

# Lucrare Szabo

*by* Ioana Cristina PLAJER

---

**Submission date:** 18-May-2022 06:43PM (UTC+0300)

**Submission ID:** 1836102833

**File name:** 1200\_Ioana\_Cristina\_PLAJER\_Lucrare\_Szabo\_150733\_559899585.pdf (682.32K)

**Word count:** 12793

**Character count:** 76103

## ÎNTRODUCERE

Realizarea orarului este unul dintre cele mai grele sarcini care apar în cadrul organizării funcționării unei unități de învățământ. De obicei orarul se realizează la începutul anului de învățământ de către unul sau mai mulți membri ai corpului didactic și presupune un efort destul de mare ca rezultatul să fie satisfăcător pentru toți cei implicați, adică cadre didactice și studenți.

La realizarea orarului trebuie avute în vedere foarte multe criterii ca de exemplu: un profesor nu poate avea ore concomitent cu mai multe grupe de elevi, este de preferat ca nici profesorii nici studenții să nu aibă mai multe ferestre pe zi, ideal ar fi ca să nu existe ferestre dar de obicei este foarte greu să obținem un astfel de rezultat. În cazul școlilor ferestrele pentru elevi sunt de obicei evitate pentru că elevii fiind minori vor trebui supravegheate oricum de cineva ( învățător, profesor, educator etc. ).

Desigur cel mai simplu este orarul claselor primare unde nu trebuie corelate decât câțiva profesori și învățătoarea/invățătorul. În acest caz criteriul de bază la realizarea orarului ar fi aranjarea orelor într-o manieră de a nu acumula multe ore "grele" într-o zi rămânând care să conducă la o extenuare a elevilor. De obicei orele mai "grele" se vor altera cu altele mai relaxante pentru a păstra cumva atenția și concentrarea copiilor.

După clasele primare organizarea învățământului devine mai complicată fiindcă fiecare materie este predată de profesori specializați. Realizarea unui "orar bun" devine mai complicată pentru că trebuie corelate mai mulți factori - profesori, grupe de elevi, săli/laboratoare de specialitate etc. De obicei în școli elevii sunt legați de sălile de clasă, adică fiecare clasă are sala ei, și profesorii se deplasează la ore în sălile claselor. Fac excepție orele de educație fizică, eventualele orele de laborator ( informatică, chimie, fizică etc. )

Situația devine și mai complicată în cazul învățământului superior adică în cazul studiilor superioare, masterat, doctorat etc. Aici în afară de problemele care apar și în cazul învățământului școlar apar anumite elemente care pot complica<sup>[34]</sup> și mai mult viața oraristului. Nu mai avem săli prestabilite pentru grupe ( clase ) și chiar dacă din anumite puncte de vedere există o libertate mai mare, de exemplu pot apărea ferestre și în orarul studenților lucru nepermis în învățământul preuniversitar, totuși realizarea unui "orar bun" este o provocare pentru orarist. Apar profesori de la alte facultăți, studenții trebuie să se deplaseze între diferite clădiri ale universității, pot să apară foarte multe restricții în orarul profesorilor, mai ales în cazul în care aceștia vin de la alte unități de învățământ sau dacă sunt doar profesori suplinitori etc.

Aplicația "Ajutorul oraristului" nu își propune să suplimească oraristul ci va încerca să ofere un sprijin eficace pentru realizarea orarului facultăților din cadrul Universității Transilvania din Brașov. Aplicația va citi datele necesare din fișierul excel a statelor de funcționi ai facultății în cauză și va genera activitățile care rezultă ( adică cursuri, seminarii, laboratoare, activități practice ) din fișier. Aceste activități vor putea fi mișcate, mutate de către orarist între diferite tabele, orare ale profesorilor, grupelor, anilor etc. Aplicația va semnala suprapunerile care apar pentru profesori sau grupe și nu va permite mutarea unei activități dacă profesorul sau grupa ( grupele ) în cauză au deja alte activități atribuite pentru perioada de timp respectivă. Ca finalitate aplicația va putea înscrie datele în fișierul de Orar al facultății.

Sper ca aplicația să ofere un real ajutor celor care vor fi însăcinați cu elaborarea orarului facultăților universității Transilvania, fiind o aplicație personalizată care se folosește de fișierele deja existente care conțin toate informațiile necesare pentru a genera activitățile curente care vor trebui să apară în orarul final al facultății. În această primă variantă aplicația va putea citi din fișierul statelor de funcțiuni orele profesorilor aparținând unei anumite facultăți și va genera activitățile acestora care se desfășoară în cadrul aceleiași facultăți, de exemplu pentru Facultatea de Matematică și Informatică va citi toate orele profesorilor din cadrul facultății care le au cu grupele acestei facultăți.

Făcând o cercetare în mediul online am descoperit că există multe aplicații de realizare de orar desigur nici unul personalizat. Aceste aplicații sunt de mai multe feluri, cele mai multe pe care le-am găsit sunt cele de editare a unui orar personal <sup>33</sup>. De fapt este un fel de calendar care trimite atenționări, remindere pentru titular dar de fapt nu au nici o legătură cu elaborarea unui orar ai unei unități de învățământ. Dacă ne referim strict la aplicații de elaborare automată a unui orar am găsit și aici câteva alternative cum ar fi

- asc Orare - <https://ascorare.ro/>
- docendo Zen - <https://zen.docendo.co/>
- GHC School timetable maker - <https://www.penalaria.com/en/RO>

Toate acestea oferă elaborarea automată a orarului și, desigur, și editarea manuală a acestuia însă chiar dacă aplicația noastră nu oferă elaborarea automată a orarului ci doar oferă suport celui care se ocupă de elaborarea orarului marele avantaj al aplicației Ajutorul Oraristului constă în faptul că aceasta fiind personalizat introducerea datelor este de fapt un proces automatizat și faptul că oraristul are libertate în alcătuirea orarelor profesorilor sau grupelor îi dă acestuia posibilitatea să țină cont de o multitudine de restricții, cerințe, doleanțe venite mai ales din partea profesorilor. Totodată există și anumite situații speciale care pot fi luate în calcul de către orarist în momentul în care elaborează orarul și chiar dacă toate aplicațiile enumerate mai sus acceptă condiționări în ceea ce privește elaborarea automată a orarului probabil că introducerea manuală a tuturor acestor restricții, condiții reprezintă un efort destul de mare și nici nu este garantată introducerea tuturor acestor condiționări care probabil că în cazul elaborării manuale a orarului cu ajutorul aplicației mele vor apărea pe parcursul elaborării și vor putea fi mult mai bine gestionate de către orarist.

Aplicația mea este concepută pentru a oferi asistență și nu pentru a elabora orarul în mod automat cum se întâmplă în cazul aplicațiilor enumerate dar, așa cum am mai precizat, sunt convins că poate oferi un real sprijin și sper că va putea fi chiar folosit de cei care vor elabora orarele facultăților de la Universitate. De fapt justificarea alegerii mele este acesta, speranța ca să creez ceva util care va fi folosit și va ajuta cumva. Această primă variantă nu este perfectă și sunt convins că se poate perfecționa și extinde funcționalitatea aplicației. În această privință îmi ofer sprijinul și în continuare dacă se va dori să se utilizeze aplicația în practică și desigur vă pun la dispoziție și codul dacă este de ajutor.

În capitolul I mă voi referi la tehnologiile folosite în elaborarea programului, mă voi referi la limbajul de programare Java și Maven, la platforma pentru interfață

grafică JavaFX, la librăriile de care mă folosesc pentru salvarea datelor ( JSON ), pentru citirea scrierea fișierelor Excel ( POI ) etc.

În capitolul II voi face o descriere a aplicației, a claselor folosite, metodelor implementate etc.

Capitolul III va reprezenta un manual de utilizare a aplicației.

## CAPITOLUL I.

În acest capitol mă voi opri asupra tehnologiilor, limbajelor de programare și bibliotecilor folosite pentru a asigura cele mai bune soluții pentru realizarea aplicației. Codul este scris 100% în Java 1.8, folosesc JavaFX pentru interfață grafică, Maven pentru bibliotecile GSON și Apache POI folosite pentru salvare/citire din baza de date proprie Json respectiv din fișierele Excel ale Universității. În continuare mă voi opri asupra fiecăreia dintre ele.

### I.1. JAVA

<sup>1</sup> Limbajul Java împreună cu mediul său de dezvoltare și execuție au fost proiectate pentru a rezolva o parte dintre problemele actuale ale programării. Proiectul Java a pornit cu scopul declarat de a dezvolta un software performant pentru aparatele electronice de larg consum. Aceste echipamente se definesc ca: mici, portabile, distribuite și lucrând în timp real. De la aceste aparate, ne-am obișnuit să cerem fiabilitate și ușurință în exploatare. Limbajul luat inițial în considerare a fost C++. Din păcate, atunci când s-a încercat crearea unui mediu de execuție care să respecte toate aceste condiții s-a observat că o serie de trăsături ale C++ sunt incompatibile cu necesitățile declarate. În principal, problema vine din faptul că C++ este prea complicat, folosește mult prea multe convenții și are încă prea multe elemente de definiție lăsate la latitudinea celor care scriu compilatoare pentru o platformă sau alta. În aceste condiții, firma Sun a pornit proiectarea unui nou limbaj de programare asemănător cu C++ dar mult mai flexibil, mai simplu și mai portabil. Așa s-a născut Java.

Părintele noului limbaj a fost James Gosling care vă este poate cunoscut ca autor al editorului emacs și al sistemului de ferestre grafice NeWS. Proiectul a început încă din 1990 dar Sun a făcut publică specificația noului limbaj abia în 1995 la SunWorld<sup>1</sup> în San Francisco.

În primul rând, Java încearcă să rămână un limbaj simplu de folosit chiar și de către programatorii neprofesioniști, programatori care doresc să se concentreze asupra aplicațiilor în principal și abia apoi asupra tehniciilor de implementare a acestora. Această trăsătură poate fi considerată ca o reacție directă la complexitatea considerabilă a limbajului C++.

Au fost îndepărtațe din Java aspectele cele mai derutante din C++ precum supraîncărcarea operatorilor și moștenirea multiplă. A fost introdus un colector automat de gunoaie care să rezolve problema dealocării memoriei în mod uniform, fără intervenția programatorului. Colectorul de gunoaie nu este o trăsătură nouă, dar implementarea acestuia în Java este făcută inteligent și eficient folosind un fir separat de execuție, pentru că Java are încorporate facilități de execuție pe mai multe fire de execuție. Astfel, colectarea gunoaielor se face de obicei în timp ce un alt fir așteaptă o operație de intrare- ieșire sau pe un semafor.

Limbajul Java este independent de arhitectura calculatorului pe care lucrează și foarte portabil. În loc să genereze cod nativ pentru o platformă sau alta, compilatorul Java generează o secvență de instrucțiuni ale unei mașini virtuale Java. Execuția aplicațiilor Java este interpretată. Singura parte din mediul de execuție Java care trebuie portată de pe o arhitectură pe alta este mediul de execuție cuprinzând

interpreterul și o parte din bibliotecile standard care depind de sistem. În acest fel, aplicații Java compilate pe o arhitectură SPARC de exemplu, pot fi rulate fără recomplilare pe un sistem bazat pe procesoare Intel.

Una dintre principalele probleme ale limbajelor interpretate este viteza de execuție, considerabil scăzută față de cea a limbajelor compilate. Dacă nu vă mulțumește viteza de execuție a unei astfel de aplicații, puteți cere mediului de execuție Java să genereze automat, plecând de la codul mașinii virtuale, codul specific mașinii pe care lucrează, obținându-se astfel un executabil nativ care poate rula la viteză maximă. De obicei însă, în Java se compilează doar acele părți ale programului mari consumatoare de timp, restul rămânând interpretate pentru a nu se pierde flexibilitatea. Mediul de execuție însuși este scris în C respectând standardele POSIX, ceea ce îl face extrem de portabil.

Interpreterul Java este gândit să lucreze pe mașini mici, precum ar fi procesoarele cu care sunt dotate aparatele casnice. Interpreterul plus bibliotecile standard cu legare dinamică nu depășesc 300 Kocteți. Chiar împreună cu interfața grafică totul rămâne mult sub 1 Moctet, exact ca-n vremurile bune.

Limbajul Java este orientat obiect. Cu el se pot crea clase de obiecte și instanțe ale acestora, se pot încapsula informațiile, se pot moșteni variabilele și metodele de la o clasă la alta, etc. Singura trăsătură specifică limbajelor orientate obiect care lipsește este moștenirea multiplă, dar pentru a suplini această lipsă, Java oferă o facilitate mai simplă, numită interfață, care permite definirea unui anumit comportament pentru o clasă de obiecte, altul decât cel definit de clasa de bază. În Java orice element este un obiect, în afară de datele primare. Din Java lipsesc funcțiile și variabilele globale. Ne rămân desigur metodele și variabilele statice ale claselor.

Java este distribuit, având implementate biblioteci pentru lucru în rețea care ne oferă TCP/IP, URL și încărcarea resurselor din rețea. Aplicațiile Java pot accesa foarte ușor rețea, folosindu-se de apelurile către un set standard de clase.

