramon jelöljük

- Probléma ha megszegjük:
 - bonyolult diagramok
 - a polimorfikus viselkedés **alt**-ként történő ábrázolása típusellenőrzést jelent: az OCP elv megszegése
 - a programozási nyelv belső implementációját ne ábrázoljuk
- Szabály:
 - a hívást a polimorfikus hívásnál fejezzük be
 - akkor is, ha interfészhez vagy absztrakt függvényhez érünk
 - rajzoljunk külön diagramokat a leszármazott típusokra
 - a diagramok a polimorfikus hívástól kezdődjenek

Q4. Megoldás



Q5. Probléma

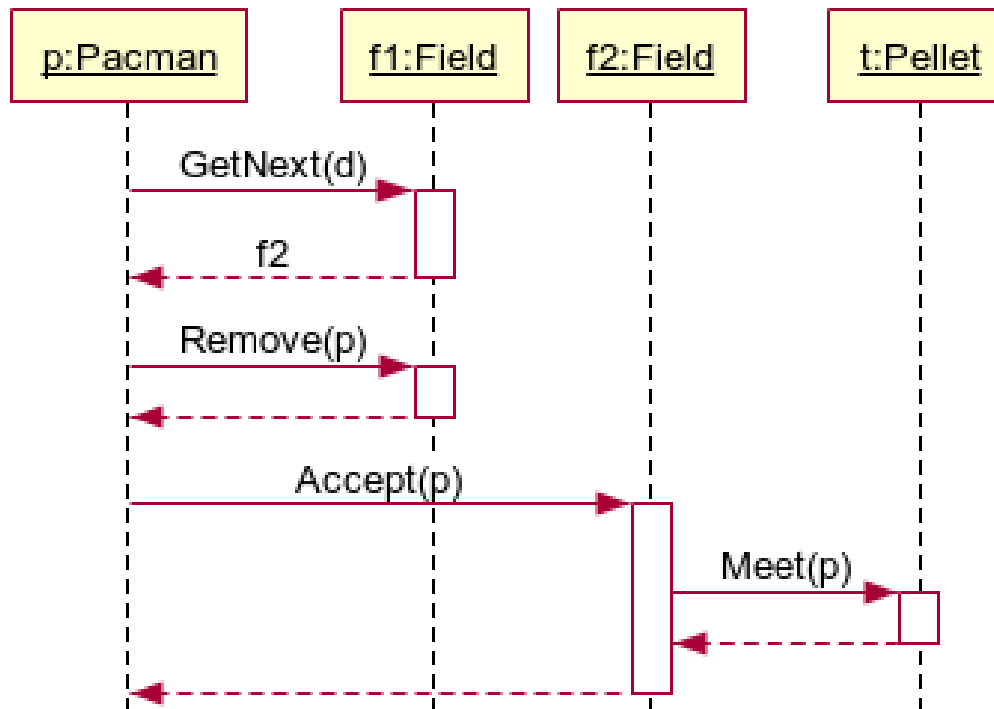


