Debreceni Egyetem

Informatika kar

A HAJDÚSÁGI HULLADÉKGAZDÁLKODÁSI KFT. SZÁLLÍTMÁNYOZÁSÁNAK INFORMATIKAI TÁMOGATÁSA

Témavezető:
Dr. Husi Géza
Főiskolai docens

Készítette:
Pipó Krisztián
Mérnök informatikus (Bsc)

Külső konzulens:
Will Csaba
HHG Kft. ügyvezető igazgató

Debrecen
2010

Tartalomjegyzék

1. Bevezetés

Szakdolgozatomban egy konkrét alkalmazásfejlesztés folyamatát, valamint e folyamat eredményeként készült alkalmazást kívánom bemutatni. Célom egy alkalmazás kifejlesztése, amely a Hajdúsági Hulladékgazdálkodási Kft. információs rendszerét egy új alrendszerrel bővíti ki. Ezen alrendszer a vállalat szállítási tevékenységét végző járműveinek futásteljesítményi adatait, valamint a gépjárművezetők adatait kezeli.

A téma kiválasztásában három fő szempontot tartottam szem előtt. Tanulmányaim során igazán nagyszabású alkalmazásfejlesztési projektben - melynek minden egyes lépésében részt vállalhattam volna - nem volt részem, így elengedhetetlennek tartottam, hogy diplomám megszerzése előtt megtapasztaljam ezt az igazán összetett és bonyolult folyamatot. Fontos számomra, hogy egy valós, létező rendszert modellezek az alkalmazással, mivel így egy teljes alkalmazásfejlesztési folyamat útján tapasztalhatom meg a szoftverfejlesztés buktatóit - a modellezendő rendszer analízisétől az elkészült alkalmazás teszteléséig. Lényeges szempont az is, hogy olyan rendszert fejleszthessek, ahol a vállalat dolgozóinak közreműködésével, tapasztalataikat felhasználva készíthetem el ezen alkalmazást. Ezt azért tartom lényegesnek, mert véleményem szerint nem minden esetben veszik figyelembe a programozók a jövőbeni felhasználók igényeit. Ez által bonyolulttá válhat a vállalat alkalmazottai - azaz a felhasználók - számára az elkészült alkalmazás használata. Ezen alkalmazás fejlesztése témaválasztási szempontjaimnak teljességgel megfeleltek, így nem volt kérdés számomra a témaválasztás.

Az alkalmazás megtervezéséhez az UML modellező nyelvben leírt diagramtípusokat fogom használni (használati esetdiagram, osztály diagram), melyek sokat segíthetnek későbbi fejlesztési munkámban. A programozási feladatokat a JAVA objektumorientált programozási nyelv eszközeinek felhasználásával végzem el, mivel ezen nyelv alkalmazását tartom a legcélravezetőbbnek céljaim eléréséhez. Úgy gondolom, tanulmányaim befejezése után is hasznos lesz a fejlesztés során az objektumorientált programozási paradigma használatában, valamint a JAVA nyelv alkalmazásában szerzett fejlesztői tapasztalatom.

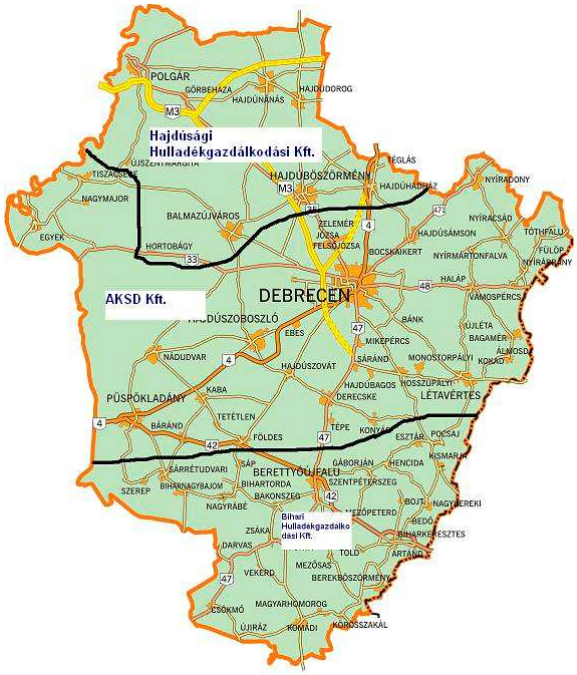
A környezet védelme számomra a mindennapok része. Minden embernek meg kellene értenie, hogy gyermekeink életét nehezítjük meg azzal, ha nem szánunk kellő időt és fáradtságos munkát erre a feladatra. A Hajdúsági Hulladékgazdálkodási Kft. munkatársai minden nap ezt szem előtt tartva végzik munkájukat, ezért alkalmazásom elkészítésével én is elmondhatom, hogy tettem valamit a környezet védelme érdekében.

2. Hajdú-Bihar megye hulladékgazdálkodása és a Hajdúsági Hulladékgazdálkodási Kft.

2.1. Hajdú-Bihar megye hulladékgazdálkodási rendszerének kialakulása

Az Európai Unióhoz való csatlakozás után Magyarországon környezetvédelmi és infrastrukturális beruházási programok indultak az ISPA program keretében. Többek között jelentős támogatást kapott Hajdú-Bihar megye is a Hulladékgazdálkodási Programjának megvalósításához.

A Program jelentős előrelépés a megye hulladékgazdálkodásában, hiszen hosszú távú megoldást nyújt a megye településein keletkező lakossági szilárd hulladékok biztonságos elhelyezésére, valamint a szelektív gyűjtés és válogatás révén elősegíti a hulladék újrahasznosítását. A projekt részeként megyei szintűre bővült a lakossági szelektív hulladékgyűjtési rendszer, valamint felszámolásra került több mint 60 megfelelő műszaki védelemmel nem rendelkező hulladéklerakó. A Programban megvalósuló beruházások 211 ezer háztartást szolgálnak ki, kezelve a megyében keletkező évenkénti több mint 500 ezer m^3 települési szilárd hulladékot és megoldva a hulladékok biztonságos kezelését és lerakását a következő évtized végéig. [6]



1. ábra: Hajdú-Bihar megye hulladékgyűjtő-területei
(forrás: [7] a régiók felosztásával kiegészítve)

Megyei szinten három hulladékgyűjtő körzetet alakítottak ki, valamint körzetenként egy-egy hulladékkezelő telep épült Debrecen, Hajdúböszörmény és Berettyóújfalu térségében. A három hulladékgazdálkodási központ három különböző és független gyűjtőterületet szolgál ki. A korszerű hulladékgazdálkodásba bevont lakosság száma Debrecenben 260 ezer fő [8], Hajdúböszörményben 120 ezer fő [9], Berettyóújfaluban 90 ezer fő [10]. A hajdúsági gyűjtőkörzet kezelésének feladatát a Hajdúsági Hulladékgazdálkodási Kft. látja el.

2.2. A Hajdúsági Hulladékgazdálkodási Kft.

A megyei hulladékgazdálkodási rendszer hajdúsági részét a körzet 14 településének társulása szervezte meg, Hajdúböszörmény vezetésével. A rendszer üzemeltetését pályázat nyomán a Hajdúböszörményi Városgazdálkodási Kft. nyerte el, s 2004. július 13-án közös vállalatot alapított az önkormányzati társulással Hajdúsági hulladékgazdálkodási Kft. néven. A Kft. nyerte el a Hajdúsági Hulladéklerakó- és Kezelő Telep üzemeltetési jogát is – melynek átadására 2004. október 21-én került sor Hajdúböszörmény külterületén – valamint a 14 településen a hulladékgazdálkodási közszolgáltatói feladatok ellátását az elkövetkezendő 20 évre. [9]

2. ábra: A Hulladékkezelő Telep légi felvételen
(forrás: [11])

2005. január 1-én indult a gazdasági tevékenység végzése, 12 településen a közszolgáltatást ettől az időponttól Hajdúsági Hulladékgazdálkodási Kft. végzi. Tiszagyulaházán 2006. februárjától, míg Hortobágyon 2005. júliusától indult a hulladékszállítás a Kft. által.

Település	**Terület (km2)**	**Lélekszám (fő)**	**Lakásállomány (db)**	**Gyűjtőszigetek száma (db)**
Balmazújváros	205,45	18149	6386	23
Hajdúböszörmény	370,78	32220	11547	44
Hajdúdorog	100,65	9640	3503	13
Hajdúhadház	91,90	13070	4126	14
Hajdúnánás	259,62	18235	6841	23
Polgár	97,46	8438	3287	8
Téglás	38,33	6385	2155	8
Bocskaikert	6,80	2637	912	2
Folyás	53,92	405	149	0
Görbeháza	80,20	2672	1052	4
Hortobágy	284,54	1737	646	4
Tiszagyulaháza	20,78	820	371	2
Újszentmargita	96,22	1591	648	4
Újtikos	35,29	956	382	2
Összesen	**1741,94**	**116955**	**42005**	**151**

1. Táblázat: települések terület, lélekszám, lakásállomány adatai, és a gyűjtőszigetek száma
(forrás: A HHG Kft. adatbázisából a társult önkormányzatok nyilvántartása alapján saját munka)

3. A modellezendő rendszer leírása

3.1. Gépjárműpark

A hulladékgazdálkodási tevékenységének elvégzéséhez a vállalat különböző létesítményein kívül kulcsszerepet kapnak a vállalat gépjárművei, melyek a hulladék transzportálását végzik. Elengedhetetlen fontosságú ezen járművek folyamatos rendelkezésre állása, valamint időszakos műszaki ellenőrzése a balesetek és fennakadások elkerülése végett. A feladat megoldása megoldhatatlan a gépjárművekről keletkező információk mindennapos gyűjtése és rögzítése nélkül.

A vállalat dolgozói ezt a közel sem egyszerű feladatkört az informatikai rendszer szinte teljes megléte nélkül végzik, azonban a feladatkör problémáinak megoldását nagyban elősegíthetné és felgyorsíthatná egy jól működő informatikai rendszer. Ezen informatikai alkalmazás megvalósítására kívánok kísérletet tenni szakdolgozatomban, bár tisztában vagyok vele, ez közel sem egyszerű feladat egy kezdő mérnök informatikus számára.

A Hajdúsági Hulladékgazdálkodási Kft. gépjárművei három csoportba sorolhatók: a hulladéklerakó belső hulladékmozgatását és kezelését megoldó gépjárművekre, a hulladékszállító járművekre valamint a kiegészítő járművekre (anyagbeszerző járművek, területi képviselők által használt járművek, stb.). Dolgozatomban csak a hulladékszállító gépjárművek futásteljesítményi adataival kívánok foglalkozni, de későbbi terveim között szerepel a másik két gépjárműcsoport adatkezelésének informatikai támogatása is.

A vállalat gépjárműparkjában 14 hulladékszállító jármű foglal helyet, melyek között legnagyobb számban az IVECO gyártmányú tehergépjárművek állnak rendelkezésre (8 db), ezeken kívül RENAULT (6 db) és MAN (1 db) gyártmányú tehergépjárművek végzik a szállítási feladatokat.

3.2. A gépjárművek futásteljesítményi adatainak kezelése

A gépjárművek általános és futásteljesítményi adatait a gépjármű-ügyintéző kezeli. Az általános adatok nyilvántartásához tartozik az egyes gépjárművek felvétele és törlése a gépjárművek listájáról, valamint a járművek kötelező időszakos műszaki vizsgálatainak elvégeztetése. A futásteljesítményi adatok kezeléséhez tartozó feladatok a következők:
Napi menetlevelek adatinak ellenőrzése és rögzítése, havi üzemanyag elszámolás készítése, valamint havi üzemanyag- és kilométer-teljesítmény összesítő készítése a menetlevelek adatai alapján.

3.3. Az informatikai rendszer

Az alkalmazottak adatait a Humán Információs Rendszer (HIR32) kezeli, amelynek főbb feladatai a következők: Bérszámfejtés, statisztikák készítése, létszámnyilvántartás, munkaóra-nyilvántartás, valamint a TB nyilvántartások kezelése.

A hulladéklerakóba beérkező szállítmányok adatit egy MIX-R nevű számítógépes program tarja nyílván, amely a hídmérleggel áll összeköttetésben. A rögzített adatok a következők: A beszállító neve, a beszállító jármű rendszáma, a beszállítás időpontja, a hulladék EWC kódja, a beérkező jármű bruttó súlya, valamint a kimenő jármű nettó súlya.

A gépjárművek futásteljesítményi adatait egyik alkalmazás sem tartja nyílván, így azok kezelése - amelyhez tartozó feladatokat az előző pontban felsoroltam - a gépjármű-ügyintéző feladata marad, amelyet informatikai segítség nélkül végez el.

Az általam készített alkalmazás ehhez a feladathoz nyújt támogatást, mely alkalmazást a későbbiekben szándékomban áll a fentebb említett rendszerekkel is kompatíbilissé tenni, mellyel egy komplexebb informatikai rendszer állna a vállalat rendelkezésére.

4. Az alkalmazás megtervezése, elvárások, diagramok

A szoftverfejlesztési folyamat végső célja, hogy a szoftvert az elvárt funkciókkal és teljesítménnyel kell a felhasználó számára szállítani, ezen felül karbantarthatónak, üzembiztosnak és használhatónak kell lennie. Azonban ezek paraméterek megvalósíthatatlanok az alkalmazás előzőleges megtervezése nélkül. A tervezésre fordított munka növekedésével csökken a fejlesztési folyamat során a problémák felmerülésének valószínűsége, valamint a modellezendő rendszer minél pontosabb megismerése segíti a programozót a készülő alkalmazás hatékonyságának növelésében. Mindezek mellett nem elhanyagolható szempont, hogy a kódolást is egyszerűbbé teszi a programozó számára egy igazán profin elkészített terv.

Az alkalmazás megtervezéséhez az UML modellező nyelvet használtam, mivel ez az általam legjobban ismert és ismereteim alapján széles körben használt modellező nyelv. Előnye, hogy egyszerű, mégis jól használható modell készítését teszi lehetővé, mely modell akár a programozásban nem jártas személyek számára is érthető lehet.

