Practica UD 11. Diagramas UML, Clases.

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.

Solución:

1) Identificación de 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)

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.

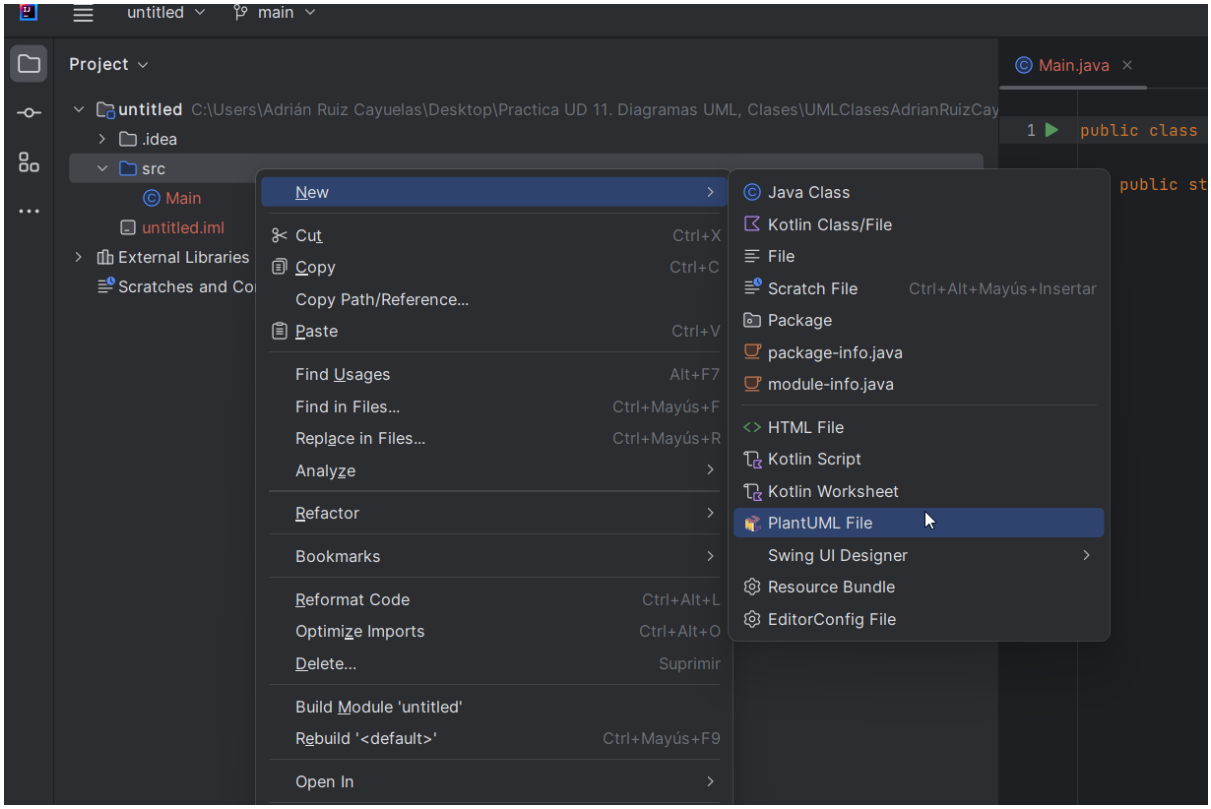
2) Creación del modelo de datos

Para crear el modelo de datos, 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. En este problema, se ha descartado la inclusión de una clase que represente a los eventos con restricciones en el número de asistentes. En caso de ampliarse la tipología de ventos, se debería considerar dicho punto.