Hiba: különböző objektumok közös életvonalon ábrázolva

Q5. Különböző objektumoknak különböző lifeline-ok kellenek

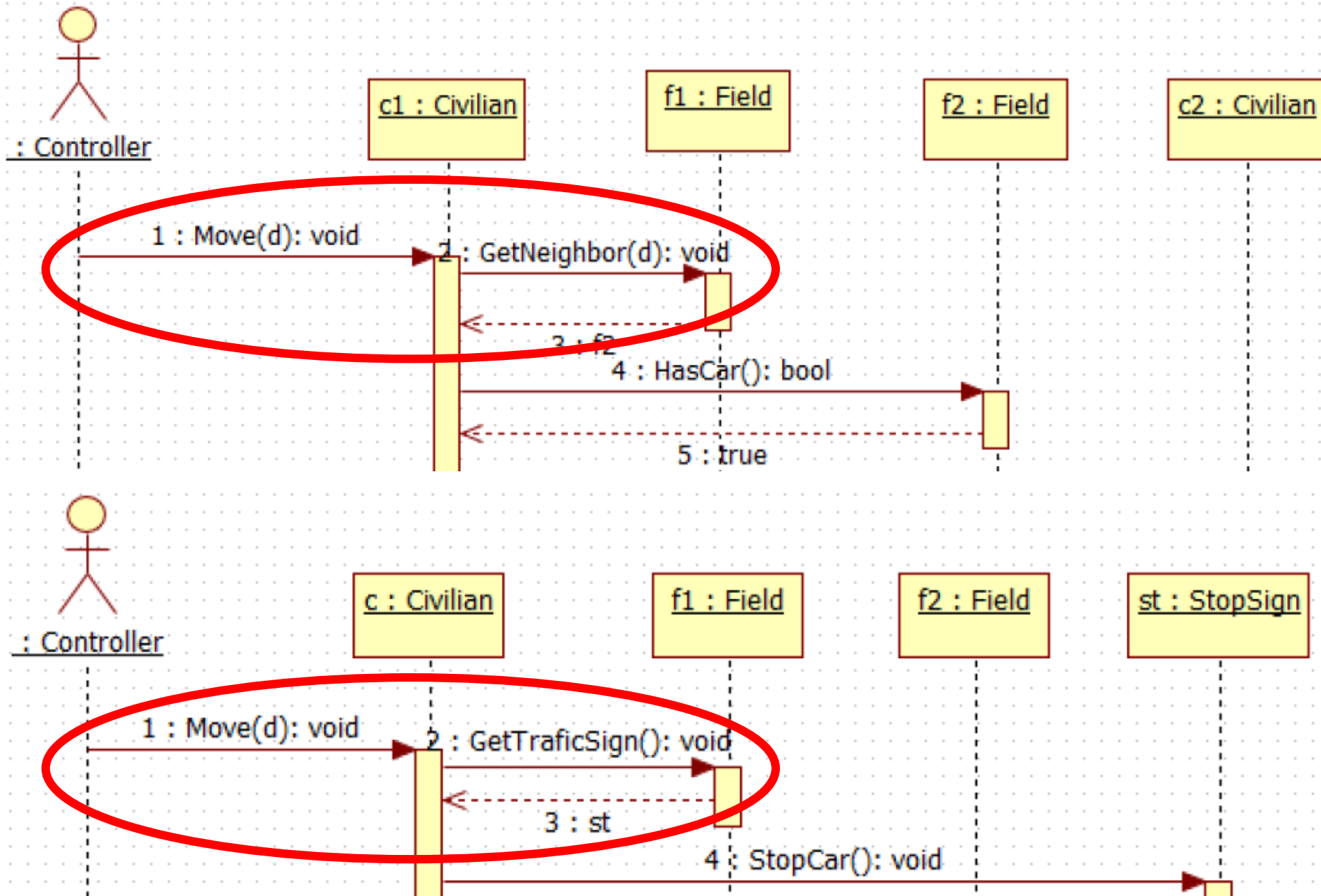
- Probléma ha megszegjük:
 - a függvényhívásokat nem lehet szétválogatni
 - a viselkedés nincs megfelelően definiálva
- Szabály:
 - különböző objektumokhoz különböző lifeline-okat rendelünk, még akkor is, ha a típusuk ugyanaz
 - adjunk nekik különböző neveket

Q5. Megoldás



Q6. Probléma

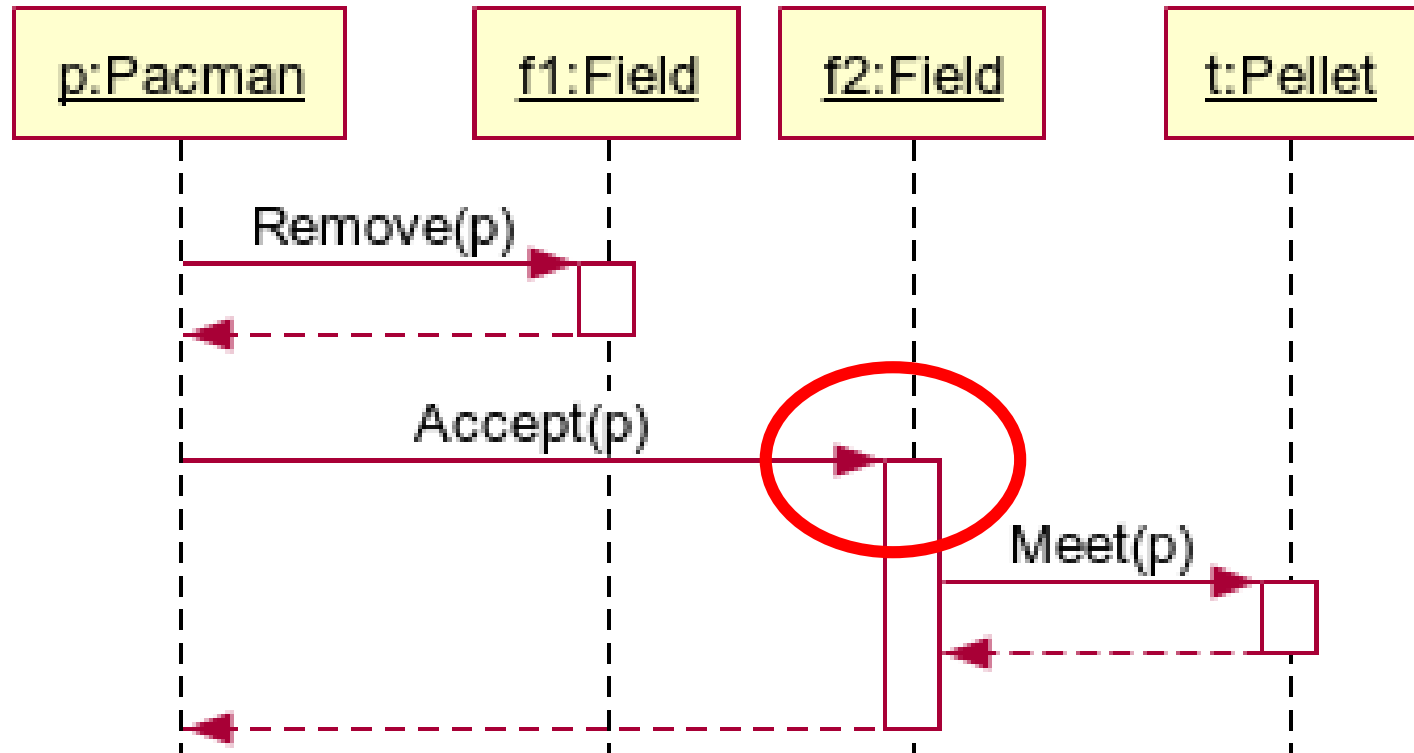
Hiba: ugyanaz a függvény két külön viselkedéssel



