PHASE 3
Coding Document

UniGo

Developed by Saien Naidu
Grade 12
Information Technology
2025

Table of Contents

1. Object Classes

1.1 University

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Objects;

/**
 *
 * @author Saien Naidu
 */
// This class represents the University Object
public class University {

    // FIELDS
    private int ID;
    private String name;
    private String desc;
    private String location;
    private int rank;
    private int estb;
    private int students;
    private double accRate;

    // CONSTRUCTOR
    /**
     * Instantiate a new University Object
     *
     * @param ID
     * @param name
     * @param desc
     * @param location
     * @param rank
     * @param estb
     * @param students
     * @param accRate
     */
    public University(int ID, String name, String desc, String location,
            int rank, int estb, int students, double accRate) {
        this.ID = ID;
        this.name = name;
        this.desc = desc;
        this.location = location;
        this.rank = rank;
        this.estb = estb;
        this.students = students;
        this.accRate = accRate;
    }

    // GETTERS & SETTERS
```

```
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public int getRank() {
        return rank;
    }

    public void setRank(int rank) {
        this.rank = rank;
    }

    public int getEstb() {
        return estb;
    }

    public void setEstb(int estb) {
        this.estb = estb;
    }

    public int getStudents() {
        return students;
    }

    public void setStudents(int students) {
        this.students = students;
    }

    public double getAccRate() {
        return accRate;
    }

    public void setAccRate(double accRate) {
        this.accRate = accRate;
    }
}
```

1.2 Faculty

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
/package Objects;

/**
 *
 * @author Saien Naidu
 */
// This class represents the Faculty Object
public class Faculty {

    // FIELDS
    private int ID;
    private String name;
    private String desc;
    private University uni;

    // CONSTRUCTOR
    /**
     * Instantiate a new Faculty Object
     *
     * @param ID
     * @param name
     * @param desc
     * @param uni
     */
    public Faculty(int ID, String name, String desc, University uni) {
        this.ID = ID;
        this.name = name;
        this.desc = desc;
        this.uni = uni;
    }

    // GETTERS & SETTERS
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesc() {
```

```
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public University getUni() {
        return uni;
    }

    public void setUni(University uni) {
        this.uni = uni;
    }
}
```

1.3 Degree

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Objects;

/**
 *
 * @author Saien Naidu
 */
// This class represents the Degree Object
public class Degree {

    // FIELDS
    private int degreeID;
    private String name;
    private University uni;
    private Faculty fac;
    private String desc;

    // CONSTRUCTOR
    /**
     * Instantiate a new Degree Object
     *
     * @param degreeID
     * @param name
     * @param uni
     * @param fac
     * @param desc
     */
    public Degree(int degreeID, String name, University uni, Faculty fac, String desc) {
        this.degreeID = degreeID;
        this.name = name;
        this.uni = uni;
```

```
        this.fac = fac;
        this.desc = desc;
    }

    // GETTERS & SETTERS
    public int getDegreeID() {
        return degreeID;
    }

    public void setDegreeID(int degreeID) {
        this.degreeID = degreeID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public University getUni() {
        return uni;
    }

    public void setUni(University uni) {
        this.uni = uni;
    }

    public Faculty getFac() {
        return fac;
    }

    public void setFac(Faculty fac) {
        this.fac = fac;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }
}
```

1.4 Requirement

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

```
package Objects;

/**
 *
 * @author Saien Naidu
 */
// This class represents the Requirement Object
public class Requirement {

    // FIELDS
    private int ID;
    private Degree deg;
    private int hlMark;
    private String hlChoice;
    private int falMark;
    private String falChoice;
    private int mathMark;
    private String mathChoice;
    private int opt1Mark;
    private String opt1Choice;
    private int opt2Mark;
    private String opt2Choice;
    private int opt3Mark;
    private String opt3Choice;
    private int lo;
    private int aps;

    // CONSTRUCTOR
    /**
     * Instantiate a new Requirement Object
     *
     * @param ID
     * @param deg
     * @param hlMark
     * @param hlChoice
     * @param falMark
     * @param falChoice
     * @param mathMark
     * @param mathChoice
     * @param opt1Mark
     * @param opt1Choice
     * @param opt2Mark
     * @param opt2Choice
     * @param opt3Mark
     * @param opt3Choice
     * @param lo
     * @param aps
     */
    public Requirement(int ID, Degree deg, int hlMark, String hlChoice, int falMark,
String falChoice, int mathMark, String mathChoice, int opt1Mark, String opt1Choice, int
opt2Mark, String opt2Choice, int opt3Mark, String opt3Choice, int lo, int aps) {
        this.ID = ID;
        this.deg = deg;
        this.hlMark = hlMark;
        this.hlChoice = hlChoice;
        this.falMark = falMark;
        this.falChoice = falChoice;
        this.mathMark = mathMark;
        this.mathChoice = mathChoice;
        this.opt1Mark = opt1Mark;
        this.opt1Choice = opt1Choice;
        this.opt2Mark = opt2Mark;
        this.opt2Choice = opt2Choice;
        this.opt3Mark = opt3Mark;
        this.opt3Choice = opt3Choice;
        this.lo = lo;
        this.aps = aps;
```

```
    }

    // GETTERS & SETTERS
    public Degree getDeg() {
        return deg;
    }

    public void setDeg(Degree deg) {
        this.deg = deg;
    }

    public int getID() {
        return this.ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public int getHlMark() {
        return hlMark;
    }

    public void setHlMark(int hlMark) {
        this.hlMark = hlMark;
    }

    public String getHlChoice() {
        return hlChoice;
    }

    public void setHlChoice(String hlChoice) {
        this.hlChoice = hlChoice;
    }

    public int getFalMark() {
        return falMark;
    }

    public void setFalMark(int falMark) {
        this.falMark = falMark;
    }

    public String getFalChoice() {
        return falChoice;
    }

    public void setFalChoice(String falChoice) {
        this.falChoice = falChoice;
    }

    public int getMathMark() {
        return mathMark;
    }

    public void setMathMark(int mathMark) {
        this.mathMark = mathMark;
    }

    public String getMathChoice() {
        return mathChoice;
    }

    public void setMathChoice(String mathChoice) {
        this.mathChoice = mathChoice;
    }
```

```
    public int getOpt1Mark() {
        return opt1Mark;
    }

    public void setOpt1Mark(int opt1Mark) {
        this.opt1Mark = opt1Mark;
    }

    public String getOpt1Choice() {
        return opt1Choice;
    }

    public void setOpt1Choice(String opt1Choice) {
        this.opt1Choice = opt1Choice;
    }

    public int getOpt2Mark() {
        return opt2Mark;
    }

    public void setOpt2Mark(int opt2Mark) {
        this.opt2Mark = opt2Mark;
    }

    public String getOpt2Choice() {
        return opt2Choice;
    }

    public void setOpt2Choice(String opt2Choice) {
        this.opt2Choice = opt2Choice;
    }

    public int getOpt3Mark() {
        return opt3Mark;
    }

    public void setOpt3Mark(int opt3Mark) {
        this.opt3Mark = opt3Mark;
    }

    public String getOpt3Choice() {
        return opt3Choice;
    }

    public void setOpt3Choice(String opt3Choice) {
        this.opt3Choice = opt3Choice;
    }

    public int getLo() {
        return lo;
    }

    public void setLo(int lo) {
        this.lo = lo;
    }

    public int getAps() {
        return aps;
    }

    public void setAps(int aps) {
        this.aps = aps;
    }
}
```

1.5 Filter

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Objects;

/**
 *
 * @author Saien Naidu
 */
public class Filter {

    // FIELDS
    private boolean useMarks;

    private boolean kzn;
    private boolean gauteng;
    private boolean eastcape;
    private boolean westcape;
    private boolean freestate;
    private boolean northwest;
    private boolean mpumalanga;
    private boolean limpopo;

    private boolean commerce;
    private boolean engineering;
    private boolean health;
    private boolean law;
    private boolean humanities;
    private boolean sciences;

    private String uni1;
    private String uni2;
    private String uni3;

    private boolean include;
    private boolean exclude;

    // CONSTRUCTOR
    /**
     * Instantiate a new Filter Object
     *
     * @param useMarks
     * @param kzn
     * @param gauteng
     * @param eastcape
     * @param westcape
     * @param freestate
     * @param northwest
     * @param mpumalanga
     * @param limpopo
     * @param commerce
     * @param engineering
```

```
     * @param health
     * @param law
     * @param humanities
     * @param sciences
     * @param uni1
     * @param uni2
     * @param uni3
     * @param include
     * @param exclude
     */
    public Filter(boolean useMarks, boolean kzn, boolean gauteng, boolean eastcape,
boolean westcape, boolean freestate, boolean northwest, boolean mpumalanga, boolean
limpopo, boolean commerce, boolean engineering, boolean health, boolean law, boolean
humanities, boolean sciences, String uni1, String uni2, String uni3, boolean include,
boolean exclude) {
        this.useMarks = useMarks;
        this.kzn = kzn;
        this.gauteng = gauteng;
        this.eastcape = eastcape;
        this.westcape = westcape;
        this.freestate = freestate;
        this.northwest = northwest;
        this.mpumalanga = mpumalanga;
        this.limpopo = limpopo;
        this.commerce = commerce;
        this.engineering = engineering;
        this.health = health;
        this.law = law;
        this.humanities = humanities;
        this.sciences = sciences;
        this.uni1 = uni1;
        this.uni2 = uni2;
        this.uni3 = uni3;
        this.include = include;
        this.exclude = exclude;
    }

    // GETTERS & SETTERS
    public boolean isUseMarks() {
        return useMarks;
    }

    public void setUseMarks(boolean useMarks) {
        this.useMarks = useMarks;
    }

    public boolean isKzn() {
        return kzn;
    }

    public void setKzn(boolean kzn) {
        this.kzn = kzn;
    }

    public boolean isGauteng() {
        return gauteng;
    }

    public void setGauteng(boolean gauteng) {
        this.gauteng = gauteng;
    }

    public boolean isEastcape() {
        return eastcape;
    }

    public void setEastcape(boolean eastcape) {
```

```
        this.eastcape = eastcape;
    }

    public boolean isWestcape() {
        return westcape;
    }

    public void setWestcape(boolean westcape) {
        this.westcape = westcape;
    }

    public boolean isFreestate() {
        return freestate;
    }

    public void setFreestate(boolean freestate) {
        this.freestate = freestate;
    }

    public boolean isNorthwest() {
        return northwest;
    }

    public void setNorthwest(boolean northwest) {
        this.northwest = northwest;
    }

    public boolean isMpumalanga() {
        return mpumalanga;
    }

    public void setMpumalanga(boolean mpumalanga) {
        this.mpumalanga = mpumalanga;
    }

    public boolean isLimpopo() {
        return limpopo;
    }

    public void setLimpopo(boolean limpopo) {
        this.limpopo = limpopo;
    }

    public boolean isCommerce() {
        return commerce;
    }

    public void setCommerce(boolean commerce) {
        this.commerce = commerce;
    }

    public boolean isEngineering() {
        return engineering;
    }

    public void setEngineering(boolean engineering) {
        this.engineering = engineering;
    }

    public boolean isHealth() {
        return health;
    }

    public void setHealth(boolean health) {
        this.health = health;
    }
```

```
    public boolean isLaw() {
        return law;
    }

    public void setLaw(boolean law) {
        this.law = law;
    }

    public boolean isHumanities() {
        return humanities;
    }

    public void setHumanities(boolean humanities) {
        this.humanities = humanities;
    }

    public boolean isSciences() {
        return sciences;
    }

    public void setSciences(boolean sciences) {
        this.sciences = sciences;
    }

    public String getUni1() {
        return uni1;
    }

    public void setUni1(String uni1) {
        this.uni1 = uni1;
    }

    public String getUni2() {
        return uni2;
    }

    public void setUni2(String uni2) {
        this.uni2 = uni2;
    }

    public String getUni3() {
        return uni3;
    }

    public void setUni3(String uni3) {
        this.uni3 = uni3;
    }

    public boolean isInclude() {
        return include;
    }

    public void setInclude(boolean include) {
        this.include = include;
    }

    public boolean isExclude() {
        return exclude;
    }

    public void setExclude(boolean exclude) {
        this.exclude = exclude;
    }

}
```

