Analízis modell tipikus hibái

Szoftver projekt laboratórium

BME, IIT

Elvárt megoldás

- Objektumorientált tervezés
  - a felelősségeket (metódusokat) egyenletesen osszuk szét az osztályok között
  - a játék világát 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

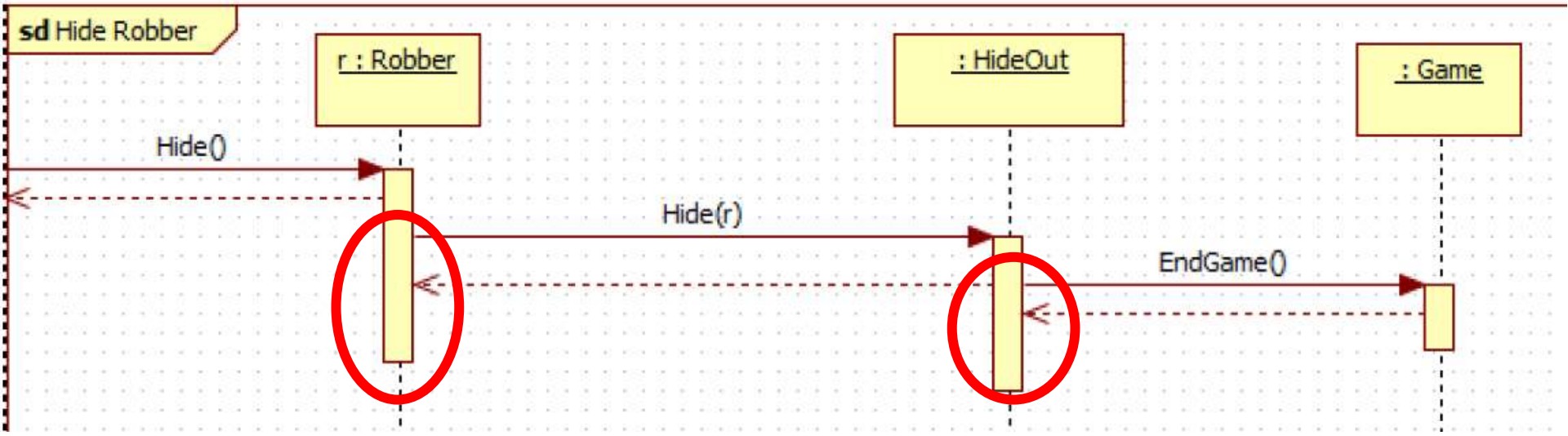
- A modellnek van egy elvárt *magja*: a legfontosabb osztályok és ezek legfontosabb kapcsolatai (függőség, asszociáció, öröklődés)
  - ügyeljünk arra, hogy ezt megtaláljuk és korrektül megoldjuk
- A diagramok legyenek konzisztensek egymással
  - a szekvenciadiagramon előforduló függvényeknek léteznie 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 meghivatkozása (interakció használat)

Az osztálydiagram súlyos 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:
    - típuskasztolás, 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

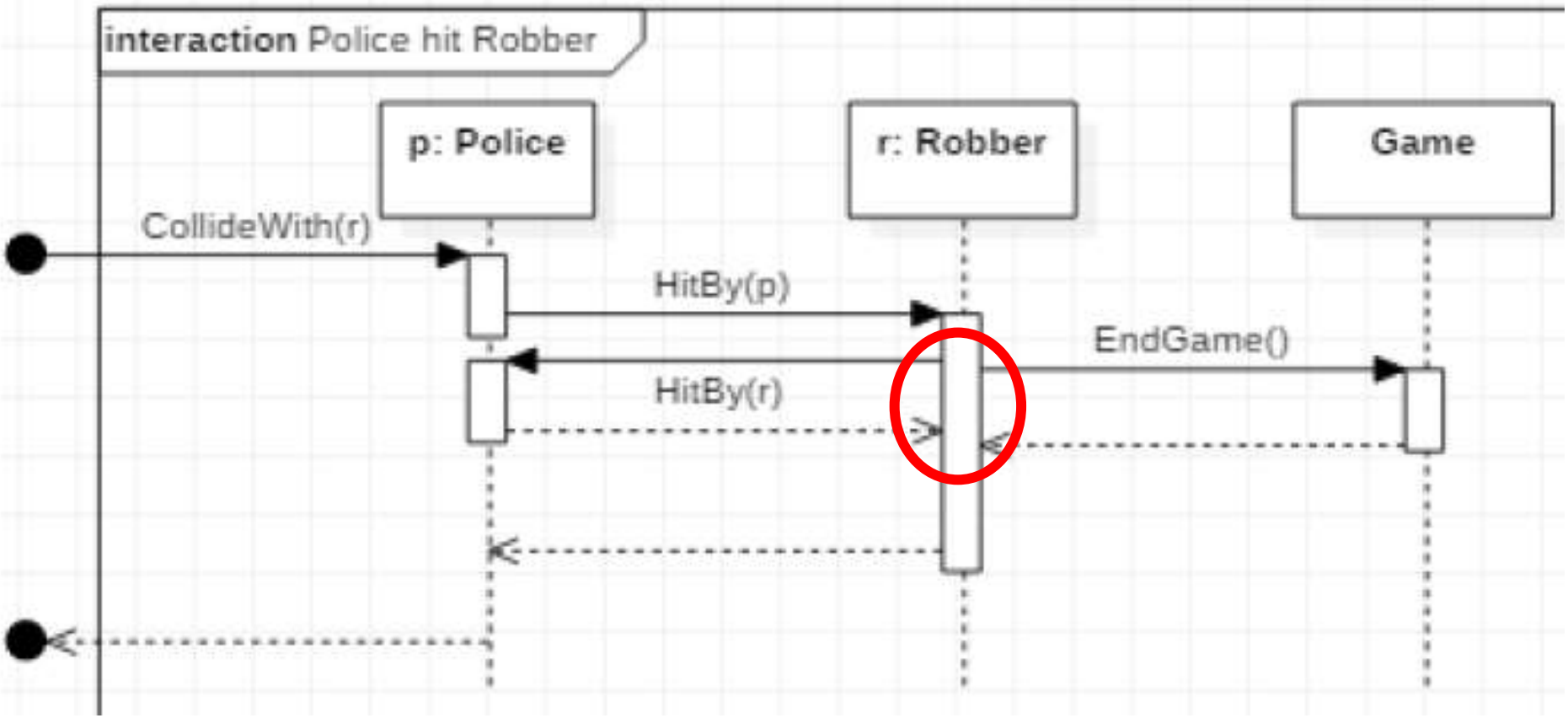
A szekvenciadiagram súlyos hibái

Q1. Probléma I.



Hiba: befejezett függvény hív új függvényt

Q1. Probléma II.