Q6. Diagramok között is konzisztens függvények

- Probléma ha megszegjük:
 - a szekvenciák nem implementálhatók
 - ugyanaz a függvény nem rendelkezhet két különböző viselkedéssel
- Szabály:
 - ugyanaz a függvény ugyanúgy viselkedjen minden diagramon
- Kivétel:
 - ugyanaz a függvény mutathat különböző viselkedést az objektum belső állapota vagy egyéb feltételek alapján, de ezeknek a viselkedéseknek *használnak* kell lenniük
 - és nem mondhatnak ellent egymásnak

Q7. Probléma

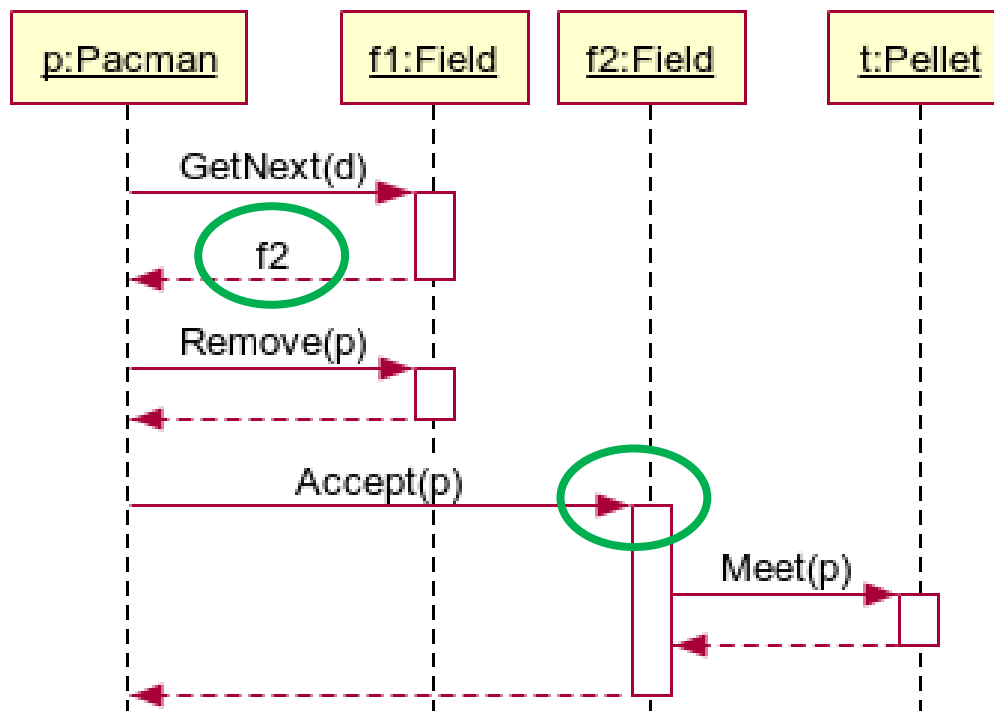


Hiba: olyan objektumon hívunk függvényt, akit nem ismerünk

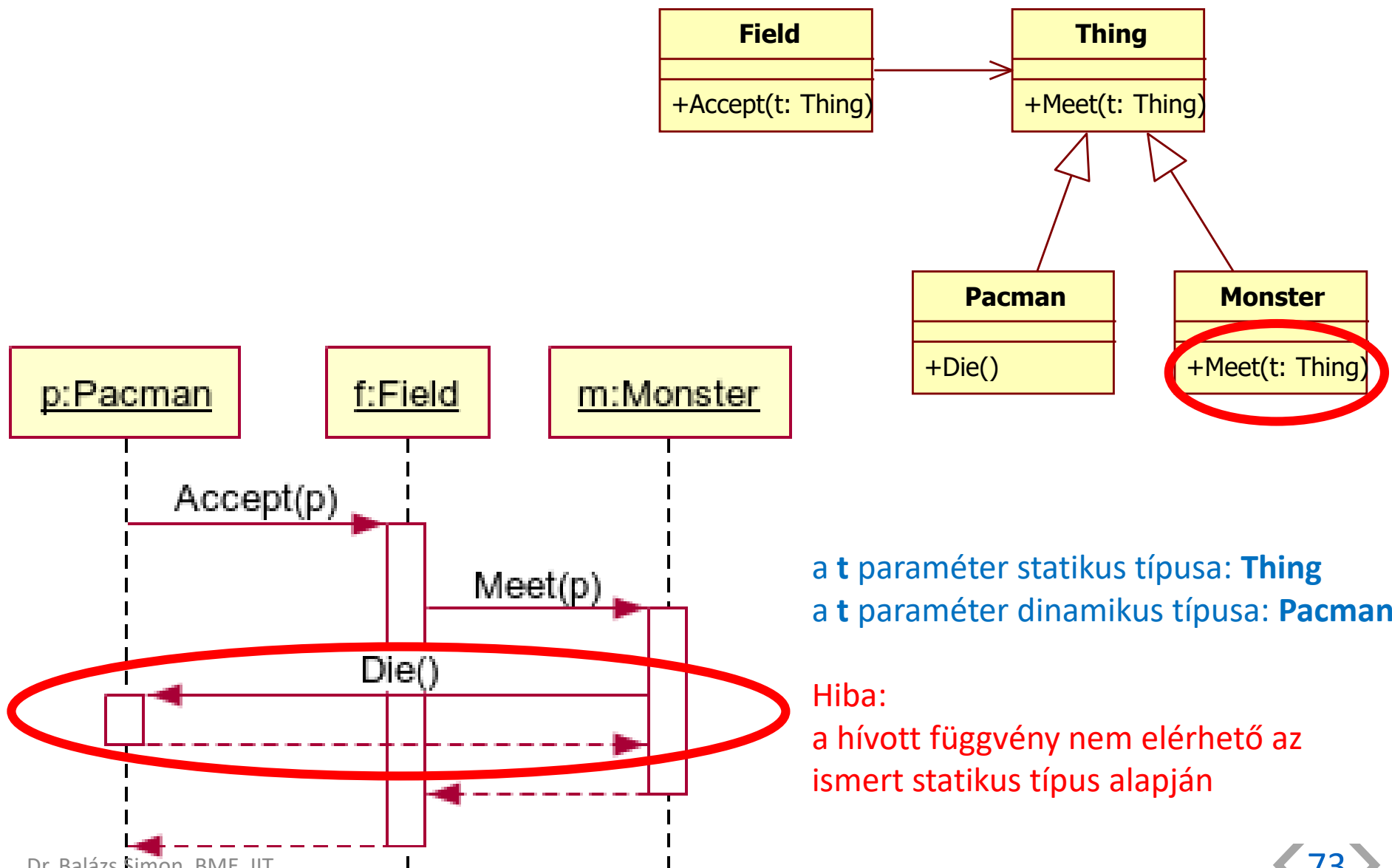
Q7. A hívónak ismernie kell a hívott objektumot

- Probléma ha megszegjük:
 - a szekvencia nem implementálható
- Szabály:
 - a hívónak ismernie kell a hívott objektumot
 - vagy van rá asszociációja
 - vagy az alábbi módokon ismeri meg (függőség):
 - paraméterként kapja
 - egy korábbi hívás visszatérési értékeként kapja
 - ő hozza létre

