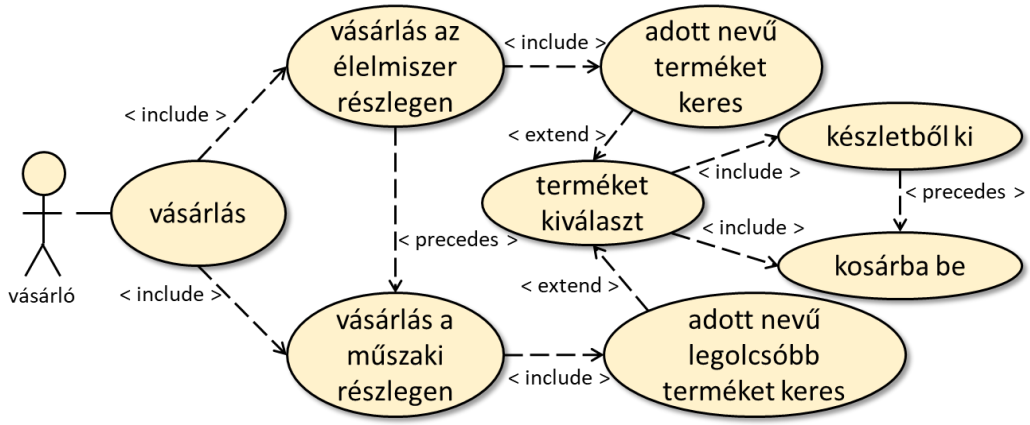
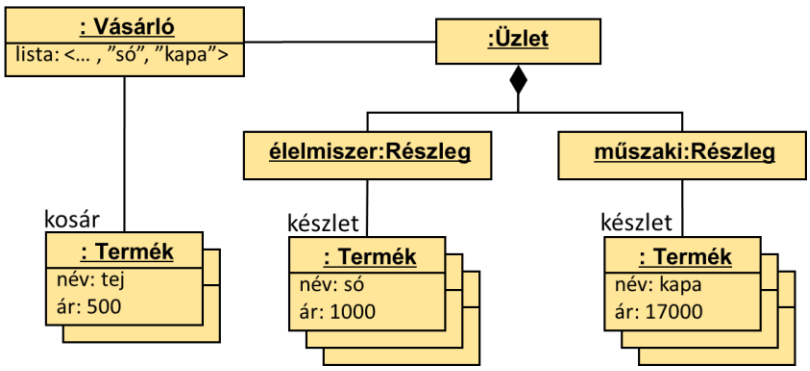
7. *Objektumok felelősségi köre*
   A feladatok modellezése az elemzési UML (használati eset-, objektum-, kommunikációs-, osztály-, szekvencia-) diagramok felrajzolásával kezdődik. Ezek során rá kell mutatni arra, hogy az egyes tevékenységek mely objektumok felelősségi körébe tartozzanak. Meg kell alkotni ezek metódusait, amelyek a kommunikációnak megfelelően hívják egymást. Itt figyelni kell a láthatósági megszorításokra, az ezt feloldó getter-ek és setter-ek bevezetésére. Különös figyelemmel kezeljük egy ’1 - n’ kapcsolatban levő objektumok közötti kommunikációt, amely számos esetben a sok oldali objektumok gyűjteményének algoritmus minták mentén történő feldolgozásába torkollik.

1. Egy kisvárosi üzlet élelmiszer részlegből és műszaki részlegből áll, ahová a vásárlók egy bevásárlólistával érkeznek, amely azon termékek neveit tartalmazza, amit megvennének. Az üzletben a listájukon szereplő termékeket keresik: először az élelmiszer részlegen nézik végig a teljes bevásárlólistát, és a megtalált termékeket magukhoz veszik (beteszik a kosarukba), majd a műszaki részlegen ezt megismétlik, de megfontoltabban: ha egy (a bevásárlólistán szereplő) árúból több is van a részlegen, akkor a legolcsóbbat választják. Feltehetjük, hogy ugyanaz a termék nem szerepelhet mindkét részlegen.

