Spring 2022 OOP Final Project Report

Graphical User Interface Recommender System

By

Sami Cemek and Sebastian Mark

Description and Motivation

After brainstorming some ideas, we both came to an agreement that some recommender system would be an interesting project to create. Since we are both gamers, creating a simple tic-tac-toe game will not satisfy a true gamer's heart. Therefore, a more pragmatic project would not only be more useful to us but it could potentially become a design that we could progressively work on throughout the future. So, since one of the many stresses of a college student is deciding what classes are available and which one will be beneficial towards our current area of concentration (AOC). The decision was simple; we wanted to create a graphical user interface (GUI) that would allow the user to select previously taken courses from a list, and that would ultimately output a suggestive list of courses for the student to enroll in. Now, this is based on current courses taken and the prerequisites for those classes. Nevertheless, we wanted to create a program that would be beneficial not only to our lives, but that could possibly help us and others with the tough task of organizing semester schedules.

Dataset

The datasets used are read from a csv file. In theory, our data set can be anything we like, because each class is read from a simple .txt file. Meanwhile, this is extremely beneficial for the reason that if we want to update our system or list, all we need to do is adjust the current file (csv). Therefore, this provides a lot of flexibility in what our program can display. Additionally, the current dataset is a list of computer science courses New College of Florida has to offer. We created a system that is easily integrated and updated without making changes to the code (modular design). In the future, we can easily add more .txt files for other AOCs if we want to expand our project.

Algorithm

Sebastian:

There is no particular algorithmic implementation as far as an “algorithm”. Meaning, that the system is run from a superclass called JFrame. Moreover, JFrame is the container that the other classes not only extend from but actually go inside. So, this is the building block of the GUI, in which we have a frame and we can add certain attributes to that frame. Since we have a superclass we can use inheritance to extend as many classes as we like from the parent class. Therefore, we can reuse the same methods or implement new features, e.g. new buttons and functionality for each feature. The sample screenshot illustrates this idea:

```
public class ComputerSystem extends JFrame {

    public ComputerSystem(JTextField nameField, JLabel result){

        JLabel label = new JLabel( text: "Computer Systems", JLabel.CENTER);
        label.setFont(new Font( name: "Select the courses you have taken", Font.PLAIN,  size: 40));

        JFrame frame;
        frame = new JFrame();
        frame.setTitle("Area of Focus");

        setTitle("New College Computer Science Courses");
        setSize( width: 1200,  height: 800);

        GridLayout gridLayout = new GridLayout( rows: 0, cols: 1);
        setLayout(gridLayout);

        File file = new File( pathname: "src/ComputerSystem.txt");
        StringBuilder builder = new StringBuilder();
        try {
            Scanner scanner = new Scanner(file);

            while(scanner.hasNextLine()){
```

The core principles being used are inheritance, object instantiation, I/O handling, and polymorphism.

```
File file = new File( pathname: "src/ComputerSystem.txt");
StringBuilder builder = new StringBuilder();
try {
    Scanner scanner = new Scanner(file);

    while(scanner.hasNextLine()){
        builder.append(scanner.nextLine() + ",");
    }
    scanner.close();

}catch(FileNotFoundException e) {

    System.out.println(e.getMessage());
}

String choices = builder.toString();

String [] fileForTheBox = choices.split( regex: ",");

JComboBox box = new JComboBox(fileForTheBox);
box.setSize( width: 300,  height: 800);
box.setBackground(Color.orange);
box.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        String name = (String) box.getSelectedItem();

        if(result.getText().equals("")){
            result.setText(name);

        }else {
            result.setText(result.getText() + ", " + name);
        }
    }
});
JButton nextButton = new JButton( text: "Next");
nextButton.setLayout(new FlowLayout(FlowLayout.CENTER));

JButton removeButton = new JButton( text: "Remove");
removeButton.setLayout(new FlowLayout(FlowLayout.CENTER));

JPanel panel = new JPanel();
panel.add(nextButton);
panel.add(removeButton);

removeButton.addActionListener(new ActionListener() {
```

Moreover, we can implement other characteristics by using the Java's built in classes like JFrame and JPanel etc…, to customize the GUI to the user's preferences, meanwhile incorporating the object oriented fundamentals. Therefore, these core principles will help with reusability with methods, making our overall code more dynamic.