Q7. Megoldás



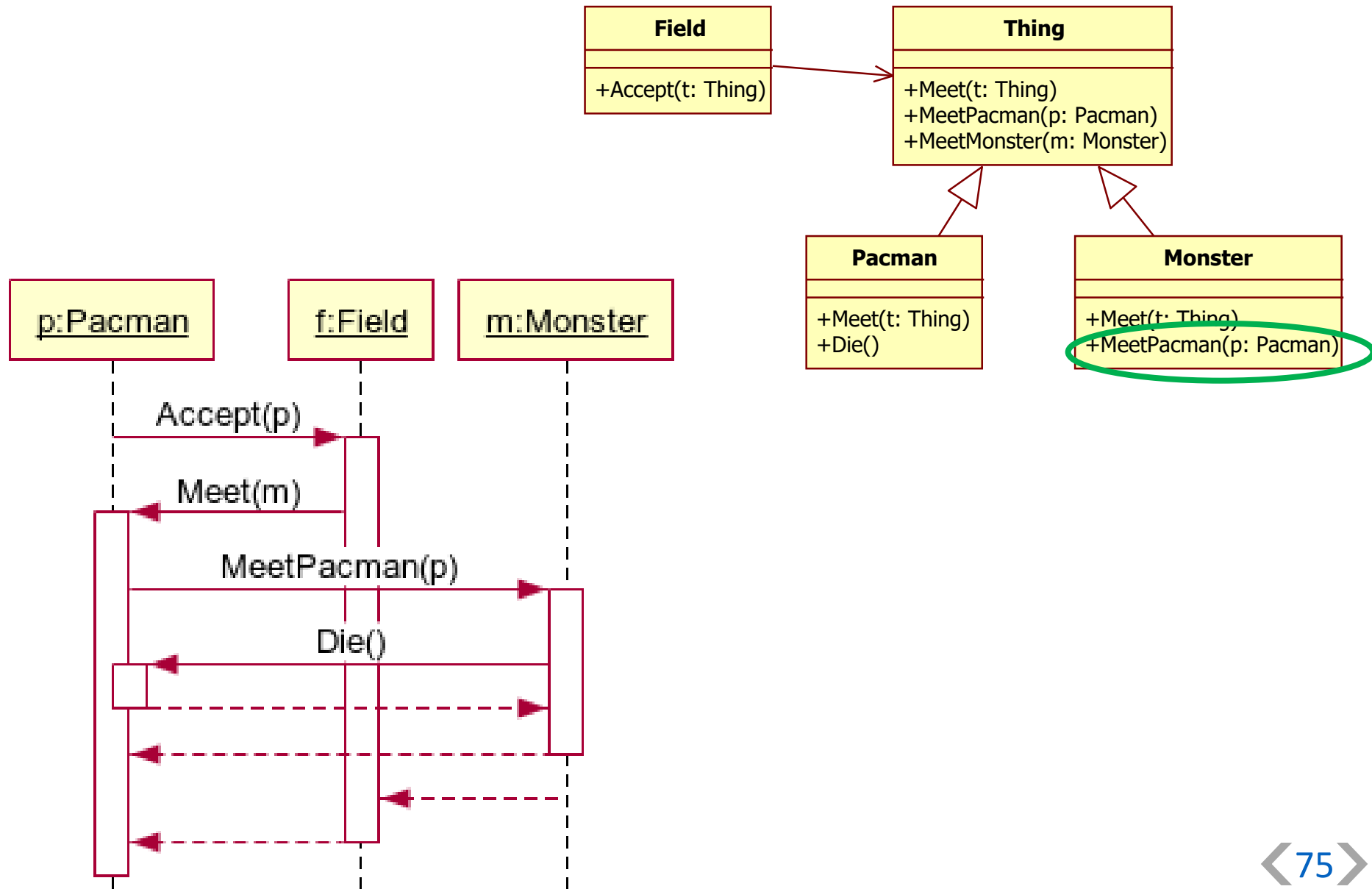
Q8. Probléma



Q8. Csak a statikus típus alapján látható függvények hívhatók

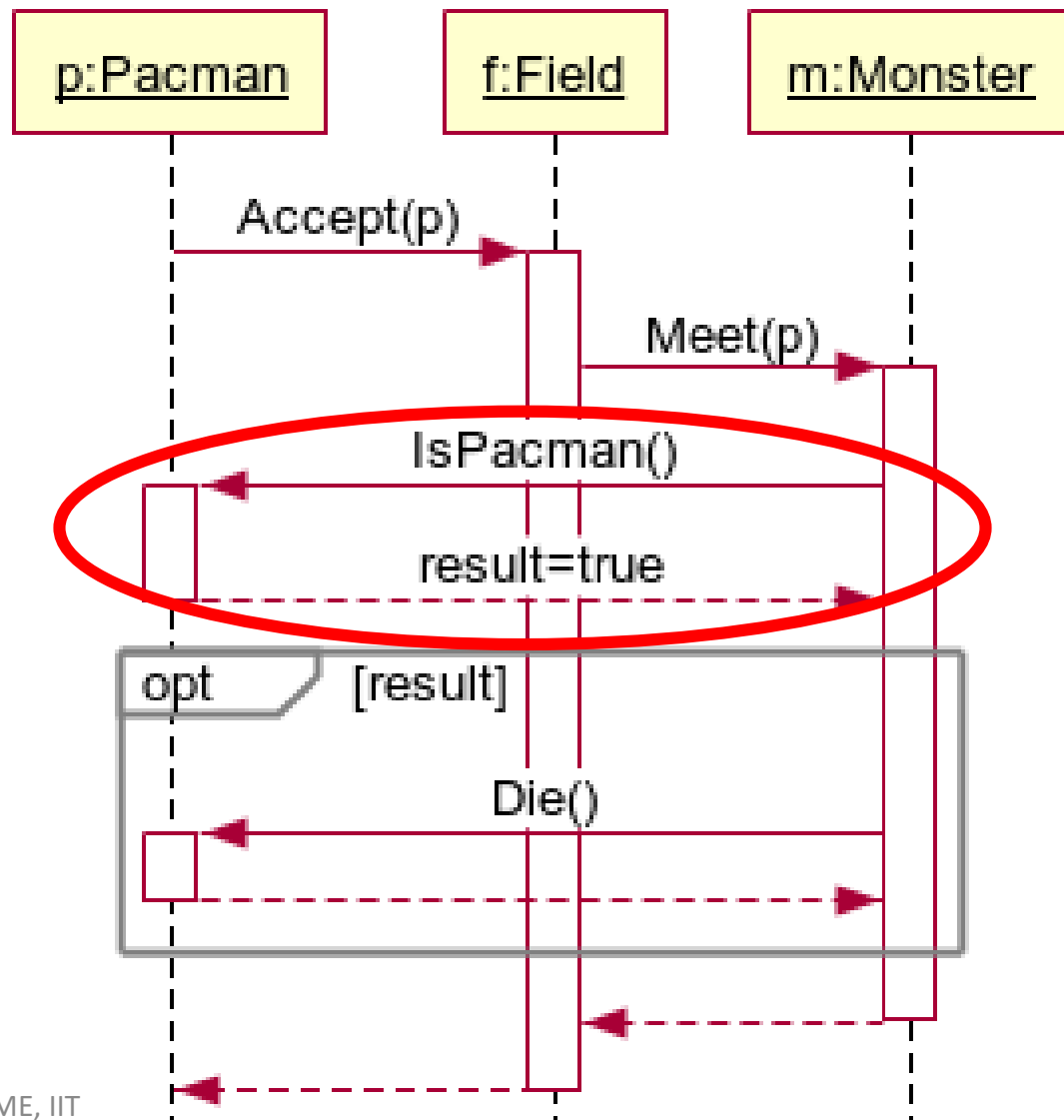
- Probléma ha megszegjük:
 - a szekvencia nem implementálható
 - a típuskasztolás megszegi az LSP és az OCP elveket
- Szabály:
 - ellenőrizzük, hogy az ismert statikus típuson keresztül meghívható-e a függvény

