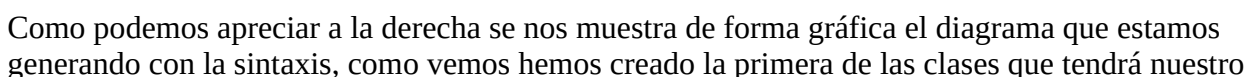
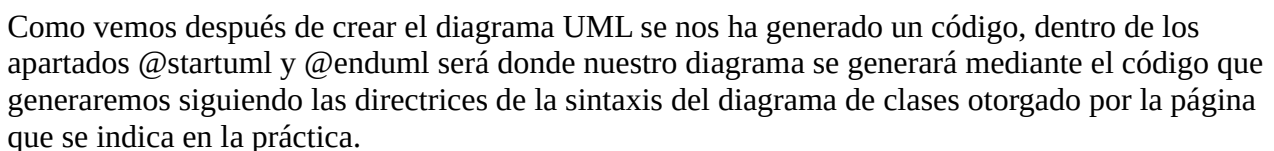


Enlace a repositorio: <https://github.com/sergikarx/UMLClases-SergioAroca.git>

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.



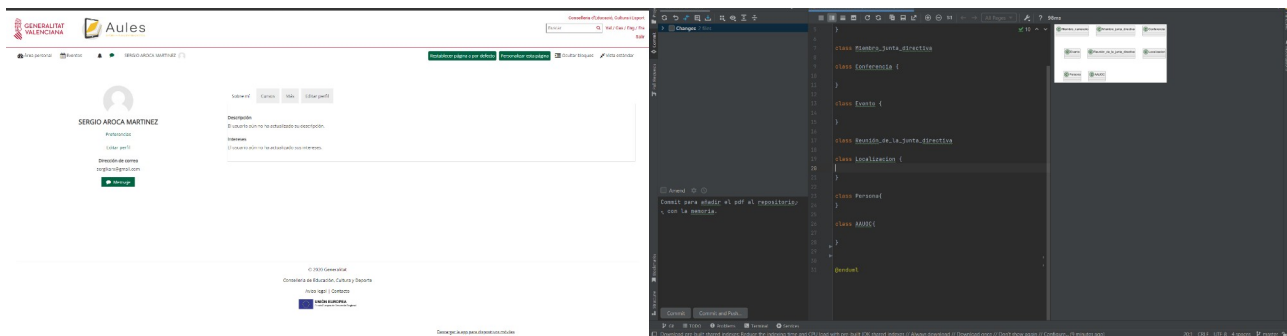
diagrama, a continuación crearé las clases, seguiré un orden por puntos y realizaré un commit en Git al terminar cada uno de los puntos.

## 1) Identificación de las clases.

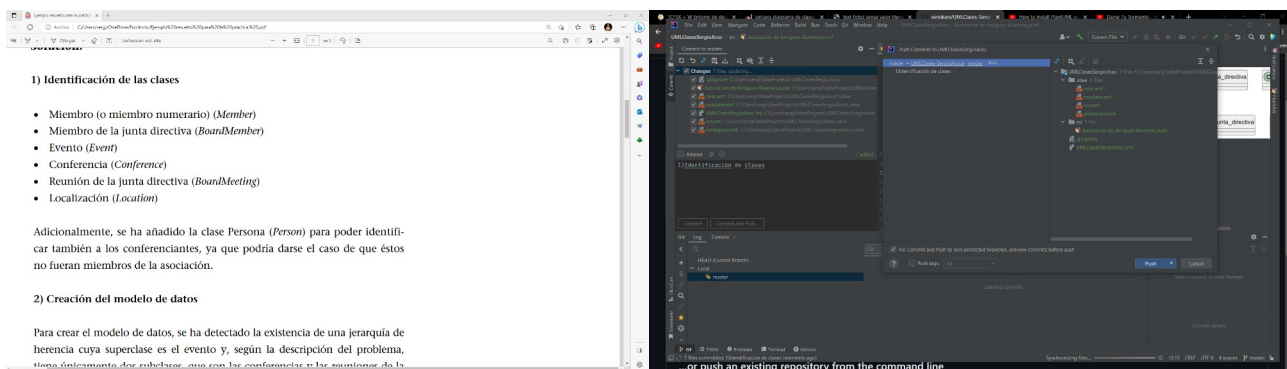
En primer lugar como hemos dicho crearemos las clases, las clases necesarias para el ejercicio siguiendo el orden del primer punto de momento son las siguientes:

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

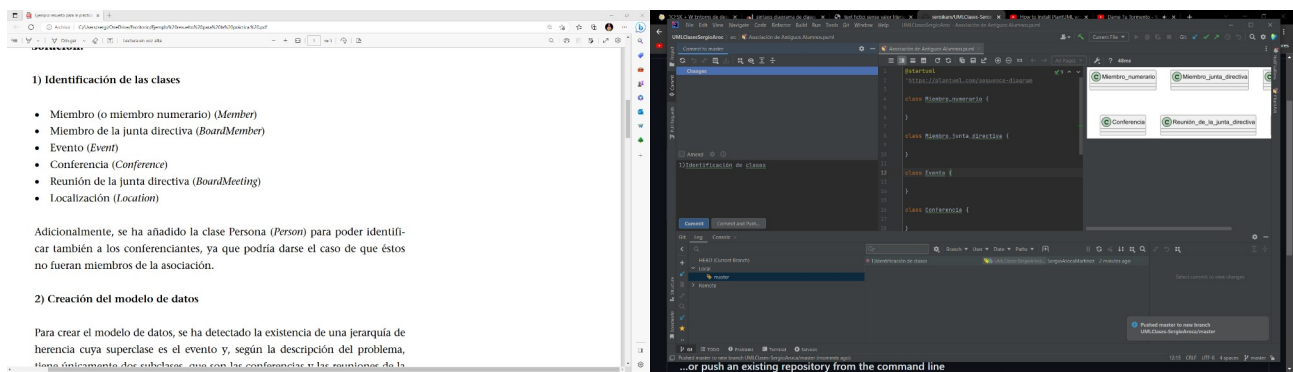
**Nota:** Adicionalmente, se ha añadido la clase Persona (Person) para poder identificar también a los conferenciantes, ya que podría darse el caso de que éstos no fueran miembros de la asociación.



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.

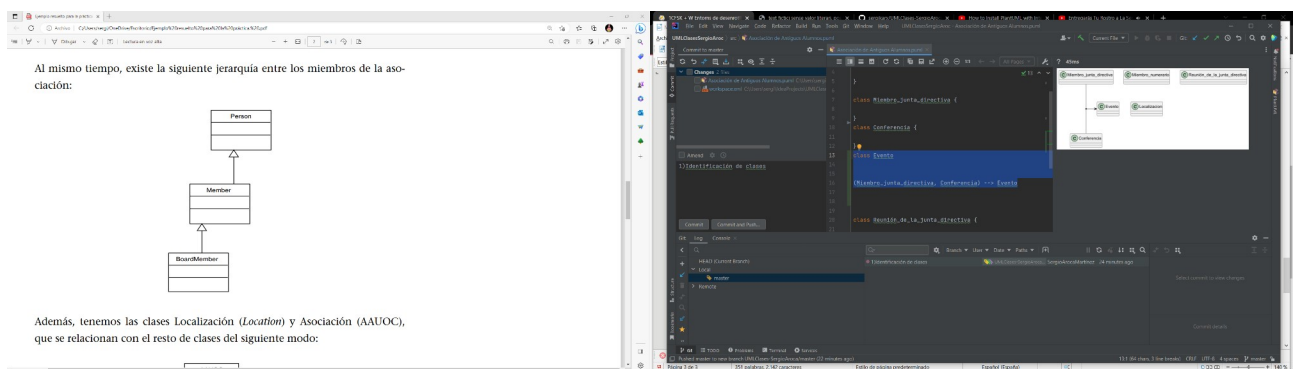


## 2) Creación del modelo de datos.

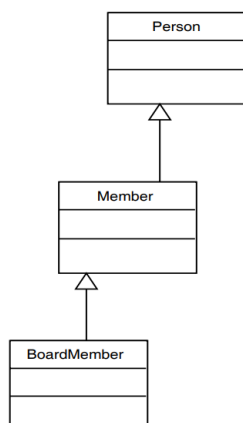
A continuación procederé a establecer la jerarquía establecida en el segundo punto del pdf.

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)

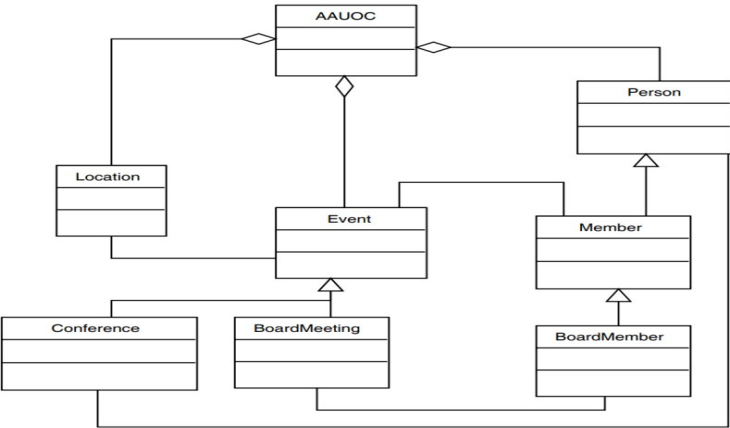
Al mismo tiempo, existe la siguiente jerarquía entre los miembros de la asociación:

```
classDiagram
    Person <|-- Member
    Member <|-- BoardMember
```