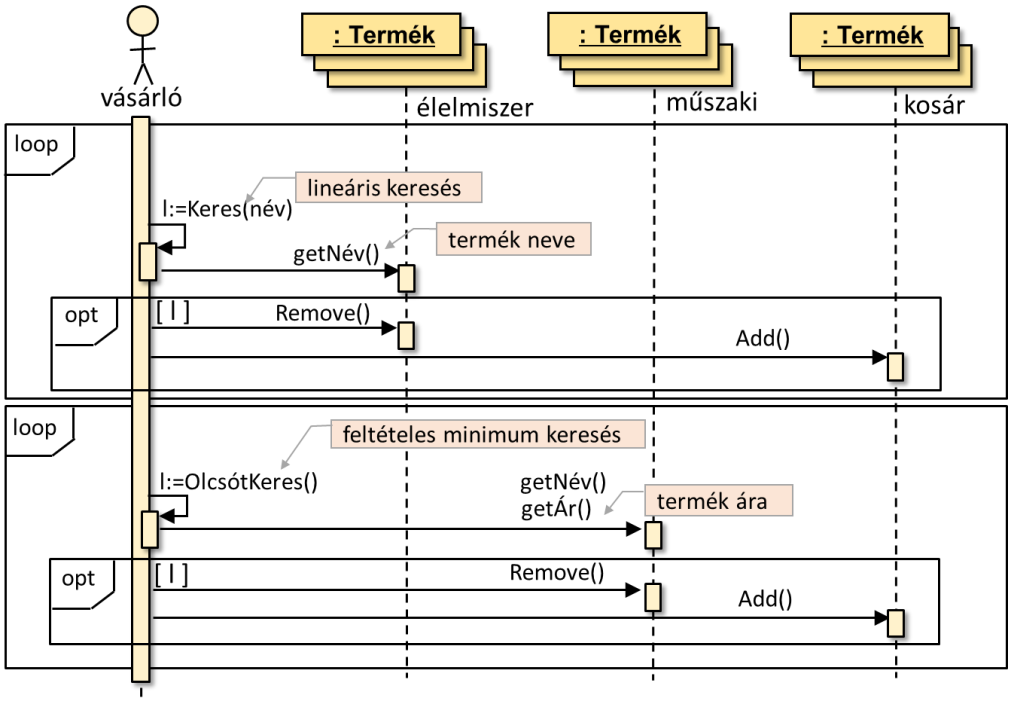
A feladat szövegét elemezve kigyűjthetjük a megoldás fő objektumait és tevékenységeit.
A tevékenységeket a használati eset diagram tartalmazza.



A megoldás objektumai: a vásárló, aki bevásárló listával rendelkezik (ezt nem fogjuk külön objektumként felvenni, hiszen csak stringek sorozata), az üzlet, amelynek két részlege van, és a részlegek árukészletei, amelyek termékeknek a gyűjteményei. Készítsünk ezek alapján objektumdiagramot.