A tervezés folyamatában a legfontosabb feladat az alkalmazás modelljének elkészítése. A modellkészítéshez elengedhetetlen a modellezendő rendszer pontos megismerése, feltérképezése, mert a modellnek olyan absztrakciós szintet kell elérnie, amelyben eltekinthetünk a konkrét megvalósítási technikák zavaró részleteitől. Ennél fogva fontosnak tartottam, hogy a Hajdúsági Hulladékgazdálkodási Kft. dolgozóival szoros közreműködésben tervezzem meg az általam implementált alkalmazást és merítsek a munkájuk során felhalmozott tapasztalatokból. A kezdeti tervekből kiindulva - amelyeket főleg vizuális és szöveges eszközökkel készítettem el a megértés és használhatóság támogatása miatt - többszöri módosítás után jutottunk el végül a szoftver elkészítése során használt modellekhez. [1]

4.1. Az alkalmazásfejlesztés során elkészített UML diagramok

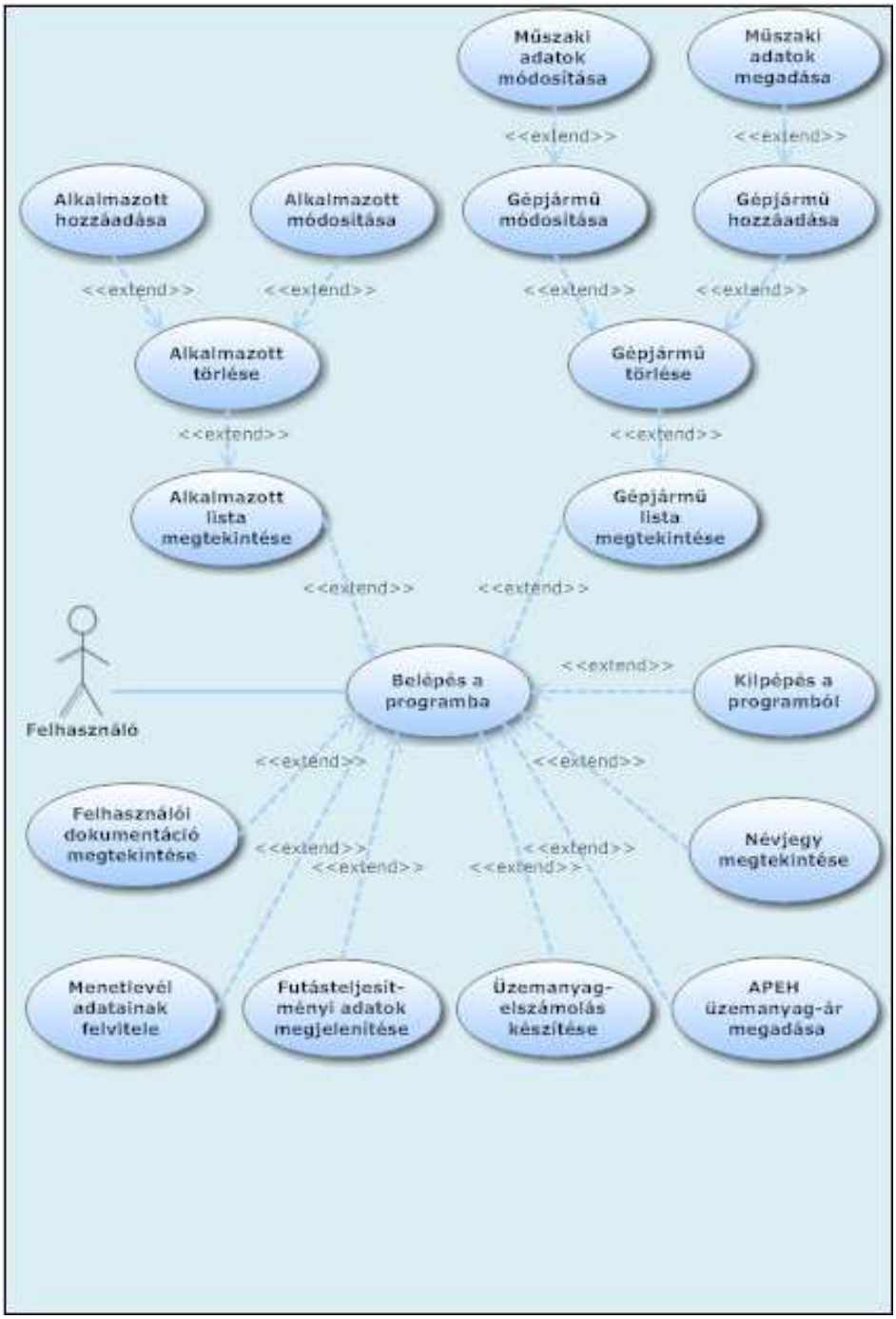
4.1.1. Használati eset diagram

Ha a rendszer környezetével való együttműködését modellezzük, egy olyan absztrakt megközelítést kell alkalmaznunk, amely nem részletezi túlságosan a rendszeren belüli kölcsönhatásokat. Az UML azt javasolja, hogy olyan használati eset modelleket fejlesszünk ki, amelyekben minden használati eset egy, a rendszerrel való együttműködést reprezentál.

Ezzel a diagrammal az alkalmazással szemben támasztott alapvető követelmények szemléltethetők, továbbá tisztázhatjuk a felhasználók kapcsolódási pontjait a rendszerhez. A kapcsolódási pontok esetén meg kell adni a különböző interakciókat – amelyek a program futását befolyásolhatják -, ezzel kiemelhetünk egyes résztevékenységeket.

A használati eset diagramon a felhasználókat és egyéb a rendszerhez kapcsolódó külső elemeket aktoroknak nevezzük. A tervezés első lépéseként meghatározhatóak ezen aktorok, melyek közé ez esetben egyedül a felhasználó került. A felhasználó az alkalmazás minden funkciójához hozzáférhet. A későbbiekben azonban jelszavas belépéshez kívánom kötni a program használatát, az egyes felhasználók hozzáférési jogainak kezelése végett. A rendszer funkcióit az ábrán látható ellipszisek írják le. Az elsődleges használati eset(ek)hez kapcsolódnak az aktorok, amely esetünkben a „Belépés” használati eset. A felsőbb szintű használati esetekhez <<extend>> sztereotípiával jelölt szaggatott vonalú nyíllal kötjük a hozzájuk kapcsolódó alsóbb szintű használati eseteket, amelyek a felsőbb szintű esetek változatai. Például az „Alkalmazott lista megtekintése” használati esethez tartozó változatok az „Új alkalmazott hozzáadása a listához”, az „Alkalmazott adatainak módosítása” és az „Alkalmazott törlése” használati esetek. [12] [13]

Az elkészített használati eset diagram alapján meghatároztam a rendszer elemeinek fejlesztési sorrendjét. A fejlesztés úgy a legésszerűbb, ha a felsőbb szintű használati esetek kódolásától az alsóbb szintű használati esetek kódolása felé haladok, így az előbbiek kialakítására, valamint tesztelésére több idő jut. Ez fontos, mivel ezek megbízhatósága elsődleges az alsóbb szintű esetek megbízhatóságával szemben.



3. ábra: Használati eset diagram
(forrás: saját munka)

4.1.2. Az osztály diagram

A másik UML diagram, amit elkészítettem, az osztály diagram, mely az alkalmazás osztályait, és a köztük fennálló kapcsolatokat írja le. A diagram részletes leírását az 5. fejezetben helyeztem el, valamint maga a diagram a függelékben található meg.

4.2. Az alkalmazásfejlesztés során elkészített szöveges dokumentumok

4.2.1. A rendszer funkciói

Az alkalmazás által biztosított funkciók tervének elkészítése segített a funkciók pontos meghatározásában. Az első tervekben általam felsorolt funkciókat a dolgozók segítségével pontosítottam, valamint kéréseik alapján több funkcióval is bővítettem a listát, melyek megkönnyíthetik munkájukat. Néhány funkciótól azonban el kellett tekintenem, mert a dolgozók úgy ítélték meg, hogy nincs azokra szükség vagy a megvalósításuk túlságosan bonyolult feladat lett volna jelenlegi szakmai tudásom mellett. A funkciók megadásának végleges verziója akár egyfajta felhasználói kézikönyvként is használható a későbbiekben.

- ***Belépés:***
  A rendszerbe való belépés.
- ***Kilépés:***
  A rendszerből való kilépés.
- ***Alkalmazott lista megtekintése:***
  A vállalat alkalmazotti listájának megjelenítése és a listából kiválasztott alkalmazott részletes adatainak megtekintése.
- ***Alkalmazott hozzáadása:***
  Új alkalmazott adatainak rögzítése a rendszerben.
- ***Alkalmazott adatainak módosítása:***
  A rendszerben rögzített alkalmazott adatainak módosítása.
- ***Alkalmazott adatainak törlése:***
  A rendszerben rögzített alkalmazott adatainak törlése a rendszerből.
- ***Gépjármű lista megtekintése:***
  A vállalat gépjármű listájának megjelenítése és a listából kiválasztott gépjármű adatainak megtekintése.
- ***Gépjármű hozzáadása:***
  Új gépjármű általános és műszaki adatainak rögzítése a rendszerben.
- ***Gépjármű adatainak módosítása:***
  A rendszerben rögzített gépjármű általános adatainak módosítása.
- ***Műszaki adatok módosítása:***
  A rendszerben rögzített gépjármű műszaki adatainak módosítása.
- ***Gépjármű adatainak törlése:***
  A rendszerben rögzített gépjármű adatainak törlése a rendszerből.

- ***Menetlevél adatainak felvitele:***
  A menetlevélre - a gépjárművezető által - felvitt adatok rögzítése a rendszerben.
- ***Gépjármű futásteljesítményi adatainak megtekintése:***
  A rendszerben rögzített futásteljesítményi adatok megjelenítése.
- ***Üzemanyag-elszámolás készítése:***
  Az üzemanyag elszámolás elkészítéséhez szükséges bementi adatok és az ezekből számított kimeneti adatok megjelenítése.
- ***APEH üzemanyagár megadása:***
  Az APEH által minden hónap elején közleményben megadott üzemanyagár rögzítése a rendszerben.
- ***Segítség:***
  A program használatához segítséget nyújtó felhasználói dokumentáció megjelenítése.
- ***Névjegy:***
  A program nevének és a készítő nevének megjelenítése.

4.2.2. Forgatókönyvek

A használati esetek részletesen nem fejtik ki az egyes funkciók működését, így a programozó és a felhasználók közötti félreértésekhez vezethetnek. Részletes leírásukkal lehetőségünk nyílik az előbb említett félreértések elkerülésére, valamint a rendszerrel szemben fennálló követelmények pontosítására. A használati esetek kifejtésének módja a forgatókönyvek elkészítése, amelyek az adott funkció eléréséhez szükséges aktorok és az alkalmazás között folyó párbeszédet írják le. A forgatókönyvek összehasonlítása az elkészült alkalmazás valós működésével a tesztelési fázisban hasznos lehet. A teljes forgatókönyv a mellékletben kapott helyet.

Fontosabb használati esetek:

- ***A felhasználó módosítja egy alkalmazott adatait:***
  - Megtekinti az alkalmazottak listáját.
  - Kiválasztja a módosítandó alkalmazottat a listáról.
  - Kiválasztja a Módosítás gombot.
  - Átírja a módosítandó adatokat.
  - Kiválasztja az OK gombot.

- ***A felhasználó hozzáad egy új gépjárművet a rendszerhez:***
  - Megtekinti a gépjárművek listáját.
  - Kiválasztja a hozzáadás gombot.
  - Megadja a gépjármű általános adatait.
  - Kiválasztja az OK gombot.
  - Megadja a gépjármű műszaki adatait.
  - Kiválasztja az OK gombot.
- ***A felhasználó töröl egy gépjárművet:***
  - Megtekinti a gépjárművek listáját.
  - Kiválasztja a törlendő gépjárművet a listáról.
  - Kiválasztja a Törlés gombot.
  - A figyelmeztető ablaknál az Igen gombot választja.
- ***A felhasználó felviszi egy menetlevél adatait:***
  - Megtekinti az alkalmazottak listáját.
  - Kiválaszt egy alkalmazottat a listáról.
  - A Menetlevél menün belül a Menetlevél adatainak felvitele menüpontot választja.
  - Megadja a menetlevél adatait.
  - Kiválasztja az OK gombot.
- ***A felhasználó a programmal elkészítteti az üzemanyag-elszámolást:***
  - Megtekinti az alkalmazottak listáját.
  - Kiválaszt egy alkalmazottat a listáról.
  - Az Üzemanyag-elszámolás menün belül az Üzemanyag-elszámolás készítése menüpontot választja.
  - Megjelenik az üzemanyag-elszámolás.
  - Kiválasztja az Ok gombot.

4.2.3. Fogalomszótár

Lényegesnek tartottam, hogy a fejlesztés során készüljön egy olyan szöveges dokumentum, amelyben a félreértések elkerülése végett pontos definíciót adok az alkalmazásban használt fogalmakhoz. Ezért elkészítettem az alkalmazás fogalomszótárát, amely a következő:

- ***Alkalmazott:***

  A vállalat által foglalkoztatott személy.

  Attribútumai:

  - Név
  - Születési hely
  - Születési idő
  - Anyja neve
  - TAJ-szám
  - Irányító szám
  - Város
  - Cím
  - Telefonszám
  - Munkakör: gépjárművezető, rakodó, területi képviselő, divízióvezető
  - Státusz: aktív vagy passzív

- ***Gépjármű:***

  A vállalat birtokában lévő vagy más gazdasági szervezettől bérelt gépjármű.

  Attribútumai:

  <u>Általános adatok:</u>

  - Rendszám
  - Gyártmány
  - Típus
  - Gyártási év
  - Önsúly
  - Összsúly
  - Felépítmény: az összsúly és az önsúly különbsége.
  - Rendeltetés: hulladékszállító, személyszállító, egyéb
  - Állapot: rendelkezésre áll, szerviz alatt, forgalomból kivonva.

Műszaki adatok:

- Alvázszám
- Hengerűrtartalom
- Teljesítmény
- Hajtóanyag
- Motorkód/szám
- Szín
- Jármű kategória
- Környezetvédelmi osztályba sorolás (kódszám)
- Ülésszám
- Sebváltó típusa (kódszám)
- Üzemanyag-fogyasztási alapnorma
- Műszaki érvényesség ideje

Futásteljesítményi adatok:

- Gépjárművezető neve
- Rendszám
- Kilométeróra kezdőállása
- Kilométeróra végállása
- Összes megtett kilométer
- Városi kilométer
- Távolsági kilométer
- Megyeszékhelyi kilométer
- Megtett fordulók száma
- Tankolt üzemanyag-mennyiség

➢ ***APEH üzemanyagár:***

Az APEH által minden hónap elején közleményben megadott üzemanyagár.

