# Java progamozás

**I. Nagy ZH - programozás Java nyelven**

A zh írására 16:15 és 19:00 között nyílik lehetőség. Fontos, hogy időhosszabbításra nincs lehetőség.

**Figyelem, 100 pont elérése számít 100%-os eredménynek!**

A feladatokat figyelmesen olvasd el, tervezd meg a kódot és készítsd el. A programnak futnia kell és a kódnak a kivételeket kezelnie kell. A ZH írása során kizárólag a **csatolt hálózati meghajtón** található anyagok illetve. Meg nem engedett segédeszköz használata a ZH sikertelen teljesítését vonja maga után.

**Időmegtakarítás végett JavaDoc NEM szükséges a teljes pontszám eléréséhez.**

**Egyetlen projektet készíts! Az egyes feladatoknak külön csomagja legyen: `hu.ppke.itk.java.DIGITUSAZONOSÍTÓ.zh1.feladatN`, ahol a DIGITUSAZONOSÍTÓ a digitus azonosítód, a feladatN-ből az N pedig a feladat sorszáma.**

# 1. Bűnügyi nyilvántartás (35 pont)

(A feladat része a megfelelő Collection típusok megtalálása, valamint típusparaméterezése.)

Adott egy csv fájl, amelyben az egyes sorok tartalmaznak adatokat, és az egyes sorokon belül az adatértékek pontosvesszővel (;) vannak elválasztva. A feladat ennek a fájlnak tartalmát beolvasni és eltárolni egy alkalmas Collection segítségével.

A fájl tartalma egy város bűnügyi eseményeinek nyilvántartása a következő oszlopokkal:

körzet; rácspont; bűnügyi kód; x_koordináta; y_koordináta

Az egyes adatok típusai:

pozitív egész, pozitív egész, pozitív egész, előjeles lebegőpontos, előjeles lebegőpontos

Például:

```
3;888;2604;38.55611545;-121.4142729
```

Ehhez egy alkalmas osztályt kell létrehozni get és set függvényekkel, valamint `toString()` és `valueOf(String s)` függvényekkel, ahol a `valueOf()` értelmezi a csv fájl egy sorát, a `toString()` pedig létrehoz egy csv fájl sort az adatokból.

Példányosítsunk egy olyan Collection-t, amely képes tárolni azt, hogy a bűnügyi kódoknak milyen szöveges magyarázat felel meg. Ehhez az adatokat egy második csv fájlból lehet beolvasni, ahol a párok szerepelnek egy-egy sorban.

```
GRAND THEFT-AUTO PARTS;2304
```

Ha az adatokat sikerült beolvasni és eltárolni, akkor a következő feladatokat kell megvalósítani:

1. Rendezzük az előfordult bűnügyeket azok típusa szerint (a típus sorszáma növekvő sorrendben)
2. Rendezzük az előfordult bűnügyeket azok körzete szerint
3. Rendezzük az 1. szempont, majd ezen belül a 2. szempont szerint

Az előző feladatokhoz használjuk a Collection framework képességeit, továbbá a `Comparator` osztályt.

Konzolról kérjünk be egy bűnügyi kódot. Gyűjtsük ki egy rendezetlen adatszerkezetbe a megfelelő adatokat, aztán írjuk ki egy fájlba az eredményt. A fájlok neve az out-CODE.csv sémát kövesse, ahol a CODE a bűnügy kódja.

Készíts a konzolra egy listát, amelyben az egyes bűnügyi kategóriák megnevezése mellett szerepel egy szám, ami azt mutatja meg, hogy hányszor fordult elő az adatbázis szerint. (Ehhez érdemes lehet az előfordulásokat egy megfelelő Collection-be áttenni.)