Sami:

I did not implement an existing algorithm, rather I looked at some examples of existing algorithms (listed on references) to get an idea of what I can do and implemented my own. Currently, this system is console-based, meaning that you see most of the outputs on the console. The recommender class takes the final output of the GIU (CSV file) and in order to make comparisons, first we need to split at each comma (because the output is a string separated by commas), and afterwards put them into an ArrayList. The reason being that I used ArrayList is because I wanted to have a dynamic data structure. I cannot change the length of the array once it is created. For the core computer science courses there is a sequence of events that need to take place first. Meaning, a person needs to have the proper prerequisites before they can enroll into a class. So, I need to consider the prerequisites and terms offered. For example, if the user only took an introduction to Python class, then by the sequence the recommender system would say "you should take OOP, Discrete Math, OOD, Algorithms and Software Engineering next semester." We have an ArrayList that stores the classes taken and compares them by the Core Course ArrayList. If a class in classTaken Arraylist is also in Core course ArrayList then we remove that class from core course Arraylist and finally print whatever is left on core course Arraylist.

For each elective area, I needed to add additional code to the GUI (see the screenshot on the next page). So I look at how many classes are selected from GUI by looking at how many classes are selected using the dropbox. Everytime the user selects a class, I increase my counter by 1 (countChecked) and print whatever is selected to the console. There are a certain number of elective area classes you need to take to meet the requirements. These requirements are: 2 Applications, 1 AI, 2 Mathematics, 2 Systems, 1 Languages, 1 Theory. I declared these numbers as a field on each related elective area class. Then I compare if countChecked is greater than or equal to required course count. If it is, that means you already met this requirement and don't need to take a class from that area. If not, for Theory and Languages, you can take any of the classes (since required course count is 1 for these two). So I print all of the classes from the related txt file. For the other elective areas, I remove the classTaken from all possible classes in that field and print what is left(similar to core CS courses).

The parts that I added to GUI for each elective area (highlighted with blue):