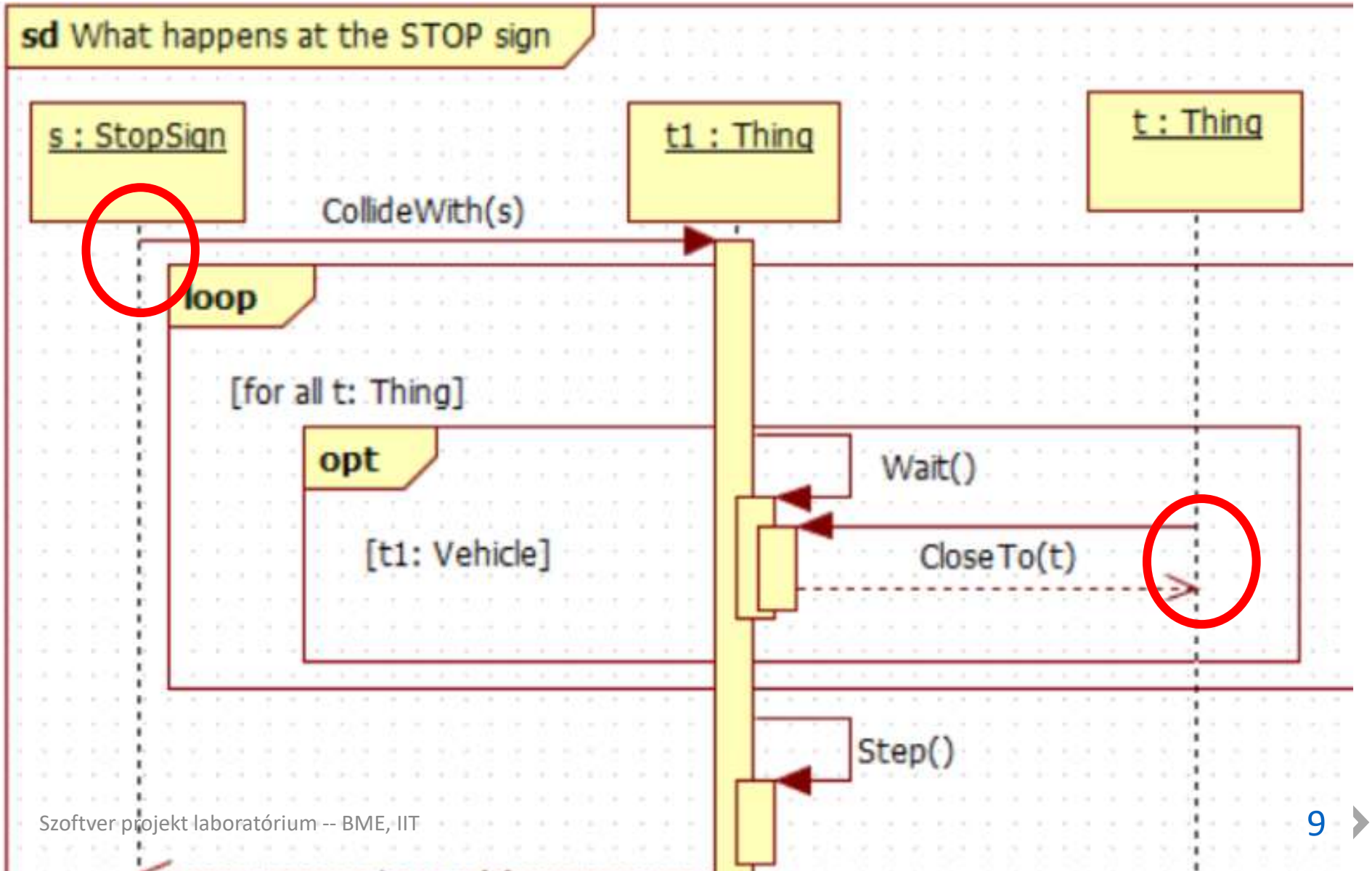
Hiba: a függvény két másik függvényt hív egyszerre

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

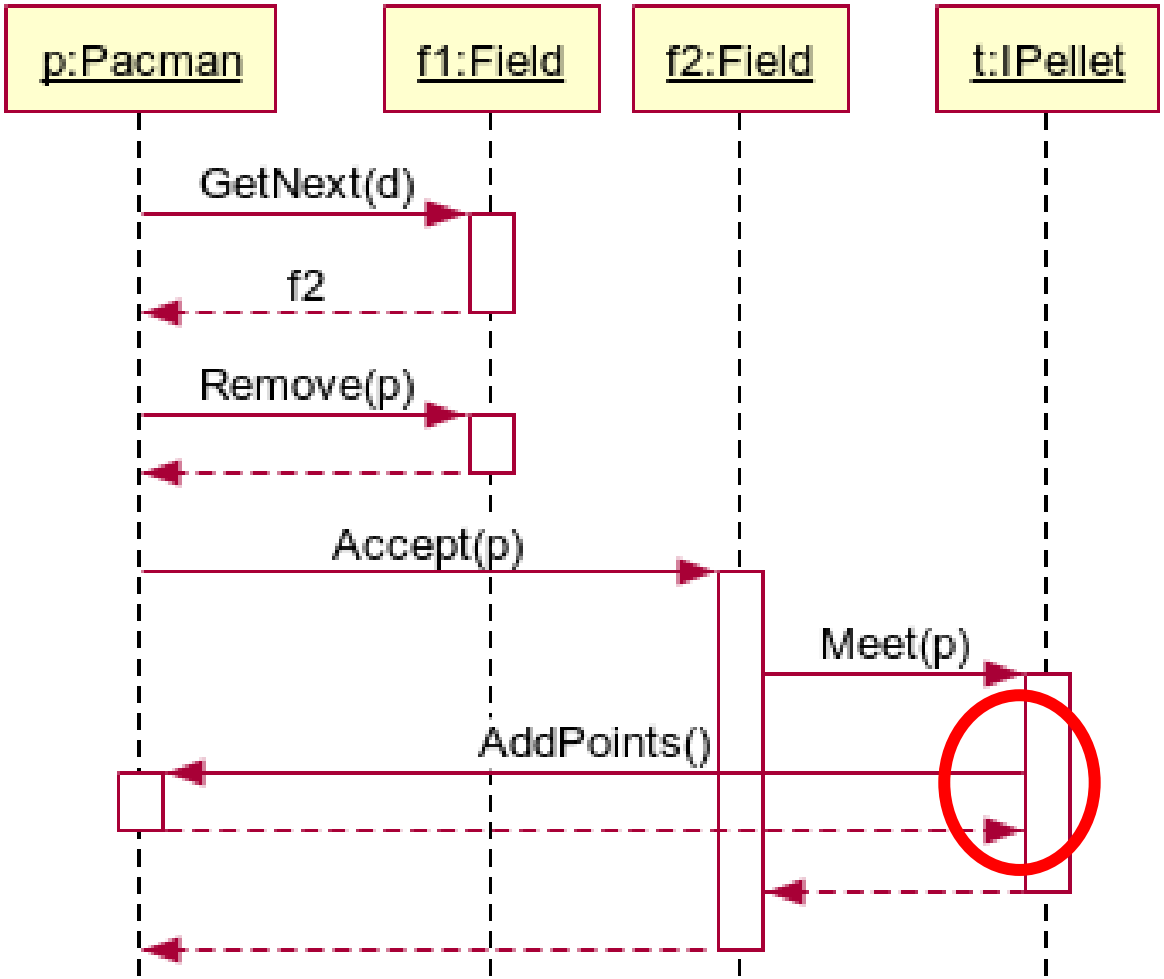
Hibák:
- inaktív lifeline kezdeményez hívást
- a szekvencia két külön pontból is elindul