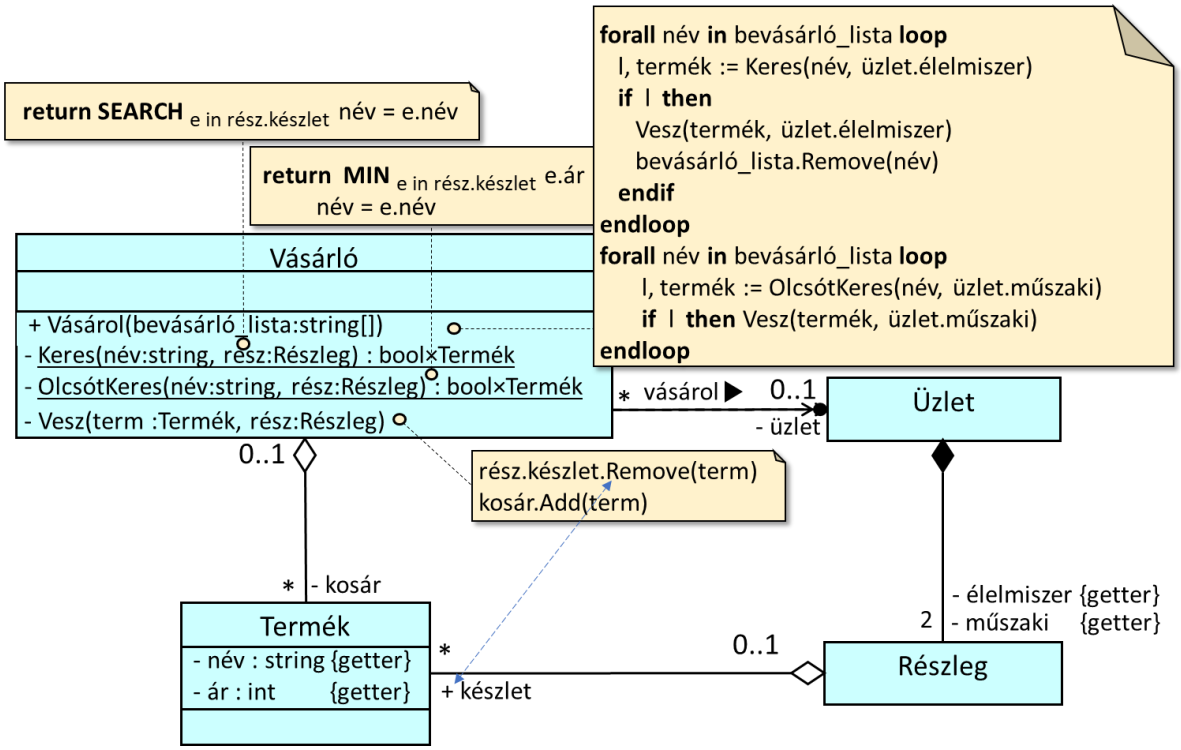


Az objektum diagram nem mutatja meg az egyes objektumok felelősségi köreit, azaz azt, hogy milyen tevékenységek (metódusok) kapcsolhatók hozzájuk. Ennél a feladatnál a tevékenyégek döntő többsége a vásárlóhoz tartoznak. A vásárló fő tevékenysége a Vásárlás(), amely két lépésből áll: az élelmiszer részlegen történő vásárlásból, és a műszaki részlegen történő vásárlásból. Mindkettőnél a bevásárló listán szereplő termékeket kell megkeresni: az élelmiszer részlegen az első találatig (Keres()), a műszaki részlegen pedig a legjobb árral rendelkezőt (OlcsótKeres()). Itt szükség lesz egy termék nevének és árának lekérdezésére (getterek). A kiválasztott terméket úgy vesszük meg (Vesz()), hogy az átkerül a részleg készletéből a vásárló kosarába, amihez a készletek és a kosár gyűjteményeinek kezelését kell biztosítani (Ad(), Remove()). Ezt az egész folyamatot a vásárló Vásárol() metódusa tartalmazza, amelynek működését egy szekvencia diagrammal ábrázolhatjuk.



Látható, hogy a vásárlás két ciklusból áll: először az élelmiszer részlegen keressük a listán levő termékeket, és betesszük a kosarunkba, ha megtaláltuk; utána a műszaki részlegen keressük a lista termékeinek legolcsóbbikát, és a kosárba tesszük, ha van.

Készítsük el most az osztálydiagramot.



A vásárló és az üzlet kapcsolatát jelölhettük volna függőséggel is, de asszociáció esetén kifejezhető az is, hogy egy vásárló egyszerre legfeljebb egy üzletben vásárolhat, míg egy üzletnek több vásárlója is lehet. Ahhoz, hogy egy vásárló hatékonyan elérje az üzletet, ahol vásárol, nem kell az üzlet hivatkozását egy adattagjában tárolnia, elég azt a vásárlás folyamatát leíró metódusának paraméterként átadni.

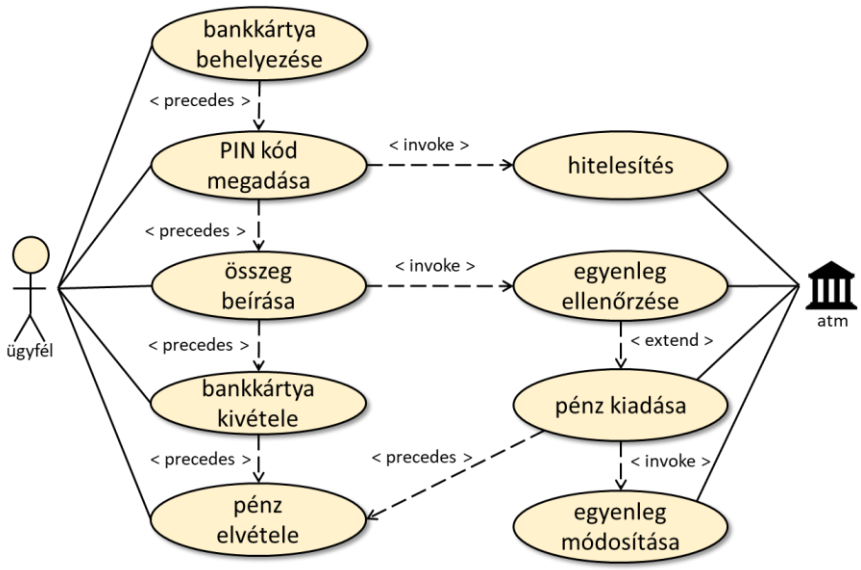
Az üzlet részei a részlegek (kompozíció), ezekre az Üzlet osztály adattagjaiként megjelenő két szerepnév (élelmiszer, műszaki) hivatkozik. Ezeket az Üzlet konstruktora inicializálja. Egyelőre nincs szükség arra, hogy a részlegek hivatkozzanak az őket tartalmazó üzletre.

A Részleg-Termék asszociációban a * multiplicitás szerepneve (készlet) egy termékekből álló gyűjteményt azonosít. Minden részleg rendelkezik egy ilyen készlet adattaggal, amely azért publikus, hogy a vásárló is tudja módosítani a Remove() művelettel. Hasonló mondható el a Vásárló-Termék kapcsolatban a * multiplicitású kosár szerepnévről is, de ez lehet privát, hiszen a vásárló így is el tudja érni.

A Keres() és OlcsóKeres() metódusok nem használnak fel vásárlóspecifikus adatokat: ezeket külső vagy osztályszintű metódusként is be lehetne vezetni.