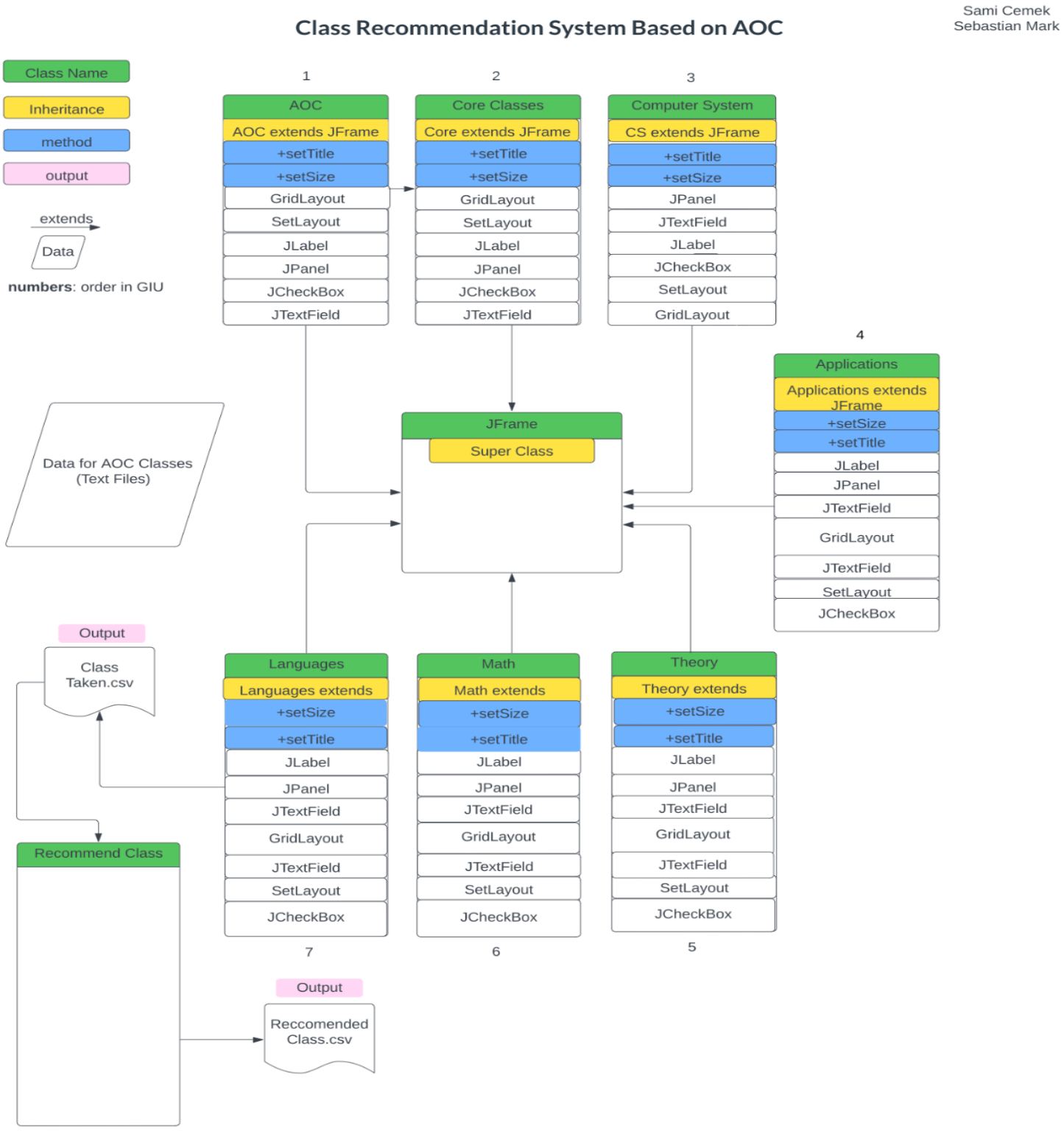
```
// ActionListener makes it so the user can actually click the button
box.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        String name = (String) box.getSelectedItem();
        // This if statement is saying that if we have the first element then a comma is not needed
        // Then it will just pass the name
        if(result.getText().equals(anObject: "")){
            result.setText(name);

            if(e.getSource()==box){
                System.out.println(box.getSelectedItem());
                countChecked++;
                System.out.println(countChecked);
            }
```

```
            // If it is not the first condition then everything after the first result will have a comma
        }else {
            result.setText(result.getText() + ", " + name);
            System.out.println(box.getSelectedItem());
            countChecked++;
            System.out.println(countChecked);
        }
    }
```

```
if(countChecked >= requiredCourseCount){
    System.out.println(x: "You already meet this requirment. You do not need to take a class from this area.");

}else{
    System.out.println(x: "For Application class you can take the following classes: ");

    List<String> list = new ArrayList<>(){{
        add(e: "C");
        add(e: "C++");
        add(e: "C#");
        add(e: "Compilers Tutorial (also in Theory)");
        add(e: "Embedded Systems (also in Systems)");
        add(e: "Functional Programming (no prerequisite)");
        add(e: "Hacking Tutorial (also in Systems)");
        add(e: "Mobile Applications (also in Applications)");
        add(e: "Principles of Programming Languages");
    }};

    for(String i : list){
        System.out.println(i);
    }
```

