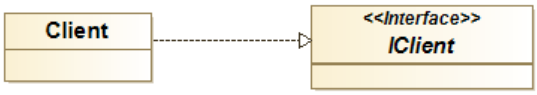
Szoftvertechnológia és -technikák vizsgadolgozat – Minta – megoldás

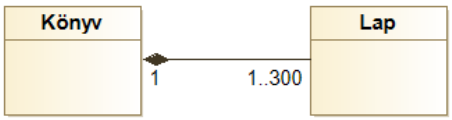
1. Rajzolj le a szabványos jelölést használva (UML) a következő feladatokat a kihagyott helyre! (5p)

Ennél a feladatnál az UML diagramok alapvető jelöléseit kérjük számon. Leghangsúlyosabb az osztálydiagram, de ezen kívül lehet kérdés a Szekvenciadiagram, Aktivitásdiagram, Állapotgép és a Használati eset diagram témaköréből is. A feladat a gyakrabban használt jelölések rutinszerű alkalmazását méri.

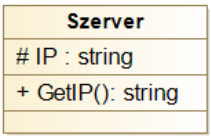
a) Osztálydiagram: A „Client” osztály megvalósítja az „IClient” interfészt



b) Osztálydiagram: A „Könyv” osztály és a „Lap” osztály közti kapcsolat (max 300 lap lehet egy könyvben)

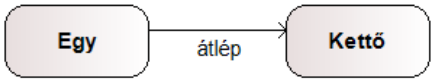


c) Osztálydiagram: A „Szerver” osztály IP címét csak a leszármazott osztályok (ill. saját maga) állíthatják be



(Elfogadható a getter metódus nélkül is.)

d) Állapotgép: Az „Egy” állapotból „Kettő” állapotba jutunk az „átlép” esemény hatására



e) Aktivitásdiagram: A folyamat adott szála véget ér (de a teljes folyamat nem)

2. Tervezési minták/architektúrák

Hasonló feladatok várhatóak a következő minták kapcsán: Singleton, Observer, Strategy, Template Method, Dependency Injection, Command és Command Processor, Composite, Separation of Concerns (SoC), Document-view.

a) Melyek a Singleton tervezési minta megvalósításának kellékei? Jelölje meg a helyes válaszokat! Rossz válasz megadása pontlevonással jár. (3p)

☐ Globális változó	☐ Virtuális metódus	☐ Absztrakt metódus
X Statikus tagváltozó	*X Statikus metódus vagy statikus property*	☐ Védett destruktor
☐ Származtatás	*X Védett konstruktor*	☐ Interfész

b) Adja meg a Separation of Concerns (SoC, vonatkozások/felelősségi körök különválasztása) elv két előnyét (2p)

1. Az egyes részek önmagukban könnyebben megérthetők, átláthatók, párhuzamosan fejleszthetőek.
2. Könnyebb bővíteni, karbantartani, továbbfejleszteni.

3. Készítsen az alábbi specifikációhoz osztálydiagramot! Alkalmazza a tanult tervezési mintákat, ahol szükséges! A szövegben kiemelt fogalmaknak mindenképpen vegyen fel saját **osztályt** vagy *interfészt*! Döntéseit, ahol szükséges, rövid szöveges indoklással egészítse ki! Törekedjen rá, hogy a modell minél kifejezőbb legyen! A rendszer dinamikus viselkedését modellezze szekvenciadiagram segítségével a lenti leírás szerint! Ha a feladatot nem tudja megoldani, részpontért ismertetheti az osztálydiagram elemeit egy másik példán, vagy UML diagramok helyett adhat C# vagy Java kódvázat! (28p)

