



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

KOMPUTERALGEBRA TANSZÉK

# Gyártástervező alkalmazás

*Szerző:*

Harada Richárd Ryuichi

programtervező informatikus BSc

*Témavezető:*

Fancsali Szabolcs Levente

egyetemi adjunktus

*Külső konzulens:*

Tóth István

műszaki menedzser

*Budapest, 2022*

# EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

## SZAKDOLGOZAT TÉMABEJELENTŐ

### Hallgató adatai:

Név: Harada Richárd Ryuichi

Neptun kód: PWFJYZ

### Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Külső témavezetővel rendelkezem

*Külső témavezető neve: Tóth István*

*munkahelyének neve: AlpsAlpine Európai Elektronikai Ipari Kft.*

*munkahelyének címe: 2051 Biatorbágy, Budai út 1.*

*beosztás és iskolai végzettsége: MES Specialista, Műszaki menedzser*

*e-mail címe: is-toth@alpine.hu*

*Belső konzulens neve: Fancsali Szabolcs Levente*

*munkahelyének neve, tanszéke: ELTE-IK, Komputeralgebra Tanszék*

*munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.*

*beosztás és iskolai végzettsége: egyetemi adjunktus*

**A szakdolgozat címe: Gyártástervező alkalmazás**

### A szakdolgozat témája:

*(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását )*

A gyártással foglalkozó vállalatok meghatározó területe a gyártástervezés. Azoknál a cégeknél, melyek több, tulajdonságaiban eltérő terméket állítanak elő, alapvető problémaként jelentkezik a leghatékonyabb gyártási sorrend beállítása. A nem megfelelő sorrendből fakadó idővesztés egyrészt hatással van a működési költségekre és a munkaerőköltségeire, emellett csökkenti a vállalkozás termelékenységét, hatékonyságát. Ezen tényezők hatása egyenes arányban növekszik a cég méretével, így egy olyan nemzetközi cégnek, mint az AlpsAlpine, stratégiai fontosságú olyan gyártástervező szoftver alkalmazása, mely minimalizálja a költségeket és maximalizálja a termelékenységet. Az alkalmazással szembeni elvárás, hogy a különböző megrendeléseket olyan sorrendbe helyezze, hogy azzal minimumra csökkentsen a modellváltásokat, illetve az ezekhez kapcsolódó átállási időket. A folyamat nehézségét az adja, hogy a beérkező rendelésekkel faktoriálisan növekszik a lehetséges kombinációk száma. A leghatékonyabb sorrend szoftverrel történő megtervezése szignifikánsan időtakarékosabb – egy erre specializált algoritmussal – a humán tervezésnél, ami azt jelenti összességében, hogy nem csak gyártási, de tervezési oldalon is időt és ezzel együtt költséget takarítunk meg, növelve a vállalat hatékonyságát. Tehát programom célja a legpraktikusabb sorrend kiszámítása és demonstrálása, amely adatmódosítás esetén alkalmazkodik és opcionálisan újra rendezi az adatokat.

# Tartalomjegyzék

|                                     |           |
|-------------------------------------|-----------|
| <b>1. Bevezetés</b>                 | <b>3</b>  |
| <b>2. Felhasználói dokumentáció</b> | <b>4</b>  |
| 2.1. Megoldott probléma             | 4         |
| 2.2. Felhasznált módszerek          | 4         |
| 2.3. A program használata           | 5         |
| 2.3.1. Fejléc                       | 6         |
| 2.3.2. Táblázat                     | 7         |
| 2.3.3. Gantt-diagram                | 9         |
| 2.4. Ábrák                          | 10        |
| <b>3. Fejlesztői dokumentáció</b>   | <b>11</b> |
| 3.1. Adatbázis                      | 12        |
| 3.1.1. Kategóriák                   | 12        |
| 3.1.2. Rendelések                   | 13        |
| 3.2. Adatbázis kezelő               | 14        |
| 3.3. Vezérlő                        | 17        |
| 3.3.1. UML osztálydiagram           | 17        |
| 3.3.2. Mellékosztályok              | 18        |
| 3.3.3. Factory osztály              | 20        |
| 3.4. Grafikus felhasználói felület  | 25        |
| 3.4.1 UML osztálydiagram            | 25        |
| 3.4.2. FactoryGUI osztály           | 26        |
| 3.4.3. TableGUI osztály             | 28        |
| 3.4.4. GanttChartGUI osztály        | 29        |
| 3.5. Teszt jegyzőkönyv              | 30        |
| <b>4. Források</b>                  | <b>33</b> |
| <b>5. Irodalomjegyzék</b>           | <b>34</b> |
| <b>6. Ábrajegyzék</b>               | <b>35</b> |
| <b>7. Köszönetnyilvánítás</b>       | <b>36</b> |

# 1. fejezet

## Bevezetés

A gyártással foglalkozó vállalatok meghatározó területe a gyártástervezés. Azoknál a cégeknél, melyek több, tulajdonságaiban eltérő terméket állítanak elő, alapvető problémaként jelentkezik a leghatékonyabb gyártási sorrend beállítása. A nem megfelelő sorrendből fakadó idővesztés egyrészt hatással van a működési költségekre és a munkaerőköltségeire, emellett csökkenti a vállalkozás termelékenységét, hatékonyságát. Ezen tényezők hatása egyenes arányban növekszik a cég méretével, így egy olyan nemzetközi cégnek, mint az AlpsAlpine, stratégiai fontosságú olyan gyártástervező szoftver alkalmazása, mely minimalizálja a költségeket és maximalizálja a termelékenységet.

Az alkalmazással szembeni elvárás, hogy a különböző megrendeléseket olyan sorrendbe helyezze, hogy azzal minimumra csökkentse a modellváltásokat, illetve az ezekhez kapcsolódó átállási időket. A folyamat nehézségét az adja, hogy a beérkező rendelésekkel faktoriálisan növekszik a lehetséges kombinációk száma. A leghatékonyabb sorrend szoftverrel történő megtervezése szignifikánsan időtakarékosabb – egy erre specializált algoritmussal – a humán tervezésnél, ami azt jelenti összességében, hogy nem csak gyártási, de tervezési oldalon is időt és ezzel együtt költséget takarítunk meg, növelve a vállalat hatékonyságát. Tehát programom célja a legpraktikusabb sorrend kiszámítása és demonstrálása, amely adatmódosítás esetén alkalmazkodik és opcionálisan újra rendezi az adatokat.

## 2. fejezet

# Felhasználói dokumentáció


### 2.1. Megoldott probléma

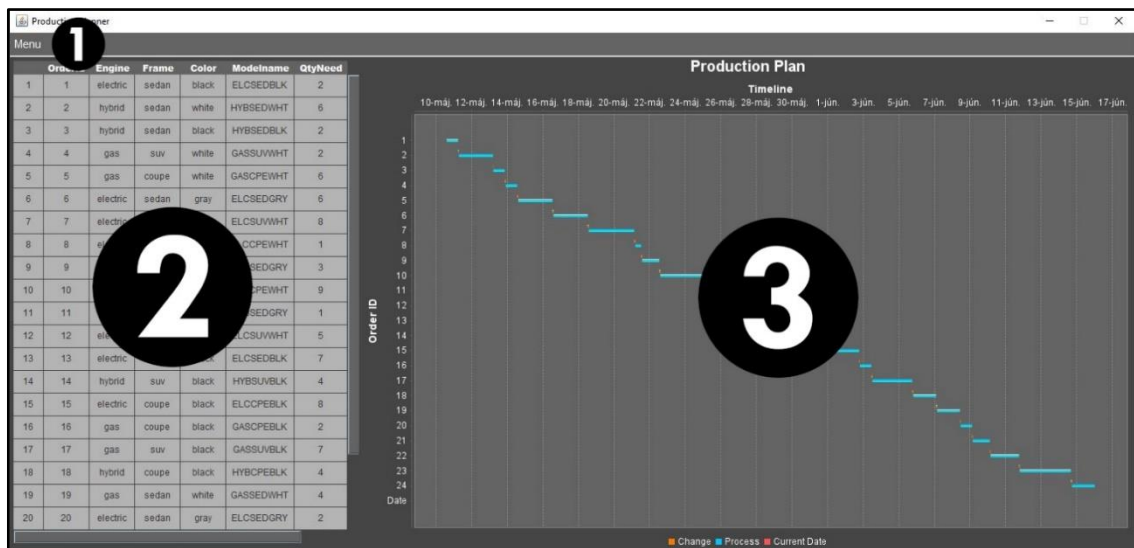
Párosával összehasonlítani az egyes rendeléseket, és ezáltal megtervezni egy hatékony gyártási sorrendet rendkívül összetett és időigényes feladat. Egy  $n$  elemszámú lista esetén a lehetséges kombinációk száma  $n!$  ( $n$  faktoriális), így csupán 10 beérkezett rendelésre is már 3.628.800 különböző terv készülhet. A gyártástervező alkalmazás egy kattintással elkészíti számunkra a legoptimálisabb tervet, amely opcionálisan a felhasználó által is módosítható.