2. Egy ATM automatánál az ügyfelek sorban állnak, hogy pénzt vehessenek fel. Az ügyfelek rendelkeznek bankkártyákkal. Egy bankkártya egy bankszámlához tartozik, és van egy PIN kódja. Egy ügyfél odaadja a bankkártyáját és a PIN kódját az ATM-nek, és az ellenőrzi ennek hitelességét. Ezután az ügyfél megadja a felvenni kívánt összeget. Ha az összeget levonva az ügyfél számlájának egyenlegéből az továbbra is pozitív marad, akkor az ATM kiadja az összeget. Ehhez a folyamathoz az ATM egy központon keresztül lekéri az ügyfél számlaegyenlegét a kártyájának adatai alapján, illetve elküld a egy jelentést a lebonyolított tranzakcióról az ügyfél bankjának, amely ezalapján leveszi az összeget az ügyfél számlájáról.

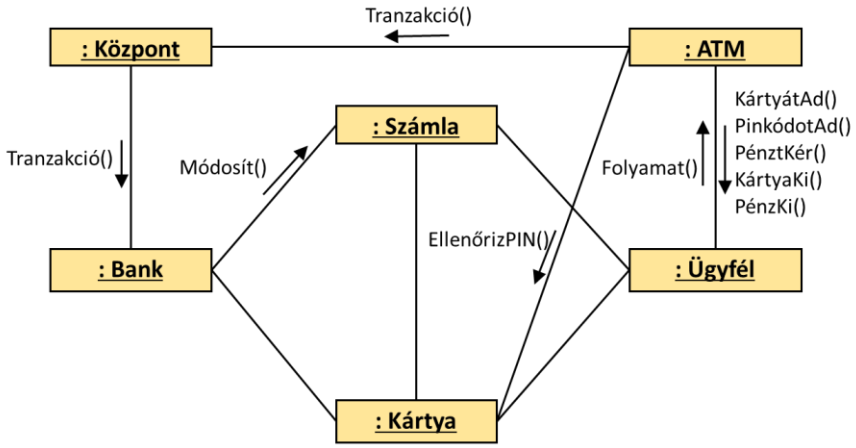
A használati esetek felsorolásánál még nincs jelentősége annak, hogy az atm mögött egy központ van, amely a bankokkal áll kapcsolatban, ahol az ügyfélszámlákat vezetik.



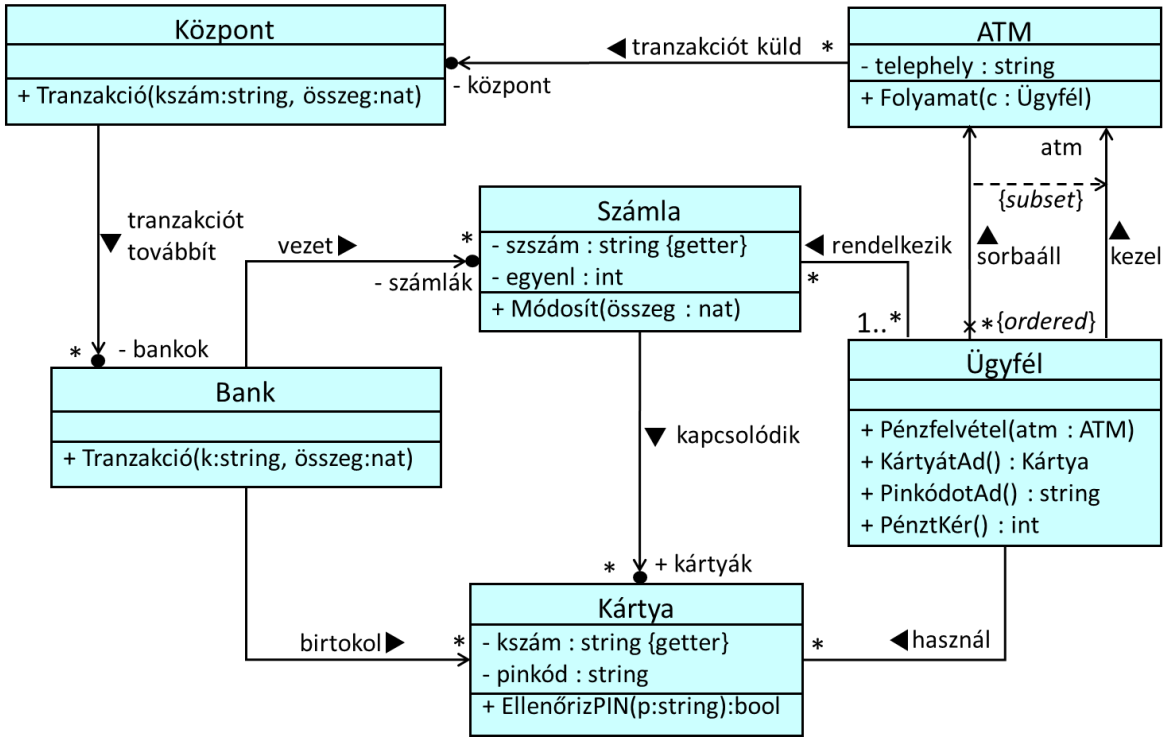
A pénzfelvétel folyamata az alábbi:

- az automata elkéri az ügyféltől a kártyáját
- bekéri a pinkódot
- ellenőrzi, hogy a megadott pinkód megegyezik a kártya pinkódjával
- bekéri a felveendő összeget
- ellenőrzi, hogy az ügyfél számláján van-e elegendő pénz, ehhez elkéri a számla egyenleget
- értesíti a bankot a tranzakcióról, aki levonja az összeget az ügyfél számlájáról.

