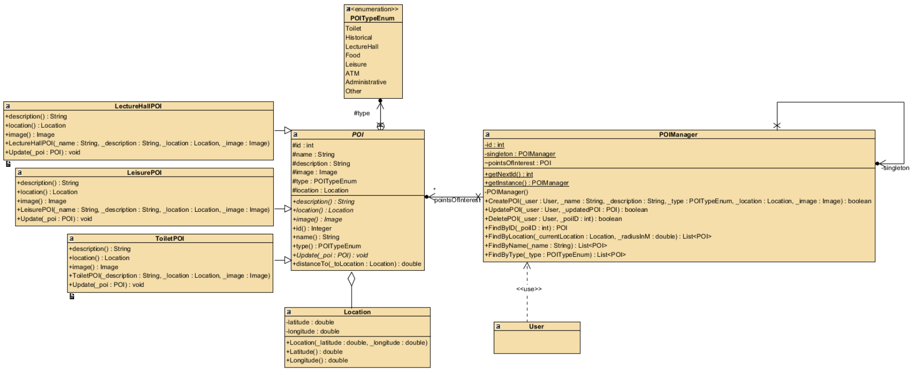
1 Points of Interest Module

1.1 UML Class Diagram



<<enumeration>>
POITypeEnum
Toilet
Historical
LectureHall
Food
Leisure
ATM
Administrative
Other
#type
LectureHallPOI
+description() : String
+location() : Location
+image() : Image
+LectureHallPOI(_name : String, _description : String, _location : Location, _image : Image)
+Update(_poi : POI) : void
LeisurePOI
+description() : String
+location() : Location
+image() : Image
+LeisurePOI(_name : String, _description : String, _location : Location, _image : Image)
+Update(_poi : POI) : void
ToiletPOI
+description() : String
+location() : Location
+image() : Image
+ToiletPOI(_description : String, _location : Location, _image : Image)
+Update(_poi : POI) : void
POI
#id : int
#name : String
#description : String
#image : Image
#type : POITypeEnum
#location : Location
+description() : String
+location() : Location
+image() : Image
+id() : Integer
+name() : String
+type() : POITypeEnum
+Update(_poi : POI) : void
+distanceTo(_toLocation : Location) : double
*
pointsOfInterest
Location
-latitude : double
-longitude : double
+Location(_latitude : double, _longitude : double)
+Latitude() : double
+Longitude() : double
POIManager
-id : int
-singleton : POIManager
~pointsOfInterest : POI
+getNextId() : int
+getInstance() : POIManager
-POIManager()
+CreatePOI(_user : User, _name : String, _description : String, _type : POITypeEnum, _location : Location, _image : Image) : boolean
+UpdatePOI(_user : User, _updatedPOI : POI) : boolean
+DeletePOI(_user : User, _poiID : int) : boolean
+FindByID(_poiID : int) : POI
+FindByLocation(_currentLocation : Location, _radiusInM : double) : List<POI>
+FindByName(_name : String) : List<POI>
+FindByType(_type : POITypeEnum) : List<POI>
-singleton
<<use>>
User

Figure 1: Class Diagram for the Points of interest module

1.2 Use Case Diagram

1.3 Architectural Design of Module

The Points of Interest module will be used to persist and deliver data regarding locations that a user might find of interest. It will interact with the Events module, as well as the GIS module in order to retrieve and save it geographical location.

One can assume that different types of Points of Interest(POI) will have different fields, constraints and requirements, therefore the POI module will make use of the factory design pattern, in order to create objects without exposing the creation logic to the client and allow us to refer to the newly created object using a common interface.

The Points of Interest Manager class will make use of the Singleton design pattern to ensure that only one instance of the class exists, it will also act as the factory class to the points of interest class.

At the core a Point of Interest object will have a location, description, unique id, type, name and image. The concrete object will then be able to impose additional attributes and constraints, such as Gender for bathrooms for example.

Finally, the abstract point of interest class will be equipped with a stringify function to get a JSON representation of the object.

