FINAL PROJECT

JUSTIN LIU

MR. TWIET

AP COMPUTER SCIENCE A, PERIOD 3

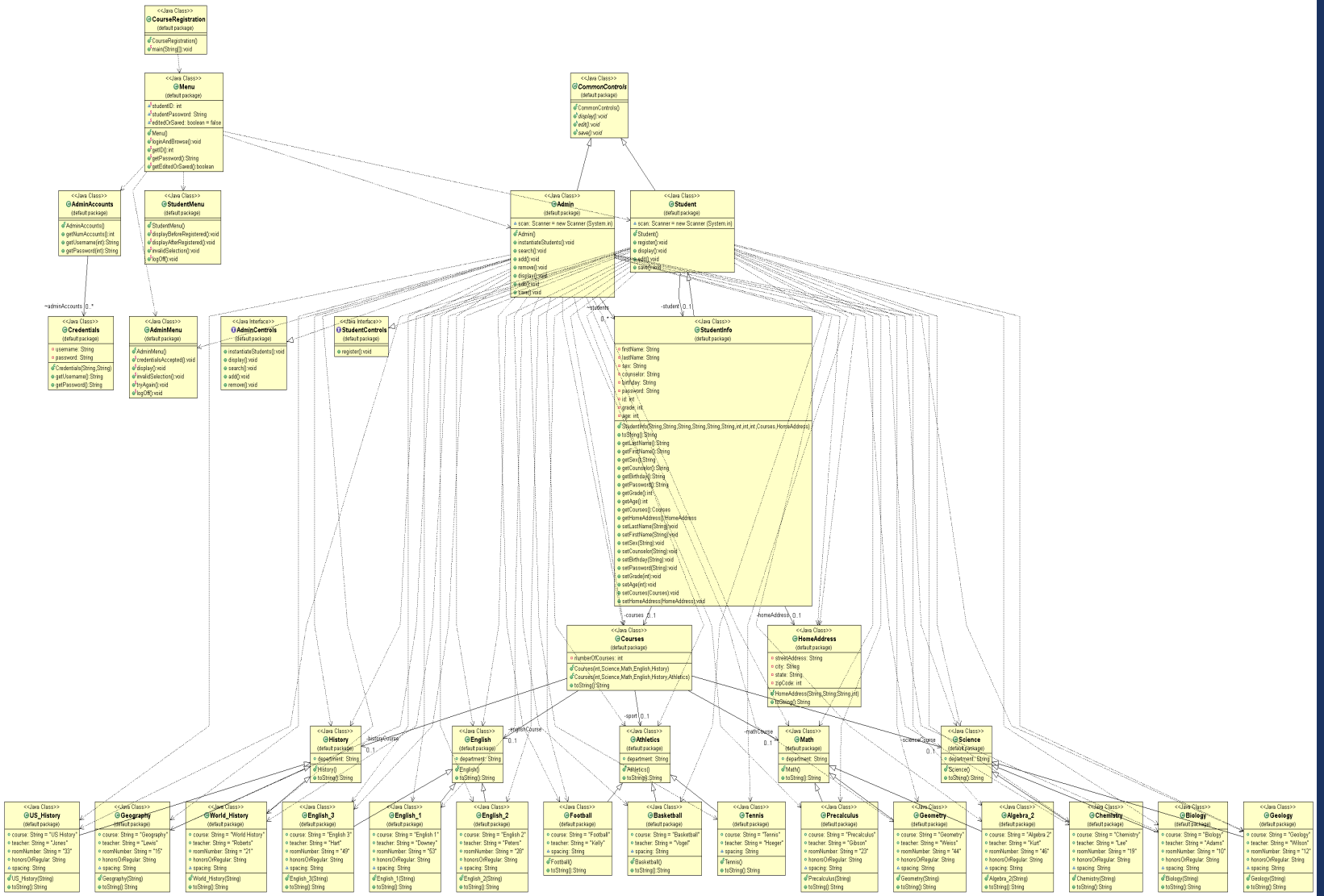
04/20/2020

PROGRAM OPERATION DESCRIPTION

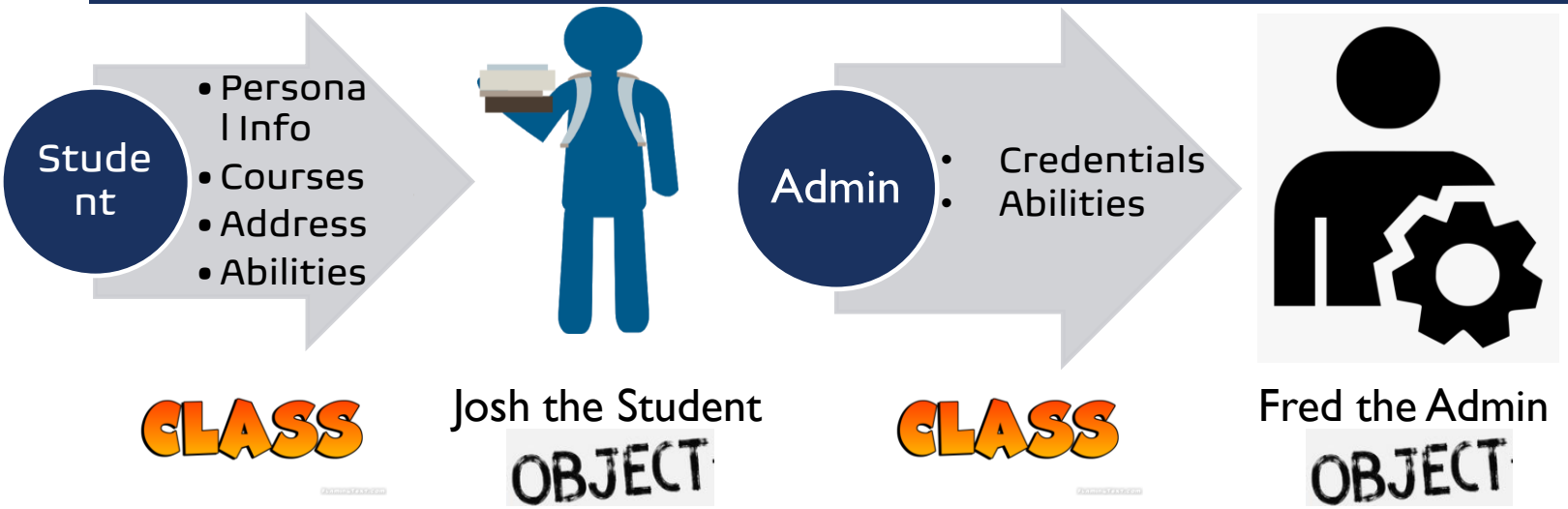
- This program aims to simulate a course registration system
- Based upon what a user is (admin/student), specific controls will be applicable
- Students can display account, register, edit info, and save info
- Admins can display accounts, search accounts, edit info, save info, and add/delete accounts
- UI operates by prompting user for a number that will perform a specific task (ex. If admin selects option 1, accounts will be displayed. If admin selects option 2, they can search for an account)
- Aimed to be realistic, i.e., similar demographic/information layout, UI, menus, user capabilities etc.

