CSC 142 FINAL PROJECT

Ryan Krumbholz

Table of Contents

Overview

UML Diagrams

Design

Testing

Program Overview

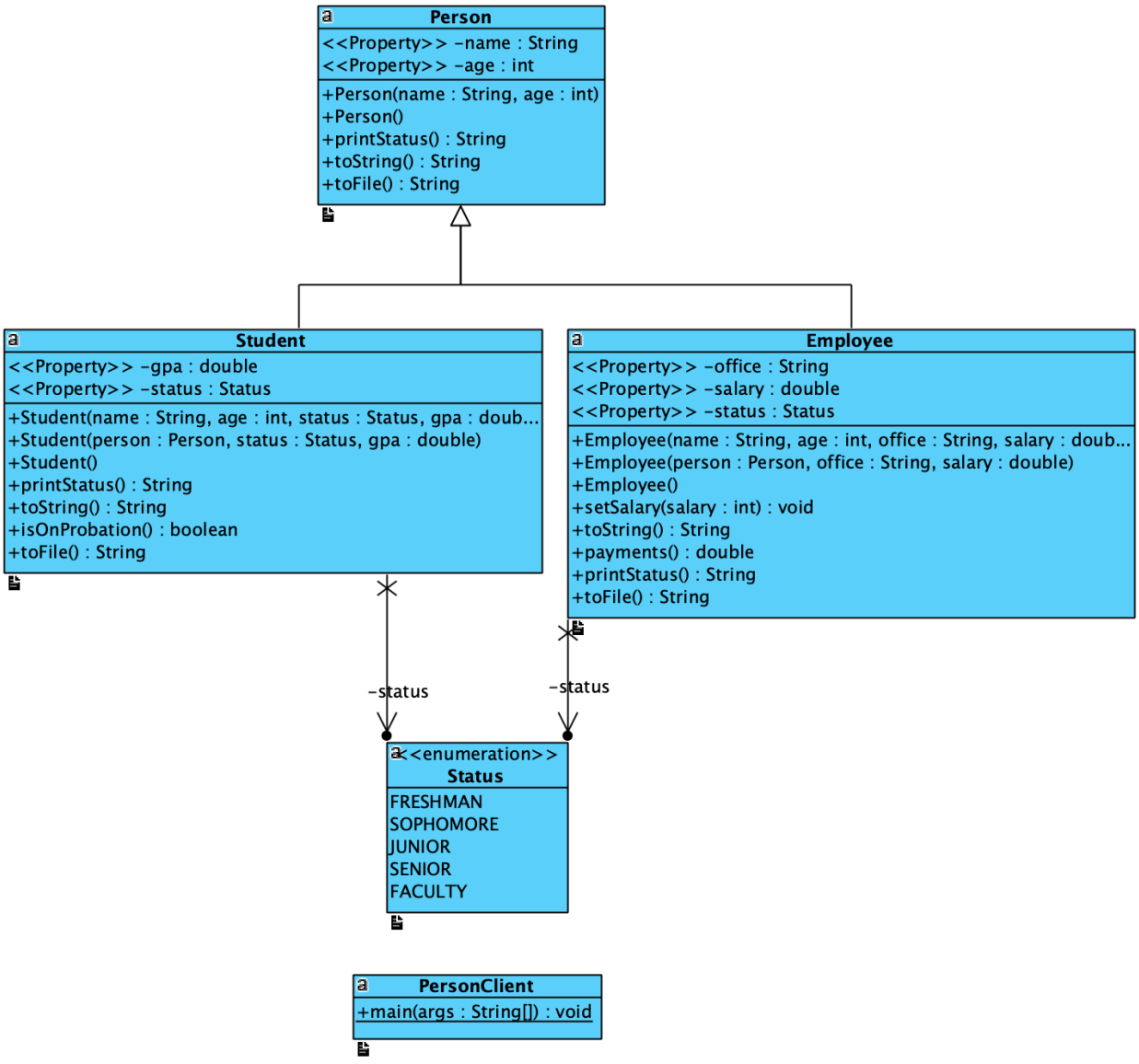
This program is a college database management tool that allows the user the ability to read and alter their database. In order for these functions to work, the database file must be end with the ".txt" extension and must be formatted as follows: role (Employee or Student), first and last Name, age, grade (students) or office (employees), and grade point average (students) or salary (employees). For example, a line in the file should look as such (excluding the period): Student, Ryan Krumbholz, 20, Sophomore, 3.9. The program will take in that line of text from the file and generates a person with the given characteristics. The program will repeat these steps until the file has been fully read.