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

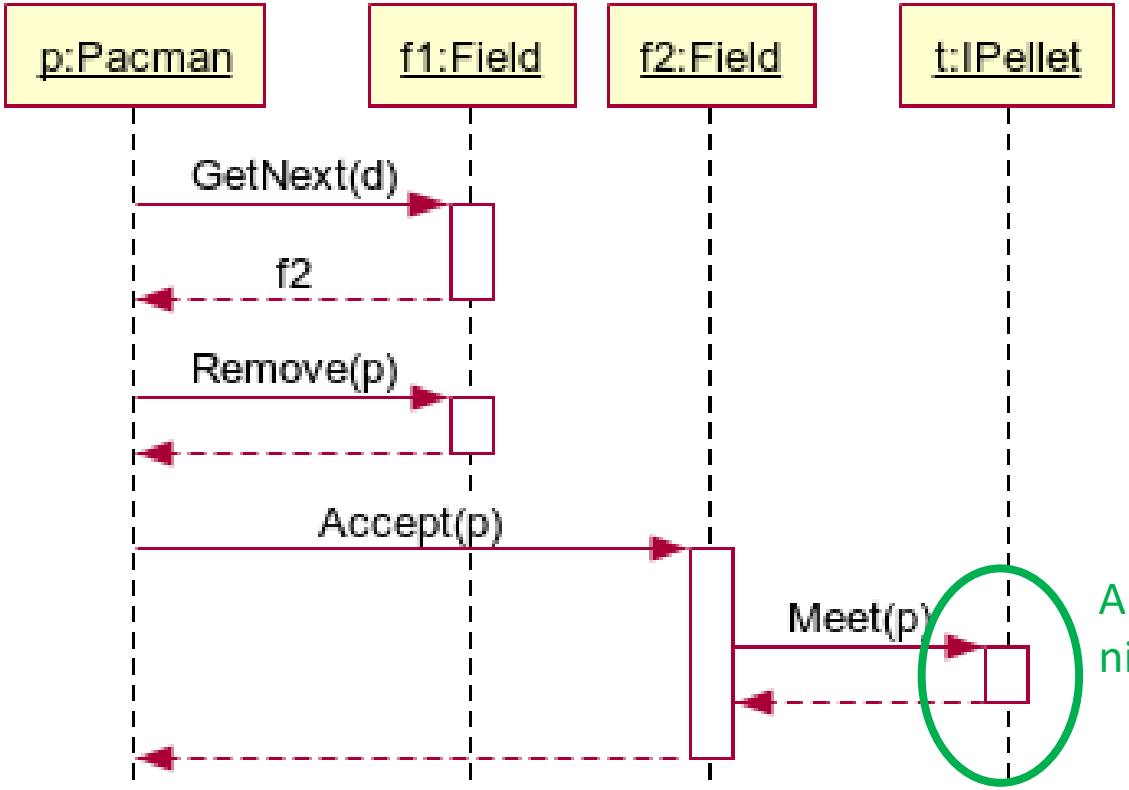


Hiba: interfész kezdeményez hívást

Q3. Csak konkrét függvények kezdeményezhetnek hívást

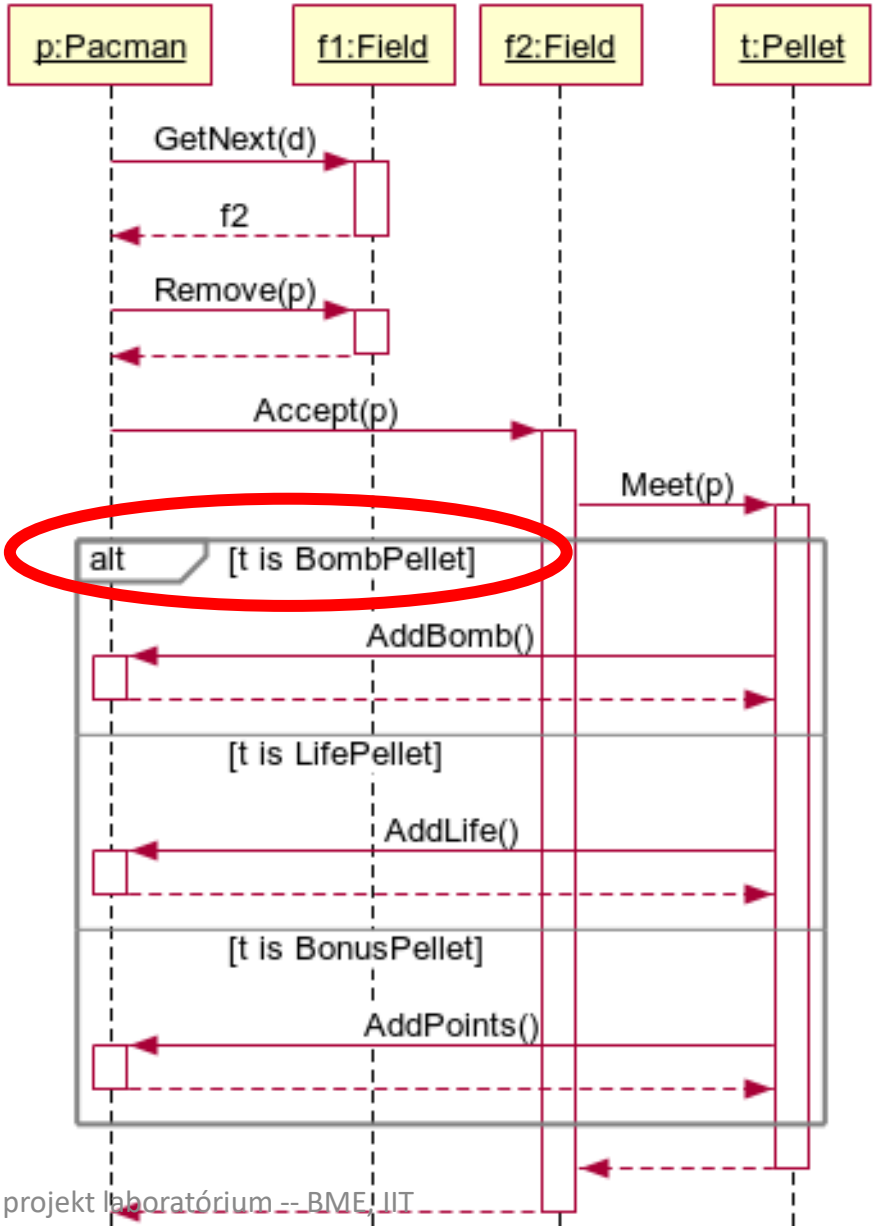
- Probléma ha megszegjük:
  - a diagram absztrakt függvénynek vagy interfész egy függvényének működését mutatja
- Szabály:
  - interfészek és absztrakt függvények nem kezdeményezhetnek hívást
  - csak konkrét függvény kezdeményezhet hívást
  - a polimorfikus viselkedést ne az interfészen vagy absztrakt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

Q3. Megoldás



A diagramot itt elvágjuk, nincs implementáció.