PROGRAM DEMONSTRATION

UML CLASS DIAGRAM OVERVIEW (ECLIPSE)

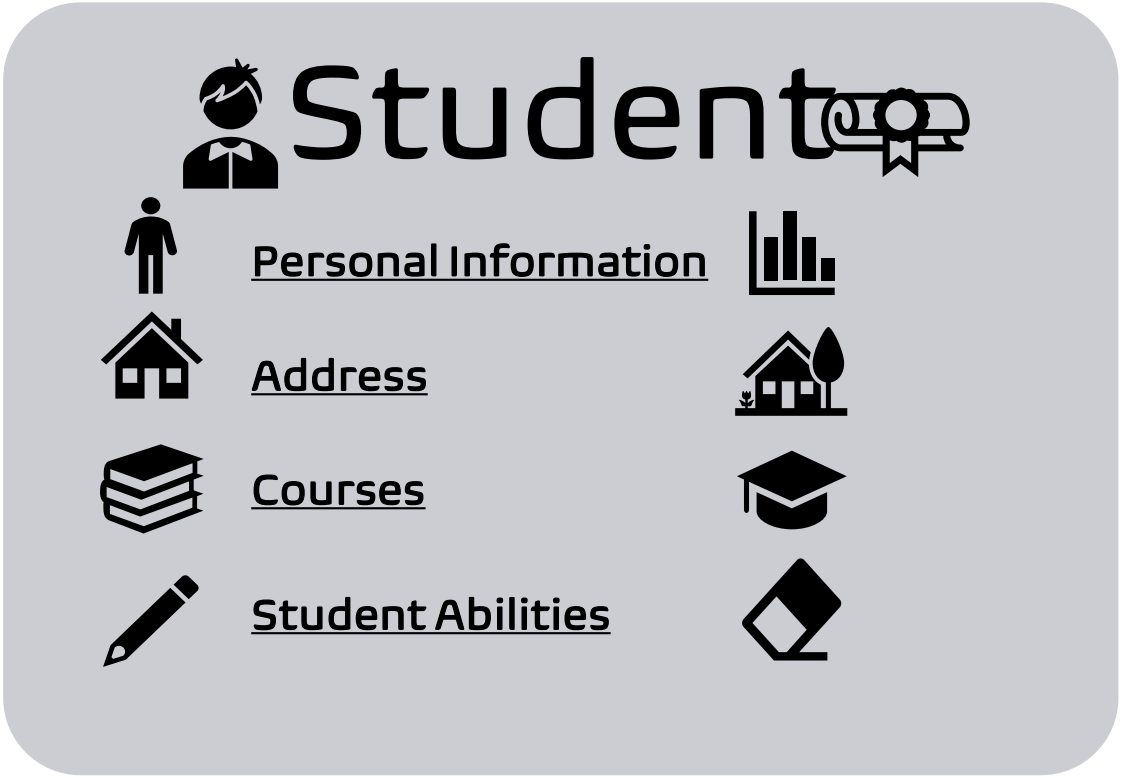


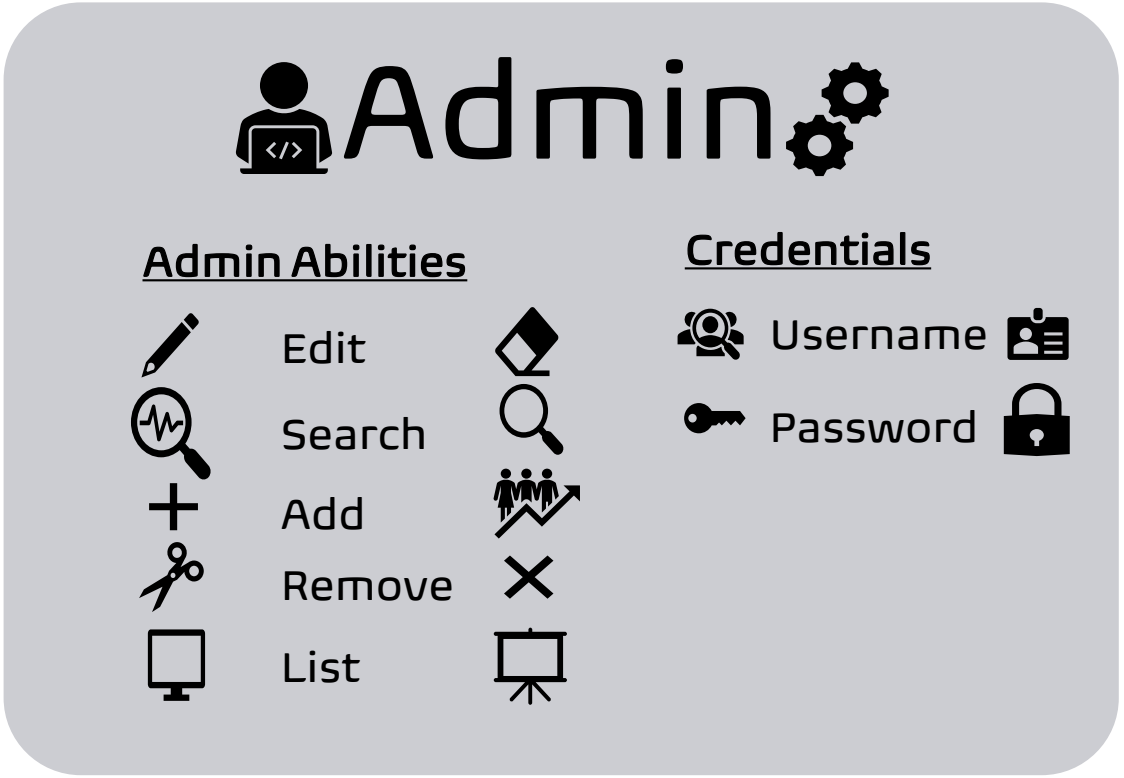
USE OF CLASSES/OBJECTS



- I utilized classes as instructions/user-defined blueprint from which objects are created to represent the set of properties or methods that are common to all objects of one type. Example above with student and admin.
- I utilized objects to represent real life entities, to correspond to things found in the real world, and had them interact by invoking methods. Example above with student and admin.
- I also tried logically and methodically employing classes to keep code organized.
- Instances of classes were instantiated to resemble an actual entity or property of something.
- Creating a system of classes and objects also enabled me to have a communicating, object-oriented project. Calling miscellaneous methods from each other, functioning distinctly, having relationships, and formulating coherent, logical dependencies all tie in