2. Manager Classes

2.1 UniManager

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Managers;

// IMPORTS
import Driver.dbDriver;
import Objects.University;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Saien Naidu
 */
public class UniManager {

    // FIELDS
    private University[] universities = new University[10];
    private int size = 0;
    private final dbDriver db = new dbDriver();
    private int[] tableArr = new int[10];

    // CONSTRUCTOR
    /**
     * Fetch all the universities from the DB and save it into the array
     */
    public UniManager() {
        try {
            // Fetch data
            ResultSet rs = db.query("SELECT * FROM University_Table;");

            // If there is more data
            while (!rs.isLast()) {
                // Save University into array
                universities[size] = createUni(rs, size);
                // Increment size
                size++;
            }

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #04: Failed while retrieving Universities from
DB.");
        }
```

```
    }

    // PROPERTIES
    // Create and return a University object using a specific row from the DB
    private University createUni(ResultSet rs, int row) {
        try {
            // Prepare the next row of results
            rs.absolute(row + 1);

            // Get the value of each column in the row
            int id = rs.getInt("ID");
            String name = rs.getString("UniversityName");
            String desc = rs.getString("Description");
            String location = rs.getString("Location");
            int rank = rs.getInt("Rank");
            int estb = rs.getInt("Established");
            int students = rs.getInt("Students");
            double accRate = rs.getDouble("AcceptanceRate");

            // Create new University object using the values, then return it
            return (new University(id, name, desc, location, rank, estb, students,
accRate));

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #04: Failed while retrieving Universities from
DB.");
        }
        return null;
    }

    /**
     * Fetch University Object with a specific ID from the array
     *
     * @param ID The ID of the University Object that you want to fetch
     * @return The matching University Object
     */
    public University getUniWithID(int ID) {
        // Go through every entry in the array until a match is found
        for (int i = 0; i < size; i++) {
            int current = universities[i].getID();
            // If match found:
            if (current == ID) {
                // Return matching University object
                return universities[i];
            }
        }
        // If no match is found, return nothing
        return null;
    }

    /**
     * Fetch universities with a similar name compared to the one provided from
     * the DB
     *
     * @param query The query to send to the DB
     * @return The list of University Objects that satisfies the query
     */
    public University[] getUniWithName(String query) {
        try {
            // Fetch results
            ResultSet rs = db.query(query);
            // Instantiate temporary array with size of 0
            University[] temp = new University[size];
            int tempSize = 0;
```

```
            // If there is more data:
            while (!rs.isLast()) {
                // Create a new University object using an entry from the results and
save it to the temporary array
                temp[tempSize] = createUni(rs, tempSize);
                // Increment size
                tempSize++;
                /*
                Error handling:
                If the results are found to be empty:
                (i.e. there are no universities whose name is similar to what is given)
                 */
                if ((tempSize + 1) >= this.size) {
                    // Create a university array that can only hold 1 entry
                    University[] error = new University[1];
                    // Create a useless University object with a name of "None"
                    error[0] = new University(0, "None", null, null, 0, 0, 0, 0);
                    // return this University object and interrupt this method
                    return error;
                    // This means that the word "None" will display on the table
                }
            }

            // Instantiate output array with a size that's equal to the amount of
results
            University[] output = new University[tempSize];
            // Copy array
            for (int i = 0; i < tempSize; i++) {
                output[i] = temp[i];
            }
            // Return output array
            return output;

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #05: Failed while retrieving University with a
specific name from DB.");
        }
        // If there is an error, return null
        return null;
    }

    /**
     * Get the names of all the Universities in the DB This method is used to
     * set the models of the combo boxes in the Filter frame.
     *
     * @return The names of all the University Objects
     */
    public String[] getAll() {
        // Create empty String array with a size equal to the total amount of
universities + 1
        String[] all = new String[size + 1];
        // The first entry is "None", which is what the user will pick if they don't
want to choose a specific University
        all[0] = "None";
        // Save the name of every university to an entry in the string array
        for (int i = 0; i < size; i++) {
            all[i + 1] = universities[i].getName();
        }
        // return the string array
        return all;
    }

    /**
     * Create a table row model using all the names of all the universities
     *
```

```
     * @return A table row model
     */
    public Object[][] createTable() {

        // Instantiate model with length equal to that of the class's university array
        Object[][] data = new Object[size][1];
        // The program will also need to store an array of the IDs of the universities
being used
        int[] tempTableArr = new int[size];

        // Go through each university name and save it to the model array
        // Also go through each university ID and save it to the integer array
        for (int i = 0; i < size; i++) {
            data[i][0] = universities[i].getName();
            tempTableArr[i] = universities[i].getID();
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return the model
        return data;
    }

    /**
     * Create a table row model using all the names of specific universities
     *
     * @param input The array of University Objects to use
     * @return A table row model
     */
    public Object[][] createTable(University[] input) {

        // Instantiate model with size equal to the length of the given university array
        Object[][] data = new Object[input.length][1];
        // The program will also need to store an array of the IDs of the universities
being used
        int[] tempTableArr = new int[input.length];

        // Go through each university name and save it to the model array
        // Also go through each university ID and save it to the integer array
        for (int i = 0; i < input.length; i++) {
            data[i][0] = input[i].getName();
            tempTableArr[i] = universities[i].getID();
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return the model
        return data;
    }

    /**
     * Return the IDs of the universities that were used in the making of the
     * latest table row model
     *
     * @return The list of IDs
     */
    public int[] getTableArr() {
        return tableArr;
    }
}
```

2.2 FacManager

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Managers;

// IMPORTS
import Driver.dbDriver;
import Objects.Faculty;
import Objects.University;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Saien Naidu
 */
public class FacManager {

    // FIELDS
    private Faculty[] faculties = new Faculty[100];
    private int size;
    private final dbDriver db = new dbDriver();
    private final UniManager um = new UniManager();
    private int[] tableArr = new int[100];

    // CONSTRUCTOR
    /**
     * Fetch all the Faculties from the DB and save it into the array
     */
    public FacManager() {
        try {
            // Fetch data
            ResultSet rs = db.query("SELECT * FROM Faculty_Table;");

            // If there is more data
            while (!rs.isLast()) {
                // Save Faculty into array
                faculties[size] = createFac(rs, size);
                // Increment size
                size++;
            }

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #06: Failed while retrieving Faculties from DB.");
        }
    }

    // PROPERTIES
```

```
    // Create and return a Faculty object using a specific row from the DB
    private Faculty createFac(ResultSet rs, int row) {
        try {
            // Prepare the next row of results
            rs.absolute(row + 1);

            // Get the value of each column in the row
            int ID = rs.getInt("ID");
            String name = rs.getString("FacultyName");
            String desc = rs.getString("Description");
            int uni = rs.getInt("UniversityID");

            // Create new Faculty object using the values, then return it
            return (new Faculty(ID, name, desc, um.getUniWithID(uni)));

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #06: Failed while retrieving Faculties from DB.");
        }

        return null;
    }

    /**
     * Fetch a faculty with a specific ID from the array
     *
     * @param ID The ID of the Faculty Object that you want to fetch
     * @return The matching Faculty Object
     */
    public Faculty getFacWithID(int ID) {
        // Go through every entry in the array until a match is found
        for (int i = 0; i < size; i++) {
            int current = faculties[i].getID();
            // If match found:
            if (current == ID) {
                // Return current Faculty object
                return faculties[i];
            }
        }
        // If no match is found, return nothing
        return null;
    }

    /**
     * Create a table row model using all the faculties that belong to a
     * specific university
     *
     * @param uni The University Object to use
     * @return A table row model
     */
    public Object[][] createTable(University uni) {
        // Instantiate temporary model with the length equal to that of the class's
faculty array
        Object[][] temp = new Object[size][1];
        // Keep track of the amount of results
        int tempSize = 0;

        // Go through each faculty in the class array
        for (int i = 0; i < size; i++) {
            // If the name of the current faculty's university is equal to the one
given:
            if (faculties[i].getUni().getName().equals(uni.getName())) {
                // Save the faculty's name into the temporary model
                temp[tempSize][0] = faculties[i].getName();
                // Increment size
                tempSize++;
```

```
            }
        }

        // The program will also need to store an array of the IDs of the faculties
being used
        // Create new integer array with size equal to the length of the temporary model
        int[] tempTableArr = new int[tempSize];
        // Keep track of the size of the integer array
        int tempTableArrSize = 0;

        // Go through each faculty in the class array
        for (int i = 0; i < size; i++) {
            // If the name of the current faculty's university is equal to the one
given:
            if (faculties[i].getUni().getName().equals(uni.getName())) {
                // Save the faculty's ID into the integer array
                tempTableArr[tempTableArrSize] = faculties[i].getID();
                // Increment size
                tempTableArrSize++;
            }
        }

        // Instantiate output model with the length equal to that of the size of the
temporary model
        Object[][] output = new Object[tempSize][1];
        // Copy array
        for (int i = 0; i < tempSize; i++) {
            output[i][0] = temp[i][0];
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return output model
        return output;
    }

    /**
     * Return the IDs of the faculties that were used in the making of the
     * latest table row model
     *
     * @return The list of IDs
     */
    public int[] getTableArr() {
        return tableArr;
    }
}
```

2.3 DegManager

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Managers;

// IMPORTS
import Driver.dbDriver;
import Objects.Degree;
import Objects.Faculty;
import Objects.Filter;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Saien Naidu
 */
public class DegManager {

    // FIELDS
    private Degree[] degrees = new Degree[1000];
    private int size;
    private final dbDriver db = new dbDriver();
    private final UniManager um = new UniManager();
    private final FacManager fm = new FacManager();
    private int[] tableArr = new int[1000];

    // CONSTRUCTOR
    /**
     * Fetch all the Degrees from the DB and save it into the array
     */
    public DegManager() {
        try {
            // Fetch data
            ResultSet rs = db.query("SELECT * FROM Degree_Table;");

            // If there is more data;
            while (!rs.isLast()) {
                // Save Degree into array
                degrees[size] = createDeg(rs, size);
                // Increment size
                size++;
            }

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #08: Failed while retrieving Degrees from DB.");
        }
    }

    // PROPERTIES
    // Create and return a Degree object using a specific row from the DB
    private Degree createDeg(ResultSet rs, int row) {
        try {
            // Prepare the next row of results
            rs.absolute(row + 1);

            // Get the value of each column in the row
            int ID = rs.getInt("ID");
            String name = rs.getString("DegreeName");
            int uni = rs.getInt("UniversityID");
            int fac = rs.getInt("FacultyID");
            String desc = rs.getString("Description");

            // Create new Degree object using the values, then return it
```

```
            return (new Degree(ID, name, um.getUniWithID(uni), fm.getFacWithID(fac),
desc));

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #08: Failed while retrieving Degrees from DB.");
        }

        return null;
    }

    /**
     * Fetch a degree with a specific ID from the array
     *
     * @param ID The ID of the Degree Object that you want to fetch
     * @return The matching Degree Object
     */
    public Degree getDegWithID(int ID) {
        // Go through every entry in the array until a match is found
        for (int i = 0; i < size; i++) {
            int current = degrees[i].getDegreeID();
            // If match found:
            if (current == ID) {
                // Return current Degree object
                return degrees[i];
            }
        }
        // If no match is found, return nothing
        return null;
    }

    /**
     * Fetch list of degrees based on DB query
     *
     * @param query The query to send to the DB
     * @return Returns the list of Degree Objects matching the query
     */
    public Degree[] getDegWithQuery(String query) {
        try {
            // Fetch data
            ResultSet rs = db.query(query);
            // Instantiate temporary Degree array with the length equal to that of the
class's degree array
            Degree[] temp = new Degree[size];
            // Keep track of the length of the array
            int tempSize = 0;