Java este robust. În Java legarea funcțiilor se face în timpul execuției și informațiile de compilare sunt disponibile până în momentul rulării aplicației. Aceste mod de lucru face ca sistemul să poată determina în orice moment neconcordanța dintre tipul referit la compilare și cel referit în timpul execuției evitându-se astfel posibile intruziuni răuvoitoare în sistem prin intermediul unor referințe falsificate. În același timp, Java detectează referințele nule dacă acestea sunt folosite în operații de acces. Indicii în tablourile Java sunt verificăți permanent în timpul execuției și tablourile nu se pot parcurge prin intermediul unor pointeri aşa cum se întâmplă în C/C++. De altfel, pointerii lipsesc complet din limbajul Java, împreună cu întreaga lor aritmetică, eliminându-se astfel una din principalele surse de erori. În plus, eliberarea memoriei ocupate de obiecte și tablouri se face automat, prin mecanismul de colectare de gunoaie, evitându-se astfel încercările de eliberare multiplă a unei zone de memorie.

Java este un limbaj cu securitate ridicată. El verifică la fiecare încărcare codul prin mecanisme de CRC și prin verificarea operațiilor disponibile pentru fiecare set de obiecte. Robustetea este și ea o trăsătură de securitate. La un al doilea nivel, Java are incorporate facilități de protecție a obiectelor din sistem la scriere și/sau citire. Variabilele protejate într-un obiect Java nu pot fi accesate fără a avea drepturile necesare, verificarea fiind făcută în timpul execuției. În plus, mediul de execuție Java

**poate fi configurat pentru a proteja rețeaua locală, fișierele și celelalte resurse ale calculatorului pe care rulează o aplicație Java.**

Limbajul Java are inclus suportul nativ pentru aplicații care lucrează cu mai multe fire de execuție, inclusiv primitive de sincronizare între firele de execuție. Acest suport este independent de sistemul de operare, dar poate fi conectat, pentru o performanță mai bună, la facilitățile sistemului dacă acestea există.

Java este dinamic. Bibliotecile de clase în Java pot fi reutilizate cu foarte mare ușurință. Cunoscuta problemă a fragilității superclasei este rezolvată mai bine decât în C++. Acolo, dacă o superclasă este modificată, trebuie recompilate toate subclasele acesteia pentru că obiectele au o altă structură în memorie. În Java această problemă este rezolvată prin legarea tardivă a variabilelor, doar la execuție. Regăsirea variabilelor se face prin nume și nu printr-un deplasament fix. Dacă superclasa nu a șters o parte dintre vechile variabile și metode, ea va putea fi refolosită fără să fie necesară recompilarea subclaszelor acesteia.

## I.2. JAVA FX

JavaFX este proiectat pentru a oferi dezvoltatorilor Java o nouă platformă grafică ușoară și performantă. Intenția este ca noile aplicații să utilizeze JavaFX mai degrabă decât Swing pentru a construi interfața grafică (GUI) a aplicației. Dat fiind faptul că foarte multe aplicații sunt concepute pe Swing JavaFX este conceput astfel încât cele două API-uri grafice se execută una lângă cealaltă. JavaFX poate fi folosit pentru a crea interfețe grafice pentru utilizatori pentru orice platformă (de ex. Desktop, web, mobil etc.).

Înțial, accentul pentru platforma JavaFX a fost în principal pentru aplicațiile de internet bogate (RIA). A existat un limbaj de scripting JavaFX destinat să faciliteze crearea unei interfețe web. În timpul vieții timpurii a JavaFX nu a fost niciodată foarte clar dacă JavaFX ar înlocui în cele din urmă Swing. După ce Oracle a preluat conducerea Java de la Sun, focusul a fost mutat pentru a face JavaFX platformă grafică de alegere pentru toate tipurile de aplicații Java. Versiunile JavaFX 1.x au o dată la sfârșitul vieții din 20 decembrie 2012. După aceea, aceste versiuni nu vor mai fi disponibile și este recomandat ca orice aplicații de producție JavaFX 1.x să fie migre spre JavaFX 2.0.

În octombrie 2011, JavaFX 2.0 a fost lansat. Acest lucru a semnalat sfârșitul limbajului de scripting JavaFX și mutarea funcției JavaFX într-un Java API. Acest lucru a însemnat că dezvoltatorii Java nu au nevoie să învețe o nouă limbă grafică și, în schimb, să fie confortabil să creeze aplicații JavaFX utilizând sintaxa obișnuită Java. API-ul JavaFX conține tot ce v-ați aștepta de la o platformă grafică - controale UI, animații, efecte etc. Principala diferență pentru dezvoltatorii care trec de la Swing la JavaFX va fi obișnuită cu modul în care sunt prezentate componentele grafice și noua terminologie. O interfață de utilizator este încă construită folosind o serie de straturi care sunt cuprinse într-un grafic de scenă. Graficul grafic este afișat pe un container de nivel superior numit etapă.

Conform site-ului Oracle, principalele avantaje ale JavaFX sunt:

- integrare completă cu Java SE și JDK, începând cu versiunea 7, update 6 (7u6), ceea ce implică faptul că aplicațiile JavaFX vor putea fi dezvoltate și rulate de către orice client cu această versiune de Java;

- inițial JavaFX a fost un limbaj de scripting, dar acum Oracle pune la dispoziție un API pentru dezvoltarea aplicațiilor direct în Java, pentru un mai bun management și reutilizare a codului;
- un nou motor (engine) grafic, numit Prism, care face uz de accelerarea hardware oferită de GPU-urile moderne, precum și un nou manager de ferestre (Window Toolkit) numit Glass;
- un nou limbaj bazat pe XML, numit FXML, folosit pentru descrierea interfețelor grafice, astfel încât să nu fie nevoie de recompilarea codului la fiecare modificare;
- un nou engine multimedia, bazat pe GStreamer, care permite redarea de conținut multimedia;
- componentă care poate afișa conținut web și care poate fi integrată în orice interfață grafică JavaFX;
- serie de componente noi de interfață, precum grafice, tabele, meniuri și panouri;
- un sistem de a împacheta aplicațiile astfel încât acestea să fie livrate cu toate bibliotecile necesare execuției;
- portabilitate pe Linux, Windows și Mac OS X;

Un alt avantaj alt JavaFX spre deosebire de Swing, este faptul că aceeași aplicație poate fi rulată de sine stătător, ca applet, sau ca aplicație de tip Web Start. În plus, <sup>2</sup> oice component din JavaFX poate fi modificat ca aspect folosind directive CSS.

JavaFX se bazează pe o ierarhie de clase care implementează diferite componente și containere ce reprezintă elementele grafice. Clasa care descrie fereastra principală a unei aplicații este javafx.stage.Stage. O aplicație JavaFX (spre deosebire de o aplicație Java obișnuită, care pornește cu metoda main()) trebuie să extindă clasa javafx.application.Application. Aceasta este o clasă abstractă, deci utilizatorul este obligat să definească metoda public void start(Stage \_primaryStage), care este metoda de start a aplicației, analog metodei main().

Un obiect de tip Stage conține, la un moment dat, o singură scenă (javafx.scene.Scene). Această scenă este inițializată dându-i-se dimensiunile scenei și container-ul care conține toate celalalte elemente din fereastră. Acest container este, de cele mai multe ori, un panou. Panoul, pe lângă rolul de container, specifică și modul în care sunt afișate componente.

<sup>2</sup> JavaFX folosește o singură interfață pentru tratarea diferitelor evenimente, javafx.event.EventHandler cu o singură metodă definită (handle(T)), unde T este un şablon folosit pentru a face diferență între diferitele tipuri de evenimente: javafx.event.ActionEvent, javafx.stage.WindowEvent, javafx.scene.web.WebEvent, etc.

Pentru JavaFX există o aplicație care permite dezvoltarea vizuală de interfețe grafice, numită **JavaFX Scene Builder**.

Maven este un instrument de gestionare și înțelegere a proiectelor software utilizat în principal cu proiecte bazate pe Java, dar care poate fi folosit și pentru

gestionarea proiectelor în alte limbaje de programare precum C # și Ruby. Maven ajută la gestionarea compilărilor, documentației, raportării, dependențelor, gestionării configurației software (SCM), lansărilor și distribuției. Multe medii de dezvoltare integrate (IDE) furnizează plug-in-uri sau suplimente pentru Maven, ceea ce permite Maven să compileze proiecte din cadrul IDE.

Maven a fost creat de Jason Van Zyl în 2002 ca parte a proiectului Apache Turbine. A devenit un proiect Apache Software Foundation în 2003. După aceea, au fost lansate mai multe versiuni ale Maven, inclusiv Maven v1.0, v2.0 și v3.0.

Unitatea fundamentală în Maven este modelul obiectului de proiect (POM), un fișier XML care include informații despre proiectul software, detalii de configurare pe care Maven le folosește la construirea acestui proiect, orice dependențe de componente sau module externe și ordinea de construire. Funcționalitatea Maven depinde și de plug-in-uri, care oferă un set de obiective care pot fi executate. De fapt, toate lucrările sunt gestionate de pluginuri. Există numeroase plug-in-uri Maven pentru construire, testare, SCM, rularea unui server Web etc. Plug-in-urile sunt configurate în fișierul POM, unde unele plugin-uri de bază sunt incluse în mod implicit.

Caracteristicile cheie ale Maven includ:

- Un mod simplu și ușor de a construi proiecte în care sunt ascunse detalii inutile
- Un sistem de construire uniform, în care este urmată o strategie standard la construirea oricărui proiect
- Informații de proiect de calitate, cum ar fi liste de dependență, surse cu referință încrucisată și rapoarte de testare a unității
- Managementul dependenței, inclusiv actualizarea automată și închiderea dependenței
- Posibilitatea de a gestiona mai multe proiecte simultan
- Descărcarea dinamică a bibliotecilor și plug-in-urilor Java necesare din depozitele Maven

În cazul nostru Maven este folosit în primul rând pentru gestionarea dependențelor, plug-in-urilor GSON și POI pe care le folosim pentru operațiunile de salvare/citire de date intermediare și finale.

#### I.4. Gson

6

Gson este o bibliotecă Java care poate fi folosită pentru a converti obiectele Java în reprezentarea lor JSON. Poate fi folosit și pentru a converti un sir JSON într-un obiect Java echivalent. Gson poate lucra cu obiecte Java arbitrar, inclusiv obiecte preexistente pentru care nu aveți cod sursă.

Gson furnizează mecanisme ușor de utilizat, cum ar fi `toString()` și constructor pentru a converti Java în JSON și invers. Permite ca obiectele preexistente nemodificabile să fie convertite în și din JSON. Permite reprezentări personalizate pentru obiecte. Sprijină obiecte arbitrar complexe. Generează o ieșire JSON compactă și lizibilă.

Gson a fost creat inițial pentru a fi utilizat în interiorul Google, unde este utilizat în prezent într-o serie de proiecte. Acum este folosit de o serie de proiecte publice și companii.

În cazul nostru ne folosim de biblioteca Gson pentru a salva obiectele claselor de bază, în format JSON, ale aplicației în fazele intermediare a elaborării orarului pentru a se putea relua lucrul într-o altă sesiune, când tot cu ajutorul Gson obiectele se vor citi și se reconverti din JSON în obiectele claselor aplicației.

### I.5. Apache POI

POI este cealaltă bibliotecă de conversie folosită pentru a face legătura cu fișierele excel ale universității, un fișier de intrare care conține toate informațiile necesare privind profesorii, grupele, activitățile din cadrul unei facultăți și fișierele de ieșire care vor <sup>13</sup> conține orarul finalizat.

Misiunea proiectului <sup>13</sup> Apache POI este de a crea și menține API-uri Java pentru manipularea diferitelor formate de fișiere bazate pe standardele Office Open XML (OOXML) și formatul Microsoft OLE 2 Compound Document (OLE2). Pe scurt, puteți <sup>3</sup> căsi scrie fișiere MS Excel folosind Java. În plus, puteți căsi scrie fișiere MS Word și MS PowerPoint folosind Java. Apache POI este soluția Java Excel (pentru Excel 97-2008).

POI are o interfață API completă pentru portarea altor formate OOXML și OLE2. Fișierele OLE2 includ majoritatea fișierelor Microsoft Office, cum ar fi XLS, DOC și PPT, precum și formate de fișiere bazate pe API de serializare MFC. Proiectul oferă API-uri pentru sistemul de fișiere OLE2 (POIFS) și proprietățile documentului OLE2 (HPSF).

Office OpenXML Format este noul format de fișier XML bazat pe standarde găsit în Microsoft Office 2007 și 2008. Acesta include XLSX, DOCX și PPTX. Proiectul oferă un API <sup>3</sup> de nivel scăzut pentru a sprijini convențiile de ambalare deschisă folosind openxml4j. Pentru fiecare aplicație MS Office există un modul de componentă care încearcă să ofere un API Java comun de nivel înalt atât pentru formatele de document OLE2, cât și pentru OOXML.

Acesta este cel mai dezvoltat pentru registrele de lucru Excel (SS=HSSF+XSSF), de fapt asta folosesc și eu în cazul aplicației.

## CAPITOLUL II

După cum se deduce din capitolul anterior aplicația este realizată în Java fiind o aplicație Java FX cu suport Maven folosind bibliotecile Gson și Apache POI pentru realizarea operațiunilor de citire/scriere a diverselor documente.

### II.1. Clasa Activity

Clasa "Activity.java" poate fi considerată clasa de bază a aplicației, obiectele instanțiate ale clasei reprezintă activitățile care trebuie să se desfășoare în cadrul facultății în anul și semestrul pentru care se va elabora orarul, adică cursuri, seminarii, ore de laborator sau activități practice. Obiectele se vor instanția pe baza fișierului excel care conține statul de funcțiuni ale facultății în cauză sau se vor citi dintr-un fișier salvat de către orarist într-o anumită fază de construcție a orarului.

