FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGIA (UNT)

PROGRAMACIÓN II (E11)

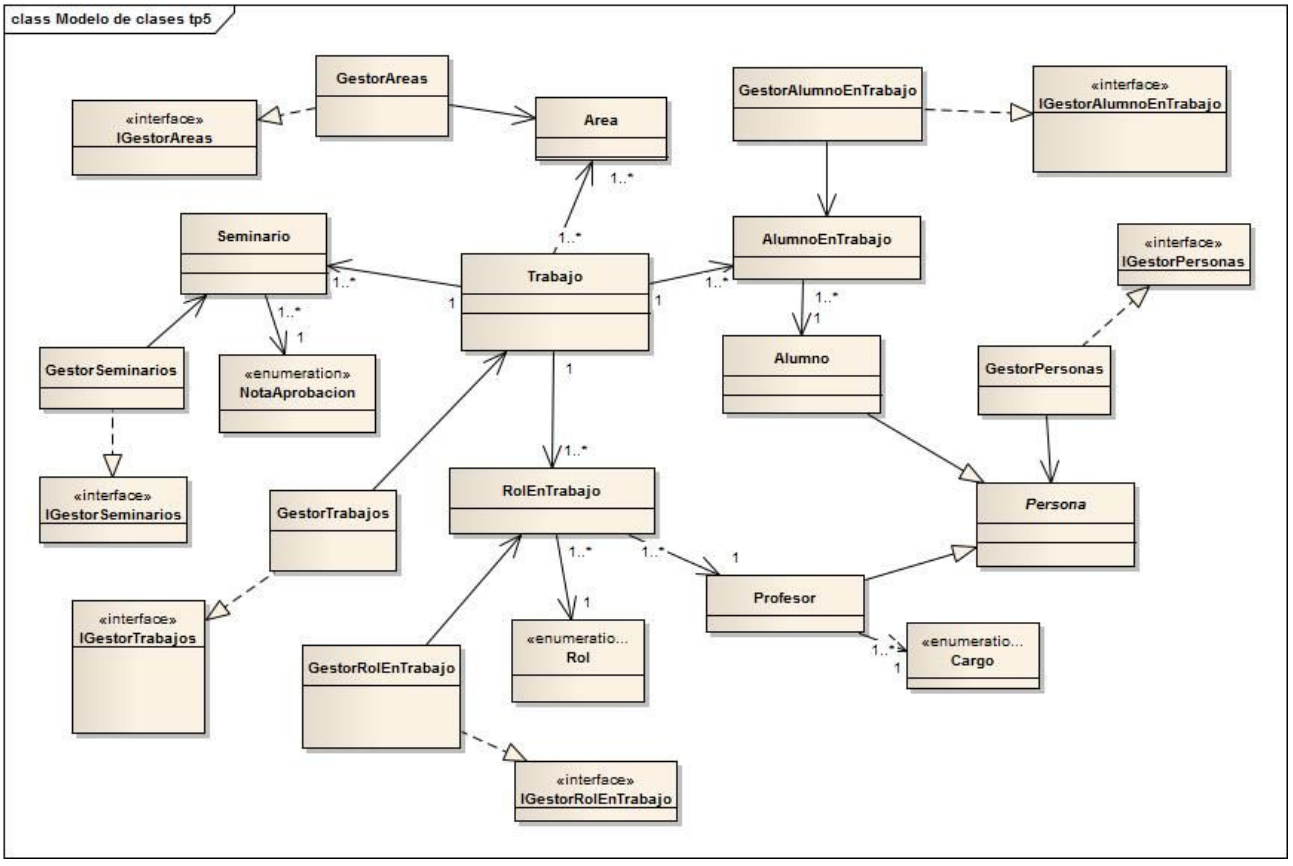
Evaluación 2 - 2018

Objetivos

- Diseño básico de una interfaz gráfica
- Integrar los distintos conceptos vistos durante el cursado de la asignatura
- Trabajar grupalmente en un proyecto

Para resolver

A partir de todos los Trabajos Prácticos realizados y del siguiente diagrama de clases, realizar las modificaciones y agregados necesarios para implementar una aplicación con una interfaz gráfica empleando el patrón MVC.



A continuación se detallan los atributos y operaciones de las distintas clases (en el caso de los atributos, no se especifican aquellos que se deduzcan de las relaciones. En el caso de las operaciones, sólo se especifican las públicas, y sus nombres deberán ser respetados por cada grupo:

Area	
Atributos	**Operaciones**
nombre: String	Area(String): constructor
	String verNombre(): devuelve el nombre del área
	String toString(): devuelve la representación de un área como cadena

Persona	
Atributos	**Operaciones**
apellidos: String	Persona(String apellidos, String nombres, int dni): constructor
nombres: String	String verApellidos(): devuelve los apellidos de la persona
dni: int	void asignarApellidos(String): asigna los apellidos a la persona
	String verNombres(): devuelve los nombres de la persona
	void asignarNombres(String): asigna los nombres a la persona
	int verDNI(): devuelve el dni de la persona
	String toString(): devuelve la representación de una persona como cadena

