Tema 11:

Documentación y Diagramas 1

Práctica:

Diagramas UML Clases

Repositorio de github:

https://github.com/Jonycuenca/UMLClases_Jonattan.git

Vamos a realizar la práctica del tema 11, en la cual se nos pide instalar el plugin PlantUml en el entorno de desarrollo Intellij IDEA. Hemos instalado este plugin de la misma manera que se expone en la descripción de la práctica:

Hemos creado un nuevo proyecto con Intellij IDEA. Una vez tenemos el proyecto, en Intellij hemos accedido a: File/Settings/Plugins y hemos buscado e instalado desde el Marketplace el plugin PlantUML.

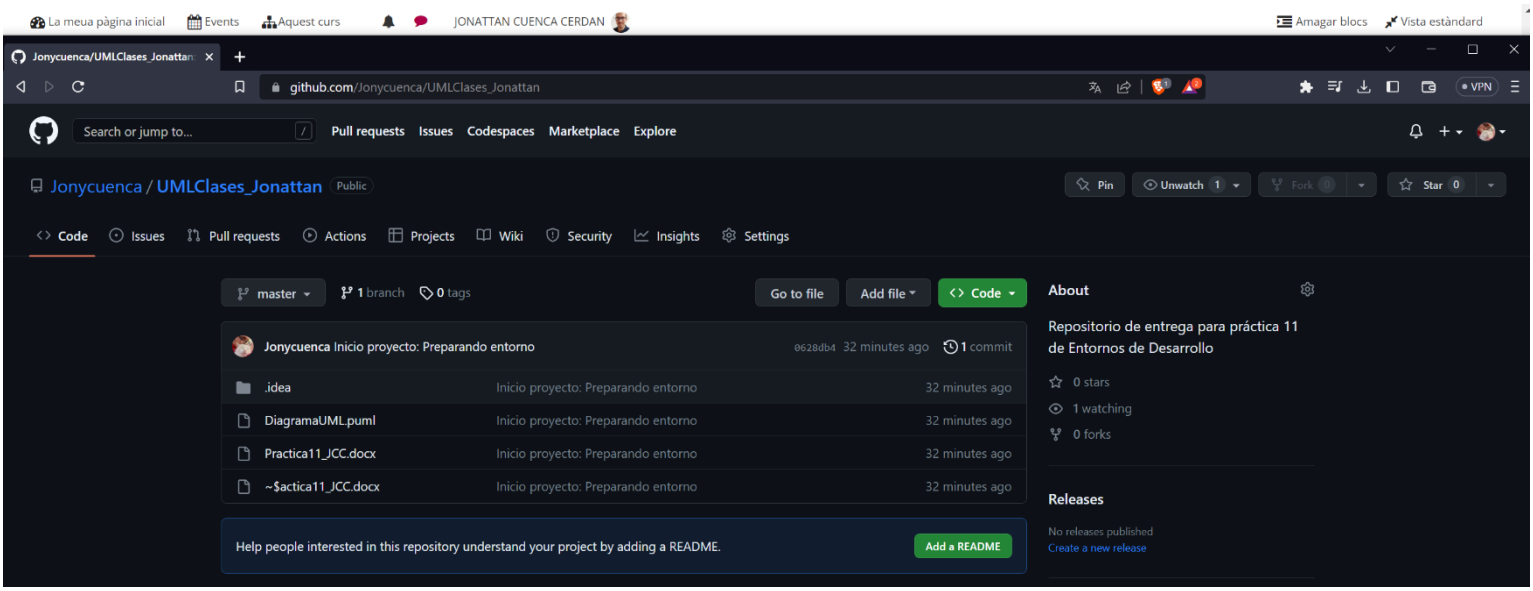
Una vez tenemos instalado el plugin, hemos creado dentro del proyecto un documento UML: File/New/ PlantUML file. Lo hemos nombrado “DiagramaUML”.

Una vez tenemos creado el documento UML, he creado un repositorio remoto en nuestra cuenta de GitHub como ya hemos realizado en prácticas anteriores: En nuestra cuenta, accedemos a los repositorios y creamos uno nuevo, le damos el nombre que se nos pide en la descripción de la práctica “UMLClases_Jonattan” y añadimos una breve descripción. Una vez tenemos creado el repositorio copiamos el enlace HTTPS que podemos conseguir clicando en el apartado “Code” del repositorio.

