Universidad Nacional Autónoma de México
Facultad de Ciencias
Modelado y programación 2019-2
Proyecto 1

Alumnos:

Amaya Lopez Dulce Fernanda

Lechuga Martinez Jose Eduardo

Justificación de patrones

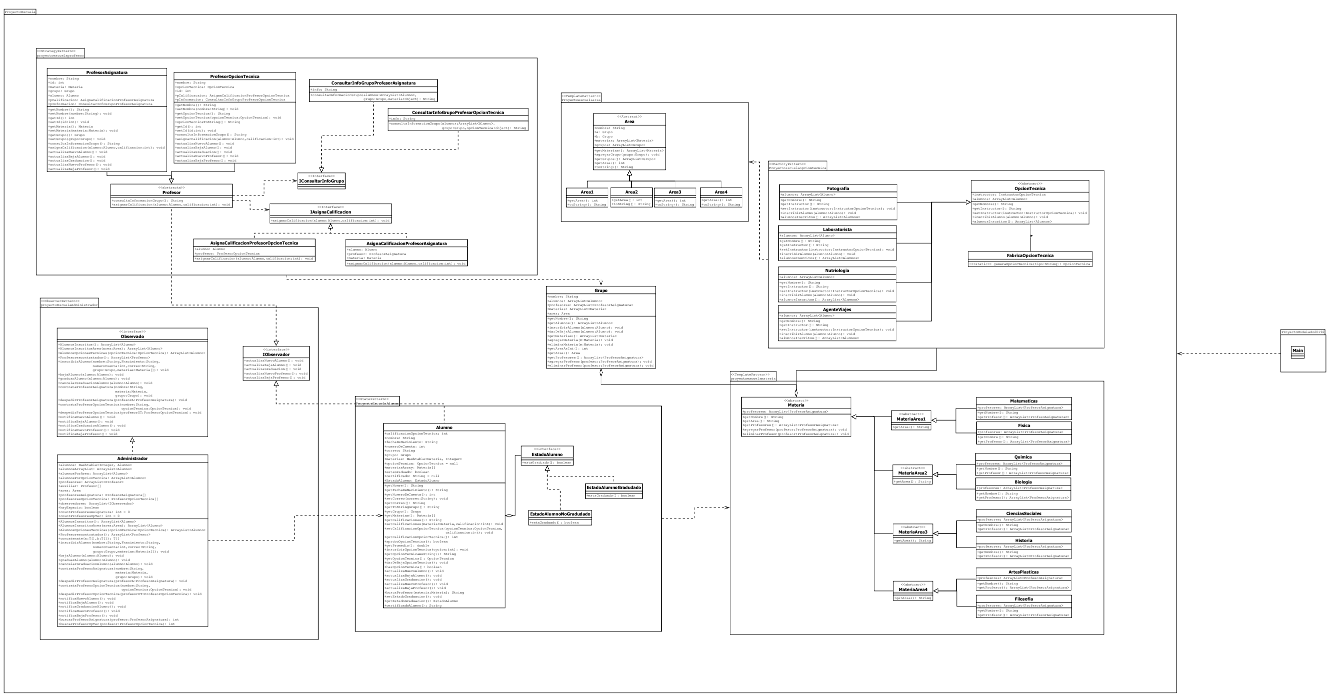
1. Strategy: Utilizamos a Strategy en la creación de los profesores ya que teníamos de dos tipos, de asignatura y de opción técnica. El objetivo de usar Strategy es hacer algoritmos intercambiables entonces aprovechamos esa parte para decidir como un profesor (dependiendo de su categoría) seleccionaría la forma de interactuar con el grupo ya sea consultando la información por grupo o asignando la calificación a algún alumno.

2. Template: El uso de este patrón fue en la creación de materias y en la creación de áreas, como en el caso de uso se especifica que cada área tiene materias especificas y tomando en cuenta que témplate se usa para crear un conjunto de subclases que ejecutan un código similar decidimos aprovecharlo. En el caso de materia, creando una clase abstracta materia que se encargaría de el comportamiento base de las materias (las cuales dividimos por área es decir, MateriaArea1, MateriaArea2,..) y usando sus subclases para crear instancias especificas de cada área En el caso de Área, creando una clase abstracta área que contenga el comportamiento base de las sub-áreas

3. Factory: Para crear las distintas opciones técnicas en la implementación nos ayudamos de este patrón cuyo objetivo es definir una interfaz para crear objetos estáticamente (sin el uso directo de new), en este caso creamos una clase abstracta OpcionTecnica que nos permitió definir el comportamiento base de las distintas opciones, a su vez ya que el objetivo es no usar new, creamos una clase FabricaOpcionTecnica que contiene un método abstracto que precisamente, se encarga de crear instancias de las opciones.