Alumno	
Atributos	**Operaciones**
cx: String	Alumno(String apellidos, String nombres, int dni, String cx): constructor
	String verCX(): devuelve el cx del alumno
	void asignarCX(String): asigna el cx al alumno

Profesor	
Atributos	**Operaciones**
cargo: Cargo	Profesor(String apellidos, String nombres, int dni, Cargo cargo): constructor
	Cargo verCargo(): devuelve el cargo del profesor
	void asignarCargo(Cargo): asigna el cargo al profesor

Seminario	
Atributos	**Operaciones**
fechaExposicion: LocalDate	Seminario(LocalDate, NotaAprobacion, String): constructor
notaAprobacion: NotaAprobacion	LocalDate verFechaExposicion(): devuelve la fecha de exposición del seminario
observaciones: String	NotaAprobacion verNotaAprobacion(): devuelve la nota de aprobación del seminario
	void asignarNotaAprobacion(NotaAprobacion): asigna la nota de aprobación
	String verObservaciones(): devuelve las observaciones del seminario
	void asignarObservaciones(String): asigna las observaciones al seminario

Trabajo	
Atributos	**Operaciones**
titulo: String	Trabajo(String, int, List<Area>, LocalDate fPresentacion, LocalDate fAprobacion, List<RolEnTrabajo>, List<AlumnoEnTrabajo>): constructor
duracion: int	Trabajo(String, int, List<Area>, LocalDate fPresentacion, LocalDate fAprobacion, LocalDate fFinalizacion, List<RolEnTrabajo>, List<AlumnoEnTrabajo>): constructor
fechaPresentacion: LocalDate	String verTitulo(): devuelve el título del trabajo
fechaAprobacion: LocalDate	int verDuracion(): devuelve la duración del trabajo
fechaFinalizacion: LocalDate	List<Area> verAreas(): devuelve las áreas del trabajo
	LocalDate verFechaPresentacion(): devuelve la fecha de presentación del trabajo
	LocalDate verFechaAprobacion(): devuelve la fecha de aprobación del trabajo
	LocalDate verFechaFinalizacion(): devuelve la fecha de finalización del trabajo
	void asignarFechaFinalizacion(LocalDate): asigna la fecha de finalización al trabajo
	Profesor verTutorOCotutor(Rol): devuelve el profesor ques e encuentra

	actualmente con el rol especificado (tutor o cotutor). Si no hay tutor/cotutor, devuelve null.
	List<Profesor> verJurado(): devuelve el jurado del trabajo, ordenado por apellido y nombre.
	List<RolEnTrabajo> verProfesoresConRoles(): devuelve la lista de profesores con sus roles en el trabajo
	List<AlumnoEnTrabajo> verAlumnos(): devuelve la lista de alumnos del trabajo (los que actualmente participan y los que no)
	List<AlumnoEnTrabajo> verAlumnosActuales(): devuelve la lista de los alumnos que actualmente participan del trabajo (sin fecha de finalización)
	int cantidadProfesoresConRol(Rol): devuelve la cantidad de profesores con el rol especificado en el trabajo
	int cantidadAlumnos(): devuelve la cantidad de alumnos (actuales y no) en el trabajo
	int cantidadSeminarios(): devuelve la cantidad de seminarios que tiene el trabajo
	boolean tieneSeminarios(): informa si el trabajo tiene presentado seminarios
	boolean tieneEsteProfesor(Profesor): informa si el profesor especificado participa en el trabajo
	void agregarRolEnTrabajo(RolEnTrabajo): agrega el profesor con su rol al trabajo
	void agregarSeminario(Seminario): agrega el seminario especificado (es para leer los seminarios del archivo)
	boolean tieneEsteSeminario(Seminario): informa si el trabajo tiene o no el seminario especificado
	String nuevoSeminario(LocalDate, NotaAprobacion, String): crea un seminario
	String modificarSeminario(Seminario, NotaAprobacion, String): modifica un seminario
	int verUltimoSeminario() *: devuelve la posición del último seminario agregado/modificado (es para manejar la tabla de seminarios)
	List<Seminario> verSeminarios(): devuelve los seminarios ordenados según su fecha de exposición
	boolean estaFinalizado(): informa si el trabajo está o no finalizado
	void cancelar() *: cancela el agregado/modificación del seminario (es para manejar la tabla de seminarios)

RolEnTrabajo	
Atributos	**Operaciones**
fechaDesde: LocalDate	RolEnTrabajo(Profesor, Rol, LocalDate): constructor
fechaHasta: LocalDate	RolEnTrabajo(Profesor, Rol, LocalDate, LocalDate, String): constructor
razon: String	Profesor verProfesor(): devuelve el profesor
	Rol verRol(): devuelve el rol
	LocalDate verFechaDesde(): devuelve la fecha a partir de la cual el profesor comienza el trabajo
	LocalDate verFechaHasta(): devuelve la fecha hasta la cual el profesor participó en el trabajo
	String verRazon(): devuelve la razón por la cual el profesor dejó de participar en el trabajo
	void asignarFechaHasta(LocalDate): asigna la fecha de finalización de un profesor en el trabajo
	void asignarRazon(String): asigna la razón por la cual un profesor finalizó en el trabajo