            // If there is more data:
            while (!rs.isLast()) {
                // Create a Degree Object using the row's data and save it into the
temporaray array
                temp[tempSize] = createDeg(rs, tempSize);
                // Increment size
                tempSize++;
                /*
                Error handling:
                If the results are found to be empty:
                (i.e. there are no degrees that satisfy the DB query)
                 */
                if ((tempSize + 1) >= this.size) {
                    // Create a degree array that can only hold 1 entry
                    Degree[] error = new Degree[1];
                    // Create a useless Degree object with a name of "None"
                    error[0] = new Degree(0, "None", null, null, null);
                    // return this Degree object and interrupt this method
                    return error;
```

```
                    // This means that the word "None" will display on the table
                }
            }

            // Instantiate output array with a size that's equal to the amount of
results
            Degree[] output = new Degree[tempSize];
            // Copy array
            for (int i = 0; i < tempSize; i++) {
                output[i] = temp[i];
            }
            // Return the output array
            return output;

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #09: Failed while retrieving specific Degrees from
DB.");
        }
        return null;
    }

    /**
     * Create a table row model using all the names of all the degrees
     *
     * @return A table row model
     */
    public Object[][] createTable() {
        // Instantiate model with length equal to that of the class's degree array
        Object[][] data = new Object[size][1];
        // The program will also need to store an array of the IDs of the degrees being
used
        int[] tempTableArr = new int[size];

        // Go through each degree name and save it to the model array
        // Also go through each degree ID and save it to the integer array
        for (int i = 0; i < size; i++) {
            data[i][0] = degrees[i].getName();
            tempTableArr[i] = degrees[i].getDegreeID();
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return the model
        return data;
    }

    /**
     * Create a table row model using all the names of specific degrees
     *
     * @param input The list of Degree Objects to use
     * @return A table row model
     */
    public Object[][] createTable(Degree[] input) {

        // Instantiate model with size equal to the length of the given degree array
        Object[][] data = new Object[input.length][1];
        // The program will also need to store an array of the IDs of the degrees being
used
        int[] tempTableArr = new int[input.length];

        // Go through each degree name and save it to the model array
        // Also go through each degree ID and save it to the integer array
        for (int i = 0; i < input.length; i++) {
            data[i][0] = input[i].getName();
```

```
            tempTableArr[i] = input[i].getDegreeID();
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return the model
        return data;
    }

    /**
     * Create a table row model using all the names of degrees that belong to a
     * specific faculty
     *
     * @param fac The Faculty Object to use
     * @return A table row model
     */
    public Object[][] createTable(Faculty fac) {

        // Instantiate temporary model with size equal to the length of the given
faculty array
        Object[][] temp = new Object[size][1];
        // Keep track of the amount of results
        int tempSize = 0;

        // Go through each Degree in the class's degree array
        for (int i = 0; i < size; i++) {
            // If the ID of the degree's faculty matches that of the given faculty:
            if (degrees[i].getFac().getID() == (fac.getID())) {
                // Save the name of the degree into the temporary array
                temp[tempSize][0] = degrees[i].getName();
                // Increment size
                tempSize++;
            }
        }

        // The program will also need to store an array of the IDs of the degrees being
used
        // Instantiate an integer array with size equal to the length of the temporary
model array
        int[] tempTableArr = new int[tempSize];
        // Keep track of the size of the integer array
        int tempTableArrSize = 0;

        // Go through each degree in the class array
        for (int i = 0; i < size; i++) {
            // If the ID of the current degree's faculty is equal to the one given:
            if (degrees[i].getFac().getID() == (fac.getID())) {
                // Save the degree's ID into the integer array
                tempTableArr[tempTableArrSize] = degrees[i].getDegreeID();
                // Increment size
                tempTableArrSize++;
            }
        }

        // Instantiate output model with the length equal to that of the size of the
temporary model
        Object[][] output = new Object[tempSize][1];
        // Copy array
        for (int i = 0; i < tempSize; i++) {
            output[i][0] = temp[i][0];
        }

        // Store the integer array for future use
        tableArr = tempTableArr;

        // Return output model
```

```
        return output;
    }

    /**
     * Return a list of degrees that match the filter
     *
     * @param f The filter containing the user's choices
     * @return The list of Degree Objects that satisfy the filter
     */
    public Degree[] degreeFinder(Filter f) {
        // SETUP
        ReqManager rm = new ReqManager();
        Degree[] output = new Degree[size];

        // To start, the list will contain all the degrees in the DB
        for (int i = 0; i < size; i++) {
            output[i] = degrees[i];
        }

        // Track the size, which equals the amount of degrees
        int outputSize = size;

        // <editor-fold defaultstate="collapsed" desc="Search using user's marks">
        // If 'Use your marks' is selected:
        if (f.isUseMarks()) {
            // Get the IDs of the degrees that the user has met the requirements of
            int[] IDs = rm.reqMet();
            // Create a temporary degree array with a size equal to the amount of IDs
            Degree[] temp = new Degree[IDs.length];

            // Go through each ID, getting the Degree using it and save it to the Degree
array.
            for (int i = 0; i < IDs.length; i++) {
                temp[i] = getDegWithID(IDs[i]);
            }

            // The output array becomes this Degree array
            output = new Degree[temp.length];
            output = temp;
            // Keep track of the size
            outputSize = temp.length;

        }
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="Search based on Location">
        // If the user has a location selected:
        // First check to see if this is true
        boolean locationOn = false;
        // The selected provinces will need to be added to the query
        String locations = "";

        // If user has selected KZN:
        if (f.isKzn()) {
            // Adds the province to the string, and what to add is based on whether this
is the first location or not
            locations = addLocation(locations, "'KZN'");
            // Declare that the user does have at least one location selected
            locationOn = true;
        }
        // If user has selected Gauteng:
        if (f.isGauteng()) {
            locations = addLocation(locations, "'G'");
            locationOn = true;
        }
        // If user has selected Eastern Cape:
        if (f.isEastcape()) {
```

```
            locations = addLocation(locations, "'EC'");
            locationOn = true;
        }
        // If user has selected Western Cape:
        if (f.isWestcape()) {
            locations = addLocation(locations, "'WC'");
            locationOn = true;
        }
        // If user has selected Free State:
        if (f.isFreestate()) {
            locations = addLocation(locations, "'FS'");
            locationOn = true;
        }
        // If user has selected North West Province:
        if (f.isNorthwest()) {
            locations = addLocation(locations, "'NW'");
            locationOn = true;
        }
        // If user has selected Mpumalanga:
        if (f.isMpumalanga()) {
            locations = addLocation(locations, "'M'");
            locationOn = true;
        }
        // If user has selected Limpopo:
        if (f.isLimpopo()) {
            locations = addLocation(locations, "'L'");
            locationOn = true;
        }

        // If the user has selected at least one province:
        if (locationOn) {
            // Create a degree array with a size equal to the current size of the output
array
            Degree[] results = new Degree[outputSize];

            // Fetch the desired Degrees from the DB using a query
            results = getDegWithQuery("""
                                      SELECT
                                          Degree_Table.ID,
                                          DegreeName,
                                          Degree_Table.UniversityID,
                                          Degree_Table.FacultyID,
                                          Degree_Table.Description
                                      FROM
                                          Degree_Table
                                          INNER JOIN (
                                              University_Table
                                              INNER JOIN Faculty_Table ON
University_Table.ID = Faculty_Table.UniversityID
                                          ) ON (Degree_Table.UniversityID =
University_Table.ID)
                                          AND (Degree_Table.FacultyID =
Faculty_Table.ID)
                                      WHERE
                                          Location IN ("" + locations + ");");

            // Declare a boolean that will turn true after the last row of the results
is reached
            boolean endReached = false;
            // Reset the size of the output array
            outputSize = 0;
            // While the end of the results has not been reached
            // AND the output array size is less than that of the result array (which it
always should be)
            while (!endReached && (outputSize < results.length)) {
                // If the result in the current row is not nothing
                if (results[outputSize] != null) {
```

```
                    // Increment size
                    outputSize++;
                } else { // Otherwise
                    // Declare that the end of the results has been reached and end the
while loop
                    endReached = true;
                }
            }

            // Create a temporary Degree array with a size equal to that of the output
array
            Degree[] temp = new Degree[outputSize];
            // Keep track of the size of the temporary array
            int tempSize = 0;
            // Add the degrees that are part of both the output and the result array to
the temporary array
            for (Degree output1 : output) {
                for (Degree result : results) {
                    // If the degree is in both arrays:
                    if (output1.getDegreeID() == result.getDegreeID()) {
                        // Add the degree into the temporary array
                        temp[tempSize] = result;
                        // Increment size
                        tempSize++;
                    }
                }
            }

            // Reset the output array, giving it a size equal to that of the temporary
array
            output = new Degree[tempSize];
            // Copy the array
            for (int i = 0; i < tempSize; i++) {
                output[i] = temp[i];
            }
            // Update the output array size
            outputSize = tempSize;
        }
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="Search based on Faculty">
        // If the user has a location selected:
        // First check to see if this is true
        boolean facultyOn = false;
        // The selected faculties will need to be added to the query
        String faculties = "";

        // If user has selected Commerce:
        if (f.isCommerce()) {
            // Adds the faculty to the string, and what to add is based on whether this
is the first faculty or not
            faculties = addFaculty(faculties, "'%Commerce%'");
            // Declare that the user does have at least one faculty selected
            facultyOn = true;
        }
        // If user has selected Engineering:
        if (f.isEngineering()) {
            faculties = addFaculty(faculties, "'%Engineering%'");
            facultyOn = true;
        }
        // If user has selected Health Sciences:
        if (f.isHealth()) {
            faculties = addFaculty(faculties, "'%Health%'");
            facultyOn = true;
        }
        // If user has selected Law:
        if (f.isLaw()) {
```

```
            faculties = addFaculty(faculties, "'%Law%'");
            facultyOn = true;
        }
        // If user has selected Humanities:
        if (f.isHumanities()) {
            faculties = addFaculty(faculties, "'%Humanities%'");
            facultyOn = true;
        }
        // If user has selected Sciences:
        if (f.isSciences()) {
            faculties = addFaculty(faculties, "'%Science%'");
            facultyOn = true;
        }

        // If the user has selected at least one faculty:
        if (facultyOn) {
            // Create a degree array with a size equal to the current size of the output
array
            Degree[] results = new Degree[outputSize];

            // Fetch the degrees belonging to the selected faculties from the DB
            results = getDegWithQuery("""
                                      SELECT
                                          Degree_Table.ID,
                                          DegreeName,
                                          Degree_Table.UniversityID,
                                          Degree_Table.FacultyID,
                                          Degree_Table.Description
                                      FROM
                                          Degree_Table
                                          INNER JOIN (
                                              University_Table
                                              INNER JOIN Faculty_Table ON
University_Table.ID = Faculty_Table.UniversityID
                                          ) ON (Degree_Table.UniversityID =
University_Table.ID)
                                          AND (Degree_Table.FacultyID =
Faculty_Table.ID)
                                      WHERE
                                          Degree_Table.FacultyName LIKE """
                    + faculties + ";");

            // Declare a boolean that will turn true after the last row of the results
is reached
            boolean endReached = false;
            // Reset the size of the output array
            outputSize = 0;
            // While the end of the results has not been reached
            // AND the output array size is less than that of the result array (which it
always should be)
            while (!endReached && (outputSize < results.length)) {
                // If the result in the current row is not nothing
                if (results[outputSize] != null) {
                    // Increment size
                    outputSize++;
                } else { // Otherwise
                    // Declare that the end of the results has been reached and end the
while loop
                    endReached = true;
                }
            }

