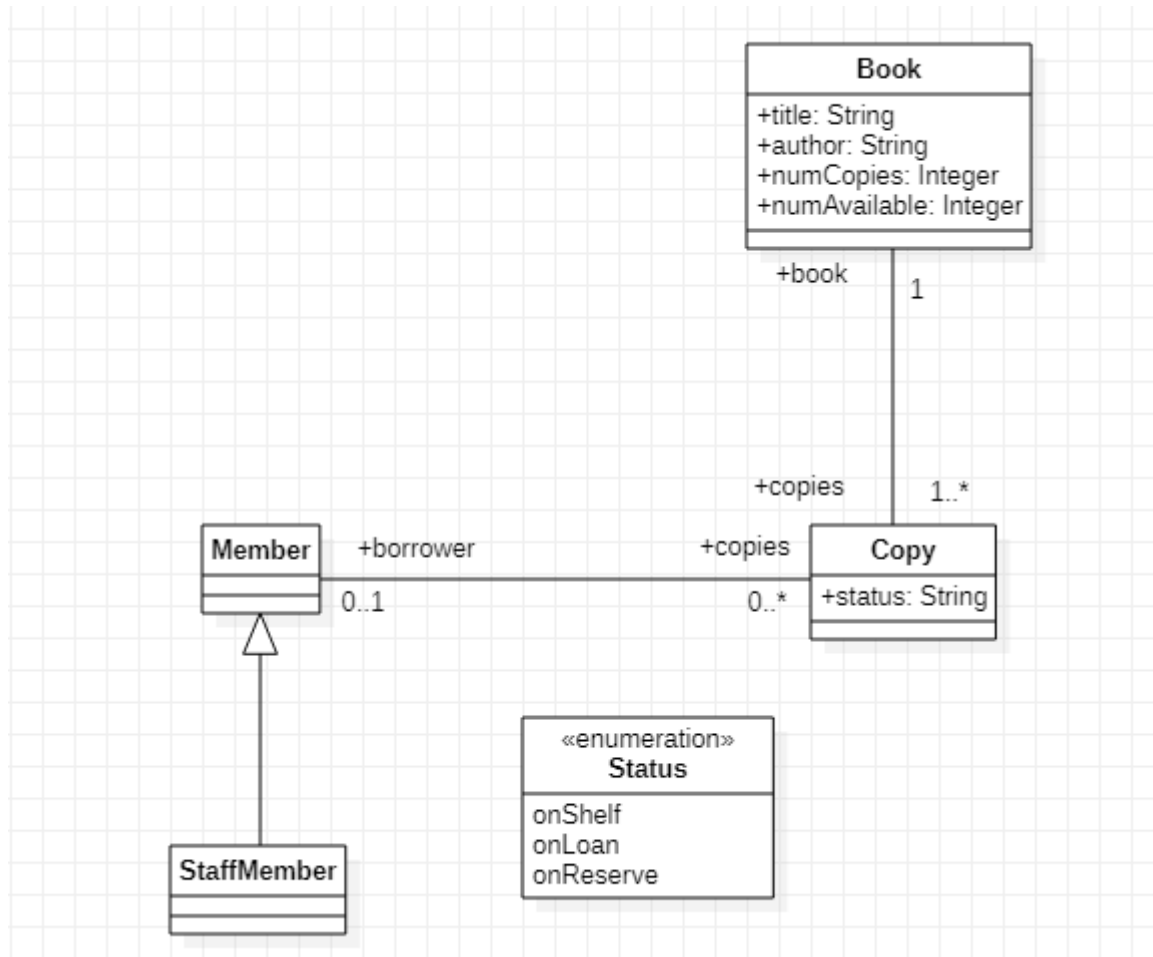


Library Usecase Realisation

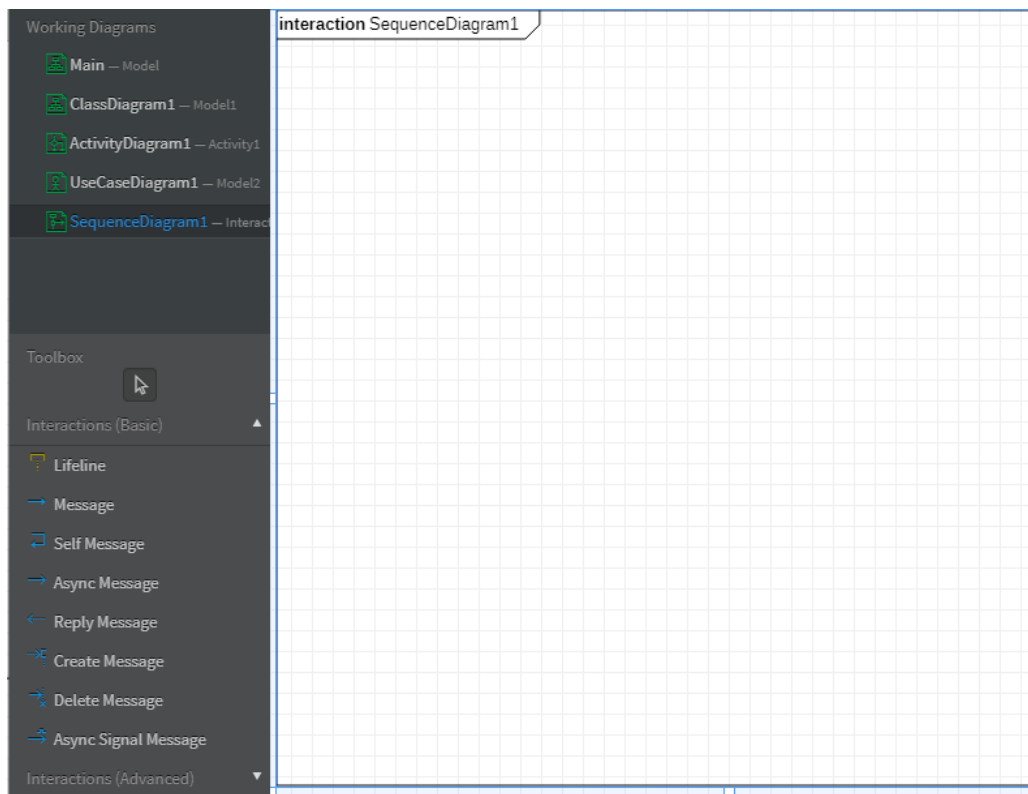
From last week we got as far as a simple domain model or class diagram. We now wish to refine it using usecase realisation which means seeing how a usecase may be realised in terms of objects interacting in a sequence diagram.