Clasa are următoarele **attribute**:

- int **idActivity** – este indicele, ID-ul unic al activității prin care activitatea poate fi identificată cel mai rapid și simplu, o să vedem în continuare că activitățile sunt identificate de alte clase prin acest ID
- String **subject** – titlul activității cum apare în statul de funcțiuni, de fapt materia care constituie subiectul activității
- String **codeSubject** – este o variantă prescurtată a subiectului activității, se citește deasemenea din același fișier. Această prescurtare se folosește de obicei în afișarea grafică a activităților din cauza că subiectul unei activități poate fi foarte lung și nu poate fi afișat într-o formă grafică care să permită afișarea grafică a cât mai multor activități în niște tabele de orar global sau personal, de exemplu materia "Algebră liniară, geometrie analitică și diferențială" are codul "AG03". Aceste coduri apar în statul de funcțiuni și pe baza lor materiile sunt ușor de identificat de către oricine.
- int **professorId** – reprezintă ID-ul profesorului căruia îi este atribuită activitatea, adică cel care va "ține ora" respectivă. Ca o mică paranteză vreau să precizez că inițial aici am avut chiar un obiect Professor însă din cauza a apărut o relație bidirectională între cele două obiecte, adică clasa Activity avea un atribut Professor iar obiectul Professor avea o matrice și un vector de obiecte Activity ceea ce în momentul stocării prin GSON a dus la o cantitate de date nefondat de mare, astfel am ajuns la concluzia că mai bine introduc aceste attribute de identificare, adică ID-urile, prin care obiectele pot fi identificate mult mai simplu și rapid
- int **type** – tipul activității. Am identificat 4 tipuri de activități în statele de funcțiuni ale Universității și anume Curs, Seminar, Laborator, Practică. Tipul activității este o variabilă de tip număr întreg care este în concordanță cu tipul adică: 1 – CURS, 2 – SEMINAR, 3 – LABORATOR, 4 – ACTIVITATE PRACTICĂ.
- int[] **groupsId** – este un array de numere de identificare a grupelor cărora se adresează activitatea respectivă. Aici este nevoie de o listă pentru că de obicei cursurile sunt pentru mai multe grupe, iar celelalte activități se desfășoară de obicei pe grupe, dar în primul caz o singură activitate trebuie să aibă mai multe

grupe. Acest ID, ca și în cazul professorId, este un ID unic pentru fiecare grupă din cadrul facultății

- int **semester** – reprezintă numărul semestrului în care se va desfășura activitatea
- int **yearOfStudy** – anul de studiu căruia se adresează activitatea. Deși acesta să ar putea deduce și din grupele arondate am considerat necesar un atribut separat pentru o mai ușoară și mai directă identificare a anului de studiu ale activităților
- int **time** – durata activității, este de obicei de două ore dar apar și activități de o oră sau trei ore sau activități bisăptămâna care sunt tratate de fapt ca activități de o oră putând fi alterate cu altă activitate în același interval de orar de două ore ca și câte o activitate de o oră sau câte o activitate de două ore bisăptămâna, care se altereză din două în două săptămâni
- boolean **weekly** – se referă exact la cele amintite despre activitățile bisăptămâna, acesta este un atribut boolean care are valoarea true dacă activitatea se repetă în mod normal, adică săptămânal și este false dacă activitatea se repetă din două în două săptămâni
- int **classRoomId** – este ID-ul sălii de clasă în care se va desfășura activitatea, un ID ca și toate celelalte unic. Săliile de cursuri nu apar în fișierul de state de funcțiuni deci acestea vor trebui introduse manual de către orarist prin selectarea meniului de adăugare sală de curs. Săliile de curs vor fi stocate într-o listă și alegerea acestora se va face în momentul în care o activitate va fi plasată în orar, este de fapt singurul atribut care nu este final, toate celelalte sunt atribuite odată cu instanțierea fiecărui obiect prin constructorul unic despre care vorbim în continuare. De fapt mai există un atribut care poate fi modificat, este vorba de lista de grupe, însă de fapt acesta se va modifica în anumite cazuri în faza de citire a datelor din fișierul de sursă (State de funcțiuni) dar după această fază nu se mai modifică.

Clasa are un singur **constructor** care are forma:

```
public Activity(int idActivity, String subject, String codeSubject,
int professorId, int tip, int[] groupsId, int semester, int
yearOfStudy, int time, boolean weekly){
    this.idActivity=idActivity;
    this.subject = subject;
    this.codeSubject = codeSubject;
    this.professorId = professorId;
    this.type = tip;
    this.groupsId = groupsId;
    this.semester = semester;
    this.yearOfStudy = yearOfStudy;
    this.time = time;
    this.weekly = weekly;
    this.classRoomId = -1;
}
```

Se poate observa că prin acest constructor sunt determinate toate atributele inclusiv classRoomId care primește valoarea negativă care va însemna de fapt că această activitate nu are încă o sală alocată.

### **Metodele clasei:**

- `@Override`  
`public String toString() {`  
    `return idActivity +`  
    `"- " + subject +`  
    `", " + codeSubject +`  
    `", " + type +`  
    `", " + professorId +`  
    `", " + Arrays.toString(groupsId) +`  
    `", semestrul=" + semester +`  
    `", an=" + yearOfStudy +`  
    `", durata=" + time +`  
    `", saptamanal=" + weekly;`  
}
- – metodă supradefinită și care are rol mai mult în debugging-ul actual și eventual viitor, returnează un sir de caractere prin care se poate identifica activitatea.
- `public void addGroups(Group[] newGroups) {`  
    `ArrayList<Integer> groupsToAdd=new ArrayList<>();`  
    `for (int idGroup : groupsId) {`  
        `groupsToAdd.add(idGroup);`  
        `for (Group newGroup : newGroups) {`  
            `if (newGroup.getIdGroup()==idGroup) {`  
                `newGroup.setGroupName("");`  
            }  
        }  
    }  
    `for (Group group : newGroups) {`  
        `if (!group.getGroupName().equals("")) {`  
            `groupsToAdd.add(group.getIdGroup());`  
        }  
    }  
    `int[] newGroup=new int[groupsToAdd.size()];`  
    `for (int i=0;i<newGroup.length;i++)`  
        `newGroup[i] = groupsToAdd.get(i);`  
    `groupsId = newGroup;`  
}

– o metodă prin care se adaugă grupe la lista de grupe ale activității. Această metodă se folosește pe parcursul operației de citire date / creare obiecte în cazul unor activități care au grupe din diverse specializări, în cazul acesta grupele din a doua ( sau treia ) specializare vor fi alocate mai târziu, după instanțierea obiectului.

- `public String getCodeSubject() {`  
    `return codeSubject;`  
}
- returnează codul activității
- `public String getSubject() {`  
    `return subject;`  
}

- returnează titlul activității
- ```
public int getTime() {
    return time;
}
```

  - returnează durata activității
- ```
public int getType() {
    return type;
}
```

  - returnează tipul activității în format numeric
- ```
public int getProfessorId() {
    return professorId;
}
```

  - returnează ID-ul profesorului titular al activității
- ```
public int[] getGroupsId() {
    return groupsId;
}
```

  - returnează lista ID-urilor grupelor arondate activității
- ```
public int getSemester() {
    return semester;
}
```

  - returnează semestrul activității
- ```
public int getIdActivity() {
    return idActivity;
}
```

  - returnează ID-ul activității
- ```
public String getTypeChar() {
    String[] typeChar={"C","S","L","P"};
    try {
        return typeChar[this.type-1];
    }
    catch (Exception ex) {
        return "X";
    }
}
```

  - returnează tipul activității în format de caracter, adică "C" – pentru curs, "S" – pentru seminar, "L" – pentru laborator și "P" – în cazul activităților practice. În cazul în care tipul activității nu este nici unul dintre cele enumerate, caz teoretic inexistent, metoda returnează "X". Metoda se folosește în momentul creării etichetelor indexate (IndexedLabel) care constituie reprezentarea grafică a obiectelor Activity.
- ```
public int getClassRoomId() {
    return classRoomId;
}
```

  - returnează ID-ul sălii de clasă atribuite activității. În cazul în care nu este nici o sală atribuită încă acesta are valoarea de -1.

- ```
public void setClassRoomId(int classRoom) {  
    this.classRoomId =classRoom;  
}  
– setează ID-ul sălii de clasă care se atribuie activității.
```

## II.2. Clasa Professor

Clasa "Professor.java" este clasa obiectelor de tip profesor, membrii acestei clase sunt profesorii care sunt titularii activităților din cadrul facultății. Ca și în cazul activităților obiectele se vor instanția pe baza fișierului excel care conține statul de funcțiuni ale facultății în cauză sau se vor citi dintr-un fișier salvat de către orarist într-o anumită fază de construcție a orarului.

Clasa are următoarele **attribute**:

- int **idProfessor** – este un identificator unic pentru fiecare obiect profesor instantiat, acest ID este folosit de toate celelalte clase sau metode pentru identificarea profesorilor.
- String **name** – conține numele profesorului aşa cum apare în fișierul sursă
- String **shortName** – este numele prescurtat al profesorului care se generează de către constructor. Această prescurtare se folosește de obicei în afișarea grafică a activităților din cauza că numele complet poate fi prea lung și nu poate fi afișat într-o formă grafică care să permită afișarea grafică a cât mai multor activități în niște tabele de orar global sau personal.
- int[][][] **scheduleProfessor** – este o matrice tridimensională care conține, de fapt, orarul profesorului. Este matrice tridimensională pentru că conține orarul pe semestru / oră / zi. Avem 2 semestre, 7 intervale orare și 12 zile. Avem 12 zile pentru că de fapt avem câte o zi pentru săptămânile pare și cele impare. Numerele conținute sunt de fapt numere de identificare pentru activități iar dacă numărul este -1 înseamnă că în ziua și ora respectivă profesorul nu are alocată nici o activitate.
- int[] **activitiesOfProfessor** - este un array care conține toate activitățile profesorului, activitățile fiind reprezentate și aici de ID-ul fiecărui.

Unicul **constructor** al clasei arată în felul următor:

```
public Professor(int id, String name){  
    final int HOURS=7,DAYS=12;  
    this.idProfessor = id;  
    this.name = name;  
    String[] names=name.split(" ");  
    this.shortName =  
    name[240].substring(0,1)+names[0].substring(1).toLowerCase();  
    for (int i=1;i<names.length;i++){  
        this.shortName += "_" + names[i].substring(0, 1);  
    }  
    this.scheduleProfessor = new int[2][HOURS][DAYS];  
    for(int i=0;i<2;i++)  
        for(int j=0;j<HOURS;j++)  
            for(int k=0;k<DAYS;k++)  
                scheduleProfessor[i][j][k]=-1;  
    activitiesOfProfessor = new int[0];  
}
```

După cum se poate observa constructorul are doi parametrii iar valorile restului de attribute sunt generate de către constructor. **shortName** este generat din

**name**, se generează orarul fără nici o activitate, adică avem numai valori de -1 în toate câmpurile matricei, iar arrayul de activități se inițiază fără elemente, acestea se vor adăuga în momentul citirii datelor de intrare din fișierul statelor de funcțiuni.

#### **Metodele clasei:**

- `public int getIdProfessor() {  
 return idProfessor;  
}  
– returnează numărul de identificare unic al profesorului`
- `public void setIdProfessor(int id) {  
 this.idProfessor=id;  
}  
– setează ( schimbă ) ID-ul profesorului, metoda se folosește în cadrul procesului de citire / generare date`
- `public String getName() {  
 return name;  
}  
– returnează un String cu numele profesorului`
- `public String getShortName() {  
 return shortName;  
}  
– returnează un String cu numele prescurtat`
- `public int[] getActivitiesOfProfessor() {  
 return activitiesOfProfessor;  
}  
– returnează un array de numere întregi care conține activitățile alocate profesorului pe parcursului anului școlar pentru care facem orarul`
- `public int getActivityProfessor(int semester, int hour, int day) {  
 try {  
 return scheduleProfessor[semester-1][hour][day];  
 }  
 catch (Exception ex) {  
 return -1;  
 }  
}  
– returnează ID-ul activității din orarul profesorului din semestrul, ora, ziua primite ca parametrii de intrare. În cazul în care valoarea returnată este -1 înseamnă că în semestrul dat în ziua și a ora primite ca parametrii nu există activitate, profesorul este liber.`
- `public boolean setActivityProfessor(int semester, int hour, int day, int activity) {  
 try {  
 scheduleProfessor[semester-1][hour][day]=activity;  
 return true;  
 }`

```

        catch (Exception ex) {
            Utility.errorMessage("Activitatea nu a fost adăugată");
            return false;
        }
    }
}

```

– această metodă setează/schimbă activitatea profesorului din semestrul, ora, ziua cu activitatea cu ID-ul primit ca parametru de intrare. Metoda returnează "true" dacă această setare a fost reușită și returnează "false" dacă din orice motiv operația a eșuat.

- ```

public boolean addActivity (int activity) {
    for (int act:activitiesOfProfessor) {
        if (activity==act)
            return false;
    }
    int size= activitiesOfProfessor.length;
    int[] newActivites=new int[size+1];
    System.arraycopy(activitiesOfProfessor, 0, newActivites, 0,
size);
    newActivites[size]=activity;
    activitiesOfProfessor =new int[size+1];
    System.arraycopy(newActivites, 0, activitiesOfProfessor, 0,
size + 1);
    return true;
}