Q4. Probléma

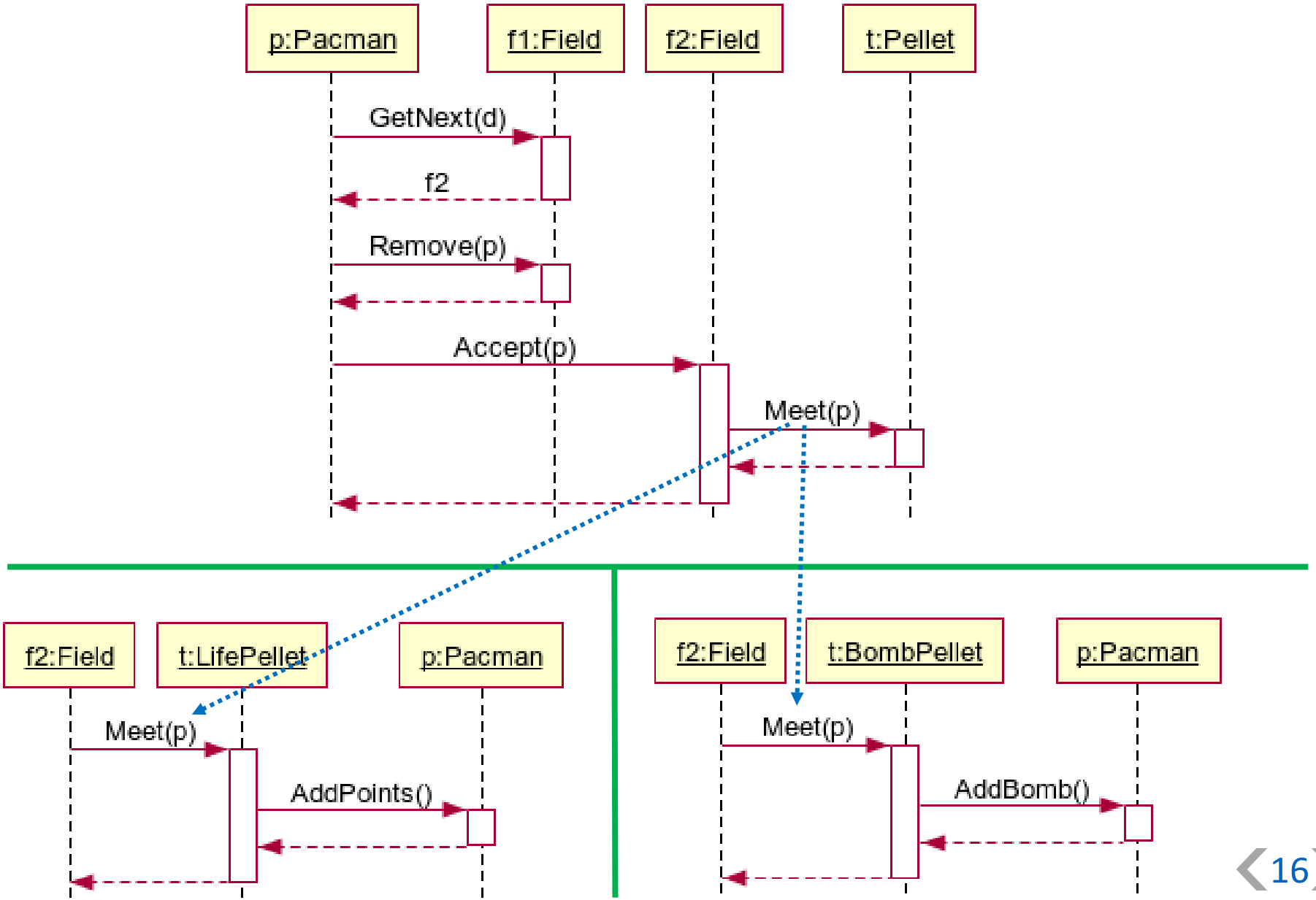


Hibák:
- típusellenőrzés
- polimorfikus viselkedés alt-ként jelölve

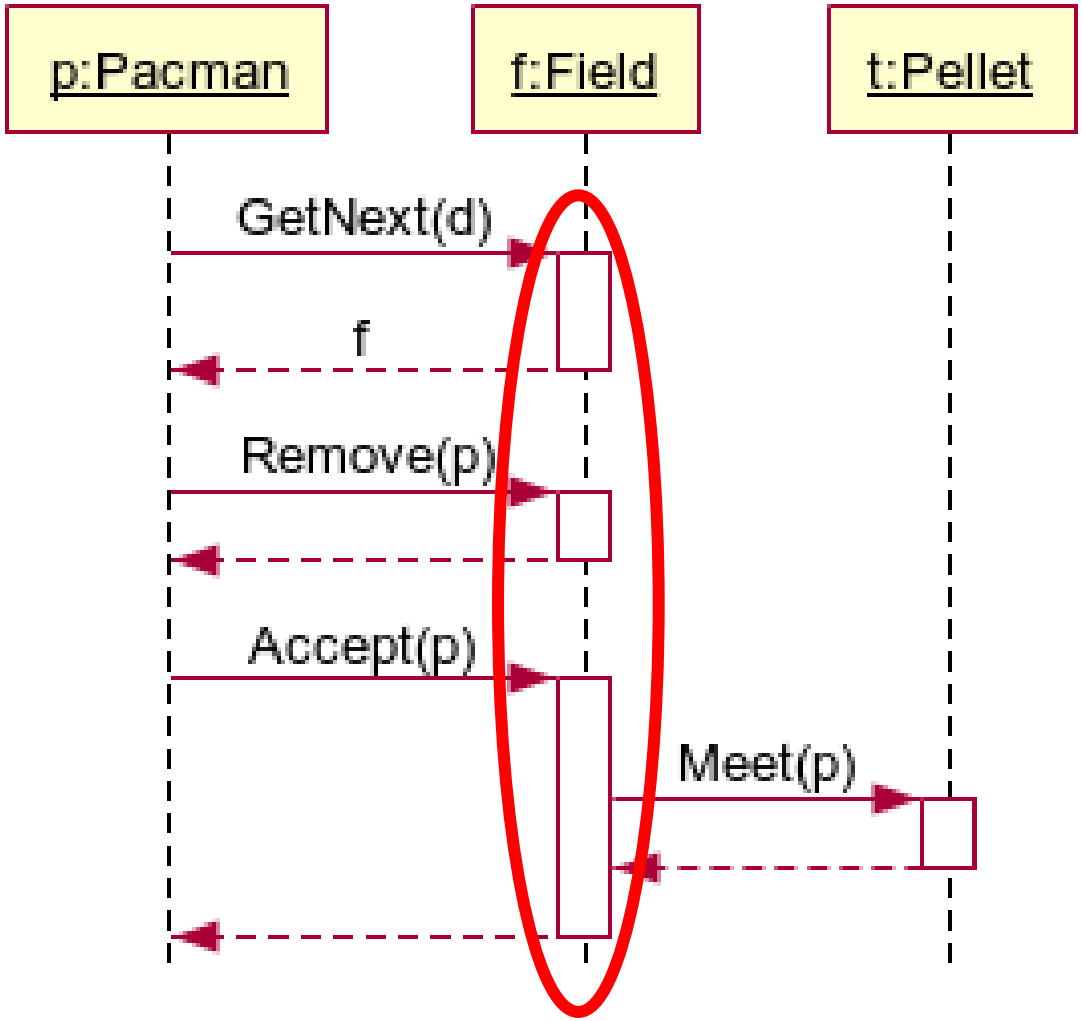
Q4. A polimorfikus viselkedést külön diagramon jelöljük

- 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


p:Pacman
f1:Field
f2:Field
t:Pellet
GetNext(d)
f2
Remove(p)
Accept(p)
Meet(p)
f2:Field
t:LifePellet
p:Pacman
Meet(p)
AddPoints()
f2:Field
t:BombPellet
p:Pacman
Meet(p)
AddBomb()

Q5. Probléma

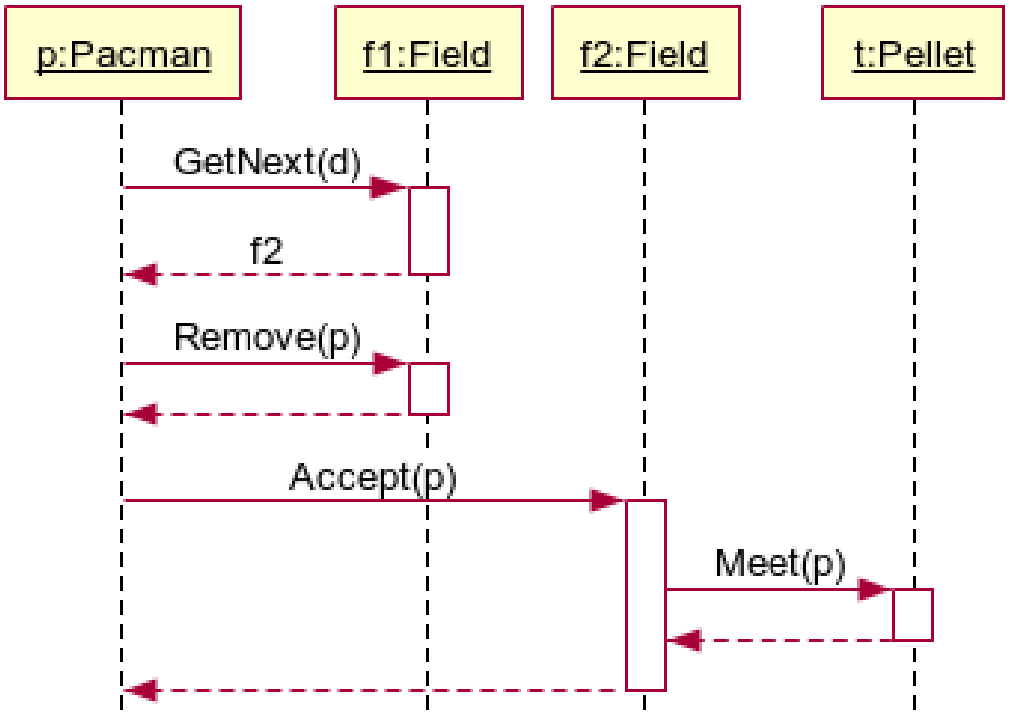


Hiba: különböző objektumok közös életvonalon ábrázolva

Q5. Különböző objektumoknak különböző lifeline-ok kellenek

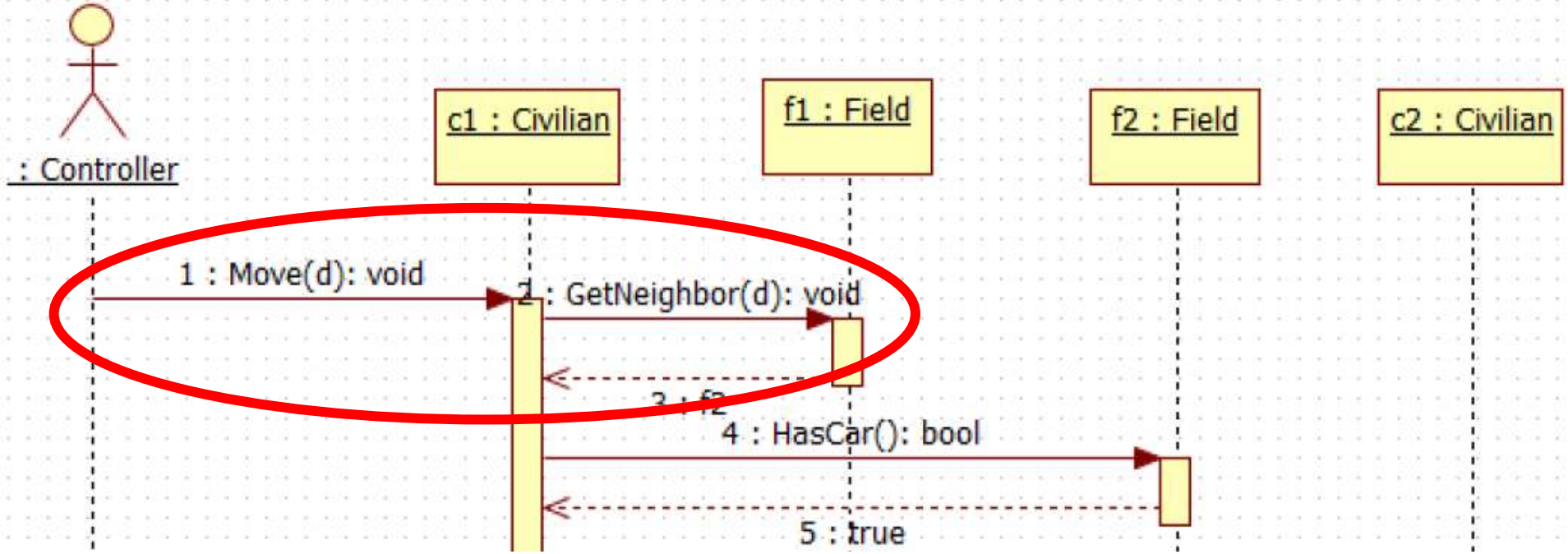
- Probléma ha megszegjük:
  - a függvényhívásokat nem lehet szétválogatni
  - a viselkedés nincs megfelelően definiálva
- Szabály:
  - különböző objektumokhoz különböző lifeline-okat rendeljünk, még akkor is, ha a típusuk ugyanaz
  - adjunk nekik különböző neveket