Once each line in the text file has been read, the user can then perform several functions within the database. These functions being, the ability to search for someone in the database by name, delete someone from the database by name, and add someone to the database by providing all of the required information for that person. When performing these functions, the user may receive an error message in the console due to providing the incorrect name or a person's attributes incorrectly. When searching for a person, if the name provided by the user is not in the database, the function will return a logical value of false. When deleting a person, if the name provided by the user is not in the database, the console will print a message letting the user know that a person with that name is not in the database. When adding a person, the program will not run if you do not correctly provide it with a completed person or the person attributes in a correct order.

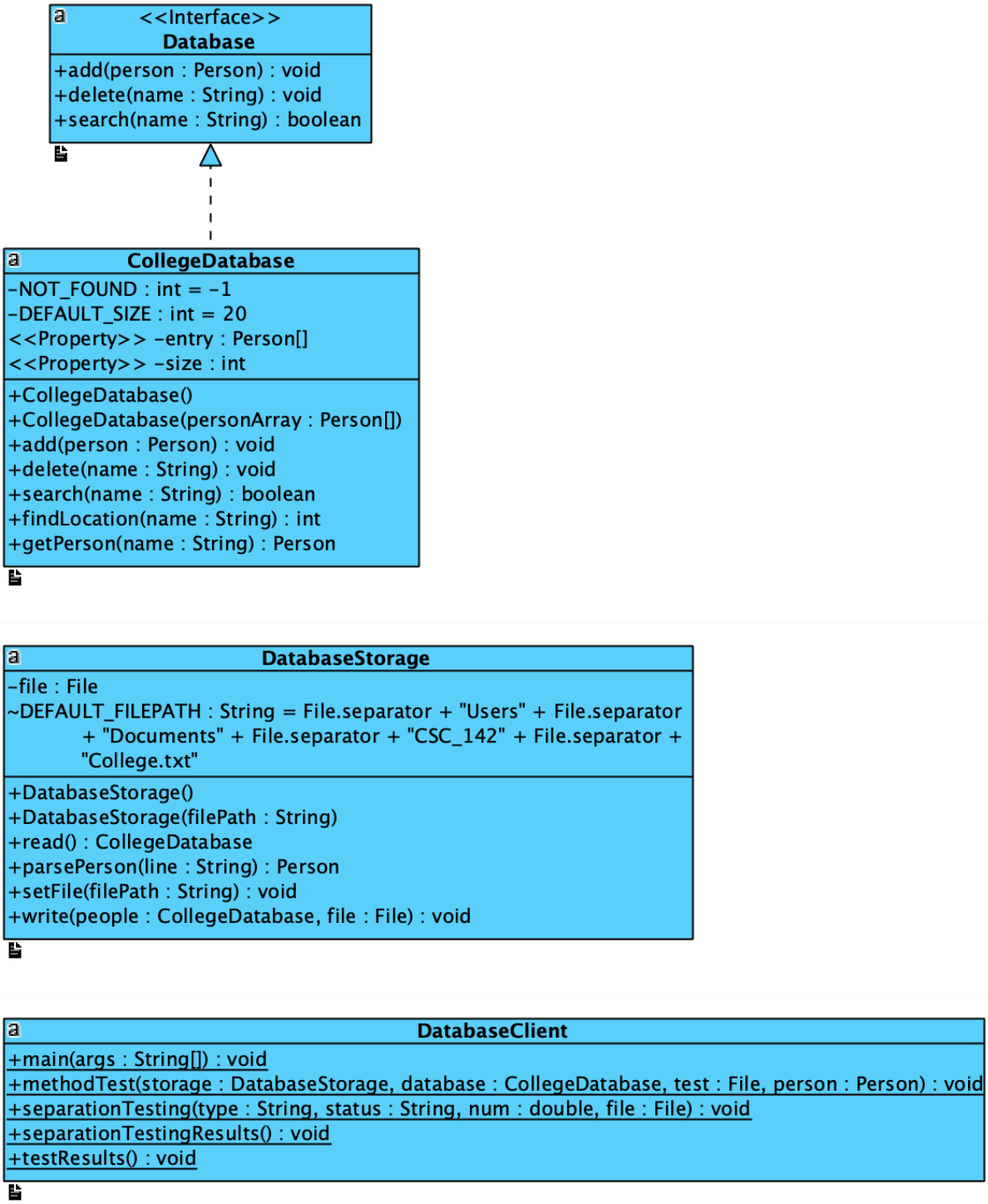
For more information please consult the following pages which display the UML Diagram (a visual representation of the program's design), explain the design and operation of each component within the program, and explain the testing methodology.