```

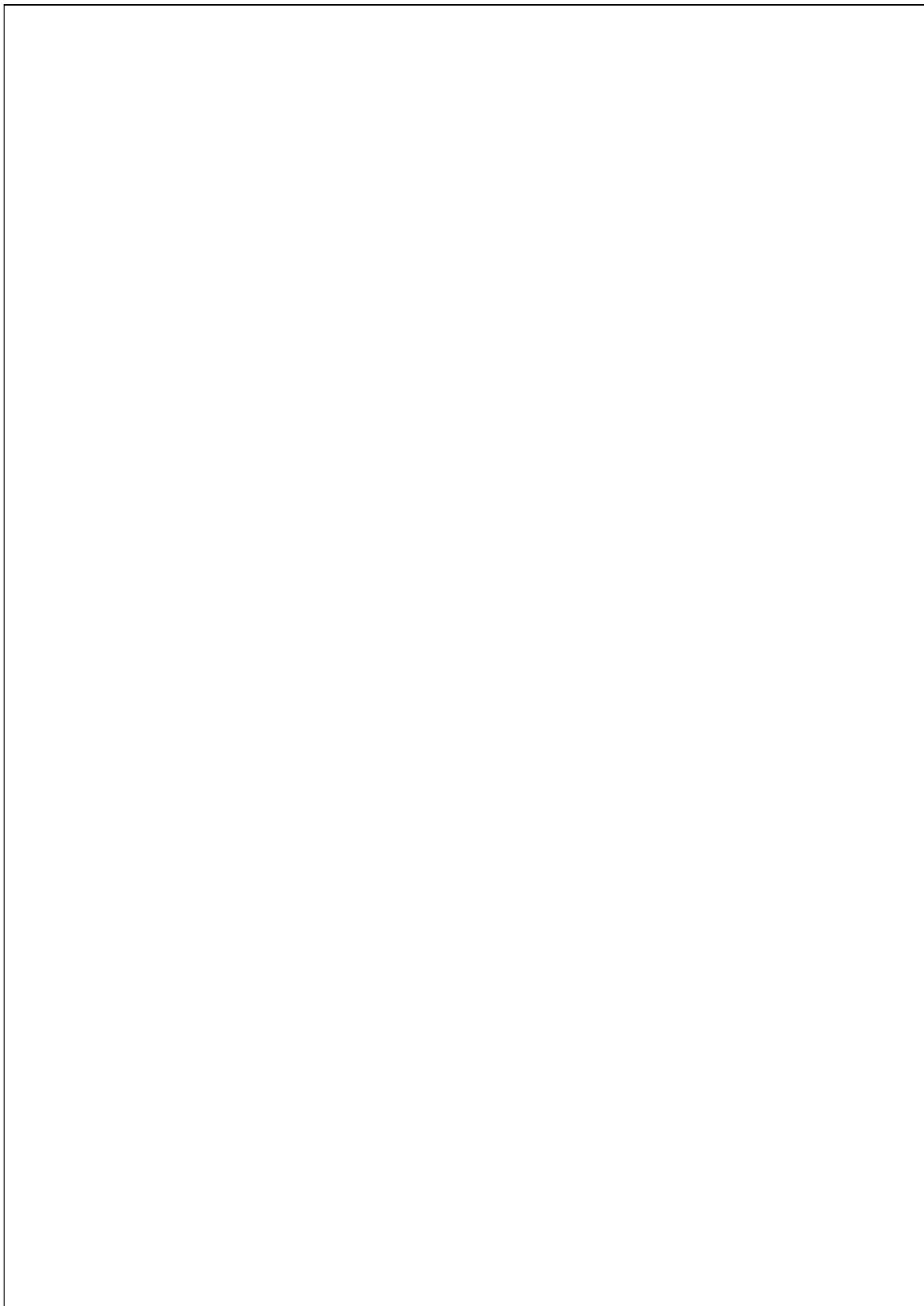
– se adaugă o activitate la arrayul de activități ale profesorului. Se returnează "true" dacă elementul a fost adăugat și "false" dacă nu s-a reușit adăugarea noului element. De exemplu dacă o activitate este deja în lista profesorului acesta nu se va adăuga din nou, în acest caz operațiunea nu reușește și se returnează "false", lista/arrayul activităților rămâne neschimbată.

- ```

public boolean removeActivity (int activity) {
    int size= activitiesOfProfessor.length;
    int[] newActivities=new int[size-1];
    int j=0;
    for (int i=0;i<size;i++) {
        try {
            if (activitiesOfProfessor[i]!=activity) {
                newActivities[j]= activitiesOfProfessor[i];
                j++;
            }
        }
        catch (Exception ex) {
            Utility.errorMessage("Activitatea nu a fost
ștersă");
            return false;
        }
    }
    activitiesOfProfessor = new int[size-1];
    if (size - 1 >= 0) System.arraycopy(newActivities, 0,
activitiesOfProfessor, 0, size - 1);
    return true;
}

```

– se șterge o anumită activitate din arrayul activităților profesorului. Metoda returnează "true" dacă operațiunea a reușit și "false" în caz contrar.



### II.3. Clasa Group

Clasa grupelor este clasa a căror obiecte instanțiate reprezintă grupele de la facultatea a cărei orar se elaborează. Grupele sunt instanțiate ca și activitățile și profesorii în momentul citirii și prelucrării datelor din statul de funcțiuni al facultății în cauză. Clasa Group este asemănătoare cu clasa Professor însă sunt și anumite diferențe față de aceasta dat fiind particularitățile celor două entități.

**Atributele** clasei Group sunt următoarele:

- int **idGroup** – numărul unic de identificare a grupei, număr care se folosește pentru identificarea grupei de către alte obiecte, metode etc.
- private String **groupName** - numele grupei care în primă instanță este generat în momentul citirii datelor și a generării obiectelor de tip Activity, Professor și Group. Numele generat al grupei se compune din codul specialității, anul de studiu și numărul grupei. De exemplu pentru grupa a doua din anul I. la specializarea ( programul de studii ) Matematică-Informatică numele generat este "MI12", adică Matematică-Informatică, anul 1, grupa 2.
- private int[][] **scheduleGroup** – matricea orarului grupei. Este o matrice similară ca în cazul profesorilor, care conține activitățile din orar pe pozițiile setate și -1 unde grupa nu are alocată nici o activitate
- private int[] **activitiesOfGroup** – array care conține toate activitățile alocate grupei.
- private String **speciality** – specializarea sau programul de studii al grupei
- private int **year** – anul de studiu
- private int **groupNumber** – numărul grupei din specializarea și anul date

**Constructorul** unic al clasei:

```
public Group(int id, String speciality, int year, int groupNumber){  
    final int HOURS = 7, DAYS = 12;  
    this.idGroup=id;  
    this.groupName=speciality+year+groupNumber;  
    this.scheduleGroup =new int[2] [HOURS] [DAYS];  
    for(int i=0;i<2;i++)  
        for(int j=0;j<HOURS;j++)  
            for(int k=0;k<DAYS;k++)  
                scheduleGroup[i][j][k]=-1;  
    activitiesOfGroup = new int[0];  
    this.speciality=speciality;  
    this.year=year;  
    this.groupNumber=groupNumber;  
}
```

Constructorul prezintă similarități cu cel al clasei Professor, se generează numele grupei, nume care poate fi modificat de către orarist la un moment dat dacă se dorește acest lucru. Matricea asemănătoare de orar se inițiează cu valorile de -1, însemnând că în semestrul, ziua și ora respectivă în orar nu este nici o activitate. Se

inițiează array-ul de activități a grupei, activitățile se vor adăuga în cadrul procesului de citire/initializare date.

**Metodele** clasei sunt următoarele:

- ```
public String getSpeciality() {
    return speciality;
}
```

– returnează specialitatea ( programul de studii ) de care aparține grupa
- ```
public int getYear() {
    return year;
}
```

– returnează anul de studii al grupei
- ```
public String getGroupName() {
    return groupName;
}
```

– returnează numele grupei
- ```
public void setGroupName(String name){
    this.groupName=name;
}
```

– setează/schimbă numele grupei
- ```
public int getIdGroup() {
    return idGroup;
}
```

– returnează numărul de identificare unic al grupei
- ```
public void setIdGroup(int id) {
    this.idGroup=id;
}
```

– setează/schimbă numărul de identificare, ID-ul grupei
- ```
public int getActivityGroup(int semester,int hour,int day) {
    try {
        return scheduleGroup[semester-1][hour][day];
    }
    catch (Exception ex) {
        return -1;
    }
}
```

– returnează ID-ul activității din orarul grupei din semestrul, ora și ziua specificate de parametrii transmiși. Dacă pe poziția respectivă nu este nici o activitate atunci se va returna -1, valoare standard pentru orele fără activități
- ```
public boolean setActivityGroup(int semester,int hour,int day,
int activity) {
    try {
        scheduleGroup[semester-1][hour][day]=activity;
        return true;
    }
    catch (Exception ex) {
```

```

        Utility.errorMessage("Activitatea nu a fost adăugată");
        return false;
    }
}

```

- se setează o activitate pentru un anumit moment din orar, metoda returnează "true" dacă setarea s-a reușit și va returna false dacă operațiunea a eșuat

- ```

public boolean addActivity (int activity) {
    for (int act:activitiesOfGroup) {
        if (activity==act)
            return false;
    }
    int size=activitiesOfGroup.length;
    int[] newActivities=new int[size+1];
    System.arraycopy(activitiesOfGroup, 0, newActivities, 0,
size);
    newActivities[size-1]=activity;
    activitiesOfGroup=new int[size+1];
    System.arraycopy(newActivities, 0, activitiesOfGroup, 0,
size + 1);
    return true;
}

```

- se adaugă un ID de activitate la array-ul de activități a grupei. Metoda returnează "true" dacă operațiunea s-a realizat cu succes și returnează "false" dacă nu s-a reușit adăugarea activității din diverse motive.
- ```

public boolean removeActivity (int activity) {
    int size=activitiesOfGroup.length;
    int[] newActivities=new int[size-1];
    int j=0;
    for (int i=0;i<size;i++) {
        try {
            if (activitiesOfGroup[i]!=activity) {
                newActivities[j]=activitiesOfGroup[i];
                j++;
            }
        }
        catch (Exception ex) {
            Utility.errorMessage("Activitatea nu a fost
ștersă");
            return false;
        }
    }
    activitiesOfGroup = new int[size-1];
    if (size - 1 >= 0) System.arraycopy(newActivities, 0,
activitiesOfGroup, 0, size - 1);
    return true;
}

```

- se șterge o anumită activitate din arrayul activităților grupei. Metoda returnează "true" dacă operațiunea a reușit și "false" în caz contrar.

## II.4. Clasa Room

Este clasa sălilor disponibile pentru desfășurarea activităților. Va fi de datoria oraristului să încarce manual lista sălilor disponibile înainte să se aplice de orar pentru că va fi nevoie să aleagă și sala de desfășurare a activităților în momentul când plasează o activitate pe orarul unui profesor sau pe orarul complet al grupelor.

**Atributele** clasei sunt:

- int **roomId** – numărul unic de identificare a sălii, se folosește în relațiile cu alte clase
- String **roomName** – denumirea sălii, se va introduce de către orarist înainte de începerea elaborării efective a orarului
- Int[][][] **scheduleRoom** – este matricea orarului sălii. Este o matrice similară ca în cazul profesorilor sau grupelor, care conține activitățile din orar pe pozițiile setate și -1 unde sala nu are alocată nici o activitate

**Constructorul** clasei este:

```
public Room(int id, String name) {  
    final int HOURS = 7, DAYS = 12;  
    this.roomId = id;  
    this.roomName = name;  
    for(int i=0; i<2; i++)  
        for(int j=0; j<HOURS; j++)  
            for(int k=0; k<DAYS; k++)  
                scheduleRoom[i][j][k] = -1;  
}
```

Constructorul are parametrii de intrare id-ul sălii și denumirea acesteia, în momentul instantierii unui obiect Room constructorul construiește matricea de orar a sălii și o inițiază cu valori de -1, însemnând că nu există nici o activitate în sală.

**Metodele** clasei sunt:

- **public int getRoomId()** {  
 return roomId;  
}  
- returnează numărul de identificare unic al sălii de curs
- **public String getRoomName()** {  
 return roomName;  
}  
- returnează numele sălii
- **public void setRoomName(String roomName)** {  
 this.roomName = roomName;  
}  
- setează/schimbă numele sălii

- ```
public int getActivityRoom(int semester, int hour, int day) {
    try {
        return scheduleRoom[semester-1][hour][day];
    }
    catch (Exception ex) {
        return -1;
    }
}
```

– returnează ID-ul activității din orarul grupei din semestrul, ora și ziua specificate de parametrii transmiși. Dacă pe poziția respectivă nu este nici o activitate atunci se va returna -1, valoare standard pentru orele fără activități
- ```
public boolean setActivityRoom(int semester, int hour, int day,
int activity) {
    try {
        scheduleRoom[semester-1][hour][day]=activity;
        return true;
    }
    catch (Exception ex){
        Utility.message("Activitatea nu a fost adăugată");
        return false;
    }
}
```

– se setează o activitate pentru un anumit moment din orar, metoda returnează "true" dacă setarea s-a reușit și va returna false dacă operațiunea a eșuat

## II.5. Clasa IndexedLabel

Este o clasă care extinde clasa **Label** al JavaFX și este o etichetă ( **Label** ) care față de obiectele din clasa extinsă are 3 atribute și anume:

- int **activityId** – conține ID-ul activității a cărei reprezentare grafică este acest obiect
- int **professorId** – este ID-ul profesorului arondat activității
- int[] **groupsId** – un array cu ID-urile grupelor activității

**Constructorul** este unul simplu:

```
public IndexedLabel(int activityId, int professorId, int[] groupsId)
{
    this.activityId = activityId;
    this.professorId = professorId;
    this.groupsId = groupsId;
}
```

este adăugat și un constructor fără parametri.