### 2.2. Felhasznált módszerek

Az adatbázisban tárolt rendeléseket az algoritmusok hasonlóság és prioritás szerint rendezik. Minden megrendelt autóhoz tartozik egy motor, egy váz, és annak színe, melyek kulcsfontosságú specifikációk. Ha két rendelés specifikációja mindhárom szempont szerint megegyezik, azokat a program biztosan egymás mellé helyezi a listán, ezzel nullára csökkentve a modellváltás idejét. Amennyiben ez nem lehetséges, kiválaszt egy olyan rendelést, amely legjobb esetben két, legrosszabb esetben egy tulajdonságban sem hasonlít az előző autóhoz. Mivel a gyártási soron a váz cseréje igényli a legtöbb időt, majd azt követi a motor, végül a festék, ezért a kiválasztó algoritmus eme fontossági sorrendet is figyelembe veszi. A program annyi különböző szimulációt futtat le ahány rendelés érkezik, majd ezek közül a leghatékonyabb kombinációt táblázatban és diagramban ábrázolja.

### 2.3. A program használata

 A program gépigénye jelentősen alacsony, így szinte irreleváns hogy milyen hardverekkel rendelkező eszközön futtatjuk, használatához viszont [2] nélkülözhetetlen a Java futásidejű környezet [1] (Java Runtime Environment) telepítése, amely kompatibilis a Windows (7, 8, 8.1, 10), Mac OS X, Linux, és Solaris operációs rendszerekkel. További információkért látogasson el a következő oldalakra: szükséges gép és rendszer feltételekhez <https://java.com/en/download/help/sysreq.html>, letöltéshez és telepítéshez <https://www.java.com/en/download/manual.jsp>. Indításához kattintson a Production Planner parancsikontra, amely a dokumentációval azonos mappában található. Kattintás után várjon amíg megjelenik az alábbi ablak.



1.1. ábra: A gyártástervező program kezdőoldala

A program felhasználói felülete három alapvető elemből áll, ábra szerint számokkal megjelölve:

- ❶ Fejléc
- ❷ Táblázat
- ❸ Gantt-diagram

### 2.3.1. Fejléc

Itt található a Menu (menü) gomb, melyre kattintva kapunk egy legördülő listát. Számos funkcióhoz biztosít hozzáférést:

- **Smart Sort (okos rendezés):** A program legmeghatározóbb algoritmus. A meglévő rendeléseket effektív sorrendbe helyezi, ezzel minimálisra csökkentve a soron a modellcseréket és az abból következő csereidőt. A Smart Sort alkalmazásakor a táblázat és a diagram egyaránt frissül. A gomb csak akkor működik ha a lista még nincs rendezve, ellenkező esetben megjelenít egy hibajelző ablakot.
- **Original List (eredeti lista):** Visszarendezi a listaelemeket a Smart Sort előtti, eredeti állapotba. A gomb csak akkor működik ha történt okos rendezés, ellenkező esetben megjelenít egy hibajelző ablakot.
- **Filter/Reset (szűrés/visszaállítás):** Lehetőség van az adatokat szűrni, amennyiben csak bizonyos tulajdonsággal vagy tulajdonságokkal bíró rendeléseket szeretnénk megjeleníteni. Kattintásra felugrik egy ablak, ahol opcionálisan bepipálható hogy a szűrő milyen specifikációval rendelkező autókra szűkítse a kört, továbbá a szövegmezőkben szintén fakultatívan megadható egy intervallum, hogy mennyi legyen a minimum és maximum rendelési mennyiség (QtyNeed). Ha a szűrésnél irreleváns egy specifikáció (például a motor típusa), nem kötelező mindhárom típust kipipálni. A szövegmezőkbe sem kötelező számot beírni, mert ha üresen hagyjuk, a program nem veszi figyelembe a szűrés során. Az OK gombra kattintva a program szelektálja az adatokat, és azokat megjeleníti a táblázatban és a diagramban is. Ekkor viszont a rendelések módosítására nincs lehetőség, a szűrő kizárólag szűkített adat megjelenítést szolgál. A szűrő törölhető a Filter/Reset gomb újbóli megnyomásával. Amennyiben a program nem talált a szűrés feltételeinek eleget tevő rendeléseket, megjelenít egy hibajelző ablakot.
- **Update (frissítés):** Frissíti a gyártástervező megnyitása óta eltelt időt (a diagramon a vörös Date csík növekszik az eltelt idő függvényében).
- **Exit (kilépés):** Megszakítja a kapcsolatot az adatbázissal és bezárja a programot.

### 2.3.2. Táblázat

A táblázatban megtalálható az összes beérkezett rendelés, melyeket alapesetben az azonosítójuk (**OrderID**) alapján sorrendbe helyez. Alul, illetve a jobb oldalon található sötétszürke csúszkák segítségével van lehetőség navigálni, de a függőleges csúszka helyett egérgörgőt is lehet használni. Összesen nyolc fejlécre osztva jeleníti meg az adatokat.

| (üres fejléc) | A táblázat sorait számozza.   |
|---------------|---|
| OrderID       | A rendelésekhez tartozó azonosító. Minden rendeléshez egyedi szám tartozik.                                 |
| Engine        | Adott rendeléshez tartozó motortípus, ami lehet benzines (gas), dízel (diesel), vagy elektromos (electric). |
| Frame         | Adott rendeléshez tartozó váztypus, ami lehet sedan, suv, vagy coupe.                                       |
| Color         | Adott rendeléshez tartozó szín, ami lehet fehér (white), szürke (gray), vagy fekete (black).                |
| Modelname     | A megrendelt autó modellneve.   |
| QtyNeed       | Megrendelt mennyiség, amely 1-nél nem lehet kevesebb.   |
| QtyDone       | Legyártott mennyiség.   |



Ha a táblázatra jobb egérgombbal klikkelünk, kapunk egy öt opciós legördülő helyi menüt, amely segítségével módosítani lehet a rendeléseket. Az adatbázis folyamatosan frissül ha egy listaelem, vagy egy abban tárolt információ megváltozik.

- **New (új):** Létre lehet hozni egy új rendelést. Ebben az esetben egy felugró ablakban ki lehet választani az autó pontos specifikációit, továbbá a szövegmezőkbe beírható a megrendelt mennyiség, és a már legyártott mennyiség. Az OK gombra kattintva a program rögzíti a rendelést, feltölti az adatbázisba, majd megjeleníti a táblázatban és a Gantt-diagramban. Fontos, hogy az összes beviteli mezőt megfelelően ki kell tölteni, ellenkező esetben nem jön létre új rendelés.
- **Delete (törlés):** Egy meglévő, táblázatban kijelölt rendelés törlését hajtja végre. Kattintásnál egy felugró ablakban kell megerősíteni a törlést, ezzel megelőzve a félre klikkelésből adandó hibaeshetőségeket. Törlés előtt mindig kattintsunk az általunk kiválasztott rendelés sorára! Egy elem eltávolítása után nincs lehetőség visszavonásra.
- **Change QtyNeed (megrendelt mennyiség módosítása):** Ennek segítségével felülírhatjuk az általunk kijelölt sorban lévő rendelés mennyiségét.
- **Change QtyDone (legyártott mennyiség módosítása):** Hasonló az előző menüponthoz, csak a kijelölt rendelésnél a legyártott mennyiség számát lehet felülírni.
- **Move.. (mozgatás..):** Lehetőséget nyújt mozgatni egy kijelölt listaelemet. Ez a funkció nem módosítja az adatbázist, hanem a gyártásterv sorrendjét manipulálja. Ha rajta van az egérmutató a menüpontra, megjelenik egy mellék menü, ahol három opció közül lehet választani:
  - Up (fel): Adott elem felfelé mozgatása.
  - Down (le): Adott elem lefelé mozgatása.
  - Specific location (meghatározott hely): Ez esetben egy felugró ablakban kell beírni a sorszámot ahová szeretnénk átmozgatni az elemet.