HOW CLASSES REPRESENT PHYSICAL OBJECTS



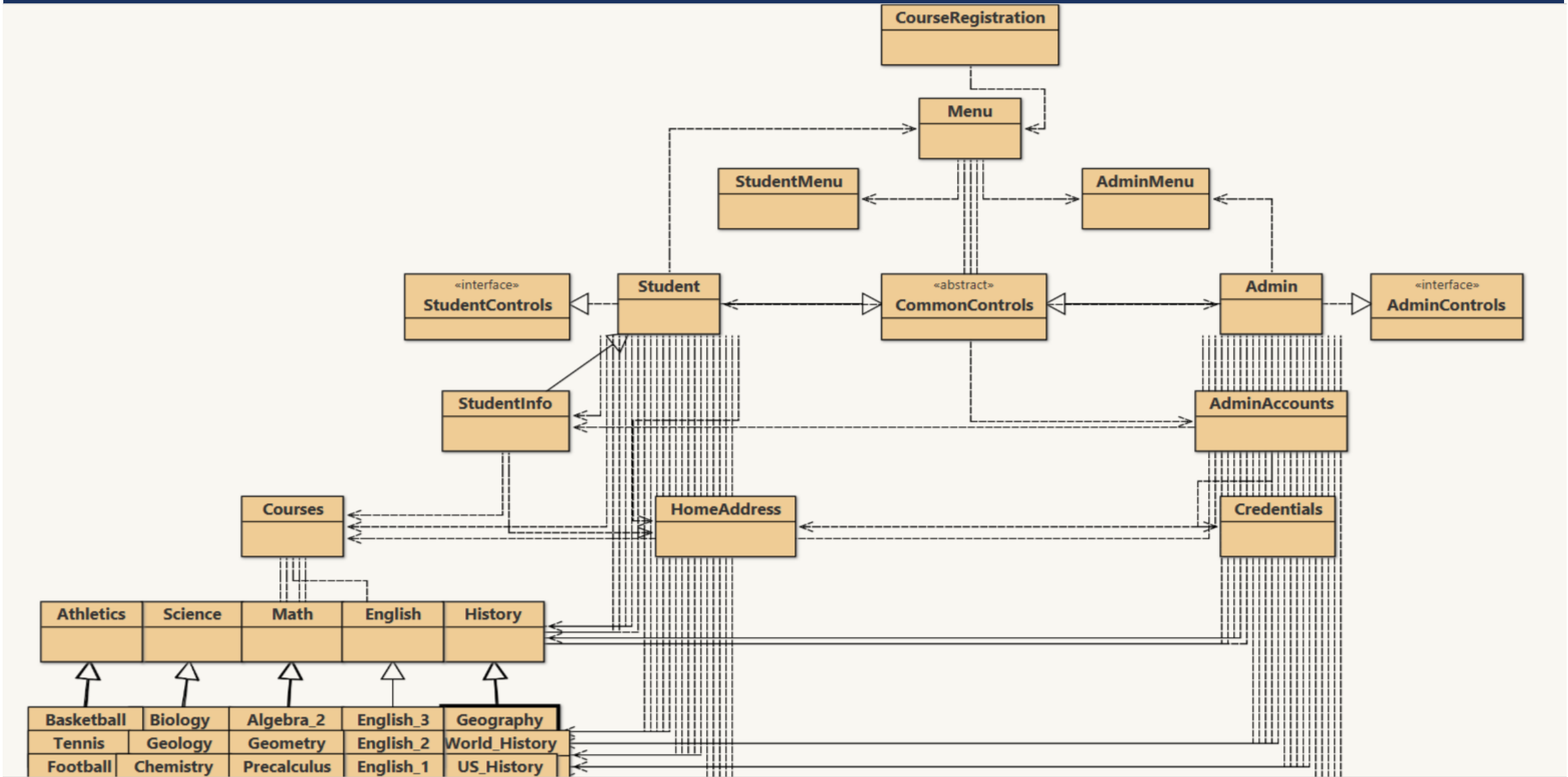


- Physical objects are represented by attributes (variables), abilities (methods), and other characteristics of classes

What's in a Name?

NAMES OF CLASSES, DATA MEMBERS/VARIABLES, AND METHODS

CLASS DIAGRAM



CLASSES

Classes were named in accordance with a course registration system.

Ties into creating a sensible hierachy and inheritance.

Named classes with simple, straightforward, concise, and descriptive nouns.

First letters of each word in a class's name are conventionally capitalized for readability and organization.

Named to represent something within the system or that played a role.

When interacting, extending, functioning, and communicating with one another, class names had to be justifiable. Instigating a coherent system with hierarchical orientation also demanded proper class naming.

VARIABLES/DATA MEMBERS

```
// Returns and organizes an arbitrary student's info as a String
public String toString()
{
    String info = "";

    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += "Last Name"   + "\t"   + "First Name"   + "\t"   + "Sex"    + "\t\t" + "Grade"    + "\t\t" + "Birthday"    + "\t\t" + "Age"    +
    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += this.lastName + "\t\t" + this.firstName + "\t\t" + this.sex + "\t\t" + this.grade + "\t\t" + this.birthday + "\t\t" + this.age +

    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += "Student ID" + "\t"   + "Password"    + "\n";
    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += this.id      + "\t\t" + this.password + "\n\n";

    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += "Residence and Mailing Address" + "\n";
    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += homeAddress                     + "\n\n";

    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += "Courses"    + "\n";
    info += "-------------------------------------------------------------------------------------------------------------------------------------------
    info += this.courses;

    return info;
}
```

VARIABLES/DATA MEMBERS (CONTINUED)

```
// Personal information below must all be made private - only student should know/access
private String firstName, lastName, sex, counselor, birthday, password;
private int id, grade, age;
private Courses courses;
private HomeAddress homeAddress;

public StudentInfo(String lastName, String firstName, String sex, String counselor, String birthday, String password, int id, int grade, int age, Cou
{
    this.lastName = lastName;
    this.firstName = firstName;
    this.sex = sex;
    this.counselor = counselor;
    this.birthday = birthday;
    this.password = password;
    this.id = id;
    this.grade = grade;
    this.age = age;
    this.courses = courses;
    this.homeAddress = homeAddress;
}
```

VARIABLES/DATA MEMBERS (CONTINUED)

- Most variables were named in accordance with a course registration system's terms and were properties/attributes of classes such as name, grade, id, age, etc.
- Variables were named to represent something within the system or that played a role.
- Variables were made to be meaningful and concise. That way when they are used, the programmer can quickly use a variable while still discerning what it represents.
- First letters are conventionally kept lower case and the first letters after that of each word are capitalized (camelCase) for readability and organization.
- Variables were made to be consistent throughout the entire program. Structure and composition are examples that require consistency. Similar variables with similar roles were identified differently, however, to avoid confusion.
- Some variables were created to debug and counteract errors within the program, but I still tried to keep those consistent with the program

METHODS

Abstracts/Polymorphism

```
abstract void display();

abstract void edit();

abstract void save() throws IOException;
```