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

```
classDiagram
    class AAUOC {
        +Localizacion
        +Evento
    }
    class Location {
        +Evento
    }
    class Event {
        +Conferencia
        +BoardMeeting
    }
    class Person {
        +Miembro_numerario
        +Miembro_junta_directiva
    }
    class Member {
        +BoardMember
    }
    AAUOC *-- Location
    AAUOC *-- Event
    Location *-- Event
    Event *-- Conferencia
    Event *-- BoardMeeting
    Person <|-- Member
    Member <|-- BoardMember
```

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
```

```
classDiagram
    class AAUOC {
        +Localizacion
        +Evento
    }
    class Location {
        +Evento
    }
    class Event {
        +Conferencia
        +BoardMeeting
    }
    class Person {
        +Miembro_numerario
        +Miembro_junta_directiva
    }
    class Member {
        +BoardMember
    }
    AAUOC *-- Location
    AAUOC *-- Event
    Location *-- Event
    Event *-- Conferencia
    Event *-- BoardMeeting
    Person <|-- Member
    Member <|-- BoardMember
```

[illegible]

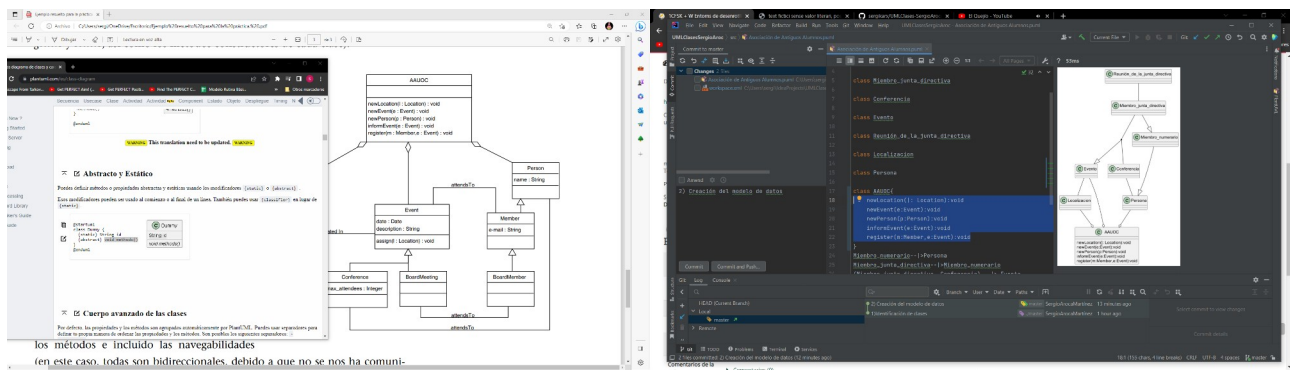
A continuación será el momento de establecer los atributos y los métodos dentro de las clases creadas y ordenadas anteriormente, para esto será necesario establecer el código de los atributos y sus métodos.

```

classDiagram
    class AAUOC {
        newLocation(l : Location) : void
        newEvent(e : Event) : void
        newPerson(p : Person) : void
        informEvent(e : Event) : void
        register(m : Member, e : Event) : void
    }
    class Location {
        description : String
        address : String
    }
    class Event {
        date : Date
        description : String
        assign(f : Location) : void
    }
    class Person {
        name : String
    }
    class Member {
        e-mail : String
    }
    class Conference {
        max_attendees : Integer
    }
    class BoardMeeting {
    }
    class BoardMember {
    }

    AAUOC --> Location
    AAUOC --> Event
    AAUOC --> Person
    Location --> Event : isLocatedIn
    Event --> Member : attendsTo
    Event --> BoardMeeting : attendsTo
    Person <|-- Member
    Member <|-- BoardMember
  
```

```
+void newLocation(l: 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:

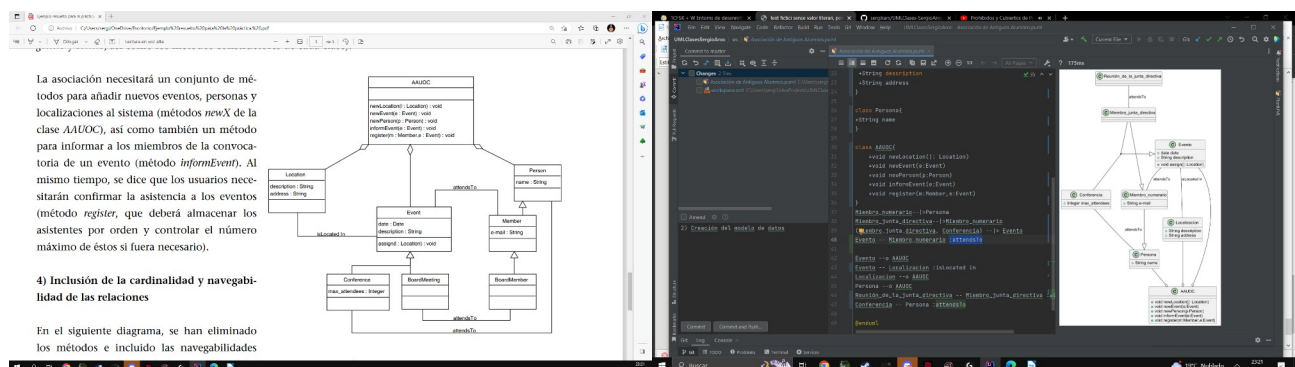
:attendsTo

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

+String name

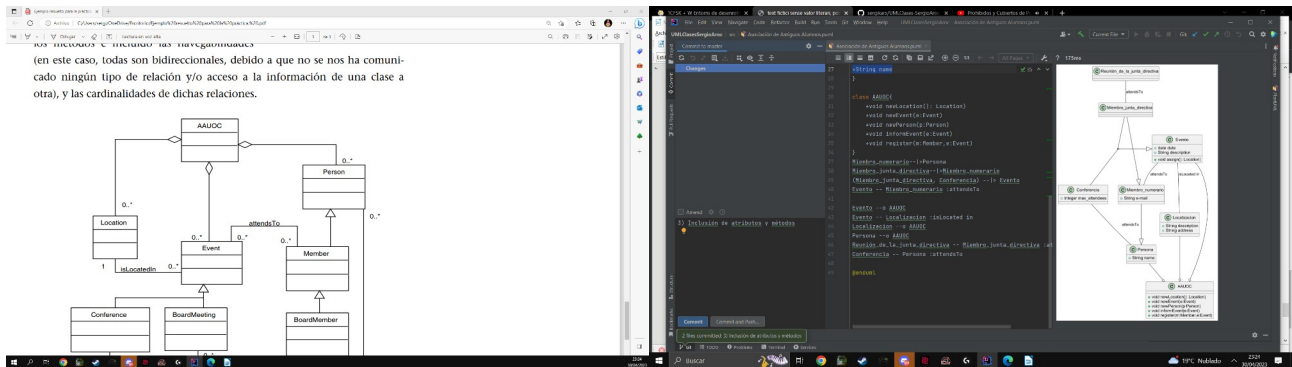
Como vemos antes de nada se indica si va a ser privado restrictivo paquete privado o publico, también se puede dejar esto en blanco, acto seguido el tipo de primitivo y el nombre de este.

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

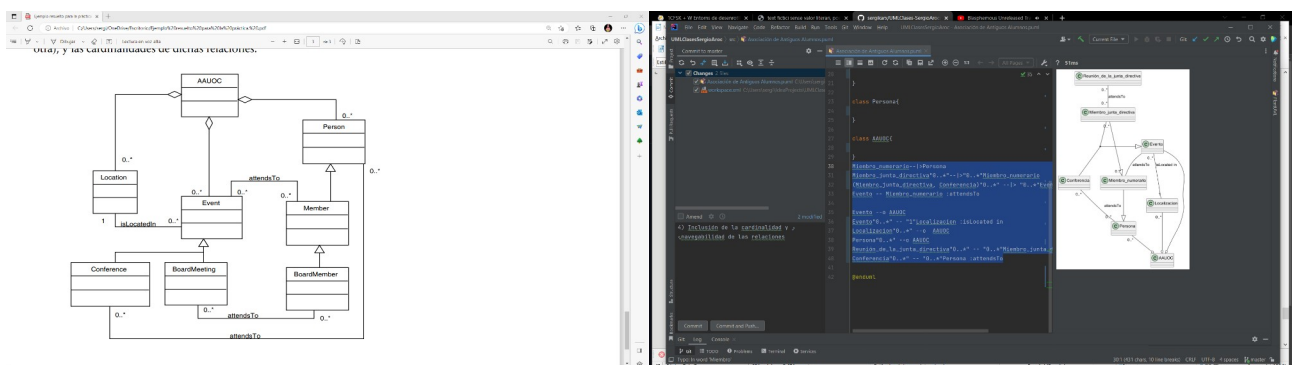
Evento"0..\*" -- "1"Localizacion :isLocated in

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.

