

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

# **SZAKDOLGOZAT**

**Nyul Péter**

**2022**

**Szegedi Tudományegyetem  
Informatikai Intézet**

**Vizsgaidőszak-ütemező alkalmazás**

**Szakdolgozat**

Készítette:

**Nyul Péter**

programtervező

informatikus BSc szakos

hallgató

Témavezető:

**Dr. Siket István**

adjunktus

Szeged  
2022

### ***Feladatkiírás***

A feladat egy Java asztali alkalmazás elkészítése, mely segítséget nyújt a vizsgaidőszak ütemezésében. A hallgató regisztráció után rögzítheti a felvett kurzusait, a hozzájuk tartozó vizsgaidőpontokat, továbbá egyéb teendőit is, amelyek befolyásolhatják a vizsgák időpontjait. Ezek alapján az alkalmazás elkészít egy javasolt vizsgabeosztást a vizsgák időpontjai, nehézségei, fontosságuk, a hallgató tanulási szokásai és egyéb elfoglaltságait figyelembe véve. Az alkalmazás része egy naptár, mely megjeleníti a hallgató által felvett eseményeket és a javasolt vizsgákat.

## Tartalmi összefoglaló

- **A téma megnevezése:**

*Vizsgaidőszak-ütemező alkalmazás, mely a hallgató kurzusainak, tanulási szokásainak, vizsgaidőpontjainak és egyéb elfoglaltságainak rögzítése után javasol egy vizsgasorrendet. A hallgató elfoglaltságai és vizsgaidőpontjai naptárban megtekinthetők.*

- **A megadott feladat megfogalmazása:**

*Feladat egy Java asztali alkalmazás elkészítése, mely segíti a hallgatókat a vizsgaidőszak ütemezésében. A hallgató rögzítheti kurzusait, az azokhoz tartozó vizsgaidőpontokat, s egyéb elfoglaltságait. Tanulási szokásainak beállítása után naptárban megtekinthetők a felvitt események és a javasolt vizsgasorrend.*

- **A megoldási mód:**

*Java asztali alkalmazás fejlesztése Maven build keretrendszerrel, az MVC tervezési mintát követve. A projekt multimodulos felépítésű, a grafikus felhasználói felület JavaFX-szel készült. Az alkalmazás online MySQL adatbázissal kommunikál.*

- **Alkalmazott eszközök, módszerek:**

*A programkódot Java nyelven fejlesztettem, fejlesztőkörnyezetként az IntelliJ IDEA-t, build keretrendszerként pedig a Maven-t használtam. Unittesteléshez a JUnit5-öt, adatbázis-eléréshez a JDBC-t, adatbázisként egy távoli MySQL adatbázist, a lekérdezésekhez pedig SQL nyelvet. A grafikus felületeket JavaFX-szel és SceneBuilder-rel hoztam létre. A verziókövetés GitHub-bal, a kódminőség ellenőrzése pedig SonarQube-bal történt. Az alkalmazás rétegeinek kialakítását az MVC tervezési minta szerint végeztem.*

- **Elért eredmények:**

*Megvalósítottam egy Java asztali alkalmazást, mely grafikus felhasználói felülettel rendelkezik és egy távoli adatbázis szerverrel kommunikál. A felhasználó regisztrálhat, s ezt követően kurzusokat vehet fel, módosíthatja ezek paramétereit. Az egyes kurzusokhoz vizsgaidőpontokat adhat meg. Beállíthatja tanulási szokásait, megadhatja a vizsgaidőszakhoz tartozó egyéb elfoglaltságait, s azok időtartamát. Az alkalmazás rendelkezik egy naptárral, melyben a hallgató javasolt vizsgaidőpontjai és egyéb tevékenységei szerepelnek.*

- **Kulcsszavak:**

*Java, IntelliJ IDEA, Maven, JavaFX, SceneBuilder, SQL, JDBC, GUI, ütemezés*

## **Tartalomjegyzék**

<b>Feladatkiírás .....</b>	<b>2</b>
<b>Tartalmi összefoglaló .....</b>	<b>3</b>
<b>Tartalomjegyzék.....</b>	<b>4</b>
<b>BEVEZETÉS .....</b>	<b>6</b>
<b>1. FELHASZNÁLT TECHNOLOGIÁK .....</b>	<b>7</b>
1.1. Java .....	7
1.2. JavaFX.....	7
1.3. Adatbázis-kezelés .....	8
1.4. Fejlesztést támogató eszközök.....	8
1.5. Tesztelés.....	9
1.6. Kódminőség-ellenőrzés .....	10
1.7. Verziókezelés.....	10
<b>2. AZ ALKALMAZÁS FELÉPÍTÉSE .....</b>	<b>11</b>
2.1. A projekt áttekintése .....	11
2.2. Model réteg .....	12
2.2.1. User osztály .....	12
2.2.2. Subject osztály.....	15
2.2.3. Event osztály .....	16
2.2.4. SortingAlgorithm osztály .....	20
2.2.5. DAO osztályok.....	24
2.3. Felhasználói felület.....	27
2.3.1. Bejelentkezés.....	28
2.3.2. Kurzusok kezelése.....	31
2.3.3. Vizsgaidőpontok kezelése .....	35
2.3.4. Általános események kezelése .....	38
2.3.5. Naptár .....	42
2.3.6. Regisztráció .....	44
2.3.7. Beállítások.....	48
<b>3. ÖSSZEFOGLALÁS.....</b>	<b>50</b>

<b>Irodalomjegyzék .....</b>	<b>51</b>
<b>Nyilatkozat .....</b>	<b>52</b>
<b>Köszönetnyilvánítás .....</b>	<b>53</b>

## BEVEZETÉS

Szakdolgozati témám egy vizsgaidőszak-ütemező asztali alkalmazás elkészítése, mely az alapvető naptár funkciókon túl – a lehetséges vizsgaidőpontok, a hallgató tanulási szokásai és az esetleges elfoglaltságai alapján – javasol egy vizsgasorrendet. A témával ismerkedve irodalomkutatást végeztem, hasonló funkcióval rendelkező alkalmazást nem találtam. A fejlesztés Java nyelven történt, fejlesztőkörnyezetként IntelliJ IDEA-t használtam, mely beépítve tartalmazza a Maven build keretrendszert is. A Maven a build-elésen túl egy átlátható struktúrájú projektet biztosít, emellett kezeli az esetleges függőségeket is. A projekt multimodulos, szerkezetének kialakításában az MVC – Model-View-Controller – tervezési mintát követtem. Az MVC tervezési minta interaktív felhasználói felülettel rendelkező alkalmazásokhoz kínál feladatkör alapján differenciált struktúrát. [1] A grafikus felhasználói felület összeállításához a JavaFX könyvtárat használtam, a grafikai elemek Scene-hez történő hozzáadása SceneBuilder-rel és programkódból történt. A tesztelést JUnit5 egységteszt-keretrendszerrel, a kódminőség ellenőrzését SonarQube-bal végeztem. Figyelmet fordítottam a „Clean Code” elvekre, s a Java kódolási konvenciók betartására. A futtatható jar állományt – mely egyetlen állományban tartalmazza az összes modult és a függőségeket is – a Maven Assembly Plugin segítségével hoztam létre.

A következő fejezetekben ismertetem a felhasznált technológiákat, bemutatom a Model és – köztük – DAO osztályokat, majd pedig a GUI-val – vagyis a grafikus felhasználói felülettel – párhuzamosan haladva részletezem az alkalmazás működését. Végül pedig összegzem az elért eredményeket.

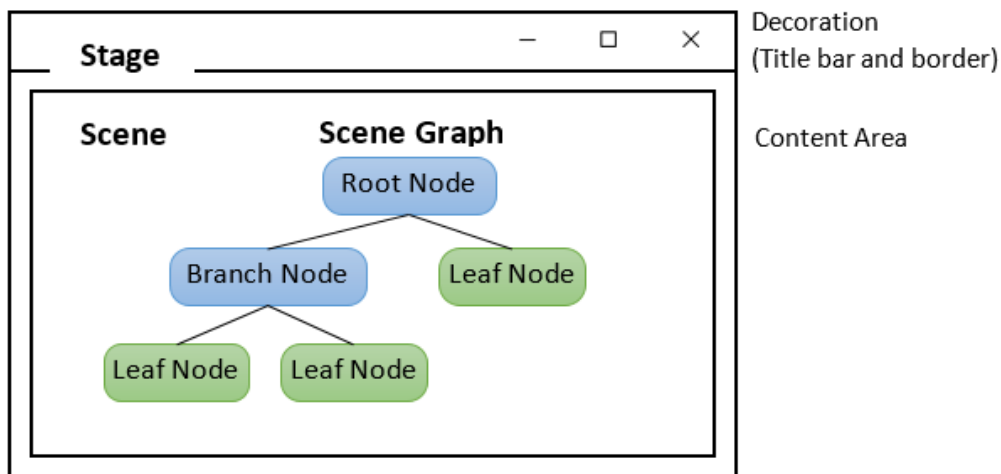
# 1. FELHASZNÁLT TECHNOLOGIÁK

## 1.1. Java

A Java nyelvet a Sun Microsystems cég fejlesztette ki 1991-ben. A nyelv egyik meghatározó tulajdonsága – s egyben létrehozásának is egyik fő oka – a platformfüggetlenség. A Java fordító által létrehozott kód nem futtatható, hanem úgynevezett bájtkódra fordul. A bájtkódot a Java Virtuális Gép (JVM) futtatja, s bármely olyan platformon futtatható, amelyre készült JVM. A JVM a Java Futtatási környezet (JRE) része. [2] A platformfüggetlenségen túl a nyelv további fontos tulajdonsága az objektumorientáltság. A TIOBE Index szerint – 12 hónap helyezéseinek átlagát tekintve – 2022-ben a 3. legnépszerűbb programozási nyelv. [3]

## 1.2. JavaFX

A grafikus felhasználói felületet JavaFX használatával hoztam létre. A JavaFX egy grafikai és média elemeket tartalmazó könyvtár, mely a projektben az MVC tervezési minta View, azaz megjelenítési rétegének megvalósításához tartalmaz osztályokat. A 2.2-es verziótól kezdődően része a Java Development Kit-nek. [4]



1.1. ábra. Az ablakok felépítése [5]

A grafikus elemeket a Scene Graph tartalmazza, csúcsán a Stage áll. A Stage egy ablak, – 1.1. ábra – mely tartalmazza a megjeleníteni kívánt Scene-t. [6] Az alkalmazásban összesen



két Stage-t hoztam létre. Mindkettő statikus, az egyik a munkaablakot, míg a másik egy modális ablakot reprezentál. A Scene Node-okat tartalmaz, melyek további Node-okkal rendelkezhetnek. A GUI összeállítása történhet közvetlenül Java kódból vagy pedig úgynevezett fxml-ből. Az fxml egy xml alapú leíró. A projekt fxml állományait a desktop modulban a resources/fxml könyvtárban hoztam létre. Az fxml állományok tartalmának összeállítását nagymértékben segíti a SceneBuilder alkalmazás. Az egyes grafikus elemek egyszerűen – kód írása nélkül – egér segítségével hozzáadhatók a különféle tárolókhoz, majd mentéskor pedig a módosított fxml-ben létrejön a Node hierarchia.

### **1.3. Adatbázis-kezelés**

Az alkalmazás egy távoli, online MySQL adatbázissal kommunikál. Az adatbázis tábláit phpMyAdmin segítségével hoztam létre. A kapcsolat létrehozásáért és az SQL-utasítások végrehajtásáért a JDBC felel. Az adatbáziskapcsolat kialakítása – Connection objektum – az egyes DAO interfészeket implementáló osztályok konstruktorában történik, a statikus DriverManager().getConnection() függvényhívással. A kapcsolat kialakításához szükséges adatok „properties” állományban kerültek tárolásra, a Properties osztály load() metódusával betöltve az említett getConnection() függvény ezeket kapja meg paraméterekként. A kapcsolat kialakításához szükséges metódusokon kívül a JDBC az SQL-utasításokhoz PreparedStatement osztályt is tartalmaz. Ezt elsősorban – a későbbiekben ismertetett – az „SQL-Injection” típusú támadások kivédése érdekében használtam.

### **1.4. Fejlesztést támogató eszközök**

Az alkalmazás fejlesztéséhez az IntelliJ IDEA-t használtam. A kód fejlesztését – többek közt – kódkiegészítéssel, különféle hibák jelzésével, refactor-álási lehetőséggel is segíti. Build keretrendszerként Maven-t használtam, melyet az IntelliJ IDEA beépítve tartalmaz. Egy átlátható struktúrájú projekt biztosítása mellett meggyorsítja a fejlesztést azzal, hogy támogatja archetype-ok – vagyis sablonok – használatát. A projekt moduljaiban és gyökérkönyvtárában található egy-egy pom.xml – Project Object Model – állomány, mely xml struktúrában tartalmazza az adott projekt verzióját, goal-jait, esetleges függőségeit. A Maven

használatát kiváltképp indokoltá teszi, hogy – pom.xml-ben történő felsorolás után – a függőségeket nem szükséges lokálisan tárolnunk, azokat egy távoli szerverről – a Maven Central Repository-ből – build-eléskor automatikusan letölti. Rendelkezik egy terminállal is, mellyel összeállíthatunk és futtathatunk egyéni goal-okat is.

## 1.5. Tesztelés

A tesztelést a JUnit5 egységteszt-keretrendszerrel végeztem. A projekthez az 1.2. ábrán látható függőséggel adhatjuk hozzá.

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.9.0</version>
</dependency>
```

1.2. ábra. Függőség megadása a pom.xml-ben

Érdemes a parent pom.xml-hez hozzáadni, ugyanis akkor minden modulban elérhető lesz. Az IntelliJ IDEA segít a tesztek létrehozásában, például az „Alt + Insert” billentyűk lenyomásával előhozható a „Generate” menü, melyből a „Test...” kiválasztásával megjelenik a „Create Test” ablak. Itt kiválaszthatjuk a teszt könyvtárat és a tesztelendő metódusokat. A tesztosztályokat külön könyvtárban – test – hozza létre. Az 1.3. ábrán egy tesztmetódus látható.