UML Diagrams

Package “people”



Package “database”



Design Overview

Note: If a number appears next to a word as a super-script the definition is in the reference section.

This program is separated into three different packages[1], people, database, and textfiles. Each of these packages handle a key component of the program. The package, people, defines what a person is, and the two types of people present in the database, students and employees.

The people package consists of four classes[2], Person, Employee, Student, and PersonClient, and an enumerator[3], Status. The Person class defines a Person object as something with a name and an age, and consist of methods[4] to alter those characteristics, retrieve those characteristics, and print those characteristics to either a file or the console. The Employee class inherits[5] those characteristics and methods from the Person class, while adding characteristics specific to an employee, office and salary. The Student class also inherits the characteristics and methods from the Person class, but instead adds a year (ex: SOPHOMORE) and grade point average. The PersonClient class allows the user ability to test the methods and functionality of all of the classes within the package. It only contains a main method, a method that executes the given code. The Status enumerator defines the five statuses which denote whether the person is a freshman, sophomore, junior, senior, or faculty.
For a more detailed description of each method within the package, please see section on this package.

The package, database, handles managing and maintaining the list of people in the college database. The database package consists of three classes, DatabaseStorage, CollegeDatabase, and DatabaseClient, as well as one text file, colleges.txt, and one interface[6], Database. The DatabaseStorage class manages reading and writing from the database file. It includes methods to read from a file, write to a file, parse[7] a file, and change the target file. The CollegeDatabase class manages the database in the form of a list. It includes methods allowing the user to search the list by name, add a person to the list, delete from the list by name, and find the location of a person in the database. The DatabaseClient class allows for the user to test the functionality and methods of any of the classes within the package, but also has a pre-built test that validates every function inside the classes within the package, which prints the results of the testing to the console and prints the results of the separation testing to files in the textfiles package. Please see the testing section for more information on the testing methodology used. The Database interface defines required methods for any class which implements[8] it. The required methods being the ability to add, delete, and search within the database. The CollegeDatabase class implements these requirements as

previously mentioned. For a more detailed description of each method within the package, please see section on this package.

The package, textfiles, includes all of the textfiles for the separation testing. These text files are written to whenever the DatabaseClient class is ran. Please see the testing section for more information on the contents of these file.

Package “people”

Class “Employee”:

Default Constructor Method – A parameter-less constructor method that doesn’t set any of the employee’s characteristics and prevents the program from crashing.

Other Constructor Method – Parameters: String name, int age, String office, and a double salary. Uses these parameters to set the characteristics of the employee to these values.

Other Constructor Method – Parameters: Person person, String office, double salary. Uses the name and age of the passed person object to initialize the name and age of the employee and uses given office and salary to initialize those characteristics.

Methods getOffice & setOffice – Allows for safe retrieval and alteration to the employee’s office. setOffice uses the given parameter to change the office of the employee.

Methods getSalary & setSalary – Allows for safe retrieval and alteration to the employee’s salary. setSalary uses the given parameter to change the salary for the employee.

Method toString – Returns the characteristics of the employee as a string for easy readability when printing to the console.

Methods getStatus & setStatus – Allows for safe retrieval and alteration to the status of the employee. setStatus uses the given parameter to set the status of the employee.

Method payments – Returns the salary amount split into 24 individual payments.

Method printStatus – Returns the string “Employee” to designate the employee’s role.

Method toFile – Returns the characteristics of the employee as a string in comma separated value formatting for easy printing to a file.

Class “Person”:

Default Constructor Method – A parameter-less constructor method that doesn’t set any of the person’s characteristics and prevents the program from crashing.

Other Constructor Method – Parameters: String name, int age. Uses these parameters to set the person’s name and age.

Other Constructor Method – Parameters: Person person. Uses the Person object’s name and age to set the name and age of a new person.