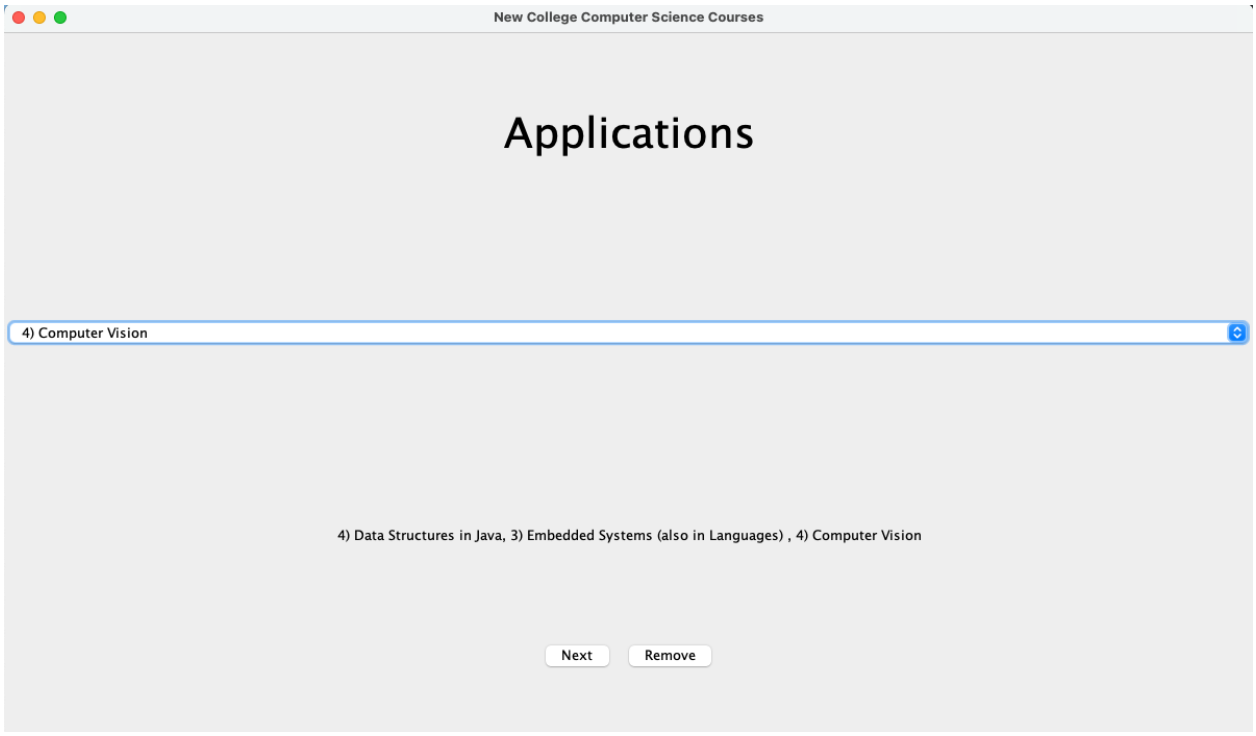
The core principles being used are inheritance, encapsulation, modular design, abstraction, interfaces and I/O handling.

UML Diagram


Class Recommendation System Based on AOC
Sami Cemek
Sebastian Mark
Class Name
Inheritance
method
output
extends
Data
numbers: order in GIU
1
AOC
AOC extends JFrame
+setTitle
+setSize
GridLayout
SetLayout
JLabel
JPanel
JCheckBox
JTextField
2
Core Classes
Core extends JFrame
+setTitle
+setSize
GridLayout
SetLayout
JLabel
JPanel
JCheckBox
JTextField
3
Computer System
CS extends JFrame
+setTitle
+setSize
JPanel
JTextField
JLabel
JCheckBox
SetLayout
GridLayout
4
Applications
Applications extends JFrame
+setSize
+setTitle
JLabel
JPanel
JTextField
GridLayout
JTextField
SetLayout
JCheckBox
Data for AOC Classes (Text Files)
JFrame
Super Class
Output
Class Taken.csv
Recommend Class
Languages
Languages extends
+setSize
+setTitle
JLabel
JPanel
JTextField
GridLayout
JTextField
SetLayout
JCheckBox
7
Math
Math extends
+setSize
+setTitle
JLabel
JPanel
JTextField
GridLayout
JTextField
SetLayout
JCheckBox
6
Theory
Theory extends
+setSize
+setTitle
JLabel
JPanel
JTextField
GridLayout
JTextField
SetLayout
JCheckBox
5
Output
Reccomended Class.csv

Sample Input /Output

First test:

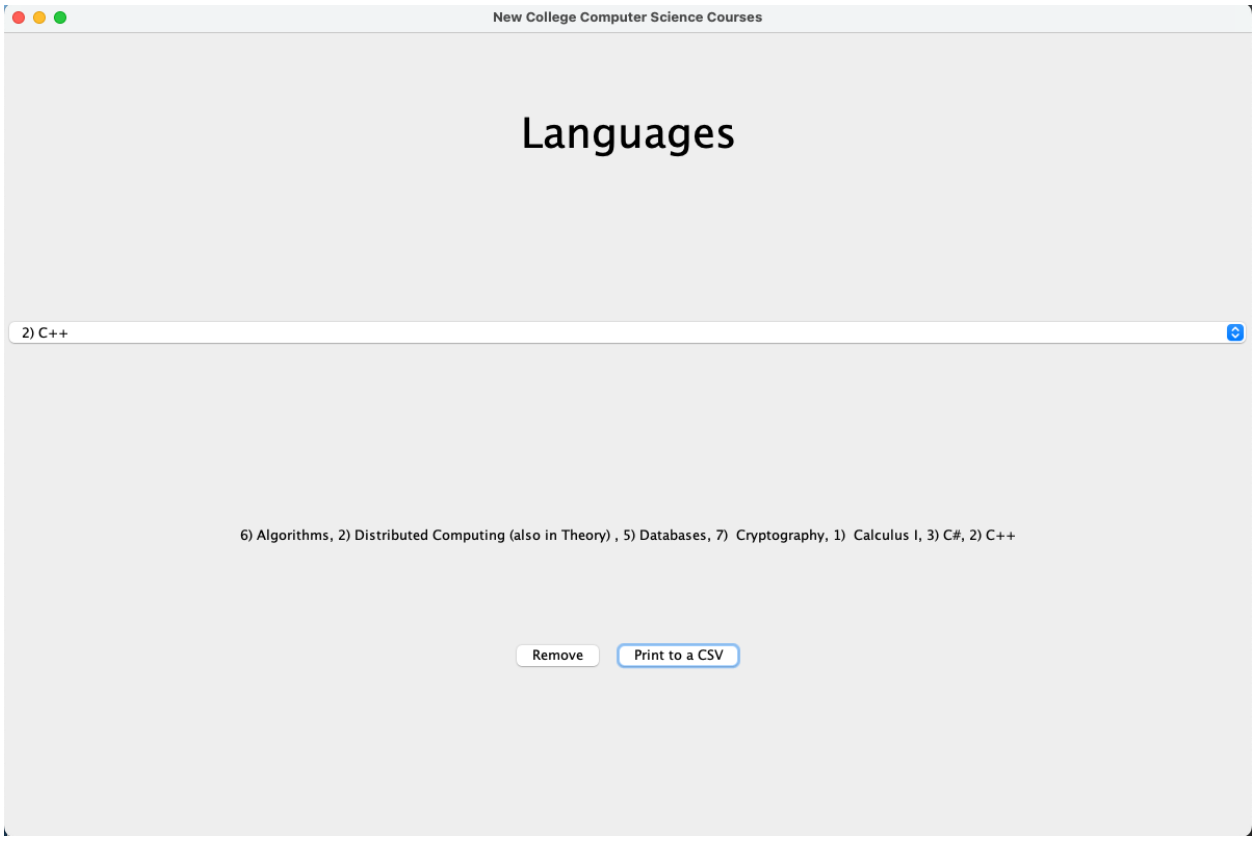


CSV output: `4) Data Structures in Java, 3) Embedded Systems (also in Languages)`

Output File: Class Taken.csv ---> Recommender.java takes this file as an input

```
Output read from the file: Introduction to Programming with Python, Discrete Mathematics, Object-Oriented Programming with Java

Array version: [Introduction to Programming with Python,  Discrete Mathematics,  Object-Oriented Programming with Java
]
So far you take 3 core classes. These are:
Introduction to Programming with Python
 Discrete Mathematics
 Object-Oriented Programming with Java

You need to take 4 more core courses
You should take
 Object-Oriented Programming with Java
```

Second test:

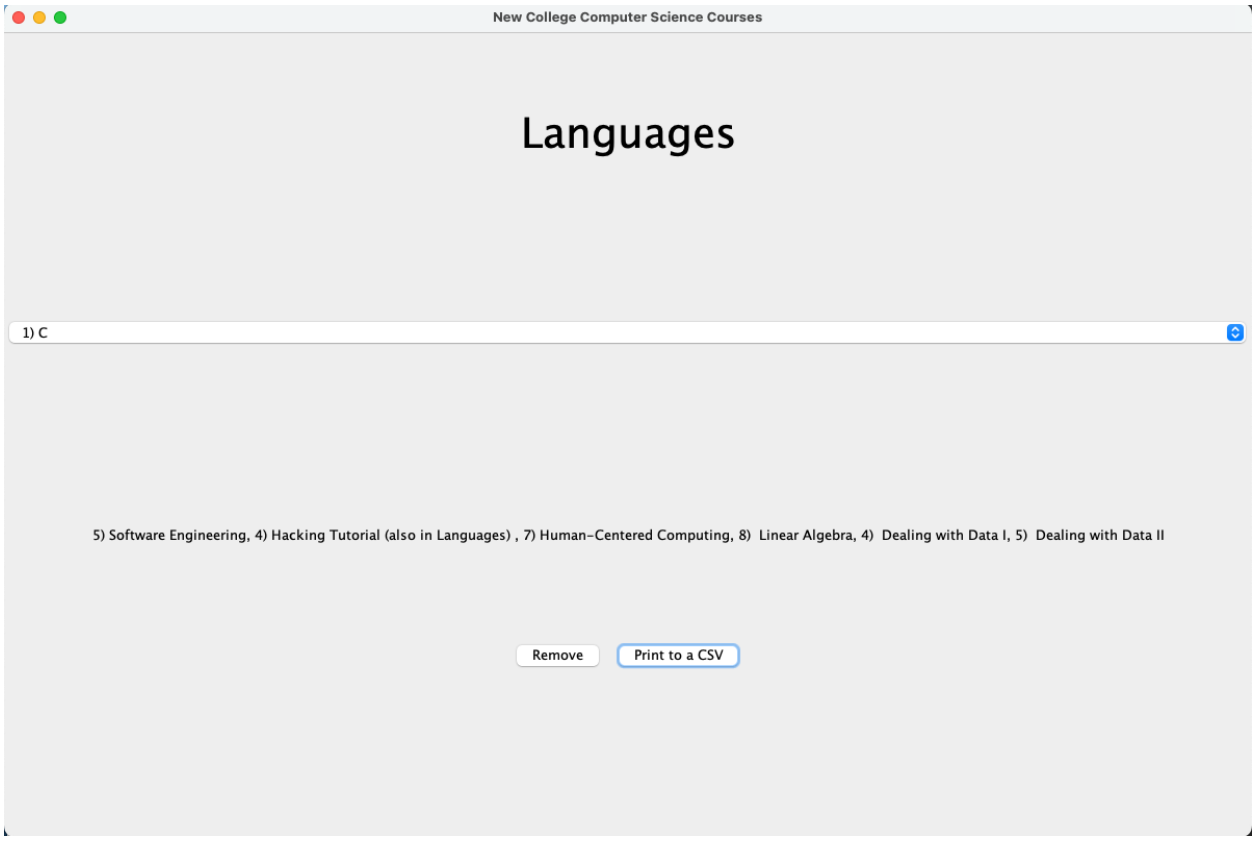


CSV output: `6) Algorithms, 2) Distributed Computing (also in Theory) , 5) Databases, 7) Cryptography, 1) Calculus I, 3) C#, 2) C++`

Output File: Class Taken.csv ---> Recommender.java takes this file as an input

```
Output read from the file: Introduction to Programming with Python, Discrete Mathematics, Object-Oriented Programming with Java

Array version: [Introduction to Programming with Python,  Discrete Mathematics,  Object-Oriented Programming with Java
]
So far you take 3 core classes. These are:
Introduction to Programming with Python
 Discrete Mathematics
 Object-Oriented Programming with Java

You need to take 4 more core courses
You should take
 Object-Oriented Programming with Java
```

Third test:



CSV output: `5) Software Engineering, 4) Hacking Tutorial (also in Languages) , 7) Human-Centered Computing, 8) Linear Algebra, 4) Dealing with Data I, 5) Dealing with Data II`

Output File: Class Taken.csv ---> Recommender.java takes this file as an input

```
Output read from the file: Introduction to Programming with Python, Discrete Mathematics, Object-Oriented Programming with Java

Array version: [Introduction to Programming with Python,  Discrete Mathematics,  Object-Oriented Programming with Java
]
So far you take 3 core classes. These are:
Introduction to Programming with Python
 Discrete Mathematics
 Object-Oriented Programming with Java

You need to take 4 more core courses
You should take
 Object-Oriented Programming with Java
```