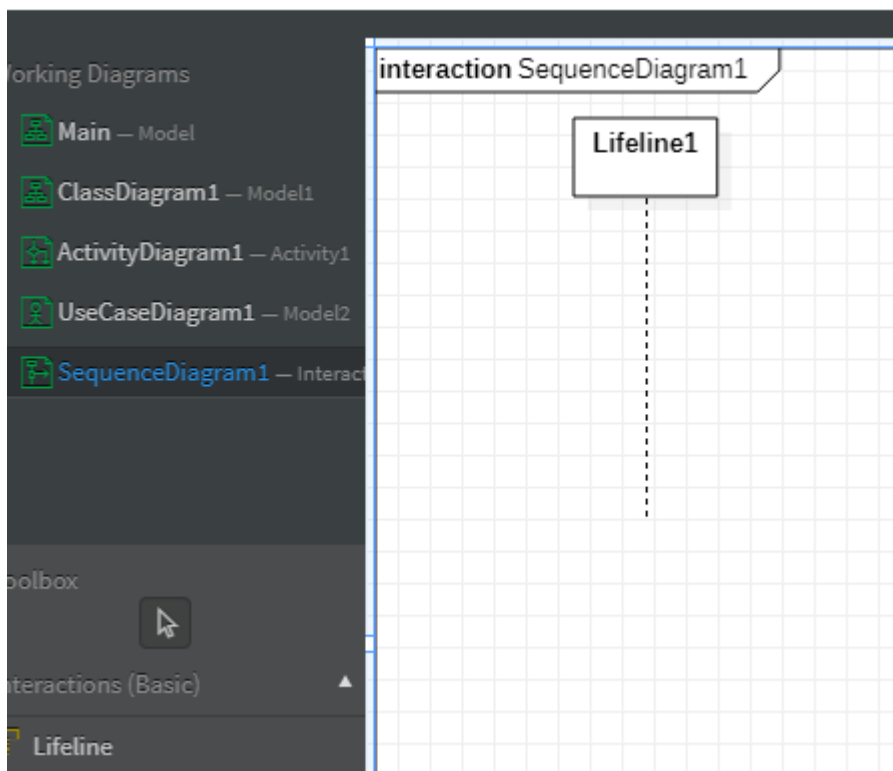
Open the StarUML file with this class diagram and the library usecases. They should be in the one file.

Sequence Diagram

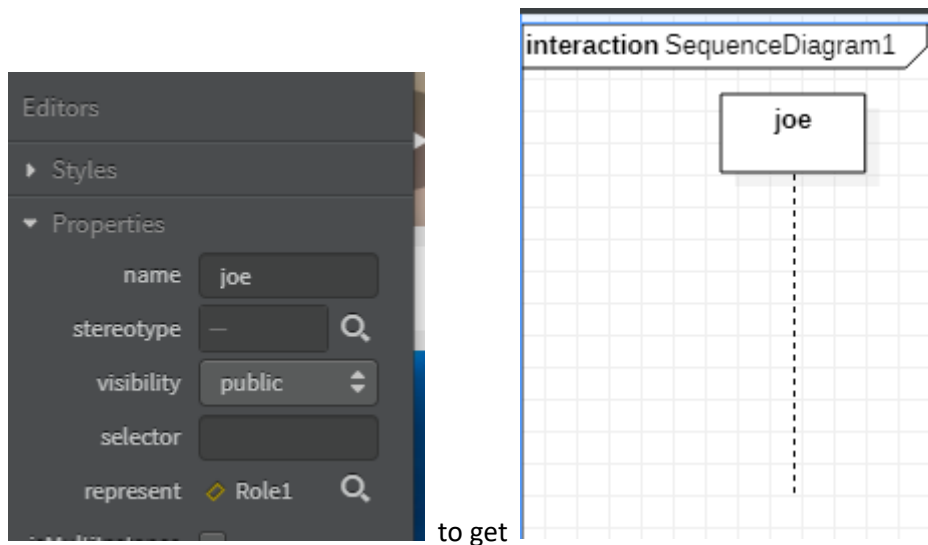
Add a sequence diagram to your library model to get:



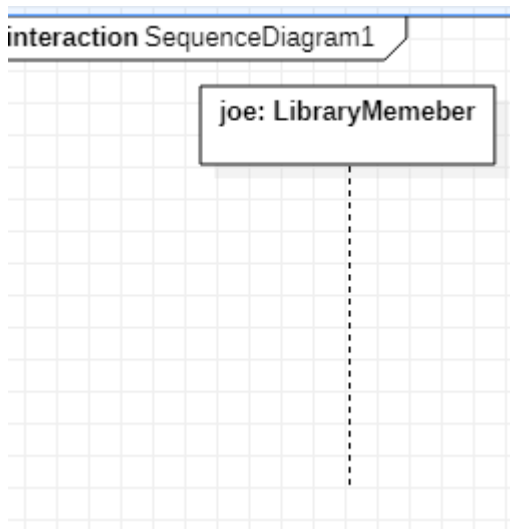
Using the tool palette for drawing sequence diagram, place a **Lifeline** object on the drawing canvas.



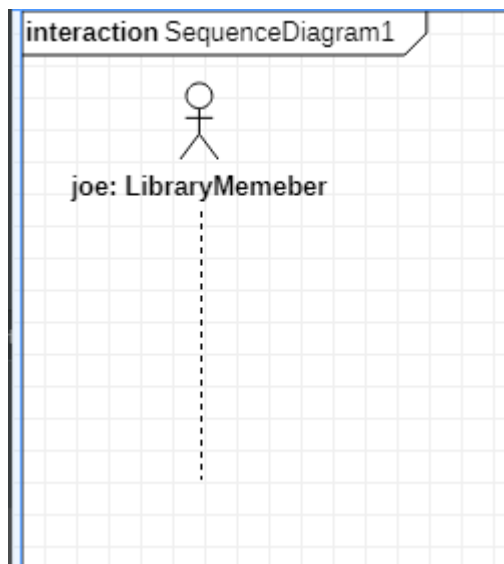
Use the properties editor on bottom right hand side to rename this object lifeline to **joe**.



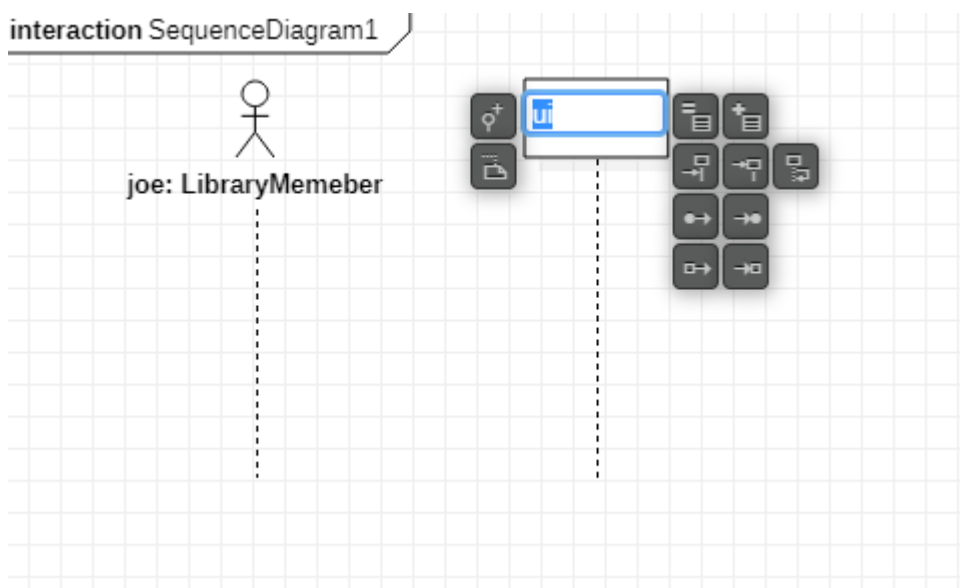
On the Model Explorer pane, on top right hand side, select Role1, its properties will then appear in the pane underneath. In the **type** field, start typing LibraryMember until the actor icon appear (after typing a few letters) and then press enter. We are making the **joe** lifeline an instance of the actor **LibraryMember**.



select the lifeline, right click on it and select **Format / Stereotype Display / Icon** to get:

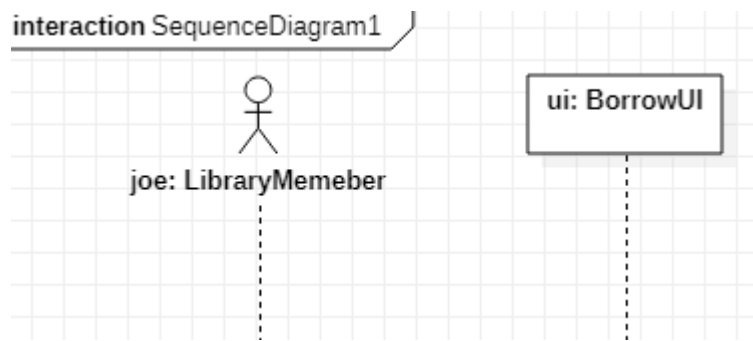


Add another lifeline object names ui. Double click on it so that property buttons appear as show. Select **Create Type** and then type in **BorrowUI** as the type or class name for the ui lifeline.