AlumnoEnTrabajo	
Atributos	**Operaciones**
fechaDesde: LocalDate	AlumnoEnTrabajo(Alumno, LocalDate): constructor
fechaHasta: LocalDate	AlumnoEnTrabajo(Alumno, LocalDate, LocalDate, String): constructor
razon: String	Alumno verAlumno(): devuelve el alumno
	LocalDate verFechaDesde(): devuelve la fecha a partir de la cual el alumno comienza el trabajo
	LocalDate verFechaHasta(): devuelve la fecha hasta la cual el alumno participó en el trabajo

	String verRazon(): devuelve la razón por la cual el alumno dejó de participar en el trabajo
	void asignarFechaHasta(LocalDate): asigna la fecha de finalización de un alumno en el trabajo
	void asignarRazon(String): asigna la razón por la cual un alumno finalizó en el trabajo

* Estas operaciones sirven exclusivamente para la interfaz gráfica. También están en las interfaces IGestorAreas, IGestorPersonas e IGestorTrabajos. Queda a criterio de cada grupo si las implementa o no.

Sobre los gestores e interfaces

- Cada gestor deberá dar persistencia a sus objetos en un archivo de texto. Los nombres de los mismos se especificarán mediante cadenas constantes.
- En las distintas interfaces hay definidas una serie de constantes. Queda a criterio de cada grupo si las usa o no, como así también la elección de modificarlas con los valores que cada uno considere.

Sobre la apariencia de la aplicación

Cada grupo podrá diseñar las ventanas empleadas del modo en que desee teniendo en cuenta que:

- Se deberán poder realizar las funcionalidades requeridas de acuerdo al tema que se les haya asignado.
- El listado y resultado de la búsqueda de los diferentes objetos se presenten en una tabla.

Sobre la estructura de la aplicación

La aplicación se estructurará según la siguiente descomposición:

Paquete	Descripción
gui.interfaces	Todas las interfaces usadas en el proyecto
gui.areas.controladores	Clases para los controladores relacionados con las áreas
gui.areas.modelos	Clases para los modelos relacionados con las áreas
gui.areas.vistas	Clases para las vistas relacionadas con las áreas
gui.personas.controladores	Clases para los controladores relacionados con las personas
gui.personas.modelos	Clases para los modelos relacionados con las personas
gui.personas.vistas	Clases para las vistas relacionadas con las personas
gui.seminarios.controladores	Clases para los controladores relacionados con los seminarios
gui.seminarios.modelos	Clases para los modelos relacionados con los seminarios

gui.seminarios.vistas	Clases para las vistas relacionadas con los seminarios
gui.trabajos.controladores	Clases para los controladores relacionados con los trabajos
gui.trabajos.modelos	Clases para los modelos relacionados con los trabajos
gui.trabajos.vistas	Clases para las vistas relacionadas con los trabajos
gui.principal.controladores	Clases para los controladores relacionados con la ventana principal
gui.principal.vistas	Clases para las vistas relacionadas con la ventana principal

Componente para selección de fecha

- Para el componente que permite seleccionar una fecha, se puede usar `JDateChooser`:
- En la carpeta del proyecto se encuentra el archivo `jcalendar-1.4.jar`.
- Sobre la paleta de componentes, hacer clic derecho y seleccionar "Administrador de Paleta..."
- Seleccionar "Añadir archivo JAR.."
- Seleccionar el archivo `jcalendar-1.4.jar`
- Seleccionar el componente `JDateChooser`
- Seleccionar la categoría de paleta (Controles Swing)

Para leer una fecha:

- `Date d = dateChooser.getCalendar().getTime();`
- //`dateChooser` es el nombre del componente `JDateChooser`
- `LocalDate f = d.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();`

Para escribir una fecha:

- `GregorianCalendar f;`
- `f=GregorianCalendar.from(trabajo.verFechaAprobacion().atStartOfDay(ZoneId.systemDefault()));`
- `dateChooser.setCalendar(f);`
- //`dateChooser` es el nombre del componente `JDateChooser`

En cuanto al proyecto:

- Todo el trabajo se dividirá en 5 equipos, formado cada uno por 4 grupos.
- Cada grupo tendrá asignado una rama de trabajo en el repositorio GitHub. Sobre esta rama implementará únicamente lo requerido a su grupo, teniendo la libertad de crear tantas ramas como considere necesario, siempre y cuando su trabajo final se encuentre en la rama que le fue asignada.
- De los 4 grupos que forman un equipo, 3 estarán formados por 2 personas y el cuarto por 3. Los grupos de 2 personas trabajarán sobre el manejo de áreas, personas y seminarios, y el grupo de 3 personas trabajará sobre el manejo de los trabajos.
- Se tendrá en cuenta la cantidad de *commits* producidos por cada integrante de grupo, como así también el contenido de los mismos.