   Robotok gyártását modellezzük. A **robotok** különböző **komponensekből** állnak. A komponensek lehetnek **összetett komponensek** vagy **alap alkatrészek**. Az összetett komponensek állhatnak további (alap, vagy összetett) komponensekből, maximum 10 darabból. Az alap alkatrészek vonatkozásában az alkatrész azonosítószámát, anyagát és tömegét (lebegőpontos szám) kell nyilvántartani: ezeket létrehozásuk után csak lekérdezni lehet kívülről, megváltoztatni nem. Az összetett komponensek esetében csak az azonosítószámot tartjuk nyilván, de a tömeg is lekérdezhető: ilyenkor mindig újra ki kell kiszámolni az értéket. Az azonosító és a tömeg egységes formában legyen lekérdezhető az összetett komponensekre és az alap alkatrészekre. Az alkatrészek három különböző típusú anyagból készülhetnek: alumínium, réz és műanyag. Az összetett komponensek támogatják egy komponens (mely lehet összetett is) beszerelését és kiszerelését.

   A robot gyárban *gyártási ellenőrök* dolgoznak, akik folyamatosan figyelik az összetett komponenseken történt kritikus változásokat, amit a beszerelés és eltávolítás folyamat okozhat. Jelenleg két típusú gyártási ellenőr van a gyárban, aki értesülni tud a változásokról: **mérlegelő**, aki a tartalmazott komponensek tömegét ellenőrzi (minden változásnál) és **minőségbiztosító**, aki pedig a tartalmazott komponensek azonosítóját ellenőrzi. Ügyeljen arra, hogy minél könnyebb legyen új típusú gyártási ellenőröket bevezetni!

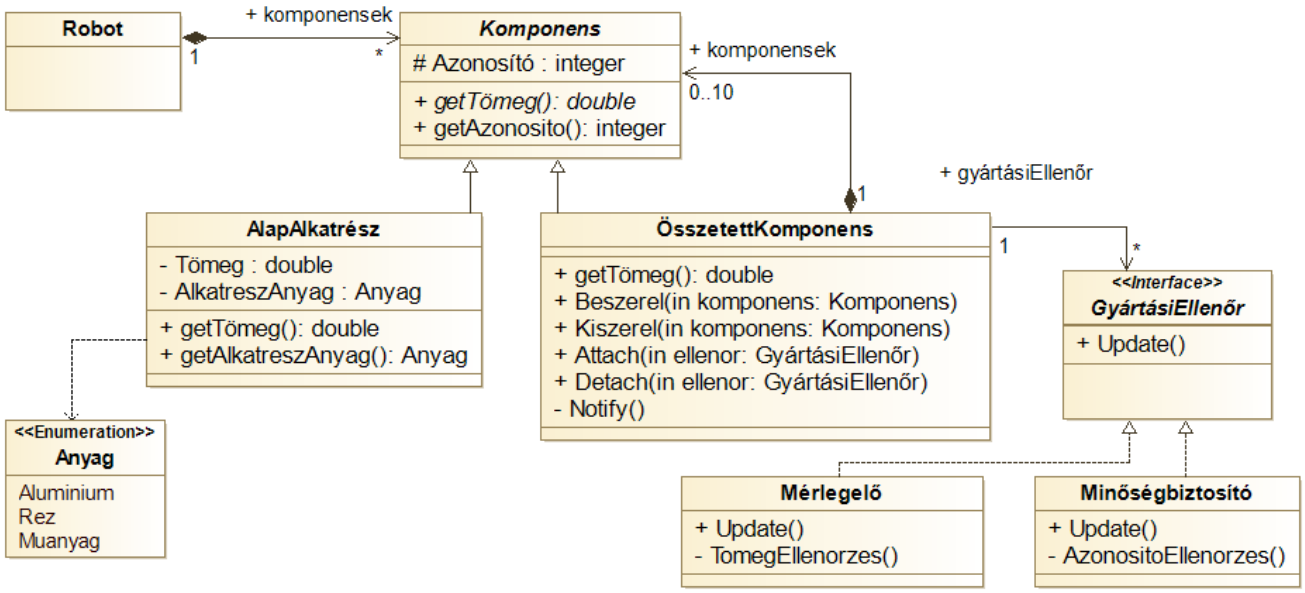
   Jellemezze a dinamikus működést egy robotkaron (összetett komponens) keresztül: egy új mérlegelő feliratkozik a komponens értesítéseire, majd a robotkarba egy új szervomotort (alapkomponens) szerelünk be.