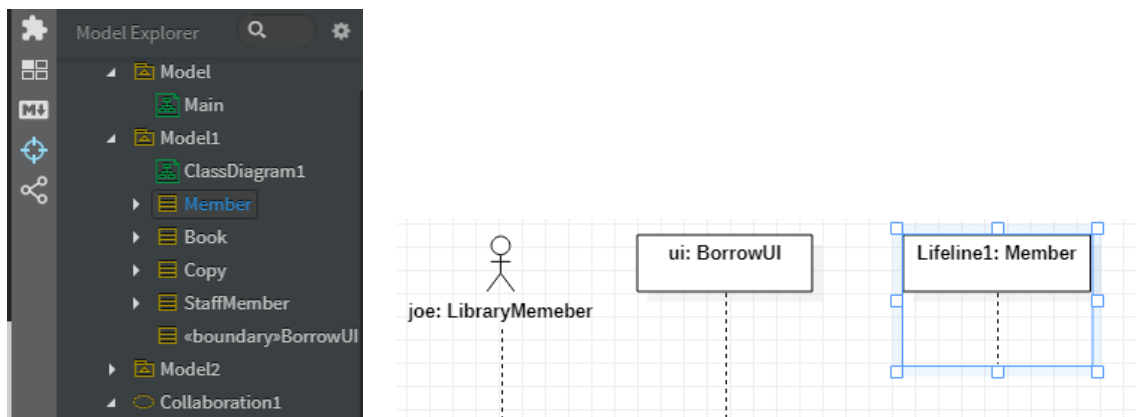


An "Input" dialog box is shown. It has a title bar with "Input" and a close button (X). The main text says "Enter a type name to create". Below this is a text input field containing the text "BorrowUI". At the bottom of the dialog are two buttons: "OK" and "Cancel".

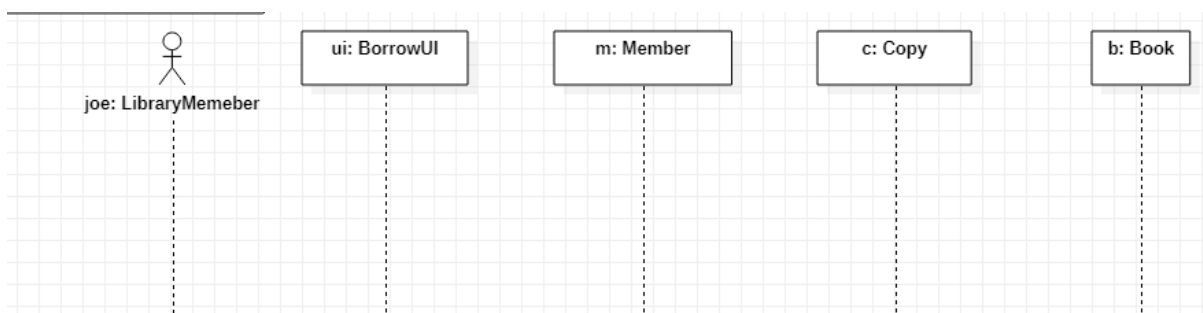
Should get



You have now seen two ways to link a lifeline object with a type or class. We will now try a third way. From Model Explorer pane on top right hand side, click on **Member** class and drag it to the sequence diagram drawing area.

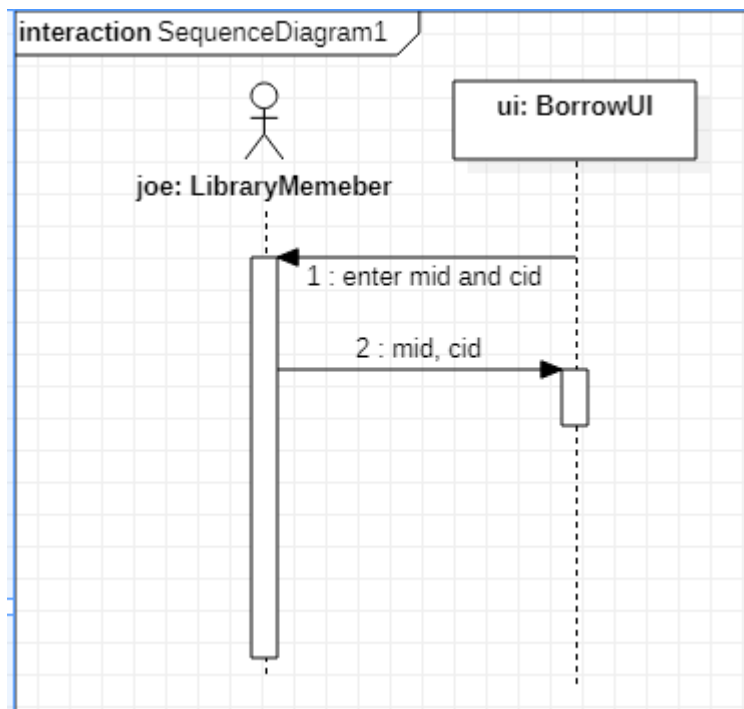


Rename it to **m** and two more lifeline objects as show next.



