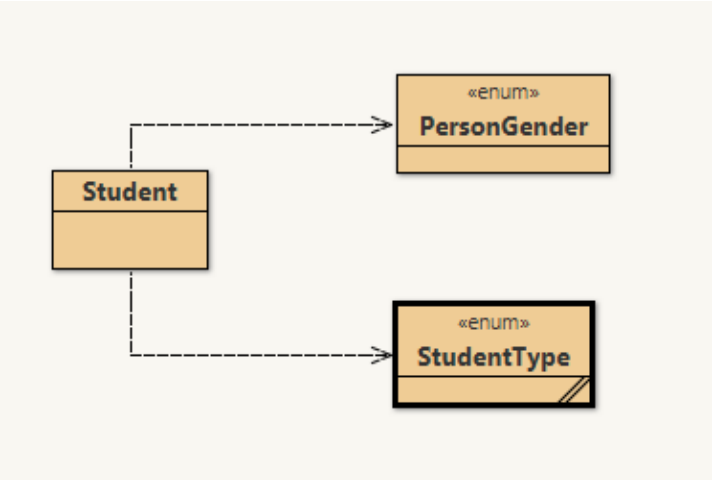
Problem Analysis

First of all I've decided to name the project based on the exercise as "StudentSystem", and I will refer to the project as "the system" in this document.

The system is based on the idea of reduced system that can manage student data for an institution like a school, or a kinder garden, so I decided to model it as a simple as possible, but keeping in mind the extensibility of it, and also its performance.

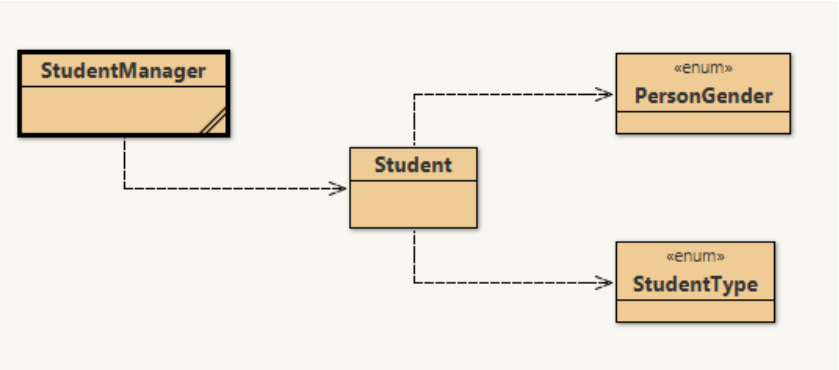
Before going to the performance area, I´ve modeled the initial classes as simple as possible:



So like in the picture above I defined a Student Class, that will define the students in the system, the exercise remarks four main attributes of all students:

1. Type, like High, University, Kinder, etc. We need to keep the design extensible so I´ve decided to make it an Enum class that allows me to iterate through it, add new types of student, or even delete an existing one.
2. Name, like all names, it can be modeled as a string.
3. Gender, this seems easy to define, like a Boolean specially in some conservative contexts, but the world is changing and we need to keep in mind that, maybe a kinder will not need that extensibility, but maybe a University can use that one, so as the Type, I´m defining it as an Enum.
4. A TimeStamp that indicates the last time that a student was modified, every language and system has some tools that allows the manipulation of this kind of data in the case of C#, is DateTime, then I´m using it.

So with this object Student, and its attributes I was able to model all the data of each student, and associate the data within an object, but only one, it was not enough, so I needed a larger class that can manage the data of different Students, delete them, create new ones, and perform searchs/queries in an efficient way, so I´ve created "StudentManager" a class that contains Collections composed by Student objects, resulting in this new model:



At this point I´ve decided to use two different collections that use the same data structure, Binary Search Tree that allows me to use logarithmic costs (O(log(n)) or O(n*(log(n))))in the queries instead of a linear cost (O(n)), that means more time and less efficiency, and we need it, because we are simulating the operations in a server.