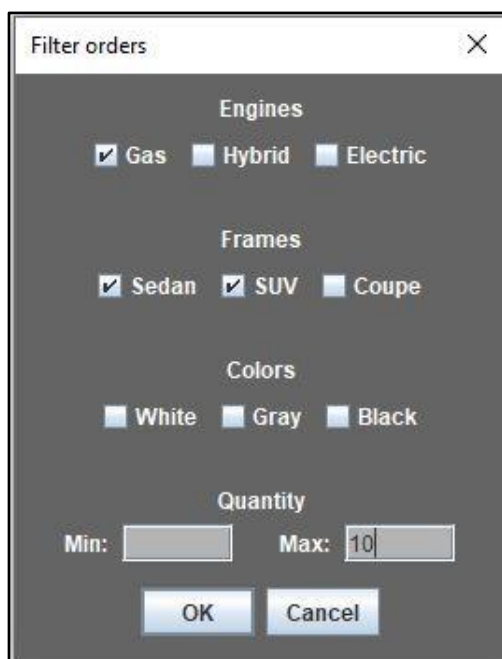
### 2.3.3. Gantt-diagram [3]

Ez a JFree diagram ábrázolja a gyártási folyamatot idő és rendelési azonosító függvényében. Minden egyes rendeléshez tartozik egy világoskék színjelzésű csík, amely hossza a rendelés mennyiségétől függ, elhelyezkedése pedig azonos magasságban van a hozzátartozó azonosítóval. A narancssárga színjelzésű csíkok a cseredőket ábrázolják, amik hosszát befolyásolja a cserélendő modellek típusa. A legalsó sorban a program indítása óta eltelt időt reprezentálja, amely egyben jelzi hogy ideális esetben hol tart a gyártási folyamat. Ha a diagramon bal egérgombot lenyomva tartva jobbra húzzuk az egeret, ki lehet jelölni a folyamatábrán belül egy intervallumot, melyre az egérgomb felengedését követően a diagram ráközelít. Ez a funkció többször is megismételhető, ám ha ki szeretnénk lépni a közelítésből, kattintsunk bal egérgommbal a diagramra, majd azt folyamatos nyomva tartás közben húzzuk balra az egeret. Jobb klikk esetén megjelenik egy helyi menü a következő választási lehetőségekkel:

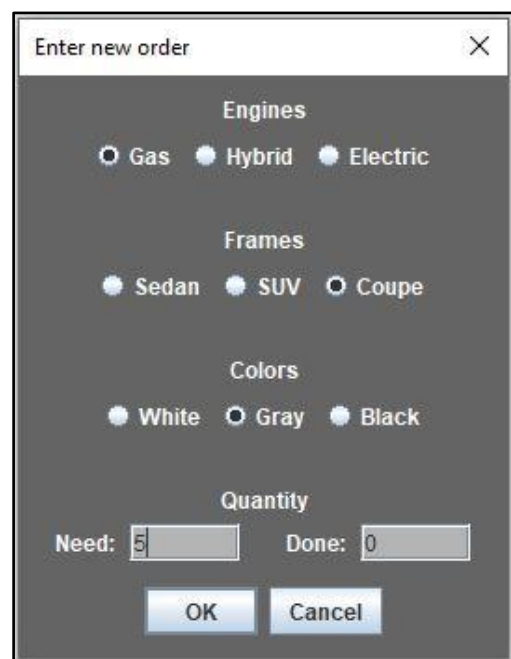
- **Properties... (tulajdonságok):** Erre kattintva megjelenik egy felugró ablak, ahol személyre lehet szabni a diagram megjelenését:
  - Title (cím): Át lehet írni a diagram címét, illetve be lehet állítani annak színét és betűtípusát.
  - Plot (grafikon): A diagram háttérszíneit, a címkék betűtípusát, színét, és méretét lehet módosítani.
  - Other (egyéb): Beállítható a diagram külső rétegének a színe.
- **Copy (másolás):** Vágólapra másolja a diagramot.
- **Save as (mentés másként):** A diagramot PNG formátumban kiexportálja az általunk kiválasztott könyvtárba. A kép lehet a diagrammal azonos méretarányú, illetve 1920x1080-as felbontású.
- **Print... (nyomtatás...):** A Gantt-diagram kinyomtatásáért felelős. Részletes beállításokhoz felugrik egy ablak.

- **Zoom In (nagyítás):** Ráközelít a diagramra. Mivel a diagram Gantt típusú, ezért csak az idő síkjában (Range Axis) lehet nagyítani.
- **Zoom Out (kicsinyítés):** Lekicsinyíti a diagramot az idő síkjában.
- **Auto Range (automatikus távolság):** A diagramot ideálisra méretezi, hogy a legelső és legutolsó gyártási folyamat egyaránt elérjen rajta.

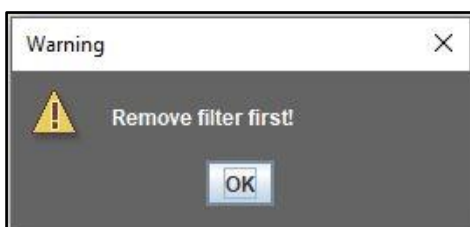
## 2.4. Ábrák



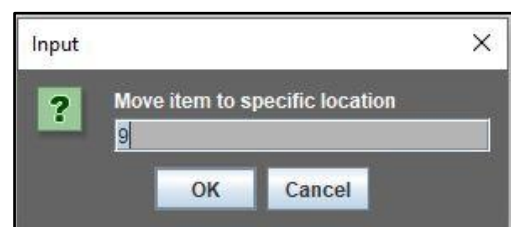
1.2. ábra: Rendelések szűrése



1.3. ábra: Új rendelés létrehozása



1.4. ábra: Figyelmeztető ablak, ha szűrt adatokat próbálnak módosítani



1.5. ábra: Rendelés mozgatása specifikus sorba

### 3. fejezet

## Fejlesztői dokumentáció

A gyártástervező programhoz tartozó forráskódok, adatbázis, és egyéb kiegészítő fájlok a `Production` mappán belül találhatóak. A kódok átfogó tanulmányozásához, illetve fejlesztéséhez szükséges letölteni a Java Development Kit 13. [4] verzióját, továbbá az Apache NetBeans [5] integrált fejlesztői környezetet, amelyben a Java Ant projekt készült.

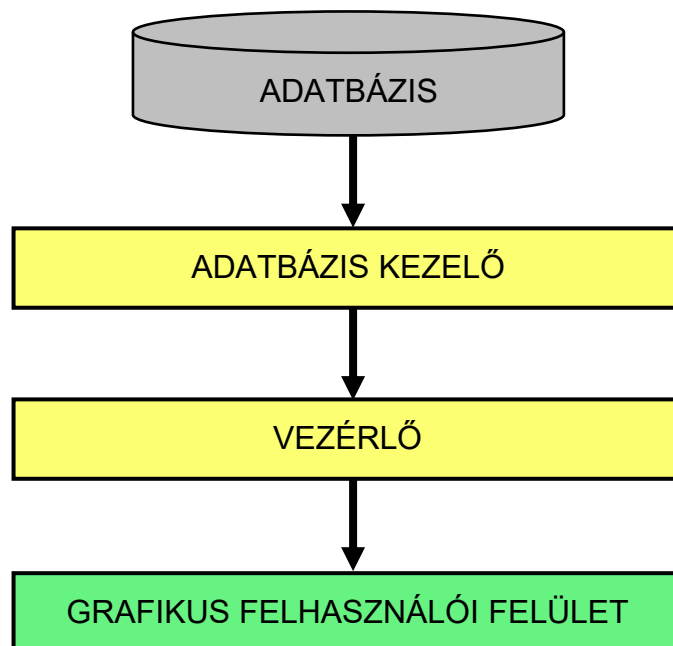
*Java Development Kit (JDK) 13* [4]:

<https://www.oracle.com/java/technologies/javase/jdk13-archive-downloads.html>

*Apache NetBeans IDE* [5]:

<https://netbeans.apache.org/download/index.html>

A program struktúrájának négy alappillére van, kezdve backend-ből a frontend irányába a sorrend a következő:



2.1. ábra: A program rétegei

A program által használt külső könyvtárak:

|                    |      |
|--------------------|------|
| <i>java.util</i>   | [9]  |
| <i>javax.swing</i> | [10] |
| <i>java.sql</i>    | [8]  |
| <i>java.awt</i>    | [11] |
| <i>java.time</i>   | [12] |
| <i>java.jfree</i>  | [3]  |

## 3.1. Adatbázis

A programhoz szükséges információk SQLite [6] adatbázisban vannak tárolva. Az adatbázis megtalálható a `Production/dist/database` könyvtárban `database.db` néven, de található a NetBeans projekthez külön egy teszt adatbázis is a `Production/database` mappában, szintén `database.db` néven. A program és az SQLite adatbázis kapcsolatát biztosító Java Database Connectivity driver [7], azaz JDBC meghajtó `sqlite-jdbc-3.20.0.jar` néven (az adatbázishoz hasonlóan) két helyen is megtalálható, az elsődleges a `Production/dist/lib` mappában, míg a projekthez tartozó a `Production/database` mappában. Ha a projektnél lefut a Clean and Build, az IDE törli az elsődleges adatbázist és drivert, így a másodlagos fájlok biztonsági mentésként is funkcionálnak.

Az adatbázis két táblában tárolja az adatokat. Az első az autók kategóriáit tartalmazza **categories** néven, a másik a rendeléseket **orders** néven.