➢ ***Menetlevél:***

Az egyes gépjárművek vezetői által naponta a valós futásteljesítményi adatok alapján kitöltött nyomtatvány.

➢ ***Üzemanyag-elszámolás:***

Az összesített futásteljesítményi adatok alapján az egyes gépjárművek üzemanyag-fogyasztásáról készített kimutatás.

Jellemző adatai:

Bementi adatok:

- Gépjárművezető neve
- Gépjármű rendszáma
- Kilométeróra kezdőállása
- Kilométeróra végállása
- Összes megtett kilométer
- Városi kilométer
- Távolsági kilométer
- Megyeszékhelyi kilométer
- Tankolt üzemanyag-mennyiség
- Üzemanyag-fogyasztási alapnorma

Kimeneti adatok

- Felhasználható üzemanyag-mennyiség
- Megtakarítás vagy túlfogyasztás
- A megtakarított vagy túlfogyasztott üzemanyag ára

- ***Felhasználható üzemanyag-mennyiség:***

  Az üzemanyag-fogyasztási alapnorma és a megtett városi-, távolsági-, valamint megyeszékhelyi kilométerek alapján számolt mennyiség.

- ***Megtakarítás vagy túlfogyasztás:***

  A felhasználható üzemanyag-mennyiség és a tankolt üzemanyag-mennyiség különbsége. Ha pozitív szám, akkor megtakarításról beszélünk, ha negatív szám, akkor túlfogyasztásról.

- ***A megtakarított vagy túlfogyasztott üzemanyag ára:***

  Az előző pontban tárgyalt üzemanyag-mennyiségek ára, melyet az üzemanyag-mennyiség és az éppen érvényes APEH üzemanyagár szorzataként számíthatunk ki.

- ***APEH üzemanyagár:***

  Az APEH által minden hónap elején közleményben megadott üzemanyagár.

- ***Segítség:***

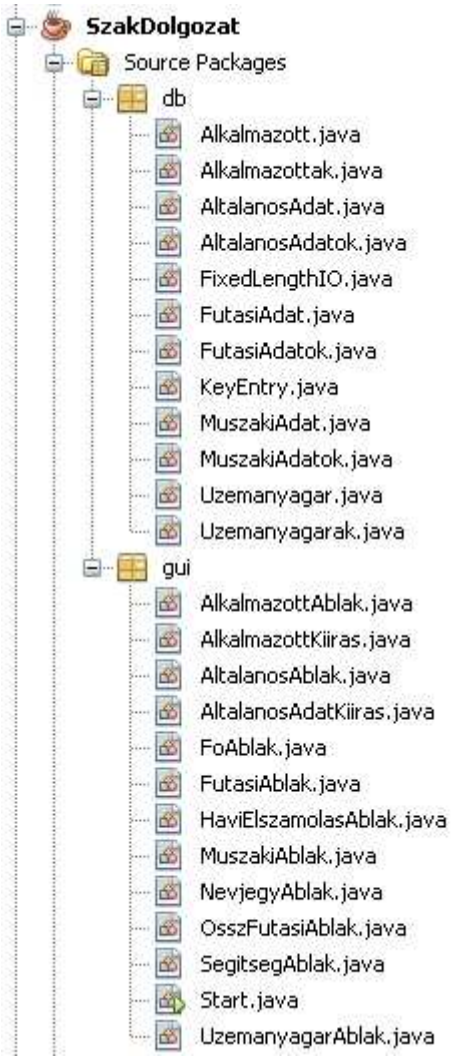
  A program használatához segítséget nyújtó felhasználói dokumentáció, amely egy ablakban megjeleníthető.

- ***Névjegy:***

  A program nevét és a készítője nevét megjelenítő ablak.

5. Az elkészült alkalmazás leírása a JAVA oldaláról

A csomagrendszer két, funkcionalitásában jól elkülöníthető részből áll: a **db** és **gui** csomagokból. A **db** csomag osztályai valósítják meg az adatok letárolását és visszaolvasását, valamint az adatok kezelését, így egyfajta adatbázis-kezelést valósítanak meg annak ellenére, hogy előre megírt adatbázis-kezelőt nem használ az alkalmazás. A **gui** csomag a felhasználói felületek létrehozását és megjelenítését valósítják meg. Az osztályok két külön csomagba való sorolásával a program forráskódja megítélésem szerint sokkal átláthatóbbá és könnyebben értelmezhetővé vált.



4. ábra: Az alkalmazás struktúrája
(forrás: saját munka)

Elkészítettem az alkalmazás osztály diagramját, amely a függelékben megtalálható.

Az osztály diagram felépítése a következő:

A téglalapok a program osztályait jelölik, a nyilak, pedig azok kapcsolódásait. A téglalapokon belül a felső tartomány az osztály nevét, a középső az osztály attribútumait, és az alsó az osztály metódusait tartalmazza. Egy osztályból kiinduló nyíl jelöli, hogy az osztály meghívja a nyíl által mutatott osztályt (vagy annak egy metódusát), az osztályra mutató nyíl, pedig azt jelöli, hogy az osztályt (vagy egy metódusát) meghívja az osztály, melyből a nyíl kiindul. Az osztályokat működés alapján csoportosítva helyeztem el (ahogy az a diagramon is látszik) a könnyebb kezelhetőség és az érthetőség biztosítása miatt. Az osztály diagram nagy segítséget nyújthat az alkalmazás működésének megértéséhez, ezért hangsúlyoznám részletes tanulmányozásának fontosságát. [2]

5.1. Az adatok kezelését megvalósító csomag

Ahogy már fentebb is említettem, az adatok kezelését a **db** csomag valósítja meg egyfajta adatbázis-kezelőként. A csomagban implementáltam az adatok fájlba írását és onnan visszaolvasását kezelő osztályt, valamint a feladat megoldásához elengedhetetlen kulcsbejegyzéseket létrehozó osztályt. Ezeken az osztályokon felül az adattábláknak megfelelő osztályok és az általuk definiált adatok kezelését megvalósító osztályok kerültek definiálásra a db csomagban.

A csomag osztályai alfabetikus sorrendben:

- **Alkalmazott**
- **Alkalmazottak**
- **AltalanosAdat**
- **AltalanosAdatok**
- **FixedLengtIO**
- **FutasiAdat**
- **FutasiAdatok**
- **KeyEnty**
- **MuszakiAdat**
- **MuszakiAdatok**
- **Uzemanyagar**
- **Uzemanyagarak**

5.1.1. A fájlba írást és fájlból olvasást kezelő osztály

A **FixedLengtIO** osztály valósítja meg az alkalmazásban a fájlba írást és fájlból olvasást:

```
class FixedLengthIO {

    public static String readString(DataInput in, int size) throws IOException {
        char c[]=new char[size];
        for(int i=0; i<size; i++)
            c[i]=in.readChar();
        return new String(c);
    }

    public static void writeString(DataOutput out, String s, int size)
                                                        throws IOException {
        char buffer[]=new char[size];
        int len=s.length();
        if(len > size)
            len=size;
        s.getChars(0, len, buffer, 0);
        for(int i=len; i<buffer.length; i++)
            buffer[i]=' ';
        out.writeChars(new String(buffer));
    }
}
```

Az osztály forráskódjában látható, hogy az osztálynak két metódusa van: a **readString** és a **writeString** metódusok. A **readString** valósítja meg a fájlból olvasást, mely a bemeneti adatfolyamról az olvasandó fájl méretig karaktereket olvas be egy új String változóba. A **writeString** metódus, pedig a fájlba írja egy kimeneti adatfolyamon keresztül a megadott String változóban tárolt karakterláncot.

5.1.2. A kulcsbejegyzéseket kezelő osztály

A kulcsbejegyzések kezelését a **KeyEntry** osztály valósítja meg:

```
public class KeyEntry implements Comparable {
    public String key;
    public long position;

    public KeyEntry(String key, long position) {
        this.key=key;
        this.position=position;
    }

    public int compareTo(Object o) {
        return key.compareTo(((KeyEntry)o).key);
    }

    public boolean equals(Object o) {
        return key.equals(((KeyEntry)o).key) &&
            position==((KeyEntry)o).position;
    }

    public String toString() {
        return key;
    }
}
```

Az osztály attribútumai a *key* és a *position* változók, melyek a kulcsbejegyzések kulcsértékét és pozícióját írják le.

Az osztálynak négy metódusa van: a **KeyEntry**, a Comparable interfészt implementáló **compareTo**, az **equals** valamint a **toString** metódusok. A **KeyEntry** metódus egy kulcsbejegyzést állít elő az attribútumai alapján. A **compareTo** metódus összehasonlítja két kulcsbejegyzés kulcsát és igaz értéket ad vissza, ha azok megegyeznek, egyébként hamis értéket ad vissza. Az **equals** metódus megvizsgálja, hogy két kulcsbejegyzés megegyezik-e – vagyis mind kulcsuk, mind a pozíciójuk azonos – és igaz értéket ad vissza, ha megegyezik a két kulcsbejegyzés, egyébként hamis értéket. A **toString** metódus, pedig visszaadja egy kulcsbejegyzés kulcsát.

5.1.3. Az adattáblákat implementáló osztályok

A rendszerben tárolt adatok jellemzőit az adattáblák szerepét betöltő osztályokban definiáltam, mely osztályok felelősek az egyes adatcsoportok példányainak létrehozásáért. Az implementált osztályok a következők:

Az alkalmazottak adatait leíró osztály:

```
public class Alkalmazott {
    public String nev;          //Az alkalmazott neve
    public String szulHely;     //születési helye
    public String szulIdo;      //születési ideje
    public String anyjaNeve;    //anyja neve
    public String TAJszam;      //személyi száma
    public String irszam;       //irányítószám
    public String varos;        //város
    public String cim;          //címe
    public String telefon;      //telefonszáma
    public String munkakor;     //munkaköre
    public String statusz;      //státusza

    //A mezők tárolt hosszai:
    public final static int NEV_MERET=25;
    public final static int SZULHELY_MERET=25;
    public final static int SZULIDO_MERET=15;
    public final static int ANYJANEVE_MERET=25;
    public final static int TAJSZAM_MERET=9;
    public final static int IRSZAM_MERET=4;
    public final static int VAROS_MERET=25;
    public final static int CIM_MERET=25;
    public final static int TELEFON_MERET=15;
    public final static int MUNKAKOR_MERET=20;
    public final static int STATUSZ_MERET=15;
```

Az alkalmazottak adatmezőit az **Alkalmazott** osztály definiálja. Amint a forráskód megjegyzéseiből is tisztán látható, definiálásra kerültek a fogalomszótárban fellelhető alkalmazottakra jellemző tulajdonságok, melyeket karakterlánc típusúként definiáltam a fájlba való letárolás megoldásának egyszerűsítése végett. Az egyes adatmezők hosszát is letároltam, mivel ez szükséges ahhoz, hogy véletlen elérésű állományokban el tudjuk azokat érni a pozíciójuk alapján.

Az alkalmazottak példányosítását a következő kódrészletben található két metódus végzi el:

```
    public Alkalmazott() {
        this("","","","","","","","","","","");
    }

    public Alkalmazott(String nev, String szulHely, String szulIdo,
                       String anyjaNeve, String TAJszam, String irszam,
                       String varos, String cim, String telefon,
                       String munkakor, String statusz) {
        this.nev=nev;
        this.szulHely=szulHely;
        this.szulIdo=szulIdo;
        this.anyjaNeve=anyjaNeve;
        this.TAJszam=TAJszam;
        this.irszam=irszam;
        this.varos=varos;
        this.cim=cim;
        this.telefon=telefon;
        this.munkakor=munkakor;
        this.statusz=statusz;
    }
}
```

Az első, paraméter nélküli példányosító metódus egy üres bejegyzésnek megfelelő alkalmazott objektumot hoz létre. Az üres bejegyzés adatmezőit a másik, paraméterezett példányosító metódus a paraméterként kapott adatok alapján feltölti értékekkel. A példányosítás eredményeként egy új alkalmazott objektum jön létre, melyet az adatfájl egy bejegyzéseként tárolunk el. A bejegyzés adatmezőinek értékei a későbbiekben módosíthatók és a bejegyzés törlésére is lehetőséget szolgáltat az alkalmazás.

Az alfejezet további részében csak a forráskód attribútumokat definiáló, valamint az azokhoz tartozó mezőhosszak nevesített konstansait leíró kódrészt mutatom be, mivel a példányosító metódusok az előbb tárgyalttal azonos működésűek.

A gépjárművek adatait leíró osztályokat csoportokra bontottam a kezelhetőség egyszerűsítése végett. A csoportok: általános adatok, műszaki adatok, valamint futásteljesítményi adatok.

Az ezeket leíró osztályok a következők:

A gépjárművek általános adatait leíró osztály:

```
public class AltalanosAdat {
    public String rendszam;          //A gépjármű rendszáma
    public String gyatmany;          //gyártmánya
    public String tipus;             //tipusa
    public String ev;                //gyártási éve
    public String onsuly;            //önsúlya
    public String osszsuly;          //összsúlya
    public String rendeltetes;       //rendeltetése
    public String allapot;           //állapota

    //A mezők tárolt hosszai:
    public static final int RENDSZAM_MERET=11;
    public static final int GYARTMANY_MERET=25;
    public static final int TIPUS_MERET=25;
    public static final int EV_MERET=4;
    public static final int ONSULY_MERET=5;
    public static final int OSSZSULY_MERET=5;
    public static final int RENDELTETES_MERET=25;
    public static final int ALLAPOT_MERET=20;
```

Az **AltalanosAdat** osztályban a gépjárművekre jellemző általános tulajdonságok alapján definiáltam az osztály attribútumait és a hozzájuk tartozó mezőhosszakat.

A gépjárművek műszaki adatait leíró osztály:

```
public class MuszakiAdat {
    public String alvazszam;          //A gépjármű alvázszáma
    public String hengerurtartalom;   //hengerűrtartalma
    public String teljesitmeny;       //teljesítménye
    public String hajtoanyag;         //hajtóanyaga
    public String motorkod;           //motorkódja/száma
    public String szin;               //színe
    public String kategoria;          //jármű kategóriája
    public String kornyOsztaly;       //környezetvédelmi osztálya
    public String ulesszam;           //üléseinek száma
    public String sebvalto;           //sebességváltó fajtája (kódszáma)
    public String alapnorma;          //üzemanyag-fogyasztási alapnormája
    public String muszakiErvenyesseg; //műszaki érvényességének ideje

    //A mezők tárolt hosszai:
    public static final int ALVAZSZAM_MERET=20;
    public static final int HENGERURTARTALOM_MERET=5;
    public static final int TELJESITMENY_MERET=5;
    public static final int HAJTOANYAG_MERET=20;
    public static final int MOTORKOD_MERET=20;
    public static final int SZIN_MERET=15;
    public static final int KATEGORIA_MERET=5;
    public static final int KORNYOSZTALY_MERET=2;
    public static final int ULESSZAM_MERET=2;
    public static final int SEBVALTO_MERET=2;
    public static final int ALAPNORMA_MERET=5;
    public static final int MUSZAKIERVENYESSEG_MERET=10;
```

A **MuszakiAdat** osztályban a gépjárművek műszaki adatai alapján definiáltam az osztály attribútumait és a hozzájuk tartozó mezőhosszakat.