**Metodele** sunt doar gettere și setttere pentru cei trei parametrii și desigur toate metodele moștenite de la clasa **Label**.

- public int getActivityId() {  
 return activityId;
- [21] public int getProfessorId() {  
 return professorId;  
}
- public int[] getGroupsId() {  
 return groupsId;  
}
- public void setActivityId(int activityId) {  
 this.activityId = activityId;  
}
- public void setProfessorId(int professorId) {  
 this.professorId = professorId;  
}
- public void setGroupsId(int[] groupsId) {  
 this.groupsId = groupsId;  
}

## II.5. Clasa Utility

Este o clasă de utilități care are multe metode statice utilizator, adică de ajutor și metodele de input/output ale aplicației. Clasa nu este folosită ca și clasă, de fapt nici nu se poate instanția, este mai mult o colecție de metode de care ne folosim în aplicație fără să fie necesar instantierea unui obiect. Această abordare nu este neapărat specific Java, de fapt nu este deloc în stilul Java și sunt multe voci care chiar îndeamnă la evitarea folosirii unor clase de genul Utility, dar desigur există și mulți care susțin utilitatea lor. Dar fiind faptul că aplicația mea are o arhitectură de acțiune, adică suport pentru elaborarea orarului facultăților din cadrul Universității Transilvania din Brașov, nu cred că aplicația va trece printr-o asemenea dezvoltare în care această clasă să prezinte probleme pentru dezvoltarea aplicației, deci m-am hotărât să folosesc această clasă de utilități pentru metodele care nu sunt legate neapărat de o clasă, obiecte sau colecții de obiecte și care nu sunt necesare a fi metode aparținând unor obiecte. În continuare o să prezint metodele clasei.

### II.5.1. Metoda `readXls`

este metoda prin care se citește fișierul stat de funcțiuni și care creează liste de activități, lista profesorilor și cel al grupelor. În continuare o să prezint metoda și o să comentez pe faze de execuție.

```
public static ArrayList<Activity> readXls(String fileName,
ArrayList<Professor> professors, ArrayList<Group> groups, String
faculty) {
    ArrayList<Activity> activities=new ArrayList<>();
    File file=new File(fileName);
    professors.clear();
    groups.clear();
    profIndex=0;
    groupIndex=0;
    activityIndex=0;
```

Aici sunt fost inițiate liste de profesori și grupe și variabilele de indice ale celor trei entități.

```
try {
    FileInputStream fileInputStream = new FileInputStream(file);
    HSSFWorkbook hssfWorkbook = new HSSFWorkbook(fileInputStream);
    int numberOfWorksheets=hssfWorkbook.getNumberOfSheets();
    int sheetToRead=-1;
    for (int i=0;i<numberOfWorksheets;i++){
        if (hssfWorkbook.getSheetName(i).equals("Centr")) {
            sheetToRead=i;
            i=numberOfWorksheets;
        }
    }
    if (sheetToRead== -1) {
        throw new NoSheetException("Sheet nonexistent");
```

```

        }
HSSFSheet hssfSheet=hssfWorkbook.getSheetAt(sheetToRead);

```

Aici se deschide fișierul "file", parametru de intrare a metodei, și se caută foaia (sheet) "Centr" care conține datele necesare creării obiectelor de tip Activity, Professor și Group. Dacă fișierul nu conține această foaie metoda aruncă o excepție denumită NoSheetException, excepție creată de către mine special pentru această excepție.

Datele care trebuie citite încep din rândul 10 al foii, de aici vom începe și noi citirea datelor și vom citi date până în momentul în care nu mai există date de citit în celule. Fișierul conține date pentru fiecare materie în parte, datele unei materii sunt stocate în două rânduri concomitente, prima materie este poziționată în rândurile 10-11. Mai jos un exemplu de date pentru o materie:

| Nr crt | DISCIPLINA             | Fac. | Dep. | Specializarea | An de studiu | Codul disciplinei | SEMESTRUL I |   |    |    |    | SEMESTRUL II |    |    |    |
|--------|------------------------|------|------|---------------|--------------|-------------------|-------------|---|----|----|----|--------------|----|----|----|
|        |                        |      |      |               |              |                   | C           | S | L  | P  |    | C            | S  | L  | P  |
| 1.0    | 2                      | 3    | 4    | 5             | 6            | 7                 | 8           | 9 | 10 | 11 | 12 | 14           | 15 | 16 | 17 |
| 157    | Algoritmi fundamentali | MI   |      | MIN           | ITR          | 1                 | IT11        | 0 | 0  | 0  | 0  | 0            | 0  | 0  | 0  |
|        | IT11,1,4,4,0           |      |      |               | 175          |                   |             | 2 | 4  | 8  |    |              |    |    |    |

| Verif.attrib.StafF | Discipline cuplate interfacultati |         | Posturile din statul de functii in care se afla disciplina |  |  |                           |  |  |                           |  |  |  |
|--------------------|-----------------------------------|---------|------------------------------------------------------------|--|--|---------------------------|--|--|---------------------------|--|--|--|
|                    | la curs                           | la sem. |                                                            |  |  |                           |  |  |                           |  |  |  |
| 18                 | 21                                | 22      | 28                                                         |  |  |                           |  |  |                           |  |  |  |
| ok                 | 1                                 |         | 48 ; B ; 1                                                 |  |  | 71 ; B ; 1                |  |  | 81 ; PO ; 1               |  |  |  |
|                    |                                   |         | BAICOIANU Alexandra;2;2;0;0                                |  |  | MAJERCSIK Luciana;0;0;8;0 |  |  | MAJERCSIK Luciana;0;2;0;0 |  |  |  |

Avem în primul rând următoarele informații folosite: disciplina "Algoritmi fundamentali", de la facultatea de Matematică și Informatică ("MI"), departamentul de Matematică și Informatică ("MIN"), anul de studiu 1, codul disciplinei "IT11". În rândul 2 avem informații legate de activitățile și profesorii disciplinei. În coloana 2 din rândul 2 apare "IT11,1,4,4,0" ceea ce înseamnă că disciplina cu codul IT11 este predat în următoarele formațiuni : la curs 1 formațiune, adică toate grupele la un singur curs, la seminar 4 formațiuni, la laborator 4 formațiuni iar la practică 0, ceea ce înseamnă că nu există practică la această disciplină. În 18 bloanele 9-17 apare durata totală a activităților disciplinei, adică în cazul de față 2 ore de curs, 4 ore de seminar și 8 ore de laborator. În coloanele cu indexul 28 apar profesorii cu activitățile atribuite la această disciplină în exemplul de față avem în felul următor: BAICOIANU Alexandra are 2 ore de curs și 2 ore de seminar iar MAJERCSIK Luciana are 8 ore de laborator și 2 ore de seminar. În ceea ce urmează în metodă avem o primă fază de citire a datelor din fișier cu tot felul de verificări, conversii etc.

```

//start row
int r=10;
while (!hssfSheet.getRow(r).getCell(1).toString().equals(""))
{
    Row firstRow=hssfSheet.getRow(r);
    if (faculty.equals(firstRow.getCell(2).toString()))
    {

```

```

String ok=firstRow.getCell(17).toString();
if (ok.equals("ok")) {
    Row secondRow = hssfSheet.getRow(r + 1);
    String subject = firstRow.getCell(1).toString().trim();
    String departament = firstRow.getCell(4).toString().trim();
    String[] speciality =
firstRow.getCell(5).toString().split("\\+");
    for (String spec : speciality)
        System.out.println(spec);
    int year = (int)
firstRow.getCell(6).getNumericCellValue();
    String[] codeFormation =
secondRow.getCell(1).toString().split(",");
    String codeSubject = codeFormation[12];
    int fCrs = Integer.parseInt(codeFormation[1]);
    int fSem = Integer.parseInt(codeFormation[2]);
    int fLab = Integer.parseInt(codeFormation[3]);
    int fPrc = Integer.parseInt(codeFormation[4]);
    int s1C, s1S, s1L, s1P, s2C, s2S, s2L, s2P;
    if (secondRow.getCell(8, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s1C = (int) secondRow.getCell(8).getNumericCellValue();
    else s1C = 0;
    if (secondRow.getCell(9, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s1S = (int) secondRow.getCell(9).getNumericCellValue();
    else s1S = 0;
    if (secondRow.getCell(10, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s1L = (int) secondRow.getCell(10).getNumericCellValue();
    else s1L = 0;
    if (secondRow.getCell(11, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s1P = (int) secondRow.getCell(11).getNumericCellValue();
    else s1P = 0;
    if (secondRow.getCell(13, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s2C = (int) secondRow.getCell(13).getNumericCellValue();
    else s2C = 0;
    if (secondRow.getCell(14, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s2S = (int) secondRow.getCell(14).getNumericCellValue();
    else s2S = 0;
    if (secondRow.getCell(15, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s2L = (int) secondRow.getCell(15).getNumericCellValue();
    else s2L = 0;
    if (secondRow.getCell(16, Row.MissingCellPolicy.
        RETURN_BLANK_AS_NULL) != null)
        s2P = (int) secondRow.getCell(16).getNumericCellValue();
    else s2P = 0;

    int semester = 0;
    if (s1C + s1S + s1L + s1P > 0) semester = 1;
    else if (s2C + s2S + s2L + s2P > 0) semester = 2;

```

În secțiunea de mai sus sunt citite datele cu privire la disciplină până adică denumire disciplină, departament, specializare, codul disciplinei, numărul de formațiuni, numărul de ore și ajungem la fază următoare în care se parsează datele despre profesori și aplicația "distribuie" orele la profesori ajungând în final la crearea

obiectelor de Activity, adică a activităților. După cum am văzut aici avem doi profesori care au diverse activități.

În primă fază metoda numără profesorii și rulează un algoritm de distribuire a activităților. Avem un prim ciclu pe numărul de profesori în care parsăm datele privind numărul de ore alocate fiecărui profesor și le stocăm în arrayurile aferente ( crsTime[x], semTime[x], labTime[x], prcTime[x] ), totodată însumăm aceste valori în niște variante care conțin numărul de ore pentru fiecare tip de activitate, adică crsTotal, semTotal, labTotal, prcTotal, însemnând numărul de ore de cursuri/seminarii/laboratoare/practică totale alocate profesorilor arondări disciplinei. Se apelează metoda **addIfNotInProfs** în felul următor

```
actualProfs[i] = addIfNotInProfs(actualProfs[i], professors);
```

aici de fapt se verifică dacă obiectul profesor a fost deja creat sau trebuie creat acumă, acesta fiind prima disciplină unde apare profesorul în cauză.

```
if (semester > 0) {
    int c = 27;
    while
(!secondRow.getCell(c, Row.MissingCellPolicy.CREATE_NULL_AS_BLANK) .
    toString().equals("")) {
        c++;
    }
    int nrProfs = c - 27;
    if (nrProfs > 0) {
        Professor[] actualProfs = new Professor[nrProfs];
        String[] profNames = new String[nrProfs];
        int[] crsTime = new int[nrProfs];
        int[] semTime = new int[nrProfs];
        int[] labTime = new int[nrProfs];
        int[] prcTime = new int[nrProfs];
        int crsTotal = 0;
        int semTotal = 0;
        int labTotal = 0;
        int prcTotal = 0;
        for (int i = 0; i < nrProfs; i++) {
            String[] profData = secondRow.getCell(27 +
                i).toString().split(";");
            profNames[i] = profData[0];
            crsTime[i] = Integer.parseInt(profData[1]);
            crsTotal += crsTime[i];
            semTime[i] = Integer.parseInt(profData[2]);
            semTotal += semTime[i];
            labTime[i] = Integer.parseInt(profData[3]);
            labTotal += labTime[i];
            prcTime[i] = Integer.parseInt(profData[4]);
            prcTotal += prcTime[i];
            actualProfs[i] = new Professor(0, profNames[i]);
            actualProfs[i] = addIfNotInProfs(actualProfs[i],
                professors);
        }
    }
}
```

Un al doilea ciclu verifică dacă nu cumva avem de a face cu o dublare în fișierul excel a vreunia dintre profesorii disciplinei. În exemplul nostru avem de a face cu această situație, unul dintre profesori apare în două coloane, informația se poate

comasa astfel că orele alocate la a două instanță vor fi transferate la prima instanță a profesorului și a două instanță va fi ștearsă din lista profesorilor alocați disciplinei.

```
15
for (int i = 0; i < nrProfs; i++) {
    for (int j = i + 1; j < nrProfs; j++) {
        if (actualProfs[i] != null && actualProfs[i].equals(actualProfs[j])) {
            crsTime[i] += crsTime[j];
            crsTime[j] = 0;
            semTime[i] += semTime[j];
            semTime[j] = 0;
            labTime[i] += labTime[j];
            labTime[j] = 0;
            prcTime[i] += prcTime[j];
            prcTime[j] = 0;
            actualProfs[j] = null;
        }
    }
}
```