Student Methods

```
public void register();
```

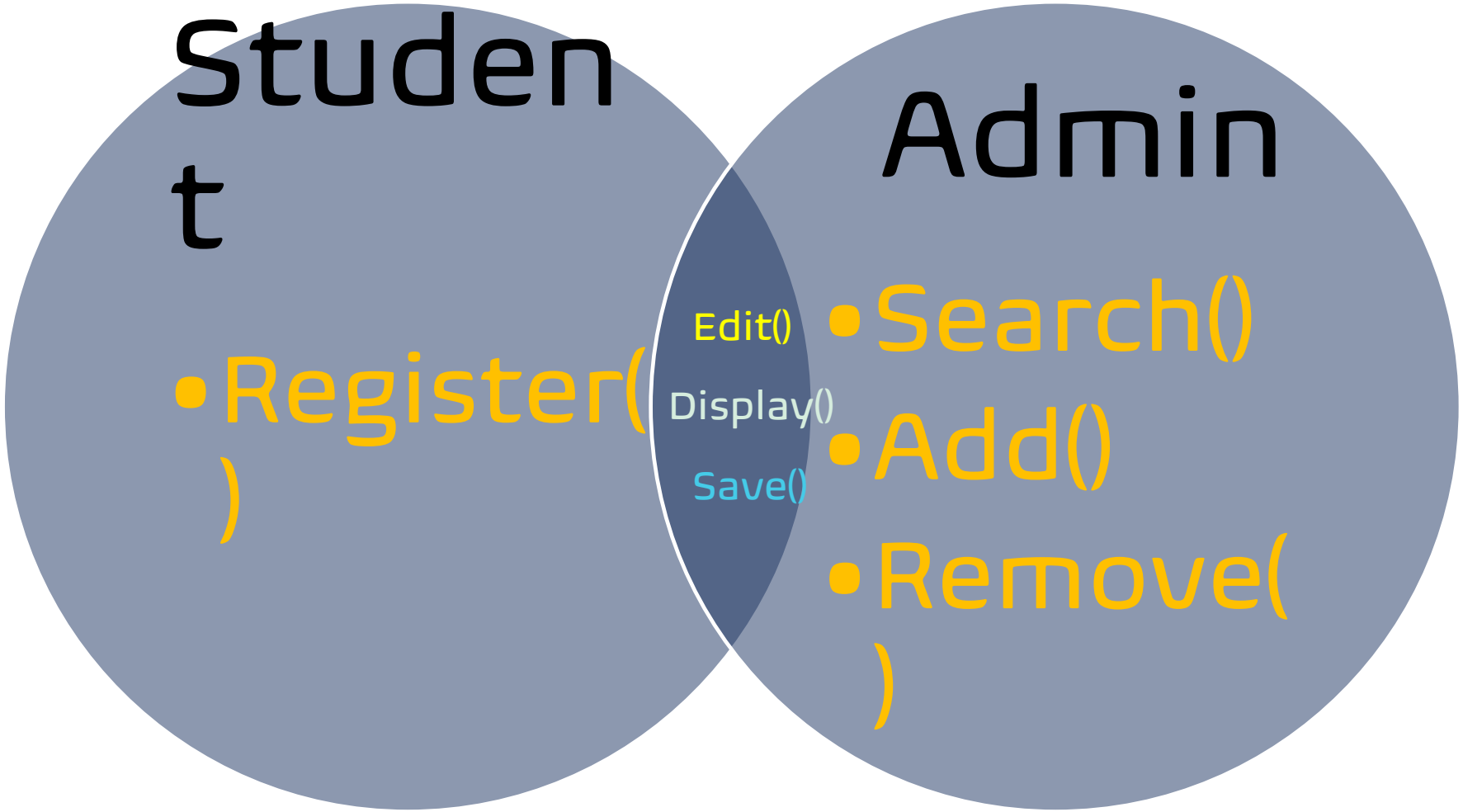
Admin Methods

```
public void instantiateStudents();

public void search();

public void add();

public void remove();
```

METHODS (CONTINUED)

- Methods corresponded with a course registration system.
- All my method names were **verbs/actions** that performed specific tasks within a course registration system
- Methods' first letters are conventionally kept lower case and the first letters after that of each word are capitalized (camelcase) for readability and organization.
- Some methods were created to debug and counteract errors within the program, but I still tried to keep those ones coherent as well.

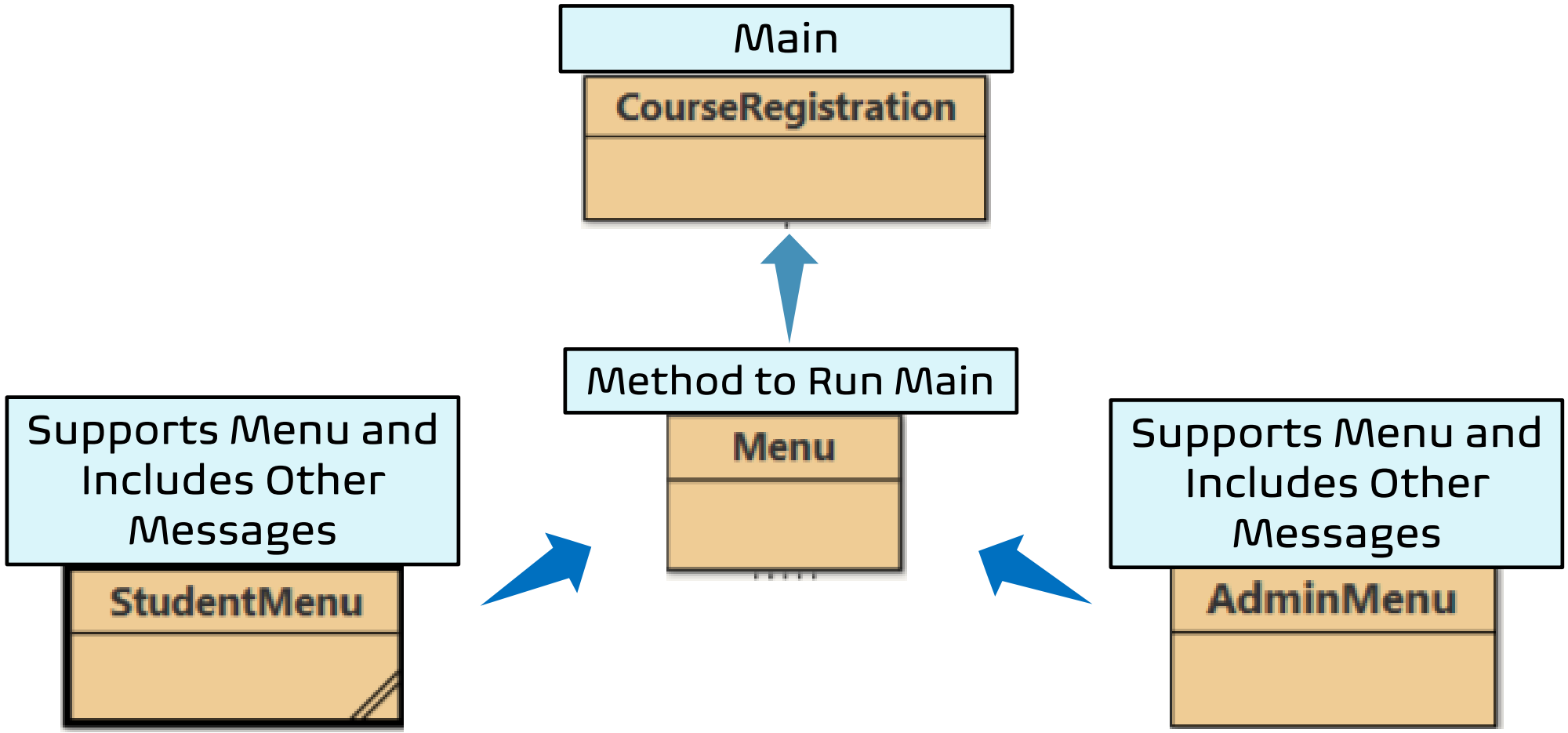
METHODS (CONTINUED)