Q8. Megoldás



Q9. Probléma

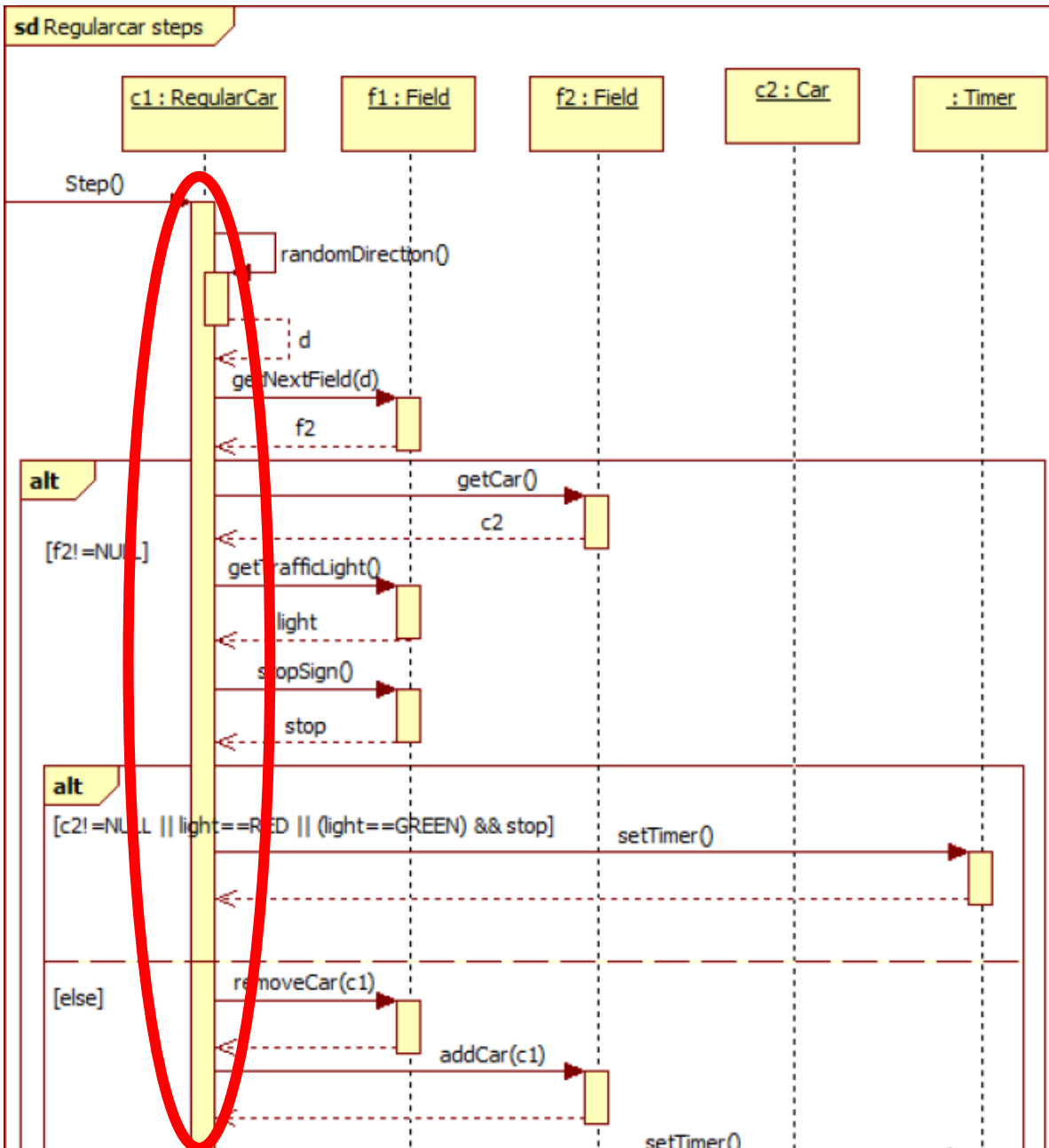
Hiba: a TDA elv megsértése



Q9. Tell, Don't Ask (TDA)

- Probléma ha megszegjük:
 - a feltétel ellenőrzése lemaradhat más helyeken
 - a DRY elv megsértése
 - ha a típust ellenőrizzük: az OCP elv megsértése
- A TDA elv megsértése azt jelenti, hogy a felelősség rossz helyen van
- Szabály:
 - ne kérdezzük le a hívott objektum típusát
 - ne kérdezzük le a hívott objektum állapotát
 - hagyjuk, hogy a hívott objektum saját maga viselkedjen a saját típusa és saját belső állapota alapján
- Kivétel:
 - előfeltételek ellenőrzése egy függvény meghívása előtt
 - pl. kivétel elkerülése üres veremből olvasáskor