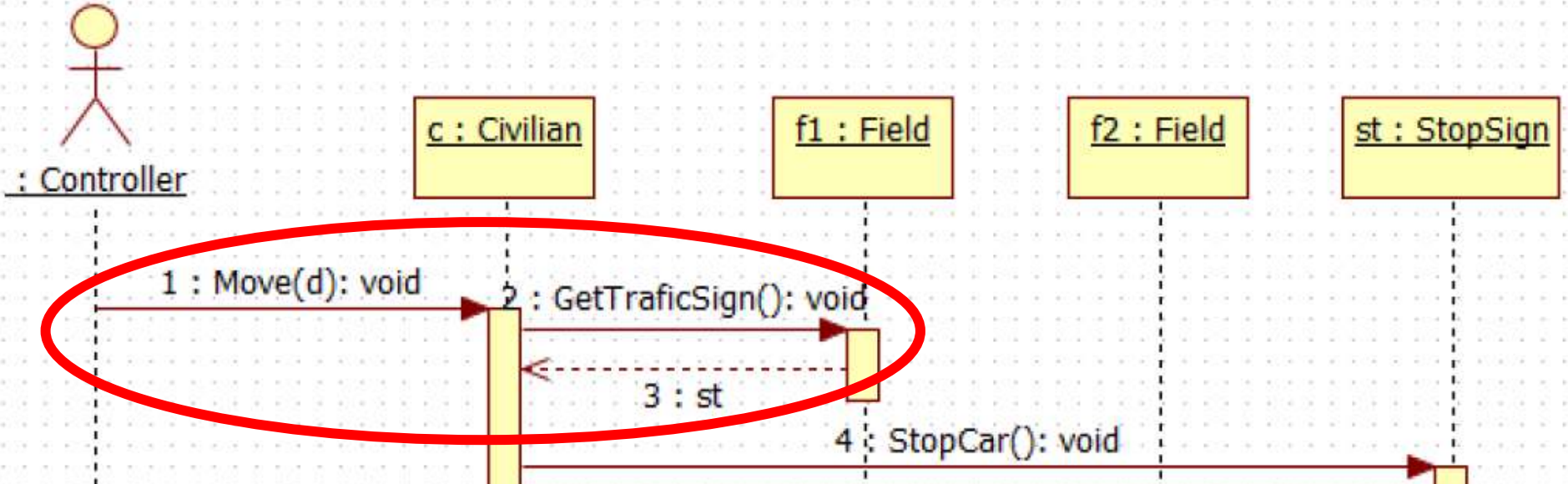
Q5. Megoldás


p:Pacman
f1:Field
f2:Field
t:Pellet
GetNext(d)
f2
Remove(p)
Accept(p)
Meet(p)

Q6. Probléma

Hiba: ugyanaz a függvény két külön viselkedéssel

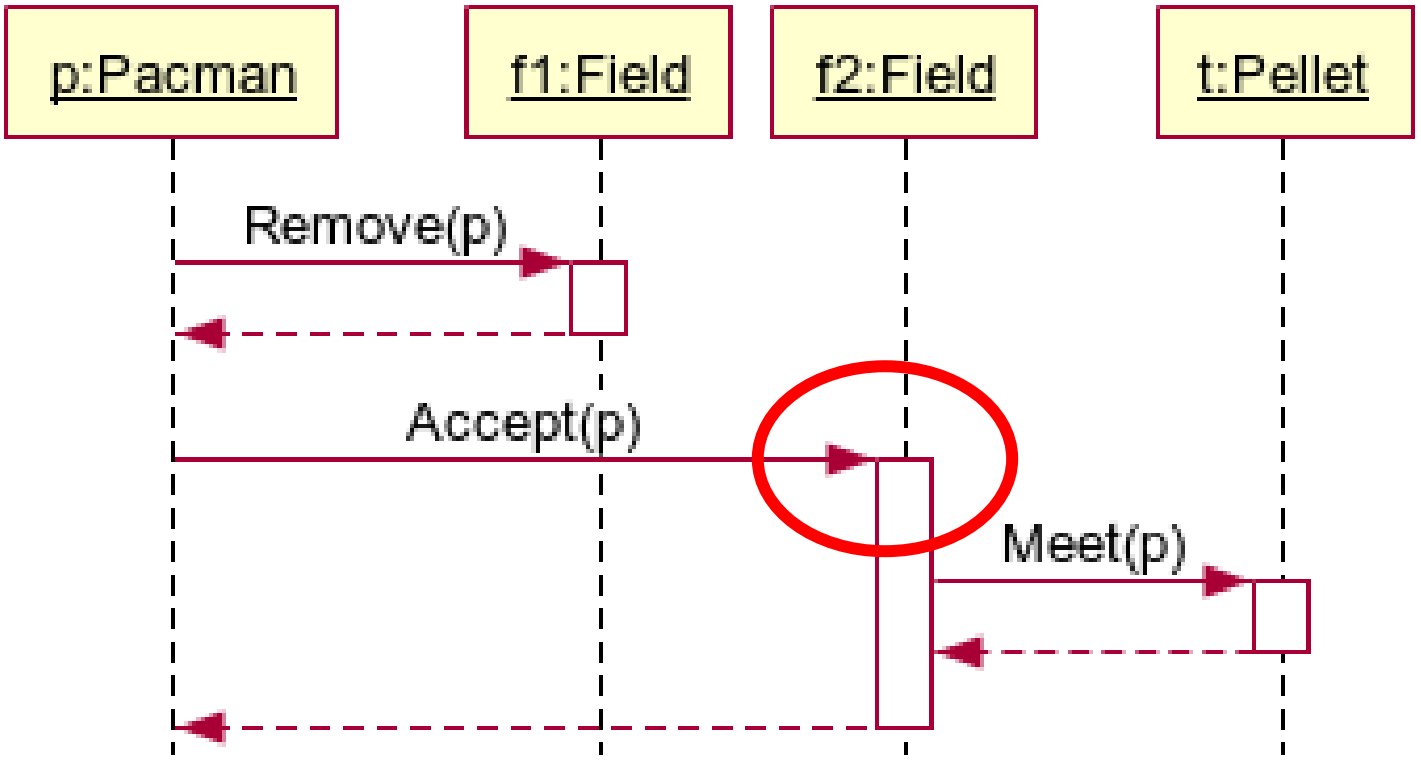




Q6. Diagramok között is konzisztens függvények

- 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 *hasonlónak* kell lenniük
    - és nem mondhatnak ellent egymásnak