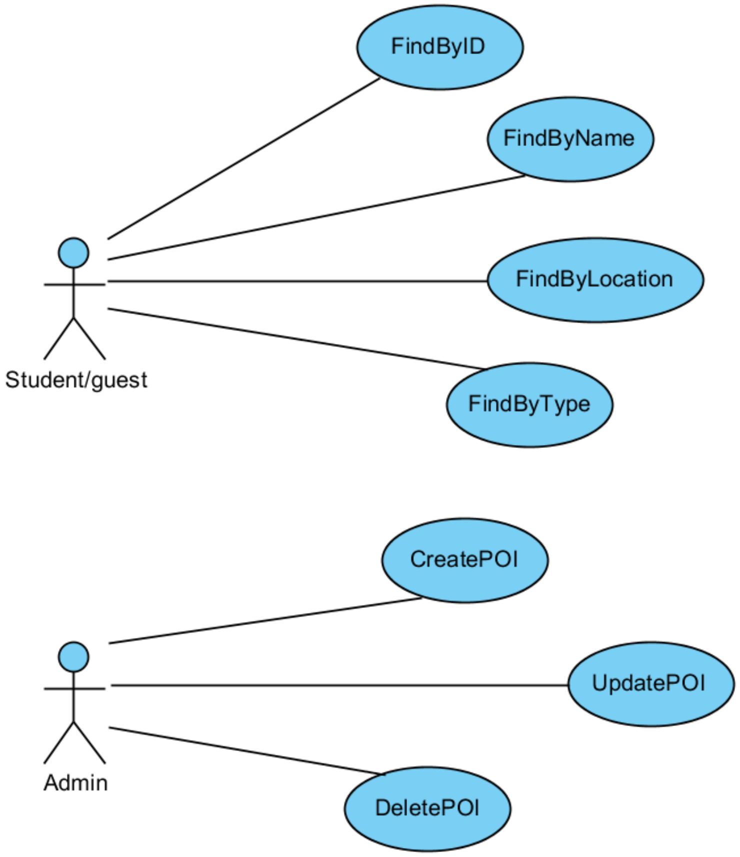


Figure 2: Use Case Diagram for the Points of interest module

1.4 Technologies Used

It can be assumed that different types of Points of Interest will have different fields, constraints and requirements, therefore a traditional relational database will not be suitable for this type of data. Hence we will rather go for a document-oriented database like MongoDB, as the database server.

The Points of Interest CRUD functions will be exposed as Simple Object Access Protocol (SOAP) web services, using ASP.NET core and Entity framework carrying a JSON payload.

The logic of the module will be written in .NET.

1.5 Non-Trivial Implementation Tasks

The point of interest module will be called in two ways. Firstly, by administrative user who would be able to add, remove and update points of interest around campus. Secondly, students and guests will be able to use it, by searching for specific locations by name, type of location, specific areas etc. Lastly the points of interest classes SearchByLocation function will be called by the navigation module, as the user navigates campus.

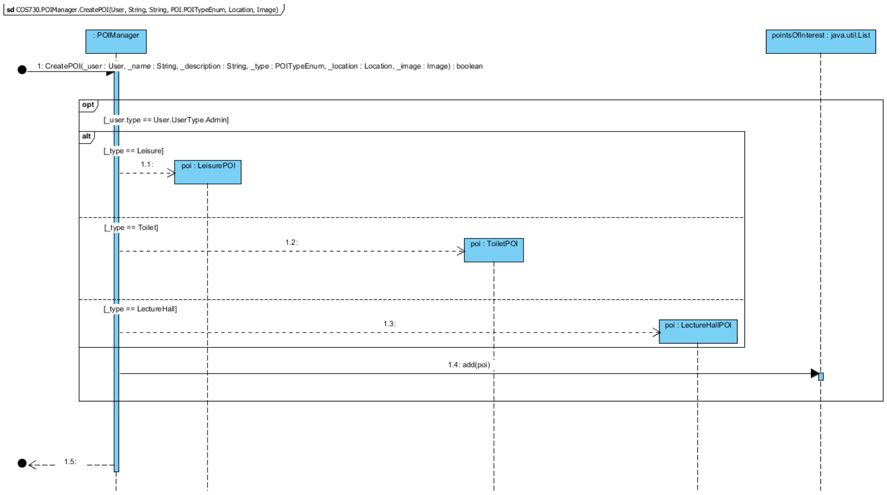

sd COS730.POIManager.CreatePOI(User, String, String, POI.POITypeEnum, Location, Image)
: POIManager
pointsOfInterest : java.util.List
1: CreatePOI(_user : User, _name : String, _description : String, _type : POITypeEnum, _location : Location, _image : Image) : boolean
opt
[_user.type == User.UserType.Admin]
alt
[_type == Leisure]
1.1:
poi : LeisurePOI
[_type == Toilet]
1.2:
poi : ToiletPOI
[_type == LectureHall]
1.3:
poi : LectureHallPOI
1.4: add(poi)
1.5:

Figure 3: Sequence Diagram for the CreatePOI function

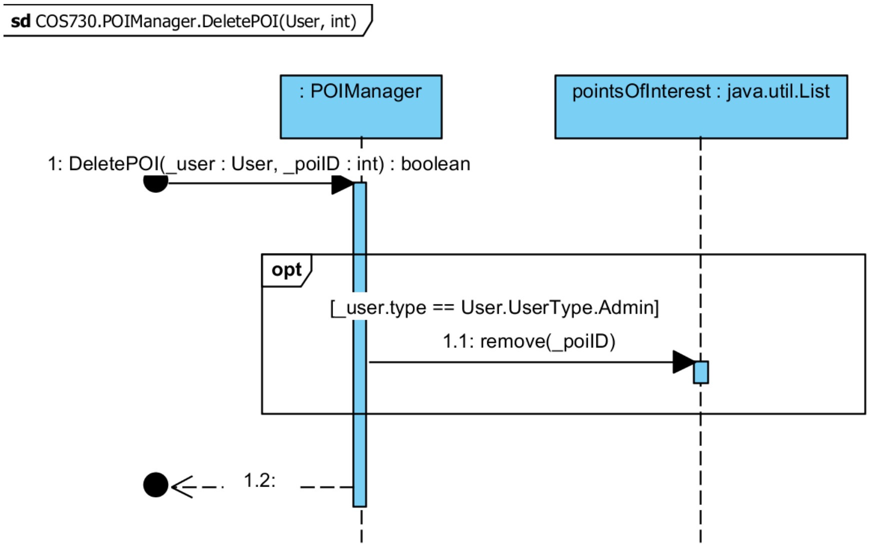

sd COS730.POIManager.DeletePOI(User, int)
: POIManager
pointsOfInterest : java.util.List
1: DeletePOI(_user : User, _poiID : int) : boolean
opt
[_user.type == User.UserType.Admin]
1.1: remove(_poiID)
1.2:

Figure 4: Sequence Diagram for the DeletePOI function

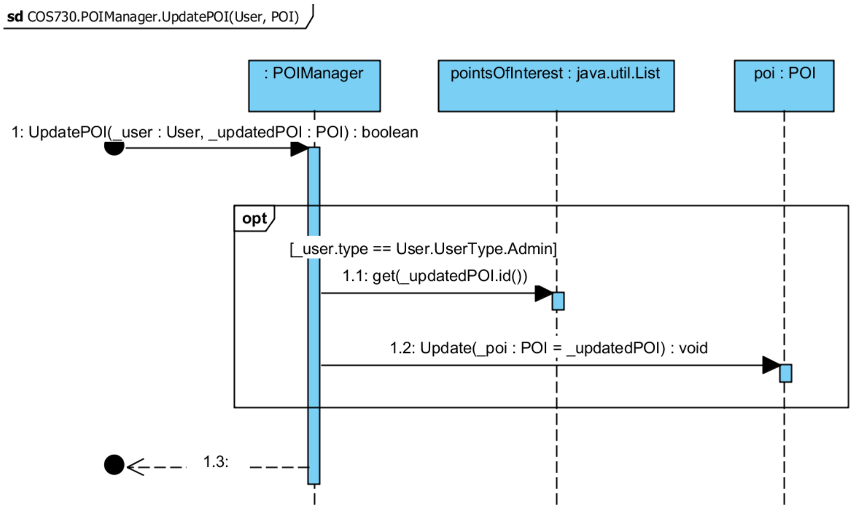

sd COS730.POIManager.UpdatePOI(User, POI)
: POIManager
pointsOfInterest : java.util.List
poi : POI
1: UpdatePOI(_user : User, _updatedPOI : POI) : boolean
opt
[_user.type == User.UserType.Admin]
1.1: get(_updatedPOI.id())
1.2: Update(_poi : POI = _updatedPOI) : void
1.3:

Figure 5: Sequence Diagram for the UpdatePOI function

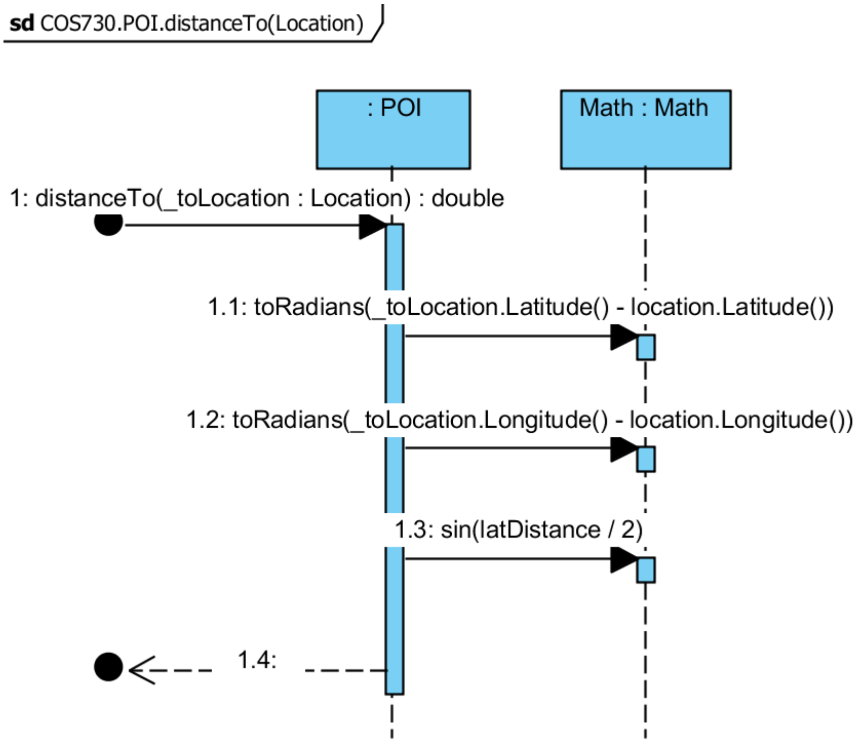

sd COS730.POI.distanceTo(Location)
: POI
Math : Math
1: distanceTo(_toLocation : Location) : double
1.1: toRadians(_toLocation.Latitude() - location.Latitude())
1.2: toRadians(_toLocation.Longitude() - location.Longitude())
1.3: sin(latDistance / 2)
1.4:

Figure 6: Sequence Diagram for the DistanceTo function

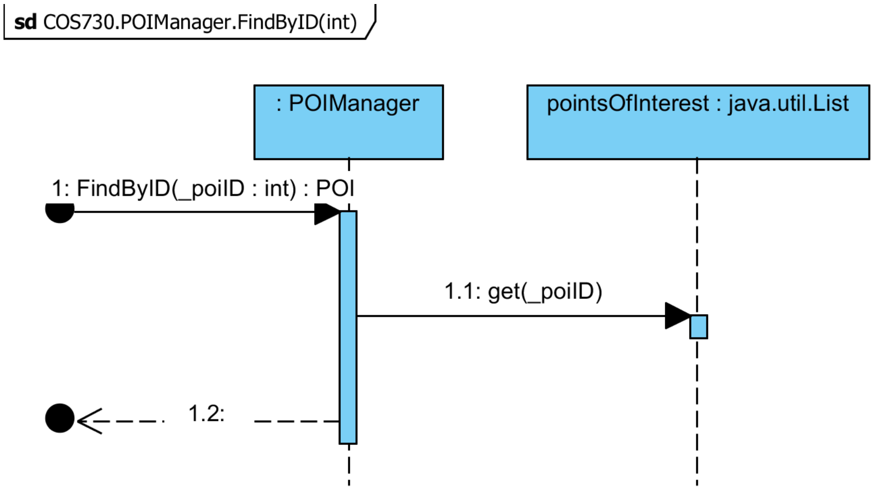

sd COS730.POIManager.FindByID(int)
: POIManager
pointsOfInterest : java.util.List
1: FindByID(_poiID : int) : POI
1.1: get(_poiID)
1.2:

Figure 7: Sequence Diagram for the FindByID function

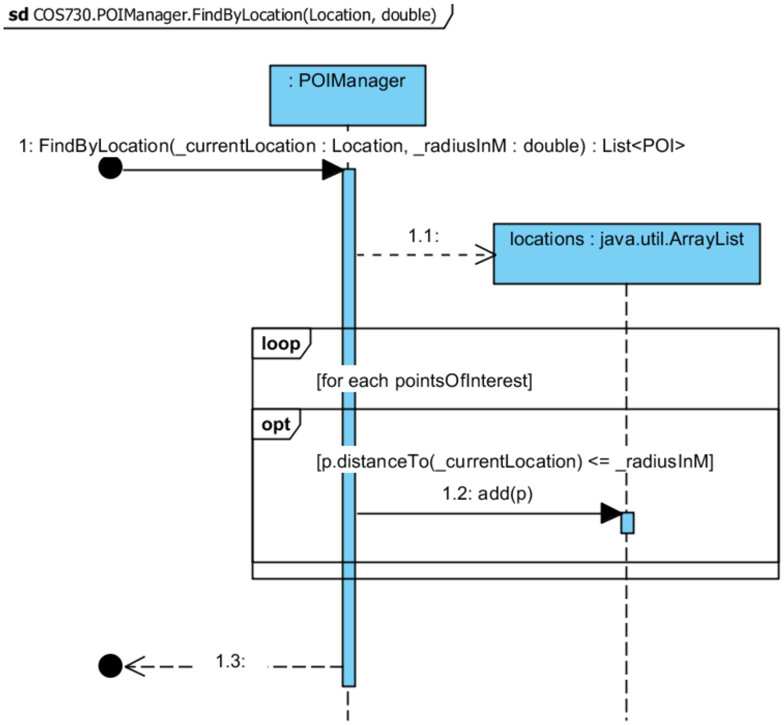

sd COS730.POIManager.FindByLocation(Location, double)
: POIManager
1: FindByLocation(_currentLocation : Location, _radiusInM : double) : List<POI>
1.1:
locations : java.util.ArrayList
loop
[for each pointsOfInterest]
opt
[p.distanceTo(_currentLocation) <= _radiusInM]
1.2: add(p)
1.3:

Figure 8: Sequence Diagram for the FindByLocation function

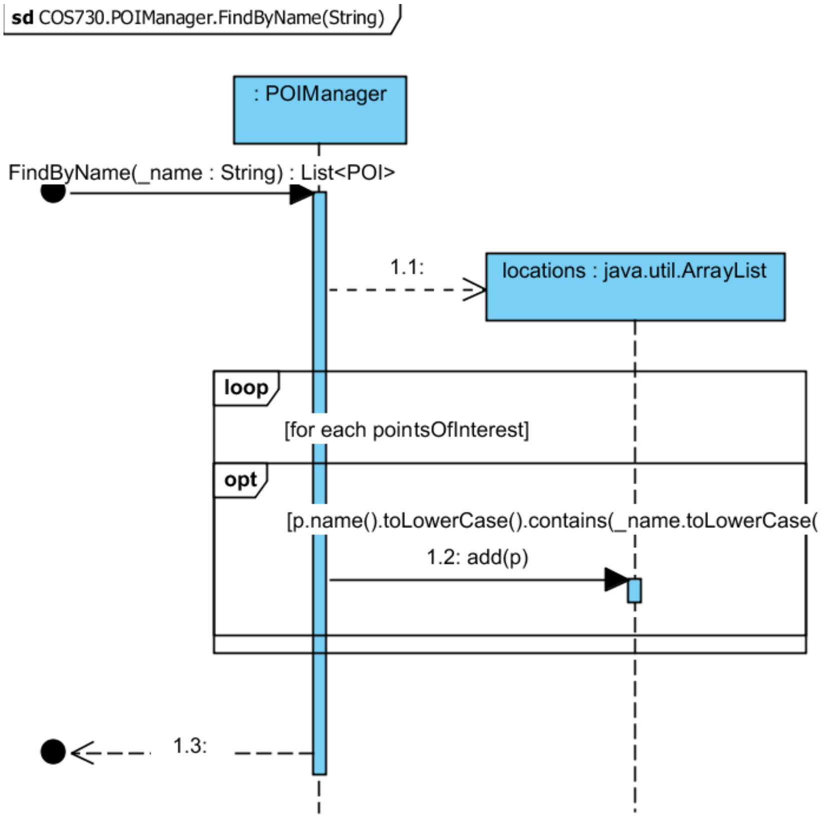

sd COS730.POIManager.FindByName(String)
: POIManager
FindByName(_name : String) : List<POI>
1.1:
locations : java.util.ArrayList
loop
[for each pointsOfInterest]
opt
[p.name().toLowerCase().contains(_name.toLowerCase(
1.2: add(p)
1.3:

Figure 9: Sequence Diagram for the FindByName function

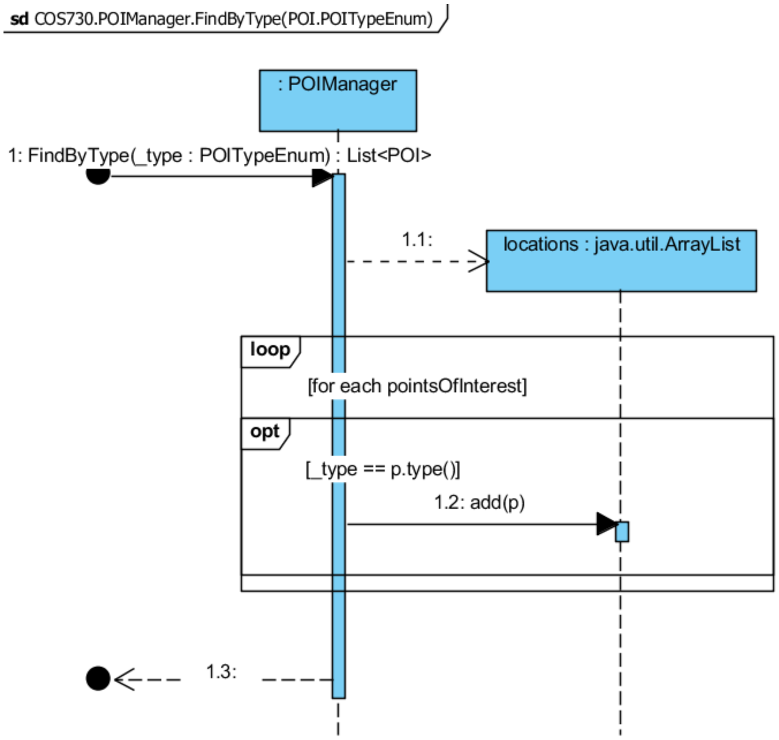

sd COS730.POIManager.FindByType(POI.POITypeEnum)
: POIManager
1: FindByType(_type : POITypeEnum) : List<POI>
1.1:
locations : java.util.ArrayList
loop
[for each pointsOfInterest]
opt
[_type == p.type()]
1.2: add(p)
1.3:

Figure 10: Sequence Diagram for the FindByType function

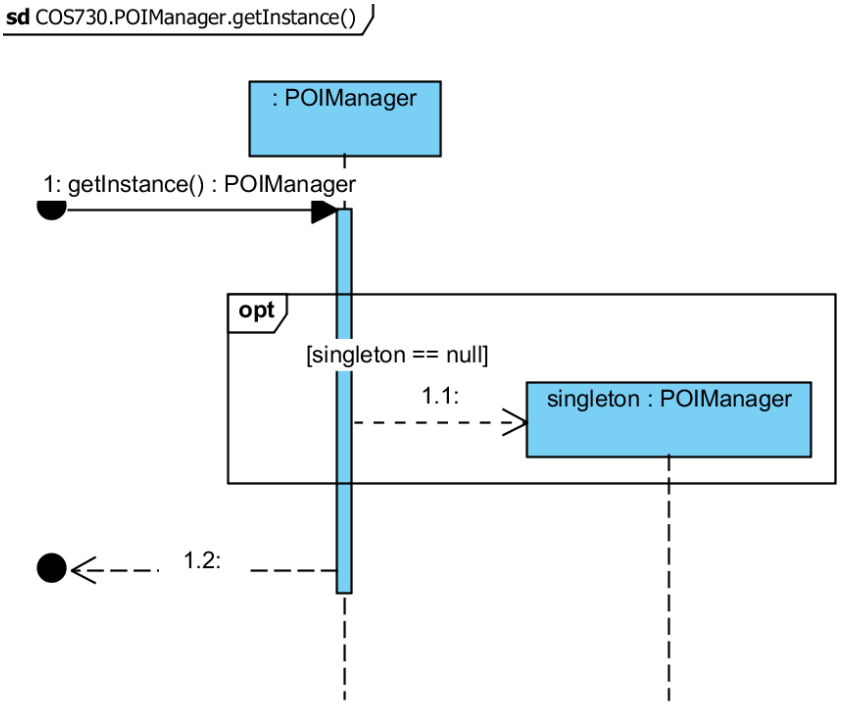


Figure 11: Sequence Diagram for the getInstance function