În faza următoare se vor crea sau updata grupele care vor avea această disciplină în orar. În ceea ce urmează vedem cum se creează 4 arrayuri de arrayuri pentru fiecare tip de activitate posibilă. În acestea vom stoca grupele care vor avea activități legate de disciplină. Si aici folosim o metodă asemănătoare de creare/allocare grupă **addIfNotInGroups**. Metoda creează o nouă grupă și o adaugă listei de grupe sau returnează o grupă existentă deja în lista grupelor.

```
int numberGroups = max(fCrs, fSem, fLab, fPrc);
5 Group[][] actualGroups = new Group[4][numberGroups * nrProfs];
for (int i = 0; i < numberGroups; i++) {
    actualGroups[0][i] = new Group(0, speciality[0], year, i + 1); 38
    actualGroups[0][i] = addIfNotInGroup(actualGroups[0][i], groups);
    for (int j = 1; j < 4; j++) {
        actualGroups[j][i] = actualGroups[0][i];
    }
}
for (int i = 1; i < nrProfs; i++) {
    for (int k = 0; k < numberGroups; k++) {
        for (int j = 0; j < 4; j++) {
            actualGroups[j][i * numberGroups + k] = actualGroups[j][k];
        }
    }
}
```

După această fază ajungem la crearea efectivă a activităților, al obiectelor de tip Activity care se adaugă la lista **activities** care va conține toate activitățile din anul universitar și pe care le vom putea manipula sub forma unor etichete între tabele și în interiorul tabelelor de orar. Activitățile se creează de o metodă denumită **createActivities** care, de fapt, creează mai multe activități deodată. Metoda se apelează pentru fiecare tip de activitate care urmează să fie creat/create și se vor crea una sau mai multe activități, depinde de câte trebuie, câte sunt prevăzute pentru fiecare tip de activitate în parte ( curs, seminar, laborator, practică ). Mă voi referi separat la metoda **createActivities** puțin mai târziu.

```
31
for (int i = 0; i < nrProfs; i++) {
    if (crsTime[i] > 0) {
```

```

int type = 1;
createActivities(activities, professors, groups, i,
    numberOfGroups, fCrs, actualGroups, subject, codeSubject,
    actualProfs, type, semester, year, crsTotal, crsTime, s1C+s2C);
}
if (semTime[i] > 0) {
    int type = 2;
    createActivities(activities, professors, groups, i,
        numberOfGroups, fSem, actualGroups, subject, codeSubject,
        actualProfs, type, semester, year, semTotal, semTime, s1C+s2C);
}
if (labTime[i] > 0) {
    int type = 3;
    createActivities(activities, professors, groups, i,
        numberOfGroups, fLab, actualGroups, subject, codeSubject,
        actualProfs, type, semester, year, labTotal, labTime, s1C+s2C+1);
}
if (prcTime[i] > 0) {
    int type = 4;
    createActivities(activities, professors, groups, i,
        numberOfGroups, fPrc, actualGroups, subject, codeSubject,
        actualProfs, type, semester, year, prcTotal, prcTime, s1C+s2C+1);
}
}

```

Ultima secțiune a metodei tratează un caz special și anume cazul în care o activitate, de obicei în cazul cursurilor se întâmplă, este destinat mai multor grupe din cadrul mai multor specializări. În acest caz cursul ca și activitate alocată unui profesor apare în fișa de funcțiuni numai în cazul uneia dintre specializări ( de obicei prima ) și la specializarea 2 apare disciplina însă fără curs alocat vreunui profesor pentru că a fost alocat deja. În tabel aici apar celelalte activități, laboratoare, seminarii care se referă la grupele cu această specializare. Următoarea secțiune de cod tratează această situație specială adăugând grupele din specializarea la care nu mai apare cursul la activitatea deja creată și adăugată la lista de activități, **activities**.

```

if (s1C + s2C == 0) {
    List<Group> groupsToAddToCourse=new ArrayList<Group>();
    boolean isIn;
    for (Group newGroup:actualGroups[0]) {
        isIn=false;
        for(Group oldGroup:groupsToAddToCourse) {
            if (oldGroup==newGroup) {
                isIn=true;
            }
        }
        if (!isIn) {
            groupsToAddToCourse.add(newGroup);
        }
    }
    Group[] groupsToAdd=groupsToAddToCourse.toArray(new
        Group[0]);
    for (Activity completable:activities) {
        if (completable.getCodeSubject().equals(codeSubject)) {
            if (completable.getType()==1) {
                completable.addGroups(groupsToAdd);
            }
        }
    }
}

```

```
}
```

În final se trece la următoarea disciplină variabilă r ( de la rând ) este mărită astfel încât să se citească rândul unu al disciplinei următoare.

```
    r=r+2;  
}
```

Dacă în cursul citirii/prelucrării datelor se aruncă orice excepție metoda va afișa excepția respectivă și va returna **null**, meniul principal va afișa că există o greșeală de citire/prelucrare a datelor.

```
32     catch (Exception ex){  
33         System.out.println(ex.toString());  
34         return null;  
35     }  
36     return activities;  
37 }
```

### II.5.2. Metoda `writeXls`

este metoda care exportă orarul în formatul excel. Metoda se folosește, desigur, de aceeași bibliotecă Apache POI pentru conversia în formatul Excel dorit.

XXXXXXXXXXXXXXXXXXXX

### II.5.3. Metoda addIfNotInGroup

este o metodă de ajutor pentru `readXls` și care verifică dacă o grupă există deja și este adăugată în lista grupelor, **`groups`**.

```
static Group addIfNotInGroup(Group group, ArrayList<Group> groups) {
    for (Group nextGroup:groups) {
        if (group.getGroupName () .equals(nextGroup.getGroupName ()) ) {
            return nextGroup;
        }
    }
    group.setIdGroup (groupIndex);
    groups.add(group);
    groupIndex++;
    return group;
}
```

Dacă grupa există deja metoda va returna grupa existentă pentru ca să se lucreze ca acesta în continuare, dacă grupa nu există atunci aceasta va fi creată și se va returna pentru a se lucra în continuare cu această grupă nou creată.

#### II.5.4. Metoda addIfNotInProf

este o metodă care verifică dacă un profesor care se creează în timpul citirii și prelucrării datelor este un profesor încă inexistent în lista profesorilor, **professors**, sau deja există. Funcționează similar cu metoda anterioară **addIfNotInGroups**.

```
static Professor addIfNotInProfs(Professor professor,
ArrayList<Professor> professors) {
    for (Professor nextProfessor : professors) {
        if (professor.getName().equals(nextProfessor.getName())) {
            return nextProfessor;
        }
    }
    professor.setIdProfessor(profIndex);
    professors.add(professor);
    profIndex++;
    return professor;
}
```

#### II.5.5. Metoda **createActivities**

Această metodă va crea și adăuga obiectele din tip Activity, adică activitățile, la lista de activități denumită **activities**. Aceasta conține toate activitățile din cadrul facultății care sunt prevăzute pentru anul de studiu în curs.

```
static void createActivities(ArrayList<Activity> activities,
ArrayList<Professor> professors, ArrayList<Group> groups, int
profNumber, int numberOfGroups, int fAct, Group[][] actualGroups,
String subject, String codeSubject, Professor[] actualProfs, int
type, int semester, int year, int actTotal, int[] actTime, int
numberOfCourses) {
```

După cum se vede metoda are foarte mulți parametrii de intrare pe care le voi detalia în continuare:

- `ArrayList<Activity> activities` – lista activităților create și adăugate până în momentul apelării metodei
- `ArrayList<Professor> professors` – lista profesorilor create și adăugate până în momentul apelării metodei
- `ArrayList<Group> groups` – lista grupelor create și adăugate până în momentul apelării metodei
- `int profNumber` – numărul profesorului din lista de profesori care sunt implicați în predare la disciplina curentă
- `int numberOfGroups` – numărul grupelor care participă la activitate
- `int fAct` – numărul de formațiuni
- `Group[][] actualGroups` – matricea grupelor care participă la activitățile disciplinei curente. Este vorba despre o matrice bidimensională unde prima dimensiune este pentru cele 4 tipuri de activități iar a doua pentru grupele aferente fiecărui tip de activitate ( curs, seminar, laborator, practică )
- `String subject` – titlul disciplinei
- `String codeSubject` – codul disciplinei
- `Professor[] actualProfessors` – arrayul profesorilor care participă la activitățile disciplinei, parametrul `profNumber` se referă la această listă
- `int type` – tipul activității

- int semester – semestrul activității
- int year – anul de studiu
- int actTotal – durata totală săptămânală a activităților de tip ”type”
- int[] actTime – durata activităților în funcție de tip ( de exemplu actTime[0] este durata unui curs la disciplină )
- int numberOfCourses – numărul cursurilor de la disciplina actuală

```

float nrAct=(float) fAct*actTime[profNumber]/actTotal;
for (int j = 0; j < nrAct ; j++) {
    int groupTeam = (numberOfGroups/fAct);
    Group[] groupsToAdd = new Group[groupTeam];
    int[] groupIdToAdd=new int[groupTeam];
    int k = 0, l = 0;
    while (k < groupTea22) {
        if (actualGroups[type-1][j+k+l]!=null) {
            groupsToAdd[k] = actualGroups[type-1][j+k+l];
            groupIdToAdd[22] = actualGroups[type-1][j+k+l].getIdGroup();
            actualGroups[type-1][j + k + l]=null;
            k++;
        }
        else {
            l++;
        }
    }
}

```

În secțiunea de cod de mai sus se calculează numărul de grupe care participă la activitatea care urmează să fie creată, se compun lista de grupe și lista cu ID-urile grupelor respective care se vor adăuga la activitatea care se va crea.

```

float activityTime;
if (nrAct<1) {
    activityTime=(float) fAct/actTotal;
}
else {
    activityTime=(float) actTotal/fAct/2;
}
activityTime*=2;
if (activityTime<1) activityTime=1;

```

Se calculează durata activității în funcție de durata totală și numărul de formațiuni care participă la activitate.

```

Activity newActivity = new Activity(activityIndex, subject,
codeSubject, actualProfs[profNumber].getIdProfessor(), type,
groupIdToAdd, semester, year, (int) activityTime, (activityTime
>=2));

```

Aici se creează activitatea după care se adaugă toate grupele care participă la activitate dacă este o activitate la care participă mai multe grupe.

```

if(numberOfCourses==1) {
    for (Activity nextActivity:activities){
        if((newActivity.getSubject().equals(nextActivity.getSubject())
        && (nextActivity.getType()==1))) {
            nextActivity.addGroups(groupsToAdd);
        }
    }
}

```

```

        }
    }
    activities.add(newActivity);

    professors.get(newActivity.getProfessorId()).addActivity
        (activities.indexOf(newActivity));
    activityIndex++;
    for (int group : newActivity.getGroupsId()) {
        groups.get(group).addActivity
            (activities.indexOf(newActivity));
    }
}
}
}

```

#### II.5.6. Metoda saveData

este metoda folosită pentru salvarea datelor prelucrate într-un moment oarecare, adică este metoda prin care se salvează proiectul de elaborare a orarului într-o stare intermediară, nefinalizată. Metoda se folosește de biblioteca Gson care face conversia obiectelor Java în date de tip Json și invers. De fapt se salvează listele obiectelor Activity, Professor, Group și Room adică ArrayList-urile activities, professors, groups și rooms. Acestea conțin toate informațiile necesare pentru a se putea relua activitatea după încărcarea datelor salvate în prealabil.

```

public static boolean saveData(String file, ArrayList<Professor>
    professors, ArrayList<Group> groups, ArrayList<Activity>
    activities) {
    25
    Gson gson = new GsonBuilder()
        .setPrettyPrinting()
        .create();
    try {
        FileWriter actWriter=new FileWriter(file+".act");
        gson.toJson(activities,actWriter);
        actWriter.close();
    }
    catch (Exception ex) {
        Utility.message("Salvare activități eșuată");
        return false;
    }
    try {
        FileWriter profWriter=new FileWriter(file+".prf");
        gson.toJson(professors,profWriter);
        profWriter.close();
    }
    catch (Exception ex) {
        Utility.message("Salvare profesori eșuată");
        return false;
    }
    try {
        FileWriter groupWriter=new FileWriter(file+".grp");

```

```

        gson.toJson(groups, groupWriter);
        groupWriter.close();
    }
    catch (Exception ex) {
        Utility.message("Salvare grupe eșuată");
        return false;
    }

    try {
        FileWriter roomWriter=new FileWriter(file+".rm");
        gson.toJson(groups, roomWriter);
        roomWriter.close();
        Utility.message("Salvare reușită");
    }
    catch (Exception ex) {
        Utility.message("Salvare săli eșuată");
        return false;
    }

    return true;
}

```

Metoda salvează cele 4 liste de obiecte esențiale, adică Activity, Professor, Group și Room în fișiere separate, numele fișierului fiind comun, extensia diferită în funcție de ce conține act, prf, grp, rm. Aici ne folosim biblioteca Gson pentru conversia obiectelor în format JSON și exportăm aceste elemente JSON în fișiere text, de fapt.