Q7. Probléma

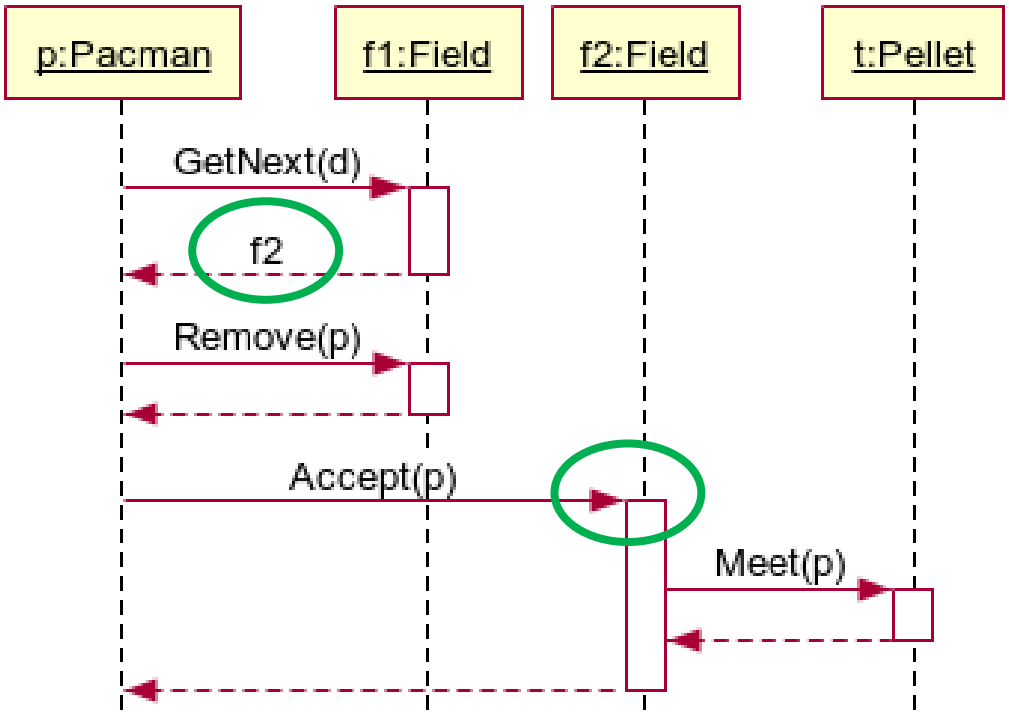


Hiba: olyan objektumon hívunk függvényt, akit nem ismerünk

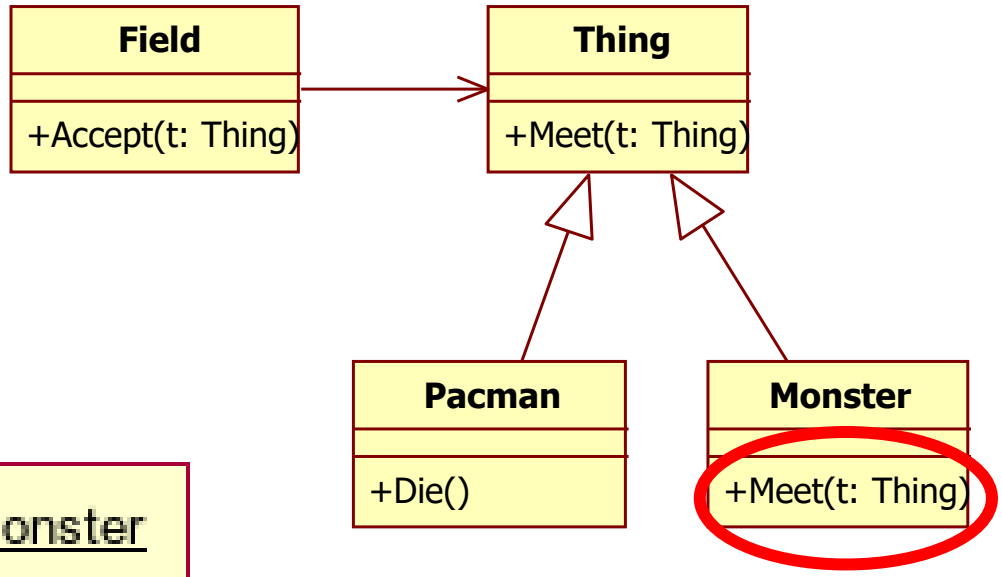
Q7. A hívónak ismernie kell a hívott objektumot

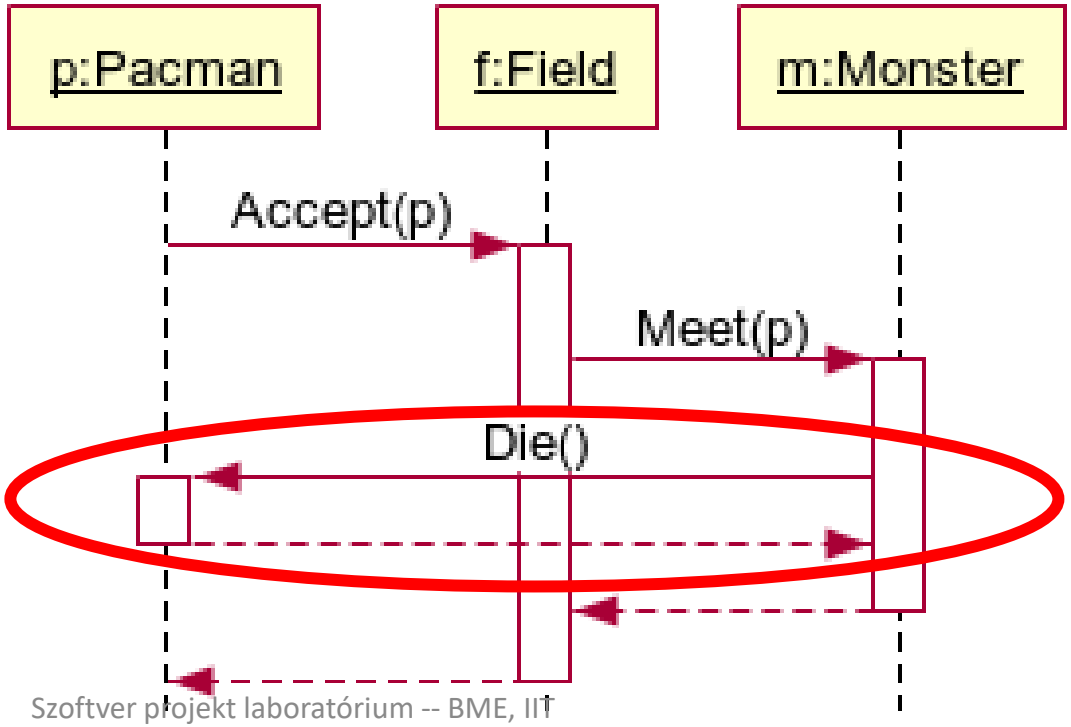
- Probléma ha megszegjük:
  - a szekvencia nem implementálható
- Szabály:
  - a hívónak ismernie kell a hívott objektumot
  - vagy van rá asszociációja
  - vagy az alábbi módokon ismeri meg (függőség):
    - paraméterként kapja
    - egy korábbi hívás visszatérési értékeként kapja
    - ő hozza létre

Q7. Megoldás


p:Pacman
f1:Field
f2:Field
t:Pellet
GetNext(d)
f2
Remove(p)
Accept(p)
Meet(p)

Q8. Probléma





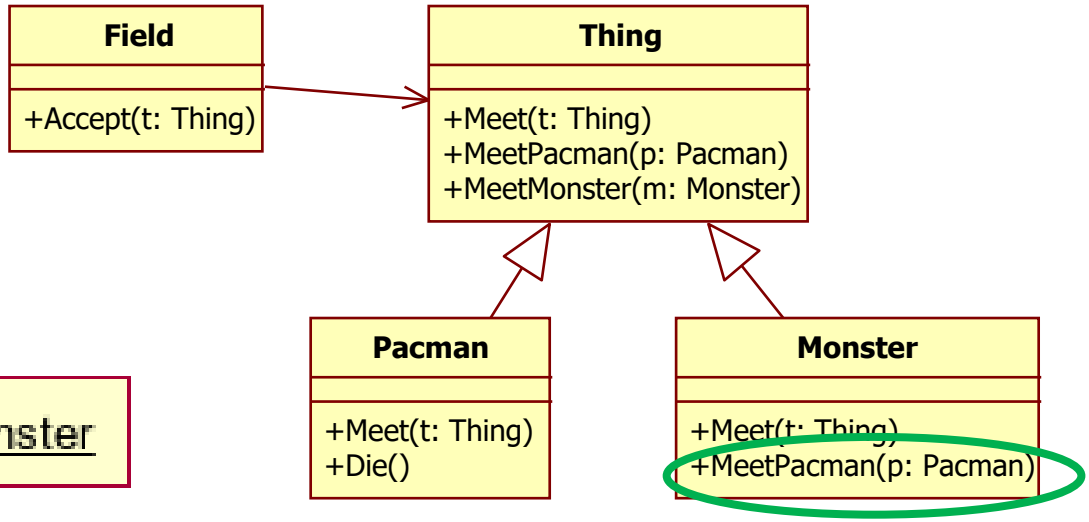
a **t** paraméter statikus típusa: **Thing**
a **t** paraméter dinamikus típusa: **Pacman**

