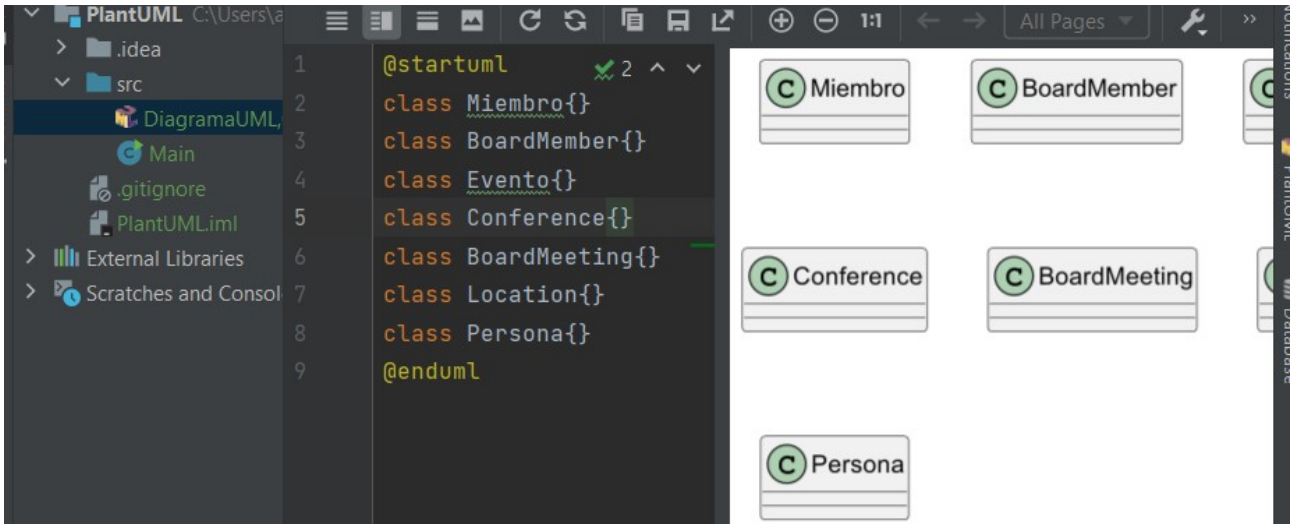
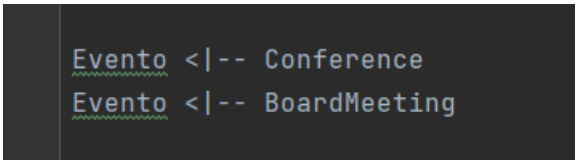
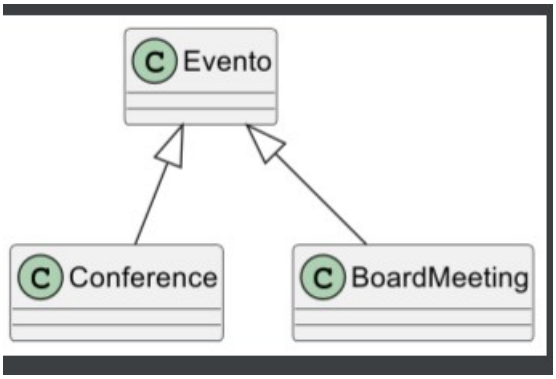
Vamos a documentar la creación del diagrama que nos pide la Asociación de Antiguos Alumnos.

1º Abrimos el IntelliJ, creamos un nuevo PlantUml y vamos creando las distintas clases que nos dice el documento.



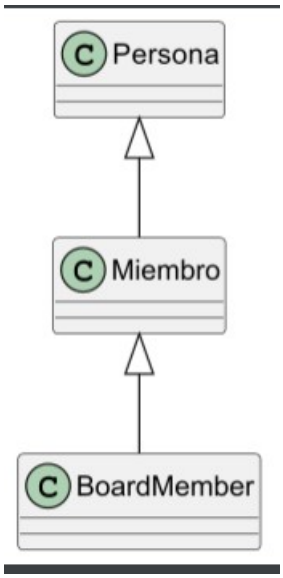

2º Vamos a crear el modelo de datos, para ello, usaremos la jerarquía de herencia cuya superclase es ‘EVENTO’ y sus “hijas” son ‘CONFERENCE’ y ‘BOARDMEETING’.