Methods getAge & setAge – Allows for the safe retrieval and alteration of the person’s age. setAge uses the parameter given to change the age of the person.

Methods getName & setName – Allows for the safe retrieval and alteration of the person’s name. setName uses the given parameter to change the age of the person.

Method printStatus – Returns the string “Person” to designate the employee’s role.

Method toString – Returns the characteristics of the person as a string for easy readability when printing to the console.

Method toFile – Returns the characteristics of the person as a string in comma separated value formatting for easy printing to a file.

Enumerator “Status”:

FRESHMAN – Denotes the status of a freshman in college.

SOPHOMORE – Denotes the status of a sophomore in college.

JUNIOR – Denotes the status of a junior in college.

SENIOR – Denotes the status of a senior in college.

FACULTY – Denotes the status of a faculty at the college.

Class “Student”:

Default Constructor Method – A parameter-less constructor method that doesn’t set any of the employee’s characteristics and prevents the program from crashing.

Other Constructor Method – Parameters: String name, int age, Status status, and double gpa. Uses these parameters to set the characteristics of the student to these values.

Other Constructor Method – Parameters: Person person, Status status, double gpa. Uses the name and age of the passed person object to initialize the name and age of the student and uses given office and salary to initialize those characteristics.

Methods getGpa & setGpa– Allows for safe retrieval and alteration to the student’s gpa. setGpa uses the given parameter to change the gpa for the student.

Method toString – Returns the characteristics of the student as a string for easy readability when printing to the console.

Methods getStatus & setStatus – Allows for safe retrieval and alteration to the status of the student. setStatus uses the given parameter to set the status of the employee.

Method isOnProbation – Returns a Boolean value based off of what the student’s gpa is. If the student’s gpa is less than 2.0, then the method will return true, indicating that the student is on probation. If the student’s gpa is not less that 2.0, the method will return false, indicating that the student is not on probation.

Method printStatus – Returns the string “Student” to designate the employee’s role.

Method toFile – Returns the characteristics of the student as a string in comma separated value formatting for easy printing to a file.

Class “PersonClient”:

Main Method – Executes the given code. Contains a try catch method to prevent the crashing of the program.

Package “database”

Class “CollegeDatabase”:

Default Constructor Method – Parameter-less constructor. Sets Person[] entry = to DEFAULT_SIZE (20) and sets the size to the length of entry.

Other Contsructor Method – Parameters: Perons[] personArray. Sets entry to personArray and sets size to the length of entry.

Method add – Parameters are: Person person. Adds person to the array by taking a temporary copy of the entry array, expanding the entry array, then iterating through a for loop to copy each element back into the array, and then adds the inputted person object at the end of the array.

Method delete – Parameters: String name. Removes a Person object from the Array by utilizing name within the helper method search. If a matching name is found, the Person object associated with that name will be removed and the array will be shifted over and shrunk. If name is not found however, an error message will print to console.

Method search – Parameters: String name. Searches for a person in entry with a name that is equal to the parameter name. Does this by linearly comparing each person’s name in the array to the name parameter using a for loop. If a person’s name is equal the parameter name, the loop will break and return true. If a person is not found with the same name as the parameter name, then the method will return false.

Method findLocation – Parameters: String name. Uses the search method as a helper method, passing it the parameter name. If the name is in the database, a while loop will execute until a person with a matching name is found. While the while loop is iterating, int i, which is initialized to zero outside of the loop is incremented by one. When the name is found, the loop will break, and return i, denoting where the person is in the array. If a person with a matching name is not found however, the method will return NOT_FOUND (-1), denoting that a person with that name is not in the database.

Method getEntry – Allows for the safe retrieval of entry.

Method getSize – Allows for the safe retrieval of size.

Method getPerson – Parameters are: String name. Allows for the safe retrieval of a Person object from entry. Uses findLocation as a helper method.

Interface "Database":

Method add – Parameters: Person person. Any class which implements Database must have this method.

Method delete – Parameters: String name. Any class which implements Database must have this method.

Method search – Parameters: String name. Any class which implements Database must have this method.

Class "DatabaseStorage":

Default Constructor Method – Parameter-less constructor method. Calls setFile and passes the variable DEFAULT_FILEPATH.

Other Constructor Method – Parameters: String filepath. Calls setFile and passes the filepath.

Method read – Reads the file by utilizing a while loop and converts it into a string array where each line of the file is a new index in the array. After doing this, if uses parsePeople to turn each line into a Person object and then adds it to the CollegeDatabase.

