Shop Project

# The shop manager project

Alkalmazásunkban olyan példákat tárgyalunk, amelyek boltok tevékenységének

nyilvántartását segítik. A példák fokozatosan vezetik végig az Olvasót a különböző Java

eszközök alkalmazási lehetőségein, a hangsúlyt a programozási fogásokra helyezve.

# Part I

**Objectives:** Access modifiers, java class organization, Overloading

## 1. Task:

Modellezzük a következő osztályokat és implementáljuk ezeket:

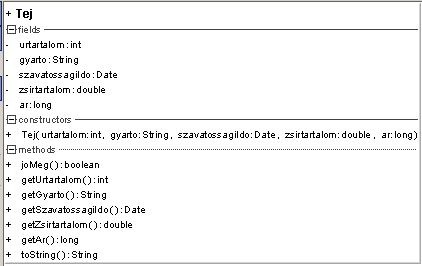
- Elelmiszerbolt

- Tej

Lássuk milyen tulajdonságok és viselkedések reprezentálnak egy Tej típusú objektumot:

Ehhez tartozó osztálydefiníció:

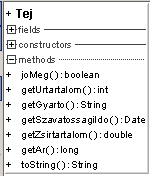
package bolt;



## 1.2. Task:

Amikor egy Tej objektumot meg akarunk jeleníteni a képernyőn, akkor a belső memóriacíme íródik ki, és nem azok az adatok, melyek rá, mint Tej típusú objektumra

jellemzőek. Módosítsuk az osztályt úgy, hogy a sztring-reprezentációja is megfelelő legyen.

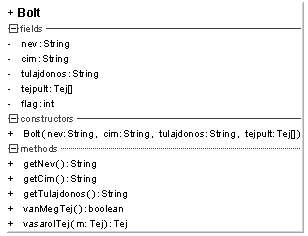


# Part II

**Objectives:** Implementation, API change, array, Vector, ArrayList, Inner class

## 2. Task:

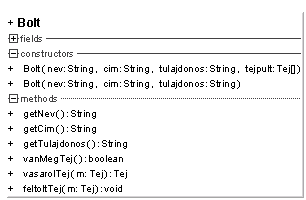
Most hozzuk létre a Bolt osztályt:



## 2.1. Task:

Egy boltból nemcsak vásárolni lehet, hanem időnként fel is kell azt tölteni. Ennek

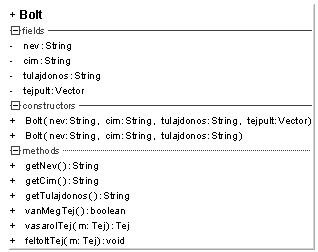
megfelelően módosítsuk a Bolt osztályt.



## 2.2. Task:

A tejek mennyisége folyamatosan változik, ezért alkalmazzunk dinamikus

adattárolást



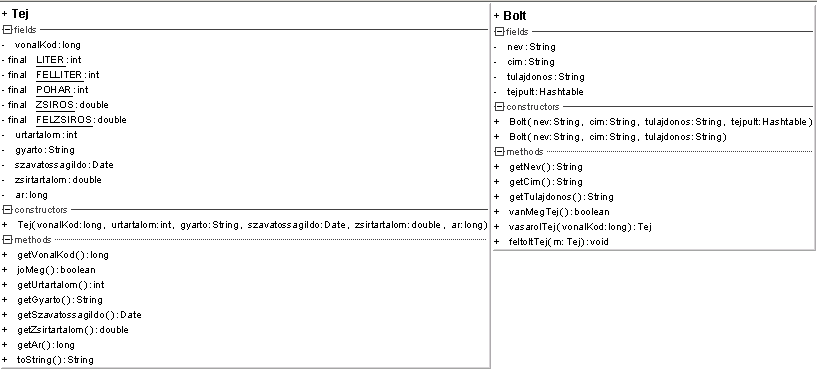
## 2.3. Task:

Vásárlás során sohasem azt mondjuk, hogy „ezt a két tej objektumot kérem” és

átadjuk a tejek referenciáját. A mai boltok minden esetben vonalkóddal dolgoznak, így mind a

feltöltés, mind a vásárlás alapja az áru kódja. Valósítsuk meg a vonalkód alapú tárolást.