```
@Test
@DisplayName("isEventInThePast test")
void isEventInThePast() {
    assertTrue(Event.isEventInThePast(LocalDate.now().atStartOfDay()));
    assertTrue(Event.isEventInThePast(LocalDateTime.now().minus(1, ChronoUnit.MINUTES)));
    assertFalse(Event.isEventInThePast(LocalDateTime.now()));
    assertFalse(Event.isEventInThePast(LocalDateTime.now().plus(1, ChronoUnit.MINUTES)));
    assertFalse(Event.isEventInThePast(LocalDate.now().plusDays(1).atStartOfDay()));
    assertFalse(Event.isEventInThePast(LocalDate.now().plusWeeks(1).atStartOfDay()));
}
```

1.3. ábra. Tesztmetódus

A @Test annotáció jelzi, hogy az isEventInThePast() metódus egy tesztmetódus és lefuttatásra kerül. A @DisplayName annotációval megadott String lesz a tesztmetódus neve, ez jelenik meg a tesztesetek futtatásakor. A metódus törzsében kétféle metódus – az

`Assert` osztály metódusai – láthatóak. A példában az `assertTrue()` metódus paraméterként megkapja a tesztelendő statikus függvényt. Amennyiben a függvény `true`-val tér vissza, akkor a teszt sikeres. `False` esetén viszont `AssertionError` kivételt dob. Az `assertFalse()` függvény működési elve hasonló, viszont a visszatérési értéket `false`-szal veti össze. Használható még az `assertEquals()` függvény is, mellyel két érték hasonlítható össze, s amennyiben eltérnek, az általunk paraméterben megadott `String`-gel hibaüzenetet dob. A tesztosztály sikerességének feltétele, hogy az összes tesztmetódus sikeresen lefusszon.

## 1.6. Kódminőség-ellenőrzés

A kód minőségének ellenőrzését SonarQube szoftverrel végeztem. A vizsgálandó állományok elérési helyének megadása után, – Maven parancssorból történő futtatást követően – az eredményeket megtekinthetjük – localhost-on, alapértelmezetten a 9000-es porton – egy böngésző ablakban. Többféle hibakategóriát különböztet meg, többek között: bug-ok, sebezhetőségek, kódszag. Nagymértékben segíti a hibák lokalizálását, hogy az eredmények listázásakor megjeleníti a hibát tartalmazó osztály nevét és sorszámmal együtt a hibát tartalmazó sort. A vizsgálatok eredménye több hibára is rámutatott, többek között jelezte a kódismétléseket és a potenciális bug-okat is. Felhívta a figyelmet a felesleges – kikommentelt, – kódrészletekre, melyek rontják a kód olvashatóságát. Egy esetben jelezte például azt is, hogy az egyik inicializált változó nevében szerepel a „password” `String`, így előfordulhat, hogy a kód beégetett jelszót – s ezzel biztonsági kockázatot - tartalmaz. Az IntelliJ IDEA plugin-jai közt megtalálható a SonarLint, mely a SonarQube-hoz hasonló funkcionalitással rendelkezik, viszont közvetlenül a kódban jelöli a hibákat, lerövidítve a hibakeresési folyamatot. Mindkét szoftvert kifejezetten hasznosnak tartom, a kilistázott hibák javítása közben több szebb – vagy épp hatékonyabb – megoldást sajátítottam el.

## 1.7. Verziókezelés

Verziókezelő használatával nyomon követhetjük a projektünk változásait, szükség esetén visszaállíthatunk egy korábbi állapotot. [7] Az projekt verziókövetéséhez az egyik

legismertebb verziókezelőt a Git-et, a projekt online tárolásához pedig a GitHub-ot használtam.

## 2. AZ ALKALMAZÁS FELÉPÍTÉSE

### 2.1. A projekt áttekintése

A projekt Maven build keretrendszerrel létrehozott, multimodulos szerkezetű, továbbá megvalósítja az MVC tervezési mintát. Két modult tartalmaz, melyek a `core` és a `desktop`. A `core` modulhoz az MVC tervezési minta `Model` osztályai tartoznak. A `core` modul tartalmaz egy `model` csomagot, melybe a `Bean` osztályok és egy `dao` csomagot, melybe pedig a `DAO` osztályok kerültek. A `core` modul része még egy test könyvtár, melyben a JUnit5-tel létrehozott tesztosztályok találhatók. A `desktop` modul – mely létrehozása `javafx-maven-archetypes`-ből történt – csomagjaiban található osztályok és `fxml` állományok valósítják meg az MVC tervezési minta `View` – azaz Nézet – rétegét. A `View` és a `Model` rétegek közti kapcsolatot a `Controller` valósítja meg. A `desktop` modul tartalmaz egy `resources` könyvtárat, melyben az `fxml` állományokat tartalmazó `fxml` könyvtár és a stíluslapokat tartalmazó `css` könyvtár található. A `resources` könyvtár ezenfelül tartalmaz még egy `icons` könyvtárat, melyben az ablakok és a `Controller` osztályok `initialize()` felüldefiniált metódusában beállításra kerülő gombok ikonjai<sup>1</sup> találhatók. Az ismertetetteken túl a `desktop` modul tartalmaz még egy `controller` csomagot, melybe a `Controller` osztályok kerülnek. Minden `Controller` osztályhoz tartozik egy `fxml` állomány. Az `fxml`-ekben található – `fx:id`-val ellátott – elemek neve megegyezik az `fxml`-hez tartozó `Controller` osztályban található `field`-ek nevével, s a kettő közötti kapcsolatot az `@FXML` annotáció teremti meg. A multimodulos struktúra előnye, hogy a modulok közös függőségei – és további közös konfigurációi – kiszervezhetők az úgynevezett parent POM-ba.

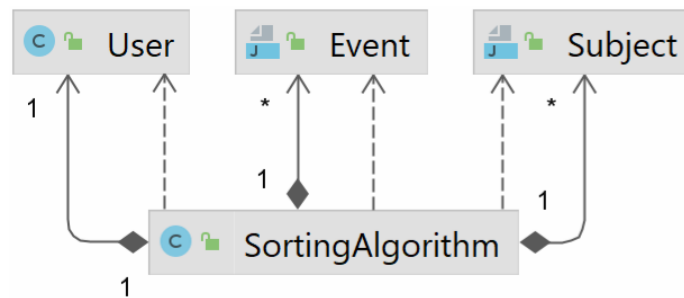
A továbbiakban ismertetem a `Model` osztályokat, majd pedig a GUI-val párhuzamosan haladva részletezem a `Controller` osztályok működését.

---

<sup>1</sup> A felhasznált ikonok a [www.icons8.com](http://www.icons8.com) weboldalról származnak.

## 2.2. Model réteg

Az egyes Model osztályok egy-egy felhasználó adatait, kurzusait, vizsgaidőpontjait, egyéb eseményeit tárolják. A `SortingAlgorithm` osztály sorrendek kialakításához és listák szűréséhez használt metódusokat tartalmaz, felhasználva a többi osztályból létrehozott objektumokat. A 2.1. ábrán látható diagram<sup>2</sup> a Model osztályok közötti kapcsolatot szemlélteti. Az osztályok adatai és metódusai az egyes osztályok ismertetésénél kerülnek bemutatásra.



2.1. ábra. Model osztályok

A DAO osztályok is a Model réteghez sorolhatók, a jobb kódbeli differenciáltság és az egységes feladatkör végett külön csomagban – dao – hoztam létre őket.

### 2.2.1. User osztály

Az `User` publikus láthatóságú osztály egy-egy felhasználót tárol. Az egyes `User` példányok megfeleltethetők az adatbázis `USERS` táblája egy-egy rekordjának. Az osztály adatai privát láthatóságúak, így elérésük és módosításuk is csak publikus metódusokon, úgynevezett getter-eken és setter-eken keresztül lehetséges. Egy darab paraméter nélküli konstruktorral rendelkezik. Ezt – ha nincs több konstruktor – a fordító automatikusan létrehozza, nem szükséges beletenni a kódba. Minden adattag egy-egy `JavaFX Property`, melyek megvalósítják az Observer tervezési mintát. Adatkötéseket hozhatunk létre, s az Observer tervezési mintának köszönhetően észlelhetjük a property-kben bekövetkező változást. Mivel absztrakt osztályok – így nem példányosíthatók – ezért megvalósításuk `Simple` – a típusoknak megfelelő nevű, például `int`-hez `SimpleIntegerProperty` - osztályokon keresztül történik. A kétparaméteres konstruktorát használtam, melynek első paramétere – az

<sup>2</sup> Az UML diagramokat IntelliJ IDEA-val hoztam létre.

osztály összes property-jénél `this` – jelzi, hogy melyik Bean osztályhoz tartozik, a második paraméter pedig a property neve. A JavaFX Property-k esetében mind a getter-eknek, mind pedig a setter-eknek kétféle megvalósítása létezik. `get()` és `set()` a primitív típusokhoz, `getValue()` és `setValue()` pedig az objektumokhoz. A továbbiakban osztálydiagrammal – 2.2. ábra – szemléltetem az User osztály felépítését és – ahol szükségesnek látom – ismertetem az adattagok jelentését.

User		
f 🔒	username	StringProperty
f 🔒	endOfExamPeriod	ObjectProperty<LocalDate>
f 🔒	beginOfSupplementaryExamPeriod	ObjectProperty<LocalDate>
f 🔒	id	IntegerProperty
f 🔒	maxPossibleExamPerSemester	IntegerProperty
f 🔒	restDayOnSaturday	BooleanProperty
f 🔒	password	StringProperty
f 🔒	durationOfDailyActivities	IntegerProperty
f 🔒	beginOfExamPeriod	ObjectProperty<LocalDate>
f 🔒	learningTimePerDay	IntegerProperty
f 🔒	endOfSupplementaryExamPeriod	ObjectProperty<LocalDate>
f 🔒	maxPossibleExam	IntegerProperty
f 🔒	restDayOnSunday	BooleanProperty
m 🏠	User()	
m 🏠	isThePasswordCorrect(String, String)	boolean
m 🏠	passwordHasher(String)	String

2.2. ábra. Az User osztály

IntegerProperty maxPossibleExam:

Ez az adattag tárolja a felhasználó maximális vizsgalehetőségeinek számát.

IntegerProperty maxPossibleExamPerSemester:

Ez az adattag tárolja a felhasználó maximális vizsgalehetőségeinek számát szemeszterenként.

IntegerProperty durationOfDailyActivities:

Ez az adattag a felhasználó napi általános teendőinek összidejét tárolja órában. Például pihenés, étkezések összidőtartama.

IntegerProperty learningTimePerDay:

Ez az adattag a felhasználó napi tanulási idejét tárolja órában.

BooleanProperty restDayOnSaturday:

A felhasználó beállíthatja, hogy szombati napokon kíván-e tanulni.

BooleanProperty restDayOnSunday:

A felhasználó beállíthatja, hogy vasárnapokon kíván-e tanulni.

Az osztály a getter-eken, setter-eken és a felüldefiniált toString() metóduson kívül két metódust tartalmaz, melyek a checkPassword() és a passwordHasher(). A checkPassword() két String paramétert vár, az első a megadott jelszó, a második pedig a hash-eljt jelszó. A metódus boolean-nel tér vissza az összehasonlítás eredménye szerint. A passwordHasher() metódus egy String paramétert vár – a hash-elendő jelszót – majd a hash-eljt jelszóval tér vissza, mely String típusú. Az User osztály megfeleltethető egy adattáblának, mely oszlopai az egyes adattagok. A MySQL adatbázis USERS tábláját a 2.3. ábrán látható SQL-utasítással hoztam létre:

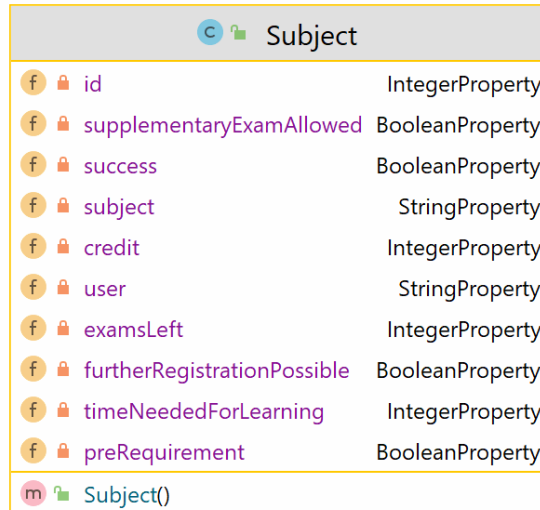
```
CREATE TABLE `USERS` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` char(15) NOT NULL,
  `password` text NOT NULL,
  `max_exam` int(2) NOT NULL,
  `max_exam_per_semester` int(1) NOT NULL,
  `begin_of_exam_period` date NOT NULL,
  `end_of_exam_period` date NOT NULL,
  `begin_of_supp_exam_period` date NOT NULL,
  `end_of_supp_exam_period` date NOT NULL,
  `duration_of_daily_activities` int(2) NOT NULL,
  `learning_hours_per_day` int(2) NOT NULL,
  `saturday` tinyint(1) NOT NULL,
  `sunday` tinyint(1) NOT NULL,
  PRIMARY KEY(`id`)
);
```

2.3. ábra. Az USERS táblát létrehozó SQL-utasítás

Elsődleges kulcs az id, egész típusú, értéke eggyel nő minden egyes rekord beszúrásakor. A felhasználónevet karakterként tárolja, maximálisan megengedett 15 karakteres hosszúsággal. A jelszó text típusú, tárolása az adattáblában hash-elve történik. A maximális vizsgaszám és a szemeszterenkénti maximális vizsgaszám egészként kerül letárolásra. Az időszakok kezdetei és végei date típusúak. A napi tanulási mennyiség egészként tárolódik. A szünnapok boolean típusúak, ezek letárolására egy-egy 1 hosszúságú tinyint elegendő. A maximális hosszak ellenőrzése programkódból történik.

### 2.2.2. Subject osztály

A `Subject` osztály a felhasználó egy-egy kurzusát tárolja. Az `User` osztály felépítéséhez hasonlóan itt is privát láthatóságúak az adattagok. Egyetlen default konstruktorral rendelkezik. A 2.4. ábrán látható a `Subject` osztály osztálydiagramja, melyen nem jelöltem a getter-eket, setter-eket és a felüldefiniált `toString()` metódust sem.



2.4. ábra. A `Subject` osztály

Részletezem az esetlegesen magyarázatra szoruló adattagok jelentését:

`IntegerProperty examsLeft`:

Ez az adattag tárolja a felhasználó megmaradt vizsgalehetőségeinek számát az adott kurzusra vonatkozóan.

`IntegerProperty timeNeededForLearning`:

Ez az adattag tárolja az adott kurzus vizsgájához szükséges felkészülési időt órákban.

`BooleanProperty furtherRegistrationPossible`:

Sikertelenség esetén felveheti-e még a kurzust.

`BooleanProperty success`:

Teljesítette-e a kurzust.

`BooleanProperty preRequirement`:

Van-e előfeltétele a kurzusnak.

`BooleanProperty suppExamAllowed`:

Utóvizsgára jelentkezhet-e az adott kurzusból.



Az `USERS` táblát létrehozó SQL-utasítás a következő, 2.5. ábrán látható:

```
CREATE TABLE `SUBJECTS` (
  `id` int NOT NULL AUTO_INCREMENT,
  `subject` text NOT NULL,
  `username` char(15) NOT NULL,
  `credit` int(2) NOT NULL,
  `exams_left` int(2) NOT NULL,
  `further_reg` tinyint(1) NOT NULL,
  `hours_needed_for_learning` int(2) NOT NULL,
  `prerequisite` tinyint(1) NOT NULL,
  `success` tinyint(1) NOT NULL,
  `supp_exam_allowed` tinyint(1) NOT NULL,
  PRIMARY KEY(`id`)
);
```

2.5. ábra. Az `USERS` táblát létrehozó SQL-utasítás

### 2.2.3. Event osztály

Az `Event` osztály tárolja a hallgató által rögzíthető eseményeket. Az osztály egy default konstruktorral rendelkezik. A 2.6. ábrán látható az `Event` osztály osztálydiagramja. A diagramon nem jelenítettem meg a getter-eket, setter-eket és a felüldefiniált `toString()` metódust sem. A metódusok és paramétereik részletezése az adattagok ismertetése után következik. Az `Event` osztály adattagjai – az esetlegesen magyarázatra szorulókat – a következők:

`IntegerProperty` type:

Az esemény típusát jelenti, 0 ha vizsga, 1 ha általános esemény. Beállítása minden esetben `enum` használatával történik.

`BooleanProperty` active:

Ha értéke `true`, akkor az adott esemény bekerül a – későbbiekben ismertetett – különféle listákba, ha `false` akkor pedig nem.

Event	
<code>MAX_DURATION_OF_GENERAL_EVENT_IN_MINUTE</code>	<code>int</code>
<code>id</code>	<code>IntegerProperty</code>
<code>MINUTES_IN_A_DAY</code>	<code>int</code>
<code>MAX_DURATION_OF_EXAM_EVENT_IN_MINUTE</code>	<code>int</code>
<code>DATE_TIME_PATTERN</code>	<code>String</code>
<code>nameOfEvent</code>	<code>StringProperty</code>
<code>endOfEvent</code>	<code>ObjectProperty&lt;LocalDateTime&gt;</code>
<code>active</code>	<code>BooleanProperty</code>
<code>username</code>	<code>StringProperty</code>
<code>beginOfEvent</code>	<code>ObjectProperty&lt;LocalDateTime&gt;</code>
<code>HOURLY_TO_MINUTES</code>	<code>int</code>
<code>type</code>	<code>IntegerProperty</code>
<hr/>	
<code>Event()</code>	
<code>localDateTimeParser(String, String)</code>	<code>LocalDateTime</code>
<code>durationOfEventInMinutes(LocalDateTime, LocalDateTime)</code>	<code>int</code>
<code>dateAndTimeToString(DatePicker, ComboBox, ComboBox)</code>	<code>String</code>
<code>toString()</code>	<code>String</code>
<code>areTheDateAndTimeOfAOneTimeEventCorrect(DatePicker, DatePicker, ComboBox, ComboBox, ComboBox, ComboBox)</code>	<code>boolean</code>
<code>isTheTimeOfARegularEventCorrect(Integer, Integer, Integer, Integer)</code>	<code>boolean</code>
<code>isTheEventUnique(List&lt;Event&gt;, String, LocalDateTime, LocalDateTime)</code>	<code>boolean</code>
<code>isEventInTheExamPeriod(Integer, LocalDateTime, LocalDateTime, LocalDate, LocalDate)</code>	<code>boolean</code>
<code>areEventsColliding(List&lt;Event&gt;, LocalDateTime, LocalDateTime)</code>	<code>boolean</code>
<code>isEventInThePast(LocalDateTime)</code>	<code>boolean</code>

2.6. ábra. Az Event osztály

A fejlesztés során többször volt szükségem mértékegység átváltásokra, meghatározott mennyiségek – például egy nap hossza percekben – használatára, ebből kifolyólag több konstanszt hoztam létre. Bár ezeket egyszerűbb műveletekhez használtam, viszont figyelembe vettem, hogy a kódolás során kerülendő az úgynevezett „mágikus számok” – vagyis bizonytalan jelentésű, beégetett értékek – használata. Így a kód is átláthatóbb, s ha az értékeken változtatni kell, akkor csak egy helyen szükséges. A létrehozott konstansok publikusak, statikusak, ezen kívül `final` kulcsszóval vannak ellátva, így értékadás után – amely történhet a változó definiálásakor vagy pedig a konstruktorban – már nem változtathatók. Egy gyakran használt konstans – melyen látható a Java nyelv esetén használt konstans elnevezési konvenció – például:

```
public static final int MINUTES_IN_A_DAY = 24 * 60;
```

Mivel logikailag összetartozó konstansokra is szükségem volt, ezért létrehoztam a 2.7. ábrán látható `enum`-ot is.

```

public enum EventType {

    EXAM(0),

    GENERAL(1);

    private final Integer value;

    EventType(Integer value) {
        this.value = value;
    }

    public Integer getValue() {
        return value;
    }

}

```

2.7. ábra. *EventType* enum

Ezen enum használatának valódi előnye az egyes események type adattagjának beállításakor mutatkozik. Például az alábbi utasításra rátekintve azonnal látszik, hogy vizsgát vagy pedig általános eseményt állítunk-e be típusként:

```

this.examEvent.setType(EventType.EXAM.getValue());

```

Az `Event` osztályban – getter-ein, setter-ein kívül – hoztam létre azokat a metódusokat, amelyek az dátum és idő ellenőrzésével, konvertálásukkal kapcsolatosak. Ezen metódusok mind publikusak és statikusak. Ennek oka, hogy a kódban – osztálytól függetlenül – többször is szükség van rájuk, viszont az `Event` osztály adattagjait nem módosítják, így nem sérül az egységbezárás elve sem. A metódusok:

`dateAndTimeToString()`: paraméterként egy `DatePicker`-t és két `ComboBox`-ot kap. Az első `ComboBox` az órát, a második pedig a percet tartalmazza. A metódus egy olyan `String`-el tér vissza, amely a dátumot és az időt `yyyy-MM-dd HH:mm` formátumban tartalmazza.

`localDateTimeParser()`: két paramétert vár, egy megfelelően – például az imént ismertetett metódus által – formázott `String`-et és egy dátum és idő formátum sablont tartalmazó `String`-et. E két paraméterből egy `LocalDateTime` típusú példányt hoz létre, majd vissza is tér vele.

`areTheDateAndTimeOfAOneTimeEventCorrect()`: hat darab paramétert fogad, az első két paraméter `DatePicker` típusú és az esemény kezdő és befejező dátumát tartalmazza, a további négy paraméter pedig `ComboBox` típusú és az esemény kezdő, befejező óráját és percét tartalmazza. A módszer egy `boolean`-nel tér vissza aszerint, hogy a kapott paraméterek alapján létrehozható-e az adott időpont. Például – többek közt – a kezdő óra nincs-e később a befejezőnél. Ez a módszer azon események dátumát és idejét vizsgálja, melyek egy egyszeri eseményhez tartoznak és azért `DatePicker`-eket és `ComboBox`-okat vesz át, mert hibásan megadott értékek esetén a `setStyle()` módszereiken keresztül – megváltozik a `border color` tulajdonságuk, ezzel is jelezve a hiba okát.

`isTheTimeOfARegularEventCorrect()`: négy darab `Integer` típusú változót fogad paraméterként, melyek egy kezdő órát és percet, s egy befejező órát és percet tartalmaznak. Az előző módszerhez hasonlóan megvizsgálja, hogy időrendileg helyes-e a létrehozni kívánt időintervallum, s ennek megfelelően egy `boolean`-nel tér vissza. Az előzővel ellenben ez a módszer a rendszeres – napi – események időintervallumait ellenőrzi.

`isEventInTheExamPeriod()`: négy paramétert vár, két `LocalDateTime` típusút és két `LocalDate`-et. Az első két paraméter a vizsgált esemény idejének kezdete és vége, a második két paraméter pedig a vizsgaidőszak nyitó és záró napja. A módszer leellenőrzi, hogy a kapott időintervallum a vizsgaidőszakon belül van-e, s ennek megfelelő `boolean`-nel tér vissza.

`isEventInThePast()`: egy darab `LocalDateTime` típusú paramétert fogad, `boolean`-nel tér vissza annak megfelelően, hogy a kapott időpont a `LocalDateTime.now()` statikus módszer által visszaadott érték előtt helyezkedik-e el időben.

`areEventsColliding()`: három paramétert vár, egy `Event`-eket tartalmazó generikus listát, s két `LocalDateTime` típusú kezdő és befejező időpontot. A módszer a lista elemein végigiterálva megvizsgálja, hogy a paraméterként kapott időintervallumban van-e olyan esemény, amely `active` adattagja `true`-e, vagyis ütközik-e az esemény egy másik aktív eseménnyel. Amennyiben nincs ütközés az események között, `true`-val tér vissza.

`isTheEventUnique()`: négy paramétert vár, egy `Event`-eket tartalmazó generikus listát, egy – az esemény nevét tartalmazó – `String`-et és két darab `LocalDateTime` típusú paramétert, az esemény kezdő és befejező időpontját. A módszer `true`-val tér vissza, ha az esemény egyedi, vagyis az `EVENTS` táblában nem szerepel ilyen rekord. Az `EVENTS` adattáblán belül több felhasználóhoz is tartozhat ugyanolyan nevű és idejű esemény, de az

Event-eket tartalmazó generikus lista minden esetben csak a bejelentkezett felhasználó eseményeit tartalmazza.

`durationOfEventInMinutes()`: két `LocalDateTime` típusú paramétert vár, melyek alapján visszaadja az általuk meghatározott időintervallum hosszát.

A 2.8. ábrán látható az EVENTS táblát létrehozó SQL utasítás.

```
CREATE TABLE `EVENTS` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name_of_event` text NOT NULL,
  `username` char(15) NOT NULL,
  `begin_of_event` date NOT NULL,
  `end_of_event` date NOT NULL,
  `type_of_event` tinyint(1) NOT NULL,
  `active` tinyint(1) NOT NULL,
  PRIMARY KEY(`id`)
);
```

2.8. ábra. Az EVENTS táblát létrehozó SQL-utasítás

#### 2.2.4. SortingAlgorithm osztály

Ezen osztály metódusai sorrendeket határoznak meg, listákat szűrnék és rendeznek, létrehozzák a naptár feltöltéséhez szükséges események listáját, beleértve a javasolt vizsgák időpontjait is. Az osztályban többször is felhasználok az `Event` osztály konstansait, melyeket a jobb olvashatóság és a szebb kód végett statikusan importálok. A `SortingAlgorithm` osztály osztálydiagramja a 2.9. ábrán látható.

SortingAlgorithm		
f	examsTaken	List<Event>
f	subjects	List<Subject>
f	exams	List<Event>
f	generalEvents	List<Event>
f	user	User
f	current	LocalDate
m	SortingAlgorithm(User, List<Subject>, List<Event>)	
m	filterAndOrderGeneralEventsByDate(List<Event>)	List<Event>
m	copyEventObject(List<Event>)	List<Event>
m	availableMinutesOnGivenDayCalculator(LocalDate)	int
m	proposedExamDatesWithGeneralEvents()	List<Event>
m	minutesCounter(Event, LocalDate)	int
m	getSubjects()	List<Subject>
m	greedyStrategyForSubjectsOrder(List<Subject>)	List<Subject>
m	filterAndOrderExamsByDate(List<Event>)	List<Event>
m	hasFreeTimeOnExamDay(Event)	boolean
m	hasCollisionOnGivenDay(Event, LocalDate)	boolean
m	isAllExamsEnrolled()	boolean

2.9. ábra. A *SortingAlgorithm* osztály

Az osztály egy darab – a 2.10. ábrán látható – publikus háromparaméteres konstruktorral rendelkezik.

```
public SortingAlgorithm(User user, List<Subject> subjects, List<Event> events) {
    this.user = user;
    this.current = LocalDate.now().isBefore(user.getBeginOfExamPeriod().minusWeeks(1)) ||
        LocalDate.now().isEqual(user.getBeginOfExamPeriod().minusWeeks(1)) ?
        user.getBeginOfExamPeriod().minusWeeks(1) : LocalDate.now();
    this.subjects = greedyStrategyForSubjectsOrder(subjects);
    this.generalEvents = filterAndOrderGeneralEventsByDate(copyEventObject(events));
    this.exams = filterAndOrderExamsByDate(copyEventObject(events));
    this.examsTaken = new ArrayList<>();
}
```

2.10. ábra. A *SortingAlgorithm* osztály konstruktora

A bejelentkezett felhasználó adatait tároló user adattag beállítása után a current adattagot egy úgynevezett „ternary operator” segítségével állítom be a következőképpen: amennyiben az aktuális dátum előbb van, mint a vizsgaidőszak kezdő napja mínusz egy hét, akkor a current adattag a vizsgaidőszak kezdete mínusz egy hétre lesz beállítva, ha pedig később,

akkor az aktuális dátumra. A felhasználó kurzusait tartalmazó subjects adattag beállítása a `greedyStrategyForSubjectsOrder()` privát metódus segítségével történik. Paraméterként kap egy Subject-eket tartalmazó generikus listát. Először az alábbi módon törli a listából azokat a kurzusokat, amelyek success adattagja true-ra van beállítva, vagyis amely kurzusok teljesítettek:

```
subjects.removeIf(subject -> subject.isSuccess());
```

A `Collection.removeIf()` statikus metódus végigiterál az adott kollekció – itt Subject-eket tartalmazó generikus lista – minden elemén, s amelyik elem esetén a lambda kifejezés true-val tér vissza, azt törli. Nagyméretű listák esetén használata célszerűbb lehet iterátorok helyett, ugyanis az idővel lineárisan skálázódik –  $O(n)$  –, míg az előbbiek négyzetesen. [8] Miután a lista kizárólag olyan kurzusokat tartalmaz, amelyeket még nem teljesített a felhasználó – vagy éppen javítani szeretne belőlük – az alábbi Comparator segítségével elvégzem a vizsgasorrend kialakítását:

```
Comparator<Subject> subjectsComparator = Comparator.comparing(Subject::getExamsLeft)
    .thenComparing(Subject::isFurtherRegistrationPossible)
    .thenComparing(Comparator.comparingInt(Subject::getCredit)
        .reversed());
subjects.sort(subjectsComparator);
```

A `Comparator.comparing()` statikus függvény paraméterben megkapja a `Subject::getExamsLeft` úgynevezett metódusreferenciát. Így először a kurzusok összehasonlítása az alapján történik, hogy a felhasználónak hány vizsgalehetősége van vissza az adott kurzusból. Ezután – egyenlőség esetén – a `Comparator.thenComparing()` statikus függvény a `Subject::isFurtherRegistrationPossible` metódusreferencia – vagyis az `isFurtherRegistrationPossible()` getter metódus visszatérési értéke – alapján rendez tovább. További fennálló egyenlőség esetén pedig az imént ismertetetthez hasonlóan összeveti a kurzusok kreditértékét. Az így kapott rendezés viszont csökkenő sorrendű, ezért meghívásra kerül a `reversed()` metódus. Végül pedig a kapott lista elemeit sorba rendezi a `Collection.sort()` metódus, mely paraméterként megkapja a `subjectsComparator`-t. Így egy olyan listát kapunk, amely rendezve van az ismertetett kritériumok szerint. A felhasználó általános eseményeit tartalmazó `generalEvents` adattag beállítása a privát `filterAndOrderGeneralEventsByDate()` metódus segítségével történik. Paraméterként a felhasználó összes eseményét tartalmazó `events` generikus paraméterű lista másolatát kapja. Az objektum másolatát azért szükséges létrehozni, mert a listán végzett módosítás a referenciát is érintené, arra viszont szükség van a

felhasználó vizsgáit tároló exam adattag beállításához is. A másolat úgynevezett „deep copy”, vagyis a rajta végzett módosítások nincsenek hatással az eredeti objektumra. A módszer törzsében először szűrés történik, a paraméterként kapott listából törlődnek az aktuális időpontot megelőző események, a vizsga típusú események, és a nem aktív események. A szűrés után egy foreach ciklussal bejárásra kerül a kurzusokat tartalmazó subjects lista. Egy, a ciklus törzsében elhelyezett feltételes vezérlési szerkezettel megvizsgálja, hogy van-e olyan kurzus a listában, mely esetében engedélyezve van utóvizsga felvétele. Ha nincs ilyen, akkor az utóvizsgaidőszak kezdődátumától kezdődően törli az összes eseményt a listából. Fontos megjegyezni, hogy ezen események törlése csak a paraméterként kapott listából történik, az adatbázis EVENTS táblájából nem, ugyanis szükséges, hogy az egyes kurzusok paramétereit a hallgató még módosíthassa a későbbiekben. Végül egy Comparator beállítása és az események kezdőidőpontjai alapján történő sorba rendezés után a `filterAndOrderGeneralEventsByDate()` módszer visszatér egy rendezett listával, mellyel inicializálja a `generalEvents` adattagot. Az exams adattag beállítása a privát `filterAndOrderExamsByDate()` módszeron keresztül történik. A módszer paraméterként megkapja a hallgató összes eseményét tartalmazó listájának másolatát, „deep copy”-ként. A módszer működése hasonló a `filterAndOrderGeneralEventsByDate()` módszeréhez, szűrések és egy Comparator beállítása után visszaad egy olyan vizsgákat tartalmazó listát, amely szűrve van aktivitás, típus és időpont szerint, ezenkívül nem tartalmazza a teljesített kurzusok vizsgáit. A szűrés kiterjed – amennyiben adott kurzusból nem tehető utóvizsga – az utóvizsgaidőpontok törlésére is. Az osztály az említetteken kívül tartalmaz még egy publikus `proposedExamDatesWithGeneralEvents()` módszert is. Ez a módszer hozza létre és adja vissza azt az eseményeket – vagyis Event objektumokat – tartalmazó listát, amely alapján a naptár feltöltődik. A megfelelő vizsgák kiválasztásához létrehozott algoritmus – mely a 2.11. ábrán látható egy egyszerűsített pszeudokódon – a következő: egy for ciklus bejárja a még nem teljesített kurzusok listáját, majd egy egész típusú változóba gyűjti a napi – percekben mért – tanult időt. Addig lép a következő napra, amíg az összes tanulási idő meg nem haladja az adott kurzushoz tartozó tanulási időt. Amint ez megtörtént – és még maradt idő a tanulási keretből – a következő kurzus – amennyiben nem a lista utolsó eleme – tanulási idejét csökkenti a megmaradt idővel, majd a tanulási időt tartalmazó változó értékét 0-ra állítja. Ezután egy while ciklus bejárja a kurzus lehetséges vizsgaidőpontjait és megkeresi a szükséges tanulási idő letelte utáni legkorábbi vizsgaidőpontot. Ha a vizsgált vizsgaidőpont



nem ütközik egyéb eseménnyel és ugyanarra a napra nincs egyéb vizsga felvéve, akkor felveszi az eseményt a vizsgaidőpontokat tartalmazó – Event generikus paraméterű – listába. A `vanIdő()`-nek elnevezett – a kódban `hasFreeTimeOnExamDay()` – metódust abból kifolyólag hívja meg, mert – bár az ütközés vizsgálatának eredménye látszólag feleslegessé tenné – a felhasználó mindennap végzendő feladatainak időpontjai – például étkezések, pihenés – nincsenek rögzítve, de a maximális időtartamuk viszont igen, s így e vizsgálat nélkül akár az adott nap eseményeinek és a mindennapi teendők idejének összege több is lehetne 24 óránál. Végül a vezérlés visszakerül a kurzusokat bejáró for ciklus elejére, majd az ismertetett utasítássorozat addig ismétlődik, amíg be nem járja a kurzusok – rendezett – listájának összes elemét. A metódusban ezután összefésülésre kerülnek az általános események és a felvett vizsgák, majd a – kezdő időpont szerint növekvő sorrendben történő rendezés után – metódus visszatér a rendezett listával.

```

tanulási idő = 0
vizsgaIndex = 0
lista = lista.hozzáad(összes általános esemény)
nap = első tanulási nap

for összes aktív kurzus
    while tanulási idő < kurzushoz szükséges tanulási idő
        if nap == pihenőnap
            léptet(nap)
            következő iteráció
            tanulásiIdőtNövel(nap eseményei)
            léptet(nap)

        if tanulási idő > kurzus tanulási ideje ÉS kurzus != utolsó
            tanulásiIdőtCsökkent(következő kurzus)

tanulási idő = 0

while vizsgaIndex < kurzus összes vizsgájának száma
    if vizsga kezdete >= nap ÉS !vanVizsga(nap) ÉS !ütközik(vizsga) ÉS vanIdő(vizsga)
        lista.hozzáad(vizsga)
        while vizsgaIndex < következő naphoz tartozó első vizsga indexe
            léptet(vizsgaIndex)
            kilépés a ciklusból
        léptet(vizsgaIndex)

```

2.11. ábra. Pszeudokód

### 2.2.5. DAO osztályok

A DAO – Data Accessing Object – tervezési minta használatával különválasztható az üzleti és az adatelérési logika, ezenkívül előnye, hogy egyetlen interfészt létrehozása mellett, – az adattárolás módjától függően – többféle megvalósítással is rendelkezhet. [9] A megvalósítandó metódusok listáját egy-egy interfész tartalmazza. Az implementációk megvalósítják az úgynevezett CRUD műveleteket, melyek megfeleltethetők egy-egy SQL-

utasításnak. A C – Create – SQL megfelelője az INSERT, ezzel az utasítással beszúrhatunk egy rekordot az adott táblába. Az R – Read – a SELECT SQL-utasításnak felel meg, mellyel lekérdezést hajthatunk végre. Az U – Update – SQL megfelelője az UPDATE, mellyel rekordokat módosíthatunk. Végül pedig a D – Delete – a DELETE SQL-utasításnak felel meg, s a kiválasztott rekordok törlésére szolgál. A dao csomagban összesen három DAO osztályt hoztam létre, egyet a felhasználók adatait tartalmazó – USERS – adattábla, egyet az események – EVENTS – adattábla, és egyet pedig a kurzusok – SUBJECTS – adattábla kezeléséhez. Mivel a DAO osztályok hasonló felépítésűek, ezért az UserDao interfészen és az azt megvalósító UserDaoImpl osztályon keresztül ismertetem a működésüket. Az UserDao interfész – mely a 2.12. ábrán látható – tartalmazza az implementálandó metódusokat.



2.12. ábra. Az UserDao interfész és implementáló osztálya

Az `UserDAOImpl` osztály paraméter nélküli konstruktorában kerül példányosításra a `Properties` objektum, melybe a `resources` könyvtár alatt található `property` állomány tartalma kerül kulcs-érték páronként. Ez tartalmazza az adatbázis-kapcsolat kialakításához szükséges adatokat, például az URL-t. Ezután a `Connection` objektum példányosítása következik, a `DriverManager.getConnection()` statikus metódusának meghívásával. Paraméterben megkapja a `property`-ben megadott URL-t, az adatbázis eléréséhez szükséges felhasználónevet és jelszót. Végül pedig a konstruktorban kerülnek beállításra a `PreparedStatement`-ek is. Használatukat különösképpen indokoltá teszi, hogy védelmet nyújtanak az úgynevezett „SQL-Injection” típusú támadások ellen. Ezen támadástípus azt a sérülékenységet használja ki, hogy amennyiben az SQL-utasítások és a bevitt adatok `String`-ként kerülnek tárolásra – például `Statement`-ben – és egyszerű konkatenációval kerülnek egyesítésre, akkor a támadó az input mezőkön keresztül megadhat olyan `String`-eket is, amellyel módosíthatja a kiegészített SQL-kifejezést vagy éppen saját kódját is futtathatja. `PreparedStatement` használatakor viszont a megadott paraméter ellenőrzésre kerül. Az alábbi kódrészletben egy `PreparedStatement` felépítése látható:

```
this.findUser = connection.prepareStatement("SELECT * FROM USERS WHERE USERNAME = ?");
```

A szükséges paramétert kérdőjel jelöli, a számozás 1-től indul. Használata a következőképpen történik:

```
this.findUser.setString(1, user.getUsername());
```

Így a `findUser` `PreparedStatement` első – és egyetlen – kérdőjelenek helyére beállításra kerül a felhasználó neve a `setString()` metódus segítségével. Az osztály minden egyes metódusa megvalósít egy-egy CRUD műveletet a fent említett SQL-utasítások használatával. Például a paraméterben kapott felhasználó törlésére létrehozott – felüldefiniált – metódus a következő:

```
@Override
public void deleteUser(User user) throws SQLException {
    this.deleteUser.setString(1, user.getUsername());
    this.deleteUser.execute();
}
```

és az alábbi `PreparedStatement`-et egészíti ki:

```
this.deleteUser = connection.prepareStatement("DELETE FROM USERS WHERE USERNAME = ?");
```

Az egyes User példányok létrehozásához privát láthatóságú segédmetódusokat hoztam létre. Az adatbázis-kapcsolat és a PreparedStatement-ek zárása a `close()` metódus meghívásával történik.

### **2.3. Felhasználói felület**

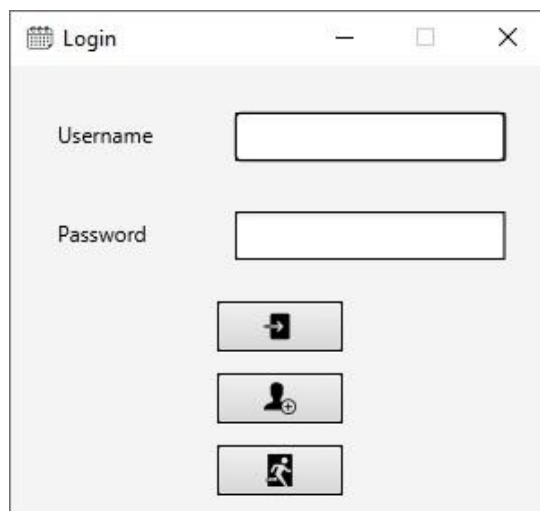
Az alkalmazás belépő pontja a desktop modul App osztályában – mely a `javafx.application.Application` osztályból származik – található `main()` függvény. Ez egyetlen utasítást tartalmaz – a statikus `launch()` metódust – mely végrehajtásával létrejön egy példány a `javafx.application.Application` osztály leszármazottjából a JavaFX Application Thread-en. Ezután meghívódik az `init()`, majd pedig a `start()` felüldefiniált metódus. [10] A `start()` metódusban történik meg a statikus User típusú `user` és a Stage típusú `stage` példányosítása. Az `user` példányosítása paraméter nélküli konstruktorával történik, adatai a későbbiekben – sikeres bejelentkezést követően – kerülnek beállításra setter metódusokkal. A `start()` metódus utolsó utasítása a `setAndShowStageAndScene()` statikus metódus meghívása. Ez a metódus két `String` paramétert vár, melyek jelen esetben a "Login" és a "login\_window". Az első paraméter a megjelenítendő ablak címét, a második pedig a betöltendő fxml állomány nevét tartalmazza. A metódus abból kifolyólag statikus, hogy a tartalmazó osztály példányosítása nélkül is használható legyen, ugyanis osztályoktól függetlenül, többször is szükség lesz rá a statikus `stage` adattag beállításához. Az App osztály tartalmaz még egy statikus Stage típusú adattagot a modális ablakokhoz, ennek beállítása a későbbiekben történik. A továbbiakban a Controller osztályokon keresztül – párhuzamosan haladva a GUI-val – ismertetem az alkalmazás működését. Minden egyes Controller osztály implementálja az `Initializable` interfészt. A felüldefiniált `initialize()` metódusban a field-ek kezdeti értékeinek beállításán kívül történik meg az egyes DAO osztályok példányosítása is implementáló osztályaikkal try-catch blokkokban, az esetlegesen fellépő hibák elkapása végett. Az elkapott hibák naplózásra kerülnek LOG4J2 használatával. A gyakran használt – feladatkörük alapján összetartozó – metódusoknak külön osztályokat hoztam létre az utility csomagban. Ezen osztályok közös jellemzője, hogy kizárólag statikus metódusokat és statikus adattagokat tartalmaznak, konstruktoruk privát láthatóságú, így nem példányosíthatóak. Az utility csomag osztályai a `StageAndSceneInitializer` – mely metódusai a Stage és

a hozzá tartozó Scene beállításáért és betöltéséért felelnek – és a ComboBoxInitializer, mely a ComboBox-ok beállításáért felel.

### 2.3.1. Bejelentkezés

*Az alkalmazás nyitó ablakában – 2.13. ábra – a felhasználó megadhatja felhasználónevét és jelszavát, majd a bejelentkezés szimbólummal ellátott gombra történő kattintással – megfelelő felhasználónév és jelszó esetén – bejelentkezhet, mellyel átkerül a főmenübe. Hibásan megadott adatok esetén figyelmeztetéseket kap. A regisztráció szimbólummal ellátott gombra történő kattintáskor a felhasználót átkerül a regisztrációs ablakba. A kilépés szimbólumra történő kattintással bezárhatja az alkalmazást.*

Az alkalmazás futtatása után a felhasználó a „Login” nevű nyitó ablakba jut – mely vezérlő osztálya a LoginWindowController –, a kurzor pedig automatikusan az „Username” Label-lel ellátott TextField-be kerül. A felhasználónév megadása mellett szükséges megadni a hozzátartozó jelszót is, ez a „Password” Label-höz tartozó PasswordField-be kerül.



2.13. ábra. Bejelentkezés ablak

A `PasswordField` sajátossága, hogy a bevitt karakterek – alapértelmezetten – pontokként jelennek meg, így a jelszó nem olvasható le a kijelzőről, ezzel is növelve a felhasználó biztonságát. Az `username TextField`-hez létrehozásra került az alábbi eseményfigyelő:

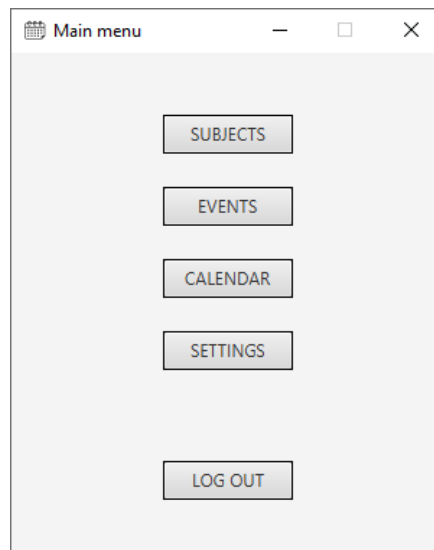
```
this.usernameTextField.setOnKeyPressed(keyEvent -> {
    if (keyEvent.getCode().equals(KeyCode.ENTER)) {
        passwordField.requestFocus();
    }
});
```

mely az `username TextField`-ben `ENTER` billentyű leütése után a fókuszt a – `requestFocus()` metódus meghívásával – a `password PasswordField`-re helyezi, így egér használata nélkül is lehetséges folytatni az adatok megadását. Miután a felhasználó begépelte a felhasználónevét és a jelszavát, majd rákattintott a `Login` gombra – meghívva ezzel a `loginButtonClicked()` metódust –, vagy pedig leütötte az `ENTER` billentyűt a `password TextField`-ben, meghívva ezzel a `login Button fire()` metódusát – többféle eset lehetséges:

1. hibás a felhasználóneve
2. hibás a jelszava
3. hibás a felhasználóneve és a jelszava is
4. megfelelő a felhasználónév és a hozzá tartozó jelszó is

Az 1-3. esetekben meghívódik a `TextField` és a `PasswordField` `setStyle()` metódusa és a „border color” tulajdonságot `#ff0000`-re állítja, vagyis mindkét beviteli mező kerete vörös lesz. Emellett az `username TextField` `setPromptText()` metódusa segítségével beállítja a „wrong username”, a `PasswordField` `setPromptText()` a „wrong password” `String`-eket, a két beviteli mező között pedig megjelenik az „OR” `String`-et tartalmazó `Label`. Ez a `Label` alapértelmezetten nem látható, a `setVisible()` metódusa `false` paraméterrel kerül beállításra. Szerepe a következő: ha a felhasználó támadó szándékkal próbál bejelentkezni és nem tudja a felhasználónevet biztosan, akkor a bejelentkezés gombra történő kattintáskor ne derüljön ki, hogy csak rossz jelszót adott meg, vagy épp nem is létezik a felhasználónév mellyel próbálkozik. A jelszó ellenőrzése a privát `checkUsernameAndPassword()` metóduson keresztül történik. Két `String` paramétert vár, a felhasználónevet és a jelszót. A metódus csak akkor fut le, ha mind az `username TextField`, mind pedig a `password PasswordField` tartalmazznak legalább

egy darab karaktert. A metódus törzsében egy for ciklus végigiterál az összes felhasználót tartalmazó users generikus paraméterrel megadott listán, majd minden egyes User esetén megvizsgálja, hogy a felhasználónév és a jelszó egyezik-e a megadottal. A megadott jelszót a BCrypt checkpw() metódusa hasonlítja össze a hash-elt jelszóval. Amennyiben true-val tér vissza – ez a 4. esetben következik be –, az App osztály statikus user példányának adattagjai setter metódusain keresztül beállításra kerülnek az USERS táblában tárolt értékekre, majd a checkUsernameAndPassword() metódus true-val tér vissza, s így meghívódik a statikus setAndShowStageAndScene() metódus. E metódus két paramétert vár. Az első paraméter a Stage címe lesz, a második pedig a betöltendő fxml állomány neve. A metódus első paraméterként a „Main menu”-t, második paraméterként pedig a „main\_menu”-t kapja, s így átkerülünk a főmenübe – 2.14. ábra –, az ablak címe pedig „Main menu” lesz.



2.14. ábra. A főmenü

*A főmenüben a felhasználó az alábbi lehetőségek közül választhat:*

- *A Subjects feliratú gombra történő kattintással átkerül a felvett kurzusai kezelésére szolgáló ablakba.*
- *Az Events gombra történő kattintással átkerül az általános eseményeit kezelő ablakba.*
- *A Calendar gombra történő kattintással átkerül az eseményeit és vizsgáit tartalmazó naptárt tartalmazó ablakba.*
- *A Settings gombra történő kattintással átkerül az – alapesetben a regisztrációkor megadott – adatait tartalmazó ablakba.*

- *A Log out gombra történő kattintáskor pedig kijelentkezik és visszalép a nyitó, Login ablakba.*

A főmenü gombjaira történő kattintás által kiváltott eseményeket a MainWindowController osztály kezeli. Az initialize() felüldefiniált metódusban a SubjectDAO és az EventDAO osztályok példányosítása után, a findAllSubjects() és a findAllEvents() metódusok meghívásával a felhasználó kurzusaival és eseményeivel feltöltődnek a generikus List<Subject> subjectsList és List<Event> eventsList listák. Ezután egy foreach ciklus segítségével bejárásra kerül az eventsList és az EventDAO deleteEvent() metódusának meghívásával töröl minden – az adott felhasználóhoz tartozó – múltbéli eseményt az EVENTS táblából. Ezután bontásra kerül az adatbázis-kapcsolat. Az egyes gombokra történő kattintáskor meghívódik a setAndShowStageAndScene() statikus metódus és megjelenik a választott ablak. A calendarButtonPressed() metódusban viszont a naptár betöltése előtt ellenőrzésre kerül, hogy üres-e a felhasználó kurzusait tartalmazó lista. Amennyiben üres – vagy pedig nincs egyetlen aktív vizsgaidőpontja sem –, akkor figyelmeztető szöveggel megjelenik egy modális ablak, melynek jóváhagyása után a felhasználó átkerül a kurzusok kezelésére – SUBJECTS – szolgáló ablakba.

### 2.3.2. Kurzusok kezelése

*A „Subjects” ablakban – 2.15. ábra – a felhasználó megtekintheti kurzusait, azok állapotát. Amennyiben szükséges megváltoztathatja az adott kurzus kreditértékét, a kurzusból vissza levő vizsgalehetőségeinek számát, a vizsgára szánt készülési időt, a kurzus sikerességét, az előfeltétel-e tulajdonságot és hogy jelentkezhet-e utóvizsgára. Az egyes kurzusokhoz tartozó „Show” feliratú gombra kattintva megjelennek az adott kurzus lehetséges vizsgaidőpontjai. A „Show” gomb melletti cellában található gombra – „Delete” oszlop, szemétkgyűjtő ikon – kattintva pedig törölhető a kurzus a táblázatból. Kurzus hozzáadása a „+” szimbólumot tartalmazó gombbal lehetséges. A felhasználó a változtatások mentése után – a hajlékonylemez szimbólumra kattintva – visszakérül a főmenübe. A vissza szimbólumot tartalmazó gombra kattintva is visszakérül, viszont akkor az esetleges módosítások nem kerülnek mentésre. A szimbólumokat tartalmazó gombokhoz és a táblázat - esetlegesen*



magyarázatra szoruló - oszlopneveihez megjegyzés is rögzítésre került, mely az egér mutatójának az említett elemek fölé vitele esetén jelenik meg.

Subjects									
Subject	Credit	Exam(s) left(sum)	Reg?	Learning time(hours)	Pre-Requirement	Success	Supp. exam poss?	Exam dates	Delete
Calculus I.	2	3	<input checked="" type="checkbox"/>	15	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Show	
Programming II.	2	2	<input checked="" type="checkbox"/>	18	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Show	
Web Design	2	1	<input type="checkbox"/>	18	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Show	
Discrete Mathematics II.	2	5	<input checked="" type="checkbox"/>	25	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Show	
Application Development I.	1	4	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Show	
Database Systems	2	4	<input checked="" type="checkbox"/>	16	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Show	

2.15. ábra. A felvett kurzusok listája és paraméterei

A kurzusokkal kapcsolatos események kezelését a `SubjectWindowController` osztály metódusai végzik. Az `initialize()` felüldefiniált metódusban, a kurzusokat és az eseményeket kezelő DAO osztályok példányosítása után a `subjects Subject` generikus paraméterű listába bekerülnek a bejelentkezett felhasználóhoz tartozó kurzusok a `findAllSubjects()` metódus segítségével. Egyetlen paraméterként a sikeresen bejelentkezett felhasználó felhasználónevét – `user.getUsername()` – kapja meg. Ezután – a felhasználó vizsgáival – az `exams Event` generikus paraméterű lista töltődik fel a `findAllEvents()` metódus meghívásával. Két paramétert fogad, első paraméter a felhasználó neve, második az esemény típusa. Itt az esemény vizsga típusú, így paraméterként az `Event.EventType.EXAM.getValue()` enum EXAM értékét kapja meg. Ezután a gombok beállítása történik. A `setGraphic()` metódushívással ikon kerül a gombokra, mely megvalósítása például a mentés gomb esetén az alábbi:

```
saveButton.setGraphic(new ImageView(s: "/icons/save.png"));
```

Az ikon hozzárendelése után `Tooltip` beállítása következik:

```
Tooltip.install(this.saveButton, new Tooltip(s: "Save"));
```

Ennek hatására – amennyiben az egér mutatója a gomb fölé kerül – megjelenik a gomb neve. A gombok beállítása után egy `for` ciklus bejárja a kurzusokat tartalmazó listát és elemenként másolatot készít róla. A változtatások a másolat egyes `Subject` típusú példányainak adatait írják felül. Erre azért van szükség, hogy a munka befejeztével, a mentés gombra történő kattintáskor el lehessen dönteni, hogy szükséges-e frissíteni a módosított kurzus adatait az adatbázis `SUBJECTS` táblájában. Ezután a táblázathoz – a `TableView`

`getItems().setAll()` metódusa segítségével – hozzárendelődnek a kurzusokat tartalmazó lista elemei, vagyis az egyes `Subject` példányok. Az egyes oszlopokhoz a lista `Subject` példányainak `field`-jei `setCellValueFactory()` metódusok hívásával képeződnek le. Például a kurzusok neveit tartalmazó első oszlop esetében:

```
subjectColumn.setCellValueFactory(new PropertyValueFactory<>("subject"));
```

Paraméterben a `subject` Bean `Property`-jét kapja. Azon cellák, melyek egy-egy `ComboBox`-ot tartalmaznak, ugyancsak `setCellFactory()` metódus meghívásával inicializálódnak, viszont paraméterben a `ComboBoxInitializer` osztály statikus `initializeCombobox()` metódusát kapják, melynek paramétere két egész szám és egy `ObservableList`-tel tér vissza. Az első paraméter a kezdő, a második pedig a lista befejező értéke, ezek fognak megjeleníteni a lenyíló listában. `ComboBox`-ból választható ki például az egyes kurzusok kreditértéke, a hátralévő vizsgalehetőségek száma. Ezen metódushívások után az egyes oszlopnevekhez `ToolTip` beállítása következik. A „Show” és a „Delete” oszlopok celláinak beállítása `Callback` metódusok segítségével történik. A „Show” esetében a `Callback` első generikus paramétere a generikus `TableColumn<Subject, Void>` – ez a `Callback` paraméterének típusa –, a második paramétere pedig `TableCell<Subject, Void>`, ez pedig a visszatérési típusa. A névtelen osztályban felüldefiniált `call()` metódusban létrehozásra került egy `private final Button`, melynek `setOnAction()` metódusának meghívása `lambda` kifejezéssel megadott paraméterrel történik. A „Show” gombra történő kattintás – esemény – hatására megjelenik egy új ablak, a kurzus nevével és egy táblázattal, mely tartalmazza a vizsgaidőpontokat, aktivitásukat. A táblázatos megjelenítés előtt ellenőrzésre kerül, hogy a kurzus sikeressége esetlegesen át lett-e állítva. Ekkor ugyanis a megjelenő táblázatban az összes aktív mező értéke egységesen átállításra kerül. Ha a felhasználó a kurzust sikeresre állította, akkor az összes vizsgaidőpont inaktív lesz és fordítva. A `call()` metódusban a másik felüldefiniált metódus az `updateItem()`, mely akkor hívódik meg, ha a tartalmazó cella értéke megváltozik, ekkor frissíti annak tartalmát. A „Delete” gombhoz tartozó `callback()` hasonló struktúrájú az imént ismertetetthez, viszont itt a `setOnAction()` névtelen osztályában a `TableView` adott sorához tartozó `Subject` példány elkérése után a `SubjectDAO` `deleteSubject()` metódusa kerül meghívásra, paraméterként a törölendő – `TableView`-től elkért – objektummal. A kurzus törlése után bejárásra kerül a vizsgák listája is és – az előzőhöz hasonlóan – törlődik az összes – kurzushoz kapcsolódó – vizsgaidőpont is. A törlés után újra betöltődik a kurzusokat tartalmazó ablak, a törölt kurzus nélkül. A táblázat

Boolean típust megjelenítő oszlopai egy-egy CheckBox-ot tartalmaznak. Ezek beállítása `setCellFactory()` metódussal történt, paraméterként egy-egy lambda kifejezéssel. Például a „Success” oszlop cellái esetében:

```
this.successColumn.setCellValueFactory(value -> value.getValue().successProperty());
this.successColumn.setCellFactory(CheckBoxTableCell.forTableColumn(successColumn));
```

Az ablak tartalmaz még egy mentés, egy tárgy hozzáadás, egy tanulási beállítások és egy vissza gombot. A mentés gombhoz hozzárendelt eseménykezelőben bejárásra kerül az eredeti kurzusokat tartalmazó lista és objektumonként összehasonlításra kerül az esetlegesen módosított listával. Az objektumok összehasonlítása adattagonként történik. Az összehasonlítás abból kifolyólag történik, hogy elegendő legyen egyszer meghívni a `SubjectDAO updateSubject()` metódusát, ne pedig minden egyes módosításkor. A metódus utolsó utasítása a főmenü betöltése, az adatbázis-kapcsolat bontása után. A hozzáadás gombbal az „Add subject” – 2.16. ábra – ablakba kerülünk.

The image shows a Java Swing window titled "Add subject". Inside the window, there are several form elements: a text field for "Subject", a spinner field for "Credit" with a value of 0, a spinner field for "Exam(s) left" with a value of 1, a checkbox for "Further registration possible", a spinner field for "Hours needed for learning" with a value of 5, and another checkbox for "Prerequisite". At the bottom of the window, there are two buttons: "Save" (with a floppy disk icon) and "Cancel" (with a circular arrow icon).

2.16. ábra. Kurzus hozzáadása

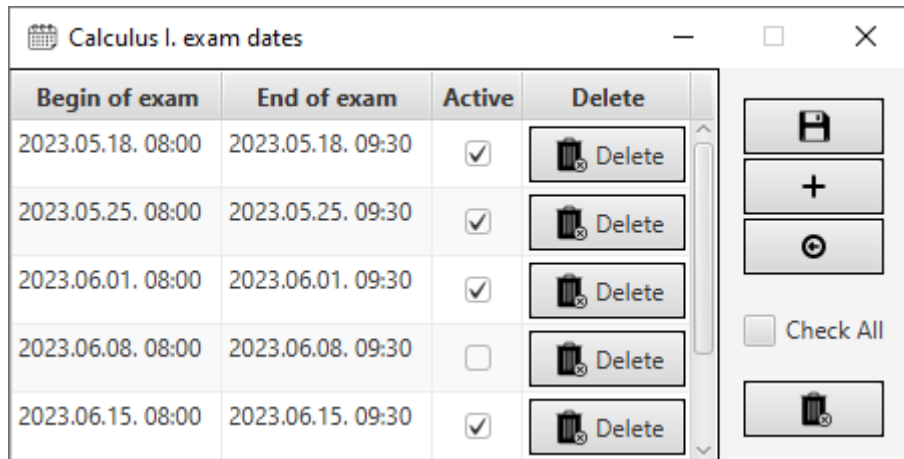
Ebben az ablakban a felveendő kurzus összes paramétere beállítható. Amennyiben a felhasználó nem ad meg nevet vagy pedig létezik kurzus a megadott névvel, hibaüzenet jelenik meg a névhez tartozó input mezőben. Az összes többi beállítás kattintással – `ComboBox`, `CheckBox` – végezhető, ezzel igyekeztem kizárni az esetleges hibás formátumú adatbevitelt. Sikeres hozzáadás esetén a felhasználó visszakerül a kurzusokat tartalmazó ablakba, mely már az új kurzust is tartalmazni fogja a megadott paraméterekkel. A tanulási szokásokat beállító ablak – 2.17. ábra – állítható paraméterként két `CheckBox`-ot és egy `ComboBox`-ot tartalmaz.

2.17. ábra. Tanulási szokások

A CheckBox-ok a tanulási szünnapokat jelentik, a ComboBox pedig a napi tanulási idő mennyiségét órában. Ezen értékek az egyes kurzusok vizsgaidőpontjainak kiválasztásánál kapnak szerepet. Mentés után a felhasználó visszakérül a kurzusokat tartalmazó ablakba.

### 2.3.3. Vizsgaidőpontok kezelése

A kurzusok listáját tartalmazó ablakban, amennyiben a felhasználó a kiválasztott kurzus sorában található „Show” feliratú gombra kattint, megjelenik egy ablak, mely az adott kurzus lehetséges vizsgaidőpontjait – 2.18. ábra – tartalmazza. A kezdő és befejező időpontokon kívül tartalmaz egy „Active” oszlopot is, mellyel a felhasználó jelölheti, hogy az adott időpont megfelelő lenne-e számára, vagyis az időpont részt vegyen-e a – 2.2.4. fejezetben ismertetett – vizsgasorrend generálásában. Amennyiben a kurzusokat tartalmazó táblázatban az adott kurzus teljesítetre van állítva, akkor az „Active” tulajdonság az összes vizsgánál false lesz. A felhasználó a „Delete” oszlop celláiban található gombbal törölheti az adott időpontot. Mentheti beállításait, hozzáadhat további időpontokat – a „+” szimbólumot tartalmazó gombbal – visszatérhet a kurzusokat tartalmazó ablakba, vagy pedig a jobb alsó sarokban található gombbal törölheti az összes időpontot. Az aktivitás beállítása vonatkozhat az összes időpontra is a „Check All” feliratú CheckBox-ra történő kattintással.



Begin of exam	End of exam	Active	Delete
2023.05.18. 08:00	2023.05.18. 09:30	<input checked="" type="checkbox"/>	Delete
2023.05.25. 08:00	2023.05.25. 09:30	<input checked="" type="checkbox"/>	Delete
2023.06.01. 08:00	2023.06.01. 09:30	<input checked="" type="checkbox"/>	Delete
2023.06.08. 08:00	2023.06.08. 09:30	<input type="checkbox"/>	Delete
2023.06.15. 08:00	2023.06.15. 09:30	<input checked="" type="checkbox"/>	Delete

2.18. ábra. Vizsgaidőpontok és beállításaik

A kurzusokat tartalmazó táblázatban – „Subjects” ablak – a kiválasztott kurzus sorában található „Show” gombra történő kattintással a `setAndShowStageAndScene()` metódus betölti az adott kurzus vizsgaidőpontjait és beállításait tartalmazó ablakot. A Stage címe az adott kurzus neve lesz, konkatenálva az „exam dates” String-gel. A megjelenítendő vizsgák szűréséért és az ablak eseményeinek kezeléséért az `ExamEventWindowController` osztály felel. Az osztály felüldefiniált `initialize()` metódusában először az események és a kurzusok eléréséért és adatbázisban történő kezeléséért felelős DAO osztályok kerülnek példányosításra. Az exams Event generikus paraméterű lista bekerülnek a felhasználó vizsgái a következőképpen:

```
this.exams = this.eventDAO.findAllEvents(App.user.getUsername(), Event.EventType.EXAM.getValue());
```

A `findAllEvents()` első paramétere a felhasználó neve, második pedig az esemény típusa, mely jelen esetben vizsga. Az utasítás try-catch blokkban helyezkedik el. Ezután a felhasználó kurzusai kerülnek be a subjects Subject generikus paraméterű lista:

```
this.subjects = this.subjectDAO.findAllSubjects(App.user.getUsername());
```

Egyetlen paramétere a felhasználó neve. A listák feltöltése után a gombok ikonjainak és hozzájuk tartozó ToolTip-ek beállítása következik, a korábbiakban már ismertetett módon. Ezután – a 2.19. ábrán látható módon – annak ellenőrzése következik, hogy az adott kurzus teljesítve van-e. Amennyiben igen, az összes vizsgaidőpont false-ra lesz állítva. Ebben az esetben az időpontok „Active” paramétere nem változtatható – mentéssel sem – addig, amíg az adott kurzus „Success” paramétere újra false-ra nem lesz állítva. Amennyiben a felhasználó többször kattint egymás után a „Success” oszlopban ugyanarra a CheckBox-ra, akkor az egyéni – Active adattagra vonatkozó – beállítások elvesznek és egységesen kerül beállításra

```

for (Subject subject : this.subjects) {
    if (subject.isSuccess()) {
        for (Event exam : this.exams) {
            if (subject.getSubject().equals(exam.getNameOfEvent())) {
                exam.setActive(false);
            }
        }
    }
}

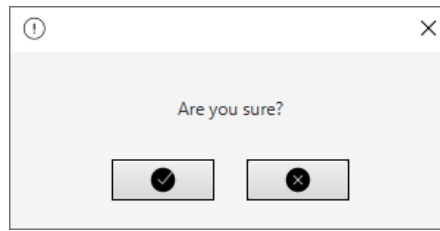
```

2.19. ábra. A teljesített kurzusok vizsgaidőpontjainak kiszűrése

az összes cella true vagy false értékre. Az `initialize()` metódusban ezután a táblázat és a cellákban lévő gombok beállítása következik az előző fejezetben ismertetett megoldásokhoz hasonlóan. A felhasználónak lehetősége van a „+” szimbólummal ellátott gombbal vizsgát hozzáadni – 2.20. ábra – a listához. A hozzáadást és az inputok ellenőrzését az `AddExamEventWindowController` osztály végzi. A vizsga kezdő dátumának beállításával a befejező dátum ugyanarra a napra állítódik, – a kezdő óránál hasonlóan – ezzel is segítve a felhasználót. Ennek megvalósítása az adott `DatePicker` `valueProperty()`-jén keresztül történt, `addListener()` hozzáadással. Az `addListener()` paraméterként egy lambda kifejezést kap. Rosszul megadott időpontok esetén a `ComboBox`-ok `border color` tulajdonsága `#ff0000`-re, azaz vörösre állítódik. A sikeresen hozzáadott vizsga időrendben a táblázat megfelelő helyére kerül és ha nem ütközik, aktív lesz. Az ablak jobb alsó sarkában található – szemétkgyűjtő szimbólummal ellátott – gombbal törölhető az adott kurzushoz tartozó összes vizsgaidőpont.

2.20. ábra. Vizsgaidőpont hozzáadása

Rákattintva megjelenik egy modális – 2.21. ábra – ablak, melyen a felhasználó kiválaszthatja, hogy valóban törölni szeretné-e az összes vizsgaidőpontot.















2.21. ábra. Modális ablak

A modális ablak nem zárható be, a felhasználót addig nem engedi tovább lépni, amíg meg nem erősíti – vagy épp el nem veti – szándékát. Jóváhagyás után visszakérül a vizsgákat kezelő ablakba.

#### 2.3.4. Általános események kezelése

A főmenüben az „EVENTS” feliratú gombot választva a felhasználó az általános eseményeit tartalmazó ablakba – 2.22. ábra – kerül, melyen táblázatos formában láthatja a rögzített eseményei nevét, kezdő és befejező időpontját. A táblázat tartalmaz még egy „Active” oszlopot. Az aktívra állított események megjelennek a naptárban. A táblázat utolsó oszlopának celláiban egy-egy gomb – személygyűjtő ikonnal ellátott – található, melyre kattintva törölhető az adott esemény. Az esetleges módosítások menthetők. Amennyiben a felhasználó egymással időben ütköző eseményeket állít aktívra, akkor mentéskor az ütköző eseményekhez tartozó rekordok sárga színnel kerülnek kiemelésre és egy modális ablakban figyelmeztetés jelenik meg az ütközés tényéről. A hiba az adott esemény „Active” mezője értékének false-ra – a CheckBox-ra történő kattintással – történő változtatásával javítható. Mivel a felhasználó általános napi tevékenységei – étkezések, pihenés – nincsenek időpontokhoz rögzítve, – viszont összeadjuk adott –, ezért előfordulhat, hogy, adott napra a felhasználó időt tekintve túl sok eseményt vesz fel. Mentéskor a probléma tényéről figyelmeztetés jelenik meg modális ablakban, ezenkívül pedig az ezen napok eseményeit tartalmazó rekordok vörös színnel jelennek meg. Az eseményekhez továbbiak hozzáadhatók. A hozzáadás ablakban az események ellenőrzésre kerülnek egyediség és időbeli ütközés szempontjából. A jobb alsó sarokban található gombbal az összes esemény törölhető. A felhasználó a módosítások mentése nélkül is visszaléphet a főmenübe, a vissza nyilat tartalmazó gombra történő kattintással.



General events				
Name of event	Begin of event	End of event	Active	Delete
Running	2023.05.08. 17:00	2023.05.08. 18:00	<input checked="" type="checkbox"/>	 Delete
Reading	2023.05.08. 18:00	2023.05.08. 19:00	<input checked="" type="checkbox"/>	 Delete
Meeting	2023.05.08. 19:30	2023.05.08. 20:00	<input checked="" type="checkbox"/>	 Delete
Running	2023.05.09. 17:00	2023.05.09. 18:00	<input type="checkbox"/>	 Delete
Reading	2023.05.09. 18:00	2023.05.09. 19:00	<input type="checkbox"/>	 Delete
Running	2023.05.10. 17:00	2023.05.10. 18:00	<input checked="" type="checkbox"/>	 Delete
Reading	2023.05.10. 18:00	2023.05.10. 19:00	<input checked="" type="checkbox"/>	 Delete
Meeting	2023.05.11. 13:00	2023.05.11. 14:00	<input checked="" type="checkbox"/>	 Delete
Running	2023.05.11. 17:00	2023.05.11. 18:00	<input type="checkbox"/>	 Delete
Reading	2023.05.11. 18:00	2023.05.11. 19:00	<input checked="" type="checkbox"/>	 Delete
Running	2023.05.12. 17:00	2023.05.12. 18:00	<input type="checkbox"/>	 Delete
Reading	2023.05.12. 18:00	2023.05.12. 19:00	<input checked="" type="checkbox"/>	 Delete

2.22. ábra. Általános események

Az általános eseményeket tartalmazó ablak vezérléséért a `GeneralEventsWindowController` osztály felel. Az osztályban létrehozásra kerültek a feltöltendő táblázat – `TableView<Event> generalEventsTableView` – oszlopai, melyek két generikus paraméterrel rendelkeznek, például az esemény nevét tartalmazó oszlop esetében:

```
@FXML
private TableColumn<Event, LocalDateTime> beginOfEventColumn;
```

Az `initialize()` módszerban először példányosításra kerül az `EventDAO` osztály. Ezután az események tárolására létrehozott generikus lista feltöltésre kerül az `EventDAO` `findAllEvents()` módszere segítségével. Ez a módszer két paramétert vár, első a felhasználó neve, második pedig az esemény típusa, mely jelen esetben általános esemény. Ennek beállítása az `Event` osztály `EventType` enum-jának segítségével történik. Miután a lista feltöltődött a felhasználó általános eseményeivel, a gombokhoz tartozó ikonok és `ToolTip`-ek beállítása következik a korábbiakban ismertetett módon. A táblázat elemei kezdő időpont szerint sorba rendezve jelennek meg. Ez megvalósítható például a következő `Comparator` használatával:

```
Comparator<Event> comparator = Comparator.comparing(Event::getBeginOfEvent);
this.events.sort(comparator);
```



Az eseményeket tartalmazó táblázat egyes rekordjainak „Active” mezője egy `CheckBox`-ot tartalmaz, ennek értéke változtatható. Azért, hogy az adatbázisban minden egyes módosításkor ne legyen szükséges felülni az utoljára felvitt értéket, létrehoztam egy eseményeket – `Event` – tartalmazó generikus listát, melyhez az események egy `foreach` ciklussal kerültek hozzáadásra. Ezt az események `TableView`-hoz történő rendelésre követi, a felhasználó általános eseményeit tartalmazó events lista felhasználásával:

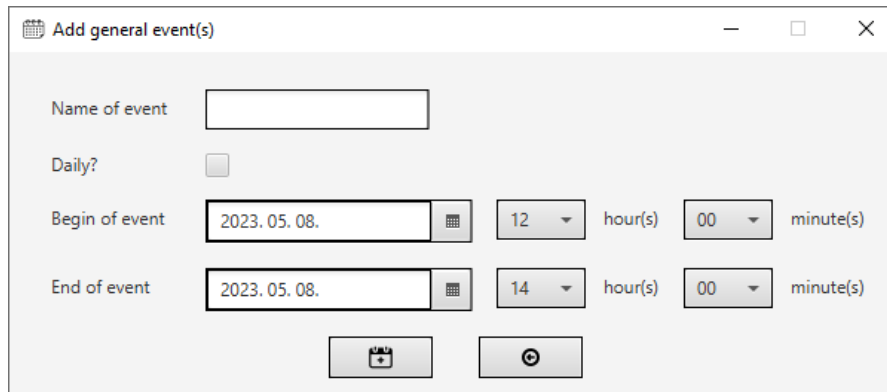
```
this.generalEventsTableView.getItems().setAll(this.events);
```

Az általános eseményeket tartalmazó ablak tartalmaz egy „Check All” feliratú `CheckBox`-ot is. Ennek bejelölése annak vizsgálata alapján történik, hogy a listában vannak-e elemek és azok aktívak-e. Ha a táblázatban nincsenek elemek vagy pedig nem az összes elem van aktívra állítva, akkor a `CheckBox` nem lesz bejelölve. Ha az összes esemény aktívra van állítva, akkor pedig a `CheckBox` be lesz jelölve. Az `initialize()` metódus további részében az oszlopok `setCellValueFactory()` metódusaik segítségével kerülnek beállításra a már ismertetett módon. Az ablak tartalmaz mentés, hozzáadás, vissza és összes elem törlése gombokat. A mentés gombra történő kattintással kiváltott események a mentés gomb `saveButtonClicked()` metódusában kerültek kidolgozásra. A metódus első utasításával az esetleges kijelölések törlésre kerülnek. Ezután két privát segédmetódus fut le. Az első ellenőrzi, hogy vannak-e olyan események aktívra állítva, amelyek ütköznek. A második pedig azt ellenőrzi, hogy az adott napi események beleférnek-e 24 órába. Ezt az ütközés vizsgálata nem teszi szükségtelenné, ugyanis a felhasználó mindennapi foglalatosságainak csak az időtartama kerül rögzítésre, a pontos ideje nem. Amennyiben ütköző események találhatók a táblázatban, akkor megjelenik egy modális ablak a "Several events collide. Marked in yellow." `String`-gel, a táblázatban pedig sárga színnel jelölésre kerülnek az ütköző események rekordjai. A kijelölés beállításai a `checkFullCollision()` privát metódusban történnek a:

```
this.generalEventsTableView.setStyle("-fx-selection-bar-non-focused: yellow");
this.generalEventsTableView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

utasításokkal, az ütköző rekordok kijelölése pedig a `TableView` `getSelectionModel.select()` metódusával, ahol paraméter a sor indexe. Abban az esetben, ha az adott nap eseményeinek időtartama több mint 24 óra, akkor megjelenik egy modális ablak a "Too many events for a given day. Marked in red." `String`-gel, az adott nap(ok) pedig vörössel jelölve jelennek meg a táblázatban. Az ellenőrzést és kijelölést végző – `checkFreeTime()` – metódus működése hasonló az ütközést vizsgáló metóduséhoz. Az

ütközéseket és az időbeli limit okozta problémát kezelni vagy események törlésével vagy pedig az aktív mező értékének false-ra történő átállításával lehetséges. Bizonytalan eseményeket nem érdemes törölni, elegendő kivenni a pipát a hozzájuk tartozó `CheckBox`-ból. Sikeres mentés után a felhasználó visszakérül a főmenübe.



2.23. ábra. Általános esemény hozzáadása

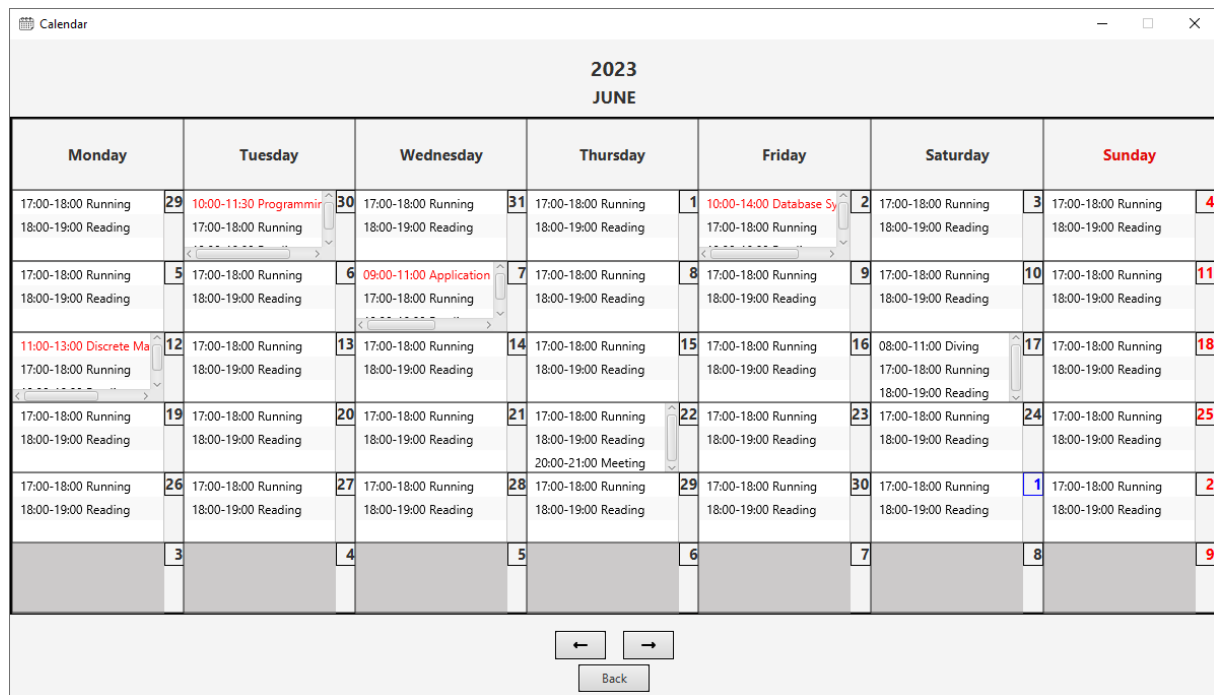
Eseményt – 2.23. ábra – hozzáadni a „+” szimbólumot tartalmazó gombra történő kattintással lehetséges. Az inputok ellenőrzését és adatbázishoz történő hozzáadásuknak vezérlését az `AddGeneralEventController` osztály kezeli. Az osztály felépítése és működése hasonló az `AddExamEventController` osztályéhoz, egy opcióban viszont eltér. Az általános eseményeket hozzáadása ablak tartalmaz egy „Daily?” nevű `CheckBox`-ot. Ennek bejelölésével a felhasználónak a naponta ismétlődő eseményeit nem szükséges többször felvinnie. A `CheckBox` bejelölésekor a `DatePicker`-ek `setDisable()` metódusa `true` paraméterrel kerül beállításra, így nem lesznek szerkeszthetők, s csak az esemény kezdő és a befejező időpontját lehetséges megadni. A megadott időpontok ellenőrzése – egyediségre, ütközésre, időtartamra, vizsgaidőszakon kívüliségre – után az `EVENTS` táblába bekerülnek az ismétlődő események. Amennyiben az esemény rögzítésének napja a vizsgaidőszakot megelőző 1 hét előtt van – vagy épp azon a napon –, akkor az esemény a vizsgaidőszakot megelőző 1 hét első napjától kezdve minden naphoz hozzárendelődik az utóvizsga időszak utolsó napjáig, amennyiben utána, akkor pedig az aktuális naptól kezdve az utóvizsga időszak utolsó napjáig. A „Daily?” `CheckBox` bejelölése nélkül egyszeri esemény felvitele történik kezdő és befejező dátum megadásával. A hozzáadás ikonot tartalmazó gombra történő kattintáskor ellenőrzésre kerül a megadott input. Hibás név esetén üzenet jelenik meg a név megadására szolgáló `TextField`-ben.

Vizsgaidőszakon kívül eső eseményt felvinni nem lehet, erre vonatkozó figyelmeztetés modális ablakban jelenik meg.

### 2.3.5. Naptár

*A főmenüben a „CALENDAR” feliratú gombra kattintva a felhasználó átkerül a vizsganaptárt – 2.24. ábra – tartalmazó ablakba, amennyiben van felvéve legalább egy kurzusa, az nem teljesített és vannak hozzátartozó aktív vizsgaidőpontok is és azok közül legalább egy – a `SortingAlgorithm` osztály sorrendgeneráló metódusában ismertetett feltételek teljesülése mellett – felvihető a naptárba. Ellenkező esetben hibaüzenetet kap egy modális ablakban. A naptár fölött helyezkedik el az év és a hónap. A naptár alatti két iránynyilat tartalmazó gombra történő kattintással a felhasználó az előző és a következő hónapra léphet. A naptár egységesen 6 hetet jelenít meg. Amennyiben a felhasználó a vizsgaidőszak kezdő dátuma előtt, vagy a kezdő dátumhoz tartozó hónapban van, akkor a vizsgaidőszak kezdetének megfelelő hónap jelenik meg, utána pedig az aktuális hónap. A felhasználó napi bontásban láthatja a felvett vizsgákat, s egyéb eseményeket. A vizsgák vörös színnel jelölve látszanak. Az adott nap eseményeit tartalmazó listára kattintva egy új ablakban jelennek meg az adott napi események. Az üres – vagy vizsgaidőszakon kívülre eső – napok szürke árnyalattal jelennek meg.*

A naptár és a felhasználói interakciók kezelése a `CalendarWindowController` osztályban történik. Az osztály `initialize()` metódusában először a DAO osztályok példányosítása történik meg. A felhasználó kurzusai a `SubjectDAO` `findAllSubjects()` metódusának meghívásával belekerülnek a felhasználó kurzusait tároló `subjectsList`-be. A metódus egy paramétert vár, a felhasználó nevét, mely az `App.user.getUsername()` getter-rel kérdezhető le. Ezután – hasonló módon – a felhasználó összes eseménye hozzáadásra kerül az `eventsList`-hez az `EventDAO` `findAllEvents()` metódusa segítségével. Egyetlen paraméter ebben az esetben is a felhasználó neve. Ezek után az adatbázis-kapcsolat a DAO-k `close()` metódusának meghívásával záródik. A naptár `ListView`-kat tartalmaz `GridPane`-en `fx:id`-k megadása nélkül. A `GridPane` celláiban egy-egy `HBox` található, melyen egy `ListView` és egy `Label` helyezkedik el `VBox`-okon. Az ezek feltöltéséhez szükséges vizsgasorrendet és az



2.24. ábra. Naptár

aktív események közös listáját a `SortingAlgorithm` osztály `proposedExamDatesWithGeneralEvents()` metódusa készíti el az 2.2.4. fejezetben ismertetett módon. A lista létrehozásához először szükséges létrehozni egy `SortingAlgorithm` példányt:

```
this.sortingAlgorithm = new SortingAlgorithm(App.user, this.subjectList, this.eventList);
```

mely paraméterként megkapja a felhasználó kurzusait és eseményeit – vizsgák és általános események is – tartalmazó listákat. A `proposedExamDatesWithGeneralEvents()` metódus egy generikus – `Event` – listával tér vissza, mely tárolásra kerül az `eventsForTheCalendar` generikus listába:

```
this.eventsForTheCalendar = this.sortingAlgorithm.proposedExamDatesWithGeneralEvents();
```

Az `initialize()` metódus utolsó utasításával meghívódik a `calculateDate()` privát metódus. A metódus törzsében először az naptár ablak felső részén látható év és dátum érték kerül – `Label`-ökon – beállításra. Ennek meghatározása a felhasználó által megadott vizsgaidőszak kezdődátuma alapján történik. Ezután egy `for` ciklus a `GridPane` 7. cellájától – mely a `STARTING_CELL` konstansban rögzített – végigiterál az utolsó – 49-es, ez is rögzített a `NUMBER_OF_CELLS` adattagban – celláig. Azért a 7. cellától kezdve, mert `GridPane` első sora a hét napjait tartalmazza. A `for` ciklusban a `GridPane` celláiban található `ListView`-khöz az `eventsForTheCalendar` lista adott naphoz tartozó eseményei adódnak hozzá. Az `eventsForTheCalendar` bejárása `foreach` ciklussal történik. Mivel az egyes

ListView-k és – a napokat tartalmazó – Label-ök nem rendelkeznek fx:id-val, ezért azonosításuk a Node-ok `getChildren()` metódusával történik, például a ListView-k esetében a következőképpen:

```
listView = (ListView<Text>) ((VBox) ((HBox) this.calendarGridPane
    .getChildren()
    .get(i))
    .getChildren()
    .get(0))
    .getChildren()
    .get(0);
```

mely az ismétlődő elrendezés miatt minden esetben az indexszel megadott GridPane cellájának ListView-ját adja vissza. Az iteráció során a ListView-khoz beállításra kerül a `setOnMouseClicked()` metódus. Névtelen osztályában a `handle()` metódus felüldefiniálásra kerül, a ListView-kra történő kattintáskor modális ablakban megjeleníti az adott nap eseményeit, amennyiben az adott ListView nem üres. A naptár léptetésére szolgáló gombokhoz rendelt eseménykezelők kattintás hatására az aktuális – éppen megtekintett – hónap értékét növelik vagy csökkentik a nyíl ikonok iránya szerint. Az osztály tartalmaz még egy `setDate()` és egy `clearAll()` privát metódust. Előbbi a GridPane felett látható évet és hónapot állítja be, utóbbi pedig végigiterál az összes ListView-n és törli azokat. Emellett a napot tartalmazó Label-öket visszaállítja formázatlanra a `setStyle()` metódus használatával. Erre azért van szükség, mert a naptárban látható következő hónap első napja kék – az `-fx-border-color` és az `-fx-text-fill` – színnel van jelölve, s enélkül léptetéskor az előző hónap első napját tartalmazó Label is kék színű maradna.