A gépjárművek futásteljesítményi adatait leíró osztály:

```
public class FutasiAdat {
    public String gepjarmuVezeto;       //A gépjármű vezetőjének neve
    public String rendszam;             //a gépjármű rendszáma
    public String kezdoKm;              //a kilométeróra kezdőállása
    public String vegKm;                //a kilométeróra végállása
    public String megtettKm;            //ténylegesen megtett kilométer
    public String varosiKm;             //megtett városi kilométer
    public String tavolsagiKm;          //megtett távolsági kilométer
    public String mszekhelyiKm;         //megtett megyeszékhelyi kilométer
    public String fordulоszam;          //megtett fordulók száma
    public String uzemanyag;            //tankolt üzemanyag-mennyiseg

    //A mezők tárolt hosszai:
    public static final int GEPJARMUVEZETO_MERET=25;
    public static final int RENDSZAM_MERET=11;
    public static final int KEZDOKM_MERET=7;
    public static final int VEGKM_MERET=7;
    public static final int MEGTETTKM_MERET=5;
    public static final int VAROSIKM_MERET=5;
    public static final int TAVOLSAGIKM_MERET=5;
    public static final int MSZEKHELYIKM_MERET=5;
    public static final int FORDULOSZAM_MERET=3;
    public static final int UZEMANYAG_MERET=5;
```

A **FutasiAdat** osztályban a futásteljesítményi adatok definiálását a fogalomszótárban leírt futásteljesítményi adatok alapján adtam meg.

Az APEH üzemanyagár változóját leíró osztály:

Létrehoztam az **Uzemanyagar** osztályt - mely nem alkot külön adattáblát, hanem egy változót tárol -, ami az APEH üzemanyagár változót definiálja, valamint a mezőhosszt és a példányosító metódusokat adja meg.

```
public class Uzemanyagar {
    public String uzemanyagar;        //APEH üzemanyag ár

    public static final int UZEMANYAGAR_MERET=4;

    public Uzemanyagar() {
        this("");
    }

    public Uzemanyagar(String uzemanyagar) {
        this.uzemanyagar=uzemanyagar;
    }
}
```

5.1.4. Az adatkezelést megvalósító osztályok

Az előző pontban leírt osztályok csak az egyes egyedek létrehozását implementálják, azonban az ezekből az egyedekből létrehozott listák kezelését nem oldják meg. A megoldást a következő osztályok szolgáltatják:
Alkalmazottak, AltalanosAdatok, FutasiAdatok, MuszakiAdatok, Uzemanyagarak.

Ezek az osztályok felépítésben és működésben szinte teljesen megegyeznek, azzal a különbséggel, hogy a hozzájuk tartozó adatosztályokhoz alkalmazkodva vannak a műveleteik megadva.

A következő attribútumok szerepelnek az osztályokban:

- *filename:*
  Az adatokat tartalmazó fájl neve.
- *rafAlkalmazottak:*
  A véletlen elérésű fájl, amelyet az adatok tárolására létrehozunk.
- *alkKeyEntries:*
  A kulcsbejegyzéseket tároló vektor.

```
public class Alkalmazottak {
    private String fileName;
    private RandomAccessFile rafAlkalmazottak;
    public Vector alkKeyEntries=new Vector();
```

Az adatállomány megnyitása és a kulcsvektor felépítése:

A konstruktor létrehozza a fájl mappáját - ha az még nem létezik - majd az adott nevű fájlt megnyitja írás-olvasásra, és felépíti a kulcsvektort.

```
    public Alkalmazottak(String fileName) throws IOException {
        this.fileName=fileName;
        File f=new File(fileName);
        File dir=f.getParentFile();
        if(!dir.exists())
            dir.mkdir();

        rafAlkalmazottak=new RandomAccessFile(fileName,"rw");
        constructKeyEntries();
    }
```

Egy új alkalmazott objektumot a fájl végére ír:

Pozícionál a fájl végére, majd meghívja a privát **writeAlkalmazott** metódust. Összeállítja a kulcsbejegyzést, és beszúrja azt a kulcsvektorba, végül rendezi a kulcsvektort.

```
public void addAlkalmazott(Alkalmazott alkalmazott) throws IOException {
    long filePointer=rafAlkalmazottak.length();
    writeAlkalmazott(rafAlkalmazottak,alkalmazott,filePointer);
    alkKeyEntries.add(new KeyEntry(alkalmazott.nev,filePointer));
    Collections.sort(alkKeyEntries);
}
```

A névsor szerint az index-edik alkalmazott objektum adatainak lekérdezése:

Kiveszi az index-edik bejegyzést a kulcsvektorból, és az alkalmazott objektumot a benne szereplő fájlpozíció alapján beolvassa. Meghívja a privát **readAlkalmazott** metódust.

```
public Alkalmazott getAlkalmazott(int index)
                        throws IndexOutOfBoundsException, IOException {
    KeyEntry keyEntry=(KeyEntry)alkKeyEntries.get(index);
    return readAlkalmazott(rafAlkalmazottak,keyEntry.position);
}
```

A névsor szerint az index-edik alkalmazott objektum törlése az állományból:

Kiveszi az index-edik bejegyzést a kulcsvektorból, és az alkalmazott adatait a benne szereplő fájlpozíció alapján törli (érvényteleníti az adatait). A kulcsvektorból törli a kulcsot.

```
public void removeAlkalmazott(int index) throws IOException {
    KeyEntry keyEntry=(KeyEntry)alkKeyEntries.get(index);
    deleteAlkalmazott(rafAlkalmazottak,keyEntry.position);
    alkKeyEntries.remove(index);
}
```

A névsor szerint index-edik alkalmazott adatainak módosítása:

Kiveszi az index-edik bejegyzést a kulcsvektorból, és a benne szereplő fájlpozíció alapján felülírja az ott szereplő alkalmazott adatait. Meghívja a privát **wrieAlkalmazott** metódust. Kicseréli a kulcsbejegyzés kulcsát (azt is módosíthatták), és a kulcsvektort rendezi.

```
public void replaceAlkalmazott(Alkalmazott alkalmazott, int index)
                                                    throws IOException {
    KeyEntry keyEntry=(KeyEntry)alkKeyEntries.get(index);
    writeAlkalmazott(rafAlkalmazottak,alkalmazott,keyEntry.position);
    keyEntry.key=alkalmazott.nev;
    Collections.sort(alkKeyEntries);
}
```

A kulcsvektor feltöltése:

A fájlpozíciót a törzsállomány elejére állítjuk. Az alkalmazottak adatait beolvassuk sorban, ehhez meghívjuk a privát **writeAlkalmazott** metódust. Ha a metódus visszatérési értéke nem null (vagyis az objektum érvényes), akkor összeállítjuk a kulcsbejegyzést, és beszúrjuk a kulcsvektorba. Ha a fájl végére érünk, akkor EOFException kivétel keletkezik, ekkor rendezzük a kulcsvektort.

```
private void constructKeyEntries() throws IOException {
    try {
        rafAlkalmazottak.seek(0);
        Alkalmazott alkalmazott;
        long filePointer;
        while(true){
            filePointer=rafAlkalmazottak.getFilePointer();
            alkalmazott=readAlkalmazott(rafAlkalmazottak);
            if(alkalmazott!=null)
                alkKeyEntries.add(new KeyEntry(
                                        alkalmazott.nev,filePointer));
        }
    }
    catch(EOFException e) {
        Collections.sort(alkKeyEntries);
    }
}
```

A fájl bezárása (a logikailag törölt objektumok elhagyása):

A metódus sűríti a fájlt, vagyis elhagyja a logikailag törölt objektumokat. Felülírja az érvényes objektumokat a kulcsvektorban található kulcsbejegyzések alapján egy ideiglenes állományba („adatok/”temp.dat”), majd mindkét fájlt lezárja. Kitörli a régi fájlt („adatok/alkalmazottak.dat”), az új fájlt pedig átnevezi a régire. Végül egyetlen sűrített fájlunk lesz, amelynek neve az eredeti: „adatok/alkalmazottak.dat”.

```
public void close() throws IOException {
    RandomAccessFile temp=null;
    Alkalmazott alkalmazott;
    temp=new RandomAccessFile("adatok/temp.dat","rw");
    temp.setLength(0);
    rafAlkalmazottak.seek(0);
    for(int i=0; i<alkKeyEntries.size(); i++){
        KeyEntry keyEntry=(KeyEntry)alkKeyEntries.get(i);
        alkalmazott=readAlkalmazott(rafAlkalmazottak,keyEntry.position);
        writeAlkalmazott(temp,alkalmazott);
    }
    rafAlkalmazottak.close();
    temp.close();

    File regi=new File(fileName);
    regi.delete();
    File uj=new File("adatok/temp.dat");
    uj.renameTo(regi);
}
```

Az alkalmazott objektum logikai törlése:

A megadott fájlpozíción lévő alkalmazott objektum érvényes kódját érvénytelenre állítja.

```
private void deleteAlkalmazott(RandomAccessFile raf,
                               long filePointer) throws IOException {
    raf.seek(filePointer);
    raf.writeBoolean(false);
}
```

A megadott fájlpozíciótól kezdve egy alkalmazott objektum beolvasása (Null-t ad vissza, ha az objektum érvénytelen):

```
private Alkalmazott readAlkalmazott(RandomAccessFile raf,
                                    long filePointer) throws IOException {
    raf.seek(filePointer);
    return readAlkalmazott(raf);
}
```

Az aktuális fájlpozíciótól kezdve egy alkalmazott objektum beolvasása az állományból:

Annyi bájtot olvas be pontosan, amennyi az objektum pontos mérete. Ehhez meghívja a **FixedLengthIO** osztály szövegbeolvasó metódusát. A visszatérési érték null lesz, ha az objektum nem érvényes.

```
private Alkalmazott readAlkalmazott(RandomAccessFile raf)
                                                    throws IOException {
    boolean valid=raf.readBoolean();
    String nev=FixedLengthIO.readString(raf,
                                    Alkalmazott.NEV_MERET).trim();
    String szulHely=FixedLengthIO.readString(raf,
                                    Alkalmazott.SZULHELY_MERET).trim();
    String szulIdo=FixedLengthIO.readString(raf,
                                    Alkalmazott.SZULIDO_MERET).trim();
    String anyjaNeve=FixedLengthIO.readString(raf,
                                    Alkalmazott.ANYJANEVE_MERET).trim();
    String TAJszam=FixedLengthIO.readString(raf,
                                    Alkalmazott.TAJSZAM_MERET).trim();
    String irszam=FixedLengthIO.readString(raf,
                                    Alkalmazott.IRSZAM_MERET).trim();
    String varos=FixedLengthIO.readString(raf,
                                    Alkalmazott.VAROS_MERET).trim();
    String cim=FixedLengthIO.readString(raf,
                                    Alkalmazott.CIM_MERET).trim();
    String telefon=FixedLengthIO.readString(raf,
                                    Alkalmazott.TELEFON_MERET).trim();
    String munkakor=FixedLengthIO.readString(raf,
                                    Alkalmazott.MUNKAKOR_MERET).trim();
    String statusz=FixedLengthIO.readString(raf,
                                    Alkalmazott.STATUSZ_MERET).trim();
    if(valid)
        return new Alkalmazott(nev, szulHely, szulIdo, anyjaNeve,
                               TAJszam, irszam, varos, cim,
                               telefon, munkakor, statusz);
    else
        return null;
}
```

Az aktuális pozíciótól kezdve az alkalmazott objektum kiírása a véletlen elérésű állományba:

Annyi bájtot ír ki pontosan, amennyi az objektum pontos mérete. Ehhez meghívja a **FixedLengthIO** osztály szövegkiíró metódusát.

```
    private void writeAlkalmazott(RandomAccessFile raf,
                                  Alkalmazott alkalmazott) throws IOException {
        raf.writeBoolean(true);
        FixedLengthIO.writeString(raf,
                        alkalmazott.nev, Alkalmazott.NEV_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.szulHely, Alkalmazott.SZULHELY_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.szulIdo, Alkalmazott.SZULIDO_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.anyjaNeve, Alkalmazott.ANYJANEVE_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.TAJszam, Alkalmazott.TAJSZAM_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.irszam, Alkalmazott.IRSZAM_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.varos, Alkalmazott.VAROS_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.cim, Alkalmazott.CIM_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.telefon, Alkalmazott.TELEFON_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.munkakor, Alkalmazott.MUNKAKOR_MERET);
        FixedLengthIO.writeString(raf,
                        alkalmazott.statusz, Alkalmazott.STATUSZ_MERET);
    }
```

A megadott pozíciótól kezdve egy alkalmazott objektum kiírása az állományba:

```
    private void writeAlkalmazott(RandomAccessFile raf,
            Alkalmazott alkalmazott, long filePointer) throws IOException {
        raf.seek(filePointer);
        writeAlkalmazott(raf, alkalmazott);
    }
}
```

A fájlba írást és fájlból olvasást végző többi osztály is az **Alkalmazottak** osztály metódusainak működésben megfelelő metódusokat használnak az általuk kezelt objektumok attribútumainak megfelelően.