4. Observer: Este patrón fue de gran ayuda para la creación directa del administrador, ya que en el caso de uso se especifican distintos puntos que requieren de una actualización (De el administrador al alumno/maestro) y como el objetivo de Observer es relacionar un objeto a muchos creímos que fue la mejor elección. En este caso creamos la interfaz IObservado que se encargaría de definir los métodos de los observadores (en este caso el único observador es administrador), subsecuentemente creamos a Administrador y con el uso de la interfaz definimos sus distintos métodos. De el lado de los observadores tenemos dos maestros y alumnos, para poder homogeneizar el comportamiento que tendrían en contacto con el Administrador usamos la interfaz IObservador e hicimos tanto a alumno como a profesores implementarla.

5. State: Usamos State para manejar los estados de un alumno, es decir si este estaba graduado o no, el objetivo de State es el de crear un objeto y cambiar su estado interno y, gracias a esto pensamos que seria conveniente usar State precisamente para modificar el estado del alumno (el cambio lo hace el administrador).


ProfesorAsignatura
ProfesorOpcionTecnica
ConsultarInfoGrupoProfesorAsignatura
ConsultarInfoGrupoProfesorOpcionTecnica
IConsultarInfoGrupo
Profesor
IAsignaCalificacion
AsignaCalificacionProfesorOpcionTecnica
AsignaCalificacionProfesorAsignatura
Area
Area1
Area2
Area3
Area4
Fotografia
Laboratorista
Nutriologia
AgenteViajes
OpcionTecnica
FabricaOpcionTecnica
Grupo
Observado
IObservador
Administrador
Alumno
EstadoAlumno
EstadoAlumnoGraduado
EstadoAlumnoNoGraduado
Materia
MateriaArea1
MateriaArea2
MateriaArea3
MateriaArea4
Matematicas
Fisica
Quimica
Biologia
CienciasSociales
Historia
ArtesPlasticas
Filosofia
Main