En IntelliJ, habilitamos el uso del Sistema de Control de Versiones: En la pestaña VCS/Enable Version Control Integration y seleccionamos Git.

Una vez hemos habilitado el VCS de Git, accedemos al menú Git/Commit. Seleccionamos todos los archivos del proyecto, ponemos un comentario y hacemos clic en Commit and Push… Nos aparecerá una ventana donde nos indica la rama del proyecto que se va a subir y hay que definir la ruta del archivo remoto clicando en “Define remote”. Le damos nombre y añadimos el enlace que habíamos copiado del repositorio remoto en el cuadro de URL. Aceptamos y seleccionamos Push.

Comprobamos que el archivo se ha subido al repositorio remoto correctamente:



Ya tenemos preparado el entorno para ir realizando el diagrama UML que se nos pide.

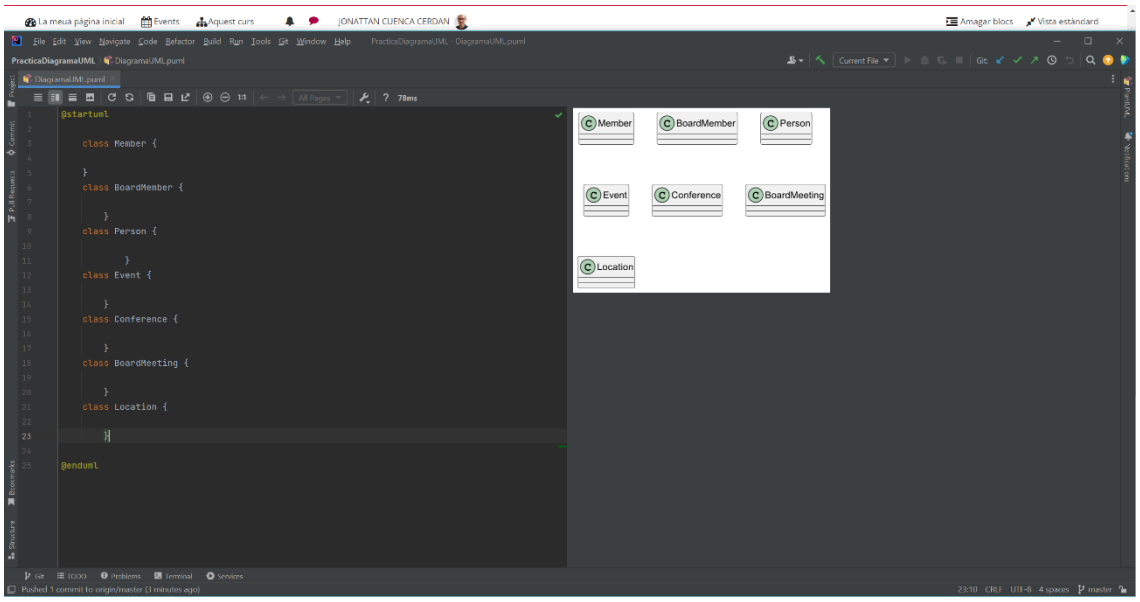
Para la práctica se nos pide realizar el diagrama UML del siguiente caso:

La Asociación de Antiguos Alumnos de la UOC nos ha pedido si podemos ayudarles a confeccionar un programa que les permita gestionar a sus asociados, **eventos** y demás elementos relacionados.
Los asociados se pueden dividir en **miembros numerarios** y en **miembros de la junta directiva**, que es elegida por votación en una asamblea general cada cuatro años. La única diferencia entre ellos es que los miembros de la junta directiva son convocados a las **reuniones de junta** y los demás miembros no, pero el resto de actividades que se realizan están abiertas a todos los miembros de la asociación.
La convocatoria de un evento se realiza por correo electrónico a todos los miembros activos en el momento del envío, recibiendo un enlace para aceptar su participación. En todos los eventos, la aceptación de los asistentes se realiza por orden de llegada ya que, en algún caso, se puede dar que el número de asistentes sea limitado, como en las **conferencias**.
En la convocatoria, también aparece información sobre el **lugar** que en muchos casos se repite, por lo que nos han dicho que quieren almacenar los datos para futuros usos.

1º Vamos a identificar las clases:

- Miembro (o miembro numerario) (Member)
- Miembro de la junta directiva (BoardMember)
- Evento (Event)
- Conferencia (Conference)
- Reunión de la junta directiva (BoardMeeting)
- Localización (Location)
- Asociación (AAUOC)

Añadimos la clase Persona (Person) para identificar a los conferenciantes, ya que éstos pueden no ser miembros de la asociación.