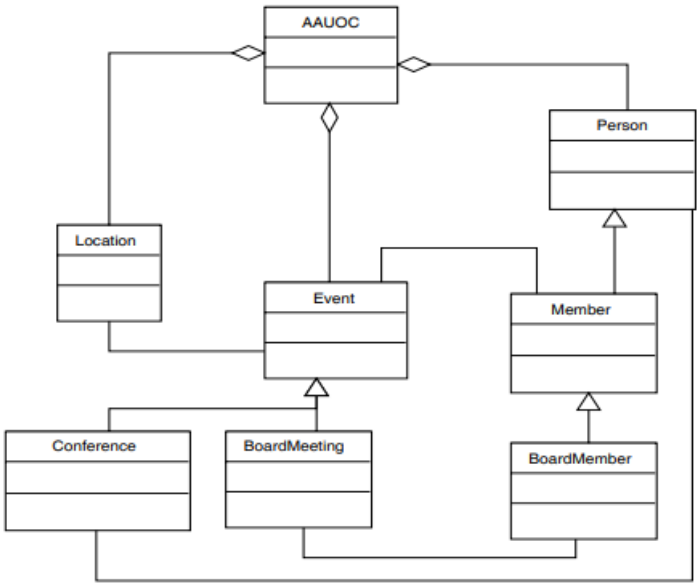



Al mismo tiempo, existe la siguiente jerarquía entre los miembros de la asociación.

```
Persona <|-- Miembro
Miembro <|-- BoardMember
@enduml
```



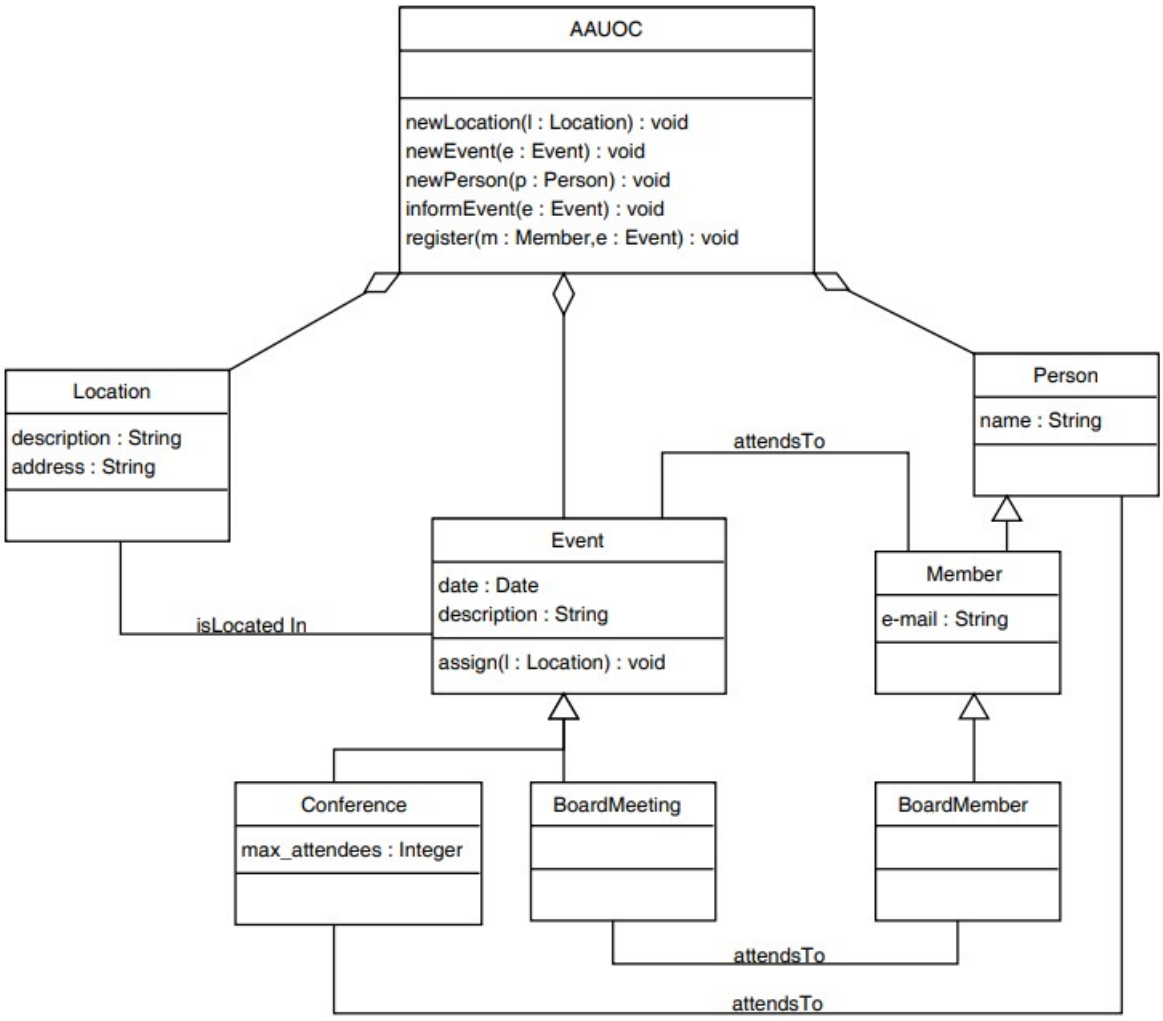

Además, tenemos las clases Localización (Location) y Asociación (AAUOC), que se relacionan con el resto de clases del siguiente modo:




Como vemos, se ha creado la nueva clase AAUOC y además, ya no sólo hay jerarquías sino que también hay asociación entre clases, que representaremos con ‘o--’ y ‘--’. Ambas asociaciones, son simples y pueden acceder a los atributos y métodos de las clases relacionadas, con la diferencia de que la segunda no implica una dependencia fuerte entre ellas.

```
AAUOC o-- Location
AAUOC o-- Persona
AAUOC o-- Evento
Evento -- Location
Evento -- Miembro
BoardMeeting -- BoardMember
Persona -- Conference
```

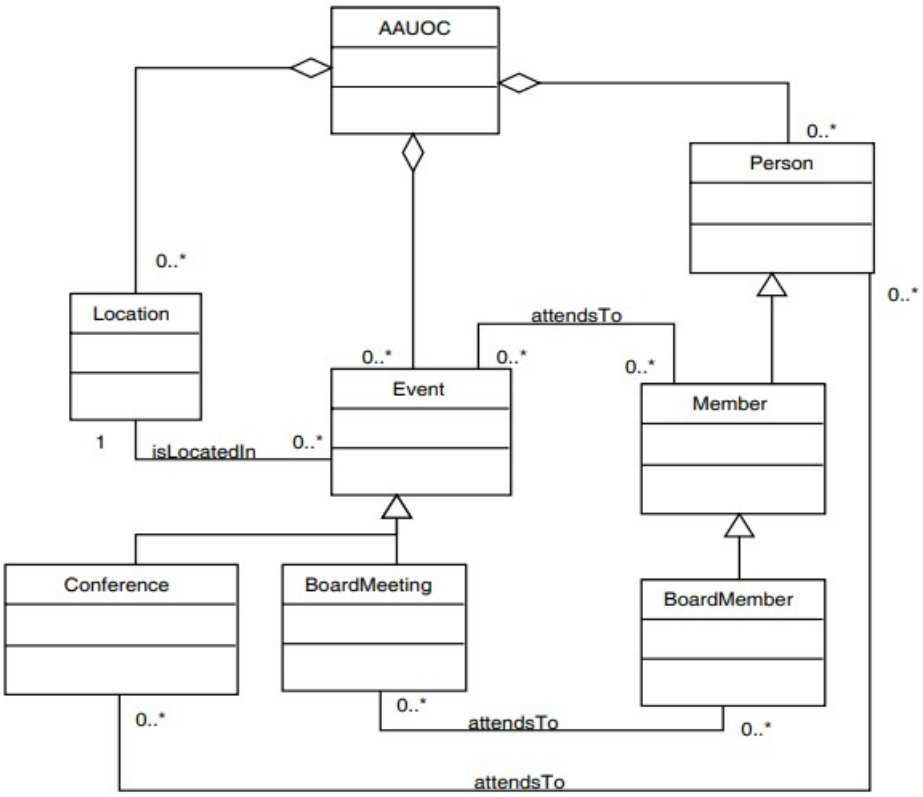
Una vez tenemos la estructura del modelo de datos, procederemos a completar las clases con sus atributos y métodos más relevantes.
La asociación necesitará un conjunto de métodos para añadir nuevos eventos, personas y localizaciones al sistema (métodos newX de la clase AAUOC), así como también un método para informar a los miembros de la convocatoria de un evento (método 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).



```
class Miembro{
e-mail: String
}

class BoardMember{}

class Evento{
date: Date
description: String
assing (l: Location): void
}

class Conference{
max_attendees: Integer
}

class BoardMeeting{}

class Location{
description: String
adress: String
}
```