CLASS INTERACTION

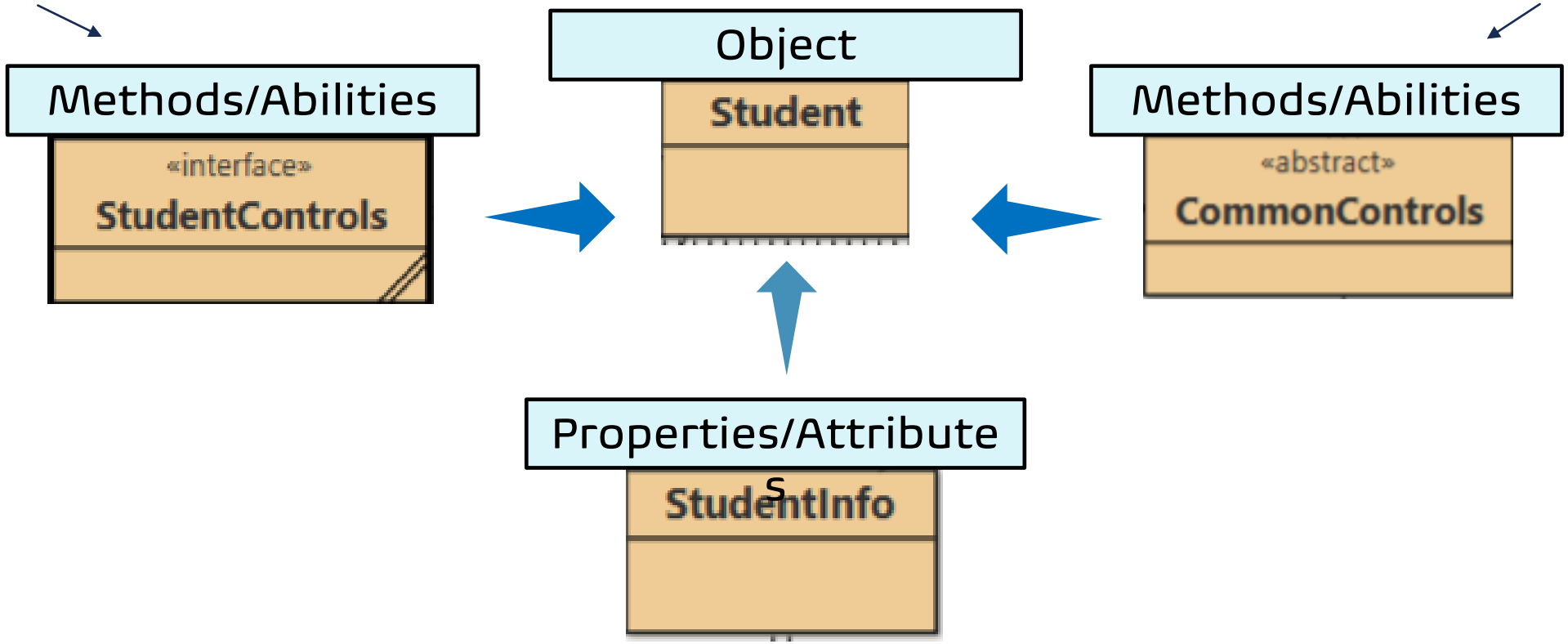
- Classes depend on each other to make a functional system (inheritance, superclasses, abstracts, interfaces, etc.)
- Instances of classes are constantly created based on user input. An example is if a user chooses to take "Biology Honors", a Biology Honors Course will be forced to instantiate. This applies for any other course. When registering for student information as well, the home address will instantiate based on the user input.
- Classes will constantly be invoking other class methods or retrieving variables. This enables efficiency, effectiveness, and organization.
- Having several classes can cause some complications, but for the most part it