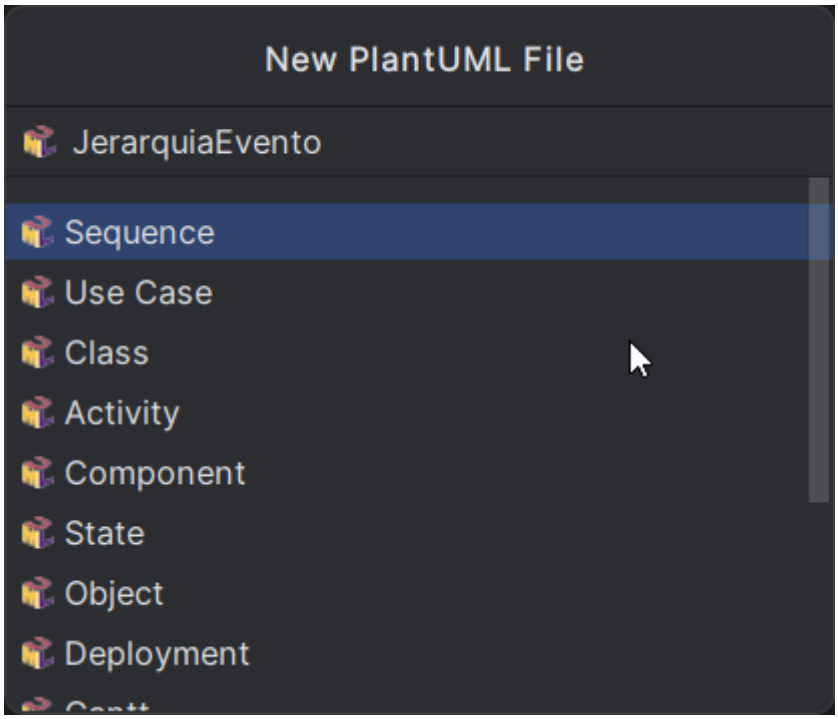
> 💡 Para esta parte de la práctica se dará por hecho que se ha iniciado un proyecto nuevo en intelliJ, y que el plugin de PlantUML ya se encuentra instalado y funcionando.

Ahora vamos a proceder a crear el esquema UML necesario para representar la jerarquía mencionada:

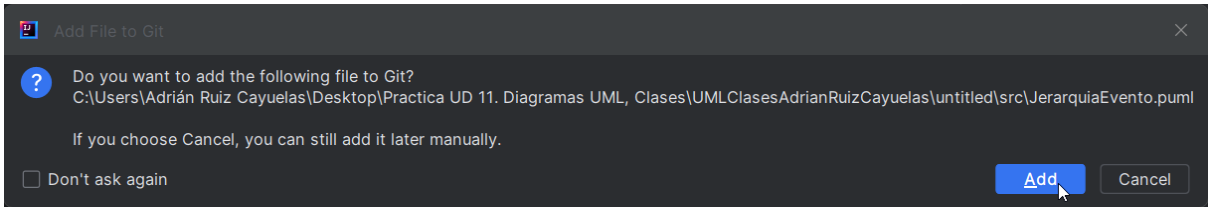
1. Creamos un archivo PlantUML:



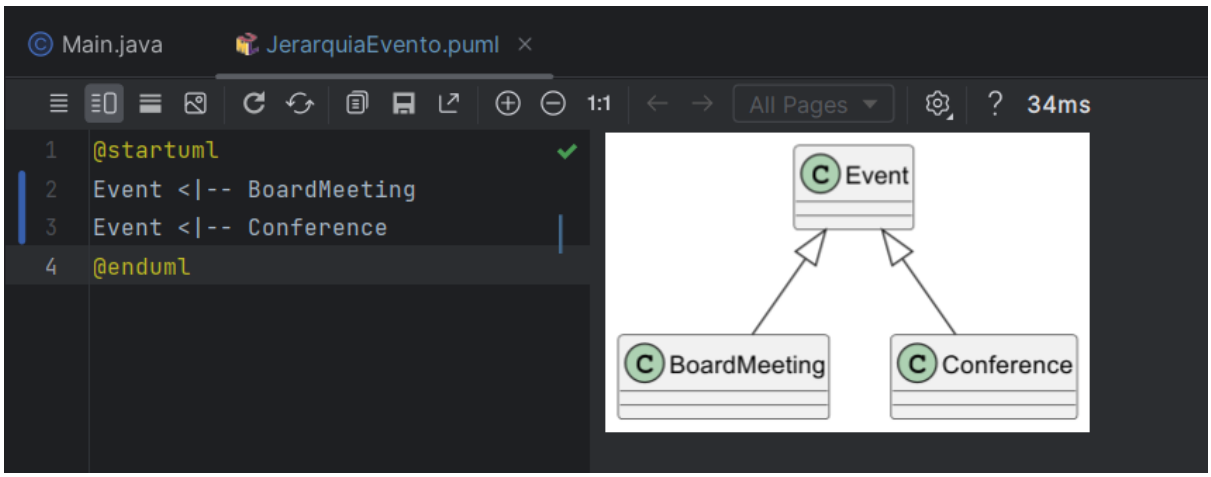
2. Le damos nombre al archivo y seleccionamos la opción “**Sequence”:**



