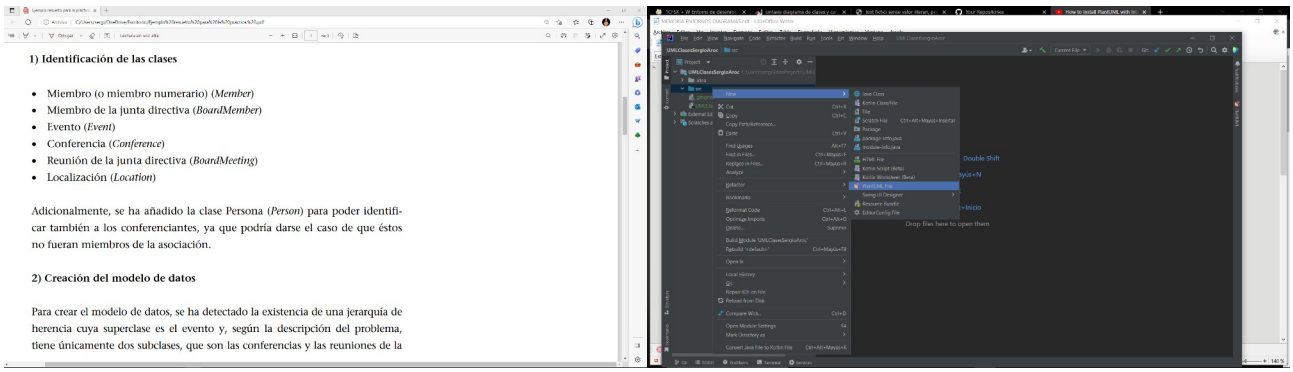
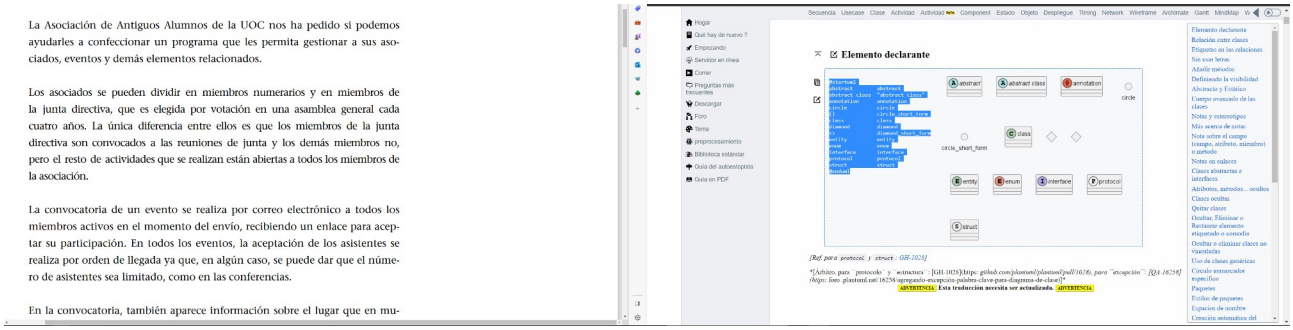
Sergio Aroca
Enlace a repositorio: https://github.com/sergikarx/UMLClases-SergioAroca.git