MAIN AND MENUS

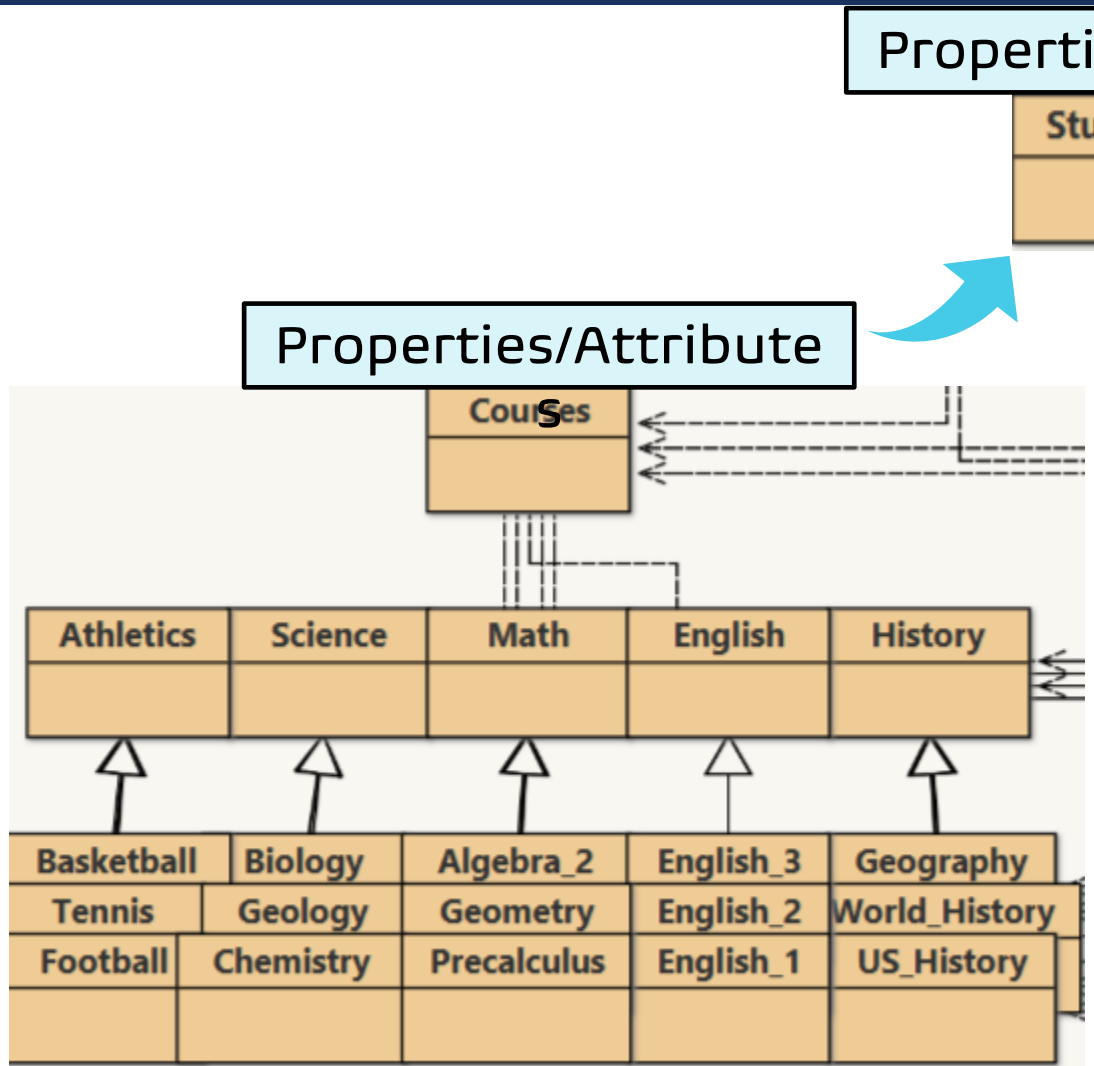


STUDENT

- Private Methods
- Shared with Admin

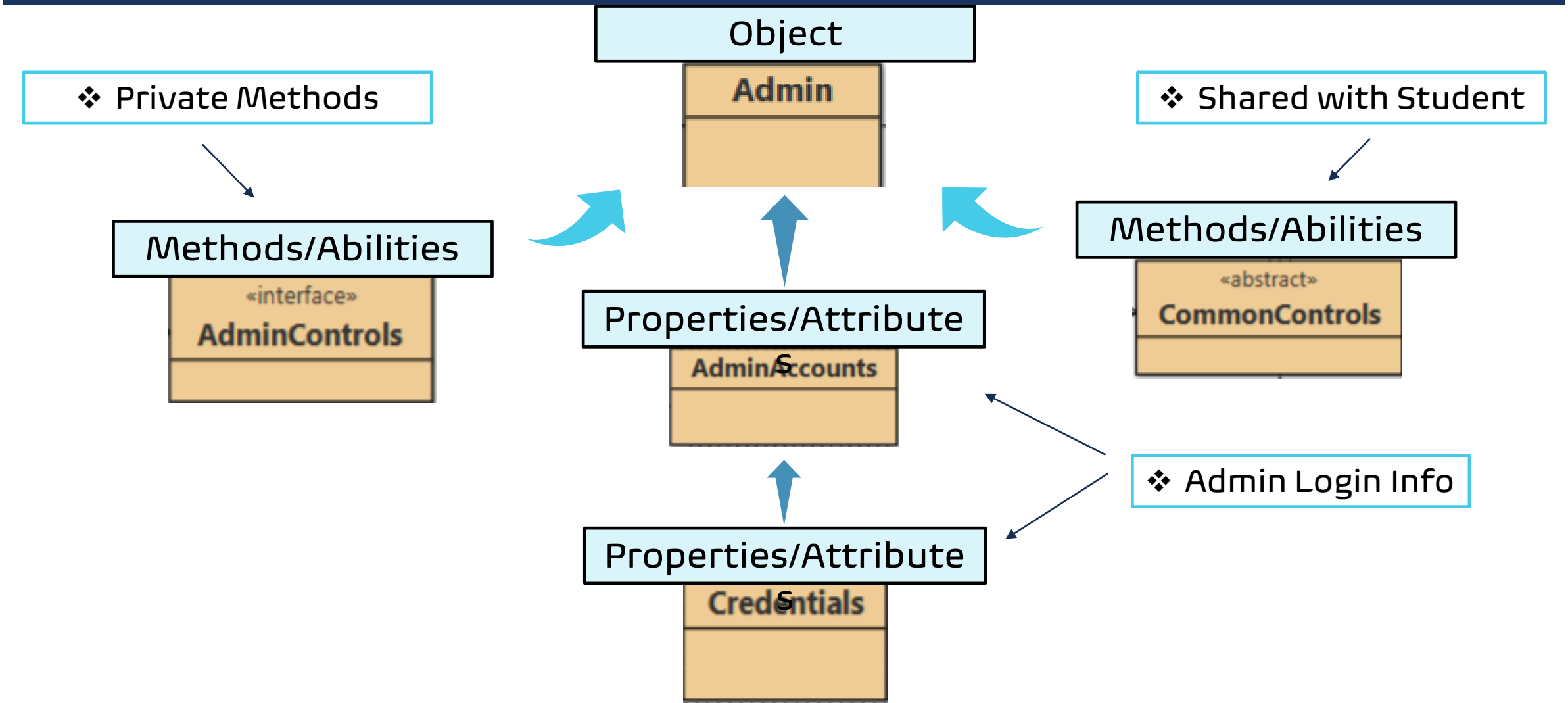


STUDENT (CONTINUED)

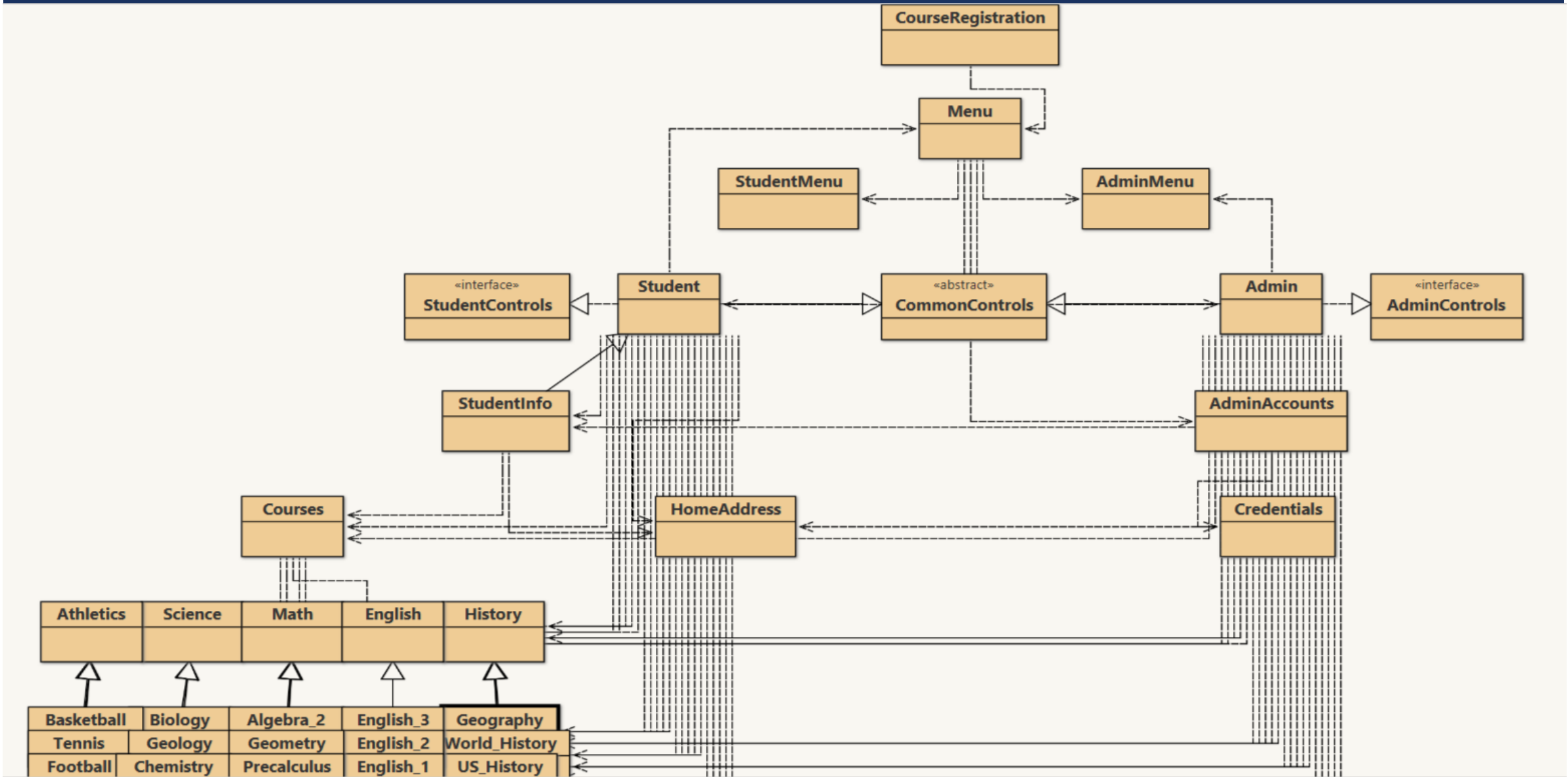


- Other student information includes name, age, grade, id, etc.