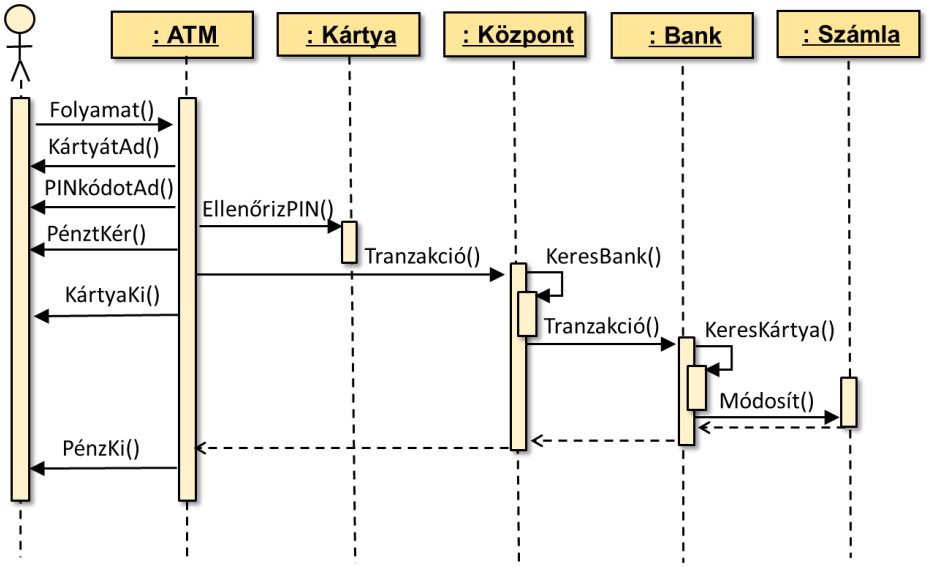
A pénzfelvételt az atm.Folyamat() metódus hívása váltja ki, amelyet a soron következő ügyfél Pénzfelvétel() metódusa hív meg. A pénzfelvétel egyes lépései különböző objektumok felelősségi köréhez tartozó metódusok végzik. Ezek az objektumok: ügyfél, atm, központ, bank, számla, bankkártya.



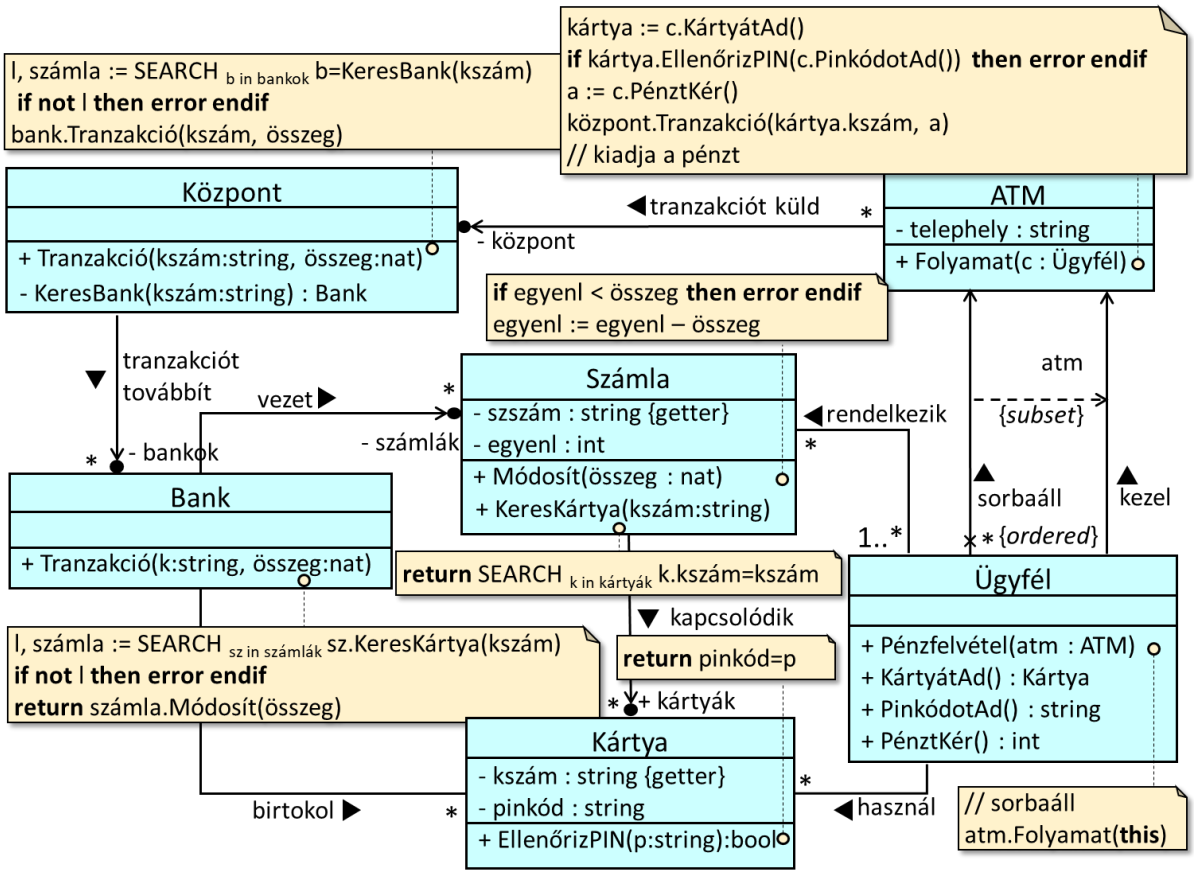
Érdekes része a megoldásnak az, ahogyan az atm hozzájut az ügyfél számlaegyenlegéhez, illetve ahogyan végrehajtja a pénzfelvételt kiegyenlítő banki tranzakciót. Mindkét tevékenység több objektumon keresztül küldött üzenet-lánc formájában, a kérdez, továbbít, vezet, és kapcsolódik asszociációk általi kirajzolt navigációs útvonalon valósul meg (lásd alábbi diagram), ezért a navigációt itt érdemes szerepnevekkel és azok tulajdonosának megjelölésével segíteni.