Q10. Probléma

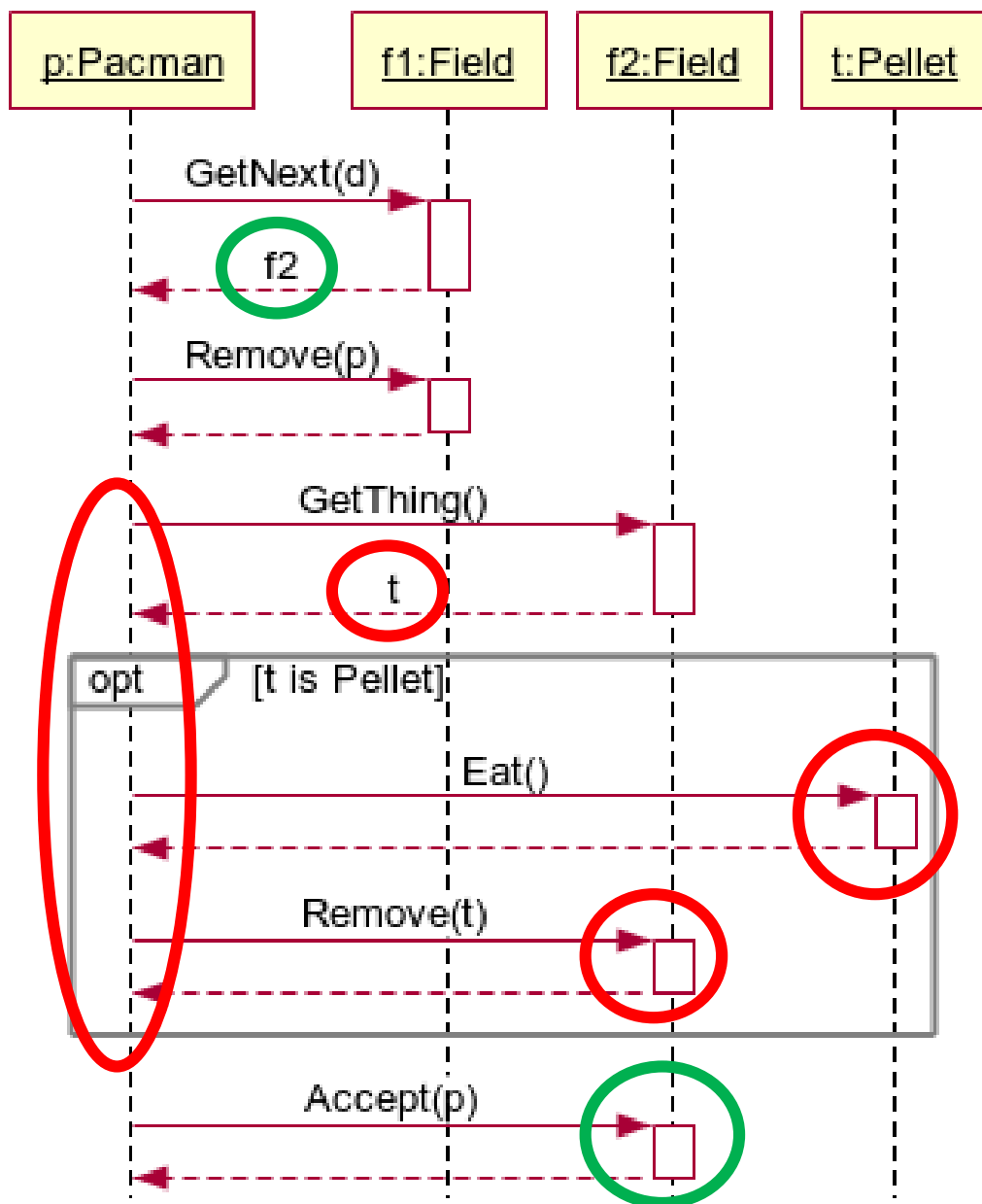


Hiba: "Karácsonyfa"
egyetlen objektum csinál mindent

Q10. A felelősségeket egyenletesen osszuk szét

- Probléma ha megszegjük:
 - isten-osztály
 - felelőségek nem a megfelelő helyen
- Szabály:
 - a felelősségeket egyenletesen osszuk szét
 - kerüljük az isten-osztályokat
- Kivétel:
 - a szekvencia diagramok lehetnek „Karácsonyfák”, de a fa törzse ne mindig ugyanaz legyen
 - ha az összes diagramot összekombinálnánk, akkor már nem egy törzsű fának nézne ki

Q11. Probléma



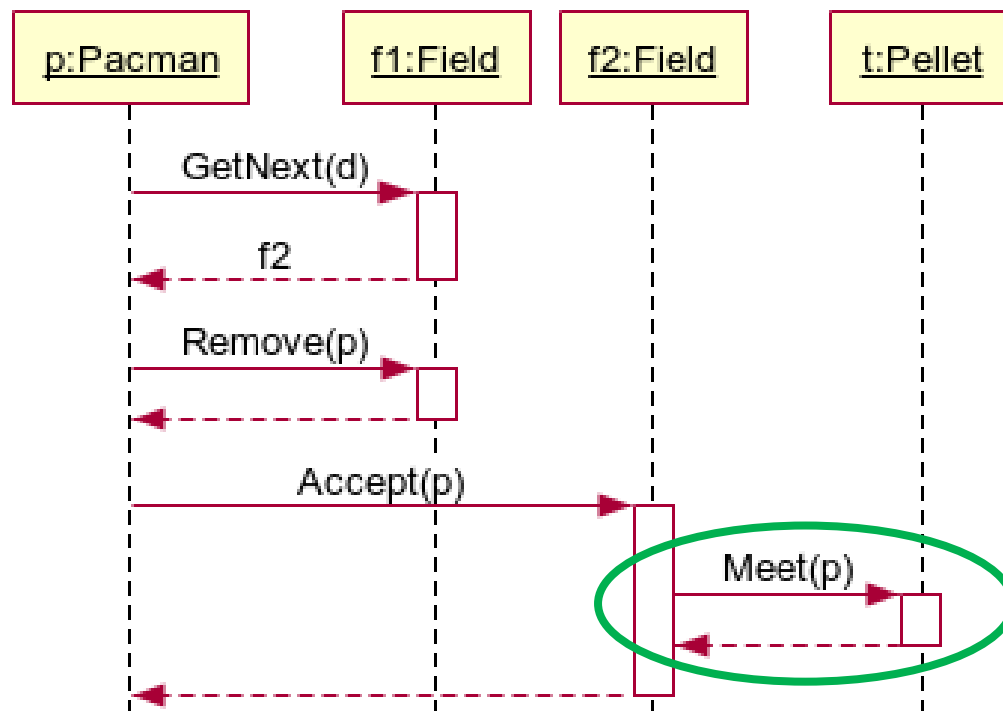
Itt a LoD és TDA megsértése rendben van, mert a Pacman magától lép, nem a következő mező „húzza”

Hiba: “Karácsonyfa”
Itt a LoD megsértése nincs rendben, mert a világ nem így működik

Q11. Demeter törvény

- Probléma ha megszegjük:
 - “Karácsonyfa” szekvenciadiagram: a függvényhívások láncolása azt jelenti, hogy a lánc minden egyes elemétől függünk
 - akkor is, ha a köztes hívások eredményeit külön lokális változóknak tároljuk
- Szabály:
 - ne álljunk szóba idegenekkel
 - csak ismerősökkel:
 - saját magunk
 - paraméterként kapott objektumok
 - saját attribútumainkban tárolt objektumok
 - objektumok, akiket mi hoztunk létre
 - minden köztes objektumban legyen egy függvény, aki továbbítja a kérést a lánc következő tagjának
 - modellezzük a valós világot
- Kivétel:
 - ha az új függvények bevezetése azok kombinatorikus robbanásával járna
 - ha a világ máshogy működik

Q11. Megoldás



Az **f2** mező delegálja tovább a hívást: a Pacman nem ismeri a Pellet-et közvetlenül

Összefoglalás

- Objektumorientált megoldás legyen
- Gondoljuk alaposan át
- A szekvenciákból össze lehessen legózni a teljes működést