ADMIN


Object
Admin
❖ Private Methods
Methods/Abilities
«interface»
AdminControls
❖ Shared with Student
Methods/Abilities
«abstract»
CommonControls
Properties/Attribute
s
AdminAccounts
❖ Admin Login Info
Properties/Attribute
s
Credentials

INHERITANCE DIAGRAM OVERVIEW (BLUEJ)



INHERITANCE HIERARCHY (CONTINUED)

- Inheritance creates a hierarchy in which classes are dependent of each other and communicate
- Hierarchy starts from the top (Main Method) and gets more specific each level down
- Classes are built upon existing classes
- Supports the reusability of code and makes the program more efficient and concise
- Some classes (Student and Admin) inherit properties/methods from both abstract and interfaces. Abstract methods are common/shared between classes while interfaces are privately implemented
- Each class serves a purpose within the system

INTERFACES

- Student Controls
- Admin Controls

INTERFACES

- Specifies a set of methods that a class must implement
- Specifies the capabilities and serves as the blueprint of the class

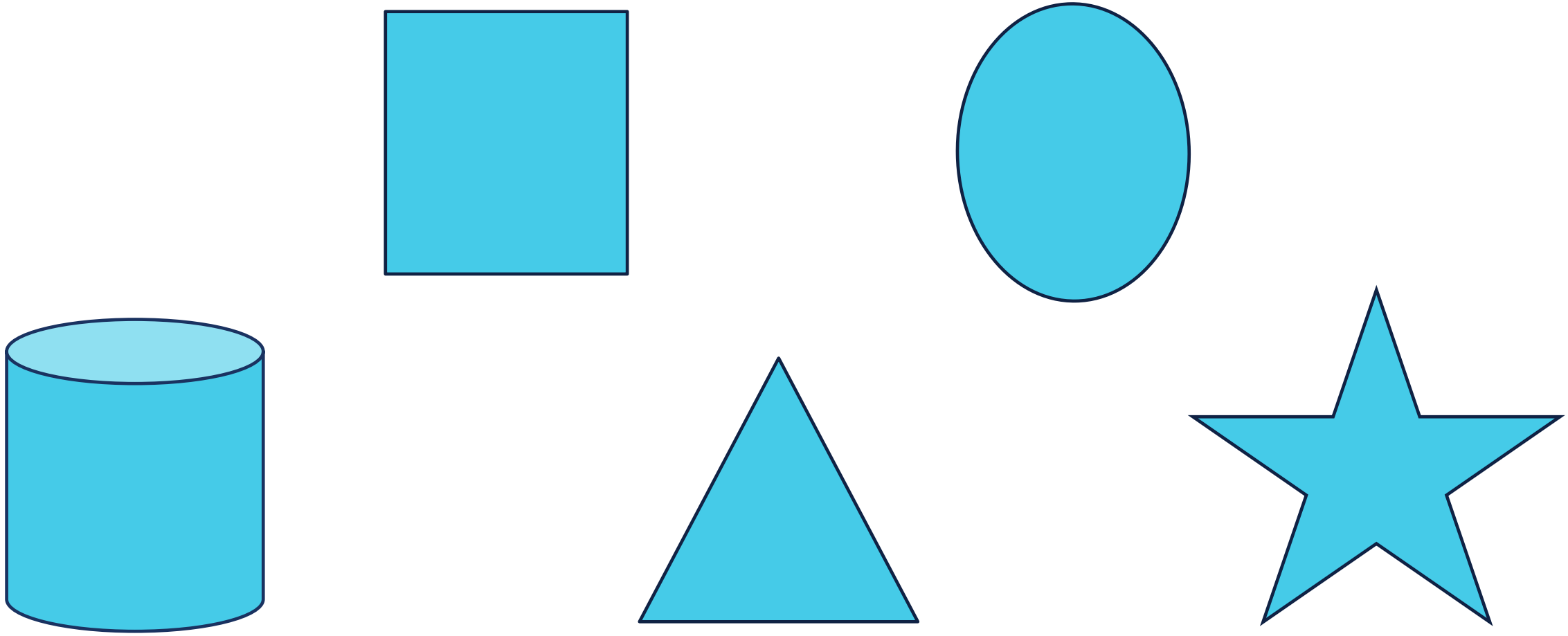
Student Interface

```
interface StudentControls
{
    //-------------------------------------------------
    // Represents the private/exclusive student ability
    //-------------------------------------------------

    void register();
}
```

Admin Interface

```
public interface AdminControls
{
    //-------------------------------------------------
    // Represents the private/exclusive admin abilities
    //-------------------------------------------------

    public void instantiateStudents();

    public void search();

    public void add();

    public void remove();
}
```

POLYMORPHISM



POLYMORPHISM

```
/*
Different people have slightly different displaying functions
ex. Admin and student:
    - Admins can oversee and display all student accounts
    - Students can only display their own account and info
*/
abstract void display();
```

```
/*
Different people have slightly different editing functions
ex. Admin and student:
    - Admins has to choose which account to edit and when prompted to edit something, the wording is more generic and official
    - Students simply edits personal info, courses, or address, and the wording is more personal and confidential
*/
abstract void edit();
```

```
/*
Different people have slightly different saving functions
ex. Admin and student:
    - Admins can oversee and save all student accounts
    - Students can only save their own account and info
*/
abstract void save() throws IOException;
```

POLYMORPHISM (CONTINUED)

Save() - Admin

```java
// There has to be at least one student in the ArrayList for information to be saved
if (students.size() != 0)
{
    for (int i = 0; i < students.size(); i++)
    {
        // Need to add one to the number since an ArrayList starts at 0, not 1
        int accountNumber = i + 1;
        // Adds the account number to the StringBuilder
        sb.append("\nAccount Number: " + (accountNumber) + "\n");
        // Adds the student's information to the StringBuilder
        sb.append(students.get(i));
        // Adds a line for distinction, visual appeal, and organization between accounts to the StringBuilder
        sb.append("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    }
}

else
{
    System.out.println("\nNo accounts available.");
}
```

POLYMORPHISM (CONTINUED)

Save() - Student

```
// Adds the student's information to the StringBuilder
sb.append(student + "\n");
```

USE OF POLYMORPHISM

- In the world of OOP and objects, every single object that has more than one shape or can be assigned to more than one class is called a polymorphic object
- Polymorphism is a feature that allows a single action to be performed in several distinct ways
- Makes code more versatile
- Enables generic frameworks to be built that take are all implemented uniquely
- Can be utilized in simplifying tasks, developing a more coherent, systemized

SPECIAL FEATURES OF MY PROGRAM

- Researched the Java IO API/Library which reads data and writes it to a file path provided by the user.

- Let user choose if they wanted to take an honors or regular cours

- Let user choose if they wanted to take 4 or 5 courses. If they chose 5, an overloaded method would be employed to let the student take s course.

- Structured output of a student's information to be formatted in accordance with official standards/la

- Implemented a mini login system for the admin. Only certain credentials would accepted.