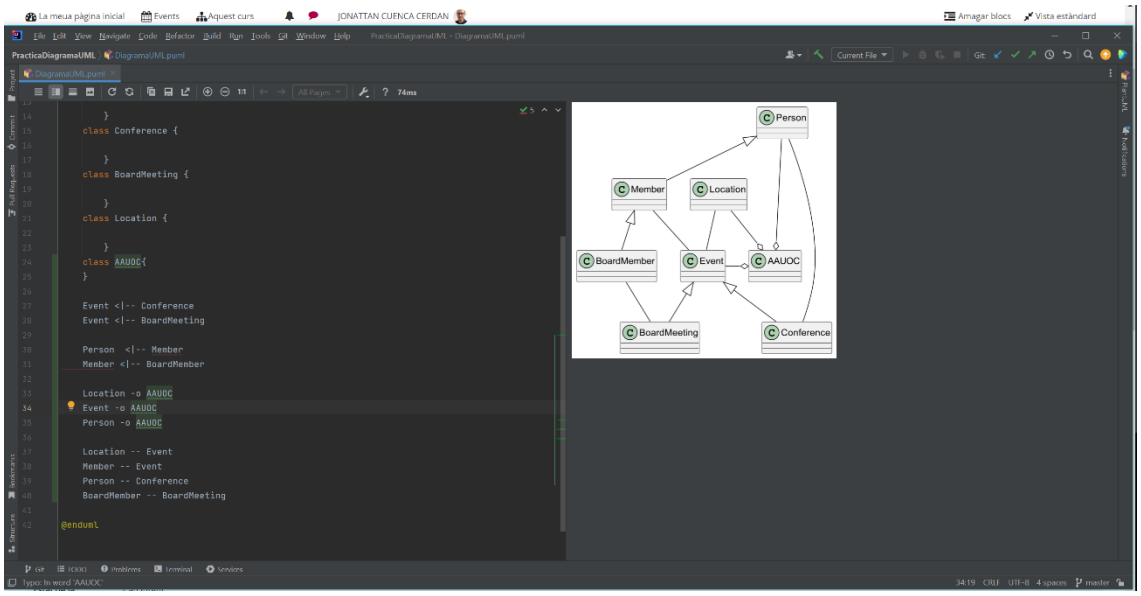
Una vez identificadas las clases, realizamos commit and push al repositorio de Github.

2º Vamos a crear el modelo de datos. Para esto, hay que detectar la jerarquía existente entre las clases existentes en el enunciado. Detectamos que existe la superclase Evento y tiene dos subclases: las conferencias y las reuniones de la junta directiva.

También existe una jerarquía entre los miembros de la asociación, cuya superclase es Persona (Person), que tiene la subclase Miembro (Member) y que ésta a su vez tiene por subclase a Miembro de la junta directiva (BoardMember).

Además, tenemos las clases Localización (location) y asociación (AAUOC), que se relacionan con el resto de clases.

Así quedaría el diagrama UML de momento:

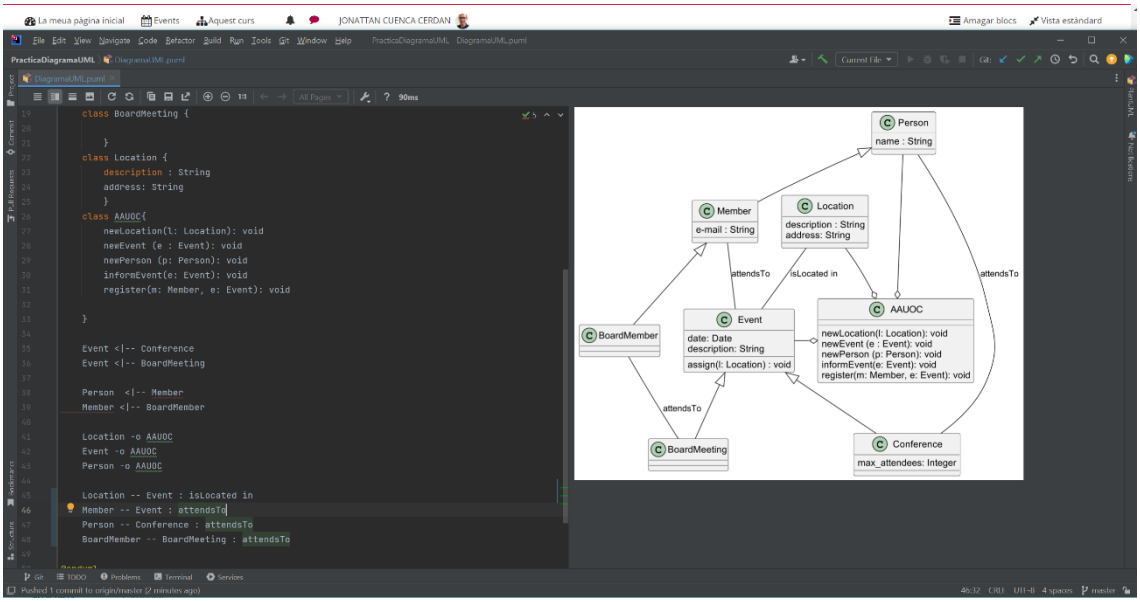


Realizamos commit and push a nuestro repositorio remoto de GitHub.

3º Incluir los atributos y los métodos más relevantes a las clases.

La clase AAUOC necesita un conjunto de métodos para añadir nuevos eventos, personas y localizaciones al sistema (newLocation(), newPerson(), newEvent()) además de un método para informar a los miembros de la convocatoria de un evento (informEvent()). Al mismo tiempo, se dice que los usuarios necesitarán confirmar la asistencia a los eventos (método register(), que deberá almacenar los asistentes por orden y controlar el número máximo de éstos si fuera necesario).

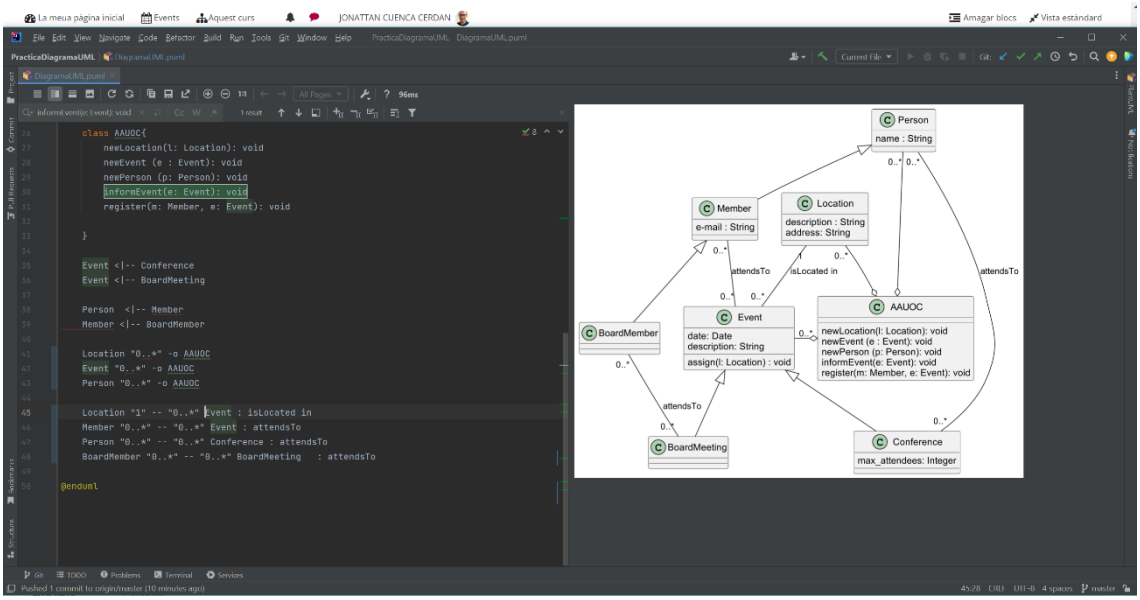
Una vez añadida la información, realizamos commit and push a nuestro repositorio remoto de GitHub.



4º Incluir la cardinalidad y navegabilidad de las relaciones entre clases.

Las navegabilidades son todas bidireccionales ya que no se nos ha comunicado ningún tipo de relación y/o acceso a la información de una clase a otra.

Para añadir la cardinalidad entre las clases hay que poner entre comillas “ “ dicha cardinalidad. El resultado del diagrama UML del enunciado dado sería el siguiente:



Realizamos commit and push del proyecto y exportamos la memoria de este proyecto en PDF para poder incluirla en el repositorio.