Az osztálymódosításokat követő osztálydiagramok:



## 2.4. Task:

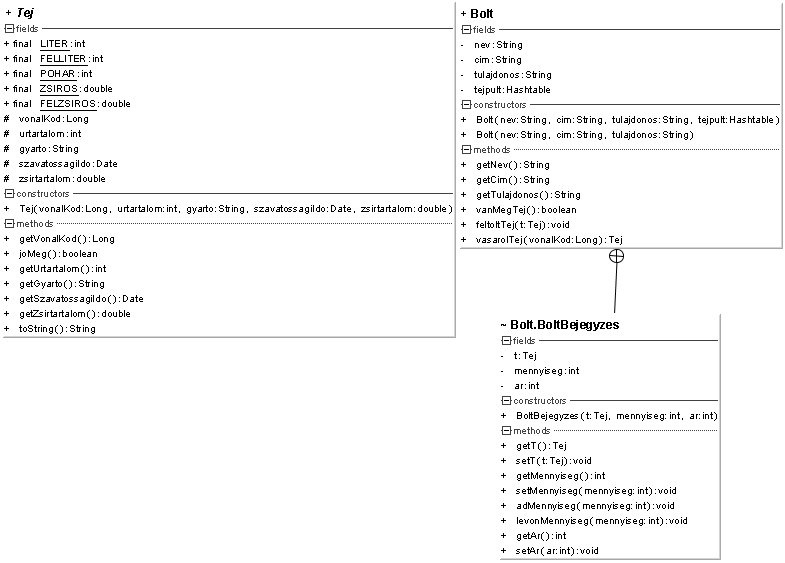
Egy bolt nem minden árut külön-külön tart nyilván, hanem minden árutípushoz

egy-egy bejegyzést rendel, mely azonosítja az árut, tartalmazza az árat, a raktáron lévő

mennyiséget. Tehát az ár nem az áru sajátossága, hanem a bolté. Módosítsuk a Bolt osztályt,

hogy ilyen bejegyzéseket kezeljen.

Módosított osztályok osztálydiagramja:



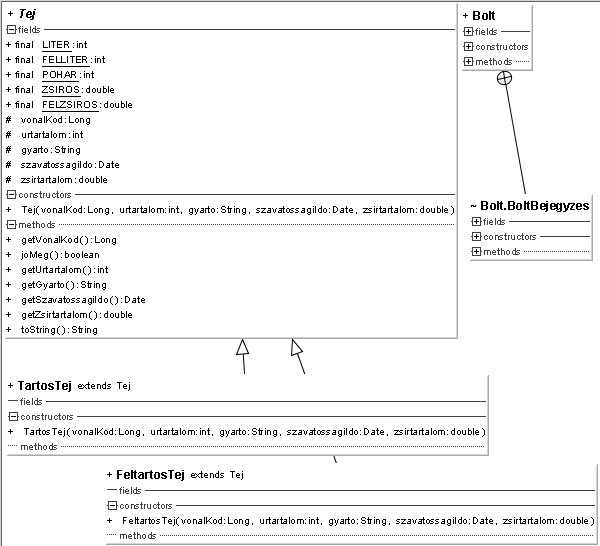
# Part III

**Objectives**: Inheritence, abstract class

## 3. Task:

Egy bolt nem csak egyfajta tejet árul, hanem tartósat, féltartósat, stb. Módosítsuk

úgy alkalmazásunkat, hogy ezen fogalmak megjelenjenek.



## 3.1. Task:

A Tej egy absztrakt fogalom, vagyis olyan típust jelöl, mely inkább gyűjtőnevet

jelöl és nem konkrétat. Ennek megfelelően módosítsuk az alkalmazást.

