David Wu
GTID:
dwu45

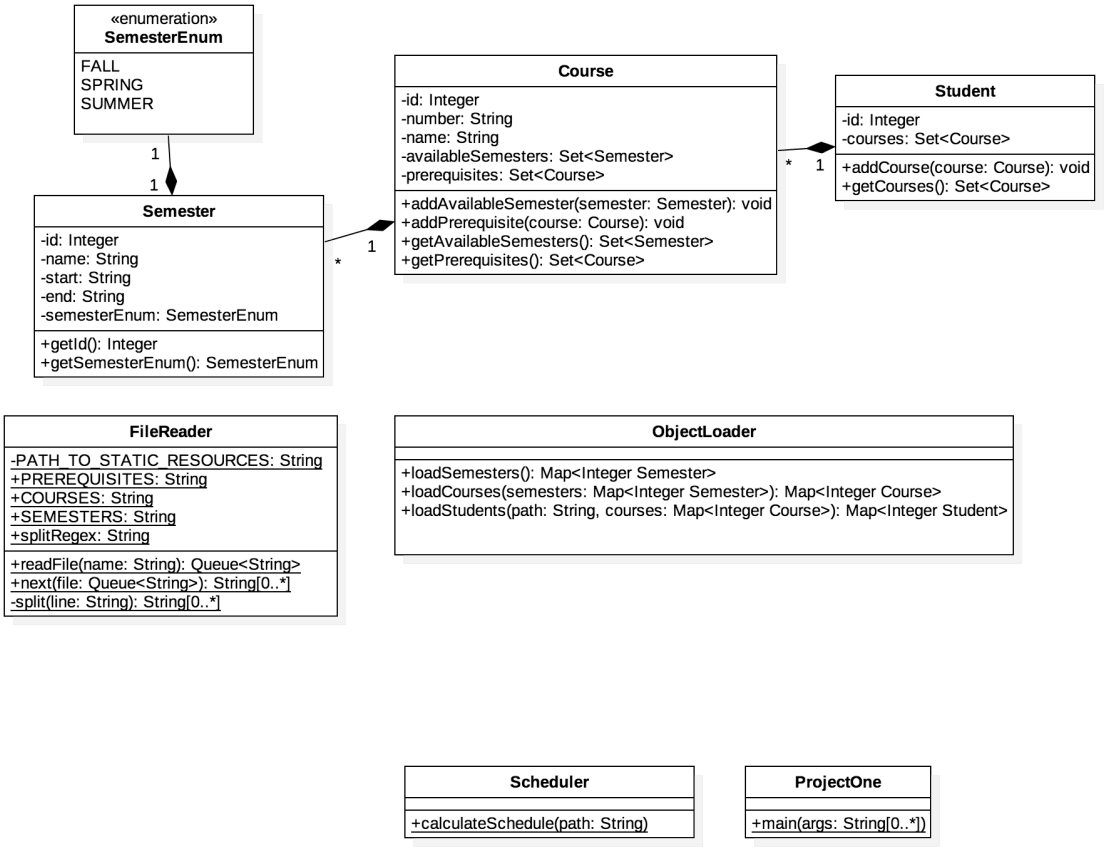
Overview:

The purpose of this project is to build a student-course pairing program. For this assignment we attempt to minimize course capacity given student schedules, available courses, course dependencies, and a list of semesters. We take as input the path to a file, which contains the student schedules. This program initially loads the static values of courses, semesters, and course dependencies from the resources directory. This program stores the values of these static resources in Plain Old Java Objects (POJOs) and performs some logic to connect them all together. We have three primary objects: Semesters, courses, and students. Once our objects are loaded, the program uses integer-programming techniques with the Gurobi optimizer to set constraints and optimize our model. This program uses Java 8.

Program Characteristics:

Lines of Code	445
Total Size in Bytes	184
Number of Classes	8
Average Number of Methods	3
Average Number of Attributes	2.5
Number of Inter-class Dependencies	3

UML Class Model design diagram:



Static description:

In this projct we use 8 different classes. Below I will list each class and its contribution to the project.

ProjectOne:

The “ProjectOne” class is the main driver for the project. It contains a single static method “main” which parses the command line arguments via a switch to find the student file input. From there it calls the “Scheduler” class, which performs the computations for minimizing course capacity.

Scheduler:

The “Scheduler” class contains the optimization logic for the project. Here we create the variables and constraints for the Gurobi model. The scheduler calls the ObjectLoader class’s static methods to load the static resources as well as the student schedule inputs.

