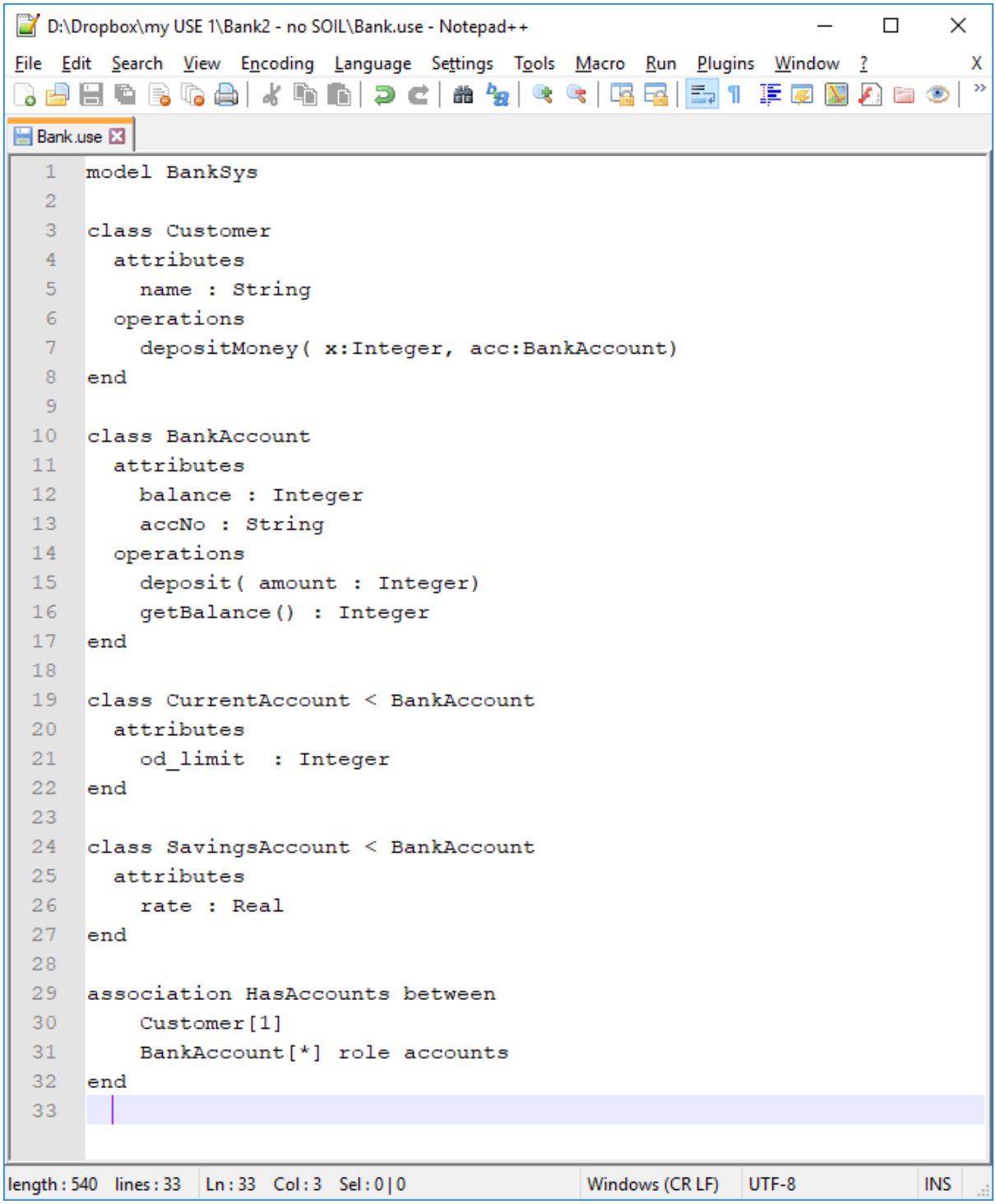
Subclasses, Associations, SOIL and OCL in USE