…

public abstract class Tej {

…

Mivel egy bolt szempontjából mindegy, hogy egy tej milyen, csak egyszerűen tejet akarnak

árulni, a BoltBejegyzes osztály nem változik. Így általános típusú objektumokat tárolunk,

nem kell a specializációval foglalkoznunk.

## 3.2. Task:

Mivel a TartosTej és a FeltartosTej egyazon fogalom kiterjesztései, ennek a

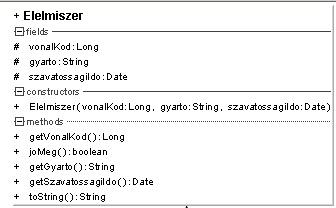
csomagspecifikációban is tükröződnie kellene. Módosítsuk ezen osztályokat.

Módosított osztályok:

## 3.5 Task:

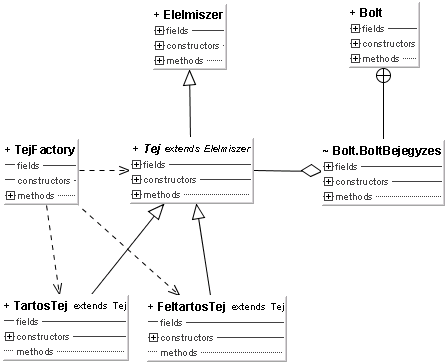
Bővítsük projektünket egy Elelmiszer osztállyal. Gondoljuk végig, hogy az

élelmiszer milyen fogalmat jelöl!



Az Elelmiszer gyűjtőnév jellege miatt absztrakt, és minden tej és sajt ősosztálya is, mivel

azok általános esetét jelenti. Így a létrejövő hierarchia:



Így az élelmiszerekre jellemző tulajdonságok és viselkedések öröklődéssel kerülnek a tej és

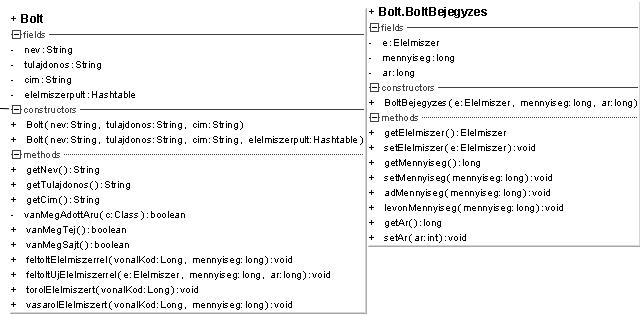
sajttípusokhoz.

## 3.6 Task:

Egy bolt számára mindegy, hogy tej, vagy sajt objektumokat tárol az adatbázisban.

Egyszerűsítsük ezen élelmiszerek kezelését, könnyítsük meg az adatbázis módosítását

segédmetódusokkal.



## Part IV

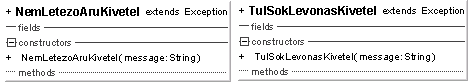
Objectives: Error Handling

## 4. Task:

Készítsük fel alkalmazásunkat az esetlegesen bekövetkező hibák kezelésére is.

Törekedjünk a konzisztens működésre.

Új kivételosztályaink:

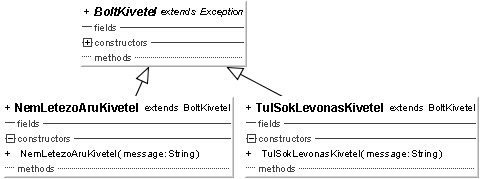


## 4.1. Task:

Egy alkalmazás során sok saját kivételtípust kezelhetünk. Ilyenkor a metódusok

specifikációjánál „tobzódás” léphet fel, így annak hívásánál is körülményes az események

kezelése. Egyszerűsítsünk az alkalmazás kivételmodelljén:

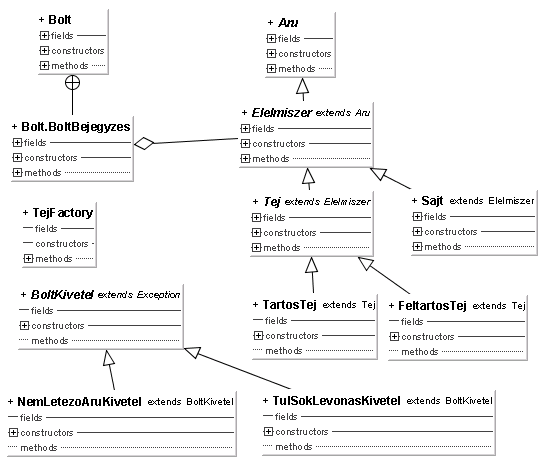


## Part V

**Objectives:** Factory design pattern

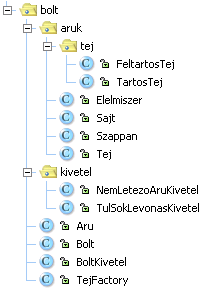
## 5.Task:

Modellünket módosítsuk úgy, hogy szappant is áruljon a boltunk.



Bevezetjük az absztrakt áru fogalmát, mely a szappan és az élelmiszer gyűjtőneve is egyben.

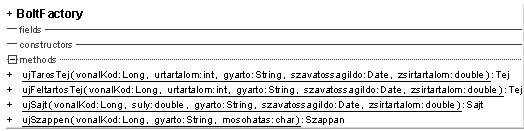
Az osztálystruktúra módosított felépítése:



## 5.1. Task:

TejFactory kiszolgáló osztályunk elég „kiszolgált” már. Módosítsuk az újonnan

elkészült bolti alkalmazás alapján.



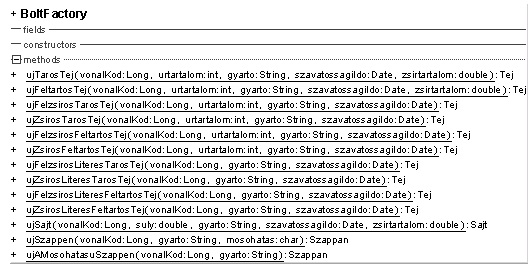
Ez az osztály minden Aru típus példányosításához segítséget nyújt.

## 5.2. Task:

A példányosítások során a sok plusz munkát az olyan paraméterek megadása

jelenti, mely értéke nagy valószínűség szerint minden esetben megegyezik. Adjunk ennek

segítésére módszereket.



# Part VI

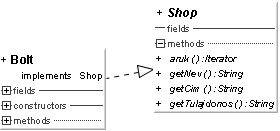
**Objectives**: interface, Hashtable, Iterator

## 6. Task:

Vegyünk a következő fogalmat: írható CD, mely élelmiszerboltban (ma még

általában) nem kapható. Alkossuk meg a Shop viselkedést, mely egy eladási viselkedés OO

reprezentációja.

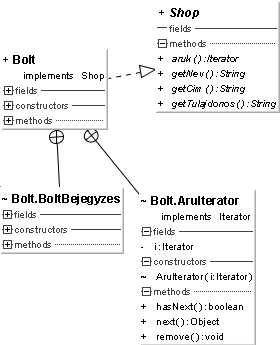


## 6.1. Task:

Mivel az áru ezen típusú iterációja a Hashtable belső implementációját használja,

kiadja minden objektum referenciáját. Írjunk saját iterátor implementációt, mely csak az áruk

vonalkódját adja ki.