### 3.1.1. Kategóriák

A tábla a megrendelhető autók összes lehetséges kombinációját tartalmazza, így véglegesen 27 sorból tevődik össze. A program semmilyen formában nem módosítja, kizárólag információk beolvasására használja. A tábla öt oszlopra osztva tárolja az adatokat:

| categories |  |
|------------|--|
| CategoryID | A kategóriákhoz tartozó azonosító. Ehhez az oszlophoz tartozó sorszámok az elsődleges kulcsok.         |
| Engine     | Kategóriához tartozó motortípus, ami lehet benzines (gas), dízel (diesel), vagy elektromos (electric). |
| Frame      | Kategóriához tartozó váz típus, ami lehet sedan, suv, vagy coupe.                                      |
| Color      | Kategóriához tartozó szín, ami lehet fehér (white), szürke (gray), vagy fekete (black).                |
| Modelname  | Az kategóriába tartozó autó modellneve.  |

### 3.1.2. Rendelések

Minden beérkezett rendelés ebben a táblában található. A kategóriákat tartalmazó táblával ellentétben ezt a táblát módosítja a gyártástervező: feltölthet, törölhet, és átírhat rendeléseket a felhasználó utasítására. A tábla négy oszlopban tárolja az adatokat:

| orders     |   |
|------------|---|
| OrderID    | A rendelésekhez tartozó azonosító. Ehhez az oszlophoz tartozó sorszámok az elsődleges kulcsok.                            |
| CategoryID | A rendeléshez tartozó autó(k) kategóriájának azonosítója. Az orders tábla ezzel csatlakozik a categories tábla adataihoz. |
| QtyNeed    | Megrendelt mennyiség.   |
| QtyDone    | Legyártott mennyiség.   |

## 3.2. Adatbázis kezelő

Ez a Java osztály az adatbázissal közvetlen kapcsolatban áll és módosítja benne az információkat a `java.sql` [8] könyvtár használatával. A fájl megtalálható a `Production/src/production` mappában `Database.java` néven. Megnyitható önmagában is, de ajánlottabb belépni NetBeans [5] környezetbe és azon belül megnyitni a projekt fájlt. Ennek az osztálynak hét metódusa van:

- **connection():** Ez az eljárás kapcsolatot létesít az adatbázissal. Ahogy az ábra is mutatja, a programba bele van égetve az adatbázis könyvtárának útvonala, ezért fontos hogy a db fájlt ne mozgassuk át! A csatlakozás eredményét, azaz hogy sikerült-e vagy nem, a konzolra kiírja. A függvény kezeli az esetleges kivételeket (Exception).

```
public static Connection connection() {  
    try {  
        Class.forName("org.sqlite.JDBC");  
        con = DriverManager.getConnection("jdbc:sqlite:database/database.db");  
        System.out.println("connection succesful");  
        return con;  
    }  
    catch (Exception e) {  
        System.out.println("connection failed");  
        return null;  
    }  
}
```

2.2. ábra: Adatbázishoz csatlakozó eljárás

- **getOrdersFromDB(orders):** A függvény paraméterként kap egy `Order` típusú listát, amit feltölt az adatbázisban tárolt rendelésekkel. Ehhez szükséges volt létrehozni egy `String`-et amely tartalmazza a lekérdezést SQL nyelven: `SELECT * FROM orders`.

A csillag jelöli hogy a függvény az összes adatot le szeretné kérdezni az orders táblából. Végrehajtáskor egy eredmény kollekcióba (ResultSet rs) kerülnek a lekérdezés értékei, amelyet rögtön beleír a paraméterként kapott lista elemeibe, azaz az Order-ekbe. A while ciklus addig fut amíg az adatolvasás el nem éri az utolsó sort is.

```
public static ArrayList<Order> getOrdersFromDB(ArrayList<Order> orders) {  
  
    try {  
        String sql = "SELECT * FROM orders";  
        ps = con.prepareStatement(sql);  
        rs = ps.executeQuery();  
  
        while(rs.next()) {  
            orders.add(new Order(  
                (long) rs.getInt("OrderID"),  
                (long) rs.getInt("CategoryID"),  
                rs.getInt("QtyNeed"),  
                rs.getInt("QtyDone")  
            ));  
        }  
    }  
    catch(Exception e) {  
        System.out.println(e);  
    }  
  
    return orders;  
}
```

2.3. ábra: Rendelések beolvasása adatbázisból

- **getCategoriesFromDB(categories):** Hasonlóan az előző függvényhez ez is adatokat kérdez le, csak a categories táblából: `SELECT * FROM categories`. Az adatbázisból beolvasott adatokkal feltölti a paraméterként kapott, Category típusú listát, ha nem történik olvasás közben kivétel.
- **addElementToDB(order):** A felhasználó által megadott adatokat tartalmazó új rendelést regisztrál az orders táblába: `INSERT INTO orders(OrderID, CategoryID, QtyNeed, QtyDone) VALUES(?, ?, ?, ?)`. Az állításban lévő kérdőjelek adott esetben a `setInt()` függvény és a paraméterként kapott új order értékeinek segítségével lettek konkretizálva. Az előző függvényekkel ellentétben



itt nincs szükség `ResultSet`-re, ugyanis csak frissíti az adatbázist, nem kérdez le adatokat.

```
public static void addElementToDB(Order order) {  
  
    try {  
        String sql = "INSERT INTO orders(OrderID, CategoryID, QtyNeed, QtyDone) VALUES(?, ?, ?, ?)";  
        ps = con.prepareStatement(sql);  
        ps.setInt(1, (int)order.getOrderID());  
        ps.setInt(2, (int)order.getCategoryID());  
        ps.setInt(3, order.getQtyNeed());  
        ps.setInt(4, order.getQtyDone());  
        ps.executeUpdate();  
    }  
    catch(Exception e) {  
        System.out.println(e);  
    }  
}
```

2.4. ábra: Új rendelés feltöltése az adatbázisba

- **deleteElementFromDB(order):** A paraméterben lévő order alapján rendelést töröl az orders táblából: `DELETE FROM orders WHERE OrderID = ?`. Az állításban a `WHERE` kulcsszó határozza meg, hogy melyik sorban lévő adatokat kell kitörölni, így a kérdőjel helyére beilleszti az eltávolítandó rendelés azonosítóját.
- **updateElementInDB(order):** Felülírja az adatbázisban annak a rendelésnek a megrendelt mennyiségét, illetve legyártott mennyiségét, amelynek paraméterét megkapta: `UPDATE orders SET QtyNeed = ?, QtyDone = ? WHERE OrderID = ?`.
- **closeConnection():** Megszakítja a kapcsolatot az adatbázissal. A program bezárásakor fut le.

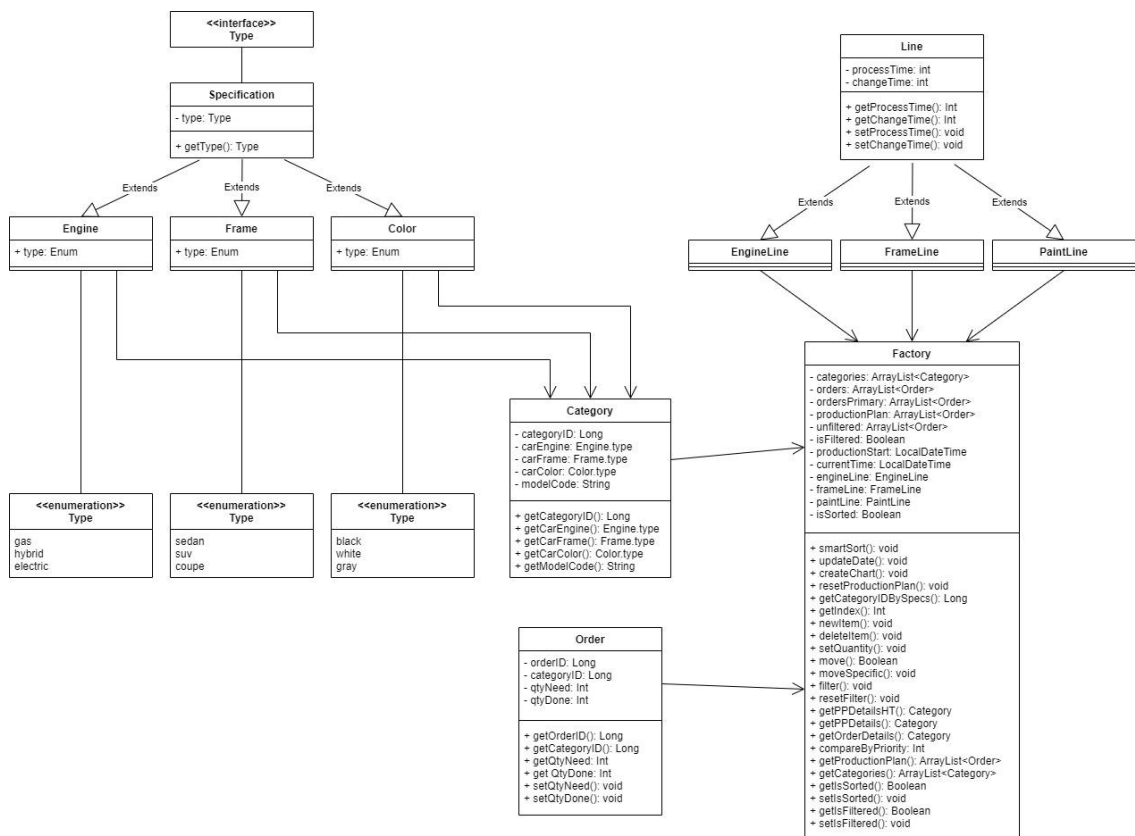
```
public static void closeConnection() {  
  
    try {  
        rs.close();  
        ps.close();  
        con.close();  
    }  
    catch(Exception e) {  
        System.out.println(e);  
    }  
}
```