MEMORIA DE DIAGRAMA DE CLASES UML

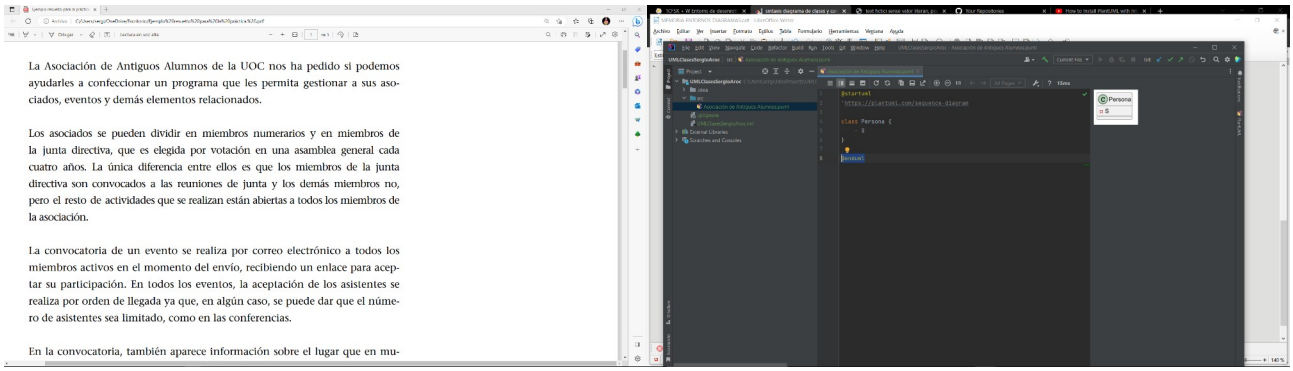
Después de haber descargado el plugin y haberlo instalado crearemos el repositorio en github y crearemos nuestro proyecto en intelij, como vemos aquí simplemente le daremos a nuevo archivo y seleccionaremos el formato de diagrama de clases UML.



Como vemos después de crear el diagrama UML se nos ha generado un código, dentro de los apartados @startuml y @enduml será donde nuestro diagrama se generará mediante el código que generaremos siguiendo las directrices de la sintaxis del diagrama de clases otorgado por la página que se indica en la práctica.



A continuación procederé a crear las clases mediante la sintaxis mencionada anteriormente, esto es un ejemplo de como crear una clase:



Como podemos apreciar a la derecha se nos muestra de forma gráfica el diagrama que estamos generando con la sintaxis, como vemos hemos creado la primera de las clases que tendrá nuestro

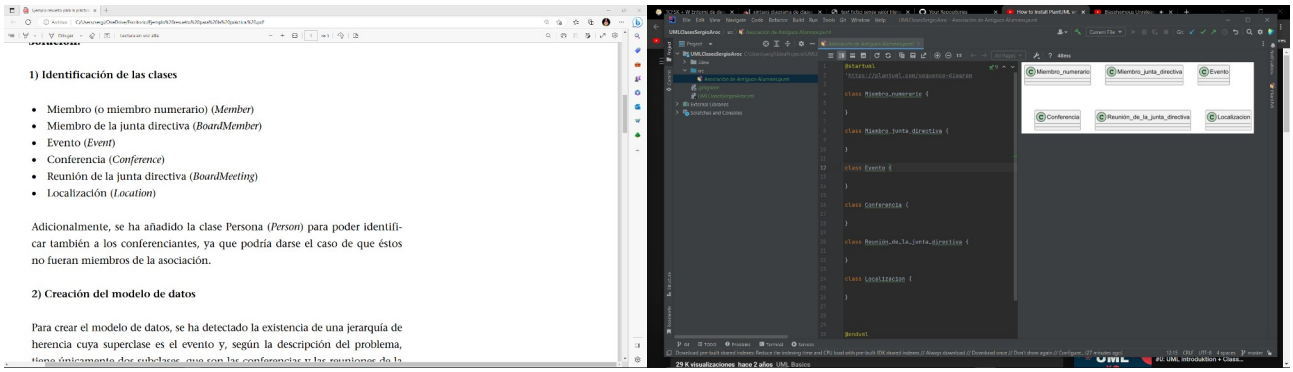
diagrama, a continuación crearé las clases, seguiré un orden por puntos y realizaré un commit en Git al terminar cada uno de los puntos.

1) Identificación de las clases.