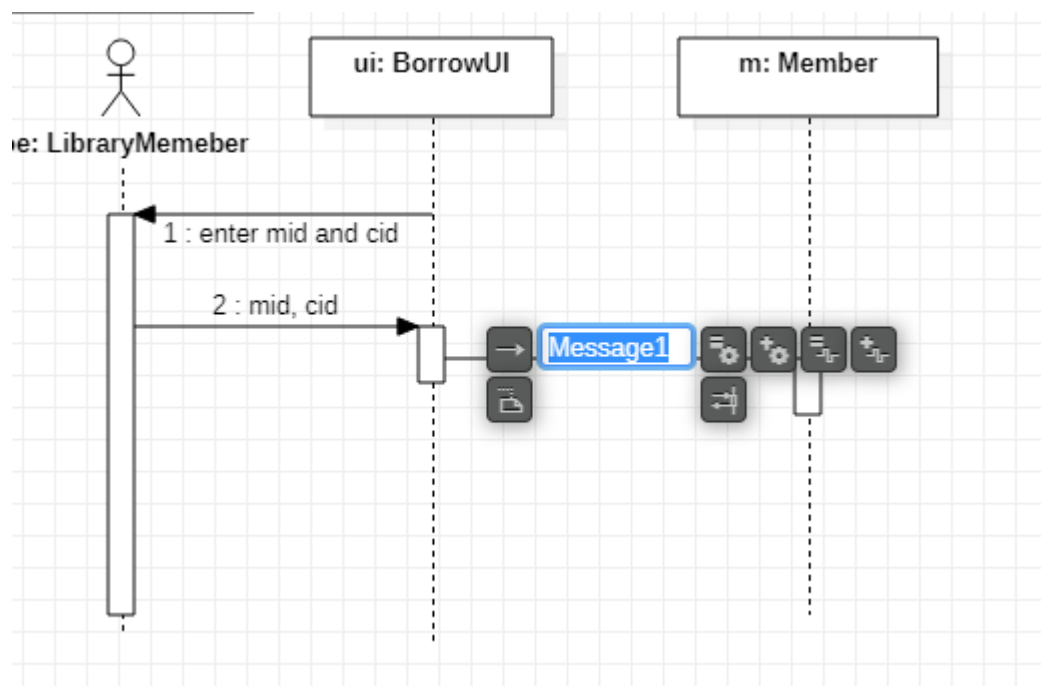
The next task is to explore how these objects might interact and send messages to one another. This is to help us understand a little of how the objects relate to one another when a usecase is executed. Note, this is a very abstract view of how object might interact, very far removed from what happens in executing code. The purpose here is to understand the usecases better, no to implement them just yet.


Use the message tool to get

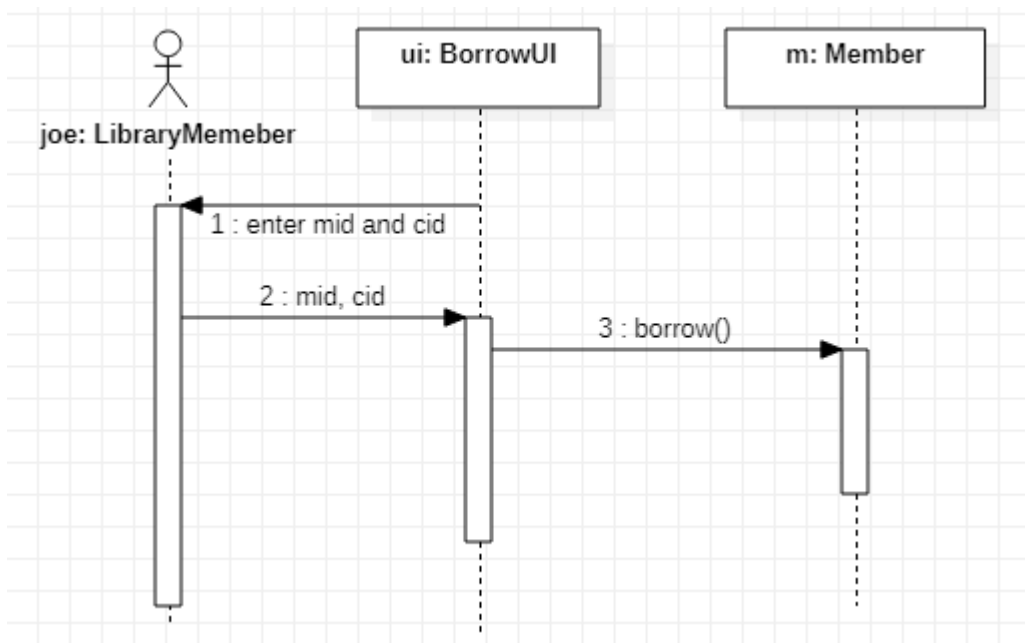
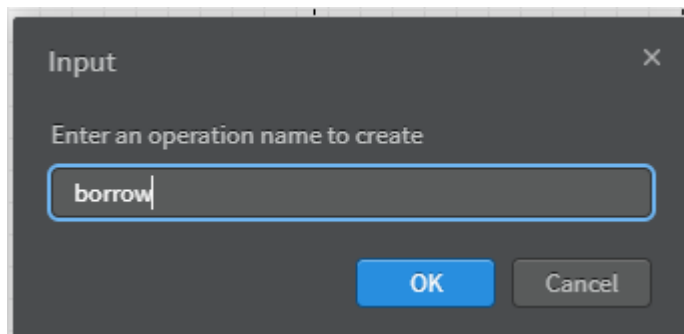


You may need to resize the lifeline narrow rectangles to make them longer so that they look like this.