2.5. ábra: Kapcsolat megszakító eljárás

### 3.3. Vezérlő

A vezérlő kategóriába tartoznak azok az osztályok, amelyek a háttérben futó adatfeldolgozó algoritmusokért, és számításokért felelnek, más szóval ez az egység a program legfontosabb, központi alkotóeleme. Egyben ez a gyártástervező utolsó rétege, amely még rejtve van a felhasználó előtt. A vezérlő pontos szerkezetét az alábbi ábra demonstrálja:

#### 3.3.1. UML osztálydiagram



3.1. ábra: Vezérlő egység UML osztálydiagramja

### 3.3.2. Mellékosztályok

A `Production/src/production` mappán belül ebbe a kategóriába sorolhatók a következő Java osztályok:

- |                                   |                                       |
|-----------------------------------|---------------------------------------|
| ✓ <code>Order.java</code>         | ✓ <code>Line.java</code>              |
| ✓ <code>Category.java</code>      | ✓ <code>EngineLine.java</code>        |
| ✓ <code>Specification.java</code> | ✓ <code>FrameLine.java</code>         |
| ✓ <code>Engine.java</code>        | ✓ <code>PaintLine.java</code>         |
| ✓ <code>Frame.java</code>         | ✓ <code>Production.java (main)</code> |
| ✓ <code>Color.java</code>         |                                       |

- **Order:** Rendelés pontos adatait tároló osztály, amely tartalmazza privát (`private`) változóiban a rendelés azonosítóját (`orderId`), a hozzá tartozó autó kategória azonosítóját (`categoryId`), a megrendelt mennyiséget (`QtyNeed`), és a legyártott mennyiséget (`QtyDone`). A változók az osztály példányosításakor lesznek inicializálva, a konstruktor által. Ehhez tartoznak publikus (`public`) getter, és bizonyos változókhoz setter metódusok is, amik segítségével a külső osztályok képesek hozzáférni, illetve módosítani az `Order` osztály adatait (például `getOrderId()`).
- **Category:** Egy kategória azonosítóját (`categoryId`), modellnevét (`modelCode`), és a pontos specifikációit tárolja privátban. Az autó(k) motor, váz, és szín változóinak típusát az `Engine`, `Frame`, és `Color` osztályokból alkotja. Az `Order` osztályhoz hasonlóan a `Category`-nak is csak getter és setter metódusai vannak.
- **Specification:** A specifikáció ősoosztály, ami tartalmaz egy védett (`protected`) `Type` nevű interface-t, egy privát `Type` típusú változót, és egy publikus `getType()` nevű getter metódust. Az interface alaptól nem tartalmaz változókat

és függvényeket, de a Specification-ból származó gyerekosztályok megöröklik, és saját típust implementálnak.

```
public class Specification {  
    protected interface Type { }  
    private Type type;  
    public Specification(Type type) {  
        this.type = type;  
    }  
    public Type getType() {  
        return type;  
    }  
}
```

3.2. ábra: Specification őszosztály

- **Engine:** A Specification őszosztály első gyereke. A Type interface-ből Enum típusú változót implementál, és feltölti gas, hybrid, és electric motorokkal. Emellett megörökli az ehhez tartozó getter metódust is.

```
public class Engine extends Specification {  
    public Engine(Type type) {  
        super(type);  
    }  
    public enum type implements Type {  
        gas,  
        hybrid,  
        electric  
    }  
    @Override  
    public Type getType() {  
        return super.getType();  
    }  
}
```

3.3. ábra: Őszosztályból származtatott Engine gyerekosztály

- **Frame:** A Specification őszosztály második gyereke. Az Engine-hez hasonlóan megörökli a getter metódust, a Type interface-ből pedig Enum típusú változót implementál, és feltölti sedan, coupe, és suv vázakkal.

- **Color:** A Specification ősosztály harmadik gyereke. Ugyanaz a szerepe mint a többi gyerekosztálynak, csak adott esetben az Enum opciói a black, white, és gray színek.
- **Line:** A gyártási sorok ősosztálya. Két privát változót tartalmaz: a gyártási folyamat idejét (processTime), és a modellcseré idejét (changeTime), továbbá rendelkezésre állnak a getter és a setter metódusok.
- **EngineLine/FrameLine/PaintLine:** A Line ősosztály gyerekei, amik megöröklik a metódusokat, és a változókat amelyekre mindegyik osztály más értéket inicializál.
- **Production:** Itt található a main() függvény, amely elindítja a programot a FactoryGUI osztály példányosításával.

### 3.3.3. Factory osztály

A Factory a vezérlőnek legdominánsabb osztálya, ami a háttérből mozgatja a szálakat. Az imént felsorolt mellékosztályokból példányosít objektumokat és listákat, majd ezek segítségével funkcionál. Kapcsolatban áll az adatbázis kezelővel, így az azáltal lekérdezett adatokat is ez az osztály tárolja, majd tovább küldi a grafikus felhasználói felületnek. Számos metódussal rendelkezik, ezek közül kiemelném a legjelentősebbeket:

- **smartSort():** Ez a program kulcsfontosságú funkciója. A meglévő rendeléseket hatékony sorrendbe helyezi, ezzel minimálisra csökkentve a soron a modellcseréket és az abból következő csereidőt.

Három listát kezel: a rendeléseket tartalmazó, elsődleges ordersPrimary listát, ennek másolatát (orders), illetve a gyártási tervet (productionPlan) ami alaphelyzetben még rendezetlen. Kezdetben a gyártási tervet üresre állítja, esetlegesen az orders listát is alaphelyzetbe állítja az ordersPrimary lista alapján. Ezt követően kiválasztja az orders listában a legelső rendeltetést, beilleszti a gyártási tervbe, majd a compareByPriority() függvénnyel végig halad a maradék elemeken, és megkeresi a hozzá leginkább hasonlót.

Ha ez megtörtént, hozzáfűződik a gyártási tervhez, és addig ismétlődik a procedúra, amíg az összes rendelés hozzá nem lett csatolva a gyártási tervhez az orders listából. A végén az algoritmus elmenti annak a rendelésnek az indexét amelyikkel elkezdte a folyamatot, továbbá megjegyzi a `compareByPriority()` által visszaküldött legkritikusabb differencia mértéket (`PlanReturn`) ami minél kisebb annál megfelelőbb. Kitörli a `productionPlan` elemeit, visszaállítja az orders listát, és egy másik elemmel kezdve megismétli a folyamatot.

A rendezést annyi féleképpen szimulálja le ahány rendelést tartalmaz a lista, majd ezek közül kiválasztja azt a sorrendet amelynél a differencia mértéke a legminimálisabb volt. Utolsó lépésben az `isSorted` nevű változót igazra állítja, mivel a gyártási terv már optimális sorrendben van.

```
public void smartSort() {  
  
    if(orders.isEmpty()) {  
        orders = new ArrayList<>(ordersPrimary);  
    }  
  
    productionPlan.clear();  
  
    int PlanReturn, LowestValue = 8, BestStarterIndex = 0;  
  
    for(int i = 0; i < ordersPrimary.size(); i++) {  
        productionPlan.add(orders.get(i));  
        orders.remove(i);  
        PlanReturn = compareByPriority();  
        if(PlanReturn < LowestValue) {  
            LowestValue = PlanReturn;  
            BestStarterIndex = i;  
        }  
        productionPlan.clear();  
        orders = new ArrayList<>(ordersPrimary);  
    }  
  
    productionPlan.add(orders.get(BestStarterIndex));  
    orders.remove(BestStarterIndex);  
    compareByPriority();  
  
    isSorted = true;  
}
```