Különféle segéd metódusokat is bevezethetünk. Például a központnál meg kell tudnunk találni egy adott kártyát birtokló bankot a kártyaszám alapján (KeresBank()). A banknál meg kell tudnunk találni azt a számlát, amelyhez egy adott bankkártya tartozik, és ehhez egy számlához tartozó kártyák között kell adott kártyaszámút keresni (KeresKártya()).
A pénzfelvétel lépéseinek sorrendjét az alábbi szekvencia diagram írja le:

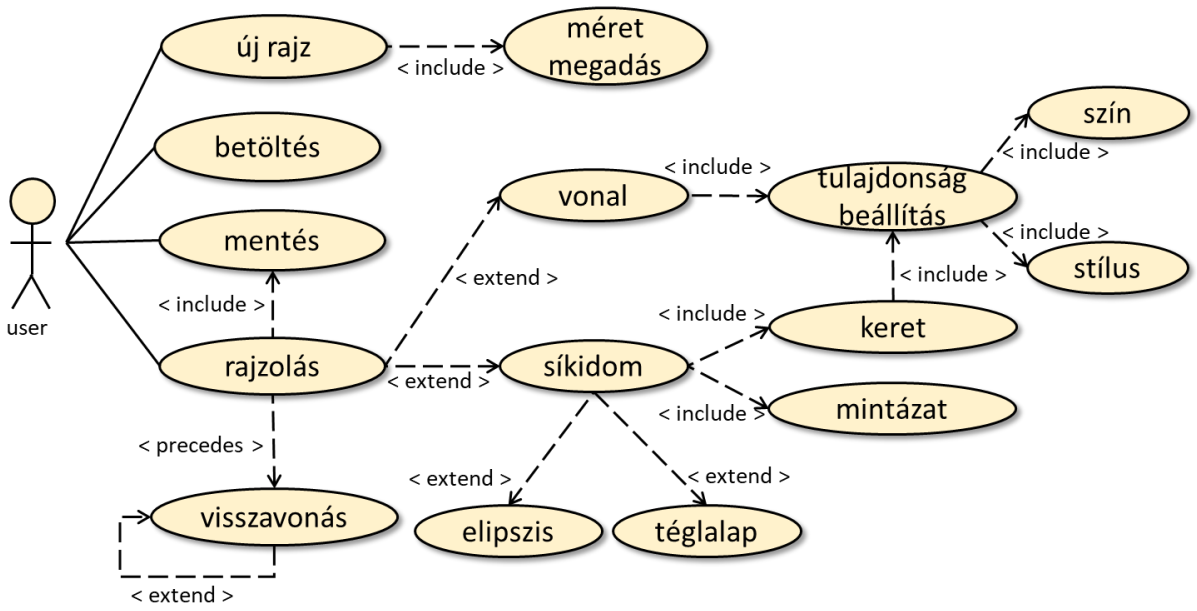


A pénzfelvételi folyamat döntő részben a „kezel” asszociáció mentén küldött üzenet-küldésekkel zajlik. A PIN kód ellenőrzéshez a „használ” asszociációt kell igénybe venni. A „kérdez”, „továbbít”, „vezet”, és „kapcsolódik” asszociációk a számlaegyenleg lekérdezését és a banki tranzakció végrehajtását szolgálják ki. Nem dolgoztuk viszint azokat a metódusokat, amelyek „rendelkezik”, „sorbaáll”, és a „birtokol” asszociációknál lenne szerepe. Nem lényeges most az Ügyfél KátyaKi() és PénzKi() metódusai sem.