5.2. A grafikus felhasználói felületet létrehozó csomag

A grafikus felhasználói felület felépítéséért és megjelenítéséért felelős az alkalmazás másik csomagja a **gui** csomag, valamint a program indítását végző osztály is itt kapott helyet. Az ablakelemek – címkék, szövegmezők, nyomógombok, stb. – létrehozásának bemutatását nem szándékozom különösebben tárgyalni, mivel az alkalmazás lényegi tevékenységét nem ezek határozzák meg, bár a program használatához elengedhetetlenek. Forráskódjuk a függelékben megtekinthető. A csomag osztályai alfabetikus sorrendben a következők:

- ***AlkalmazottAblak:***
  Alkalmazott adatainak hozzáadását vagy módosítását kezelő panelt létrehozó osztály.
- ***AlkalmazottKiiras:***
  A listából kiválasztott alkalmazott adatait megjelenítő panelt létrehozó osztály.
- ***AltalanosAblak:***
  Gépjárművek általános adatainak hozzáadását vagy módosítását kezelő panelt létrehozó osztály.
- ***AltalanosAdatKiiras:***
  A listából kiválasztott gépjármű általános adatait megjelenítő panelt létrehozó osztály.
- ***FoAblak:***
  A főablakot létrehozó és megjelenítő, valamint a hozzá tartozó dialógusablakokat példányosító osztály.
- ***FutasiAblak:***
  A menetlevél adatainak felvitelét kezelő ablakot létrehozó osztály.
- ***HaviElszamolasAblak:***
  Az üzemanyag-elszámolást megjelenítő ablakot létrehozó osztály.
- ***MuszakiAblak:***
  A műszaki adatokat megjelenítő ablakot létrehozó osztály.
- ***NevjegyAblak:***
  A névjegy ablakot létrehozó osztály.
- ***OsszFutaiAblak:***
  Az összesített futási adatokat megjelenítő ablakot létrehozó osztály.
- ***SegitsegAblak:***
  A segítség ablakot létrehozó ablak.
- ***Start:***
  A program indítását végző osztály.
- ***UzemanyagarAblak:***
- Az APEH üzemanyagár megadását kezelő ablakot létrehozó osztály.

5.2.1. Az alkalmazottak adatait megjelenítő ablakot létrehozó osztály

Ahogy az előzőekben említettem, ezen osztályok működését nem kívánom részletezni, azonban szükségszerűnek ítélem meg néhány metódus bemutatását a **gui** csomagból is, az alkalmazás működésének könnyebb megértése végett.
Az előző pontban az alkalmazott objektumok kezelésén szemléltettem a db csomag osztályainak működését, így itt is az alkalmazott objektumok adatainak megjelenítését és kezelését kívánom bemutatni, amelyet az **AlkalmazottAblak** osztály kezel.

Az ebből a szempontból lényeges metódusok a következők:

```
public boolean showDialog(Alkalmazott alkalmazott) {
    setAlkalmazott(alkalmazott);
    ok=false;
    tfNev.requestFocus();
    show();
    if(ok)
        getAlkalmazott(alkalmazott);
    return ok;
}
```

A **showDialog** metódus attribútumként megkap egy alkalmazott objektum, amely adatai alapján meghívja a **setAlkalmazott** metódust, mely az ablak mezőinek értékeit beállítja.

```
public void setAlkalmazott(Alkalmazott alkalmazott) {
    tfNev.setText(alkalmazott.nev);
    tfSzulHely.setText(alkalmazott.szulHely);
    tfSzulIdo.setText(alkalmazott.szulIdo);
    tfAnyjaNeve.setText(alkalmazott.anyjaNeve);
    tfTAJszam.setText(alkalmazott.TAJszam);
    tfIrszam.setText(alkalmazott.irszam);
    tfVaros.setText(alkalmazott.varos);
    tfCim.setText(alkalmazott.cim);
    tfTelefon.setText(alkalmazott.telefon);
    cobMunkakor.setSelectedItem(alkalmazott.munkakor);
    cobStatusz.setSelectedItem(alkalmazott.statusz);
}
```

A beállítás után a **show( )** metódus megjeleníti az alkalmazott adatait kezelő dialógusablak a képernyőn. Abban az esetben, ha az adatok módosítása szükséges, akkor az ablak szövegmezőiben szereplő értékek módosításával teheti meg azt a felhasználó. Az *ok* egy logikai változó, melynek értéke dönti el, hogy a felvitt adatok mentésre kerüljenek vagy nem. Az ablak OK gombjának lenyomásakor ez az érték igaz lesz, a Mégse gomb lenyomásával, pedig hamis értéket kap (vagyis csak az ok gomb lenyomásakor rögzítjük az új adatokat).

Ha az imént leírt módon a *ok* logikai érték igaz, akkor a **getAlkalmazott** metódus kerül meghívásra, mely az alkalmazott objektum értékeit felvitt adatoknak megfelelően állítja be.

```
private void getAlkalmazott(Alkalmazott alkalmazott) {
    alkalmazott.nev=tfNev.getText().trim();
    alkalmazott.szulHely=tfSzulHely.getText().trim();
    alkalmazott.szulIdo=tfSzulIdo.getText().trim();
    alkalmazott.anyjaNeve=tfAnyjaNeve.getText().trim();
    alkalmazott.TAJszam=tfTAJszam.getText().trim();
    alkalmazott.irszam=tfIrszam.getText().trim();
    alkalmazott.varos=tfVaros.getText().trim();
    alkalmazott.cim=tfCim.getText().trim();
    alkalmazott.telefon=tfTelefon.getText().trim();
    alkalmazott.munkakor=(String)cobMunkakor.getSelectedItem();
    alkalmazott.statusz=(String)cobStatusz.getSelectedItem();
}
```

A nem alkalmazott típusú objektumok kezelése is az előbb bemutatott módon történik, kivételt a futásteljesítményi adatok kezelése képez:

```
private void getFutasiAdat1(FutasiAdat futasiAdat) {
    futasiAdat.gepjarmuVezeto=tfGepjarmuvezeto.getText().trim();
    futasiAdat.rendszam=(String)cobRendszam.getSelectedItem();
    futasiAdat.vegKm=tfVegKm.getText().trim();
    int km=Integer.parseInt(tfVegKm.getText().trim())-
                    Integer.parseInt(futasiAdat.kezdoKm);
    futasiAdat.megtettKm=Integer.toString(km);
    int varosKm=Integer.parseInt(futasiAdat.varosiKm)+
                    Integer.parseInt(tfVarosiKm.getText().trim());
    futasiAdat.varosiKm=Integer.toString(varosKm);
    int tavolsagKm=Integer.parseInt(futasiAdat.tavolsagiKm)+
                    Integer.parseInt(tfTavolsagiKm.getText().trim());
    futasiAdat.tavolsagiKm=Integer.toString(tavolsagKm);
    int mszekhelyKm=Integer.parseInt(futasiAdat.mszekhelyiKm)+
                    Integer.parseInt(tfMszekhelyiKm.getText().trim());
    futasiAdat.mszekhelyiKm=Integer.toString(mszekhelyKm);
    int fordulo=Integer.parseInt(futasiAdat.forduloszam)+
                    Integer.parseInt(tfForduloszam.getText().trim());
    futasiAdat.forduloszam=Integer.toString(fordulo);
    int tankolas=Integer.parseInt(futasiAdat.uzemanyag)+
                    Integer.parseInt(tfUzemanyag.getText().trim());
    futasiAdat.uzemanyag=Integer.toString(tankolas);
}
```

Amint a forráskódban is látható a felvitt adatok itt matematikai transzformációk után kerülnek letárolásra, mivel itt az új adatok nem felülírják a régi adatokat, hanem az új és a régi érték összege tárolódik le.

5.2.2. A főablakot létrehozó osztály

A **FoAblak** osztály – mely a főablakot létrehozó osztály - néhány metódusának leírása elengedhetetlen, mivel ez az osztály a főablak felületi elemeinek létrehozásán kívül a leszármazott ablakok megjelenítéséért is felelős, valamint az adatfájlok megnyitásában és a havi üzemanyag-elszámolás elkészítésében is kulcsszerepet játszik.

A működés szempontjából fontos metódusok:

A **Fajlok** metódus az adatfájlokban tárolt adatok alapján meghívja az objektumlistákat példányosító konstruktorokat, így megnyitva az adatfájlokat. Ha az alkalmazás valamely adatfájlt nem tud megnyitni, akkor azt jelzi a felhasználónak.

```
public void Fajlok() {
    try {
        alkalmazottak=new Alkalmazottak("adatok/alkalmazottak.dat");
        altalanosAdatok=new AltalanosAdatok(
                                        "adatok/altalanosadatok.dat");
        muszakiAdatok=new MuszakiAdatok(
                                        "adatok/muszakiadatok.dat");
        futasiAdatok=new FutasiAdatok(
                                        "adatok/futasiadatok.dat");
        uzemanyagarak=new Uzemanyagarak(
                                        "adatok/uzemanyagar.dat");

    }
    catch(IOException ex) {
        JOptionPane.showMessageDialog(this,
        "Hiba a fájl megnyitásakor!","", JOptionPane.ERROR_MESSAGE);
        System.exit(0);
    }
}
```

Az adatok hozzáadása, módosítása valamint törlése a **db** csomagban leírt metódusok meghívásával történik.

Példaként álljon itt az új alkalmazott adatbázishoz való hozzáadását meghívó metódus, melynek működése a következő: A metódus meghívása után elsőként létrehoz egy üres bejegyzést az alkalmazottak adatait tároló fájlban, majd megjeleníti egy új ablakot, melybe az új alkalmazott adatait felviheti a felhasználó.

Az adatok mentésekor a metódus a név szövegmező értékére két vizsgálatot végez el:
Az elsőben megvizsgálja, hogy a mező üres-e, és ha üres, akkor mentés nélkül tér vissza a főablakba, egyébként menti az adatokat. A második vizsgálatban ellenőrzi, hogy a felvitt név nem szerepel-e már az alkalmazottak kulcsértékei között - amelyek az alkalmazottak nevei -, és ha a név már egyszer szerepel, akkor a felhasználót értesíti a hibáról, majd mentés nélkül visszatér a főablakba. Erre a vizsgálatra azért van szükség, mert a kulcsértékeknek egyedieknek kell lennie minden bejegyzésre nézve. Ha nem talál hibát a metódus, akkor hozzáadja az új alkalmazott adatait a bejegyzésekhez. Ez után az alkalmazottak listáját frissíti - ez által már az új alkalmazott nevei is szerepelni fog benne - és újra megjeleníti azt a főablakban. A fájl írása közben keletkező bármely hiba esetén a metódus értesíti a felhasználót a hibáról.

```
public void alkalmazottatHozzaad() {
    Alkalmazott alkalmazott=new Alkalmazott();
    ablakAlkalmazott.cobMunkakor.setSelectedIndex(0);
    ablakAlkalmazott.cobStatusz.setSelectedIndex(0);
    if(ablakAlkalmazott.showDialog(alkalmazott)){
        try {
            if(alkalmazott.nev.isEmpty())
                return;
            for(int i=0; i<alkalmazottak.nevek.length; i++){
                if(alkalmazottak.nevek[i].equals(alkalmazott.nev)){
                    JOptionPane.showMessageDialog(this,"Az alkalmazott " +
                            "már létezik!","",JOptionPane.ERROR_MESSAGE);
                    return;
                }
            }
            alkalmazottak.addAlkalmazott(alkalmazott);
            listaAlkalmazottak.setListData(alkalmazottak.alkKeyEntries);
            alkalmazottatMutat();
            listaAlkalmazottak.requestFocus();
        }
        catch(IOException ex) {
            JOptionPane.showMessageDialog(this,"Hiba a fájl " +
                                "írásakor!","",JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Az üzemanyag-elszámolás adatait a program a beolvasott futásteljesítményi adatok alapján matematikai műveletek útján számítja ki, mivel ezeket nem tárolja az alkalmazás, mert az felesleges lenne.

A futásteljesítmény után felhasználható üzemanyag-mennyiséget a megtett kilométerek és a gépjármű üzemanyag-alapnormájának súlyozott szorzataként kapjuk meg. Ebből származtatható az üzemanyag-megtakarítás értéke, ami a futásteljesítmény után felhasználható üzemanyag-mennyiség és a ténylegesen felhasznált üzemanyag-mennyiség különbségeként számítható ki. Ezeket a számításokat következőképp kezeli a program:

```
ablakHavielszamolas.tfAlapnorma.setText(muszakiAdat.alapnorma);
double alapnorma=Double.parseDouble(
            ablakHavielszamolas.tfAlapnorma.getText().trim());
int varosKm=Integer.parseInt(futasiAdat.varosiKm);
int tavolsagKm=Integer.parseInt(futasiAdat.tavolsagiKm);
int mszekhelyKm=Integer.parseInt(futasiAdat.mszekhelyiKm);
int fordulo=Integer.parseInt(futasiAdat.forduloszam);
System.out.println(varosKm*((1.9*alapnorma)/100));
System.out.println(tavolsagKm*(alapnorma/100));
System.out.println(mszekhelyKm*((1.25*alapnorma)/100));
System.out.println(fordulo*0.2);
double futasuzemanyag=varosKm*((1.9*alapnorma)/100) +
                      tavolsagKm*(alapnorma/100) +
                      mszekhelyKm*((1.25*alapnorma)/100) +
                      fordulo*0.2;
ablakHavielszamolas.tfFutasiuzemanyag.setText(
                szamformazas(Locale.ENGLISH, futasuzemanyag));
double megtakaritasLiter=futasuzemanyag-
                         Integer.parseInt(futasiAdat.uzemanyag);
```

A beolvasott értékeket egész vagy valós értékűvé kell konvertálni ahhoz, hogy elvégezhessük velük a matematikai műveleteket. A részeredményeket a felhasználó számára nem teszem láthatóvá, de a program a konzolra kiírja azokat az ellenőrzés végett. A futásteljesítmény után felhasználható üzemanyag-mennyiség kiszámításának formulája a kódból kiolvasható. Az üzemanyag-megtakarítást az előző érték és a ténylegesen felhasznált üzemanyag-mennyiség különbségeként számítja ki az alkalmazás A számformázás metódus az eredményt két tizedes jegyűvé alakítja, majd megjelenítésre kerül az új szám.

5.2.3. Az alkalmazás indítását végző osztály

A program indítását a **Start** osztály kezeli, amely a főablak egy új példányát hozza létre, ezzel megjelenítve az ablakot:

```
public class Start {
    public static void main(String[] args) {
        new FoAblak();
    }
}
```

5.3. Az elkészült adattáblák leírása

Bár az alkalmazás működéséhez előre definiált adatbázis-kezelőt nem használ, a program használhatatlan lenne adattáblák definiálása, valamint adatkezelő metódusaik implementálása nélkül. Az általam készített adattáblák a következők:

Az **Alkalmazott** tábla:

A tábla az alkalmazottak adatait tárolja. Az elsődleges kulcs a *nev* adatmező, amely az alkalmazottak nevét tárolja. Ennek egyedinek kell lennie minden alkalmazott esetén. Ezen kívül minden alkalmazotthoz tároljuk a születési helyét, születési idejét, anyja nevét, TAJ-számát, a várost, ahol lakik és annak irányítószámát, lakcímét, a telefonszámát, betöltött munkakörét és státuszát egy-egy adatmezőben. A tábla összes adatmezőjét karakterlánc típusúként definiáltam.

Az **AltalanosAdat** tábla:

Ebben az adattáblában tárolom a gépjárművek általános adatait. A *rendszam* mező az elsődleges kulcs, amely a gépjármű rendszámát tárolja. Az adattábla mezői által tárolt további adatok: gyártmány, típus, gyártási év, önsúly, összsúly, rendeltetés, állapot. Az Alkalmazott táblához hasonlóan a tábla adatmezőit karakterlánc típusúként definiáltam.

A **MuszakiAdat** tábla:

A tábla a gépjárművek műszaki adatait tárolja. Elsődleges kulcsa az *alvazszam* mező, mely a gépjármű alvázszámát tárolja. Adatmezői a következő adatokat tárolják: alvázszám, hengerűrtartalom, teljesítmény, hajtóanyag, motorkód/szám, szín, jármű kategória, környezetvédelmi osztályba sorolás, ülésszám, sebváltó típusa, üzemanyag-fogyasztási alapnorma, műszaki érvényesség ideje. A tábla adatmezői karakterlánc típusúak.

A **FutasiAdat** tábla:

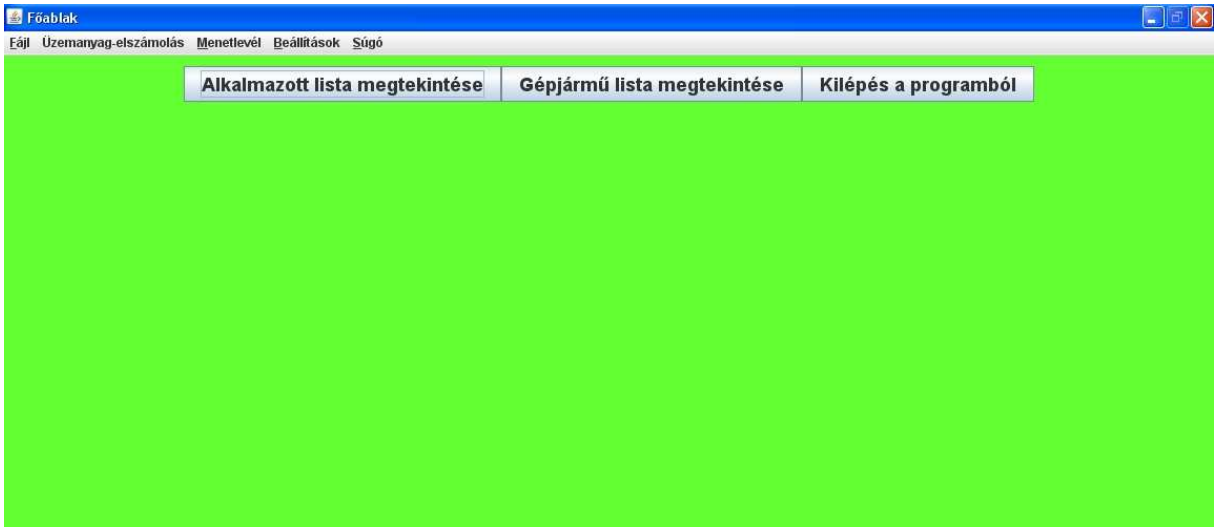
Ebben a táblában a gépjárművek futásteljesítményi adatait tárolom. A *gepjarmuvezeto* az elsődleges kulcs - ami a gépjármű vezetőjének nevét tartalmazza - külső kulcsa, pedig a *rendszam* mező - mely a gépjármű rendszámát tárolja. A további adatmezők által tárolt adatok: kilométeróra kezdőállása, kilométeróra végállása, összes megtett kilométer, megtett városi kilométer, megtett távolsági kilométer, megtett megyeszékhelyi kilométer, megtett fordulók száma, tankolt üzemanyag-mennyiség. A tábla mezői itt is karakterlánc típusúként kerültek definiálásra, azonban a felhasználó csak egész számot adhat meg az adatok felvitelét biztosító ablak szövegmezőiben, így a program egész típusúként kezelheti az adatokat.

6. Az elkészült alkalmazás leírása a felhasználó oldaláról

Egy alkalmazás használhatósági szempontból talán legfontosabb eleme a grafikus felhasználói felület, mivel a felhasználó ezzel szembesül, valamint rajta keresztül éri el a rendszer funkcióit (azaz itt folyik az információáramlás). Bármennyire is profin van elkészítve egy program működése, ha az nem párosul jól kezelhető felhasználói felülettel. Az alkalmazás megértését kívántam egyszerűbbé tenni azzal, hogy a különböző típusú objektumok adatait kezelő ablakok különböző háttérszínnel jelennek meg a képernyőn. Az alkalmazottak adatait kezelő ablakok háttérszíne kék, a gépjárművek általános és műszaki adatait kezelő ablakok sárga és a futásteljesítményi adatokat kezelő ablakok, valamint a főablak zöld színűek. Az alkalmazás a „szakdolgozat.jar" fájl megnyitásával futtatható, ami az alkalmazást tartalmazó CD gyökérkönyvtárában található meg.

6.1. A főablak

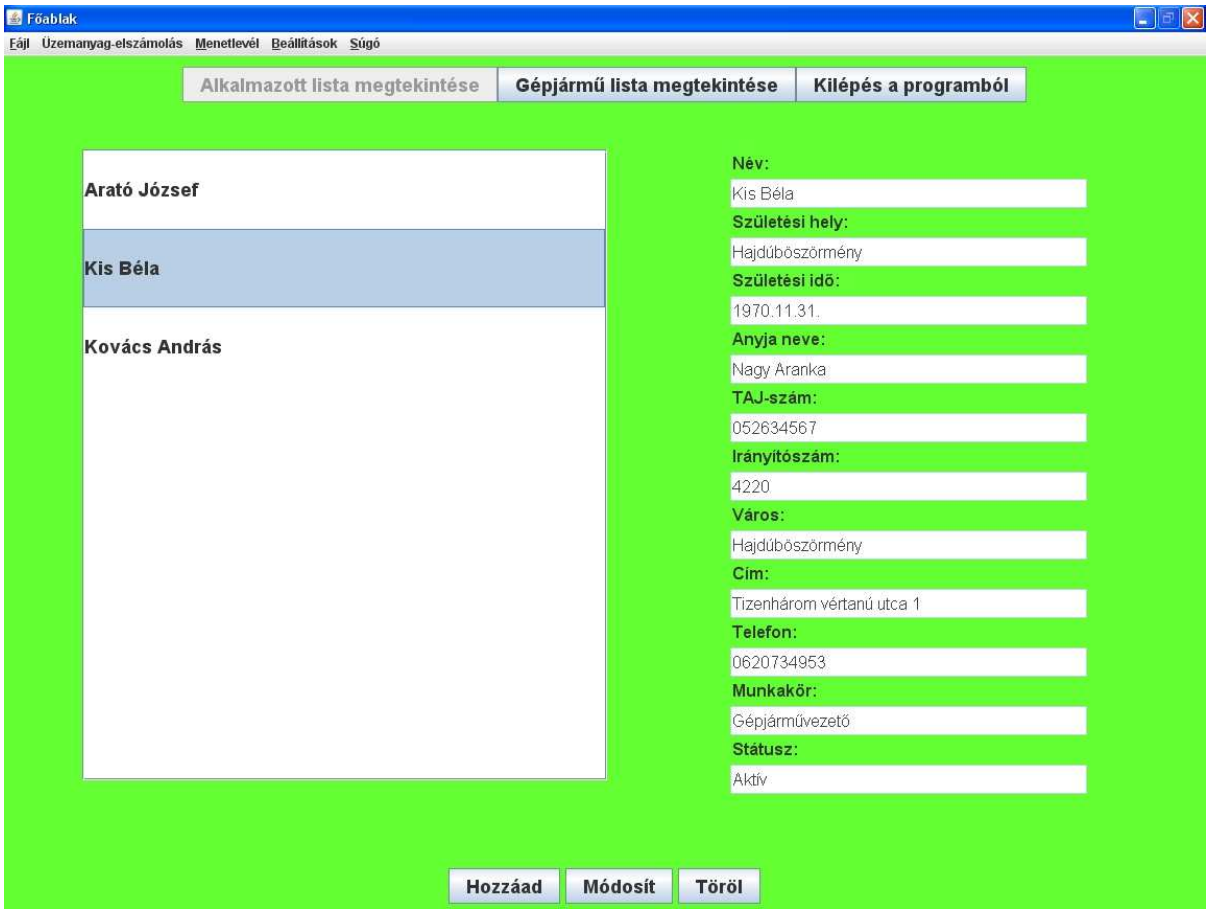
Az alkalmazás indításakor a főablak jelenik meg, melyből a rendszer funkció a nyomógombokon és a Menük menüpontjain keresztül közvetve vagy közvetlenül elérhetők.



5. ábra: Az alkalmazás főablaka (részlet)
(forrás: saját munka)

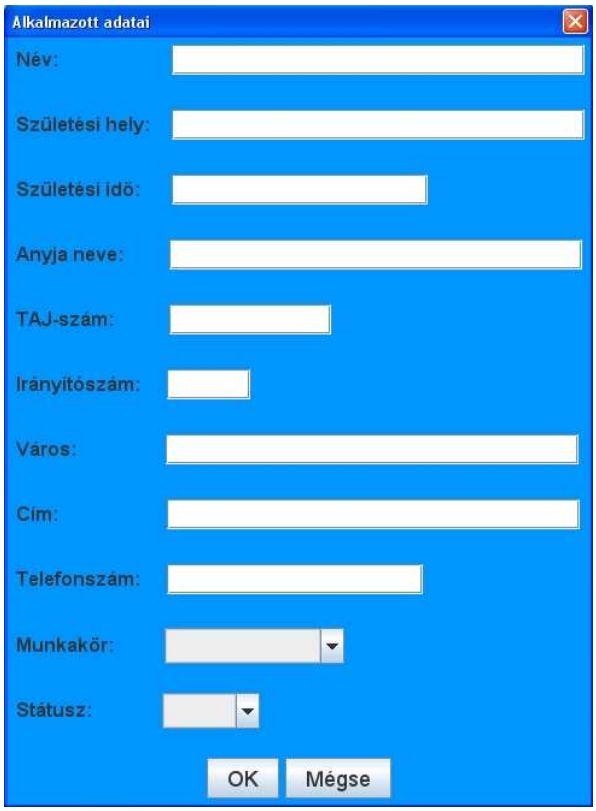
6.2. Az alkalmazottak adatainak kezelése

A felhasználó az „Alkalmazott lista megtekintése” gombra kattintva megjelenítheti a vállalatnál dolgozók listáját. A listán az alkalmazottak nevei jelennek meg alfabetikus sorrendben. A listáról valamely alkalmazottat kiválasztva, megjelennek annak adatai (név, születési hely, születési idő, anyja neve, TAJ-szám, irányítószám, város, cím, telefon, munkakör, státusz). A következő funkciók állnak a felhasználó részére a lista módosítására: A rendszerhez hozzáadhatunk új alkalmazottat, vagy egy már szereplő alkalmazott adatait módosíthatjuk, illetve törölhetjük az alkalmazottat a rendszerből.



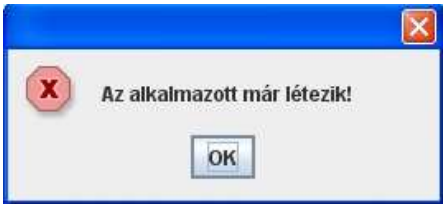
6. ábra: Alkalmazott adatinak megjelenítése
(forrás: saját munka)

Új alkalmazott hozzáadása a rendszerhez a „Hozzáadás” gombra kattintással válik lehetségessé. Ekkor egy új ablak jelenik meg, ahol az új alkalmazott adatait megadhatjuk. A felvitt adatok elmentését az „OK” gombra való kattintással fogadhatjuk el, abban az esetben azonban, ha valamilyen okból kifolyólag nem szeretnénk a felvitt adatainkat rögzíteni a rendszerben, akkor a „Mégse” gomb segítségével léphetünk vissza a főablakba az adatok rögzítése nélkül.



7. ábra: Új alkalmazott hozzáadása
(forrás: saját munka)

Az új alkalmazott hozzáadásakor felléphető kulcshiba (melyet az 5.2.2. pontban már kifejtettem) esetén egy dialógusablak jelenik meg, ami értesíti a felhasználót a hibáról.



8. ábra: Alkalmazott hozzáadási hibára figyelmeztető ablak
(forrás: saját munka)

A már rendszerben rögzített alkalmazottak adatainak módosítására is lehetőség nyílik a programban. A „Módosítás” gomb lenyomásával az alkalmazottak listájából kiválasztott személy adatait módosíthatjuk.

9. ábra: Alkalmazott adatainak módosítása
(forrás: saját munka)

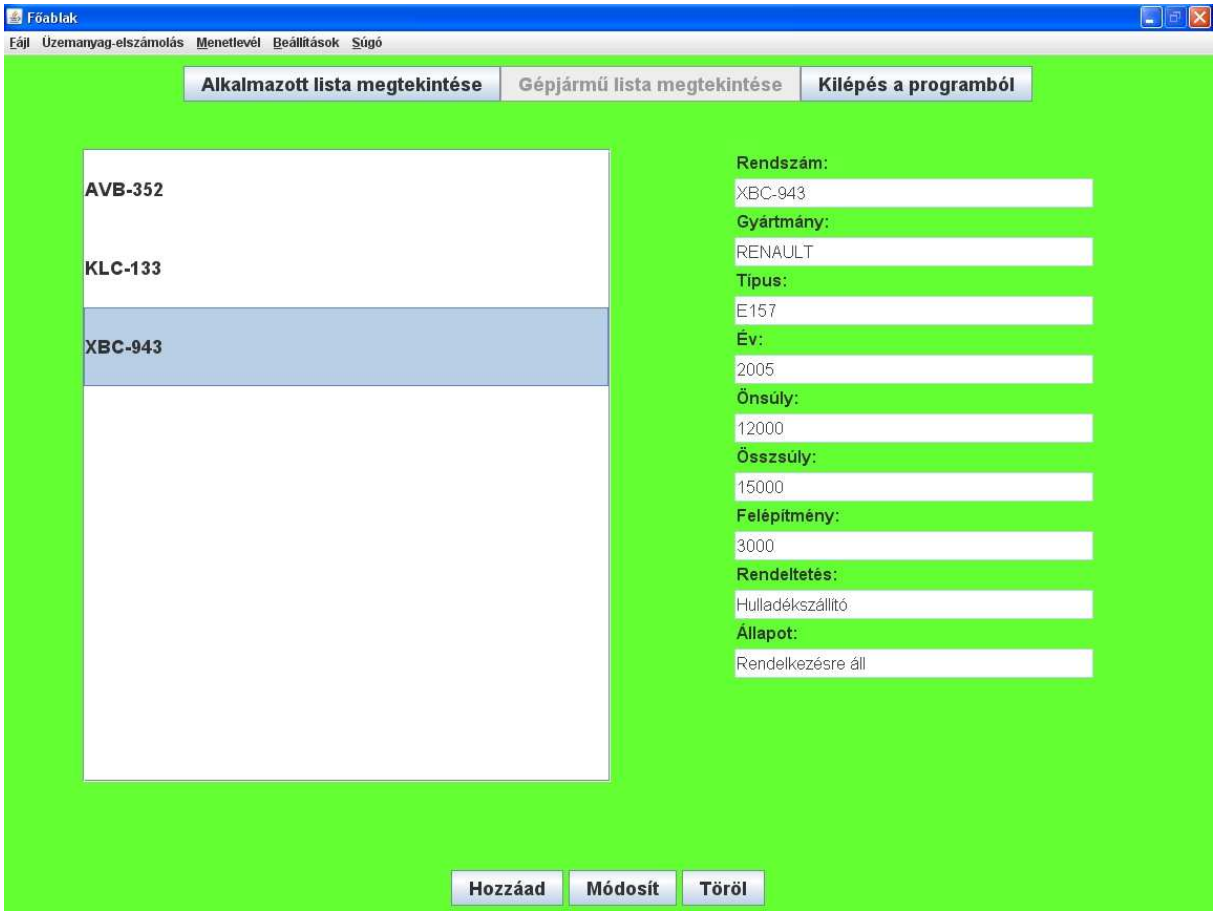
Egy – már rögzített – alkalmazott adatait törölhetjük is a „Töröl” gomb lenyomásával. A választás után megjelenik egy figyelmeztető ablak, ahol meg kell erősítenünk a rendszert döntésünk helyességéről. Ekkor még visszavonhatjuk törlési parancsunkat a „Nem” gomb kiválasztásával, azonban ha teljesen biztosak vagyunk szándékunkban, akkor az „Igen” gombra való kattintással törölhetjük a listából kiválasztott alkalmazottat.

10. ábra: Alkalmazott törlése
(forrás: saját munka)

6.3. A gépjárművek adatainak kezelése

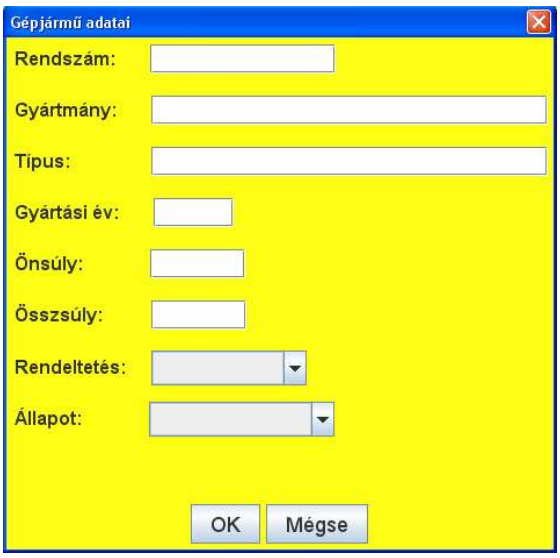
6.3.1. Általános és műszaki adatok kezelése

A vállalat gépjárműveinek listáját a „Gépjármű lista megtekintése” gombra kattintva érheti el a felhasználó. A gépjárművek listája a gépjárművek rendszámait tartalmazza alfabetikusan rendezve az alkalmazottak listájához hasonlóan. A lista valamely elemét kiválasztva a hozzá tartozó általános adatok jelennek meg (rendszám, gyártmány, típus, gyártási év, önsúly, összsúly, felépítmény, rendeltetés, állapot). A gépjárművek műszaki adatait nem jeleníti meg a főablakban az alkalmazás, mivel azok megtekintésére nagyon ritkán van szüksége a felhasználónak. A rendszerben azonban ezek az adatok is tárolásra kerültek, mivel a gépjárművek adatai nem lennének teljesek a műszaki adatok tárolása nélkül. A felhasználó itt is az alkalmazottak listájának kezelésénél megismert rendszerfunkciókat használhatják.