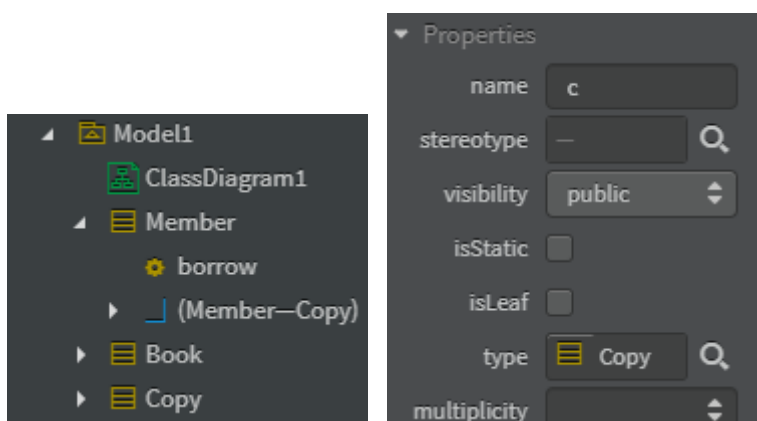
Next send a message from **ui** to **m**.

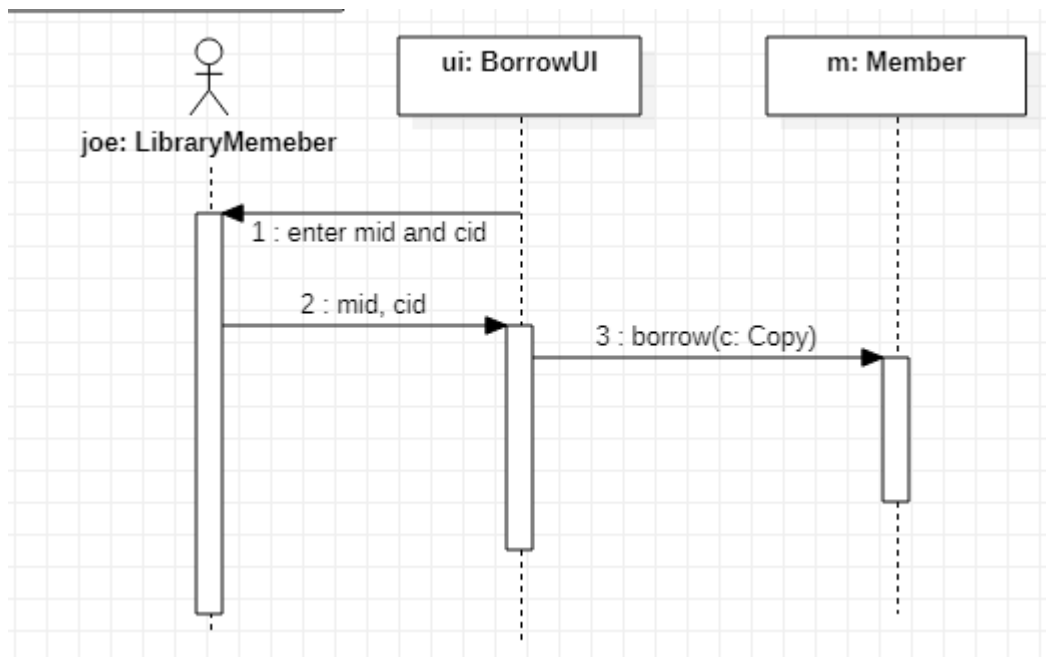


Click on the **Create Operation**  tool button and create a corresponding operation named **borrow**.

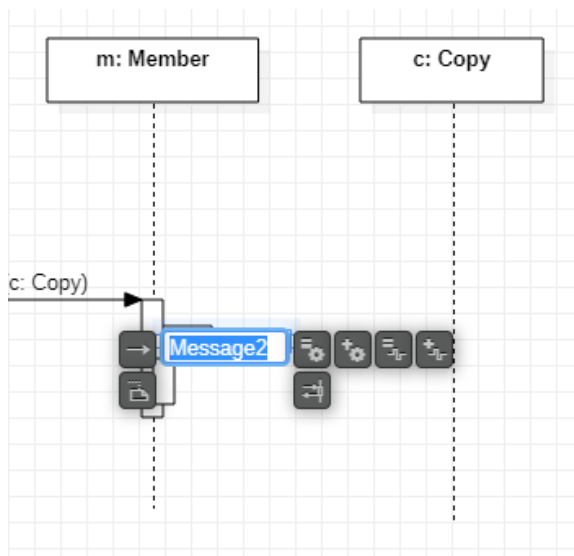


We next wish to add a parameter to this operation or function. In the Model Explorer pane, select this operation and right click on it. Then select **Add / Parameter**. Rename the parameter to **c** and give it the type Copy.





Lifeline object **m** when told that it's borrowing **c**, will do a self-check to verify that this is ok. Use the self-message tool to go this and create a corresponding operation as show next.