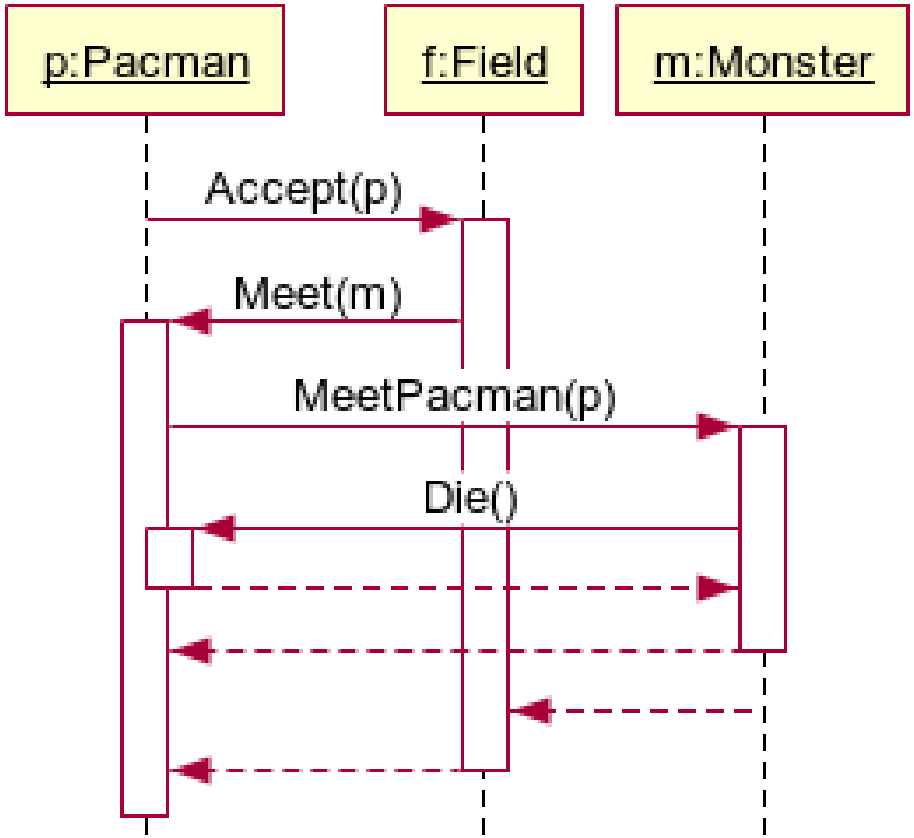
Hiba:
a hívott függvény nem elérhető az ismert statikus típus alapján

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

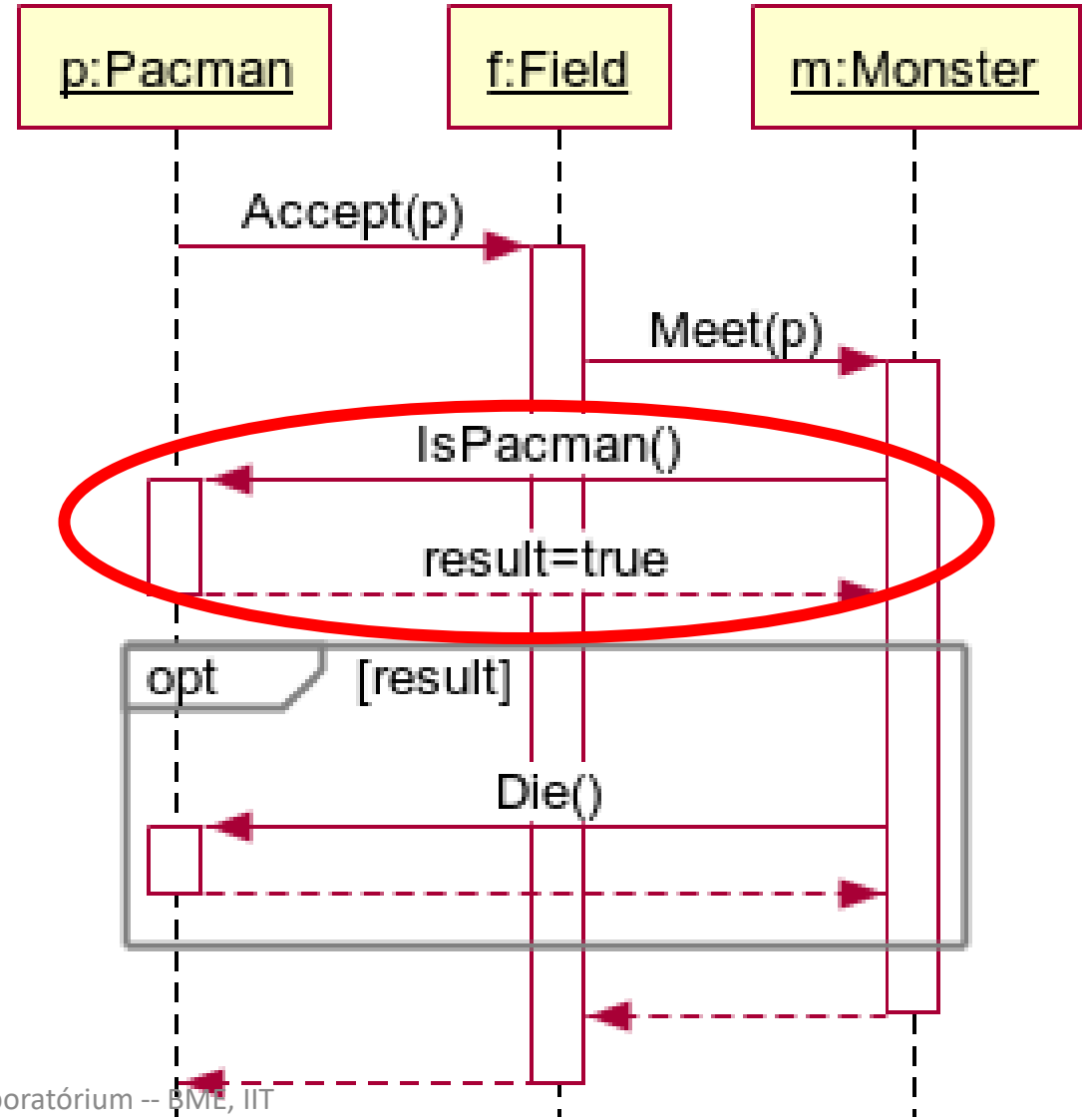
Q8. Megoldás


Field
+Accept(t: Thing)
Thing
+Meet(t: Thing)
+MeetPacman(p: Pacman)
+MeetMonster(m: Monster)
Pacman
+Meet(t: Thing)
+Die()
Monster
+Meet(t: Thing)
+MeetPacman(p: Pacman)


p:Pacman
f:Field
m:Monster
Accept(p)
Meet(m)
MeetPacman(p)
Die()