4.1. ábra: A gyártástervező okos rendező algoritmus

- **compareByPriority():** Ez a smartSort() segédfüggvénye. Feladata összehasonlítani a rendeléseket prioritás szerint: legelőször azt vizsgálja meg hogy van-e olyan rendelés az orders listában amelynek minden specifikációja egyezik a gyártási terv (productionPlan) legelső vagy legutolsó elemével. Amennyiben van ilyen, hozzáfűzi azt a gyártási tervhez, eltávolítja az orders listából, majd folytatja a keresést ugyanezzel a feltétellel. Ha nem talál ilyen rendelést, csökkenti a feltételt: egyezzen a motor és a váz, a festés nem releváns. A feltétel egészen addig redukálható, hogy a két rendelésnek egyáltalán ne legyen közös tulajdonsága. Ha minden rendelést beillesztett a gyártási tervbe, elküldi a smartSort()-nak a legkritikusabb differencia mértékét, azaz a legalacsonyabban alkalmazott összehasonlítási feltételt. A prioritási sorrend a következő:

| Feltételek |   |
|------------|---|
| 1.         | A rendelések összes specifikációja egyezzen |
| 2.         | A vázak és a motorok típusai egyezzenek     |
| 3.         | A vázak és a festések típusai egyezzenek    |
| 4.         | A motorok és a festések típusai egyezzenek  |
| 5.         | A vázak típusai egyezzenek                  |
| 6.         | A motorok típusai egyezzenek                |
| 7.         | A festések típusai egyezzenek               |
| 8.         | Nincs egyezés                               |

- **newItem(values):** A paraméterként kapott új rendelést beilleszti az ordersPrimary listába. Először ellenőrzi hogy a lista tartalmaz-e 1-es számú azonosítóval ellátott rendelést, mert ha nem talál akkor ide illeszti. Ellenkező esetben elkezd vizsgálni hogy a rendelések között van-e „rés” azaz például ha egy rendelés azonosítója 5-ös, akkor az azt követő 7-es vagy annál nagyobb számú. Amennyiben talál rést, oda helyezi a rendelést, ha nem talál akkor egyszerűen a lista végéhez csatolja. Amint bekerült a rendelések közé az új elem, meghívja az adatbázis kezelő addElementToDB() eljárását. Az adatokat String típusú tömb formában kapja meg, ennek következtében az új Order létrehozása is itt történik, mindazonáltal annak egyik változója a kategória azonosítója (CategoryID). Az új rendelés az autó(k) specifikációit tartalmazza, a kategória azonosítóját nem, így azt a getCategoryIDBySpecs() függvény segítségével kérdezi le.

```
public void newItem(String[] values) {

    long categoryID = getCategoryIDBySpecs(values);
    Order newOrder;

    //ha nincs 1. számú eleme a listának
    if(ordersPrimary.get(0).getOrderID() > 1) {
        newOrder = new Order(1, categoryID, Integer.parseInt(values[3]), Integer.parseInt(values[4]));
        ordersPrimary.add(0, newOrder);
        Database.addElementToDB(newOrder);
    }

    else {
        boolean inserted = false;

        for(int i = 0; i < ordersPrimary.size() - 1; i++) {
            //ha két listaelem között legalább egy szabad hely van
            if((ordersPrimary.get(i).getOrderID() + 1) != ordersPrimary.get(i + 1).getOrderID()) {
                //létrehoz egy új elemet a megadott pozícióban
                newOrder = new Order(i + 2, categoryID, Integer.parseInt(values[3]), Integer.parseInt(values[4]));
                ordersPrimary.add(i + 1, newOrder);
                Database.addElementToDB(newOrder);
                inserted = true;
                break;
            }
        }
        //ha csak a lista végén van szabad hely
        if(!inserted) {
            newOrder = new Order(ordersPrimary.size() + 1, categoryID, Integer.parseInt(values[3]), Integer.parseInt(values[4]));
            ordersPrimary.add(newOrder);
            Database.addElementToDB(newOrder);
        }
    }
}
```

4.2. ábra: Új rendelést listába helyező algoritmus



- **createChart()** : Kiszámolja a Gantt-diagramhoz [3] tartozó összes szükséges adatot, és továbbítja a GanttChartGUI osztálynak. Végig halad a gyártási terv elemein, kinyeri belőlük a gyártási és csere idő hosszát a Line gyerekosztályaiban tárolt adatok segítségével, majd ezek alapján felépíti a diagram időbeli szerkezetét. Egy rendelés gyártási folyamatának a hossza a megrendelt mennyiségétől is függ. Modellváltás esetén ha két egymást követő rendelés specifikációi megegyeznek, a csereidő következetesen nulla lesz, a többi esetben viszont megnézi melyik az a cserélendő modell amelyik a legtöbb időt igényli.

```

public void createChart() {
    currentTime = LocalDateTime.now();

    LocalDateTime start = productionStart, end = productionStart;
    ganttChart.newChange(productionStart, productionStart, productionPlan.get(0).getOrderID());

    for(int i = 0; i < productionPlan.get(0).getQtyNeed(); i++) {
        end = end.plusMinutes(frameLine.getProcessTime() + engineLine.getProcessTime() + paintLine.getProcessTime());
    }

    ganttChart.newProcess(start, end, productionPlan.get(0).getOrderID());

    int maxChangeTime = 0;

    for(int i = 1; i < productionPlan.size(); i++) {
        maxChangeTime = 0;

        if(productionPlan.get(i).getCategoryID() != productionPlan.get(i - 1).getCategoryID()) {
            start = end;

            if(getPPDetails(i).getCarColor() != getPPDetails(i - 1).getCarColor()) {
                if(paintLine.getChangeTime() > maxChangeTime) {
                    maxChangeTime = paintLine.getChangeTime();
                }
            }

            if(getPPDetails(i).getCarEngine() != getPPDetails(i - 1).getCarEngine()) {
                if(engineLine.getChangeTime() > maxChangeTime) {
                    maxChangeTime = engineLine.getChangeTime();
                }
            }

            if(getPPDetails(i).getCarFrame() != getPPDetails(i - 1).getCarFrame()) {
                if(frameLine.getChangeTime() > maxChangeTime) {
                    maxChangeTime = frameLine.getChangeTime();
                }
            }

            end = end.plusMinutes(maxChangeTime);
            ganttChart.newChange(start, end, productionPlan.get(i).getOrderID());
        }
        else {
            ganttChart.newChange(productionStart, productionStart, productionPlan.get(i).getOrderID());
        }

        start = end;

        //annyiszor adja hozzá a process timet ahány rendeltést kértek
        for(int j = 0; j < productionPlan.get(i).getQtyNeed(); j++) {
            end = end.plusMinutes(frameLine.getProcessTime() + engineLine.getProcessTime() + paintLine.getProcessTime());
        }
        ganttChart.newProcess(start, end, productionPlan.get(i).getOrderID());
    }
    ganttChart.newTimeline(productionStart, currentTime);
}

```

4.3. ábra: Gantt-diagramot felépítő metódus

### 3.4. Grafikus felhasználói felület

Ez a program frontend rétege, amely már látható a felhasználó számára. Ide tartozik az összes osztály amik neveiben megtalálható a GUI, azaz a Graphical User Interface kifejezés, továbbá egy kiegészítő osztály. Közös jellemzőjük hogy a Java Swing [10] eszközkészlet és Abstract Window Toolkit [11] felhasználásával lettek implementálva:

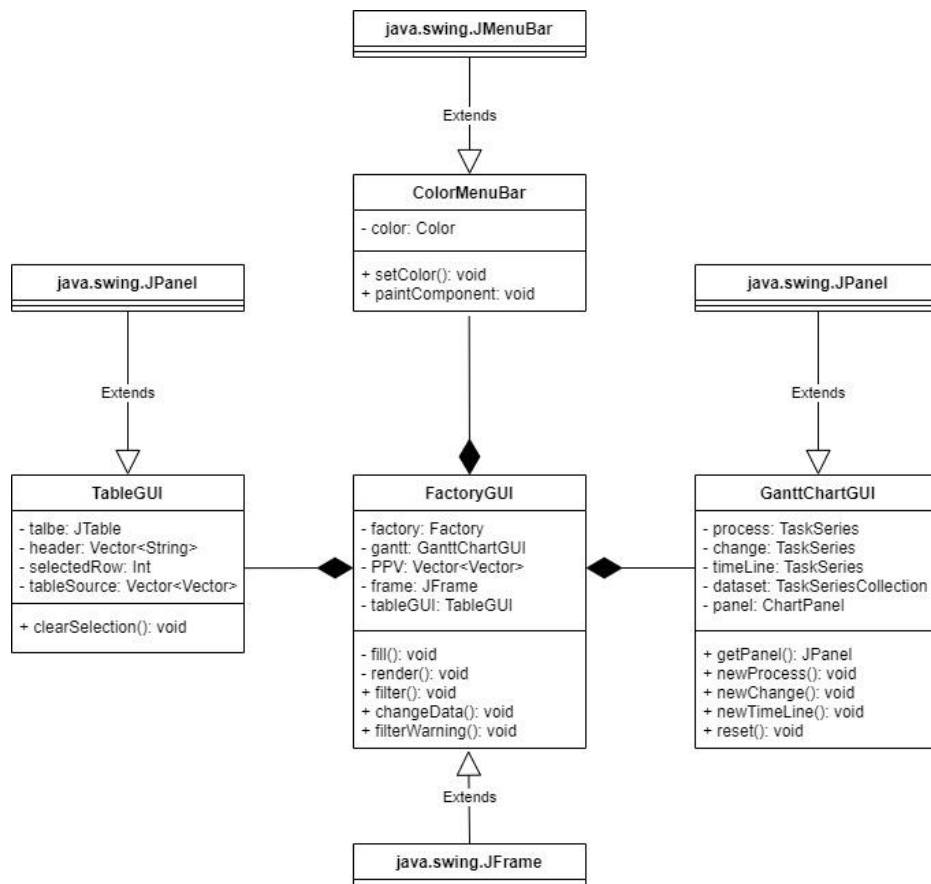
Swing: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

AWT: <https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>

A fájlok megtalálhatók a Production/src/production könyvtárban:

- ✓ FactoryGUI.java
- ✓ TableGUI.java
- ✓ GanttChartGUI.java
- ✓ ColorMenuBar.java

#### 3.4.1. UML osztálydiagram



5.1. ábra: Grafikus felhasználói felület UML osztálydiagramja

### 3.4.2. FactoryGUI osztály

A FactoryGUI osztály a grafikus felhasználói felület ablaka és háttere, ezért az őssztálya is `javax.swing.JFrame` [10]. Ebben az osztályban történik a Factory, illetve a felülethez tartozó többi osztályt példányosítása. Az ablak 1500x720 felbontású és átméretezhetetlen. A színek, a méretek, az igazítások, és betűtípusok mind `UIManager` segítségével lettek beállítva.