            // Create a temporary Degree array with a size equal to that of the output
array
            Degree[] temp = new Degree[outputSize];
            // Keep track of the size of the temporary array
            int tempSize = 0;
```

```
            // Add the degrees that are part of both the output and the result array to
the temporary array
            for (Degree output1 : output) {
                for (Degree result : results) {
                    // If the degree is in both arrays:
                    if (output1.getDegreeID() == result.getDegreeID()) {
                        // Add the degree into the temporary array
                        temp[tempSize] = result;
                        // Increment size
                        tempSize++;
                    }
                }
            }

            // Reset the output array, giving it a size equal to that of the temporary
array
            output = new Degree[tempSize];
            // Copy the array
            for (int i = 0; i < tempSize; i++) {
                output[i] = temp[i];
            }
            // Update the output array size
            outputSize = tempSize;
        }
        // </editor-fold>

        // <editor-fold defaultstate="collapsed" desc="Search based on University">
        // If the user has selected to include/exclude specific university/universities
        boolean includeOn = f.isInclude();
        boolean excludeOn = f.isExclude();

        // If the user has selected include:
        if (includeOn) {
            // Fetch the names of the selected universities
            String uni1 = f.getUni1();
            String uni2 = f.getUni2();
            String uni3 = f.getUni3();

            // The selected universities will need to be added to the query
            String universities = "";

            // Check to see if the user has not selected include without selecting a
university:
            boolean notNone = false;
            // If a university has been selected in Combo Box 1:
            if (!uni1.equals("None")) {
                // Adds the university to the string, and what to add is based on
whether this is the first university or not
                universities = addUniversity(universities, uni1, true);
                // Declare that at least one university was actually selected
                notNone = true;
            }
            // If a university has been selected in Combo Box 2:
            if (!uni2.equals("None")) {
                universities = addUniversity(universities, uni2, true);
                notNone = true;
            }
            // If a university has been selected in Combo Box 3:
            if (!uni3.equals("None")) {
                universities = addUniversity(universities, uni3, true);
                notNone = true;
            }

            // If at least 1 university has been selected:
            if (notNone) {
                // Create a degree array with a size equal to the current size of the
output array
```

```
                Degree[] results = new Degree[outputSize];

                // Fetch the degrees belonging to the selected universities from the DB
                results = getDegWithQuery("""
                                          SELECT
                                              Degree_Table.ID,
                                              DegreeName,
                                              Degree_Table.UniversityID,
                                              Degree_Table.FacultyID,
                                              Degree_Table.Description
                                          FROM
                                              Degree_Table
                                              INNER JOIN (
                                                  University_Table
                                                  INNER JOIN Faculty_Table ON
University_Table.ID = Faculty_Table.UniversityID
                                              ) ON (Degree_Table.UniversityID =
University_Table.ID)
                                              AND (Degree_Table.FacultyID =
Faculty_Table.ID)
                                          WHERE
                                              University_Table.UniversityName = """
                        + universities + ";");

                // Declare a boolean that will turn true after the last row of the
results is reached
                boolean endReached = false;
                // Reset the size of the output array
                outputSize = 0;
                // While the end of the results has not been reached
                // AND the output array size is less than that of the result array
(which it always should be)
                while (!endReached && (outputSize < results.length)) {
                    // If the result in the current row is not nothing
                    if (results[outputSize] != null) {
                        // Increment size
                        outputSize++;
                    } else { // Otherwise
                        // Declare that the end of the results has been reached and end
the while loop
                        endReached = true;
                    }
                }

                // Create a temporary Degree array with a size equal to that of the
output array
                Degree[] temp = new Degree[outputSize];
                // Keep track of the size of the temporary array
                int tempSize = 0;
                // Add the degrees that are part of both the output and the result array
to the temporary array
                for (Degree output1 : output) {
                    for (Degree result : results) {
                        // If the degree is in both arrays:
                        if (output1.getDegreeID() == result.getDegreeID()) {
                            // Add the degree into the temporary array
                            temp[tempSize] = result;
                            // Increment size
                            tempSize++;
                        }
                    }
                }

                // Reset the output array, giving it a size equal to that of the
temporary array
                output = new Degree[tempSize];
                // Copy the array
```

```
                for (int i = 0; i < tempSize; i++) {
                    output[i] = temp[i];
                }
                // Update the output array size
                outputSize = tempSize;
            }
        }

        // If the user has selected exclude:
        if (excludeOn) {
            // Fetch the names of the selected universities
            String uni1 = f.getUni1();
            String uni2 = f.getUni2();
            String uni3 = f.getUni3();

            // The selected universities will need to be added to the query
            String universities = "";

            // Check to see if the user has not selected include without selecting a
university:
            boolean notNone = false;
            // If a university has been selected in Combo Box 1:
            if (!uni1.equals("None")) {
                // Adds the university to the string, and what to add is based on
whether this is the first university or not
                universities = addUniversity(universities, uni1, false);
                // Declare that at least one university was actually selected
                notNone = true;
            }
            // If a university has been selected in Combo Box 2:
            if (!uni2.equals("None")) {
                universities = addUniversity(universities, uni2, false);
                notNone = true;
            }
            // If a university has been selected in Combo Box 3:
            if (!uni3.equals("None")) {
                universities = addUniversity(universities, uni3, false);
                notNone = true;
            }

            // If at least 1 university has been selected:
            if (notNone) {
                // Create a degree array with a size equal to the current size of the
output array
                Degree[] results = new Degree[outputSize];

                // Fetch the degrees belonging to the selected universities from the DB
                results = getDegWithQuery("""
                                          SELECT
                                              Degree_Table.ID,
                                              DegreeName,
                                              Degree_Table.UniversityID,
                                              Degree_Table.FacultyID,
                                              Degree_Table.Description
                                          FROM
                                              Degree_Table
                                              INNER JOIN (
                                                  University_Table
                                                  INNER JOIN Faculty_Table ON
University_Table.ID = Faculty_Table.UniversityID
                                              ) ON (Degree_Table.UniversityID =
University_Table.ID)
                                              AND (Degree_Table.FacultyID =
Faculty_Table.ID)
                                          WHERE
                                              University_Table.UniversityName <> """
                        + universities + ";");
```

```
                // Declare a boolean that will turn true after the last row of the
results is reached
                boolean endReached = false;
                // Reset the size of the output array
                outputSize = 0;
                // While the end of the results has not been reached
                // AND the output array size is less than that of the result array
(which it always should be)
                while (!endReached && (outputSize < results.length)) {
                    // If the result in the current row is not nothing
                    if (results[outputSize] != null) {
                        // Increment size
                        outputSize++;
                    } else { // Otherwise
                        // Declare that the end of the results has been reached and end
the while loop
                        endReached = true;
                    }
                }

                // Create a temporary Degree array with a size equal to that of the
output array
                Degree[] temp = new Degree[outputSize];
                // Keep track of the size of the temporary array
                int tempSize = 0;
                // Add the degrees that are part of both the output and the result array
to the temporary array
                for (Degree output1 : output) {
                    for (Degree result : results) {
                        // If the degree is in both arrays:
                        if (output1.getDegreeID() == result.getDegreeID()) {
                            // Add the degree into the temporary array
                            temp[tempSize] = result;
                            // Increment size
                            tempSize++;
                        }
                    }
                }

                // Reset the output array, giving it a size equal to that of the
temporary array
                output = new Degree[tempSize];
                // Copy the array
                for (int i = 0; i < tempSize; i++) {
                    output[i] = temp[i];
                }
                // Update the output array size
                outputSize = tempSize;
            }

        }
        // </editor-fold>

        // Return the output array
        return output;
    }

    // Add a location to a query
    private String addLocation(String original, String toAdd) {
        // If there are currently no locations in the query
        if (original.equals("")) {
            // Only return the location
            return toAdd;
        } else { // Otherwise
            // Return the new location appended to the others while maintaining the
integrity of the query
```

```
            return original + ", " + toAdd;
        }
    }

    // Add a faculty to a query
    private String addFaculty(String original, String toAdd) {
        // If there are currently no faculties in the query
        if (original.equals("")) {
            // Only return the faculty
            return toAdd;
        } else { // Otherwise
            // Return the new faculty appended to the others while maintaining the
integrity of the query
            return original + "\nOR Degree_Table.FacultyName LIKE " + toAdd;
        }
    }

    // Add a faculty to a query
    private String addUniversity(String original, String toAdd, boolean include) {
        // If there are currently no university in the query
        if (original.equals("")) {
            // Only return the university
            return "'" + toAdd + "'";
        } else if (include) { //Otherwise, if include is selected:
            // Return the new university appended to the others while maintaining the
integrity of the query
            return original + "\nOR University_Table.UniversityName = '" + toAdd + "'";
        } else { // Otherwise
            // Return the new university appended to the others while maintaining the
integrity of the query
            return original + "\nAND University_Table.UniversityName <> '" + toAdd +
"'";
        }
    }

    /**
     * Return the IDs of the degrees that were used in the making of the latest
     * table row model
     *
     * @return The list of IDs
     */
    public int[] getTableArr() {
        return tableArr;
    }
}
```

2.4 ReqManager

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

```
package Managers;

// IMPORTS
import Driver.dbDriver;
import Objects.Requirement;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Saien
 */
public class ReqManager {

    // FIELDS
    private Requirement[] requirements = new Requirement[2000];
    private int size;
    private final dbDriver db = new dbDriver();
    private final DegManager dm = new DegManager();

    // CONSTRUCTOR
    /**
     * Fetch all the Requirements from the DB and save it into the array
     */
    public ReqManager() {
        try {
            // Fetch data
            ResultSet rs = db.query("SELECT * FROM Requirement_Table;");
            // The first enrty in the array is reserved for the user's marks, which is
why size starts at 1
            size = 1;

            // If there is more data;
            while (!rs.isLast()) {
                // Save Degree into array
                requirements[size] = createReq(rs, size);
                // Increment size
                size++;
            }

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #10: Failed while retrieving Requirements from
DB.");
        }

        initUserMarks();
    }

    // PROPERTIES
    // Create and return a Requirement object using a specific row from the DB
    private Requirement createReq(ResultSet rs, int row) {
        try {
            // Prepare the next row of results
            rs.absolute(row + 1);

            // Fetch the requirement's ID and the requirement's degree's ID
            int id = rs.getInt("ID");
            int degID = rs.getInt("DegreeID");
```

```
            // For each subject...
            // Home Language
            // Fetch the value
            String hl = rs.getString("HL");
            // Split into the subject choice...
            String hlC = hl.substring(0, 3);
            // ...and mark
            int hlM = Integer.parseInt(hl.substring(3));

            // 1st Add Lang
            String fal = rs.getString("FAL");
            String falC = fal.substring(0, 3);
            int falM = Integer.parseInt(fal.substring(3));

            // Mathematics
            String math = rs.getString("MATH");
            String mathC = math.substring(0, 3);
            int mathM = Integer.parseInt(math.substring(3));

            // 1st Subject Choice
            String opt1 = rs.getString("OPT1");
            String opt1C = opt1.substring(0, 3);
            int opt1M = Integer.parseInt(opt1.substring(3));

            // 2nd Subject Choice
            String opt2 = rs.getString("OPT2");
            String opt2C = opt2.substring(0, 3);
            int opt2M = Integer.parseInt(opt2.substring(3));

            // 3rd Subject Choice
            String opt3 = rs.getString("OPT3");
            String opt3C = opt3.substring(0, 3);
            int opt3M = Integer.parseInt(opt3.substring(3));

            // LO has to be passed, but there's never a specific mark needed
            int lo = 50;
            // Fetch the APS
            int aps = rs.getInt("APS");

            // Create new Requirement object using the values, then return it
            return (new Requirement(id, dm.getDegWithID(degID), hlM, hlC, falM, falC,
mathM, mathC, opt1M, opt1C, opt2M, opt2C, opt3M, opt3C, lo, aps));