ObjectLoader:

The “ObjectLoader” class creates instances of our primary POJO classes: “Semester”, “Course”, and “Student”. It does so by using the FileReader class to parse the respective files and create instances of the various classes.

FileReader:

The “FileReader” class is a simple utility that was created to help parse the various files. This class stores final Strings used to define the relative path of the resources directory and uses a BufferedReader to create a queue of String objects to make parsing files easier.

Semester:

The “Semester” object is the first of our POJOs. It is loaded first in this project. The primary attributes it stores semester ids and its type. The semester’s type is defined using the SemesterEnum class.

SemesterEnum:

The “SemesterEnum” class is a simple enumeration class that distinguishes the type of semester. The available values are “FALL”, “SPRING”, and “SUMMER”

Course:

This class defines the available courses for a student. It is loaded after the semester class and contains information about each course. It depends on other courses since it stores a set of courses that are prerequisites for itself, and depends on semester objects since it stores the available semesters it is in. The object loader parses the static course file and course_dependencies file to set these attributes.

Student:

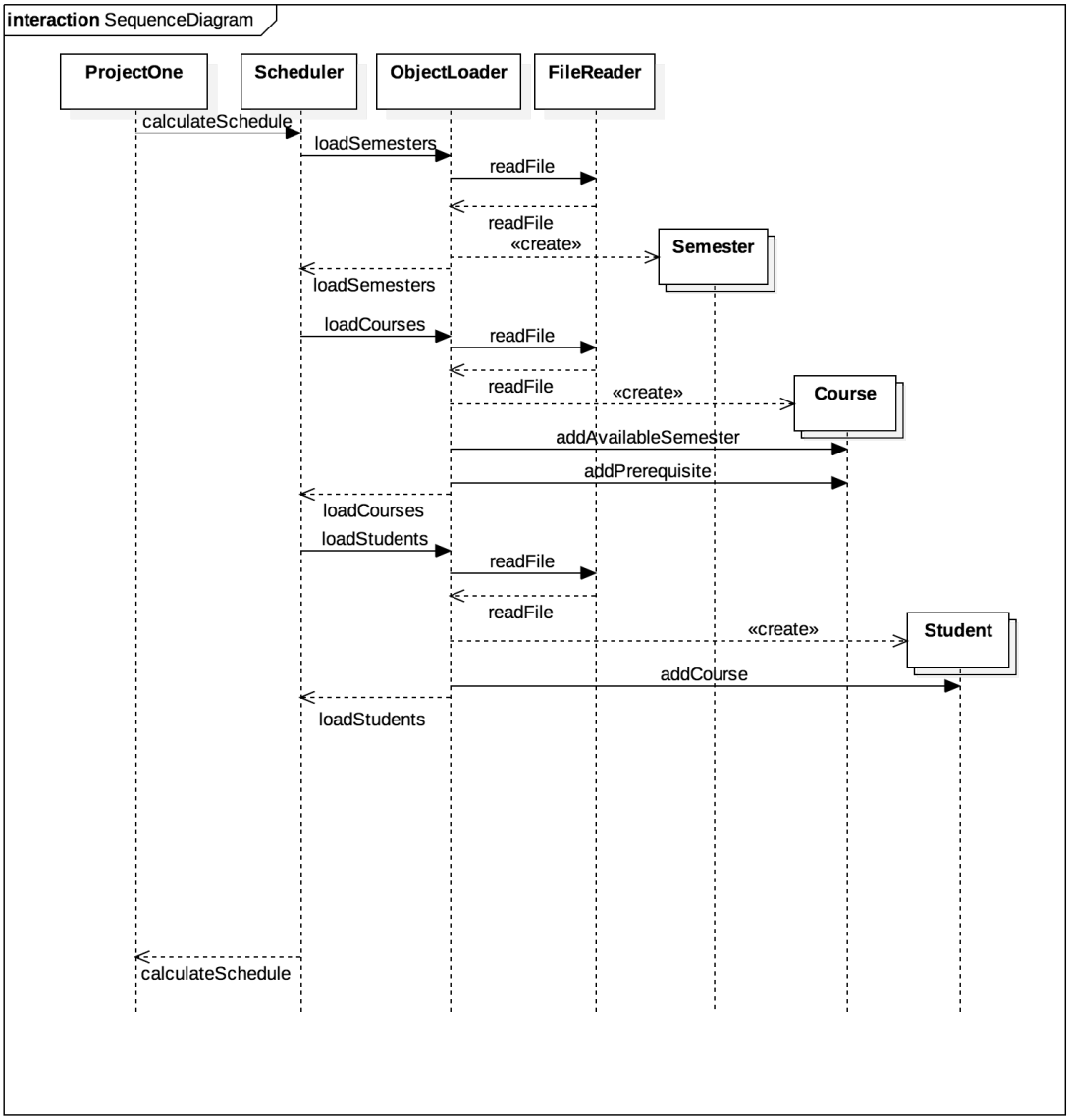
This class defines the set of courses a student wants to take. This is loaded through the object loader which parses the demanded courses for each student and adds it the respective object.