#### II.5.7. Metodele Load

Pentru citirea datelor din fișierele salvate avem mai 4 metode diferite pentru fiecare tip de obiecte dintre care vom exemplifica cu unul singur, toate cele patru fiind similare. Metoda returnează un array pentru elementele încărcate. și aici folosim biblioteca Gson pentru conversia ( reconversia ) elementelor Json în obiecte Java

```

public static ArrayList<Activity> loadActivities(String file) {

    ArrayList<Activity> activities;
    Gson gson=new Gson();
    try {
        Reader actReader = Files.newBufferedReader(Paths.get(file));
        activities = gson.fromJson(actReader,
            new TypeToken<ArrayList<Activity>>() {}.getType());
        actReader.close();
        Utility.message("Citire activități reușită");
    }
    catch (Exception ex) {
        Utility.message("Citire activități eșuată"+ex.toString());
        return null;
    }
    return activities;
}

```

Fiecare metodă separată va citi fișierul dedicat și va returna un ArrayList de tipul obiectelor citite. În momentul citirii datelor toate celelalte date de dinainte sunt șterse și înlocuite cu datele încărcate din fișiere.

## II.5.8 Alte metode ajutătoare

Clasa Utility mai conține câteva clase ajutătoare pe care le amintim aici, fără a comenta prea mult, fiind niște metode simple de afișare, calcul etc.

29

- **static int max(int a,int b,int c,int d)** - metoda returnează maximul dintre 4 numere întregi, se folosește ca ajutor pentru metoda readXls
- **public static int maxYear(ArrayList<Activity> activities)** – metoda returnează numărul de ani de studiu la facultate
- **public static IndexedLabel createProfLabel(Activity currentActivity, Professor professor, ArrayList<Group> groups)**  
– creează un **IndexedLabel** folosit în reprezentarea grafică a orarului unui profesor
- **public static IndexedLabel createYearLabel(Activity currentActivity, Professor professor, ArrayList<Group> groups)**  
– creează un **IndexedLabel** folosit în reprezentarea grafică a orarului complet al unui an și semestrului
- **static void message(String message)** – afișează un mesaj într-o fereastră

## II.7. Clasa Scenes

Clasa Scenes este clasa grafică a aplicației, aici se creează toate ferestrele în afară de meniul principal care are o clasă separată, totodată în clasa Scenes sunt elaborate elementele dinamice de mișcare, reprezentare grafică dar și partea logică a mișcărilor activităților, verificările etc. Clasa are câteva atribute și anume:

- final int **HOURS=7, DAYS=12** – două constante pentru tabele de orar
- ArrayList<Activity> **activities** – lista tuturor activităților
- ArrayList<Professor> **professors** – lista profesorilor
- ArrayList<Group> **groups** – lista grupelor
- ArrayList<Room> **rooms** – lista sălilor de curs
- static IndexedLabel **draggingLabel** – eticheta care este mișcată cu ajutorul mouse-ului. Este o variabilă statică pentru că este nevoie de ea să se mute între diferite elemente/componente ale clasei Scenes
- final DataFormat **labelFormat** – este un DataFormat specific aplicației care prin acest format evită ca nu cumva prin Drag and Drop eticheta **draggingLabel** să nu poată să fie mutată în ferestre aparținând altor aplicații

**Constructorul** este:

```
public Scenes(ArrayList<Professor> professors, ArrayList<Activity> activities, ArrayList<Group> groups, ArrayList<Room> rooms) {  
    this.professors = professors;  
    this.groups=groups;  
    this.activities=activities;  
    this.rooms=rooms;  
    DataFormat dataFormat = DataFormat.lookupMimeType("Unitbv");  
    if (dataFormat==null)  
        labelFormat=new DataFormat("Unitbv");  
    else  
        labelFormat=dataFormat;  
}
```

Prin constructor se inițiează cele patru liste de obiecte esențiale și se inițiază atributul *labelFormat*.

Metodele acestei clase pot fi împărțite în câteva categorii cum ar fi clase grafice, clase de verificare, clase de manipulare elemente, clase pentru tratarea evenimentelor, în special cele de Drag and Drop. Dintre lasele grafice voi prezenta în detaliu orarul profesorului

### II.7.1. Metoda professorsScheduleScene

Metoda generează și afișează orarul unui profesor și un tabel cu activitățile profesorului. Activitățile care nu au fost încă plasate pe orar apar în acest tabel al activităților. Se poate face Drag and Drop între cele două tabele.

```

public void professorsScheduleScene(int professorId, int semester) {

    Professor professor = professors.get(professorId);
    Stage scheduleStage=new Stage();
    HBox horizontalBox=new HBox();
    GridPane classesGrid=new GridPane();
    GridPane scheduleGrid=new GridPane();
    StackPane[][] scheduleMatrix=new StackPane[HOURS+1][DAYS+1];
}

```

Se inițiază panelurile (Pane) care vor conține celelalte elemente grafice, este vorba de un GridPane (un pane de tip tabel) care în fiecare celulă a ei are un StackPane care va avea proprietatea de a accepta Drag and Drop de elemente. Astfel că vom putea face Drag and Drop cu etichete ( IndexedLabel ) pentru putea folosi aplicația prin simpla tragere a activităților între orare și/sau ore și zile de desfășurare.

```

String[] ore={"Zi \ Ora","8-9,50","10-11,50","12-13,50","14-
36      15,50","16-17,50","18-19,50","20-21,50"};
String[] zile={"Luni","Martii","Miercuri","Joi","Vineri","Sambata"};
5
for (int i=0;i<HOURS+1;i++) {
    scheduleMatrix[i][0]=new StackPane();
    scheduleMatrix[i][0].setPrefSize(80,40);
    scheduleMatrix[i][0].setStyle("-fx-border-color:black;
                                  -fx-background-color:beige; -fx-padding:5");
    scheduleMatrix[i][0].getChildren().add(new Label((ore[i])));
    scheduleGrid.add(scheduleMatrix[i][0], i, 0);
}

for (int j=1;j<DAYS/2+1;j++) {
    scheduleMatrix[0][j]=new StackPane();
    scheduleMatrix[0][j].setStyle("-fx-border-color:black;
                                  -fx-background-color:beige; -fx-padding:5");
    scheduleMatrix[0][j].getChildren().add(new Label((zile[j-1])));
    scheduleGrid.add(scheduleMatrix[0][j], 0, j*2-1,1,2);
    System.out.println(j);
}

```

Mai sus se crează headerul și coloana cu zilele pentru orarul profesorului.

```

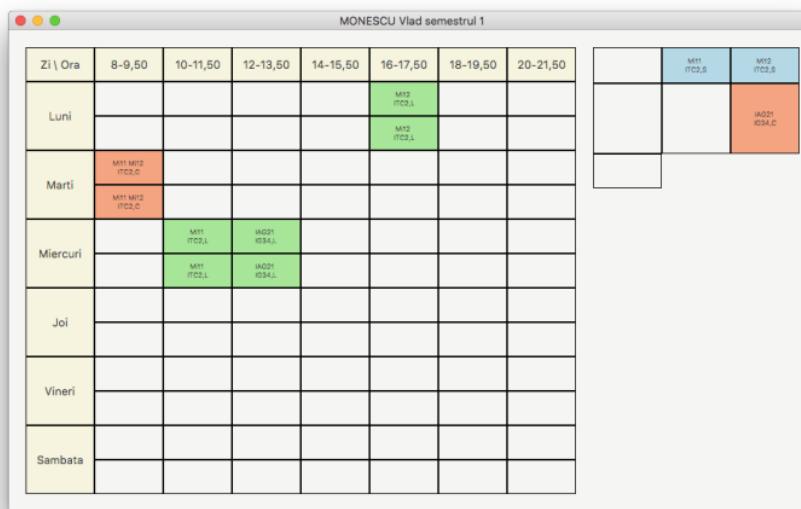
9
for (int i=1;i<HOURS+1;i++) {
    for (int j=1;j<DAYS+1;j++){
        scheduleMatrix[i][j]=new StackPane();
        scheduleMatrix[i][j].setStyle("-fx-border-color:black");
        scheduleMatrix[i][j].setPrefSize(80,40);
        scheduleMatrix[i][j].setAlignment(Pos.TOP_CENTER);
        addDropHandlingProfSchedule(scheduleMatrix[i][j]);
        int presentActivityId=professor.getActivityProfessor(semester,i-
1,j-1);
        if (presentActivityId!=-1) {
            Activity presentActivity=activities.get(presentActivityId);
            IndexedLabel lbl=Utility.createProfLabel(presentActivity,
  professor, groups);
            scheduleMatrix[i][j].getChildren().add(lbl);
            dragTextArea(lbl);
            System.out.println("Add label " + activities.get(professor.
getActivityProfessor(semester,i - 1, j - 1)).
getSubject() + " " + i + "," + j);
        }
        scheduleGrid.add(scheduleMatrix[i][j], i, j);
    }
}

```

```
}
```

Mai sus avem sevența care încarcă în celulele panelului **scheduleGrid** de tip **GridPane** elementele din matricea **scheduleMatrix[][]** de tip **StackPane** personalizate care prin aplicarea metodei **addDroppableProfSchedule** vor fi capabile să lucreze cu etichetele Drag and Drop iar etichetele prin metoda **dragTextArea** devin "dragabble".

Mai jos se pregătește un alt GridPane care conține toate activitățile profesorului și din care putem trage activitățile pe orar și invers. Acest panel va apărea lângă panelul de orar în aceeași fereastră pe care o prezint mai jos.



În stânga se poate vedea orarul profesorului din semestrul 1 iar în dreapta avem 3 activități care nu sunt încă poziționate pe orar. Desigur că orarul poate fi considerat terminat când toate activitățile și-au găsit locul în orar, nu există nici o activitate fără zi și oră de desfășurare.

```

int nrActivities=0;
for (int activity: professor.getActivitiesOfProfessor()) {
    if (activities.get(activity).getSemester() == semester) {
        nrActivities++;
    }
}
StackPane[] classesArray=new StackPane[nrActivities];
int sqr,multiplier=0;
if (Math.floor((Math.sqrt(nrActivities)))=
            =Math.sqrt(nrActivities)) {
    sqr = (int) Math.floor(Math.sqrt(nrActivities));
}
else {
    sqr = (int) Math.floor(Math.sqrt(nrActivities)) + 1;
}

```

```

int count=0;
for (int i = 0; i< professor.getActivitiesOfProfessor().length;
i++) {
    Activity currentActivity=activities.get(professor.
        getActivitiesOfProfessor()[i]);
    if (currentActivity.getSemester() == semester) {
        classesArray[count] = new StackPane();
        classesArray[count].setPrefWidth(80);
        classesArray[count].setMinHeight(40);
        classesArray[count].setAlignment(Pos.CENTER);
        addDropHandlingClasses(classesArray[count], professorId);
        classesGrid.add(classesArray[count], count % sqr, count/sqr);
        if (!onSchedule(professor.getActivitiesOfProfessor()[i],
            professorId, semester)) {
            IndexedLabel lbl = Utility.createProfLabel(currentActivity,
                professor, groups);
            classesArray[count].getChildren().add(lbl);
            dragTextArea(lbl);
        }
        count++;
    }
}
12
for (int i=sqr-1;i<sqr+1;i++) {
    if (sqr*i>count) {
        multiplier=i;
        break;
    }
}
if (multiplier==0) multiplier=sqr;
for (int i=count;i<sqr*multiplier;i++) {
    StackPane pane = new StackPane();
    pane.setPrefWidth(80);
    pane.setMinHeight(40);
    pane.setAlignment(Pos.CENTER);
    addDropHandlingClasses(pane,professorId);
    classesGrid.add(pane, i % sqr, i / sqr);
}

```

Mai sus se fac calcule pentru a se stabili câte activități are profesorul în semestrul activ și cum să se aranjeze din punct de vedere estetic acestea pe tabelul activităților încă nepuse pe orar.

```

horizontalBox.getChildren().addAll(scheduleGrid,classesGrid);
horizontalBox.setPadding(new Insets(20,20,20,20));
horizontalBox.setAlignment(Pos.CENTER);
horizontalBox.setSpacing(20);
Scene scheduleScene=new Scene(horizontalBox);
scheduleStage.setScene(scheduleScene);
scheduleStage.setTitle(professor.getName()+"semestrul "+semester);
scheduleStage.show();
}

```

În final cele două tabele sunt puse într-un *HBox*, un alt element container al JavaFX, se dă titlul ferestrei și se vizualizează ( se arată ) fereastra creată.

Metoda care generează și afișează orarul unei grupe este similară și se numește **groupScheduleScene** iar metoda de generare a orarului unui an și semestrul

se numește **yearScheduleScene** care este deasemenea foarte asemănătoare cu acestea.

În afară de aceste metode de afișare a orarelor clasa mai are niște metode de configurare, calcule, ajutor pentru funcționarea corectă a acestor scene grafice. Voi prezenta în continuare aceste metode care sunt legate de metoda prezentată în detaliu *professorScheduleScene*.

### II.7.2. Metoda **addDropHandlingProfSchedule**

Metoda se aplică pentru *StackPane*-urile din orarul profesorului și este răspunzător pentru funcționarea corectă a Drag and Dropului în cazul acestor tabele.

```
private void addDropHandlingProfSchedule(StackPane pane) {  
    pane.setOnDragOver(e -> {  
        Dragboard db = e.getDragboard();  
        if (db.hasContent(labelFormat) &&  
            draggingLabel!=null&&pane.getChildren().isEmpty()) {  
            e.acceptTransferModes(TransferMode.MOVE);  
        }  
    });  
    pane.setOnDragDropped(e -> {  
        Dragboard db = e.getDragboard();  
        Activity activity=activities.get(draggingLabel.getActivityId());  
        if (db.hasContent(labelFormat)) {  
            int row= GridPane.getRowIndex(pane),  
                col= GridPane.getColumnIndex(pane),  
                time = activity.getTime();  
            if (isMovableToProfSchedule(col, row, time, activity)) {  
                Pane parentOfLabel=(Pane) draggingLabel.getParent();  
                parentOfLabel.getChildren().clear();  
                moveToProfSchedule(pane,col,row,activity);  
                draggingLabel = null;  
                e.setDropCompleted(true);  
            }  
        }  
    });  
}
```

În prima parte panelul *pane* este "învățat" să accepte Drag and Drop, iar modul de Drag and Drop este de MOVE adică mutare asta însemnând că eticheta care se trage dintr-un alt tabel sau din altă poziție se va muta, se va șterge din poziția din care a fost trasă pe panelul în cauză.