            // Error handling
        } catch (SQLException ex) {
            Logger.getLogger(UniManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #10: Failed while retrieving Requirements from
DB.");
        }

        return null;
    }

    // Initialize the user's marks
    // The user's marks are represented in the same format as a Requirement object,
hence it is handled here
    private void initUserMarks() {
        // Open the file
        File file = new File("data\\UserMarks.txt");
        try {
            // Create a file scanner
            Scanner fileSC = new Scanner(file).useDelimiter("#");
            // The APS score of each subject will be summed
            int aps = 0;

            // For each subject...
```

```
            // Home Language
            // Fetch the value from the file
            String hl = fileSC.next();
            // Split into the subject choice...
            String hlC = hl.substring(0, 3);
            // ...and mark
            int hlM = Integer.parseInt(hl.substring(3));
            // Calculate APS
            aps += calcAPS(hlM);

            // 1st Add Lang
            String fal = fileSC.next();
            String falC = fal.substring(0, 3);
            int falM = Integer.parseInt(fal.substring(3));
            aps += calcAPS(falM);

            // Mathematics
            String math = fileSC.next();
            String mathC = math.substring(0, 3);
            int mathM = Integer.parseInt(math.substring(3));
            aps += calcAPS(mathM);

            // 1st Subject Choice
            String opt1 = fileSC.next();
            String opt1C = opt1.substring(0, 3);
            int opt1M = Integer.parseInt(opt1.substring(3));
            aps += calcAPS(opt1M);

            // 2nd Subject Choice
            String opt2 = fileSC.next();
            String opt2C = opt2.substring(0, 3);
            int opt2M = Integer.parseInt(opt2.substring(3));
            aps += calcAPS(opt2M);

            // 3rd Subject Choice
            String opt3 = fileSC.next();
            String opt3C = opt3.substring(0, 3);
            int opt3M = Integer.parseInt(opt3.substring(3));
            aps += calcAPS(opt3M);

            // Fetch the user's LO mark
            int lo = fileSC.nextInt();

            // Create new Requirement object using the values, then save it to the FIRST
entry in the class's Requirement array
            requirements[0] = new Requirement(0, null, hlM, hlC, falM, falC, mathM,
mathC, opt1M, opt1C, opt2M, opt2C, opt3M, opt3C, lo, aps);

            // Error handling
        } catch (FileNotFoundException ex) {
            Logger.getLogger(ReqManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #11: UserMarks file not found.");
        }
    }

    // Calculate the APS score recieved for a subject
    private int calcAPS(int mark) {
        int rank = mark / 10;

        return switch (rank) {
            case 0, 1, 2 ->
                1;
            case 3 ->
                2;
            case 4 ->
                3;
            case 5 ->
```

```
                4;
            case 6 ->
                5;
            case 7 ->
                6;
            case 8, 9, 10 ->
                7;
            default ->
                0;
        };
    }

    /**
     * Return a subject's full name from its abbreviation
     *
     * @param abbreviation The subject's abbreviated form (E.g. "acc")
     * @return The subject's full name (E.g. "Accounting")
     */
    public String getSubject(String abbreviation) {
        return switch (abbreviation) {
            case "acc" ->
                "Accounting";
            case "bus" ->
                "Business Studies";
            case "cat" ->
                "CAT";
            case "con" ->
                "Consumer Studies";
            case "dan" ->
                "Dance Studies";
            case "des" ->
                "Design";
            case "dra" ->
                "Dramatic Arts";
            case "edg" ->
                "EDG";
            case "eco" ->
                "Economics";
            case "geo" ->
                "Geography";
            case "his" ->
                "History";
            case "hos" ->
                "Hospitality Studies";
            case "inf" ->
                "Information Technology";
            case "lif" ->
                "Life Sciences";
            case "mar" ->
                "Marine Sciences";
            case "mus" ->
                "Music";
            case "phy" ->
                "Physical Sciences";
            case "tou" ->
                "Tourism";
            case "vis" ->
                "Visual Arts";
            default ->
                "Other";
        };
    }

    /**
     * Return a subject's abbreviation from its full name
     *
     * @param subject The subject's full name (E.g. "Accounting")
```

```
     * @return The subject's abbreviated form (E.g. "acc")
     */
    public String getAbbreviation(String subject) {
        return switch (subject) {
            case "Accounting" ->
                "acc";
            case "Business Studies" ->
                "bus";
            case "CAT" ->
                "cat";
            case "Consumer Studies" ->
                "con";
            case "Dance Studies" ->
                "dan";
            case "Design" ->
                "des";
            case "Dramatic Arts" ->
                "dra";
            case "EDG" ->
                "edg";
            case "Economics" ->
                "eco";
            case "Geography" ->
                "geo";
            case "History" ->
                "his";
            case "Hospitality Studies" ->
                "hos";
            case "Information Technology" ->
                "inf";
            case "Life Sciences" ->
                "lif";
            case "Marine Sciences" ->
                "mar";
            case "Music" ->
                "mus";
            case "Physical Sciences" ->
                "phy";
            case "Tourism" ->
                "tou";
            case "Visual Arts" ->
                "vis";
            default ->
                "oth";
        };
    }

    /**
     * Return the Requirement representing the user's marks
     *
     * @return The user's marks as a Requirement Object
     */
    public Requirement getUserMarks() {
        // The user's marks are ALWAYS held in the first entry of the class's
requirement array
        return requirements[0];
    }

    /**
     * Change user's marks
     *
     * @param hlM The mark received for Home Language
     * @param hlC The subject choice for Home Language, either 'eng' or 'oth'
     * @param falM The mark received for 1st Add Lang
     * @param falC The subject choice for 1st Add Lang, either 'eng' or 'oth'
     * @param mathM The mark received for Mathematics
     * @param mathC The subject choice for Mathematics, either 'cor', 'lit' or
```

```
     * 'tec'
     * @param opt1M The mark received for the user's 1st Subject Choice
     * @param opt1C The user's 1st Subject Choice
     * @param opt2M The mark received for the user's 2nd Subject Choice
     * @param opt2C The user's 2nd Subject Choice
     * @param opt3M The mark received for the user's 3rd Subject Choice
     * @param opt3C The user's 3rd Subject Choice
     * @param lo The mark received for LO
     */
    public void setUserMarks(int hlM, String hlC, int falM, String falC, int mathM,
String mathC, int opt1M, String opt1C, int opt2M, String opt2C, int opt3M, String opt3C,
int lo) {
        FileWriter fileFW = null;
        try {
            // Using the user's marks, calculate the APS
            int aps = calcAPS(hlM) + calcAPS(falM) + calcAPS(mathM) + calcAPS(opt1M) +
calcAPS(opt2M) + calcAPS(opt3M);
            // Create new Requirement object using the received values, then save it to
the FIRST entry in the class's Requirement array
            requirements[0] = new Requirement(0, null, hlM, hlC, falM, falC, mathM,
mathC, opt1M, opt1C, opt2M, opt2C, opt3M, opt3C, lo, aps);

            // Write the user's new marks to the text file (OVERWRITE)
            fileFW = new FileWriter("data\\UserMarks.txt");
            fileFW.write(hlC + hlM + "#" + falC + falM + "#" + mathC + mathM + "#" +
opt1C + opt1M + "#" + opt2C + opt2M + "#" + opt3C + opt3M + "#" + lo);

            // File handling
        } catch (IOException ex) {
            Logger.getLogger(ReqManager.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #12: Error while saving user's marks to text
file.");
        } finally {
            try {
                fileFW.close();
            } catch (IOException ex) {
                Logger.getLogger(ReqManager.class.getName()).log(Level.SEVERE, null,
ex);
                System.out.println("Error #12: Error while saving user's marks to text
file.");
            }
        }
    }

    /**
     * Return all the requirements in the DB EXCEPT the user's marks
     *
     * @return The list of Requirement Objects
     */
    public Requirement[] getAll() {
        Requirement[] output = new Requirement[size - 1];
        for (int i = 0; i < (size - 1); i++) {
            output[i] = requirements[i + 1];
        }
        return output;
    }

    /**
     * Return all the degrees that the user has met the requirements of.
     *
     * @return The list of IDs of the degrees
     */
    public int[] reqMet() {
        // Create a new temporary Requirement array with a size equal to that of the
class's Requirement array
        Requirement[] temp = new Requirement[size];
        // Keep track of the amount of degrees
```

```
        int tempSize = 0;

        // Fetch the user's marks
        Requirement user = requirements[0];

        // Go through each requirement in the DB
        for (int i = 1; i < this.size; i++) {
            // Store the current requirement
            Requirement current = requirements[i];
```

```
/*
/ With all subjects, the user's mark has to be the same or higher than the mark specified
/ For HL, FAL and MATHS, the user's subject choice has to be the same as the requirement's specification
/
/ For the subject choices...it's different. I will use Physics as an example to explain the complexity
/ Let's say that Physics is a requirement of the Degree
/ Physics can be the user's 1st, 2nd or 3rd choice (opt1C, opt2C or opt3C)
/ And it can be in opt1C, opt2C or opt3C of the requirement
/ Therefore, a cross check has to be used:
/
/ The user's opt1C has to be compared with the degree's opt1C, opt2C and opt3C,
/ The user's opt2C has to be compared with the degree's opt1C, opt2C and opt3C, and
/ The user's opt3C has to be compared with the degree's opt1C, opt2C and opt3C.
/ This gives us 5 possible combinations: 123, 132, 213, 231, 312 (321 will never happen)
/
/ If the user has Physics as their 1st choice, and the degree's opt1C is Physics, this will give us the combination: 123
/ If the user has Physics as their 2nd choice, and the degree's opt1C is Physics, this will give us the combination: 213
/ If the user has Physics as their 3rd choice, and the degree's opt1C is Physics, this will give us the combination: 231
/ If the user has Physics as their 1st choice, and the degree's opt2C is Physics, this will give us the combination: 213
/ If the user has Physics as their 2nd choice, and the degree's opt2C is Physics, this will give us the combination: 123
/ If the user has Physics as their 3rd choice, and the degree's opt2C is Physics, this will give us the combination: 132
/ If the user has Physics as their 1st choice, and the degree's opt3C is Physics, this will give us the combination: 312
/ If the user has Physics as their 2nd choice, and the degree's opt3C is Physics, this will give us the combination: 132
/ If the user has Physics as their 3rd choice, and the degree's opt3C is Physics, this will give us the combination: 123

/ The user's LO mark has to be 50 or above
/ The user's APS score has to be the same or above the APS score specified
*/
if ((user.getHlChoice().equals(current.getHlChoice())) || current.getHlChoice().equals("any")) {
    if (user.getHlMark() >= current.getHlMark()) {
        if ((user.getFalChoice().equals(current.getFalChoice())) || current.getFalChoice().equals("any")) {
            if (user.getFalMark() >= current.getFalMark()) {
                if ((user.getMathChoice().equals(current.getMathChoice())) || current.getMathChoice().equals("any")) {
                    if (user.getMathMark() >= current.getMathMark()) {

                        if ((user.getOpt1Choice().equals(current.getOpt1Choice())) || current.getOpt1Choice().equals("any")) {
                            if (user.getOpt1Mark() >= current.getOpt1Mark()) {