CITATION OF SECOND-PARTY

- For the saving methods of both student and admin, I first drew inspiration of the code from our school's FRC team. We used similar code to collect, organize, and export data to a CSV (Comma-Separated Values) File on Excel. I then modified the code, cut it down, and designed it to accommodate the needs of my program.

```java
// Method to save students' account information to user's desktop
public void save() throws IOException
{
    // Instantiate a string builder, which is like a String object, but can be modified/mutated
    StringBuilder sb = new StringBuilder();

    // File path that the user will provide
    String filePath;

    // There has to be at least one student in the ArrayList for information to be saved
    if (students.size() != 0)
    {
        for (int i = 0; i < students.size(); i++)
        {
            // Need to add one to the number since an ArrayList starts at 0, not 1
            int accountNumber = i + 1;
            // Adds the account number to the StringBuilder
            sb.append("\nAccount Number: " + (accountNumber) + "\n");
            // Adds the student's information to the StringBuilder
            sb.append(students.get(i));
            // Adds a line for distinction, visual appeal, and organization between accounts to the StringBuilder
            sb.append("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        }
    }
```

CITATION OF SECOND-PARTY (CONTINUED)

```
else
{
    System.out.println("\nNo accounts available.");
}

// ********************************************************************************
// Some lines require an extra scan.nextLine() or parseInt() when switching from int to String
// ********************************************************************************

System.out.println("\nEnter a file path in which you would like to store your information:");
filePath = scan.nextLine();

/* If my project/program is opened from a flashdrive, the file path provided is nothing/null, and the user just hits enter,
the file/info will still be saved to the flashdrive. This only applies if the project/program is opened from a flashdrive, however.
If the project/program is opened from another location like a personal folder on a user's computer and not a flashdrive,
the file/info will fail to save. I don't know why this is so, as it just seems to occur for flashdrives. */

for (int i = 0; i < filePath.length(); i++)
{
    // Need to add an extra "\" for each "\" in the provided file path, since the "\" requires an escape sequence in Java
    char letter = filePath.charAt(i);
    if (letter == '\\')
    {
        filePath = filePath.replace("\\", "\\\\");
        break;
```

CITATION OF SECOND-PARTY (CONTINUED)

```
        }
    }

    // Add this to the end of the file path to give the file a name
    filePath += "\\Students Information.txt";

    // Print/export data and write to the provided file path after fully updated
    try
    (
    // FileWriter is utilized to write character-oriented streams of data to a file.
    FileWriter fw = new FileWriter(filePath);
    /* BufferedWriter writes text to character output stream, buffering characters
    so as to provide for the efficient writing of single characters, arrays, and strings. */
    BufferedWriter bw = new BufferedWriter(fw);
    // PrintWriter enables formatted data, characters, and representations of objects to be printed/written to a text-output stream.
    PrintWriter pw = new PrintWriter(bw)
    )
    {
        // Converts the data to more readable Strings or letters
        pw.print(sb.toString());

        // If the file path saving does indeed succeed, display a valid message
        System.out.println("\nSaved.");
    }
    catch (IOException e)
    {
        // In the case that the file path was invalid or there was an error with saving the information, display an invalid message
        System.out.println("\nFailed to save due to invalid file path.");
    }
}
```

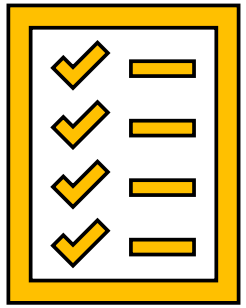
KNOWN BUGS

- Don’t know of any bugs, as I tried my best to test every possible situation and accommodate for it.
- One minor aspect that worries me relates to the Scanner, however. The Scanner has issues when switching from int to String. An example is if the user last inputted an integer, then chooses another option that immediately takes in a String. It can sometimes be hard to control the Scanner as well, since the user is determining what they want to enter.
- I did fix the Scanner and haven't discovered a scenario yet where the program has issues. By using conditional statements and adding an extra "scan.nextLine()" if the previous selection was indeed a "scan.nextInt()". However, there is always the miniscule chance that the user can break the program or hit the wrong key.

Difficulty

Fun

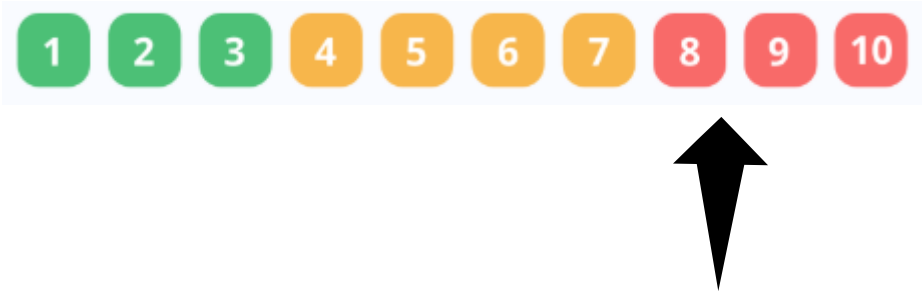
Evaluation



What I Learned

CONCLUSION

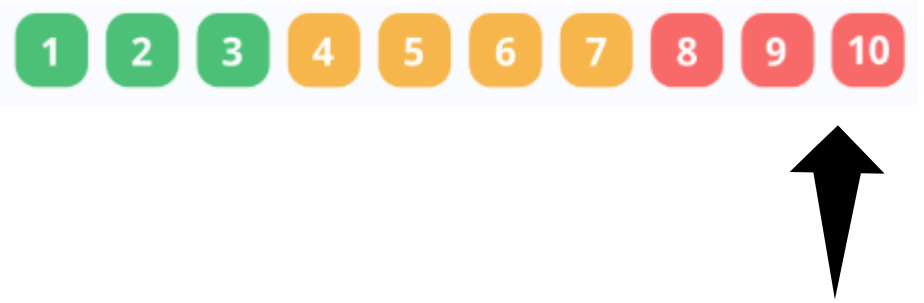
DIFFICULTY LEVEL



8.36

- Felt challenged and encountered several bugs, but I still know that there is room for improvement.
- Strived to create a less convoluted but more straightforward, coherent program that is tractable and user-friendly.

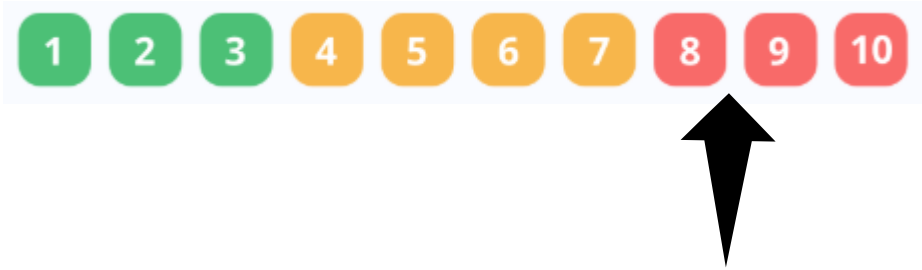
FUN LEVEL



9.78

- Enjoyed the entire process and feel like this was a great exposure to a real-life application in computer science.
- Programming and simulating such a system was also incredibly meaningful to me. Seeing and interacting with the final product that repeatedly refurbished for weeks was very satisfying.

EVALUATION



8.43

- Although I am truly proud of this project, I would love to have made it even better.
- Features I want to add are having the ability to log in after logging out while maintaining all the data, implementing a GUI, creating animations/graphics, offering more courses/sports, offering a zero period, letting the admin choose which accounts he wants to save instead of defaulting to all of them, making my code slightly more concise (under 4500 lines), and more.
- I think there could have been some more clever/effective ways in fixing my bugs.
- I like how the various outputs, organization of code, numerous methods/abilities, and inheritance hierarchy turned out.
- I believe that this project offered a remarkable opportunity to explore the true power of object-oriented programming. Learning to replicate a course registration system was also exceptionally meaningful and insightful, as I got to explore

WHAT I LEARNED

- Opened my eyes to programming real-life applications
- Became more competent of exercising the ability of building programs that simulate aspects of everyday life
- Helped me to better discern and construct object-oriented programming
- Assimilated the importance and how to implement interfaces, an inheritance hierarchy, and polymorphism.
- Gained a better understanding of structuring a class
- Helped me better understand control structures
- Learned to fabricate a coherent system of classes, make them communicate, and employ them to make code more concise/organized.

THANK YOU