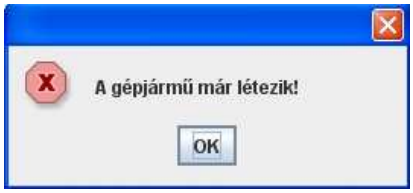
11. ábra: Gépjármű általános adatainak megjelenítése
(forrás: saját munka)

Új gépjármű adatainak rögzítésekor az általános adatokkal együtt, a műszaki adatokat is megadhatjuk. Új gépjármű hozzáadásához a „Hozzáadás” gombra kell kattintania a felhasználónak, amelynek hatására a gépjármű általános adatainak rögzítésére szolgáló ablak jelenik meg. Az általános adatok felvitelét követően két lehetséges lépés közül választhat a felhasználó: az „OK” gomb lenyomásával a felhasználó rögzíti az általános adatokat, vagy a „Mégse” gomb kiválasztásával visszalép a rögzítéstől.



12. ábra: Új gépjármű hozzáadása (általános adatok)
(forrás: saját munka)

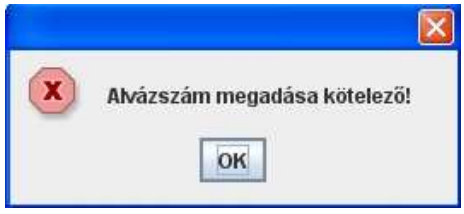
Abban az esetben, ha a felhasználó az „OK” gombot választja, akkor két lehetséges működés lehetséges. Az egyik esetben egy újabb ablak jelenik meg, ahol a gépjármű műszaki adatai adhatók meg (alvázszám, hengerűrtartalom, teljesítmény, hajtóanyag, motorkód/szám, szín, járműkategória, környezetvédelmi osztályba sorolás, ülések száma, sebességváltó típusa, üzemanyag-fogyasztási alapnorma, műszaki érvényesség ideje). A másik esetben, ha olyan rendszámot adott meg a felhasználó, ami már egy másik gépjárműhöz van rendelve, akkor az új alkalmazott hozzáadásánál ismertetett módon jár el a program.



13. ábra: Gépjármű hozzáadási hiba
(forrás: saját munka)

14. ábra: Új gépjármű hozzáadása (műszaki adatok)
(forrás: saját munka)

A műszaki adatok felvitele csak akkor érvényes, ha a felhasználó a gépjármű alvázszámát megadja - mivel ez a **MuszakiAdat** tábla elsődleges kulcsa -, ellenkező esetben a rendszer figyelmeztet a hibára, és visszalép az új gépjármű műszaki adatainak hozzáadását kezelő ablakba.



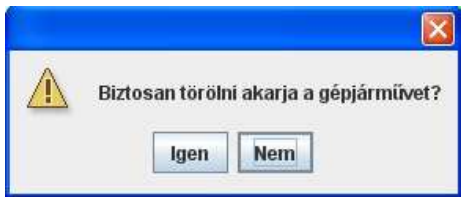
15. ábra: Műszaki adat hozzáadási hiba
(forrás: saját munka)

Gépjármű adatainak módosítását a „Módosítás” gombbal érhetjük el. A gomb lenyomása következtében megjelenő ablakban átírhatjuk a listából kiválasztott gépjármű általános adatait és a „Műszaki adatok megadása" jelölőmező állapotának függvényében megadhatjuk, hogy a gépjármű műszaki adatait kívánjuk-e módosítani.

16. ábra: Gépjármű adatainak módosítása (általános adatok)
(forrás: saját munka)

17. ábra: Gépjármű adatainak módosítása (műszaki adatok)
(forrás: saját munka)

A „Törlés” gomb lenyomása után a gépjármű adatainak törlése az ismertetett módon történik.



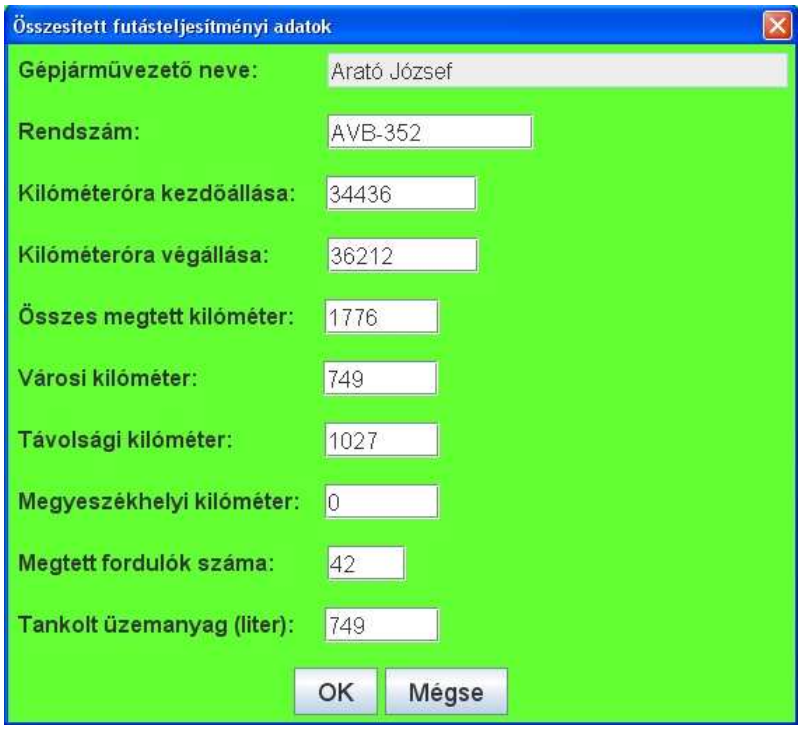
18. ábra: Gépjármű törlése
(forrás: saját munka)

6.3.2. Gépjárművek futásteljesítményi adatainak kezelése

A felhasználónak lehetősége nyílik a gépjárművezetők által kitöltött menetlevelek adatainak felvitelére (gépjárművezető neve, gépjármű rendszáma, kilométeróra kezdőállása, kilométeróra végállása, megtett városi kilométer, megtett távolsági kilométer, megtett megyeszékhelyi kilométer, megtett fordulók száma, tankolt üzemanyag-mennyiség). Ez a funkció a „Menetlevél” menü „Menetlevél adatinak felvitele” pontjával érhető el. Ekkor megjelenik az ábrán látható ablak, melyre felvihetjük a menetlevél adatait, amelyet az „OK” gomb kiválasztásával rögzíthetünk, vagy a „Mégse” gombbal elvethetjük a rögzítést. A „Gépjárművezető neve” szövegmező nem módosítható, a rendszám egy legördülő listából választható, a további szövegmezők csak számkaraktert fogadnak el bementként, mivel a hozzájuk tartozó adatmezőket mind szám típusúként kezeli az alkalmazás.

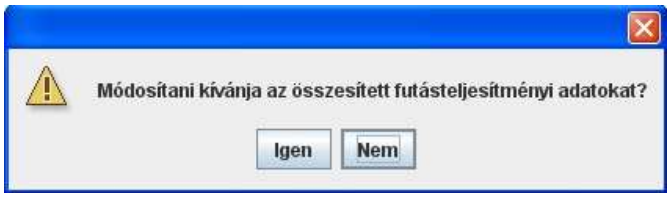
19. ábra: Menetlevél adatainak felvitele
(forrás: saját munka)

A felvitt menetlevelek adatait a rendszer összesítve tárolja el. Ezen adatok megtekintéséhez a felhasználónak a „Menetlevél” menü „Összesített futásteljesítményi adatok megjelenítése” menüpontra kell kattintania. Ekkor megjelenik az összesített futásteljesítményi adatok megjelenítő ablak, amelyről leolvashatók a kért adatok, valamint, ha hibásan vitte fel a felhasználó valamely menetlevél adatait, akkor ebben az ablakban módosíthatja az összesített értékeket. A mezők, amelyek a menetlevél felviteli ablakban csak számkaraktert fogadtak el bemenetként, azok ebben az esetben is csak ezzel a megszorítással használhatók.



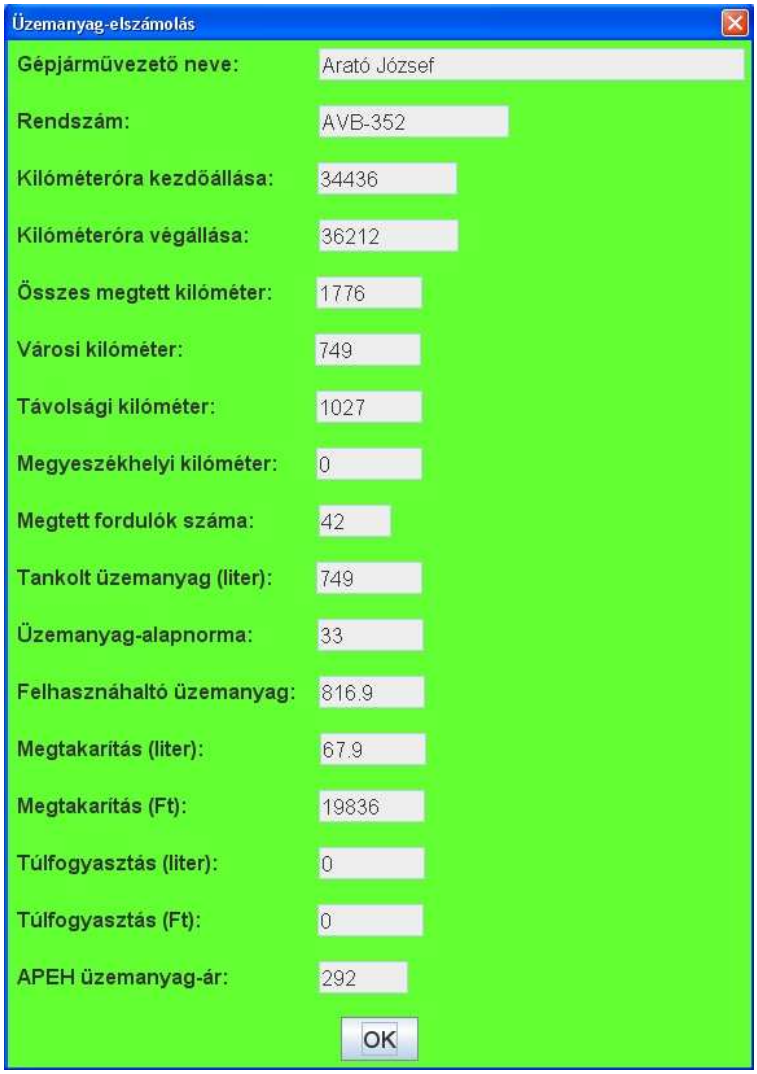
20. ábra: Összesített futásteljesítményi adatok megjelenítése
(forrás: saját munka)

Az „OK” gomb lenyomása után az alkalmazás újabb megerősítést vár a felhasználótól, hogy biztosan módosítani akarja-e az összesített futásteljesítményi adatokat. Ha a felhasználó a megjelenő dialógusablak „Igen” gombját választja, akkor megtörténik az adatmezők értékeinek módosítása, egyébként nem változik értékük.



21. ábra: Az adatok módosítására figyelmezető ablak
(forrás: saját munka)

Az alkalmazás egyik fő funkcióját, az üzemanyag-elszámolás készítését az „Üzemanyag-elszámolás” menü „Üzemanyag-elszámolás készítése” pontjának kiválasztásával érhetjük el. Ekkor a rendszer a futásteljesítményi adatok alapján elkészíti az üzemanyag-elszámolást, mely a bemeneti adatok alapján megadja a futásteljesítmény után felhasználható üzemanyag mennyiségét, valamint az üzemanyag-megtakarítást vagy a túlfogyasztást literben és forintban is kifejezve.



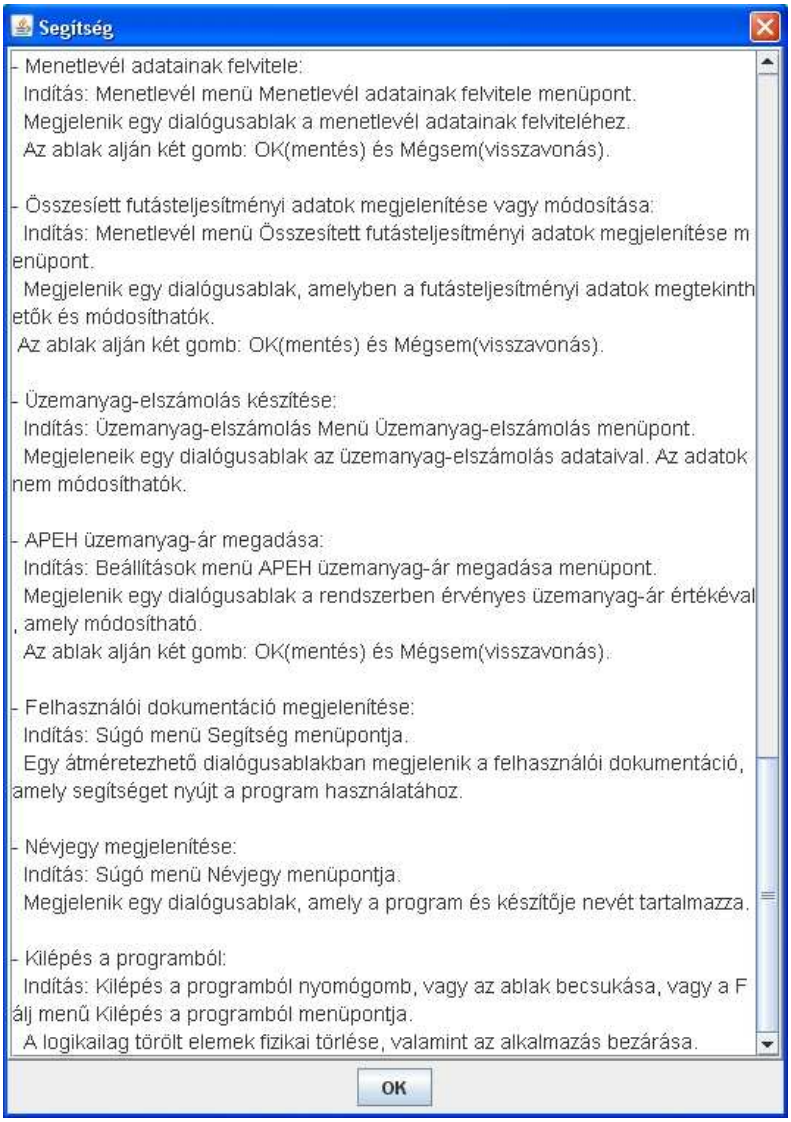
22. ábra: Üzemanyag-elszámolás
(forrás: saját munka)

6.4. Az alkalmazás egyéb funkciói

A felhasználónak az előzőekben leírt elsődleges rendszerfunkciókon kívül lehetősége nyílik másodlagos funkciók elérésére is. Ezen funkciók a következők:

- ***A felhasználói dokumentáció megtekintése:***

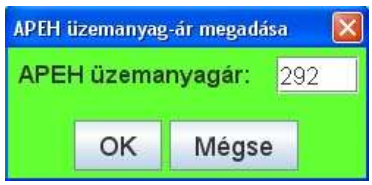
  A „Súgó" menü „Segítség" pontjával érhető el. Az új ablak tartalma egy szöveges állományból beolvasott szöveg lesz, amely az alkalmazás használatához segítséget nyújtó felhasználói dokumentációt tartalmazza.



23. ábra: A felhasználói dokumentáció (részlet)
(forrás: saját munka)

- ***Az APEH üzemanyagár megadása:***

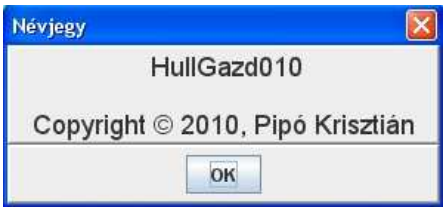
  A „Beállítások” menü „APEH üzemanyagár beállítása” menüpontjával érhető el. Az új ablakban megjelenő érték a rendszerben érvényes üzemanyagár, amely módosításával adható meg az aktuális érték.



24. ábra: APEH üzemanyagár megadása
(forrás: saját munka)

- ***A névjegy ablak megjelenítése:***

  A „Súgó” menü „Névjegy” pontjával érhető el. A dialógusablakban a program neve és a készítő neve jelenik meg.



25. ábra: Az alkalmazás névjegye
(forrás: saját munka)

- ***A programból való kilépés:***