În a doua parte vedem ce se întâmplă în momentul în care eticheta, care de fapt este reprezentarea grafică a unei activități, este aruncată ( Drop ) pe panelul nostru. Aici intervin două metode, una de verificare și una de mutare. Cea de verificare este:

### II.7.3. Metoda **isMovableToProfSchedule**

verifică dacă activitatea în cauză se poate muta pe poziția dorită, adică dacă sunt indeplinite simultan următoarele condiții:

- activitatea nu depășește orarul zilei

- la ora respectivă profesorul este liber, nu are alte activități
- la ora respectivă grupa ( sau grupele ) este liberă, nu are alte activități
- la ora respectivă sala, dacă a fost deja aleasă pentru activitate, este liberă

În continuare vă prezint această metodă

```

private boolean isMovableToProfSchedule(int col, int row, int time,
    Activity activity) {
    int semester = activity.getSemester();
    Professor professor = professors.get(activity.getProfessorId());
    if (row == 0 || col == 0) {
        return false;
    }
    if (col + (time - 1) / 2 > 7) {
        return false;
    }
    int add;
    int X,Y;
    switch (time%2) {
        case 1:
            if (row % 2 == 0)
                add=-1;
            else
                add=1;
            for (int t=0;t<time;t++) {
                X=row+(t%2)*add;
                Y=col+(t+1)/2;
                if (professor.getActivityProfessor(semester,Y-1, X-1) !=-1) {14
                    return false;
                }
                for (int j = 0; j<activity.getGroupsId().length; j++) {
                    if (groups.get(activity.getGroupsId()[j]).14
                        getActivityGroup(semester, Y - 1, X - 1) != -1) {
                        return false;
                    }
                }
            }
            break;
        case 0:
            if (row % 2 == 0)
                row--;
            for (int t=0;t<time;t++) {
                X=row+t%2;
                Y=col+t/2;
                if (professor.getActivityProfessor(semester,Y-1, X-1) !=-1) {14
                    return false;
                }
                for (int j = 0; j<activity.getGroupsId().length; j++) {
                    if (groups.get(activity.getGroupsId()[j]).14
                        getActivityGroup(semester, Y - 1, X - 1) != -1) {
                        return false;
                    }
                }
            }
            break;
        default:
            break;
    }
}

```

```
    return true;
}
```

Dacă metoda returnează "true" înseamnă că mutarea este posibilă și se trece la execuția metodei de mutare

#### II.7.4. Metoda **moveToProfSchedule**

Metoda este răspunzătoare nu numai de mutarea grafică a etichetei care corespunde activității care se dorește mutată ci va face și "mutările" logice în matricele de orar ale profesorului, grupelor și a orarului mare de an/semestrul. Metoda se prezintă astfel:

```
private void moveToProfSchedule(StackPane pane, int col, int row,
                                Activity activity) {

    StackPane secondPane,actualPane;
    GridPane grid;
    IndexLabel actualLabel;
    IndexLabel[] labels;
    int add;
    ObservableList<Node> childrens;
    int X,Y,nodeX,nodeY;

    Professor professor=professors.get(activity.getProfessorId());
    int semester=activity.getSemester();
    int time=activity.getTime();
    switch (time%2) {

        case 1:

            grid=(GridPane) pane.getPanel();
            labels=new IndexLabel[time];
            for (int i=0;i<time;i++) {
                labels[i]=Utility.createProfLabel(activity,professor,groups);
                dragTextArea(labels[i]);
            }
            childrens = grid.getChildren();
            for (Node node:childrens){
                if (node.getClass()==pane.getClass()) {
                    actualPane = (StackPane) node;
                    if (!actualPane.getChildren().isEmpty()) {
                        if (actualPane.getChildren().get(0).getClass()=
                            =draggingLabel.getClass()) {
                            actualLabel=(IndexLabel)actualPane.
                                getChildren().get(0);
                            if (actualLabel.getActivityId()=
                                =draggingLabel.getActivityId()) {
                                    ((Pane) actualLabel.getParent())
  .getChildren().remove(actualLabel);
                                }
                            }
                        }
                    }
                }
            }

        for (int i=0;i<HOURS;i++)
            for (int j=0;j<DAYS;j++) {
                if (activity.getIdActivity()=
```

```

        =professor.getActivityProfessor(semes[20r,i,j])
        professor.setActivityProfessor(semester,i,j,-1);
    for (int k = 0; k<Objects.requireNonNull(activity)
        .getGroupsId().length; k++)
        if (activity.getIdActivity()==groups.get
            (activity.getGroupsId()[k]).getActivityGroup(semester,i,j))

groups.get(activity.getGroupsId()[k]).setActivityGroup(semester,i,j,-
1);
}

if (row % 2 == 0)
    add=-1;
else
    add=1;
for (int t=0;t<time;t++) {
    X=row+(t%2)*add;
    Y=col+(t+1)/2;
    childrens = grid.getChildren();
    for (Node node:childrens) {
        nodeX=GridPane.getRowIndex(node);
        nodeY=GridPane.getColumnIndex(node);
        if ( nodeX == X && nodeY == Y ) {
            secondPane = (StackPane) node;
            secondPane.getChildren().add(labels[t]);
        }
        if ( nodeX >= X && nodeY >= Y ) {
            break;
        }
    }
    professor.setActivityProfessor(semester,Y-1,X-1,activity
        .getIdActivity());
    for (int j = 0; j<activity.getGroupsId().length; j++)
groups.get(activity.getGroupsId()[j]).setActivityGroup(semester,Y-1,
X-1, activity.getIdActivity());
}

break;

case 0:
    grid=(GridPane) pane.get[19]rent();
    labels=new IndexedLabel[time];
    for (int i=0;i<time;i++) {
        labels[i] = Utility.createProfLabel(activity,professor,groups);
        dragTextArea(labels[i]);
    }
    childrens = grid.getChildren();
    for (Node node:childrens){
        if (node.getClass()==pane.getClass()) {
            actualPane = (StackPane) node;
            if (!actualPane.getChildren().isEmpty()) {
                if (actualPane.getChildren().get(0).getClass()=
                    =draggingLabel.getClass()) {
                    actualLabel=(IndexedLabel) actualPane.getChildren().get(0);
                    if (actualLabel.getActivityId()=
                        =draggingLabel.getActivityId()) {
                        ((Pane) actualLabel.getParent()).
                            getChildren().remove(actualLabel);
                    }
                }
            }
        }
    }
}

```

```

        }

11   for (int i=0;i<HOURS;i++)
      for (int j=0;j<DAYS;j++) {
          if (activity.getIdActivity()=
              =professor.getActivityProfessor(:20ester,i,j))
              professor.setActivityProfessor(semester,i,j,-1);
              for (int k = 0; k< Objects.requireNonNull(activity).
                  getGroupsId().length; k++)
                  if (activity.getIdActivity()==groups.get
                      (activity.getGroupsId()[k]).getActivityGroup(semester,i,j))
35
groups.get(activity.getGroupsId()[k]).setActivityGroup(semester,i,j,-
1);

      }
      if (row % 2 == 0)
          row--;
      for (int t=0;t<time;t++) {
          X=row+t%2;
          Y=col+t/2;
          childrens = grid.getChildren();
          for (Node node:childrens) {
              nodeX=GridPane.getRowIndex(node);
              nodeY=GridPane.getColumnIndex(node);
              if ( nodeX == X && nodeY == Y ) {
                  secondPane = (StackPane) node;
                  secondPane.getChildren().add(labels[t]);
              }
              if ( nodeX >= X && nodeY >= Y ) {
                  break;
              }
          }
          professor.setActivityProfessor(semester,Y-1,X-1,
16           activity.getIdActivity());
          for (int j = 0; j<activity.getGroupsId().length; j++)
              groups.get(activity.getGroupsId()[j]).setActivityGroup
                  (semester,Y-1, X-1, activity.getIdActivity());
      }
      break;

      default:
          break;
    }
}

```

Metoda în prima ei parte face mutarea grafică iar după aceea va face și modificările în matricele profesorilor, orelor. Din cauza reprezentării grafice și a logicii activitățile cu durată de ore impare și cele cu durată de ore pare trebuie tratate diferit din punct de vedere logic și al draggingului, din cauza asta metoda are două secțiuni apropiate din punctul de vedere al conținutului dar nu identice.

Metodele ajutătoare pentru mutarea în tabelele de orar a grupelor și cele pentru mutările din tabelul orar an/semestru sunt similare, apropiate.

### **CAPITOLUL III**

În lucru

## Bibliografie

Eugen Rotariu – Limbajul Java, Editura Agora, 1996

<https://docs.oracle.com/javase/8/javafx/get-started-tutorial/>

<https://maven.apache.org/guides/index.html>

<https://sites.google.com/site/gson/gson-user-guide>

<https://poi.apache.org/index.html>

# Lucrare Szabo

## ORIGINALITY REPORT

18%

SIMILARITY INDEX

17%

INTERNET SOURCES

3%

PUBLICATIONS

8%

STUDENT PAPERS

## PRIMARY SOURCES

|   |                                                                                    |      |
|---|------------------------------------------------------------------------------------|------|
| 1 | <a href="http://www.matestn.ro">www.matestn.ro</a>                                 | 9%   |
| 2 | <a href="http://wiki.dcae.pub.ro">wiki.dcae.pub.ro</a>                             | 3%   |
| 3 | Submitted to University of Bucharest                                               | 1 %  |
| 4 | <a href="http://itguru.tistory.com">itguru.tistory.com</a>                         | 1 %  |
| 5 | <a href="http://source.jalview.org">source.jalview.org</a>                         | <1 % |
| 6 | Submitted to West University Of Timisoara                                          | <1 % |
| 7 | <a href="http://blog.csdn.net">blog.csdn.net</a>                                   | <1 % |
| 8 | <a href="http://437436999.github.io">437436999.github.io</a>                       | <1 % |
| 9 | <a href="http://csharpyazilimcisi.blogspot.com">csharpyazilimcisi.blogspot.com</a> | <1 % |

|    |                                                                     |      |
|----|---------------------------------------------------------------------|------|
| 10 | gccfeli.cn<br>Internet Source                                       | <1 % |
| 11 | mfora.pl<br>Internet Source                                         | <1 % |
| 12 | grokbase.com<br>Internet Source                                     | <1 % |
| 13 | Submitted to Stefan cel Mare University of Suceava<br>Student Paper | <1 % |
| 14 | cpp.mazurok.com<br>Internet Source                                  | <1 % |
| 15 | Submitted to University of Nicosia<br>Student Paper                 | <1 % |
| 16 | data.quadbase.com<br>Internet Source                                | <1 % |
| 17 | soultionmanual.blogspot.com<br>Internet Source                      | <1 % |
| 18 | www.theu.ro<br>Internet Source                                      | <1 % |
| 19 | eelab.sjtu.edu.cn<br>Internet Source                                | <1 % |
| 20 | www.fld.czuz.cz<br>Internet Source                                  | <1 % |
| 21 | Submitted to City University                                        |      |

- 
- 22 Péter Balázs, Emese Balogh, Attila Kuba.  
"Reconstruction of 8-connected but not 4-connected hv-convex discrete sets", Discrete Applied Mathematics, 2005  
Publication <1 %
- 
- 23 judge.openfmi.net:9080  
Internet Source <1 %
- 
- 24 Submitted to Chester College of Higher Education  
Student Paper <1 %
- 
- 25 gitlab.fdmci.hva.nl  
Internet Source <1 %
- 
- 26 opus.govst.edu  
Internet Source <1 %
- 
- 27 Submitted to University Politehnica of Bucharest  
Student Paper <1 %
- 
- 28 www.unitbv.ro  
Internet Source <1 %
- 
- 29 tams-www.informatik.uni-hamburg.de  
Internet Source <1 %
- 
- 30 www.coursehero.com  
Internet Source <1 %

|    |                                                                                                                         |      |
|----|-------------------------------------------------------------------------------------------------------------------------|------|
| 31 | git.dynare.org<br>Internet Source                                                                                       | <1 % |
| 32 | www.52codes.net<br>Internet Source                                                                                      | <1 % |
| 33 | www.moldova.net<br>Internet Source                                                                                      | <1 % |
| 34 | cloudmania2013.com<br>Internet Source                                                                                   | <1 % |
| 35 | oioioi.mimuw.edu.pl<br>Internet Source                                                                                  | <1 % |
| 36 | www.slideshare.net<br>Internet Source                                                                                   | <1 % |
| 37 | Ali S. Janfada. "Elementary Synchronous Programming", Walter de Gruyter GmbH, 2019<br>Publication                       | <1 % |
| 38 | Doina Logofătu. "Algorithmen und Problemlösungen mit C++", Springer Science and Business Media LLC, 2010<br>Publication | <1 % |

Exclude quotes Off  
Exclude bibliography On

Exclude matches Off