### 2.3.6. Regisztráció

*A nyitóablak középső – regisztráció szimbólumot tartalmazó – gombjára kattintva a felhasználó átkerül a – 2.25. ábra – regisztrációs ablakba. A regisztrációhoz szükséges megadnia választott felhasználónevét, jelszavát, maximális vizsgalehetőségeinek számát – szemeszterenként és összesen vett –, a vizsgaidőszakának és utóvizsgaidőszakának kezdő és befejező dátumát. Ezenkívül megadhatja még a napi tanulási idejét és a napi általános tevékenységeinek idejét – például étkezések, pihenés – órában. A felhasználónév és a jelszó*

kivételével a többi értéket előre definiált halmazból választhatja ki `ComboBox`-ok és `DatePicker`-ek használatával. Hibás input esetén üzenet jelenik meg a hibát tartalmazó `TextField`-ben. Figyelmeztet formátum hibára, nem egyező jelszavakra, létező felhasználóra. Hibásan megadott dátumok esetén – például a záró dátum előbb van, mint a nyitó – a `DatePicker` mellett felirat jelenik meg. Sikeres mentés után a felhasználó visszakérül a nyitóablakba.

2.25. ábra. Regisztrációs ablak

A regisztráció során megadott inputok ellenőrzése, adatbázisba történő felvitele a `RegisterWindowController` osztály segítségével történik. Az osztály felüldefiniált `initialize()` metódusában – az `UserDAO` példányosítása után – feltöltésre kerül az `users` generikus – `User` – paraméterű lista az `UserDAO.findAllUsers()` metódushívással, vagyis az `users` listába bekerül az összes felhasználó. Ezután a mentés és a vissza gombokhoz ikon és `Tooltip` beállítása következik. `Tooltip` kerül a felhasználónevet tartalmazó `TextField`-hez és a jelszót tartalmazó `PasswordField`-hez is. A felhasználónévhez megadott `Tooltip` arra figyelmeztet, hogy a megadott felhasználónévnek 6 és 15 alfanumerikus karakter között kell lennie. A jelszónak pedig 7 és 20 karakter között kell lennie, ezenfelül legalább 1 darab kisbetűt, 1 darab nagybetűt, 1 darab számot és 1 darab speciális karaktert kell tartalmaznia. A `Tooltip`-ek megadása után a

ComboBox-ok és a DatePicker-ek inicializálása következik. A jobb átláthatóság végett ezeket kiszerveztem egy privát visszatérési érték nélküli `setDefault()` paraméter nélküli metódusba. A ComboBox-ok beállítása az „utility” csomagban található `ComboBoxInitializer` osztály `setComboBox()` statikus függvényen keresztül történik, az alább látható módon:

```
setComboBox(maxExamComboBox, 3, 9, 3, Type.INTEGER.getValue());
```

A függvény első paraméterként megkapja a beállítandó ComboBox-ot, második paraméterként a lenyíló lista első elemét, harmadik paraméterként a lista utolsó elemét, negyedik paraméterként az alapértelmezetten kiválasztott elem sorszámát – a számozás 1-től indul –, ötödik eleméként pedig, a ComboBox generikus típusát. A lista első és utolsó eleme között az elemek – Integer-ek – eggyel nőnek. A dátumok megadására szolgáló DatePicker-eknél a dátum napi bontásban látható. Amennyiben a felhasználó módosítja a vizsgaidőszak kezdetének, végének vagy pedig az utóvizsgaidőszak kezdetének megadására szolgáló DatePicker-ben megadott dátumot, akkor az `initialize()` metódus végén létrehozott eseménykezelők módosítják a megadott dátumhoz tartozó DatePicker-t követő DatePicker(-ek) dátumát. Az eseménykezelő hozzáadása az adott DatePicker `valueProperty().addListener()` metódusával történt, paramétere pedig egy lambda kifejezés, mely a DatePicker-t követő DatePicker-ben beállított dátum értéke 1 nappal megnövelve. Ez a felhasználót hivatott segíteni azzal, hogy záró dátum ne lehessen előbb – és ne is egyezhessen – nyitó dátumnál. Természetesen az eseményfigyelő segítségével beállított értéket a felhasználó átállíthatja, viszont helytelen beállítás esetén mentéskor hibaüzenet fog megjelenni a DatePicker-ek mellett elhelyezett – alapértelmezetten nem látható – Label-ökben. A felhasználó által bevitt adatok ellenőrzése a mentés gombra történő kattintáskor a `RegisterWindowController` osztály `registerButtonClicked()` metódusában történik. Azért, hogy az USERS táblába ne kerülhessenek be hibát – például dátumoknál a befejező dátum előbb van, mint a kezdő és emiatt az ezeket felhasználó metódusok hibás értékkel térnek vissza – tartalmazó rekordok, a felhasználó által megadott inputok ellenőrzésre kerülnek a privát `checkUserExist()`, `checkDate()`, `checkExams()` metódusok által. Amennyiben a felhasználó nem létezik, ellenőrzésre kerülnek a jelszavak is. Az ellenőrzés reguláris kifejezésekkel történik a `Pattern.matches()` metódus használatával, melynek első paramétere – például a felhasználónév ellenőrzése esetén – az alábbi regex kifejezés:



```
String userRegex = "[a-zA-Z0-9]{6,15}$";
```

A második paraméter pedig a felhasználónevet tartalmazó `TextField` `getText()` metódusa segítségével `String`-ként elkért felhasználónév. Amennyiben a regex kifejezés illeszthető a megadott `String`-re, akkor a `Pattern.matches()` metódus `true`-val tér vissza, ellenkező esetben pedig `false`-szal. Hiba esetén a hibát tartalmazó `TextField`-ek `border color` tulajdonsága átállításra kerül:

```
this.usernameTextField.setStyle("-fx-text-box-border: #ff0000;");
```

Így a – példában látható – felhasználónevet tartalmazó `TextField` szegélye vörös színű lesz. Emellett a `TextField` `setPromptText()` metódusa segítségével beállításra kerülnek a különféle hibákra figyelmeztető – például „empty username” – megjegyzések is. Helyes input esetén a szegély zöld színű lesz. A szegély színének megváltoztatása a `DatePicker`-ekre is vonatkozik. A dátumok ellenőrzése a `checkDate()` metódussal történik. Hiba esetén „ERROR” `String` jelenik meg a `DatePicker`-ek mellett elhelyezett – alapértelmezetten üres, tehát nem látható – `Label`-ökben, helyes dátum esetén pedig „OK” `String`. Ellenőrzésre kerül még a vizsgák számának megadására szolgáló két `ComboBox` tartalma is a `checkExam()` privát metódusok segítségével. Ebben az esetben akkor hibás a megadott érték, ha a szemeszterenként vett vizsgák száma nagyobb az összes vizsgalehetőségek számánál. Ekkor a szegélyek színe az ismertetett módon vörös színűre változik. Amennyiben a megadott adatok helyesek – vagyis a felhasználónév formátuma megfelelő, nem létezik még az `USERS` táblában, a jelszó és a megerősítése megegyezik, s megfelelnek a megadott kritériumoknak, a dátumok egymást követő értékek és nem a múltban vannak – akkor példányosításra kerül egy `User` objektum paraméter nélküli konstruktorával, majd pedig a beállítása következik az `User` osztály setter-ein keresztül, a 2.26. ábrán látható módon.

```
User user = new User();
user.setUsername(this.usernameTextField.getText());
user.setPassword(passwordHasher(this.passwordField.getText()));
user.setMaxPossibleExam(this.maxExamComboBox.getValue());
user.setMaxPossibleExamPerSemester(this.maxSemComboBox.getValue());
user.setBeginOfExamPeriod(this.beginOfExamPeriodDatePicker.getValue());
user.setEndOfExamPeriod(this.endOfExamPeriodDatePicker.getValue());
user.setBeginOfSupplementaryExamPeriod(this.beginOfSuppExamPeriodDatePicker.getValue());
user.setEndOfSupplementaryExamPeriod(this.endOfSuppExamPeriodDatePicker.getValue());
user.setDurationOfDailyActivities(this.durationOfDailyActivitiesComboBox.getValue());
user.setLearningTimePerDay(this.learningTimePerDayComboBox.getValue());
user.setRestDayOnSaturday(false);
user.setRestDayOnSunday(true);
```

2.26. ábra. *User* példányosítása és adattagjainak beállítása



A jelszó beállítása – és tárolása is az adatbázisban – hash-elve történik. A 2.26. ábrán látható utolsó két setter alapértelmezés szerint állítja be az adattagokat, ezeket a felhasználó szüksége esetén a regisztráció után, a tanulási beállításokra vonatkozó ablakban állíthatja át. Az user példány beállítása után az `UserDAO insertUser()` metódusának – melynek egyetlen paramétere a beállított user példány – meghívásával az USERS táblába bekerül egy új rekord az új felhasználó adataival. A metódus végén az `UserDAO close()` metódusával záródik az adatbázis-kapcsolat, majd pedig megjelenik egy, a sikeres regisztrációra figyelmeztető modális ablak.

### 2.3.7. Beállítások

*A felhasználó bejelentkezés után – a felhasználóneve kivételével – átállíthatja a regisztrációkor – 2.27. ábra – megadott adatait. A jelszó formátumára vonatkozó megkötések azonosak a regisztrációkor megadottakkal, vagyis 7 és 15 karakter között kell lennie, kötelezően tartalmaznia kell legalább egy kis és egy nagybetűt, legalább egy számot és legalább egy speciális karaktert. Amennyiben a felhasználó átállítja a maximális vizsgaszámát és van olyan kurzusa, amelynél ettől nagyobb érték van megadva, akkor mentéskor az adott kurzus – vagy kurzusok – vizsgaszámait a megadott értékre maximalizálódnak. Amennyiben a felhasználó átállítja a vizsgaidőszak nyitó és záró dátumait és az újonnan megadott dátumok az eredetileg megadott intervallumon belül helyezkednek el, akkor a kieső napok eseményei törölődnek. Erről a felhasználó értesítést kap egy modális ablakban megjelenő üzenet formájában. Amennyiben tágabb intervallumot ad meg, nem történik változás. Amennyiben csak a jelszavát szeretné megváltoztatni, elegendő kitölteni a jelszó és a jelszó megerősítő mezőket, a többi mező a regisztrációkor megadott – vagy utoljára beállított – értékekkel fog feltöltődni, mentéskor nem lesz változás bennük. A dátumok átállítására is van lehetősége jelszaváltoztatás nélkül, viszont bármely beállítás mentéséhez szükséges megadnia a jelszavát.*

2.27. ábra. Beállítások

A „Settings” ablakban megadott inputokat a `SettingsWindowController` osztály kezeli. Az ablak felépítése és működése hasonló a regisztrációs ablakéhoz, viszont a felhasználónév – itt kitakart – adott, ezenkívül pedig az ablak jobb alsó sarkában elhelyezésre kerül egy – szemétygyűjtő ikont tartalmazó – törlés gomb, melyre rákattintva megjelenik a 2.28. ábrán látható modális ablak:

2.28. ábra. Törlés megerősítése

Helyes jelszó megadása esetén a pipa ikont tartalmazó gombra kattintva a felhasználóhoz tartozó rekord törlődik az `USERS` táblából, ezenkívül az összes hozzá tartozó kurzus a `SUBJECTS` és az összes általános eseménye, vizsgája is az `EVENTS` táblából. Hibás jelszó esetén meghívódik a `PasswordField setPromptText()` metódusa és beállításra kerül a paraméterként átadott „wrong password” `String`.

### 3. ÖSSZEFOGLALÁS

Megvalósítottam egy Java asztali alkalmazást, mely egy távoli MySQL adatbázissal kommunikál. A grafikus felhasználói felületet a JavaFX könyvtár használatával hoztam létre, build keretrendszerként pedig Maven-t használtam. A projekt multimodulos felépítésű, szerkezetének kialakításában az MVC tervezési mintát követtem. Szakdolgozatomat a felhasznált technológiák bemutatásával kezdtem, többek között kitértem a tesztelésre és a kódminőség ellenőrzésre is. Részleteiben ismertettem a `Model` réteg osztályait, adattagjaik szerepét, módszusaik működését. A `Bean` osztályok esetében bemutattam az adott osztály példányainak tárolásához szükséges táblát létrehozó SQL-utasítást. Ismertettem a `DAO` osztályok szerepét, kitértem biztonsági kérdésekre is. A GUI-val párhuzamosan haladva bemutattam a `Controller` osztályok felépítését, módszusaik működését.

## ***Irodalomjegyzék***

1. M. Eichberg: Introduction to Design Patterns  
[http://stg-tud.github.io/eise/WS15-SE-14-Design\\_Patterns-Introduction.pdf](http://stg-tud.github.io/eise/WS15-SE-14-Design_Patterns-Introduction.pdf) (letöltve: 2022.12.01.)
2. H. Schildt: Java: The Complete Reference. Twelfth Edition. McGraw Hill. 2022. pp. 6-10.
3. TIOBE Index: Very Long Term History  
<https://www.tiobe.com/tiobe-index/> (letöltve: 2022.12.01.)
4. M. Pawlan: What is JavaFX?  
<https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm> (letöltve: 2022.12.01.)
5. JavaFX Introduction  
[https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1\\_intro.html](https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html) (letöltve: 2022.12.01.)
6. A. Agarwal, A. Baimagambetov, C. Walker, C. Nahr, J. Giles, G. Grunwald:  
JavaFX Documentation Project  
<https://fxdocs.github.io/docs/html5/> (letöltve: 2022.12.01.)
7. 1.1 Getting Started - About Version Control  
<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control> (letöltve: 2022.12.01.)
8. Java 8 – Collection.removeIf method tutorial with examples  
<https://www.javabrahman.com/java-8/java-8-collection-removeif-method-tutorial-with-examples/>  
(letöltve: 2022.12.01.)
9. Data Access Object Also Known As DAO  
<https://www.oracle.com/java/technologies/data-access-object.html> (letöltve: 2022.12.01.)
10. Class Application  
<https://docs.oracle.com/javase/8/javafx/api/javafx/application/Application.html> (letöltve: 2022.12.01.)

## ***Nyilatkozat***

Alulírott Nyul Péter, programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

2022. november 28.

Nyul Péter

### ***Köszönetnyilvánítás***

Szeretném kifejezni köszönetemet Dr. Siket István adjunktus úrnak a szakdolgozatom elkészítésében nyújtott segítségéért.