                                if ((user.getOpt2Choice().equals(current.getOpt2Choice())) || current.getOpt2Choice().equals("any")) {
                                    if (user.getOpt2Mark() >= current.getOpt2Mark()) {
                                        if ((user.getOpt3Choice().equals(current.getOpt3Choice())) || current.getOpt3Choice().equals("any")) {
                                            if (user.getOpt3Mark() >= current.getOpt3Mark()) {
                                                //123
                                                if (user.getLo() >= 50) {
                                                    if (user.getAps() >= current.getAps()) {
                                                        // Add the current degree to the array
                                                        temp[tempSize] = requirements[i];
                                                        // Increment size
                                                        tempSize++;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
```

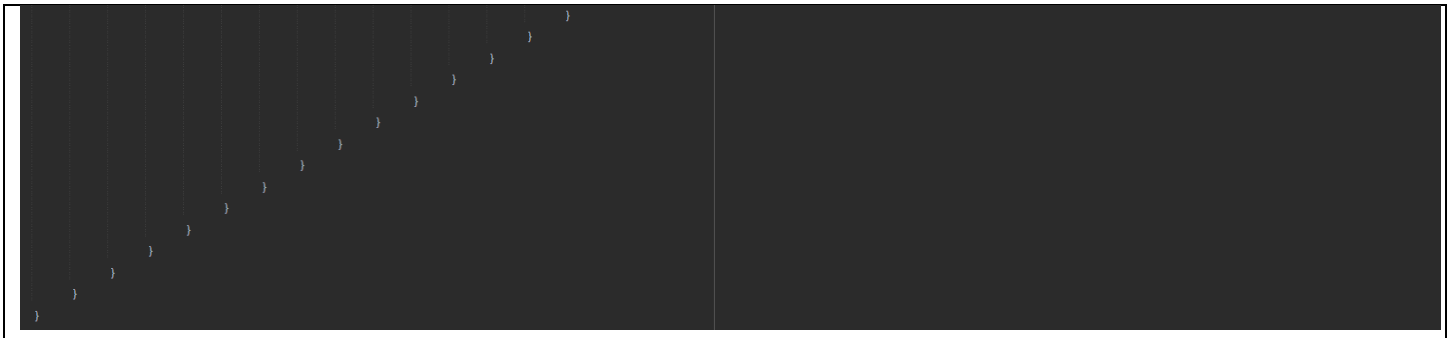
```
        } else if ((user.getOpt2Choice().equals(current.getOpt3Choice())) || current.getOpt3Choice().equals("any")) {
            if (user.getOpt2Mark() >= current.getOpt3Mark()) {
                if ((user.getOpt3Choice().equals(current.getOpt2Choice())) || current.getOpt2Choice().equals("any")) {
                    if (user.getOpt3Mark() >= current.getOpt2Mark()) {
                        //132
                        if (user.getLo() >= 50) {
                            if (user.getAps() >= current.getAps()) {
                                temp[tempSize] = requirements[i];
                                tempSize++;
                            }
                        }
                    }
                }
            }
        }
    }

} else if ((user.getOpt1Choice().equals(current.getOpt2Choice())) || current.getOpt2Choice().equals("any")) {
    if (user.getOpt1Mark() >= current.getOpt2Mark()) {

        if ((user.getOpt2Choice().equals(current.getOpt1Choice())) || current.getOpt1Choice().equals("any")) {
            if (user.getOpt2Mark() >= current.getOpt1Mark()) {
                if ((user.getOpt3Choice().equals(current.getOpt3Choice())) || current.getOpt3Choice().equals("any")) {
                    if (user.getOpt3Mark() >= current.getOpt3Mark()) {
                        //213
                        if (user.getLo() >= 50) {
                            if (user.getAps() >= current.getAps()) {
                                temp[tempSize] = requirements[i];
                                tempSize++;
                            }
                        }
                    }
                }
            }
        } else if ((user.getOpt2Choice().equals(current.getOpt3Choice())) || current.getOpt3Choice().equals("any")) {
            if (user.getOpt2Mark() >= current.getOpt3Mark()) {
                if ((user.getOpt3Choice().equals(current.getOpt1Choice())) || current.getOpt1Choice().equals("any")) {
                    if (user.getOpt3Mark() >= current.getOpt1Mark()) {
                        //231
                        if (user.getLo() >= 50) {
                            if (user.getAps() >= current.getAps()) {
                                temp[tempSize] = requirements[i];
                                tempSize++;
                            }
                        }
                    }
                }
            }
        }
    }

} else if ((user.getOpt1Choice().equals(current.getOpt3Choice())) || current.getOpt3Choice().equals("any")) {
    if (user.getOpt1Mark() >= current.getOpt3Mark()) {

        if ((user.getOpt2Choice().equals(current.getOpt1Choice())) || current.getOpt1Choice().equals("any")) {
            if (user.getOpt2Mark() >= current.getOpt1Mark()) {
                if ((user.getOpt3Choice().equals(current.getOpt2Choice())) || current.getOpt2Choice().equals("any")) {
                    if (user.getOpt3Mark() >= current.getOpt2Mark()) {
                        //312
                        if (user.getLo() >= 50) {
                            if (user.getAps() >= current.getAps()) {
                                temp[tempSize] = requirements[i];
                                tempSize++;
```



```
        // If the user has met the requirement for at least 1 degree:
        if (tempSize != 0) {
            // Create a new output integer array, with size being equal to that of the
temporary array
            int[] output = new int[tempSize];
            // Get the ID of every degree in the temporary array and add it to the
output array
            for (int i = 0; i < tempSize; i++) {
                output[i] = temp[i].getID();
            }
            // Return the array
            return output;
        } else { // Otherwise
            // Return nothing
            return null;
        }
    }

    /**
     * For a degree's dedicated tab, this method converts the degree's
     * requirement into a readable format
     *
     * @param req The Requirement Object to convert
     * @return The requirement as a readable format
     */
    public String toString(Requirement req) {
        String output = "";

        output += "Required APS:\t\t" + req.getAps();
        output += "\n\n";
        output += "Required Subjects:\n";

        if (req.getHlChoice().equals("eng")) {
            output += "English HL:\t\t" + req.getHlMark() + "%\n";
        }
        if (req.getFalChoice().equals("eng")) {
            output += "English FAL:\t\t" + req.getFalMark() + "%\n";
        }
        if (req.getMathChoice().equals("cor")) {
            output += "Core Mathematics:\t" + req.getMathMark() + "%\n";
        } else if (req.getMathChoice().equals("lit")) {
            output += "Mathematical Literacy:\t" + req.getMathMark() + "%\n";
        } else if (req.getFalChoice().equals("tec")) {
            output += "Technical Mathematics:\t" + req.getMathMark() + "%\n";
        }
        if (!req.getOpt1Choice().equals("any")) {
            output += getSubject(req.getOpt1Choice()) + ":\t" + req.getOpt1Mark() +
"%\n";
        }
        if (!req.getOpt2Choice().equals("any")) {
            output += getSubject(req.getOpt2Choice()) + ":\t" + req.getOpt2Mark() +
"%\n";
        }
        if (!req.getOpt3Choice().equals("any")) {
```

```
            output += getSubject(req.getOpt3Choice()) + ":\t" + req.getOpt3Mark() +
"%\n";
        }

        return output;
    }

    /**
     * Return the requirement belonging to a specific degree
     *
     * @param degreeID The ID of the Degree to use
     * @return The Requirement belonging to the Degree
     */
    public Requirement getReqWithDegID(int degreeID) {
        // For each requirement in the class's array (Skip user marks, which is in the
first entry of the array)
        for (int i = 1; i < size + 1; i++) {
            // If a match is found:
            if (requirements[i].getDeg().getDegreeID() == degreeID) {
                // Return the Requirement object
                return requirements[i];
            }
        } // Otherwise
        // Return nothing
        return null;
    }
}
```

2.5 SavedDegrees

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package Managers;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Saien Naidu
 */
// This class represents the SavedDegree Object
public class SavedDegrees {

    // FIELDS
```

```
    private int[] degrees = new int[250];
    private int size;
    private final String filePath = "data\\SavedDegrees.txt";

    private final DegManager dm = new DegManager();

    // CONSTRUCTOR
    /**
     * Uses data from text file
     */
    public SavedDegrees() {
        try {
            File file = new File(filePath);
            Scanner fileSC = new Scanner(file);
            while (fileSC.hasNext()) {
                int degreeID = fileSC.nextInt();
                degrees[size] = degreeID;
                size++;
            }
        } catch (FileNotFoundException ex) {
            Logger.getLogger(SavedDegrees.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #03: SavedDegrees file not found");
        }
    }

    // GETTERS
    /**
     *
     * @return The IDs of the degrees the user has saved
     */
    public int[] getSavedDegrees() {
        return degrees;
    }

    /**
     *
     * @return The amount of degrees the user has saved
     */
    public int getSize() {
        return size;
    }

    /**
     * Create a table row model using the degrees that the user has saved
     *
     * @return The table row model
     */
    public Object[][] createTable() {
        Object[][] data = new Object[size][1];

        for (int i = 0; i < size; i++) {
            data[i][0] = dm.getDegWithID(degrees[i]).getName();
        }

        return data;
    }

    /**
     * Return the list of degrees that the user has saved whose names are
     * similar to the text provided
     *
     * @param mustContain The text that the degrees must be similar to
     * @return The table row model
     */
    public Object[][] createTable(String mustContain) {
        Object[][] data = new Object[size][1];
        int j = 0;
```

```
        for (int i = 0; i < size; i++) {
            String degName = dm.getDegWithID(degrees[i]).getName();
            if (degName.toLowerCase().contains(mustContain.toLowerCase())) {
                data[j][0] = degName;
                j++;
            }
        }

        return data;
    }

    /**
     * Remove a degree from the user's saved degrees
     *
     * @param index The index of the ID to remove
     */
    public void remove(int index) {
        FileWriter fileFW = null;
        try {
            for (int i = index; i < size; i++) {
                degrees[i] = degrees[i + 1];
            }
            size--;

            fileFW = new FileWriter(filePath);
            fileFW.write("");
            for (int i = 0; i < size; i++) {
                fileFW.append(degrees[i] + "\n");
            }

        } catch (IOException ex) {
            Logger.getLogger(SavedDegrees.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #13: Error while removing a saved degree from the
text file.");

        } finally {
            try {
                fileFW.close();
            } catch (IOException ex) {
                Logger.getLogger(SavedDegrees.class.getName()).log(Level.SEVERE, null,
ex);
                System.out.println("Error #13: Error while removing a saved degree from
the text file.");
            }
        }
    }

    /**
     * Add a degree to the list of the user's saved degrees
     *
     * @param ID The ID of the Degree
     */
    public void add(int ID) {
        FileWriter fileFW = null;
        try {
            degrees[size] = ID;
            size++;

            String write = "";
            for (int i = 0; i < size; i++) {
                write += degrees[i] + "\n";
            }

            fileFW = new FileWriter(filePath);
            fileFW.write(write);
        } catch (IOException ex) {
```

```
            Logger.getLogger(SavedDegrees.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Error #14: Error while adding a saved degree to the text
file.");
        } finally {
            try {
                fileFW.close();
            } catch (IOException ex) {
                Logger.getLogger(SavedDegrees.class.getName()).log(Level.SEVERE, null,
ex);
                System.out.println("Error #14: Error while adding a saved degree to the
text file.");
            }
        }

    }
}
```

3. UI Classes

3.1 FMain

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package FrontEnd;

// IMPORTS
import Managers.DegManager;
import Managers.FacManager;
import Managers.ReqManager;
import Managers.UniManager;
import Objects.University;
import Objects.Degree;
import Objects.Requirement;
import Managers.SavedDegrees;
import Objects.Faculty;
import Objects.Filter;
import java.awt.Color;
import javax.swing.JTable;

/**
 *
 * @author Saien Naidu
 */
public class FMain extends javax.swing.JFrame {

    // FIELDS
    // Manager Objects
    private final UniManager um = new UniManager();
    private final FacManager fm = new FacManager();
```

```
    private final DegManager dm = new DegManager();
    private final ReqManager rm = new ReqManager();
    private final SavedDegrees sd = new SavedDegrees();

    // Other frames
    private final FFilter frm_filters = new FFilter();

    // So that a new String[] does not need to be made each time a table is made
    private final String[] columnName = new String[1];

    // Keep track of the Degree currently being displayed in the Degree Tab
    private Degree degreeTab;

    // CONSTRUCTOR
    /**
     * Creates new form Main
     */
    public FMain() {
        initComponents();
        initUserMarksTab();
        initSavedDegreesTable();
        initUniversityTable();
        initFinderTable();
        initFiltersFrame();
    }