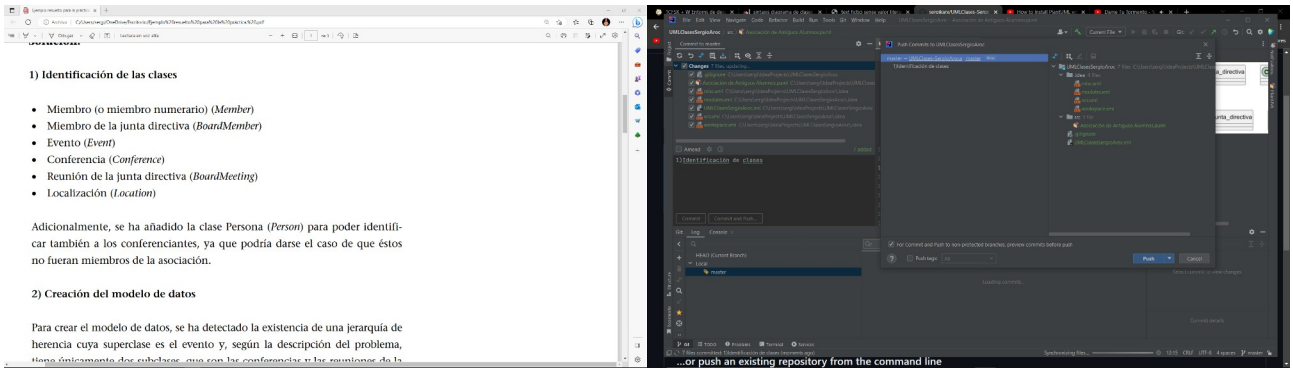
En primer lugar como hemos dicho crearemos las clases, las clases necesarias para el ejercicio siguiendo el orden del primer punto de momento son las siguientes:

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

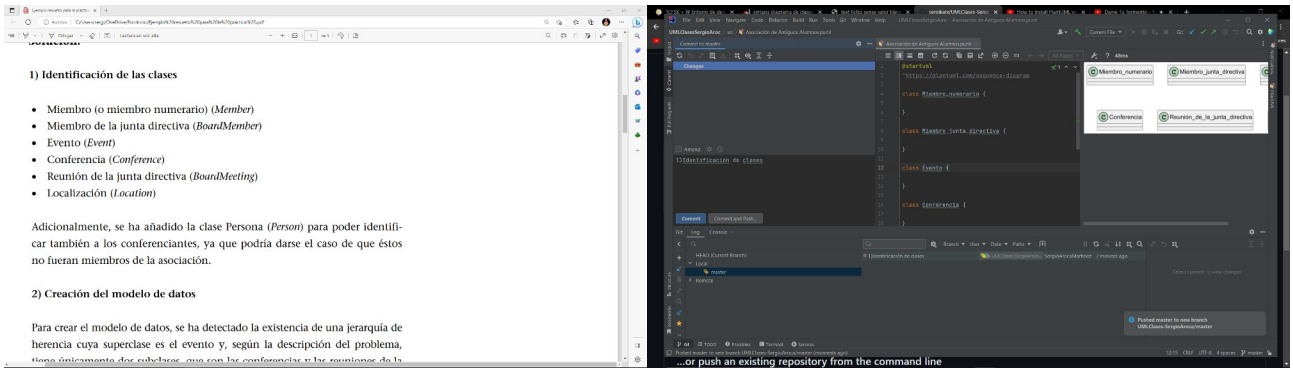
Nota: Adicionalmente, se ha añadido la clase Persona (Person) para poder identificar también a los conferenciantes, ya que podría darse el caso de que éstos no fueran miembros de la asociación.



Como vemos ya hemos creado las clases necesarias para el primer punto así que procederé a realizar un commit y push en Git.