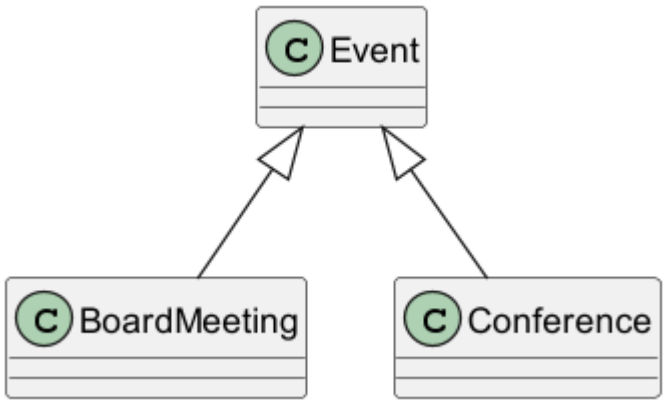
2. 2. Si estamos usando un VCS es probable que el IDE nos pregunte si queremos agregar el archivo al repositorio, le diremos que sí:



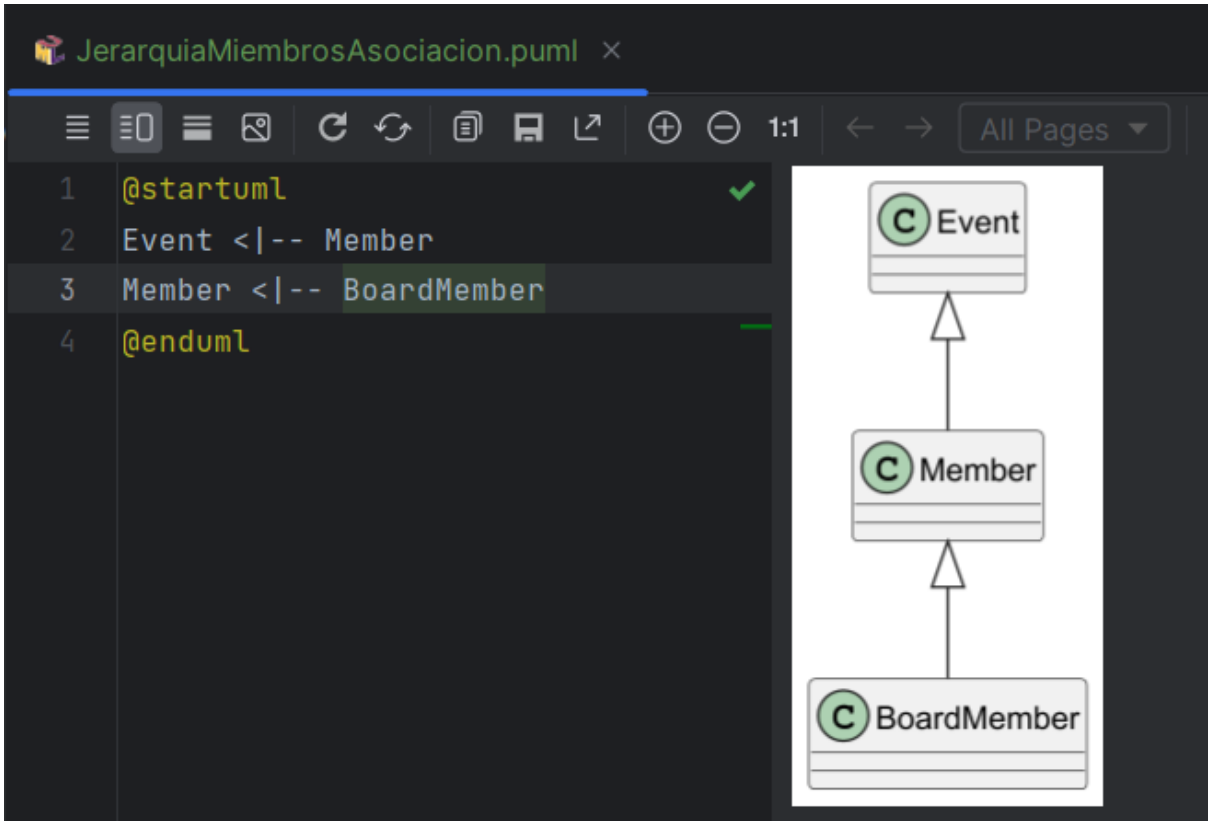
3. Ahora iremos diseñando el esquema mediante codigo. Como las clases están vacias, podemos directamente generar las relaciones ahorrando el paso de definir las clases en el documento UML



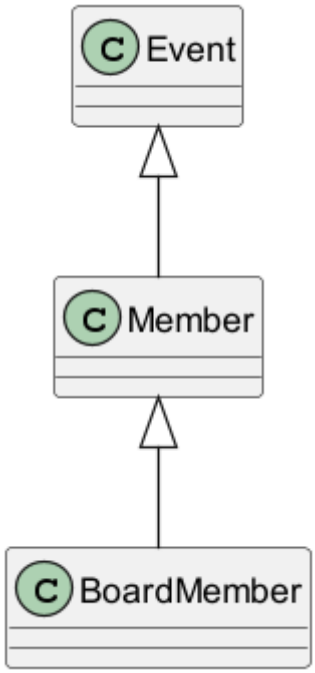
Resultado:



Al mismo tiempo, existe la siguiente jerarquía entre los miembros de la asociación. Para obtener el diagrama replicaremos todos los pasos anteriores exceptuando el ultimo. En este caso llamaremos al fichero **“JerarquiaMiembrosAsociacion”.**

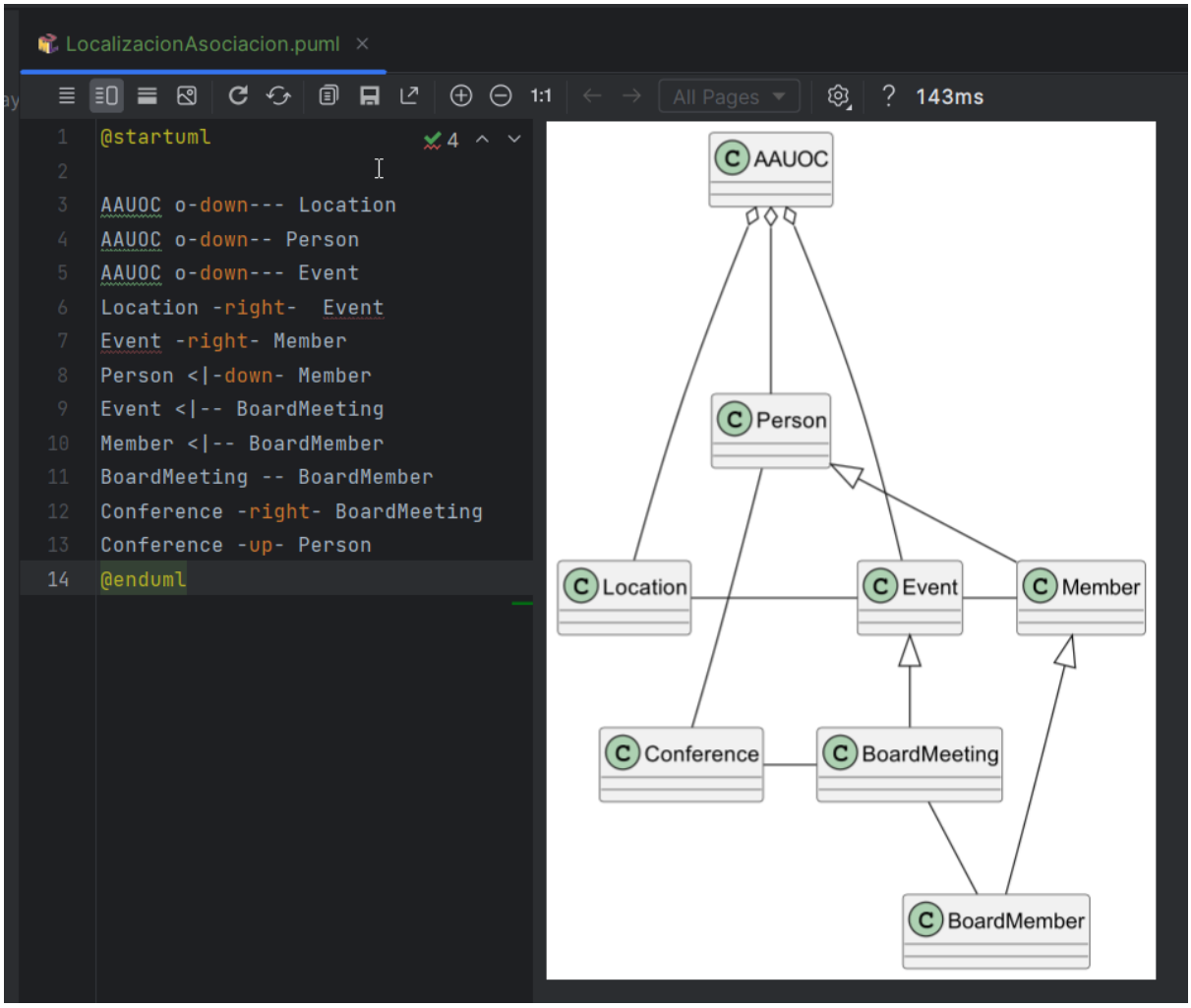


Resultado:

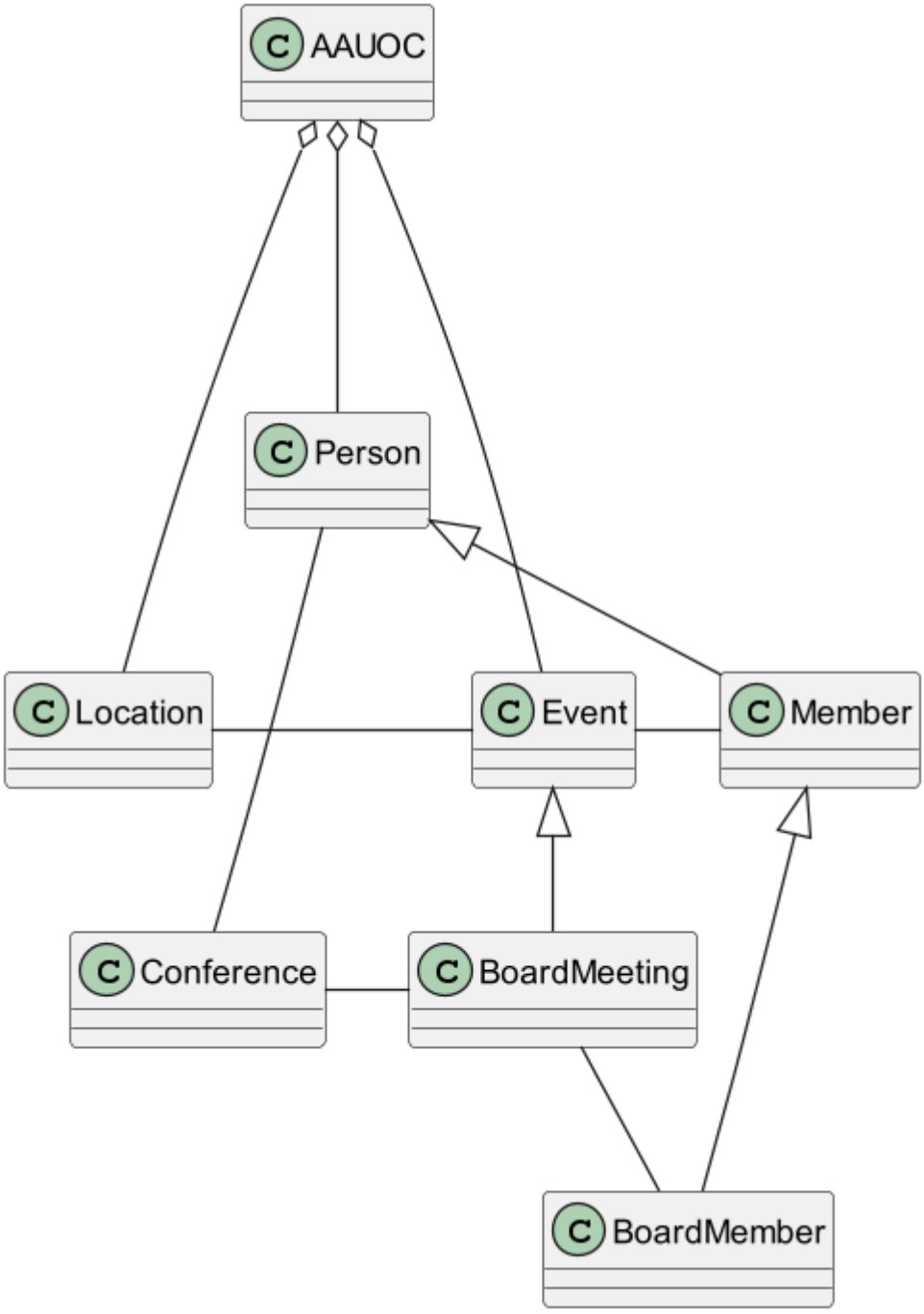


Además, tenemos las clases Localización (Location) y Asociación (AAUOC), que se relacionan con el resto de clases. Nuevamente procederemos a desarrollar

el esquema. Este esquema lo llamaremos **“LocalizacionAsociacion”** y debería de ser similar a este:



Resultado:



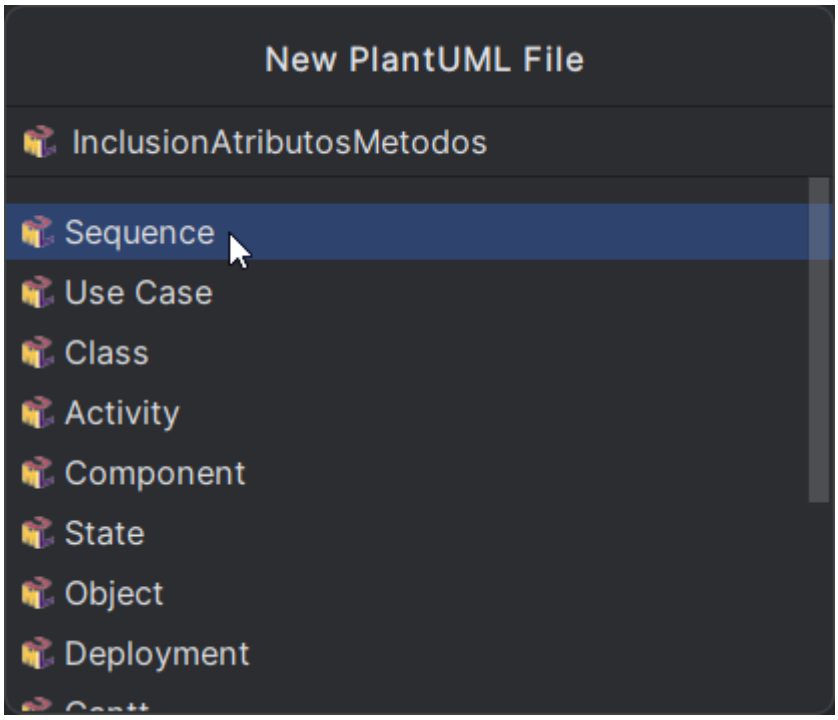
3) Inclusión de atributos y metodos

Una vez creado el modelo de datos, completamos las clases con sus atributos y los métodos más relevantes (quedan fuera de este diagrama los métodos getters y setters, así como los métodos constructores de cada clase).

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).

1. Lo primero que haremos para generar el diagrama será crear un nuevo archivo PlantUML, que yo en este caso he llamado **“InclusionAtributosMetodos”**



2. Puesto que este esquema parte del anterior podremos reutilizar el codigo, y así lo haremos.

```
@startuml
AAUOC o-down--- Location
AAUOC o-down-- Person
AAUOC o-down--- Event
Location -right-  Event
Event -right- Member
Person <|-down- Member
Event <|-- BoardMeeting
Member <|-- BoardMember
BoardMeeting -- BoardMember
Conference -right- BoardMeeting
Conference -up- Person
@enduml
```

3. Ahora deberemos de incluir las clases dentro del codigo, para ello solo deberemos de seguir la siguiente estructura.

```
class "nombreDeLaClase" {
"tipoDeDato" "nombreDeVariable"
"nombreMetodo"("nombreVariableEntrada":"tipoDeDato"):"tipoDeDatoDevuelto"}
```

4. Ahora rellenamos todas las clases de esa manera y el codigo final debería de lucir similar a esto:

```
@startuml

AAUOC o-down--- Location
AAUOC o-down-- Person
AAUOC o-down--- Event
Location -right-  Event
Event -right- Member
Person <|-down- Member
Event <|-- BoardMeeting
Member <|-- BoardMember
BoardMeeting -- BoardMember
Conference -right- BoardMeeting
Conference -up- Person
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(l : Location) : void
}

class Person{
name : String
}

class Member{
e-mail : String
}

class Conference{
max_attendees : Integer
}
```

```
@enduml
```