    //PROPERTIES
    // <editor-fold defaultstate="collapsed" desc="Initialization code">
    /**
     * This method is called from within the constructor to initialize the form.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    ...
    }// </editor-fold>

    // Sets up the User Marks tab
    private void initUserMarksTab() {
        // Fetch user marks
        Requirement userMarks = rm.getUserMarks();

        // For each subject, fetch the mark and choice
        // Select Radio Buttons according to the user's subject choices
        // Home Language
        spn_hl.setValue(userMarks.getHlMark());
        if (userMarks.getHlChoice().equals("eng")) {
            rBtn_hlEng.setSelected(true);
            rBtn_falEng.setEnabled(false);
        } else {
            rBtn_hlOther.setSelected(true);
        }

        // First Add Lang
        spn_fal.setValue(userMarks.getFalMark());
        if (userMarks.getFalChoice().equals("eng")) {
            rBtn_falEng.setSelected(true);
            rBtn_hlEng.setEnabled(false);
        } else {
            rBtn_falOther.setSelected(true);
        }

        // Mathematics
        spn_math.setValue(userMarks.getMathMark());
        switch (userMarks.getMathChoice()) {
            case "cor" ->
                rBtn_mathC.setSelected(true);
            case "lit" ->
```

```
                rBtn_mathL.setSelected(true);
            default ->
                rBtn_mathT.setSelected(true);
        }

        // 1st Subject Choice
        spn_opt1.setValue(userMarks.getOpt1Mark());
        String opt1C = rm.getSubject(userMarks.getOpt1Choice());
        cbx_opt1.setSelectedItem(opt1C);

        // 2nd Subject Choice
        spn_opt2.setValue(userMarks.getOpt2Mark());
        String opt2C = rm.getSubject(userMarks.getOpt2Choice());
        cbx_opt2.setSelectedItem(opt2C);

        // 3rd Subject Choice
        spn_opt3.setValue(userMarks.getOpt3Mark());
        String opt3C = rm.getSubject(userMarks.getOpt3Choice());
        cbx_opt3.setSelectedItem(opt3C);

        // LO
        spn_lo.setValue(userMarks.getLo());

        // Make the saved indicator invisible
        lbl_saveIndicator.setVisible(false);
    }

    // Initializes the list of the user's saved degrees
    private void initSavedDegreesTable() {
        columnName[0] = "Degrees";
        JTable temp = new JTable(sd.createTable(), columnName);
        tbl_saved.setModel(temp.getModel());
    }

    // Initializes the list of all the Universities in the DB
    private void initUniversityTable() {
        columnName[0] = "Universities";
        JTable temp = new JTable(um.createTable(), columnName);
        tbl_browse.setModel(temp.getModel());
        btn_filter.setBackground(Color.lightGray);
    }

    // Initializes the list of all the Degrees in the DB
    private void initFinderTable() {
        columnName[0] = "Degrees";
        JTable temp = new JTable(dm.createTable(), columnName);
        tbl_finder.setModel(temp.getModel());
    }

    // Sets the filter frame to be invisible when the program is opened.
    private void initFiltersFrame() {
        frm_filters.setVisible(false);
    }
    // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="Update code">
    // Update the list of Saved Degrees based on text
    private void updateSavedDegreesTable(String input) {
        columnName[0] = "Degrees";
        JTable temp = new JTable(sd.createTable(input), columnName);
        tbl_saved.setModel(temp.getModel());
    }

    // Update the list of Universities in based on an array of Universities
    private void updateUniversityTable(University[] input) {
        columnName[0] = "Universities";
        JTable temp = new JTable(um.createTable(input), columnName);
```

```
        tbl_browse.setModel(temp.getModel());
    }

    // Update the list of Degrees based on an array of Degrees
    private void updateFinderTable(Degree[] input) {
        columnName[0] = "Degrees";
        JTable temp = new JTable(dm.createTable(input), columnName);
        tbl_finder.setModel(temp.getModel());
    }

    // Display a new University in the dedictaed university tab
    private void updateUniversityTab(University uni) {
        lbl_uni.setText(uni.getName());
        txA_rank.setText(uni.getRank() + "");
        txA_location.setText(uni.getLocation());
        txA_estb.setText("" + uni.getEstb());
        txA_students.setText("" + uni.getStudents());
        txA_accRate.setText(uni.getAccRate() + "%");
        txA_uniDesc.setText(uni.getDesc());

        columnName[0] = "Faculties";
        JTable temp = new JTable(fm.createTable(uni), columnName);
        tbl_uniFac.setModel(temp.getModel());
    }

    // Display a new Faculty in the dedictaed faculty tab
    private void updateFacultyTab(Faculty fac) {
        lbl_fac.setText(fac.getName());
        txA_facDesc.setText(fac.getDesc());

        columnName[0] = "Degrees";

        JTable temp = new JTable(dm.createTable(fac), columnName);
        tbl_facDeg.setModel(temp.getModel());
    }

    // Display a new Degree in the dedictaed degree tab
    private void updateDegreeTab(Degree deg) {
        // Update the variable storing the current Degree being shown
        degreeTab = deg;

        lbl_deg.setText(deg.getName());
        txA_degDesc.setText(deg.getDesc());

        // Fetch the Requirement Object linked to this degree
        Requirement req = rm.getReqWithDegID(deg.getDegreeID());

        txA_req.setText(rm.toString(req));

        // If this degree is bookmarked, disable the save button
        int[] saved = sd.getSavedDegrees();
        boolean isSaved = false;
        for (int i = 0; i < saved.length; i++) {
            if (deg.getDegreeID() == saved[i]) {
                isSaved = true;
            }
        }
        if (isSaved) {
            btn_saveDeg.setText("Saved");
            btn_saveDeg.setEnabled(false);
        } else {
            btn_saveDeg.setEnabled(true);
            btn_saveDeg.setText("Save");
        }
    }

    // <editor-fold defaultstate="collapsed" desc="Degree Finder">
```

```
    /**
     * Update the list of degrees based on the filter provided
     *
     * @param f The filter to use
     */
    public void updateFinderTable(Filter f) {

        // If a filter is applied:
        if (f.isCommerce() || f.isEastcape() || f.isEngineering() || f.isExclude() ||
f.isFreestate() || f.isGauteng() || f.isHealth() || f.isHumanities() || f.isInclude() ||
f.isKzn() || f.isLaw() || f.isLimpopo() || f.isMpumalanga() || f.isNorthwest() ||
f.isSciences() || f.isUseMarks() || f.isWestcape()) {
            // Hide the frame
            this.setVisible(false);

            // Update table
            columnName[0] = "Degrees";
            JTable temp = new JTable(dm.createTable(dm.degreeFinder(f)), columnName);
            tbl_finder.setModel(temp.getModel());

            // Open the right tab
            tbdPn_main.setSelectedIndex(4);
            // Set the filter button color to show that a filter is being applied
            btn_filter.setBackground(Color.white);
            // Disable the search bar
            System.out.println("Searching for degrees while a filter is applied is not
implemented yet");
            txF_finder_search.setEnabled(false);
            // Show the frame
            this.setVisible(true);
        } else { // If no filter is applied:
            // Hide the frame
            this.setVisible(false);

            // Set the list of degrees to show all the degrees in the DB.
            initFinderTable();

            // Open the right tab
            tbdPn_main.setSelectedIndex(4);
            // Set the filter button color to show that a filter is NOT being applied
            btn_filter.setBackground(Color.darkGray);
            // Enable the search bar
            System.out.println("Searching for degrees while a filter is applied is not
implemented yet");
            txF_finder_search.setEditable(true);
            // Show the frame
            this.setVisible(true);
        }

    }
    // </editor-fold>

    // </editor-fold>
    // When [Degree Finder] is clicked, navigate to the 'Degree Finder' tab.
    private void btn_finderActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(4);
    }

    // When [Save] is clicked on 'Your Results' tab, save the changes made to the user's
marks to the text file
    private void btn_saveActionPerformed(java.awt.event.ActionEvent evt) {
        // For every subject, store both the...
        // ...mark:
        int hlM = (int) spn_hl.getValue();
        // ...and the choice:
        String hlC;
        if (rBtn_hlEng.isSelected()) {
```

```
            hlC = "eng";
        } else {
            hlC = "oth";
        }

        int falM = (int) spn_fal.getValue();
        String falC;
        if (rBtn_falEng.isSelected()) {
            falC = "eng";
        } else {
            falC = "oth";
        }

        int mathM = (int) spn_math.getValue();
        String mathC;
        if (rBtn_mathC.isSelected()) {
            mathC = "cor";
        } else if (rBtn_mathL.isSelected()) {
            mathC = "lit";
        } else {
            mathC = "tec";
        }

        // For every optional subject...
        // ...store the mark:
        int opt1M = (int) spn_opt1.getValue();
        // ...store the choice by fetching it from the combobox and abbreviating it
using a method:
        String opt1C = rm.getAbbreviation((String) cbx_opt1.getSelectedItem());

        int opt2M = (int) spn_opt2.getValue();
        String opt2C = rm.getAbbreviation((String) cbx_opt2.getSelectedItem());

        int opt3M = (int) spn_opt3.getValue();
        String opt3C = rm.getAbbreviation((String) cbx_opt3.getSelectedItem());

        // Only store the mark for LO
        int lo = (int) spn_lo.getValue();

        // Save marks
        rm.setUserMarks(hlM, hlC, falM, falC, mathM, mathC, opt1M, opt1C, opt2M, opt2C,
opt3M, opt3C, lo);

        // Show that the marks have been saved
        lbl_saveIndicator.setVisible(true);
    }

    // When [Remove] is clicked on 'Saved Degrees', Remove the bookmark on selected
degree
    private void btn_saved_removeActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch the index of the selected row
        int index = tbl_saved.getSelectedRow();

        // If the user actually has a row selected:
        if (index != -1) {
            // Delete the degree from the text file
            sd.remove(index);
            // Update the table
            initSavedDegreesTable();
        }
    }

    // When [View] is clicked on the 'Saved Degrees' tab, display the selected degree's
information on the dedicated Degree tab.
    private void btn_saved_viewActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch index of selected row
        int selectedRow = tbl_saved.getSelectedRow();
```

```
        // If the user actually selected a row:
        if (selectedRow != -1) {
            // Fetch the list of Degree IDs
            int[] tableArr = sd.getSavedDegrees();

            // Instantiate necessary Objects
            Degree deg = dm.getDegWithID(tableArr[selectedRow]);
            Faculty fac = deg.getFac();
            University uni = fac.getUni();

            // Update dedicated tabs as needed
            updateUniversityTab(uni);
            updateFacultyTab(fac);
            updateDegreeTab(deg);

            // Switch focus from main tabs to dedicated
            changeToDedicatedTabs(2);
        }
    }

    // When [View] is clicked on the 'Universities' tab, display the selected
university's information on the dedicated University tab.
    private void btn_browse_viewActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch index of selected row
        int selectedRow = tbl_browse.getSelectedRow();

        // If a row is actually selected:
        if (selectedRow != -1) {
            // Fetch the list of Universities which are currently being displayed
            int[] tableArr = um.getTableArr();

            // Instantiate necessary Objects
            University uni = um.getUniWithID(tableArr[selectedRow]);

            // Update dedicated tabs as needed
            updateUniversityTab(uni);

            // Switch focus from main tabs to dedicated
            changeToDedicatedTabs(0);
        }

    }

    // When [View] is clicked on the 'Degree Finder' tab, display the selected Degree's
information on the dedicated Degree tab.
    private void btn_finder_viewActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch index of selected row
        int selectedRow = tbl_finder.getSelectedRow();

        // If a row is actually selected:
        if (selectedRow != -1) {
            // Fetch the list of Degrees which are currently being displayed
            int[] tableArr = dm.getTableArr();

            // Instantiate necessary Objects
            Degree deg = dm.getDegWithID(tableArr[selectedRow]);
            Faculty fac = deg.getFac();
            University uni = fac.getUni();