Below is a table that summarizes the class dependencies of each object.
Dependencies:

Class	Dependencies	Description
ProjectOne	Scheduler	Calls scheduler’s static method to optimize Gurobi model
Scheduler	ObjectLoader	Calls ObjectLoader’s static methods to create/load objects from CSV files
ObjectLoader	FileReader	Uses FileReader to parse CSV files
FileReader	n/a	n/a
Semester	SemesterEnum	Defines what type of semester it is with enum

SemesterEnum	n/a	n/a
Course	Semester	Each course has available semesters associated with it
Student	Course	Each student has a list of courses associated to it

Sequence diagrams:



Dynamic description:

There are minimal data flows in this program. Most of the flows revolve around reading the CSV files and parsing the data. As a result, I decided to create a ObjectLoader class and a FileReader class. The ObjectLoader contains the logic of parsing the files to store in the relativent objects. This class contains a static method to correspond with each type of object. Additionally since there is an order to how the objects are loaded, I use parameters to define how each is required. In the future this should be extended to a single method that loads all the objects so we can abstract away that requirement.

The FileReader class is a utility that was built simply to hold the relative path names for the static resources and to assist the ObjectLoader in parsing the raw files.

The three POJOs (Semester, Course, Student), simply store the raw information from the CSV files. Additionally, the Course objects store a set of semesters it is available in as well as a set of prerequisite Course objects, and the Student objects store a set of Courses a given student demands to take.

Low-level design considerations:

The motivation of this design was the compartmentalize each component and piece of logic. The Scheduler object's sole responsibility is to define the constraints and optimize the schedule. The ObjectLoader and FileReader are strictly used to parse the files and translate the raw data into the project's structure/objects. The POJOs are strictly to model the raw data in an understandable manner, and assist the Scheduler in defining the constraints by making that data available in the easiest manner.

Non-functional requirements:

1. Performance:
   This project does not do a particular good job scaling up performance. I rely heavily on Gurobi's optimizations in order to quickly solve the problem. I built the constraints serially, one after another. Additionally, because of the 3-dimensional aspect of the problem, I used triple for loops to load each constraint. A superior implementation would load each constraint in parallel since they are independent of each other.
2. Precision:
   Since constraints are all binary, the modeled problem is not too complex. Additionally, the project relies on Gurobi's optimization library rather than its own computations, so the limits of precision are primarily the limits of the library.
3. Evolvability:
   Due to the modular and compartimentalized design, adding new features should be fairly easy. Since we can simply swap our components to fit our needs. This design would need to include interfaces in when extended to provide better generalizations, and enable even easier integration. Additionally, since I abstracted the functions of the program to individual

components, modifying behavior it also easy since we can quickly define where to change the logic.

4. Usability:
   Since this project pre-defined our output and input command, there were minimal considerations for usability. I simply followed the project specifications.

Reflection:

Overall I am fairly satisfied with my design. I think some refactoring can be done to provide better package names and structure; however, I believe the structure of the program makes it easier to adapt to any additional requirements that would be defined. Some next steps are to optimize the constraint loading so we do not do it serially, but instead define constraints in parallel. An abstraction may be to create a handler class or a template for constraints to follow. Abstracting the raw data into objects worked very well, and having an object loader to set some of dependencies enabled me to simply divide the project into two major steps: loading all the data, and defining the problem. Loading the data was easy to test by simply putting my program in debug mode and ensuring the small datasets were being loaded correctly. By ensuring the data was loaded in a nice format, I could focus on defining the optimization problem without worrying about the inherit dependencies between semesters, courses, and students. The FileLoader may not be the best approach. I did not like defining my paths with final static Strings and felt very "hacky". A better approach may be utilize a framework that can load resources from another datastore.