A fejléc `ColorMenuBar` típusú, ami csak színben tér el a `JMenuBar`-tól. Létrehozására azért volt szükség mert más opció nem volt a megjelenését módosítani. Ezen a fejlécen helyezkedik el a `JMenu` menü is, melyre kattintva `JMenuItem`-ekkel feltöltött legördülő listát kapunk. Minden egyes menüpont tartalmaz egy `ActionEventListener()`-t melyek azt figyelik hogy az adott menüpontra kattintott-e a felhasználó.

```
JMenuItem menuItem1 = new JMenuItem("Smart Sort");
menu.add(menuItem1);
menuItem1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(!factory.getIsFiltered()) {
            if(!factory.getIsSorted()) {
                gannt.reset();
                factory.smartSort();
                factory.createChart();
                fill();
                render();
            }
            else {
                JOptionPane.showMessageDialog(frame, "The list is already sorted");
            }
        }
        else {
            JOptionPane.showMessageDialog(frame, "Remove filter first!", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }
});
```

5.2. ábra: A SmartSort-ot hívó JMenuItem

A fenti ábrán az okos rendező algoritmust hívó menüelem található. Ha rákattintunk, először leellenőrzi hogy a lista már rendezve van-e, mert ha igen, egy felugró ablakkal közli a felhasználóval. Ha még nincs rendezve, a diagramot visszaállítja, lefuttatja a `smartSort()` eljárást, majd újra alkotja a diagramot és a táblázatot.

- **fill():** Feltölti a PPV (Production Plan Vector) nevű kollekciót a gyártásterv rendelkezéseivel. Erre azért van szükség, mert a **JTable** táblázat nem kompatibilis az **ArrayList**-el.
- **render():** Ha frissülnek az adatok, generál egy új táblázatot és lecseréli az előzőt.
- **filter():** Megjelenít a felhasználó számára egy felugró ablakot, amelyen megadhatja a szűrés feltételeit. A bevitt értékeket egy **String** listában tárolja 't', 'f' karakterekkel, és adott esetben számokkal kiegészítve, majd tovább küldi a **Factory** osztálynak. Ha már van szűrő beállítva és így történik kattintás, törli a szűrést.
- **changeData(mode, row):** A **FactoryGUI** legösszetettebb metódusa. A **TableGUI** osztályból érkező, gyártási tervet szerkesztő utasításokat végzi el. Az első paraméter adja meg hogy milyen típusú adatmódosítás történjen (**mode**), amely 1-től 7-ig terjedő skálán helyezkedik el:

|   |                                      |
|---|--------------------------------------|
| 1 | Új rendelés hozzáadása               |
| 2 | Rendelés törlése                     |
| 3 | QtyNeed átírása                      |
| 4 | QtyDone átírása                      |
| 5 | Rendelés felfelé mozgatása           |
| 6 | Rendelés lefelé mozgatása            |
| 7 | Rendelés mozgatása specifikus helyre |

Az első opció kivételével a metódusnak szükséges paraméterben megadni a módosítandó adat sorszámát (**row**). Miután megtörtént az adatmódosítás a **Factory** osztályban található műveletek hívásával, törlés és új elem esetén rákérdez a felhasználóra, hogy szeretné-e rendezni a listát a frissült listaelemekkel. Átírásnál és mozgatásnál ez az ablak nem jelenik meg, mert lekérdezi a **Factory** osztályból hogy a lista művelet előtt rendezve volt-e vagy nem. Nem megfelelő bevitt adatok esetén **JOptionPane** figyelmeztető ablakokkal interaktál.

### 3.4.3. TableGUI osztály

A FactoryGUI keretrendszerén belül elhelyezkedő, JPanel [10] típusú osztály. Példányosításnál egy Vector típusú Vector-t vár, majd abból és egy előre beégetett, fejléc neveket tartalmazó Vector-ból generál egy JTable táblázatot, továbbá vízszintes és függőleges csúszkákat. A megjelenés itt is a UIManager segítségével lett beállítva. A táblán bal klikkel ki lehet jelölni sorokat, jobb klikk esetén pedig felugrik egy helyi menü. Az új elem hozzáadásának kivételével minden esetben egy elágazással próbálja szűrni hogy ki lett-e jelölve egy sor, amely indexének 0-nál nagyobbak vagy egyenlőnek kell lennie, és a tábla sorainak számánál kisebbnek (ha nincs egy sor sem kijelölve, az index -1).

```

JPopupMenu popupMenu = new JPopupMenu();

JMenuItem menuItem1 = new JMenuItem("New");
popupMenu.add(menuItem1);
menuItem1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        FactoryGUI.changeData(1, 0);
    }
});

JMenuItem menuItem2 = new JMenuItem("Delete");
popupMenu.add(menuItem2);
menuItem2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(selectedRow >= 0 && selectedRow < tableSource.size()) {
            if(JOptionPane.showConfirmDialog(null, "Are you sure you want to delete the selected data?", "WARNING", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
                FactoryGUI.changeData(2, selectedRow);
            }
        } else {
            JOptionPane.showMessageDialog(null, "You must choose a row for this action!", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }
});

popupMenu.addSeparator();

JMenuItem menuItem3 = new JMenuItem("Change QtyNeed");
popupMenu.add(menuItem3);
menuItem3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(selectedRow >= 0 && selectedRow < tableSource.size()) {
            FactoryGUI.changeData(3, selectedRow);
        } else {
            JOptionPane.showMessageDialog(null, "You must choose a row for this action!", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }
});

```

5.3. ábra: A táblázat elem hozzáadó, eltávolító, és mennyiség módosító menüpontjai

Ezenkívül rendelkezik egy clearSelection() metódussal, ami az aktuálisan kijelölt sornál megszünteti a kiemelést.

### 3.4.4. GanttChartGUI osztály

Ez az JPanel osztály áll kapcsolatban a Production/src/org/jfree könyvtárban [3] implementált diagram készítővel: <https://www.jfree.org/jfreechart/>. Ahogy az irodalomjegyzék címkéje is jelzi, ez a könyvtár egy szabad felhasználású diagram készítő David Gilbert oldaláról.

Megadja hogy egy Gantt típusú diagramot szeretne megjeleníteni, és a lehetséges procedúrákat: az autógyártást, a modellváltást, és az eltelt időt. A createGanttChart() függvény paramétereibe feltölti a címkéket, majd az elkészült, kezdetben üres diagramot visszaküldi.

```
public JPanel getPanel() {  
  
    dataset.add(change);  
    dataset.add(process);  
    dataset.add(timeLine);  
    IntervalCategoryDataset _dataset = dataset;  
    JFreeChart chart = ChartFactory.createGanttChart(  
        "Production Plan", // Chart title  
        "Order ID", // X-Axis Label  
        "Timeline", // Y-Axis Label  
        _dataset);  
  
    panel = new ChartPanel(chart);  
    return panel;  
}
```

5.4. ábra: A Gantt-diagram létrehozása

- **newProcess(start, end, OrderID):** Létrehozza az OrderID által megadott rendelés gyártási folyamatát a diagramon, a start és end érték intervallumában.

```
public void newProcess(LocalDateTime start, LocalDateTime end, long OrderID) {  
    process.add(new Task(String.valueOf(OrderID),  
        Date.from(start.atZone(ZoneId.systemDefault()).toInstant()),  
        Date.from(end.atZone(ZoneId.systemDefault()).toInstant())  
    ));  
}
```

5.5. ábra: Új gyártási folyamat rögzítése a diagramon

- **newChange(start, end, OrderID):** Az OrderID-val jelölt gyártási folyamatot követően létrehoz a diagramban egy csereidőt a start és end intervallumában.
- **newTimeLine(start, end):** Megjeleníti a diagram legutolsó sorában a program megnyitása óta eltelt időt.
- **reset():** Eltávolítja az összes folyamatábrát a diagramról.

