Pattern Documentation

Ricardo Antonio Gutiérrez Esparza

To develop this application, a total of four software design patterns were implemented. A brief description of them is given in this document, which serves as documentation.

Creational Patterns

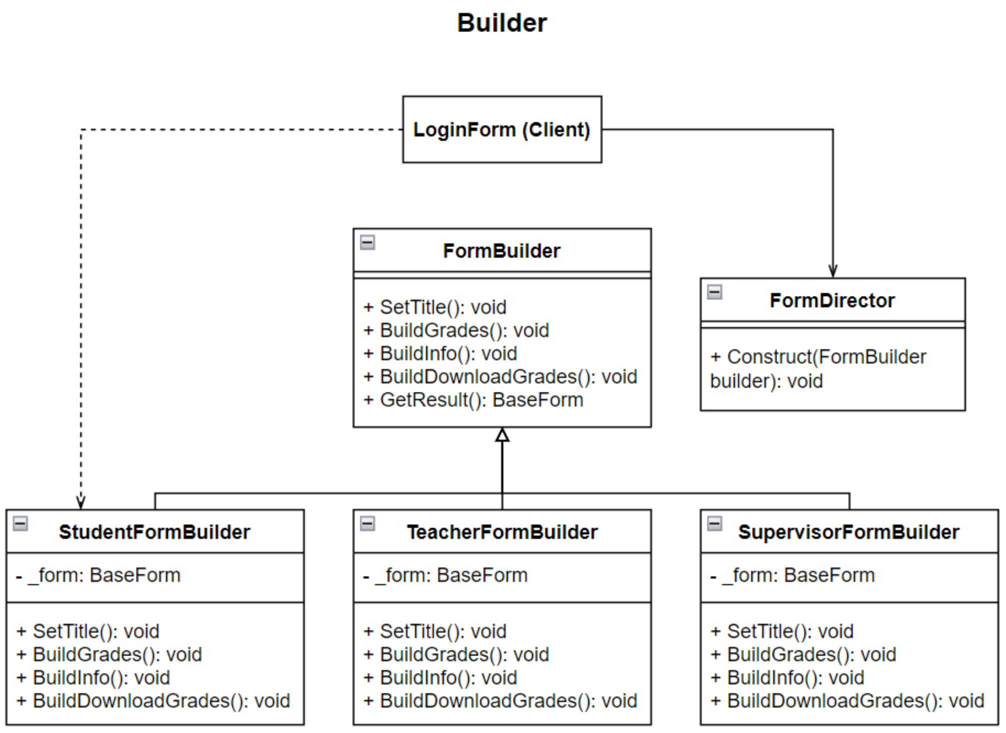
<u>Singleton</u>

Since we must read data from JSON files, the Singleton pattern was implemented in the JSONDataSource class. With the current state of the application, it is not really required because data is being read and loaded only once, so it is only helping us to have a quicker start by lazy loading the data into the application. However, if we are asked to update the app in the future to accept updates, we already have the class to avoid writing to the file from several instances at the same time.

<u>Builder</u>

To show the correct sections depending on the type of user that is using the application, a Builder pattern was implemented to create the corresponding forms. We have a total of three builders with the same steps: build grades, build profile information, and build download grades. Each builder corresponds to one of the three supported types of users, and if we were to onboard a new one to the application (let's say, tutor), then we would only have to write the new builder, but the rest of the code would remain the same.

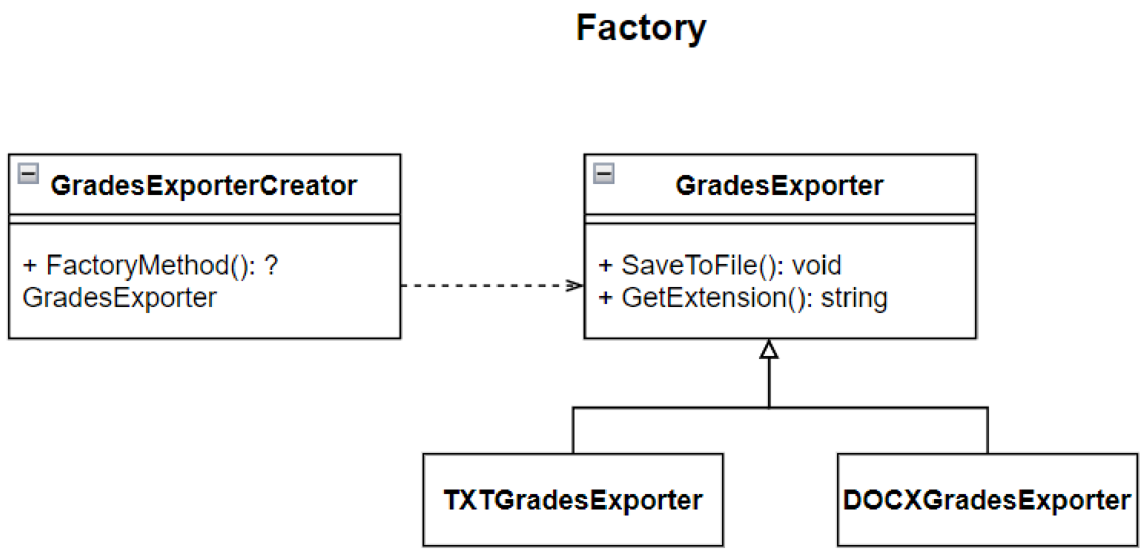
Below is a diagram that shows which classes are involved in this pattern.