There are three types of queries:

- By name: searching a student by name, alphabetically ordered results, for this one, I´ve used an ordered set, that allows me to have the data ordered, and with no repetition.
- By type: searching by the type of student, ordered by timestamp,
- By type and gender: searching by the type and gender of the student.

It seems simple, but here is the problem, the Student class must be comparable to have a given order, so you I´ve needed to implement the CompareTo method, to have a hierarchy, first Name, to have the set ordered by name, then the type, and finally the gender.

```
4 referencias
public int CompareTo(Object Obj)
{
    int result;
    if (Obj == null) {
        throw new ArgumentException("Object is not a Valid");
    }
    if (Obj is Student) {
        Student otherStudent = Obj as Student;
        result = this.Name.CompareTo(otherStudent.Name);
        if (result == 0) {
            result = this.Type.CompareTo(otherStudent.Type);
            if (result == 0) {
                result = this.Gender.CompareTo(otherStudent.Gender);
            }
        }
    } else {
        throw new ArgumentException("Object is not a Student");
    }
    return result;
}
```

Ok, with that we solve the first type of query but, is not useful for the second and third ones. For the last two I´ve used a dictionary that allows me to have a two field entry like this:

(Type of Student) -> (Ordered set of students of the type, by timestamp)

Example:

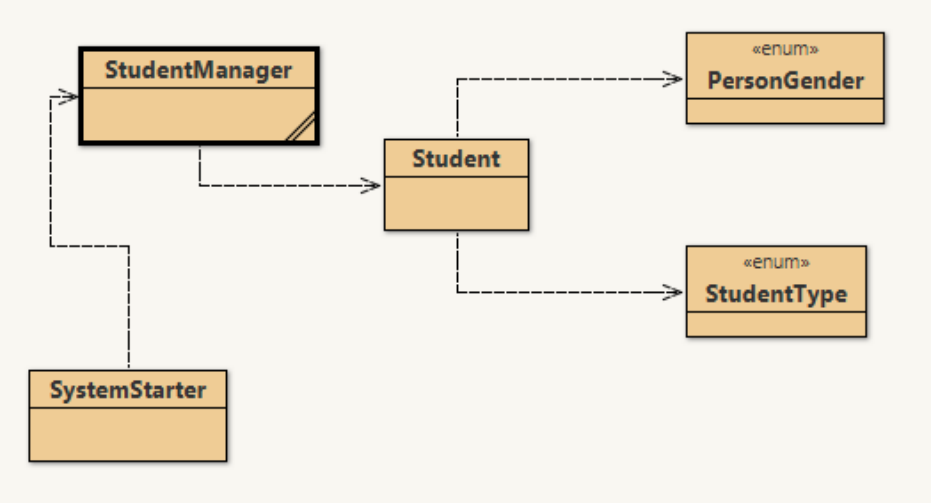
(High) -> {Anakin(2017), Luke(2015), Darth(2013), Isaac(2010)}

But it has another problem, the Student objects doesn't allow the date comparison, because are not useful for object comparisons or the other query, for that reason I´ve written a Comparer that allows me to order Students by timestamp, and have sets in that order:

```
1 referencia
class TimeStampComparer : IComparer<Student>
{
    0 referencias
    public int Compare(Student x, Student y)
    {
        int result = (-1)*(x.GetTimeStamp().CompareTo(y.GetTimeStamp()));
        if (result == 0) {
            result = x.CompareTo(y);
        }
        return result;
    }
}
```

With this last addition the problem of have the elements in a given order is solved, but we need a third class because we need to interact with some input/users, and we cannot allow the user to enter our inner classes or even worse, infect the design with a Main method in the Manager!!!

And also we need to manage the input parameters and validate them, so I´ve added a class for interaction:



The SystemStarter class allows us to interact with the rest of the model, without knowing anything about it. It is like simple view, a UI, connected to a simple class model, that can be used in a server,

for REAL world model, we can extend the model to use Threads, a DB, and even another UI can be added to this simple model., without destroying any class or functionality in the process.