5. Ahora deberemos marcar las relaciones, para esto usaremos la siguiente estructura:

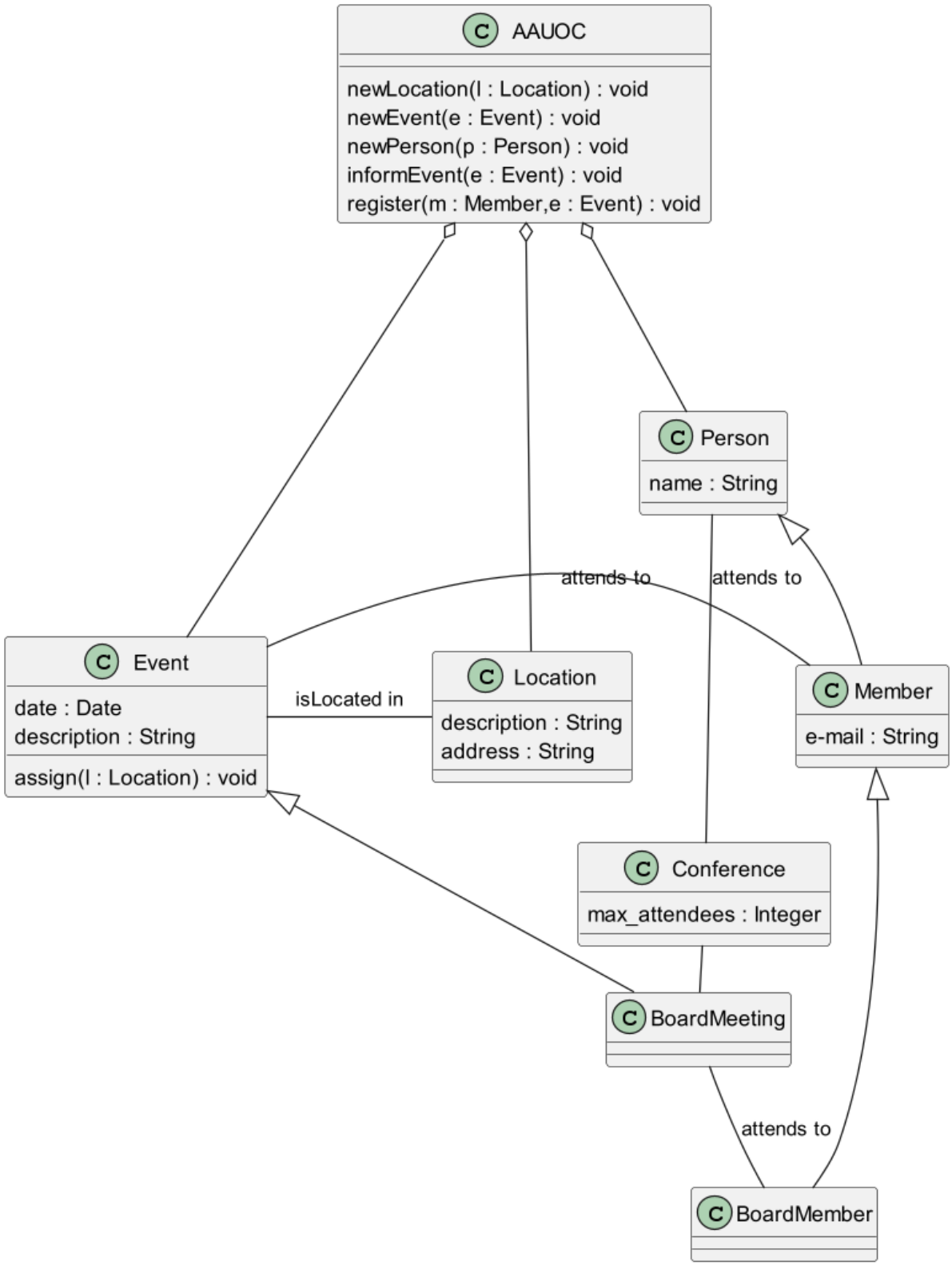
```
"claseA" -- "claseB" : "nombreRelacion"
```

6. Ahora procederemos a marcar todas las relaciones, el codigo final debería de parecerse al siguiente:

```
@startuml

AAUOC o-down--- Location
AAUOC o-down-- Person
AAUOC o-down--- Event
Location -right-  Event : isLocated in
Event -right- Member : attends to
Person <|-down- Member
Event <|-- BoardMeeting
Member <|-- BoardMember
BoardMeeting -- BoardMember : attends to
Conference -right- BoardMeeting
Conference -up- Person : attends to
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(l : Location) : void
}

class Person{
name : String
}

class Member{
e-mail : String
}

class Conference{
```

```
max_attendees : Integer
}

@enduml
```

Resultado:



4) Inclusión de la cardinalidad y navegabilidad de las relaciones

En el siguiente diagrama, se han eliminado los métodos e incluido las navegabilidades
(en este caso, todas son bidireccionales, debido a que no se nos ha comunicado ningún tipo de relación y/o acceso a la información de una clase a otra), y las cardinalidades de dichas relaciones.

1. Para este diagrama reutilizaremos nuevamente el codigo utilizado en el apartado anterior. Por tanto crearemos un nuevo fichero llamado **“CardenalidadNavegabilidad”** e eliminaremos las clases del codigo. Este debería de asemejarse al siguiente:

```
@startuml

AAUOC o-down--- Location
AAUOC o-down-- Person
AAUOC o-down--- Event
Location -right-  Event : isLocated in
Event -right- Member : attends to
Person <|-down- Member
Event <|-- BoardMeeting
Member <|-- BoardMember
BoardMeeting -- BoardMember : attends to
Conference -right- BoardMeeting
Conference -up- Person : attends to

@enduml
```

2. Para esto la inclusión de la cardinalidad se deben usar comillas dobles "" en cada lado de la relación., siguiendo la siguiente estructura:

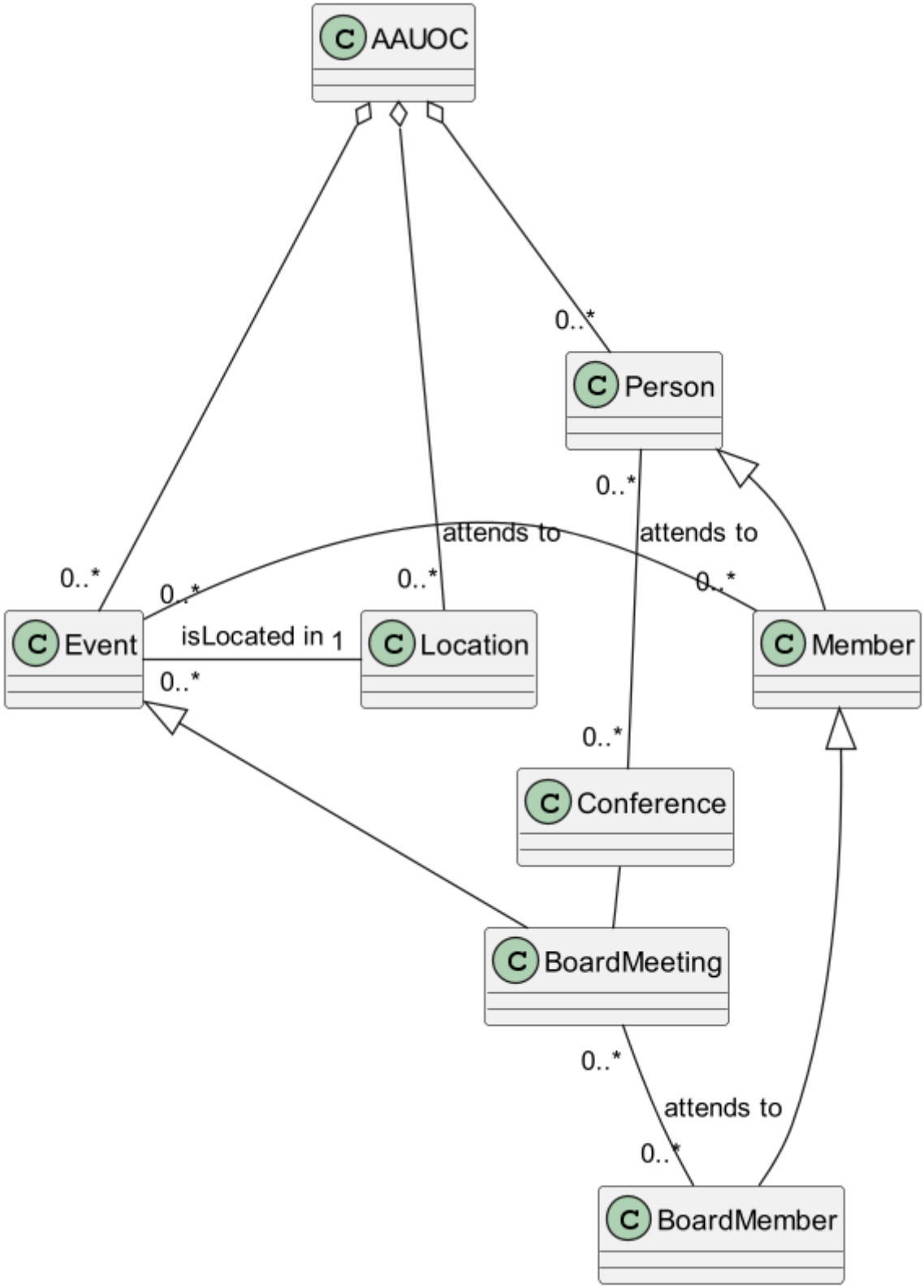
```
Class01 "1" *-- "many" Class02 : contains
```

3. Ahora rellenaremos las cardinalidades, quedando el codigo algo similar al siguiente:

```
@startuml

AAUOC o-down--- "0..*" Location
AAUOC o-down-- "0..*" Person
AAUOC o-down--- "0..*" Event
Location "1" -right- "0..*" Event : isLocated in
Event "0..*" -right- "0..*" Member : attends to
Person <|-down- Member
Event <|-- BoardMeeting
```

```
Member <|-- BoardMember
BoardMeeting "0..*" -- "0..*" BoardMember : attends to
Conference -right- BoardMeeting
Conference "0..*" -up- "0..*" Person : attends to

@enduml
```

Resultado:


AAUOC
Person
Event
Location
Member
Conference
BoardMeeting
BoardMember
isLocated in
attends to
attends to
0..*
1