Q9. Probléma

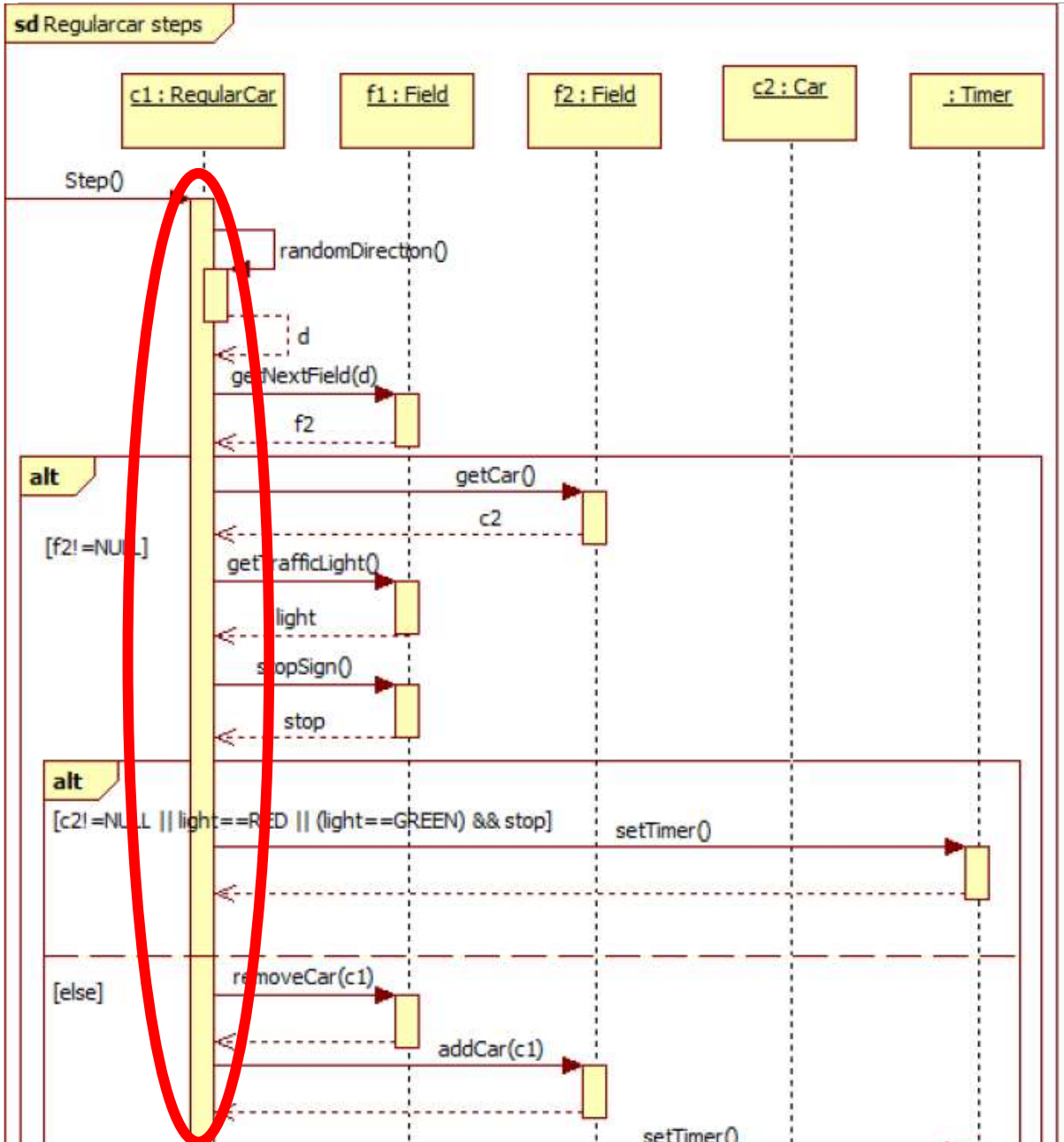
Hiba: a TDA elv megsértése



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

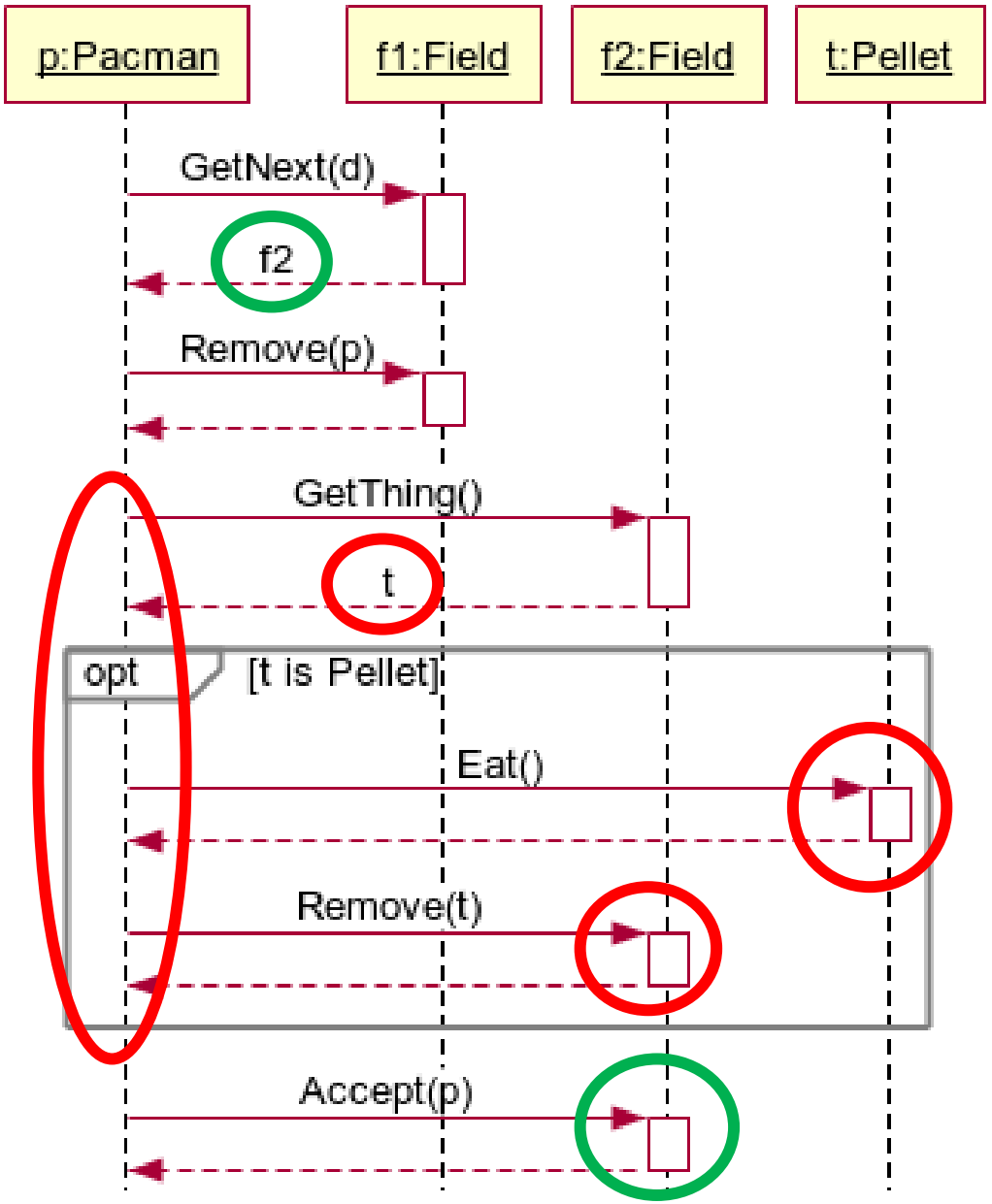


Hiba: “Karácsonyfa”
egyetlen objektum csinál mindent

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



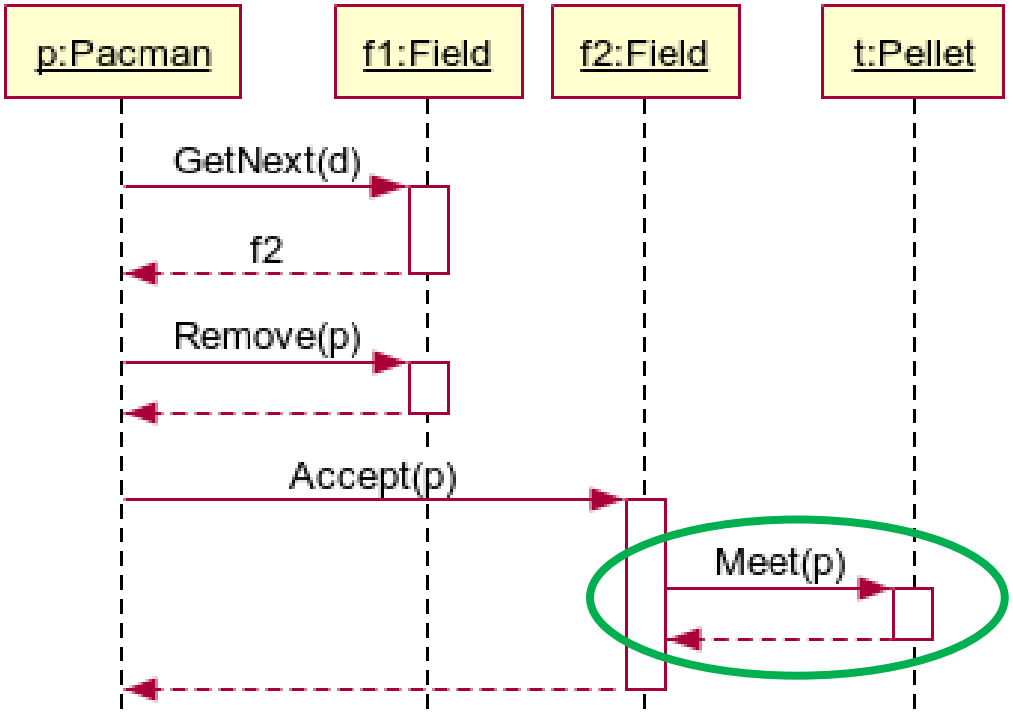
Itt a LoD és TDA megsértése rendben van, mert a Pacman magától lép, nem a következő mező „húzza”

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 változókban 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