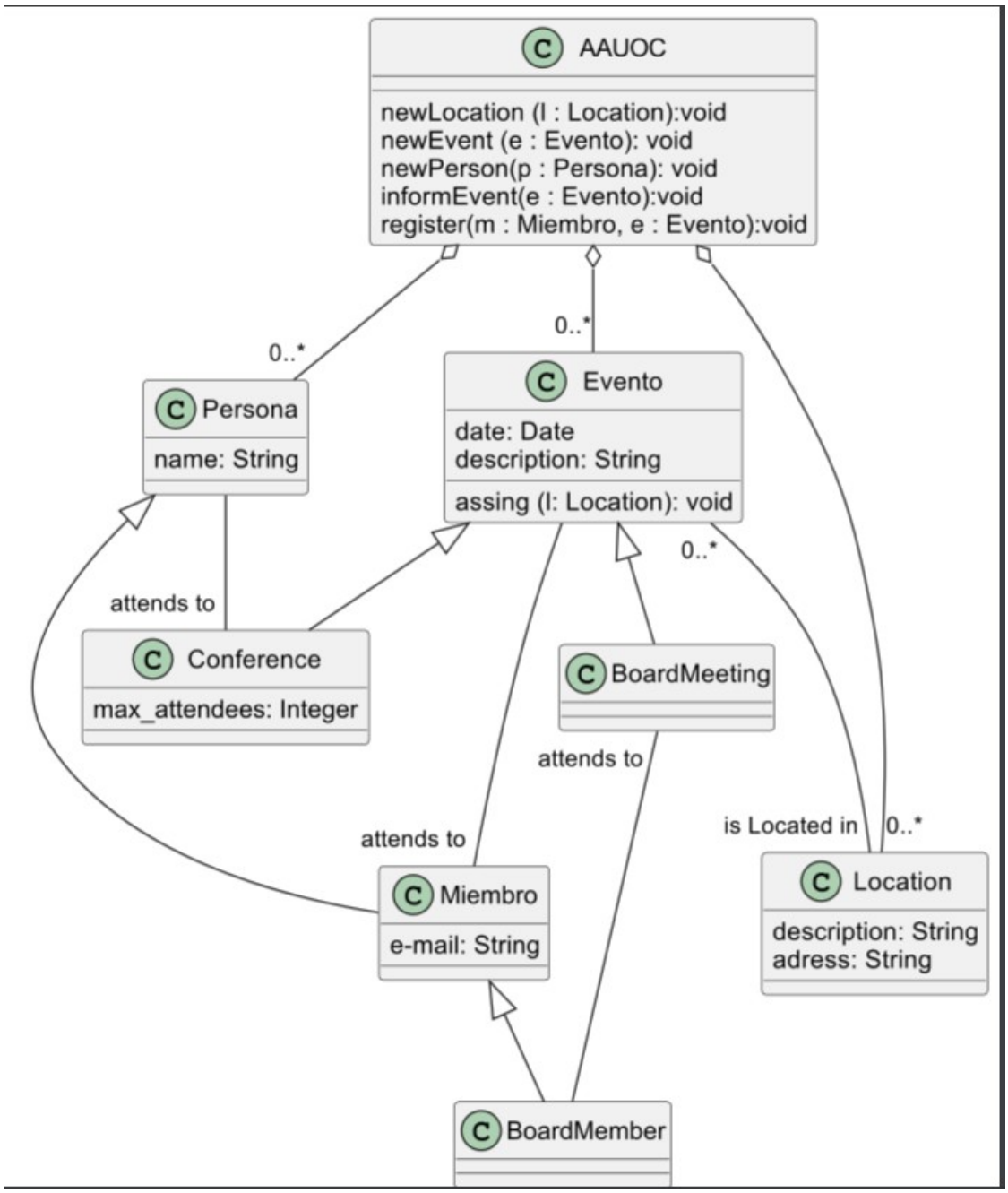
```
class Persona{
name: String
}

class AAUOC{
newLocation (l : Location):void
newEvent (e : Evento): void
newPerson(p : Persona): void
informEvent(e : Evento):void
register(m : Miembro, e : Evento):void
}

Evento <|- Conference
Evento <|- BoardMeeting

Persona <|-- Miembro
Miembro <|-- BoardMember

AAUOC o-- Location
AAUOC o-- Persona
AAUOC o-- Evento
Evento -- "is Located in " Location
Evento "attends to" -- Miembro
BoardMeeting  "attends to"-- BoardMember
Persona -- "attends to" Conference
```

Por último, solo nos queda incluir las cardinalidades entre relaciones. En este ejercicio, como no se nos especifica otra cosa, todas serán bidireccionales.




```
AAUOC o-- "0..*" Location
AAUOC o-- "0..*" Persona
AAUOC o-- "0..*" Evento
Evento "0..* "--- "is Located in" Location
Evento  --- "attends to" Miembro
BoardMeeting  "attends to"-- BoardMember
Persona -- "attends to" Conference
```

Así, las cardinalidades quedarán de la siguiente forma:


AAUOC
newLocation (l : Location):void
newEvent (e : Evento): void
newPerson(p : Persona): void
informEvent(e : Evento):void
register(m : Miembro, e : Evento):void
0..*
0..*
Persona
name: String
Evento
date: Date
description: String
assing (l: Location): void
0..*
attends to
Conference
max_attendees: Integer
BoardMeeting
attends to
attends to
is Located in
0..*
Miembro
e-mail: String
Location
description: String
adress: String
BoardMember