   Ez az összetett feladat a tervezési minták és az UML modellezés ismeretét is igényli. A feladat megoldásához egy, vagy két tervezési mintát kell felhasználni, kombinálni és osztálydiagram segítségével ábrázolni. A diagramon célszerű eltérni a minta ismertetésénél alkalmazott névtől a konkrét feladat szerint (pl. nem Observer az osztály neve, hanem GyartasiEllenor). Aki nem biztos, hogy a minta jól felismerhető, annak érdemes lehet 1-2 mondatban szóban megadnia, melyik mintát és miért alkalmazta. A dinamikus működéshez egy rövid (!) szekvenciadiagramot várunk, ahol a cél a működés szemléltetése, nem a teljes rendszer bemutatása. A pontozás során külön értékeljük a mintákat és az UML jelölésrendszer helyes használatát. Azoknak, akik valamelyikben bizonytalanok, lehetőségük van csak egyik, vagy csak másik feladatra optimalizálni a megoldásukat a feladat első bekezdésének utolsó mondata szerint.

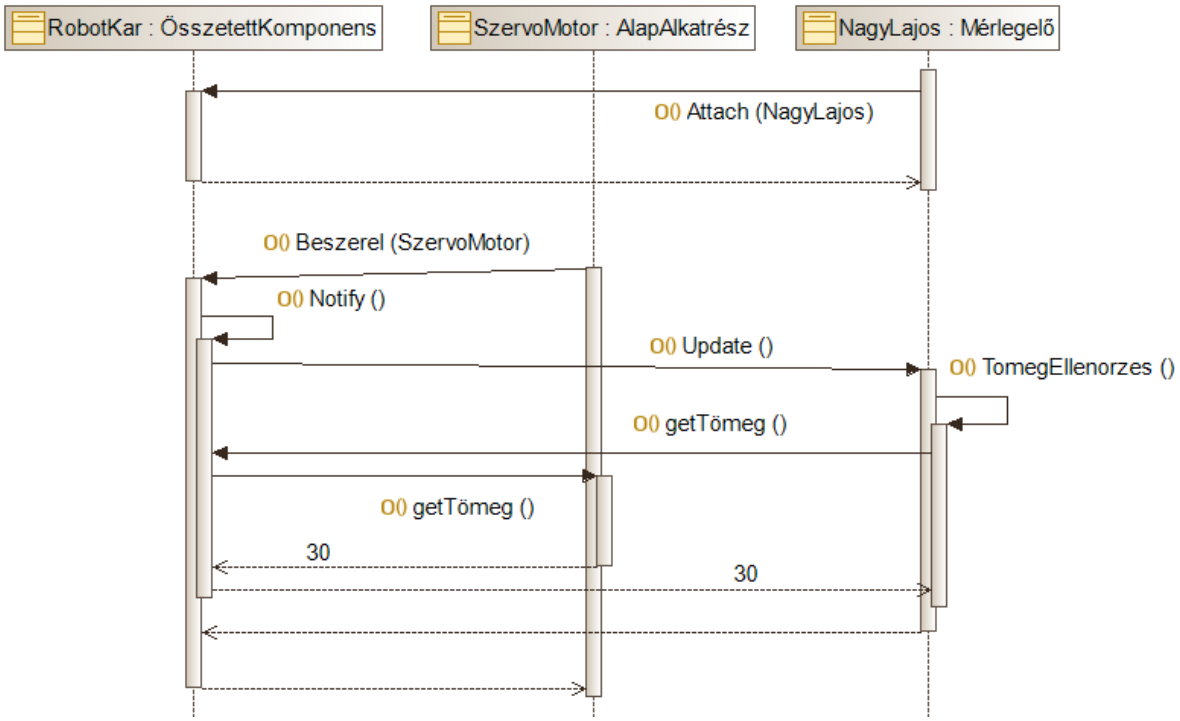
   A konkrét feladat megoldása:

   - *A komponensek megvalósítása egy klasszikus Composite minta, a minőségi ellenőrök kezelése pedig az Observer mintával valósítható meg elegánsan.*
   - *A Composite minta elemei a Komponens (absztrakt ősosztály); az AlapAlkatrész és az ÖsszetettKomponens. A leszármazás és az ÖsszetettKomponens-Komponens közti tartalmazás fontos eleme a mintának. Ez utóbbi tartalmazás azért teli rombusz, mert az összetett komponensek fizikailag részei lesznek a beszereltek. Fontos a multiplicitás megadása is.*
   - *A Komponens ősosztály tartalmazza az azonosítót, ami így protected láthatóságú, hogy a leszármazottak is elérjék. Az azonosító lekérdezését már a Komponens elvégezheti, hiszen a logika mindkét komponens típusnál azonos. A GetTömeg függvény megvalósításai nem szerepelhetnek itt, hiszen eltérő a két leszármazottban.*
   - *Az AlapAlkatrész adattagjai privátok, nem változtathatóak meg kívülről.*

- *Az anyag enumerációval adható meg, a függőség megadása fontos.*
- *A szerelés műveleteknek van paramétere, ami Komponens típusú, ha ez elmarad, hiba.*
- *Az Observer minta része az ÖsszetettKomponens osztályban az Attach/Detach (az ellenőrök regisztrálásának kezeléséhez) és a Notify (a regisztráltak értesítéséhez, ha esemény van). A Notify privát, mivel az osztály saját magán hívja meg (Be/Kiszereléskor). Szintén a minta része az Update fv., ami a GyártásiEllenőr interfészbe került. A GyártásiEllenőr interfész, nem osztály, mivel a feladatkiírásban dőlt betűvel volt szedve.*
- *A Mérlegelő és Minőségbiztosító osztályok megvalósítják a GyártásiEllenőr interfészt és konkrét ellenőr logikát valósítanak meg a privát függvényeik segítségével a szöveges leírásnak megfelelően. Az Update fv. szerepeltetése itt elfogadható, de el is hagyható.*



A dinamikus működést leíró Szekvenciadiagram:

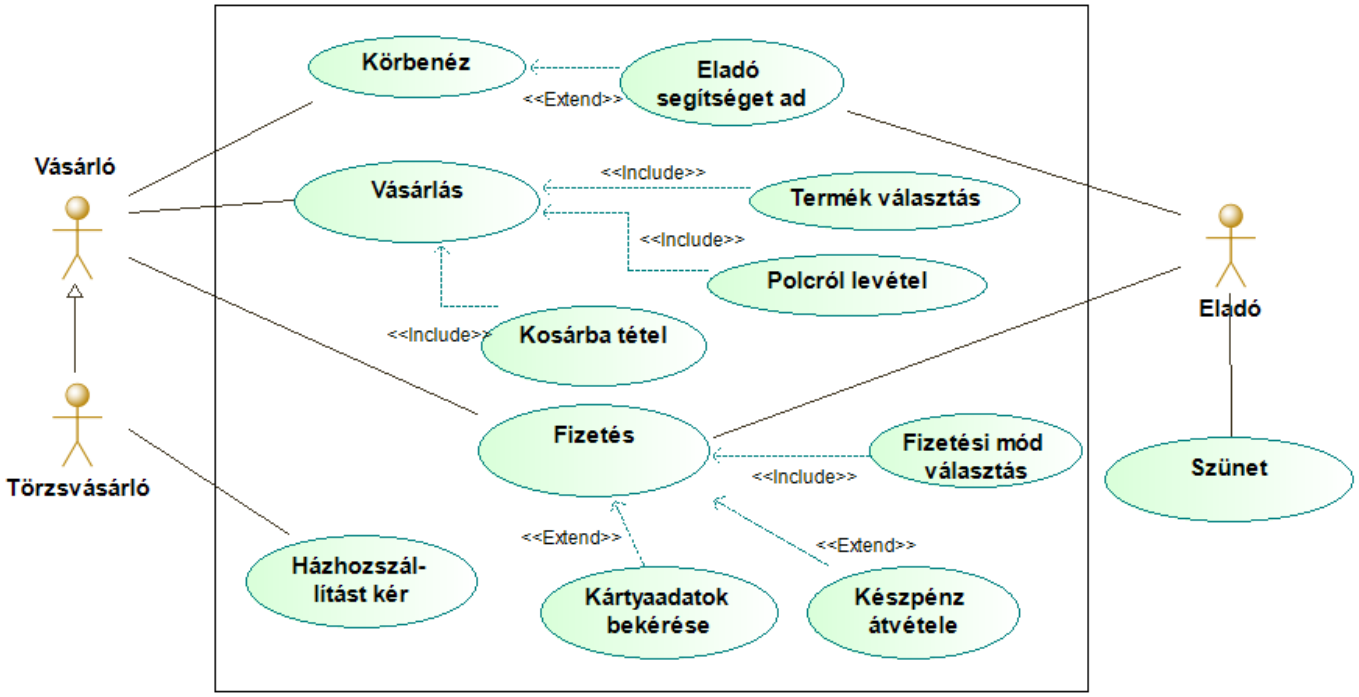


- *A Mérlegelő saját magát regisztrálja be, ez nem szükségszerű, de így a legegyszerűbb.*
- *Hasonlóképpen az egyszerűség kedvéért a SzervoMotor indítja a saját beszerelését. Ez egy elfogadható egyszerűsítés, bár szigorúan véve inkább egy Szerelő osztály egy példány indítaná a folyamatot el.*
- *Az Update minden regisztrált ellenőrre lefut (itt egy ilyen van). Hasonlóképpen a TömegEllenőrzés hatására a Robotkar az összes komponensére meghívja a getTömeg függvényt, és az eredményeket összegzi. Ezt érdemes szövegesen feltüntetni, mert a minta fontos része ez a logika.*
- *A getTömeg visszatérési értékénél a „30” a tömeg értéke.*

4. Készítsen használati eset (Use Case) diagramot az alábbi specifikáció alapján! (12p)

Egy kisboltban zajló vásárlás folyamatát szeretnénk modellezni. A boltba betérő vásárló szétnézhet a boltban, ekkor az eladó segíthet neki, de csak akkor, hogyha épp nem elfoglalt. Ha a vásárló vásárolni akar, akkor kiválasztja a terméket, leveszi a polcról, és a kosarába teszi. A vásárlónak lehetősége van fizetni is az eladónál, ekkor eldönti, hogy mivel szeretne fizetni (hitelkártya, vagy készpénz). A fizetés részletes folyamata nem lényeges a rendszerünk szempontjából. Amikor a boltban nem tartózkodik egy vásárló sem, az eladó szünetet tart, egészen addig, amíg egy újabb vásárló nem érkezik a boltba. A vásárlók között megkülönböztetünk törzsvásárlókat is, ők általában tudják, hogy miért jönnek, de néha ők is szétnéznek a boltban. Törzsvásárlóként kérhető házhozszállítás is, ami egyébként nem.