# Part VII

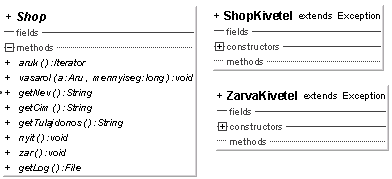
Objectives: Error handling, File handling, Iterator, Cross Cutting Concern

## 7. Task:

Minden Shop objektum rendelkezik nyitás és zárás viselkedéssel. Amikor a Shop

zárva van, semmilyen művelet nem végezhető vele. Módosítsuk ennek megfelelően a

definíciókat.



## 7.2. Task:

Mivel vásárolni az iterátor által adott Aru típusú objektumok alapján lehet, a belső

iterátor reprezentációt is módosítani kell, hogy Aru típusú legyen, de mégis elfedje a belső

reprezentációt. Erre használjuk a névtelen példányosítást!

Az iterátor osztály módosításának eredménye:

class AruIterator implements Iterator{

…

public Object next() {

Aru a = (Aru)i.next();

return new Aru( a.getVonalKod(), a.getGyarto()){};

}

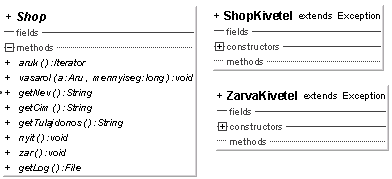
…

}

## 7.3. Task:

Naplózza minden Shop nyitva tartása idején folyamatosan tevékenységeit egy

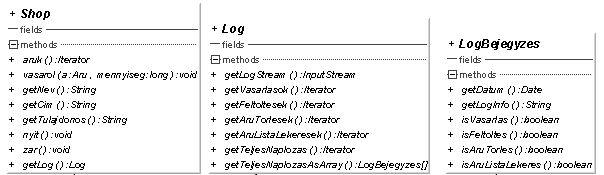
állományba, melyet zárást követően le is lehessen kérdezni!



7.4. Példa: Mivel a napló feldolgozása sokrétű lehet, interfészekkel kell segíteni a

rendszergazda információgyűjtésének folyamatát.

7.4.1. Példa: Vegyünk fel interfészeket a naplózás elérésének segítésére.



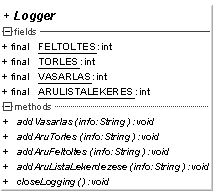
Így a bolt működése során bármikor lekérdezhetőek a naplózási információk, és az

interfészeken keresztül könnyen feldolgozhatóak.

Lássuk az interfészek definícióját:

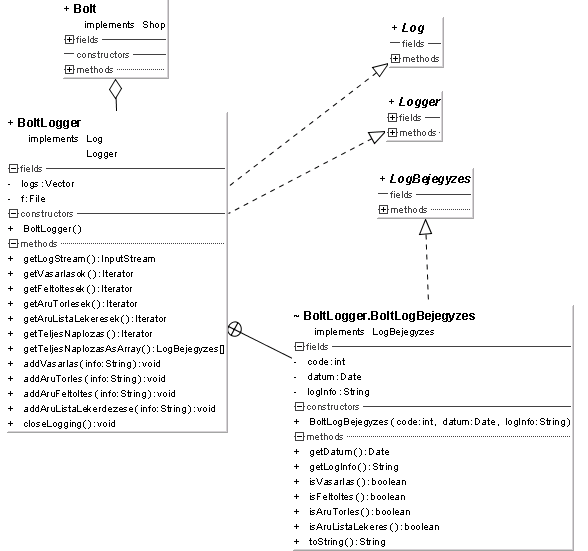
## 7.4.2 Task:

Interfésszel segítsük a napló létrehozását.



## 7.4.3. Task:

Készítsük el a naplózási rész implementációját.



## 7.4.4. Task:

Építsük bele implementációnkat a Bolt osztályba!

# Part VIII

Objectives:

## 8. Task:

A személyek vásárlásának elősegítésére plázákat építenek, így rövidül a bevásárlási

idő. Valósítsuk meg a plaza fogalmat Java környezetben! Az egyszerűség kedvéért

feltételezhetjük, hogy a plaza csak boltokat tartalmaz, és azok egyszerre nyitnak.