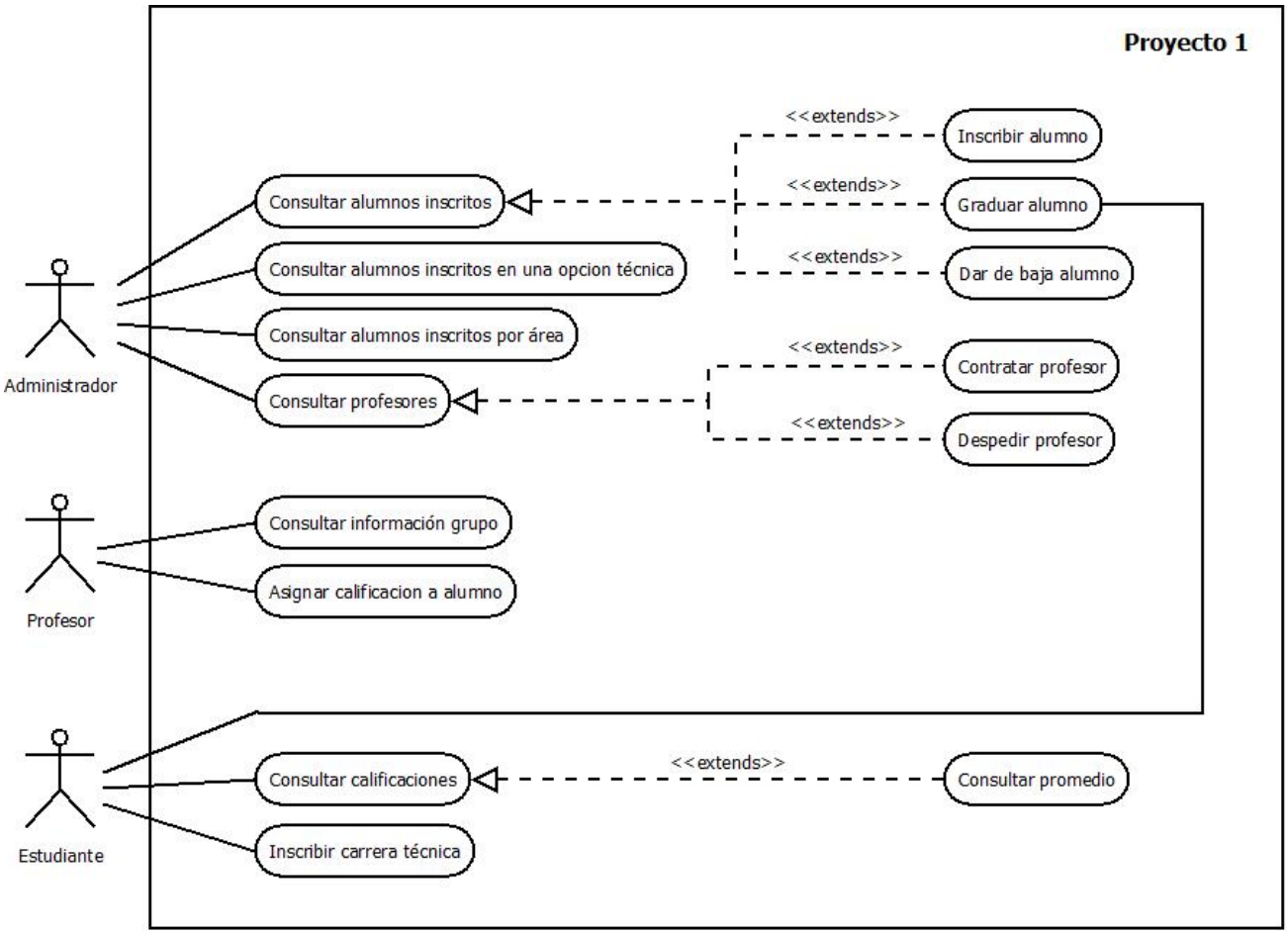


Figure 1: Caso de uso

Forma de ejecución del programa

La forma de ejecución es de mediante el uso de ANT y un archivo Build.xml. En el caso de Ant, lo que deben hacer es abrir su consola, dirigirse a la carpeta src del archivo y escribir "ant compile" subsecuentemente escribir "ant run" y seguir los pasos en la interfaz.

En su defecto pueden acceder a el repositorio en Github donde prodran descargar la versión de Netbeans y poder ejecutarlo desde ahí.

https://github.com/Lechuga194/Proyecto01Modelado2019-2

Bibliografía

Banas, D. (2012, September 01). Factory Design Pattern. Retrieved from https://www.youtube.com/watch?v=ub0DXaeV6hA

Banas, D. (2012, October 04). Template Method Design Pattern. Retrieved from https://www.youtube.com/watch?v=aR1B8MlwbRI

Design Patterns and Refactoring. (n.d.). Retrieved from https://sourcemaking.com/design$_patterns/observer$

Design Patterns and Refactoring. (n.d.). Retrieved from https://sourcemaking.com/design$_patterns/factory_method$

Design Patterns and Refactoring. (n.d.). Retrieved from https://sourcemaking.com/design$_patterns/state$

Design Patterns and Refactoring. (n.d.). Retrieved from https://sourcemaking.com/design$_patterns/strategy$

Design Patterns and Refactoring. (n.d.). Retrieved from https://sourcemaking.com/design$_patterns/template_method$

Ltd., T. P. (2018, July 19). Observer Design Pattern. Retrieved from https://www.youtube.com/watch?v=$_zGduwU6fIQ$

Okhravi, C. (2017, May 16). Factory Method Pattern – Design Patterns (ep 4). Retrieved from https://www.youtube.com/watch?v=EcFVTgRHJLM

Okhravi, C. (2017, February 01). Observer Pattern – Design Patterns (ep 2). Retrieved from https://www.youtube.com/watch?v=$_Bpmfnqjgz Q$

Okhravi, C. (2017, December 19). State Pattern – Design Patterns (ep 17). Retrieved April 11, 2019, Retrived from https://www.youtube.com/watch?v=N12L5D78MAA

Telusko. (2016, March 01). Factory Design Pattern in Java. Retrieved from https://www.youtube.com/watch?v=pt1IbV1aSZ4