Ebben a feladatban szöveges leírásból kell egy Aktivitásdiagramot, Állapotgépet vagy Használati eset diagramot létrehozni. A feladat az UML jelölésrendszer helyes használatát teszteli.
A megoldás:



Megjegyzések:

- *Az eladó nem minden esetben ad segítséget a nézelődés során, ezért a kapcsolat extend típusú.*
- *A Vásárlás folyamat három részre (lépésre) van bontva a szövegnek megfelelően. Ezek mindig megvalósulnak, ezért include típusú a kapcsolat. Ebben az Eladó nem vesz részt!*
- *A fizetés része a mód kiválasztása (include). A két fajta fizetés modellezése azonban nehézkes, mivel nincs elágazás a Use case diagramon . Itt feltételes lépésként modelleztük (extenddel) ezt, a két fizetési módra jellemző egy-egy használati esettel.*

- *Az eladó szünete a rendszeren kívül van a modell szerint. A rendszeren kívüli explicit szerepeltetés azt biztosítja, hogy kikötjük: ezt nem kell megvalósítanunk.*
- *A Törzsvásárló öröklődik a Vásárlóból és a Házhozszállítás funkciót csak ő éri el. Az utolsó előtti mondat második fele figyelmen kívül hagyható, nem modellezzük (nem is igazán modellezhető ezen a nyelven).*

5. Jellemezze a "Proxy" tervezési mintát! Mire ad megoldást a "Proxy" tervezési minta? Mutassa be általánosságában vagy a minta egy alkalmazásán keresztül (elég az egyik) a minta működését! Ezen belül rajzolja fel a minta osztálydiagramját, valamint röviden adja meg a mintában szereplő osztályok szerepét! Az osztálydiagramon a minta működésének szempontjából kritikus metódusok esetében pszeudokódot is adjon meg!

   Megvalósításra konkrét forráskódot nem kell írnia, ugyanakkor részpontszámért UML diagram helyett C# vagy Java kód alapján is ismertetheti a mintát. (15p)

   Egy tervezési mintát, vagy architektúrát kell ismertetni általánossában. Szóba jöhet bármelyik tervezési minta, ill. architektúra, amelyről az előadásban szó esett.

   A Proxy mint a 8. előadásban szerepelt :
   https://www.aut.bme.hu/Upload/Course/SzTT/hallgatoi_segedletek/EA08%20-%20Tervez%c3%a9si%20mint%c3%a1k%203.pdf

6. Foglalja össze a DevOps céljait, az általa megoldott problémák lényegét 8-10 mondatban! Mit jelent a fenntartható fejlesztés fogalma? (10p)

   Ebben a feladatban a 10-13. előadás anyagát kérjük számon kifejtős kérdés formájában.

   A DevOps témakör a 13. előadáson szerepelt:
   https://www.aut.bme.hu/Upload/Course/SzTT/hallgatoi_segedletek/EA13%20-%20CICD_Devops.pdf

7. Jelölje a jellemzően összetartozó fogalmakat a két csoport elemei közt! Lehetséges, hogy van elem, amihez nem tartozik pár és az is lehet, hogy van olyan, amihez több pár is tartozik. A rosszul jelölt kapcsolat pontlevonással jár. Döntéseit indokolhatja röviden, ha szükségesnek érzi! (5p)

a. Unit teszt	1. Alpha-Beta teszt
b. Modul teszt	2. Egy-egy metódus tesztelése
c. Integrációs teszt	3. Külső cég is végezheti
d. Rendszerteszt	4. Memóriaszivárgás tesztelése
e. Átvételi teszt	5. Kész terméket teszteli
f. Regressziós teszt	

   Ebben a feladatban a 10-13. előadás anyagát kérjük számon a fenti módon: az összetartozó fogalmakat, esetleg meghatározásokat/jellemzéseket kell összekötni. Érdemes indokolni ott, ahol a kapcsolat a két fogalom közt többféleképpen értelmezhető
   .
   A megoldás (a 10. előadás alapján):
   1E; 2A (2F is elfogadható); 3D; 4B; 5D, 5E, 5F