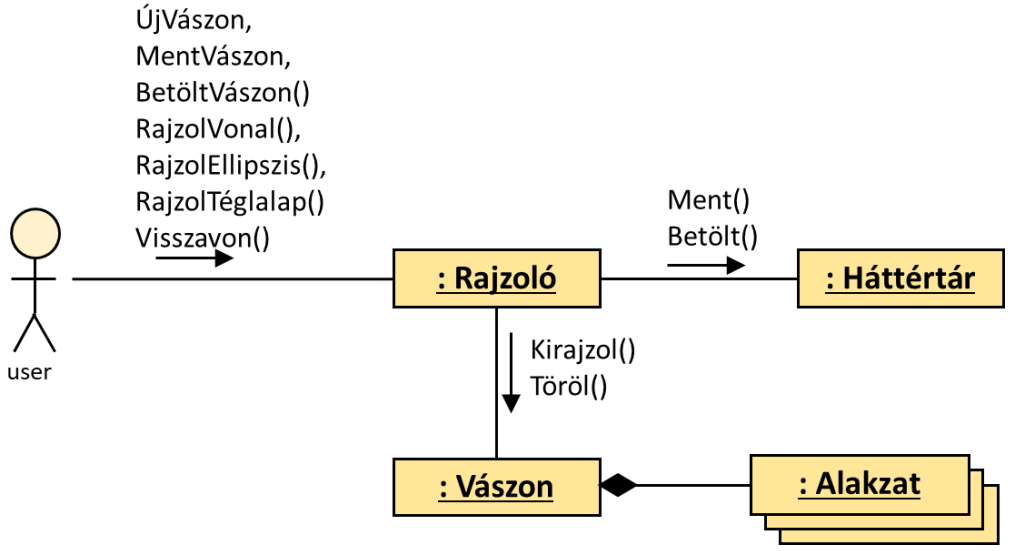


3. Egy grafikus rajzolóprogramban lehetőségünk van egy új rajz készítésére, vagy meglévők betöltésére, illetve rajz mentésére. Új rajz létrehozásakor meg kell adnunk a rajzvászon méretét (szélesség, magasságát). Egy rajzra rárajzolhatunk vonalat, téglalapot, valamint ellipszist. A vonalaknak beállíthatjuk a színét és stílusát (szaggatottság, vastagság). Lehetőségünk van az utolsó művelet visszavonására, ha már történt rajzolás, illetve visszavont művelet újbóli alkalmazására. Minden művelet után történik egy automatikus biztonsági mentés, így ha az alkalmazás összeomlik, a program automatikusan az utolsó biztonsági mentést tölti be.

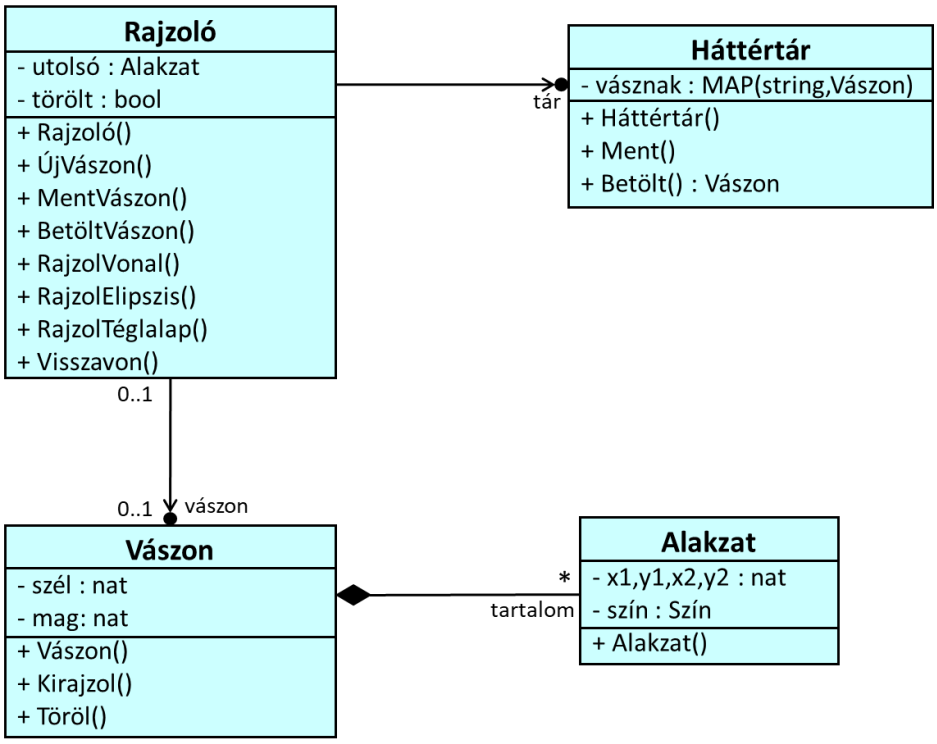
A feladat számos felhasználói esetet tartalmaz, amelyek közül néhány meglehetősen összetett.