Method parsePerson – Parameters: String line. Splits given line into an array, using a comma as the separating value. Then, because each index in the array will always have the same type of value (ex: name or age), the method creates a person with the name and age from the array. Then decides whether the person in the line is either an employee or student and then assigns the remaining characteristics for the person's respective role.

Method setFile – Parameters: String filePath. Sets file to a new file with the path of filePath.

Method write – Parameters: CollegeDatabase people, File file. Uses a for loop to write each person in the database to the file by calling their toFile method.

Class “DatabaseClient”:

Main Method – Executes the code within the method.

Method methodTest – Parameters: DatabaseStorage storage, CollegeDatabase database, File test, Person person.Test all the methods within the DatabaseStorage and CollegeDatabase classes utilizing given parameters. Specifically test writing to the test file, searching for the given person, deleting the given person, and adding the given person into the database. After running each test, the program will print whether the test passed or failed to console. See Testing section for more details.

Method separationTesting – Parameters: String type, String status, double num, File file. Separates a CollegeDatabase upon given characteristics such as Status, age, role, salary, and gpa, and then writes to a separate file based on separation type. See Testing section for more details.

Method separationTestingResults – Conducts the five given separation tests using the separationTesting Method.

Method testResults – Conducts the methodTest, separationTesting, and other given test all in one method and prints results to the console. See Testing section for more details.

Testing

The test conducted in the DatabaseClient are intended to validate the functionality of the program and all of the methods within the classes in the database package. There are three types of test, method testing, separation testing, and other test required by the assignment.

Method testing test all of the methods within the CollegeDatabase and DatabaseStorage classes by calling methods that rely on helper methods. The methods tested in the method test are: write(), search(), delete(), and add(). For the write test, a CollegeDatabase and test file are passed to the method. To determine whether the test passed or failed, you will need to check the given test file. If the contents of the CollegeDatabase appear in the test file, the test has passed. If the contents of the CollegeDatabase are not present, this is a sign that the test has failed. The next method tested in the method test is search(). If the search method returns the boolean value of true, a message indicating that the test has passed will print to the console. If the search method returns false, a message indicating the test has failed will print to the console. The next method tested is delete(). The delete function uses findLocation() as a helper method. To validate that the person has been successfully deleted from the database, the program checks whether or not it can find the name in the database. If it cannot find the name in the database and the variable index, which is assigned to the variable location before the test are executed (if the person doesn't exist index will equal -1), is greater than or equal to zero a message indicating that the test passed will print to the console. If the person still does exist in the database or index equals -1, a message indicating the test has failed will print to the console. The method in the method test is add(). To validate whether or not the person has been added, the program will check whether or not the person is present in the database via the search method. If the person is found, a message indicating that the test has passed will print to the console. If the person is not found, a message indicating that the test has failed will print to the console.

Separation testing test the program's ability to separate the database into different text files based off of the criteria provided in the final project write-up. The program separates people into text files by providing the separationTest method with the parameters required in the testing criteria. For example, if the test calls to print all of the students in the database over the age of 45, you'd indicated to the method what these criteria are. Calling the method for this scenario would like: separationTesting("age" (what we are looking for), "Student" (who we are looking for), 45 (numerical value associated with what we are looking for), filename(file name)). To determine whether or not the test passed you will need to check the file that you specified in the method call.

The other tests ran were requested by the final project write-up. Those tests being, attempt to retrieve a negative index of the database, attempt to retrieve the zero index of the database, and conduct a method test for someone in the front, middle, end, and not in the database. For the negative index test, to prevent the program from crashing and in order to see the error message I used the try-catch error handling technique. This causes the program to print the exception to the console and then perform the next test. For the zero index test, the program retrieves the first index in the array, as arrays start at zero. The next three test all print to console indicating that they have passed. The test for someone not in the database will fail all of the test except for the add test, as that person is not in the database.

Reference

Package[1] – A collection of files.

Class[2] – A Template that defines the behavior of an object.

Enumerator[3] – An ordered listing of all items in a collection.

Method[4] – A procedural set of code.

Inherits[5] – Receives the same attributes and methods.

Interface[6] – A set of required methods.

Parse[7] – Analyzing a sequence of text or symbols and making sense of it.

Implement[8] – Utilizing an interface in a class.