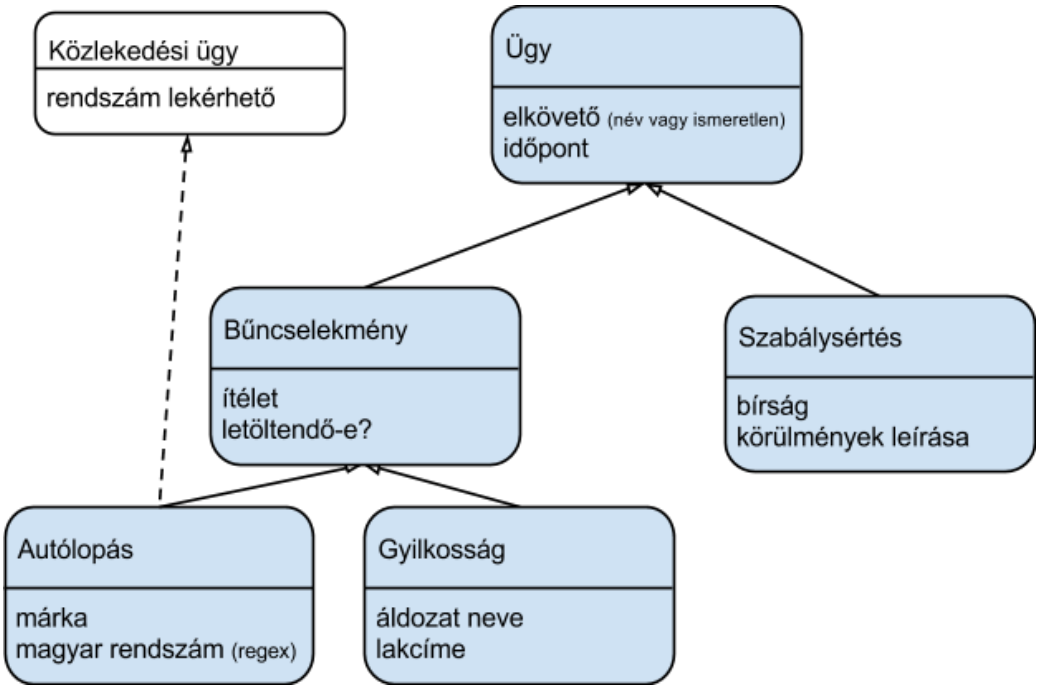
# 2. Bűnügyek kategorizálása (25 pont)

A feladat az ábra szerinti osztályhierarchia leprogramozása. (Az ábra a túloldalon látható.)

Az egyes mezők típusait értelemszerűen válaszd meg! **Csak a levelek legyenek példányosíthatóak**! Minden adattag rendelkezzen getter és setter függvényekkel, valamint egy, a szükséges adattagokat paraméterül átvevő konstruktorral.

A közlekedési ügyeknél legyen lekérhető egységes módon az autó rendszáma, amit annak beállításakor a setterben regex segítségével ellenőrizz! Ha hibás a bemenet, azt saját kivétel dobásával jelezd, és folytatódjon a program futása. Ügyelj arra, hogy ha a konstruktorban adod át a rendszámot, akkor is ugyanez a kódrészlet fusson le!

Feladat továbbá a program működésének bemutatása: hozz létre egy “bunugyek” tömböt, melynek segítségével bemutatod, hogy megfelelően működik a programod (tehát feltöltöd különböző bűnügyekkel, demonstrálva a regex helyes működését is, majd lekérdezed és kiírod ezeket az adatokat).




# 3. Könyvtár (40 pont)

Ebben a feladatban egy könyvtár működését kell szimulálni.

A könyvtárban van egy polc, amin van sok (mondjuk 7 db) könyv, mindegyiknek van egy sorszáma és egy véletlen oldalszáma (1-1000 oldalig). Kezdetben a könyvek az azonosítójuk szerinti, balról jobbra növekvő sorrendben vannak a polcon.

A könyvtárban ülnek a hallgatók (egyszerre több (mondjuk 5 fő), mindegyiket egy-egy szál valósítja meg a szimulációban), akik mind egy-egy bizonyos könyvet szeretnének elolvasni (véletlenszerűen kiválasztanak egyet a könyvtár teljes anyagából).

Ha a könyv, amit el akarnak olvasni, szabad (éppen nem olvassa senki), akkor azt a hallgató leveszi a polcról és elkezdi olvasni (a hallgatók nagyon ügyesek, másodpercenként 100 oldalt tudnak elolvasni). Ha végeztek vele, akkor utána visszateszik a könyvet a polc jobb oldali végére, és új könyvet választanak maguknak.

Ha a könyv, amit el akartak olvasni, foglalt volt, akkor addig nem foglalkoznak más könyvvel, amíg azt el nem olvasták: addig várnak, amíg az szabad nem lesz.

A könyvtárban emellett van egy könyvtáros is, aki minden 5. másodpercben feljegyzi magának, hogy milyen sorszámú könyvek vannak a polcon, balról jobbra sorrendben. Mivel fontos, hogy a hallgatók minden könyvet gyorsan megtaláljanak, ezután sorba is rendezi a könyveket, majd megint feljegyzi magának az immár rendezett listát.

A hallgatók könyvre várakozásának megoldásához **ne használj** busy waitinget, illetve időzített lekérdezést (polling), használd helyette a Java nyelv beépített lehetőségeit, amit a gyakorlaton is használtunk. A feladat megoldásakor figyelj arra, hogy a programod szálbiztos legyen!

A program folyamatosan logolja a történéseket (egy hallgató egy könyvet elkezd keresni, egy könyvet levesz a polcról, egy könyvet visszatesz a polcra) és a könyvtáros jegyzeteit (polcon levő könyvek listája a rendezésük előtt és után) a konzolra!

Segítségül a kiadott batyuban megtalálható a könyveket reprezentáló Book osztály forráskódja.