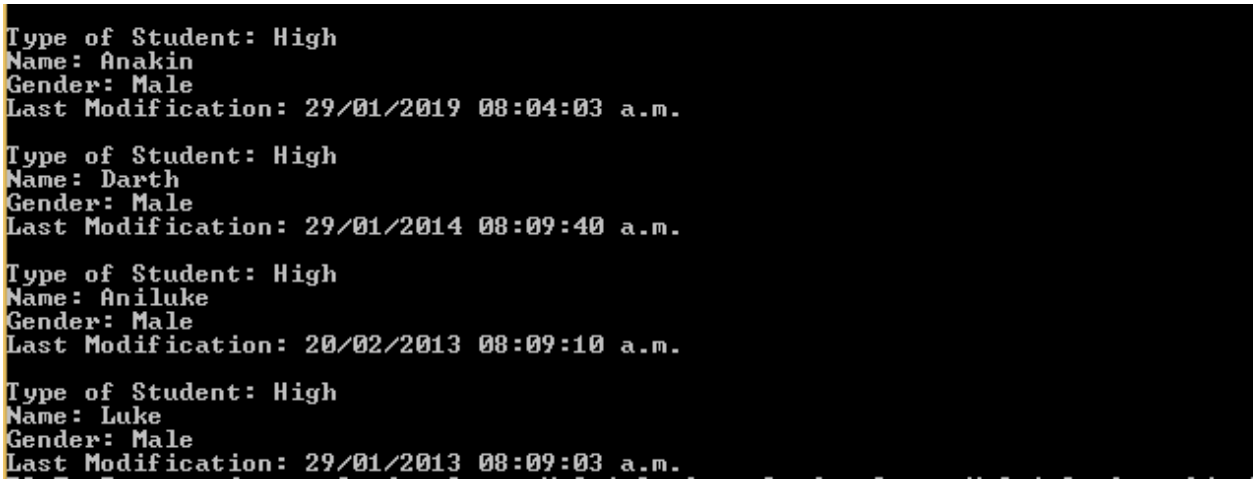
And last but not least I´ve added some Unit tests for the core methods of Student and StudentManager classes, to have the core functionality tested.

But also I´ve teste it manually myself:

The input:

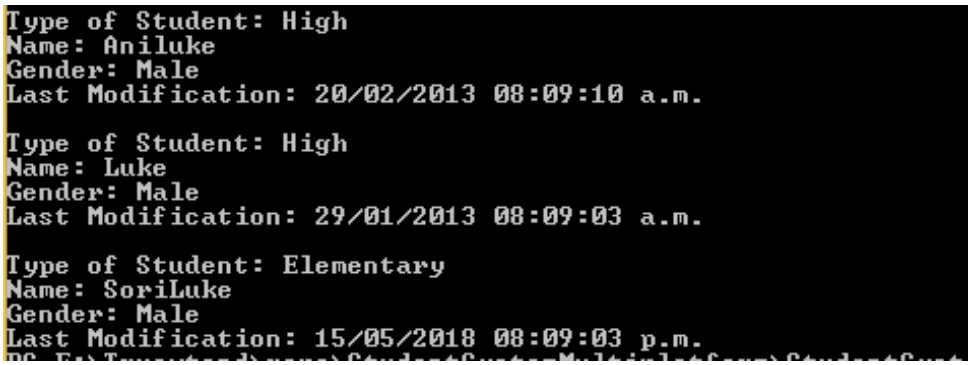
Kinder,Leia,F,20151231145934
High,Luke,M,20130129080903
High,Leia,F,20151231145934
High,Anakin,M,20190129080403
University,Leo,M,20151231145934
High,Darth,M,20140129080940
High,Aniluke,M,20130220080910
Elementary,SoriLuke,M,20180515200903

Searching by type=high gender=male



Searching by type=university

Searching by name=luke



The input file can be found in the repo, I´ve made two versions of the project one in .NET Framework, only runs in Windows, and the other one multiplatform in .NETCore, both can be found in the repo.

I´ve used VS 2019 for both projects.