            // Update dedicated tabs as needed
            updateUniversityTab(uni);
            updateFacultyTab(fac);
            updateDegreeTab(deg);

            // Switch focus from main tabs to dedicated
            changeToDedicatedTabs(2);
```

```
        }
    }

    // When [View] is clicked on the dedicated University tab, display the selected
Faculty's information on the dedicated Faculty tab.
    private void btn_facViewActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch index of selected row
        int selectedRow = tbl_uniFac.getSelectedRow();

        // If a row is actually selected:
        if (selectedRow != -1) {
            // Fetch the list of Faculties which are currently being displayed
            int[] tableArr = fm.getTableArr();

            // Instantiate necessary Objects
            Faculty fac = fm.getFacWithID(tableArr[selectedRow]);

            // Update dedicated tabs as needed
            updateFacultyTab(fac);

            // Nevigate to Faculty dedicated tab
            changeToDedicatedTabs(1);
        }
    }

    // When [View] is clicked on the dedicated Faculty tab, display the selected
Degree's information on the dedicated Degree tab.
    private void btn_degViewActionPerformed(java.awt.event.ActionEvent evt) {
        // Fetch index of selected row
        int selectedRow = tbl_facDeg.getSelectedRow();

        // If a row is actually selected:
        if (selectedRow != -1) {
            // Fetch the list of Faculties which are currently being displayed
            int[] tableArr = dm.getTableArr();

            // Instantiate necessary Objects
            Degree deg = dm.getDegWithID(tableArr[selectedRow]);

            // Update dedicated tabs as needed
            updateDegreeTab(deg);

            // Nevigate to Degree dedicated tab
            changeToDedicatedTabs(2);
        }
    }

    // When [Save] is clicked on the dedicated Degree tab, bookmark the degree
    private void btn_saveDegActionPerformed(java.awt.event.ActionEvent evt) {
        // Using the degree stored in the class variable 'degreeTab'...
        // (which changes every time a different degree is displayed on the dedicated
degree tab)...
        // bookmark the degree
        sd.add(degreeTab.getDegreeID());
        // Update saved degrees table
        initSavedDegreesTable();
        // Show the user that the degree has been saved
        btn_saveDeg.setText("Saved");
        btn_saveDeg.setEnabled(false);
    }

    // When 'Your Results' is clicked, navigate to the 'Your Results' tab.
    private void btn_resultsActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(1);
    }

    // When 'Your saved degrees' is clicked, navigate to the 'Saved Degrees' tab.
```

```
    private void btn_savedActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(2);
    }

    // When 'Browse Universities' is clicked, navigate to the 'Universities' tab.
    private void btn_browseActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(3);
    }

    // Navigate back to the main menu
    private void btn_results_backActionPerformed(java.awt.event.ActionEvent evt) {
        // Hide the save indicator
        lbl_saveIndicator.setVisible(false);

        // Navigate to the main menu
        tbdPn_main.setSelectedIndex(0);
    }

    // Since you can only take English once...
    // If the English FAL radio button is chosen, deactivate English HL
    private void rBtn_falEngActionPerformed(java.awt.event.ActionEvent evt) {
        rBtn_hlEng.setEnabled(false);
    }

    // Since you can only take English once...
    // If the English HL radio button is chosen, deactivate English FAL
    private void rBtn_hlEngActionPerformed(java.awt.event.ActionEvent evt) {
        rBtn_falEng.setEnabled(false);
    }

    // If the user chooses 'Other' as their HL, activate English FAL
    private void rBtn_hlOtherActionPerformed(java.awt.event.ActionEvent evt) {
        rBtn_falEng.setEnabled(true);
    }

    // If the user chooses 'Other' as their FAL, activate English HL
    private void rBtn_falOtherActionPerformed(java.awt.event.ActionEvent evt) {
        rBtn_hlEng.setEnabled(true);
    }

    // When [Back] is clicked on 'Saved Degrees', return to main menu
    private void btn_saved_backActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(0);
    }

    // When user types into the search bar on 'Saved Degrees', update the list according
to the search
    private void txF_saved_searchCaretUpdate(javax.swing.event.CaretEvent evt) {
        String input = txF_saved_search.getText();
        updateSavedDegreesTable(input);
    }

    // When [Back] is clicked on 'Universities', return to main menu
    private void btn_browse_backActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(0);
    }

    // When user types into the search bar on 'Universities', update the list according
to the search
    private void txF_browse_searchCaretUpdate(javax.swing.event.CaretEvent evt) {
        String input = txF_browse_search.getText();

        // If the search is not blank:
        if (!input.equals("")) {
            // Get results using DB query
            University[] results = um.getUniWithName("SELECT * FROM University_Table "
                    + "WHERE UniversityName LIKE '*" + input + "*';");
```

```
            // Update table based on fetched results
            updateUniversityTable(results);
        } else { // If it is blank:
            // Update table to show all universities within the DB
            initUniversityTable();
        }
    }

    // When [Back] is clicked on 'Degree Finder', return to main menu
    private void btn_finder_backActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_main.setSelectedIndex(0);
    }

    // When user types into the search bar on 'Degree Finder', update the list according
to to the search
    private void txF_finder_searchCaretUpdate(javax.swing.event.CaretEvent evt) {
        // If no filter is applied:
        if (btn_filter.getBackground() != Color.white) {
            // Fetch search
            String input = txF_finder_search.getText();
            // If search is not blank:
            if (!input.equals("")) {
                // Get results using DB query
                Degree[] results = dm.getDegWithQuery("SELECT * FROM Degree_Table "
                        + "WHERE DegreeName LIKE '*" + input + "*';");
                // Update table based on these results.
                updateFinderTable(results);
            } else { // If it is:
                // Update table to show all universities within the DB
                initFinderTable();
            }
        }
    }

    // When [Filters] is clicked, open the filter frame
    private void btn_filterActionPerformed(java.awt.event.ActionEvent evt) {
        this.setVisible(false);
        frm_filters.setEnabled(true);
        frm_filters.setVisible(true);
    }

    // When [Back] is clicked on dedictaed University tab, return to main menu
    private void btn_uni_backActionPerformed(java.awt.event.ActionEvent evt) {
        // Update university table to show all universities in DB
        initUniversityTable();
        // Update degree table to show all degrees in DB
        initFinderTable();
        tbdPn_main.setSelectedIndex(0);
    }

    // When [Back] is clicked on dedictaed Faculty tab, return to dedicated University
tab
    private void btn_fac_backActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_dedicated.setSelectedIndex(0);
    }

    // When [Back] is clicked on dedictaed Degree tab, return to dedicated Faculty tab
    private void btn_degree_backActionPerformed(java.awt.event.ActionEvent evt) {
        tbdPn_dedicated.setSelectedIndex(1);
    }

    // Switch focus to dedicated tabs
    private void changeToDedicatedTabs(int tab) {
        tbdPn_main.setSelectedIndex(5);
        tbdPn_dedicated.setSelectedIndex(tab);
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code
(optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default
look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException | javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FMain.class.getName()).log(java.util.logging.Level.SE
VERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(() -> {
            new FMain().setVisible(true);
        });
    }

    // Variables declaration - do not modify
    ...
    // End of variables declaration
}
```

3.2 FFilter

```
/*
 * Copyright (C) 2025 Saien Naidu
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package FrontEnd;

// IMPORTS
```

```
import Managers.UniManager;
import Objects.Filter;
import javax.swing.JComboBox;

/**
 *
 * @author Saien Naidu
 */
public class FFilter extends javax.swing.JFrame {

    // FIELDS
    // University Manager
    private final UniManager um = new UniManager();

    // CONSTRUCTOR
    /**
     * Creates new form Filters
     */
    public FFilter() {
        initComponents();
        initComboBoxes();
    }

    // PROPERTIES
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    ...
    }// </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="Initialization code">
    // Creates and sets the models of the combo boxes
    private void initComboBoxes() {
        JComboBox uni1 = new JComboBox(um.getAll());
        cbx_uni1.setModel(uni1.getModel());
        JComboBox uni2 = new JComboBox(um.getAll());
        cbx_uni2.setModel(uni2.getModel());
        JComboBox uni3 = new JComboBox(um.getAll());
        cbx_uni3.setModel(uni3.getModel());
    }
    // </editor-fold>

    // When [x] is clicked, send filter to main
    private void btn_closeActionPerformed(java.awt.event.ActionEvent evt) {

        // Use user marks?
        boolean useMarks = xbx_useMarks.isSelected();

        // Use location of?
        boolean kzn = xbx_kzn.isSelected();
        boolean gauteng = xbx_gauteng.isSelected();
        boolean eastcape = xbx_eastcape.isSelected();
        boolean westcape = xbx_westcape.isSelected();
        boolean freestate = xbx_freestate.isSelected();
        boolean northwest = xbx_northwest.isSelected();
        boolean mpumalanga = xbx_mpumalanga.isSelected();
        boolean limpopo = xbx_limpopo.isSelected();

        // Use faculty of?
        boolean commerce = xbx_commerce.isSelected();
        boolean engineering = xbx_engineering.isSelected();
        boolean health = xbx_health.isSelected();
        boolean law = xbx_law.isSelected();
```

```
        boolean humanities = xbx_humanities.isSelected();
        boolean sciences = xbx_sciences.isSelected();

        // Use university of?
        String uni1 = (String) cbx_uni1.getSelectedItem();
        String uni2 = (String) cbx_uni2.getSelectedItem();
        String uni3 = (String) cbx_uni3.getSelectedItem();

        // Include or exclude the selected universities
        // Only one of the two can be true, but both can be false
        boolean include;
        boolean exclude;

        // If no university selected:
        if ((uni1.equals("None"))
                && (uni2.equals("None"))
                && (uni3.equals("None"))) {
            // Set both to false
            include = false;
            exclude = false;
        } else { // Otherwise:
            // Set either as true based on user selection
            include = rBtn_include.isSelected();
            exclude = rBtn_exclude.isSelected();
        }

        // Create the filter object
        Filter filter = new Filter(useMarks, kzn, gauteng, eastcape, westcape,
freestate, northwest, mpumalanga, limpopo, commerce, engineering, health, law,
humanities, sciences, uni1, uni2, uni3, include, exclude);

        // Deactivate this frame
        this.setVisible(false);
        this.setEnabled(false);

        // Show main frame and send trigger
        FMain frm_main = new FMain();
        frm_main.updateFinderTable(filter);
        frm_main.setVisible(true);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code
(optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default
look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException | javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FFilter.class.getName()).log(java.util.logging.Level.
SEVERE, null, ex);
        }
```

```
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(() -> {
            new FFilter().setVisible(true);
        });
    }

    // Variables declaration - do not modify
    ...
    // End of variables declaration
}
```

4. dbDriver

```
package Driver;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author IVD Merwe, Saien Naidu
 */
public class dbDriver {

    private static final String DRIVER = "net.ucanaccess.jdbc.UcanaccessDriver";
    private static final String URL = "jdbc:ucanaccess://data/UniGo_DB.accdb";

    public Connection connection;
    private PreparedStatement statement;

    public dbDriver() {

        //Load
        try {
            Class.forName((DRIVER));
            System.out.println("Driver loaded.");
        } catch (ClassNotFoundException c) {
            System.out.println("Error #01: Driver class not found.");
        }

        //Connect
        try {
            connection = DriverManager.getConnection(URL);
            System.out.println("Driver connected.");
        } catch (SQLException e) {
            System.out.println("Error #02: SQL Connection failed.");
        }

    }

    //SELECT
    public ResultSet query(String query) throws SQLException {

        statement = connection.prepareStatement(query, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        return statement.executeQuery();
```

```
    }

    //INSERT, UPDATE, DELETE
    public void update(String query) throws SQLException {

        statement = connection.prepareStatement(query, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        statement.executeUpdate();
        statement.close();

    }
}
```