  A felhasználó több módon is elérheti ezt a rendszerfunkciót. A főablak „Kilépés a programból” gombjára kattintva, a „Fájl” menü „Kilépés” menüpontját választva, valamint a főablak jobb felső sarkában lévő „X” gombra való kattintással. A kilépéskor az érvénytelenített objektumok végső törlése (a fájlok sűrítése) és az összes megnyitott fájl bezárása is végbe megy.

26. ábra: Kilépő ablak
(forrás: saját munka)

7. Az elkészült alkalmazás továbbfejlesztési lehetőségei

Az alkalmazás lehetséges továbbfejlesztési aspektusai:

- *Felhasználói felület fejlesztése*
- *A felhasználói igények alapján felmerülő új funkciók bevezetése*
- *Nyomtatási lehetőségek biztosítása*
- *Chipkártyák beépítése a vállalat gépjárműveibe (a gépjárművek futásteljesítményi adatinak közvetlen felvitele az adatbázisba)*
- *A gépjárművek GPS rendszerrel történő követésének implementálása a rendszerben*
- *Az alkalmazás felkészítése a vállalat további informatikai rendszereiből származó adatok saját adatbázisba való bevitelére*
- *Az alkalmazás adatainak bevitele a vállalat további informatikai rendszereinek adatbázisába.*
- *A rendszer állapotainak időszakos archiválása*
- *A hiányos adatmezőkkel rendelkező bejegyzések listázásának implementálása a rendszerben*
- *Adatbázis megszorítások továbbfejlesztése*
- *Webes felületen keresztüli elérés biztosítása*
- *A belépés felhasználói szerepkörökhöz való kötése*

Természetesen a továbbfejlesztés irányát meghatározó döntés előtt elsődlegesen a tényleges felhasználói igényeket érdemes figyelembe venni. A továbbfejlesztés célja a rendszer megbízhatóságának, alkalmazhatóságának növelése, valamint a már meglévő rendszerekkel való együttes alkalmazásának elősegítése.

8. Összefoglalás

Alkalmazásom elkészítése folyamán olyan tapasztalatokkal lettem gazdagabb, amelyek megítélésem szerint elengedhetetlenek egy kezdő mérnök informatikus számára. Biztos vagyok benne, hogy későbbi munkáim során nagy hasznát fogom venni az új ismereteknek, valamint megismerve egy összetett alkalmazásfejlesztési folyamat buktatóit, nem fogok újra ugyanazokba a hibákba esni, melyek megnehezítették munkámat.

Az elsődleges tervekből többszöri egyeztetés és módosítás útján jutottam el végül, a programozás folyamán felhasznált modellhez. Az UML diagramok használata nagy segítség volt, mivel sokkal könnyebbé vált számomra a valós rendszer működésének megértése általa. A használati eset diagram nagyban elősegítette későbbi munkámat, mivel átláthatóvá tette a használati esetek kapcsolatait, és ez által meghatározta a rendszer elemeinek fejlesztési prioritását. A diagram a program működésének vizuális szemléltetésével, a vállalat dolgozóival való kapcsolattartást is nagyban megkönnyítette. A rendszer struktúrájának felépítéséhez az osztály diagram első verziójának elkészítésével, majd annak folyamatos bővítésével, javításával jutottam el. Az alkalmazás osztály diagramja szakdolgozatom informatikai ismeretekkel rendelkező olvasójának is megkönnyítheti a rendszer felépítésének és működésének megértését. Az előző állítások tükrében biztos vagyok abban, hogy a jövőbeni szoftverfejlesztési projektjeim során még nagyobb hangsúlyt fogok fektetni az UML diagramok részletes elkészítésére.

A rendszer fejlesztése folyamán az objektumorientált programozáshoz szükséges felfogásmódot úgy gondolom, kellően elsajátítottam, valamint a JAVA programozási nyelv rejtelmeiben is sikerült elmélyülnöm. Számtalan új programozási eszközt ismertem meg, valamint az általam használt integrált fejlesztői környezet (NetBeans) hasznos funkcióinak alkalmazásában is fontos tapasztalatokkal gazdagodtam.

A kitűzött célokat sikerült elérnem, az elkészített alkalmazás véleményem szerint alkalmazhatósági- és felhasználhatósági szempontból is eléri az általam felállított mércét. Az alkalmazás készítése során felhalmozott tudás birtokában azonban úgy gondolom, hogy egy, hasonló volumenű alkalmazásfejlesztési projektet az elkövetkezendőkben sokkal kevesebb idő és munka befektetésével leszek képes elkészíteni, valamint még jobban alkalmazható szoftver tervezésére, és kialakítására leszek képes.

Köszönetnyilvánítás

Szakdolgozatom zárásaként szeretném köszönetemet kifejezni mindazoknak, akik segítséget nyújtottak szakdolgozatom elkészítésében, támogatták munkámat.

Különös köszönettel tartozom témavezetőmnek Dr. Husi Gézának, akinek támogatása, szakmai útmutatása és tanácsai meghatározóak voltak diplomamunkám elkészítésében. Külső konzulensemnek Will Csabának, és a Hajdúsági Hulladékgazdálkodási Kft. munkatársainak, akik rendelkezésemre álltak, bármilyen kérdéssel is fordultam hozzájuk. Készséges segítségük nélkül nem születhetett volna meg ez a szakdolgozat. Az egyetemi tanulmányaim során szerzett elméleti ismereteim kellő alapot nyújtottak ahhoz, hogy meg tudjam oldani az elém tornyosuló problémákat. Ezért szeretnék további köszönetet mondani egyetemi tanáraimnak is. Külön köszönet illeti meg szüleimet, akik időt és fáradtságot nem sajnálva, folyamatosan támogattak szakdolgozatom elkészítésének időszakában.

Irodalomjegyzék

Nyomtatott források:

[1] **Ian Sommerville:**
Szoftverrendszerek fejlesztése
Panem Könyvkiadó, Budapest, 2002.

[2] **Angster Erzsébet:**
Az objektumorientált tervezés és programozás alapjai
Angster Erzsébet, Budapest, 1997.

[3] **Vég Csaba:**
Instant Java/Java EE/Soa I.-II.
Logos 2000 Bt., Debrecen, 2007.

[4] **Angster Erzsébet:**
Objektumorientált tervezés és programozás 1.
4KÖR Bt., Budapest, 2001.

[5] **Angster Erzsébet:**
Objektumorientált tervezés és programozás 2.
4KÖR Bt., Budapest, 2002.

Internetes források:

[6] **Miről szól a Hajdú-Bihar Megyei Hulladékgazdálkodási Program?**
http://www.ujszentmargita.hu/menubal/ispa/kiadvany/programlepo.pdf
Megnyitás: 2010-04-02

[7] **Hajdú-Bihar megye turisztikai térképe**
http://www.hajdubihar.eu/hbm_map.jpg
Megnyitás: 2010-04-01

[8] **AKSD Kft. cégismertetője**
http://www.aksd.hu/
Megnyitás: 2010-04-03

[9] **HHG Kft. cégismertetője**
http://www.hhgkft.hu/h0100.php
Megnyitás: 2010-04-03

[10] **Bihari Hulladékgazdálkodási Kft. cégismertetője**
http://www.biharikft.hu/index.html
Megnyitás: 2010-04-03

[11] **Hajdúsági Regionális Hulladékkezelő telep képe**
http://www.bio-genezis.hu/index.php?inc=hulladek
Megnyitás: 2010-04-01

[12] Wikipedia: Unified Modelling Language
http://hu.wikipedia.org/wiki/UML
Megnyitás: 2010-04-10

[13] Introduction to UML–Unified Modelling Language UML–SmartDraw Tutorials
http://www.smartdraw.com/resources/tutorials/Introduction-to-UML
Megnyitás: 2010-04-10

[14] Overview(Java Platfrom SE 6)
http://java.sun.com/javase/6/docs/api/
Megnyitás: 2010-04-14

[15] java.lang
http://doc.java.sun.com/DocWeb/api/java.lang?lang=hu&mode=Source
Megnyitás: 2010-04-13

Ábrajegyzék

Függelékek jegyzéke