Error/Exception Handling

Our system handles errors in a few ways, by first addressing the issue with a File class and StringBuilder. When reading from a CSV which is our data set we needed to implement try and catch. For example,

```
        setLayout(gridLayout);

        File file = new File( pathname: "src/ComputerSystem.txt");
        StringBuilder builder = new StringBuilder();
        try {
            Scanner scanner = new Scanner(file);

            while(scanner.hasNextLine()){
                builder.append(scanner.nextLine() + ",");
            }
            scanner.close();

        }catch(FileNotFoundException e) {

            System.out.println(e.getMessage());
        }

        String choices = builder.toString();

        String [] fileForTheBox = choices.split( regex: ",");

        JComboBox box = new JComboBox(fileForTheBox);
        box.setSize( width: 300,  height: 800);
        box.setBackground(Color.orange);
```

So, for the case when there is no file we “catch” and pass **FileNotFoundException e,** and print **e.getMessage.** You can see in multiple instances that this kind of error and exception handling are common throughout the other classes as well.

.

Shortcomings of Our Work

The graphical user interface came out functional, however, the display is quite primitive looking. When the user sees the first window, there is only the option to select one AOC, nevertheless adding another area of concentration could be easily added.

Meanwhile, we could add additional aesthetic features to the GUI, so it looks more presentable. However, those are minor details in the GUI that can be modified in the next version of the project. Although, to add more areas of focus that would imply many more classes in the project file. So, one of the shortcomings is if we wanted to have four to five different natural science fields, then we could potentially have twenty-five different classes inside of the IDE workspace.

Furthermore, one of the biggest issues we came across was the actual recommendation system. Although this kind of system can certainly become very advanced we tried to keep it simple for now; because considering our programming experience and while having a tight time constraint, it definitely hindered us from advancing the project more. Furthermore, after the user selects their courses from the drop-down window and presses the print to CSV button, the next step is the actual recommending classes, which at the moment is not functioning perfectly as we like.

Our initial thought was implementing a Hashmap for each elective area. Meaning, that for each one of our classes we would have a unique ID, and were planning to refer to those IDs when recommending classes. However, I changed my mind (Sami), and decided to use an arraylists instead. Currently for each elective our system is disregarding the prerequisites. So let’s say the user wants to take Theory of Computation, but there are two prerequisites: Discrete Mathematics and Data Structures. Therefore, our system will still recommend those classes even if you did not take Discrete Mathematics or Data Structures. Additionally, some classes fall in two categories. Those classes should then be addressed based on the user's credits needed. We specified those courses on each text file so the user can manually decide which courses are going to count towards the elective requirement.

In summary, we have created the GUI frame that has functional buttons, with a drop-down menu for the user to select the current courses taken. Additional, conditional operations were needed to prevent certain errors. For example, in the GUI the remove button would function even when the text field was empty, which caused an out of bounds error. Lastly, the recommender system is not optimal which needs to be addressed and provide continuous future work.

Appendix

A. Time Investment

Total time for Sebastian Mark was close to 15+ hours for research and development.

I tried to work on this project every day for 2 hours. The total time for Sami Cemek was close to 14+ hours for coding, development, and debugging.

B. Shared responsibilities

Together:

- Brainstorming the project and java class structure
- Creating a UML diagram for our system
- Writing the research report

Sebastian Mark:

This process consisted of building the GUI, while adding and fixing features. Developing the window panels, creating the CSV file reader, stringbuilder, etc…. Also, experimenting with different layouts and functionality of the buttons. Additionally, editing certain sections of the code while adding detailed comments to some of the classes for readability.

Sami Cemek:

- Building the recommendation part
- Learning the basics of GUI and JFrame
- Creating the CSV file reader and writer
- Integrating my part to Sebastian's GUI part

Reference:

- Wondrium: Learning Java Programming
- YouTube: Bro Code, Learn Programming in Java - Lesson 18: GUI Programming with Java Swing,

- https://docs.oracle.com
- https://www.geeksforgeeks.org/creating-frames-using-swings-java/
- https://www.javatpoint.com/java-swing
- https://griddb.net/en/blog/building-a-recommendation-system-in-java/
- https://www.baeldung.com/java-collaborative-filtering-recommendations