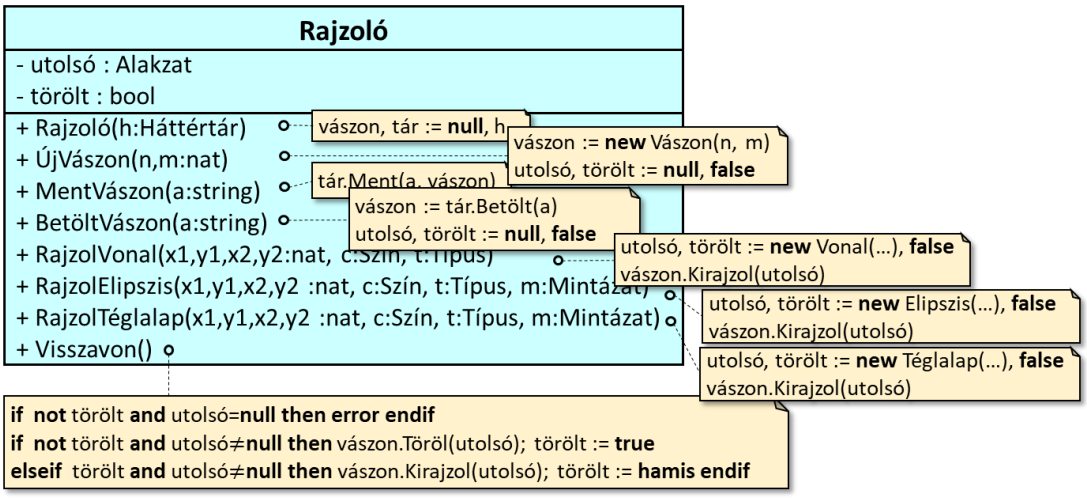
A rajzoló alkalmazás és a rajzolás felületét adó vászon két külön objektum, az utóbbi tartalmazza a vászonra kirajzolt alakzatok gyűjteményét. Külön objektum az elmentett vásznak tárolója.



A rajzoló ÚjVászon() metódusa példányosít egy vásznat az alakzatok üres gyűjteményével és azzal helyettesíti rajzoló objektum egyetlen vásznát. A különféle alakzatokat rajzoló metódusok (Rajzolxxx()) a vászon Kirajzol() metódusával adnak hozzá új alakzatot a vászonnál tárolt alakzatok gyűjteményhez. (A tényleges rajzolást végző utasítások hiányoznak modellből.) A Visszavon() vagy törli a legutoljára kirajzolt alakzatot, vagy annak törlése után újra visszarajzolja azt. Lehetőség van a vászon háttértárba való mentésére, illetve onnan korábban elmentett vászon visszatöltésére.A fenti objektumdiagram alapján elkészíthetjük az osztály diagram szerkezetét:



A Rajzoló osztály talán legérdekesebb része a legutoljára elvégzett akció visszavonása, amelyet a Visszavon() metódus biztosít.

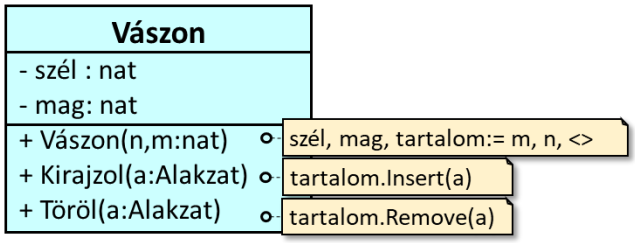


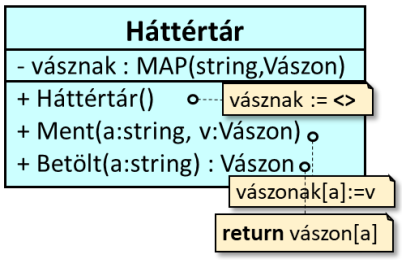
A Visszavon() metódus egy speciális, korlátozott visszavonási funkciót lát el. Ehhez vezettük be a Rajzoló objektum „utolsó” és „törölt” adattagjait.

- Kezdetben törölt=hamis, és alakzat=null.
- Amikor egy új alakzatot rajzolunk, akkor törölt=hamis, alakzat=<új alakzat hivatkozása> lesz.

Visszavonás esetén ha törölt=hamis és utolsó≠null (utolsó=null esetén hibajelzés), akkor az utolsó alakzatot töröljük a vászonról, és törölt=igaz lesz; ha pedig törölt=igaz és utolsó≠null (utolsó=null esetén nincs teendő), akkor újra kirajzoljuk az utolsó alakzatot a vászonra, és törölt=hamis lesz.

A többi osztály jóval egyszerűbb.





A konkrét alakzatok osztályait (Vonal, Ellipszis, Téglalap) érdemes lenne származtatás segítségével definiálni, de ennek részleteivel majd később foglalkozunk.