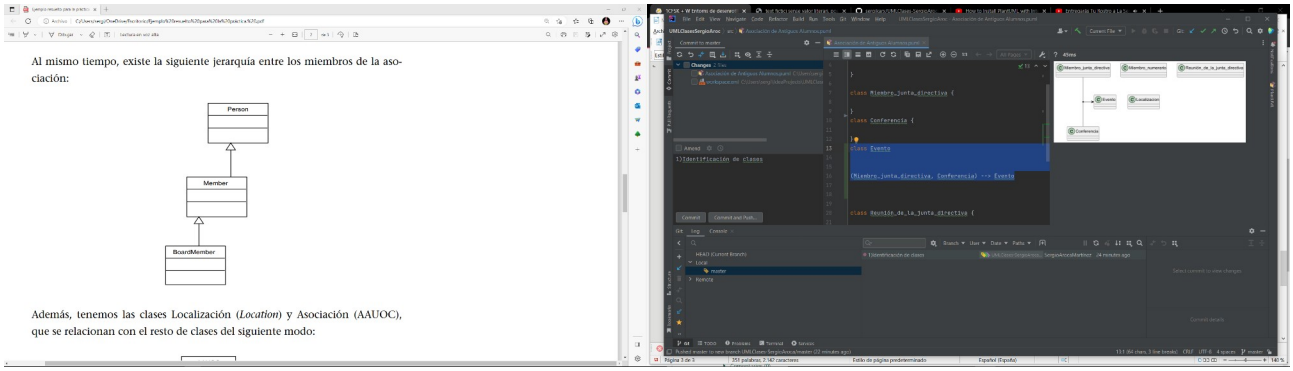
Como vemos el push se ha realizado con éxito.



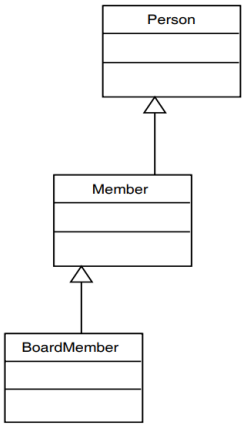
2) Creación del modelo de datos.

A continuación procederé a establecer la jerarquía establecida en el segundo punto del pdf.

Se ha detectado la existencia de una jerarquía de herencia cuya superclase es el evento y, según la descripción del problema, tiene únicamente dos subclases, que son las conferencias y las reuniones de la junta directiva.
Lo haremos de la siguiente forma:

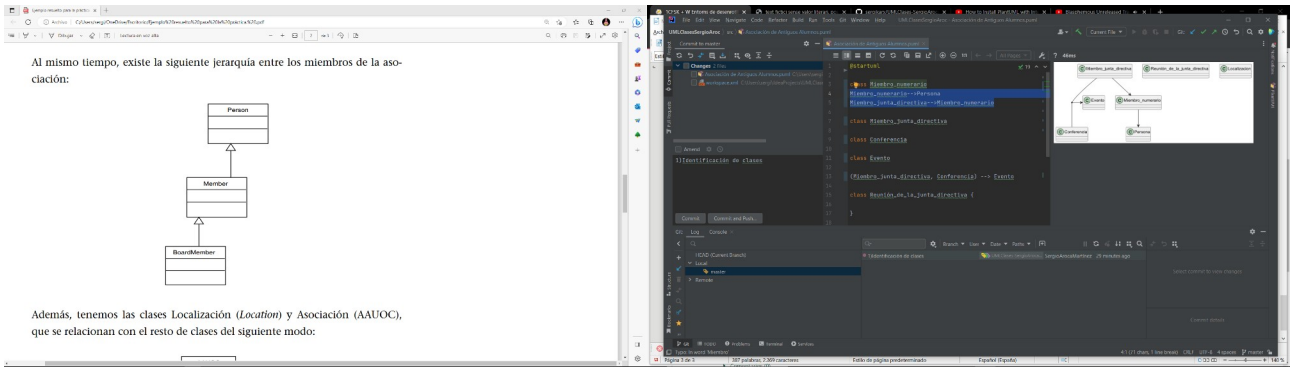


A continuación se nos indica que existe la siguiente jerarquía:

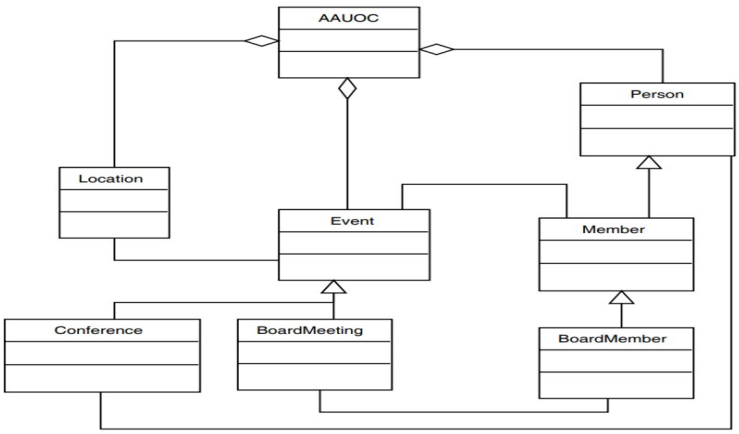


En primer lugar crearemos una clase Persona, y a continuación crearemos la jerarquía mencionada, la crearemos de la siguiente forma:

class NOMBRE (De esta forma se crean las clases)



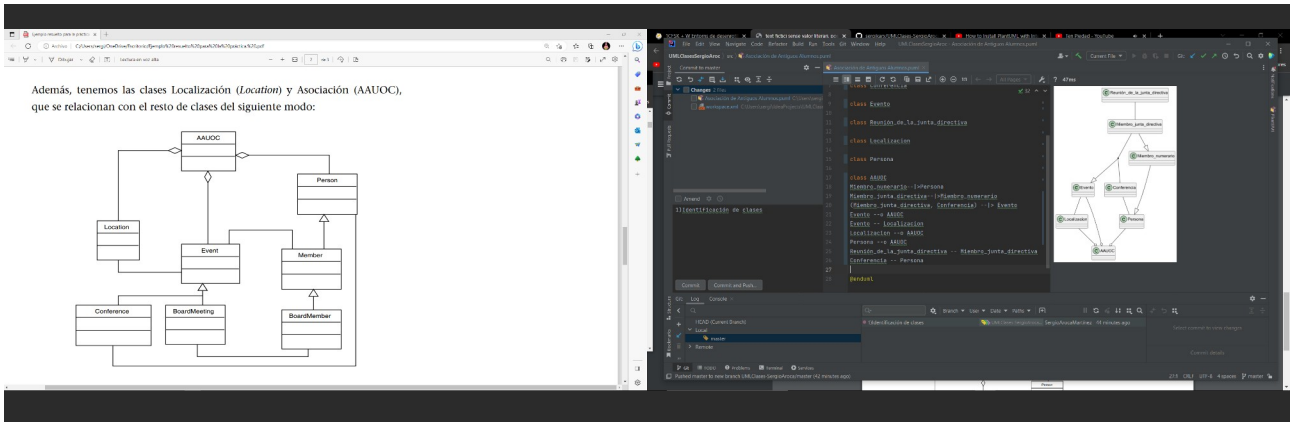
Además, tenemos las clases Localización (Location) y Asociación (AAUOC), que se relacionan con el resto de clases del siguiente modo:



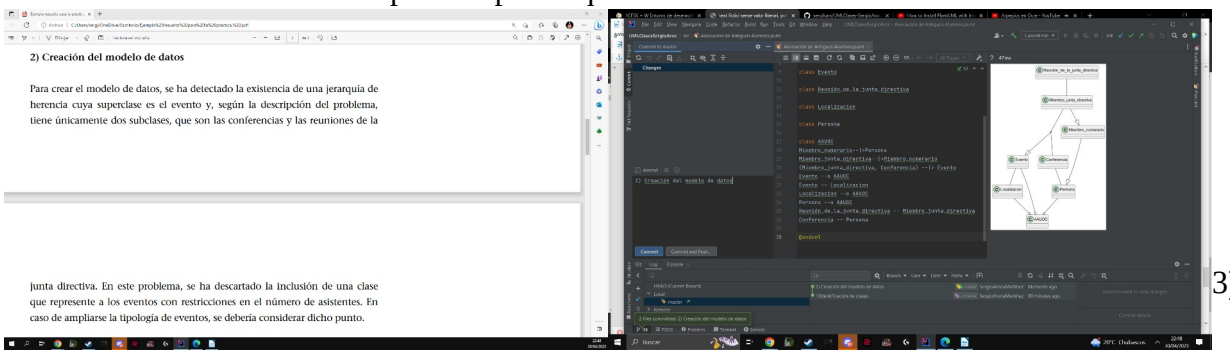
Como vemos hay diferentes maneras de relacionar las clases, estas formas indican distinto tipo de relaciones por lo que se crearan de distinta forma y con distinta finalidad.

Type	Symbol	Drawing
Extension	`<|--`	
Composition	`*--`	
Aggregation	`o--`	

```
Miembro_numerario--|>Persona
Miembro_junta_directiva--|>Miembro_numerario
(Miembro_junta_directiva, Conferencia) --|> Evento
Evento --o AAUOC
Evento -- Localizacion
Localizacion --o AAUOC
Persona --o AAUOC
Reunión_de_la_junta_directiva -- Miembro_junta_directiva
Conferencia -- Persona
```



Hecho esto hemos finalizado el punto 2 por lo que será momento de realizar un commit.

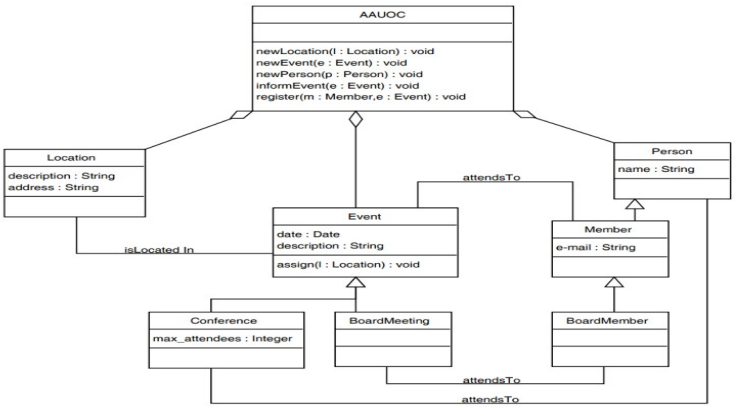


3)

Inclusión de atributos y métodos.

A continuación será el momento de establecer los atributos y los métodos dentro de las clases creadas y ordenadas anteriormente, para esto será necesario establecer el código de los atributos y sus métodos.

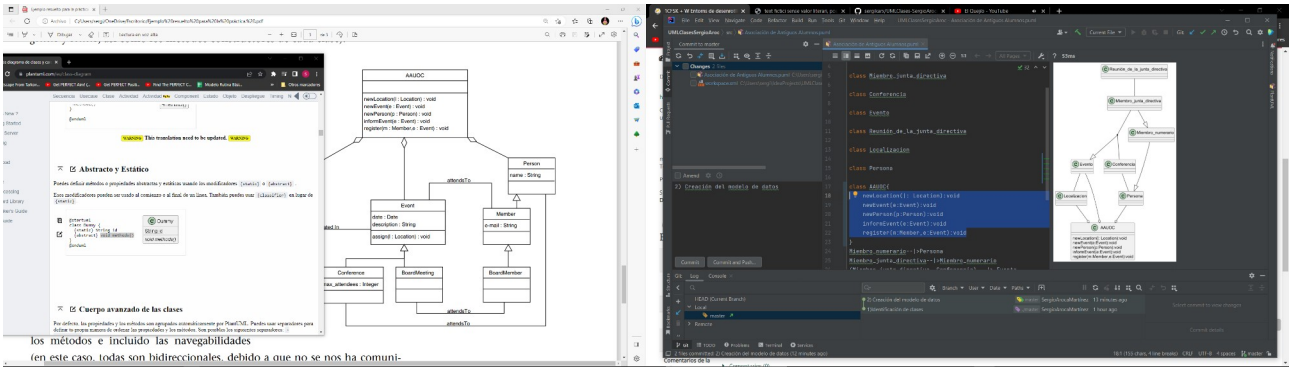
Como vemos en el pdf se nos manda que las clases contengan los siguientes atributos y métodos:



Para crear los métodos debemos utilizar la siguiente sintaxis:
metodoEjemplo ()

En el siguiente caso yo les he añadido el void como en el pdf, los métodos que he creado en AAUOC son los siguientes:

```
+void newLocation(|: Location)
+void newEvent(e:Event)
+void newPerson(p:Person)
+void informEvent(e:Event)
+void register(m:Member,e:Event)
```



A continuación le agregaré nombre a las relaciones añadiendo : y el nombre de la relación

Ejemplo:

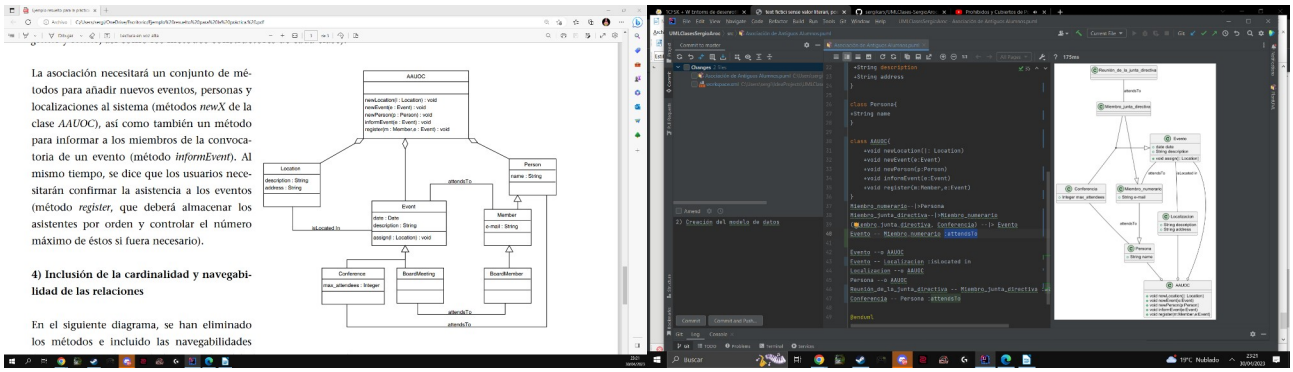
```
:attendsTo
```

También agregaré los atributos y métodos de las demás clases para añadir un atributo sería tal que así:

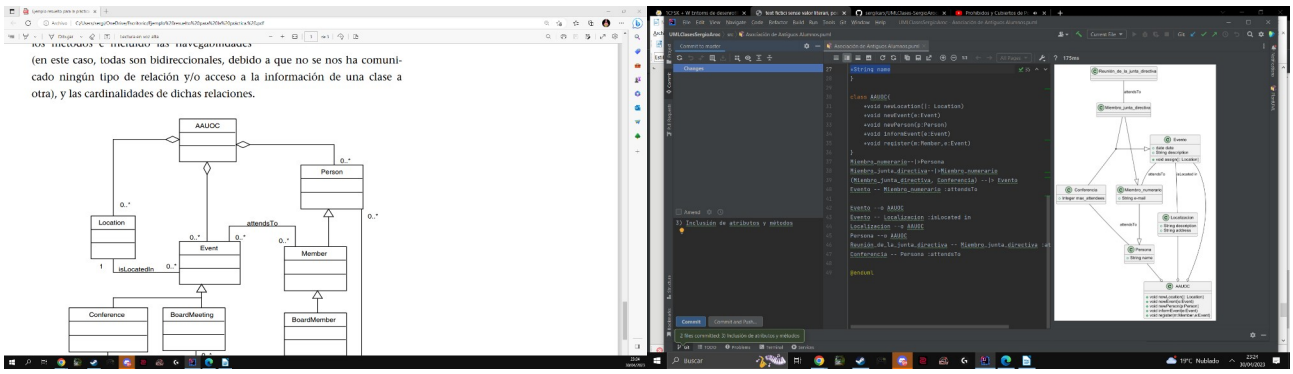
```
+String name
```

Como vemos antes de nada se indica si va a ser privado restrictivo paquete privado o publico, también se puede dejar esto en blanco, acto seguido el tipo de primitivo y el nombre de este.

Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public



Como ya he terminado con el punto 3 voy a proceder a realizar un commit.



4) Inclusión de la cardinalidad y navegabilidad de las relaciones .

A continuación como el ejercicio indica eliminaremos los atributos y dejaremos solo las relaciones, indicaremos la cardinalidad como en el ejercicio indicado, para indicar la cardinalidad yo he utilizado la siguiente sintaxis:

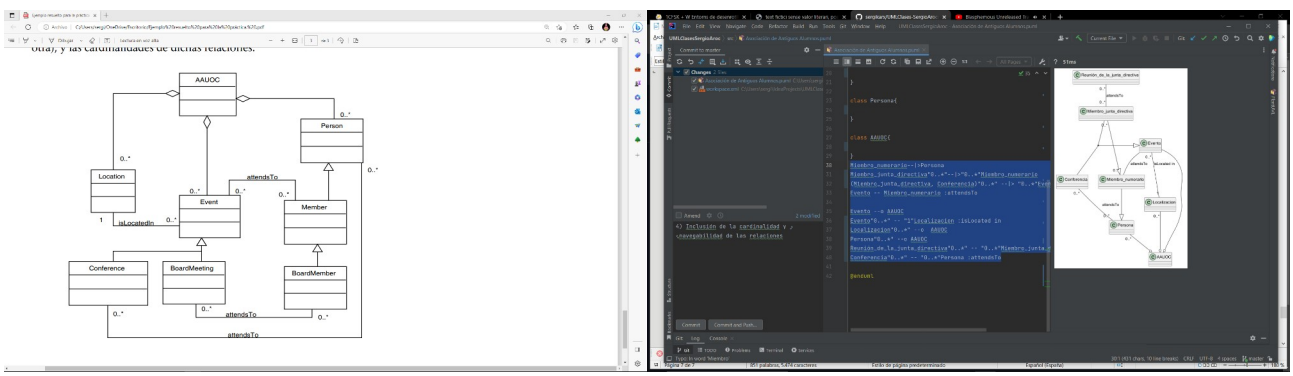
```
Persona"0..*" --o AAUOC
```

Persona es una clase, como quiero indicar que la cardinalidad sea de cero a muchos desde persona voy a añadir comillas simples y entre estas le indicaré la cardinalidad que estará junto a la clase a la que esta se asociará, en este caso Persona tendrá de 0 a muchas en la relación con la AAUOC.

Así quedarían mis relaciones:

```
Miembro_numerario--|>Persona
Miembro_junta_directiva"0..*"--|>"0..*"Miembro_numerario
(Miembro_junta_directiva, Conferencia)"0..*" --|> "0..*"Evento
Evento -- Miembro_numerario :attendsTo

Evento --o AAUOC
Evento"0..*" -- "1"Localizacion :isLocated in
Localizacion"0..*" --o  AAUOC
Persona"0..*" --o AAUOC
Reunión_de_la_junta_directiva"0..*" -- "0..*"Miembro_junta_directiva :attendsTo
Conferencia"0..*" -- "0..*"Persona :attendsTo
```



Para finalizar realizaré mi último commit y un push a mi repositorio.