### 3.5. Teszt jegyzőkönyv

| Tesztelendő funkció                              | Eredmény  |
|--|---|
| Smart Sort                                       | A rendelések effektív sorrendbe lettek helyezve.                  |
| Smart Sort után Original List                    | A rendelések eredeti helyzetbe lettek állítva.                    |
| Update   | A Gantt-diagramon lévő, eltelt időt jelző folyamatára növekedett. |
| Exit   | A program bezárult.   |
| Filter/Reset                                     | Megjelent az ablak amin be lehet állítani a szűrőt.               |
| Filter/Reset után ablak bezárás                  | Nem történt szűrés.   |
| Filter/Reset után Cancel                         | Nem történt szűrés.   |
| Filter/Reset után, szűrő beállítás nélkül OK     | Nem történt szűrés.   |
| Filter/Reset után, szűrő beállítását követően OK | Csak a szűrő feltételeinek eleget tevő rendelések jelentek meg.   |
| Szűrés után Filter/Reset                         | A szűrő törlődött, és megjelent az összes rendelés.               |
| Jobb klikkelés a táblázaton                      | Megjelent a helyi menü.   |
| Bal klikkelés a táblázaton                       | Az egérmutatónál elhelyezkedő rendelés ki lett jelölve.           |

| <b>Tesztelendő funkció</b>              | <b>Eredmény</b>  |
|---|--|
| Táblázat helyi menüjében New            | Megjelent az ablak amin fel lehet venni új rendelést   |
| Új rendelés megadása után Cancel        | Nem lett létrehozva új rendelés.   |
| Új rendelés megadása után OK            | Egy felugró ablak rákérdezett a felhasználóra, hogy az új adattal együtt kívánja-e rendezni a listát.    |
| Táblázat helyi menüjében Delete         | Egy felugró ablak rákérdezett a felhasználóra, hogy biztosan szeretné-e törölni a kijelölt rendelést.    |
| Törlés megerősítése                     | Egy felugró ablak rákérdezett a felhasználóra, hogy az eltávolított adattal kívánja-e rendezni a listát. |
| Táblázat helyi menüjében Change QtyNeed | Egy megjelenő ablak szövegmezővel megjelent, ahová beírható az új mennyiség.                             |
| Táblázat helyi menüjében Change QtyDone | Egy megjelenő ablak szövegmezővel megjelent, ahová beírható az új mennyiség.                             |
| QtyNeed módosítása                      | A megrendelt mennyiség megváltozott a kijelölt listaelemnél.   |
| QtyDone módosítása                      | A legyártott mennyiség megváltozott a kijelölt listaelemnél.   |
| Táblázat helyi menüjében Move Up        | A kijelölt listaelem felfelé mozgott   |
| Táblázat helyi menüjében Move Down      | A kijelölt listaelem lefelé mozgott  |
| Táblázat helyi menüjében Move Specific  | Egy megjelenő ablak szövegmezőjében lehetett megadni a specifikus helyet                                 |



| Felhasználói hiba   | Hibakezelés  |
|---|--|
| Smart Sort után ismét Smart Sort  | Felugró ablak jelezte hogy már megtörtént a rendezés.  |
| Original List után ismét Original List  | Felugró ablak jelezte hogy már vissza lett állítva a lista.                                      |
| Filter/Reset-nél olyan szűrőt beállítani amely kizárja az összes rendelést            | Felugró ablak figyelmeztette a felhasználót hogy ilyen rendelés nem létezik. Nem történt szűrés. |
| Aktív szűrő mellett módosítani a listát   | Felugró ablak figyelmeztette a felhasználót hogy először kapcsolja ki a szűrést.                 |
| Új rendelés felvételénél hibás adatokat megadni                                       | Felugró ablak figyelmeztette a felhasználót hogy hibás adatokat adott meg.                       |
| Change QtyNeed mezőjébe nem pozitív egész számot beírni                               | Felugró ablak figyelmeztette a felhasználót hogy hibás adatot adott meg.                         |
| Change QtyNeed mezőjébe nullát beírni   | Felugró ablak figyelmeztette a felhasználót hogy a minimum rendelés 1.                           |
| Change QtyDone mezőjébe nem pozitív egész számot beírni                               | Felugró ablak figyelmeztette a felhasználót hogy hibás adatot adott meg.                         |
| Move Up a lista legfelső eleménél   | Felugró ablak figyelmeztette a felhasználót hogy a kijelölt listaelem már legfelül van.          |
| Move Down a lista legalsó eleménél  | Felugró ablak figyelmeztette a felhasználót hogy a kijelölt listaelem már legalul van.           |
| Move Specific mezőjébe 1-nél kisebb, vagy a rendelések számánál nagyobb számot beírni | Felugró ablak figyelmeztette a felhasználót hogy hibás értéket adott meg.                        |

## 4. fejezet

### Források

- A gyártástervező program futtatható állománya:

`Production/dist/Production.jar`

- A program forráskódjai:

`Production/src/production/...`

- A JFree diagram készítő könyvére [3]:

`Production/src/org/jfree/...`

- A gyártástervező Java osztály fájljai:

`Production/build/classes/production`

- A JFree Java osztályai fájljai:

`Production/build/classes/org/jfree/...`

- SQLite adatbázis [6]:

`Production/dist/database/database.db`

- Java Database Connectivity meghajtó [7]:

`Production/dist/lib/sqlite-jdbc-3.20.0.jar`

- Dokumentációhoz felhasznált képek:

`Images/...`

- A program ikonja [2]:

`Production/icon.ico`

## 5. fejezet

### Irodalomjegyzék

A szakdolgozatomhoz felhasznált külső források, melyek **nem** saját szerzemények:

[1]: Java futás idejű környezet – Java Runtime Environment:

<https://www.java.com/en/download/manual.jsp>

[2]: A program ikonja: <https://iconarchive.com/show/kameleon.pics-icons-by-webalys/Graph-Magnifier-icon.html>

[3]: David Gilbert JFree diagram készítő könyvtára: <https://www.jfree.org/jfreechart/>

[4]: Java fejlesztőkészlet 13-as verziója – Java Development Kit 13:

<https://www.oracle.com/java/technologies/javase/jdk13-archive-downloads.html>

[5]: Apache Netbeans IDE integrált fejlesztői környezet:

<https://netbeans.apache.org/download/index.html>

[6]: SQLite adatbázis: <https://www.sqlite.org/index.html>

[7]: Java Database Connectivity meghajtó – JDBC:

<https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc>

[8]: Java SQL könyvtár: <https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>

[9]: Java Util könyvtár: <https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>

[10]: Java Swing könyvtár:

<https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

[11]: Java AWT könyvtár: <https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>

[12]: Java Time könyvtár: <https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html>

## 6. fejezet

### Ábrajegyzék

1.1. ábra: A gyártástervező program kezdőoldala

1.2. ábra: Rendelések szűrése

1.3. ábra: Új rendelés létrehozása

1.4. ábra: Figyelmeztető ablak, ha szűrt adatokat próbálnak módosítani

1.5. ábra: Rendelés mozgatása specifikus sorba

---

2.1. ábra: A program rétegei

2.2. ábra: Adatbázishoz csatlakozó eljárás

2.3. ábra: Rendelések beolvasása adatbázisból

2.4. ábra: Új rendelés feltöltése az adatbázisba

2.5. ábra: Kapcsolat megszakító eljárás

---

3.1. ábra: Vezérlő egység UML osztálydiagramja

3.2. ábra: Specification ősosztály

3.3. ábra: Ősosztályból származtatott Engine gyerekosztály

---

4.1. ábra: A gyártástervező okos rendező algoritmusa

4.2. ábra: Új rendelést listába helyező algoritmus

4.3. ábra: Gantt-diagramot felépítő metódus

---

5.1. ábra: Grafikus felhasználói felület UML osztálydiagramja

5.2. ábra: A SmartSort-ot hívó JMenuItem

5.3. ábra: A táblázat elem hozzáadó, eltávolító, és mennyiség módosító menüpontjai

5.4. ábra: A Gantt-diagram létrehozása

5.5. ábra: Új gyártási folyamat rögzítése a diagramon

## 7. fejezet

### Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani külső témavezetőmnek, Tóth Istvánnak, aki az AlpsAlpine-ban töltött szakmai gyakorlatom ideje alatt, illetve a konzultációk során szakértelmével, ötleteivel, és a gyártástervezés stratégiájával hozzájárult a szakdolgozatom elkészüléséhez. Külön köszönöm a MES csapat többi tagjának is, hogy új módszerekkel és hasznos tanácsokkal bővítették a programozási ismereteimet.

Hálával tartozom továbbá szüleimnek és testvéremnek, akik tanulmányaim során támogattak, és minden helyzetben mellettem álltak.