Factory

The last of the creational patterns used was the Factory pattern. To export the grades of the students, we needed a way of supporting different formats. Since this piece of information is given by the application's configuration file, the caller code doesn't really need to know which specific format it is being exported to, so a factory of a common interface (GradeExporter) was used.

Once again, this is the diagram of the classes used by this pattern.

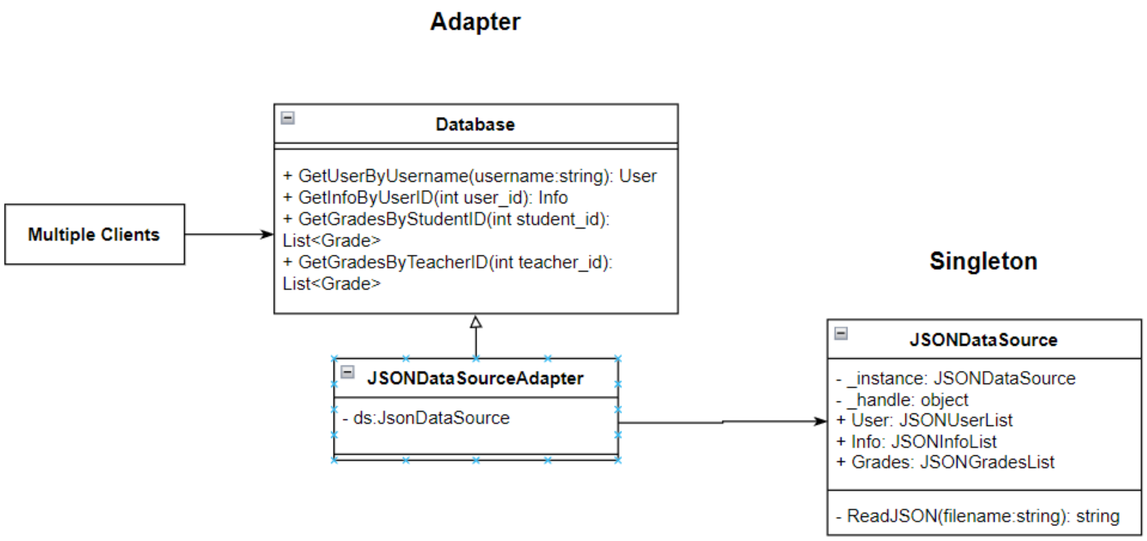


Structural Patterns

Adapter

An Adapter was used to fetch data from the JSONDataSource. Although it may seem like an unnecessary class, it is useful to introduce different data sources in the future. We are not binding the rest of the code to the JSONDataSource, but instead we are using the more general Database interface to build the adapters. If a data base is required, the corresponding adapter would have to be created without changing the rest of the client code. However, it is important to note that if we want to have interchangeable data sources at runtime, it would be better to apply a Factory instead.

Here is the diagram for this pattern.


Adapter
Singleton
Database
+ GetUserByUsername(username:string): User
+ GetInfoByUserID(int user_id): Info
+ GetGradesByStudentID(int student_id): List<Grade>
+ GetGradesByTeacherID(int teacher_id): List<Grade>
Multiple Clients
JSONDataSourceAdapter
- ds:JsonDataSource
JSONDataSource
- _instance: JSONDataSource
- _handle: object
+ User: JSONUserList
+ Info: JSONInfoList
+ Grades: JSONGradesList
- ReadJSON(filename:string): string