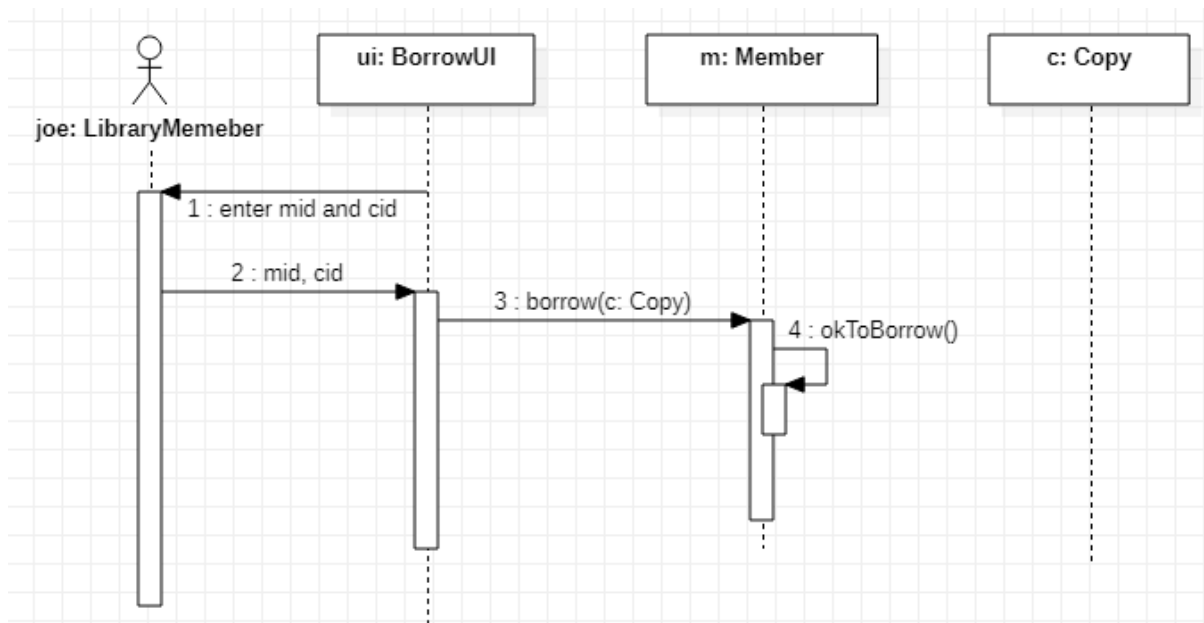


✕

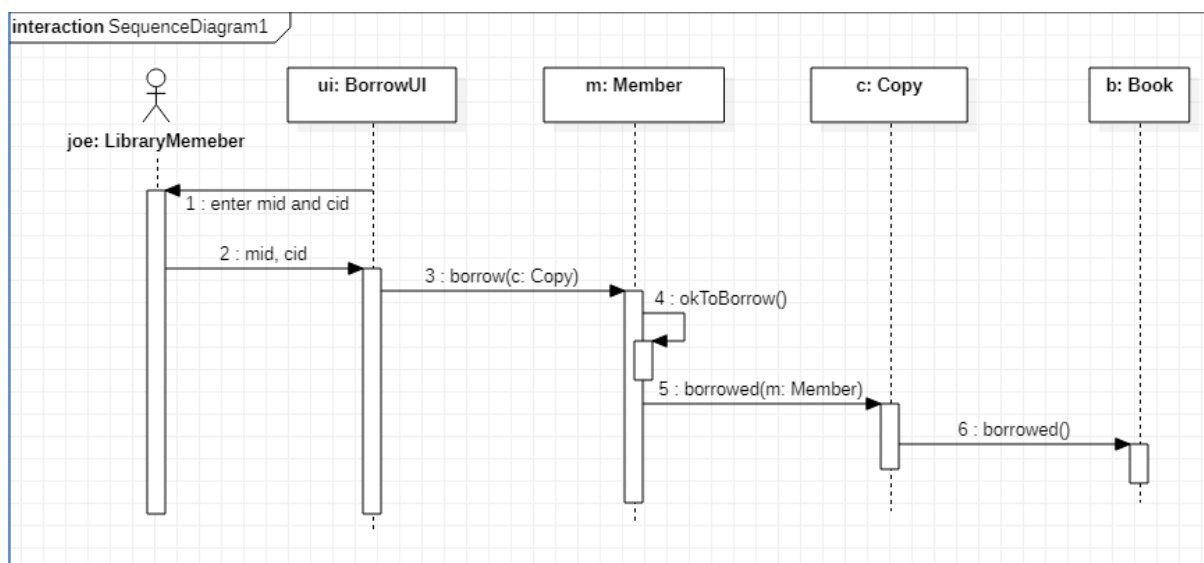
Enter an operation name to create

OK
Cancel

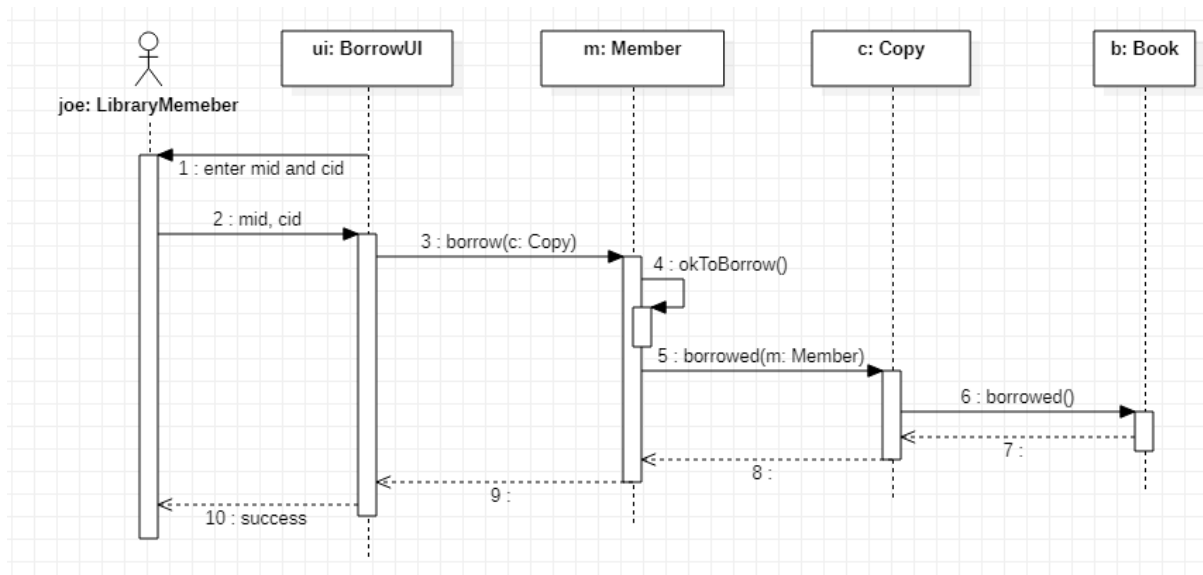
You might need to lengthen the object lifelines to get this layout.



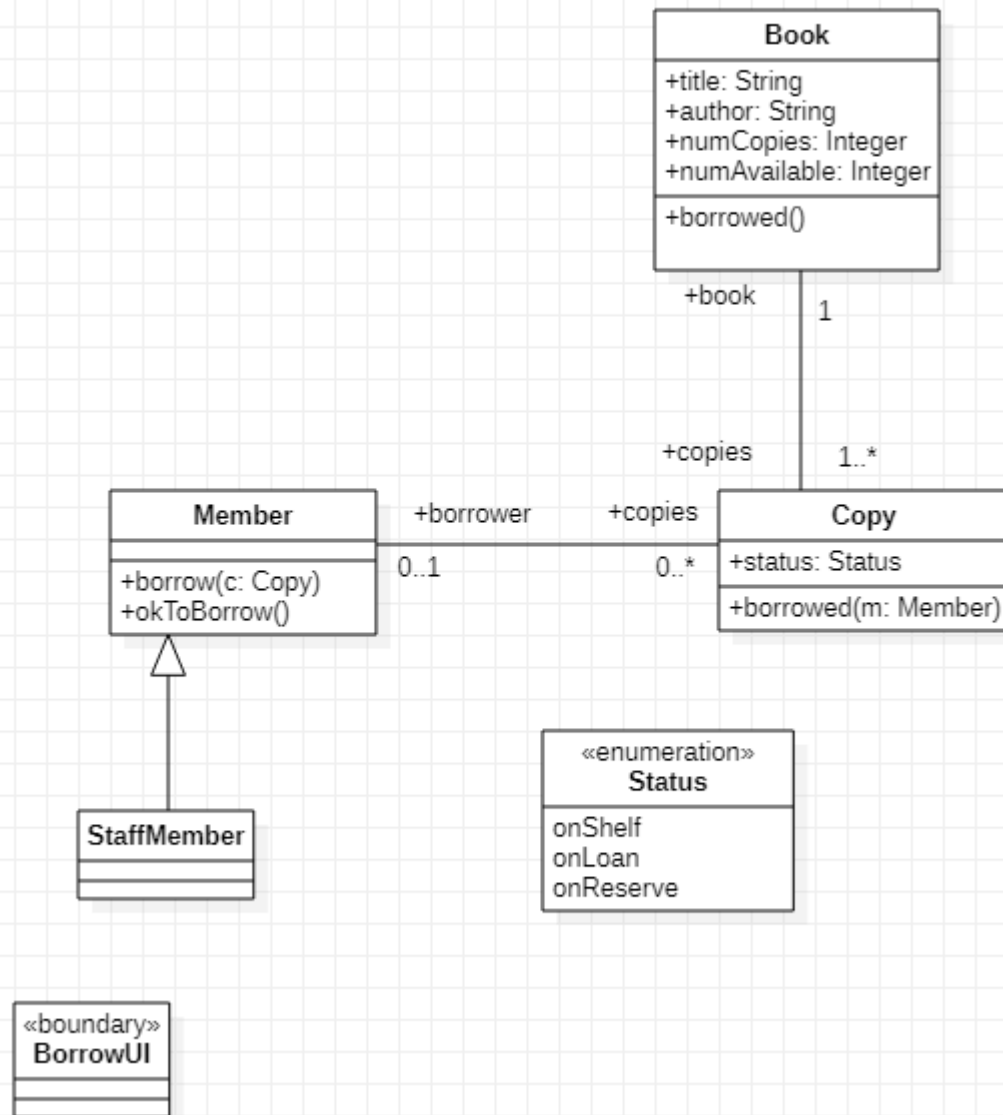
Then create add two more messages, creating an operation for each one, and a parameter for one of them to get:



If you wish you can add reply messages as shown below. A reply message is like a function returning control after it has finished executing.



Finally, switch to you class diagram. Creating all these operation should have updated it.



Exercise

Add sequence diagram for scenario of borrowing and paying a fine

Add a return book sequence diagram and another one that you consider relevant.