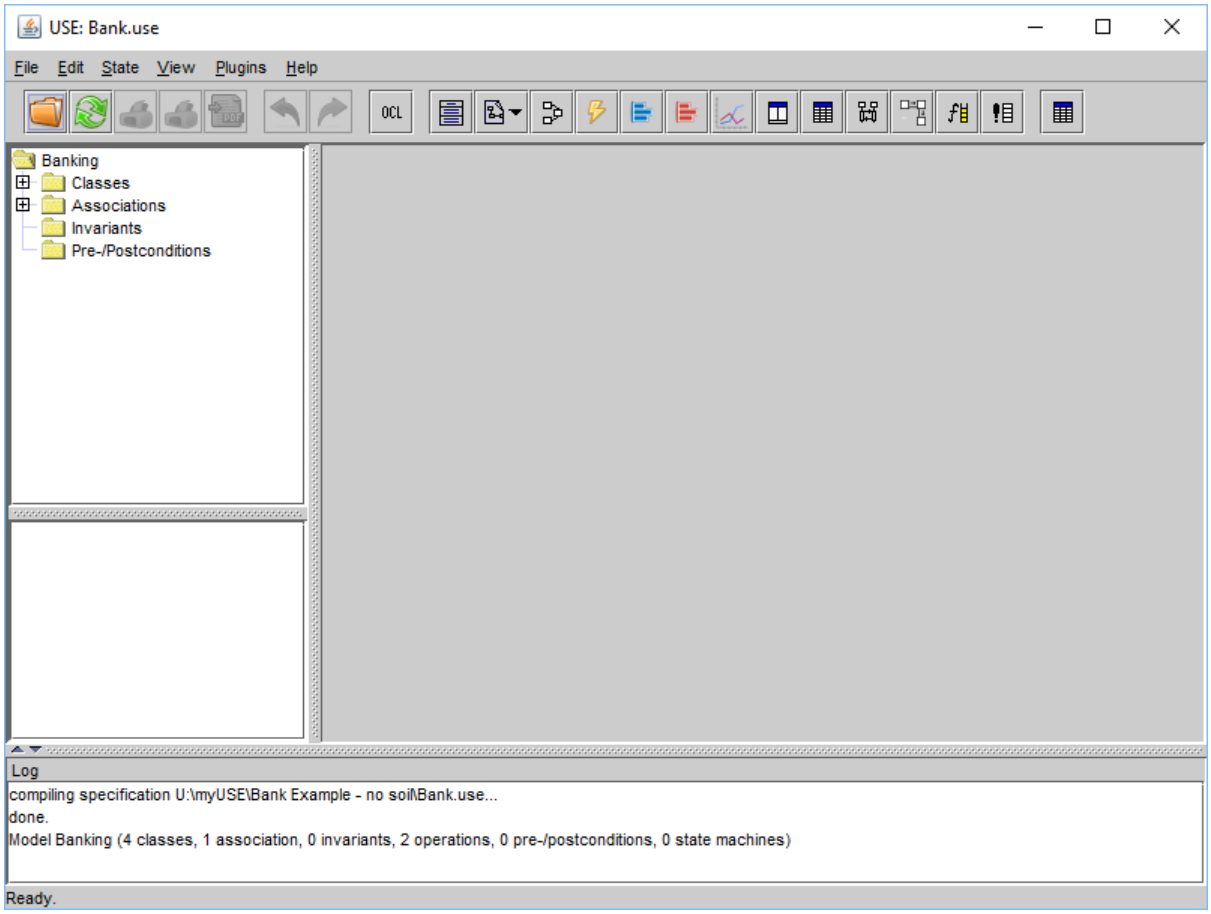
Now we will create an association between classes and also create subclasses. We will extend the USE model from last week but first copy it to a **new folder** called Bank2. Rename it to bank.use.

Then use Notepad++ or Sublime to extend and modify the original specification as follows. Note that we remove the implementation of deposit() for the moment and add another operation to BankAccount.

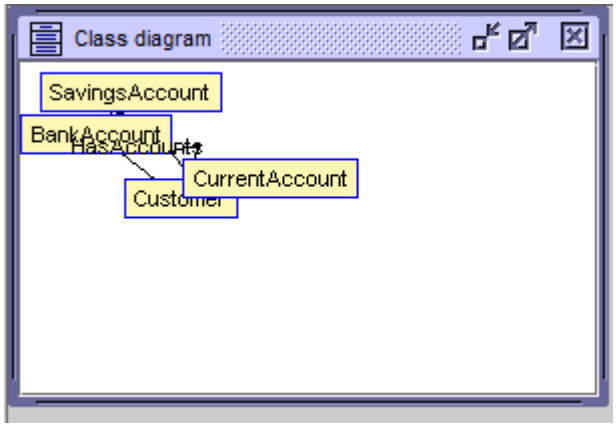


Load Specification into USE

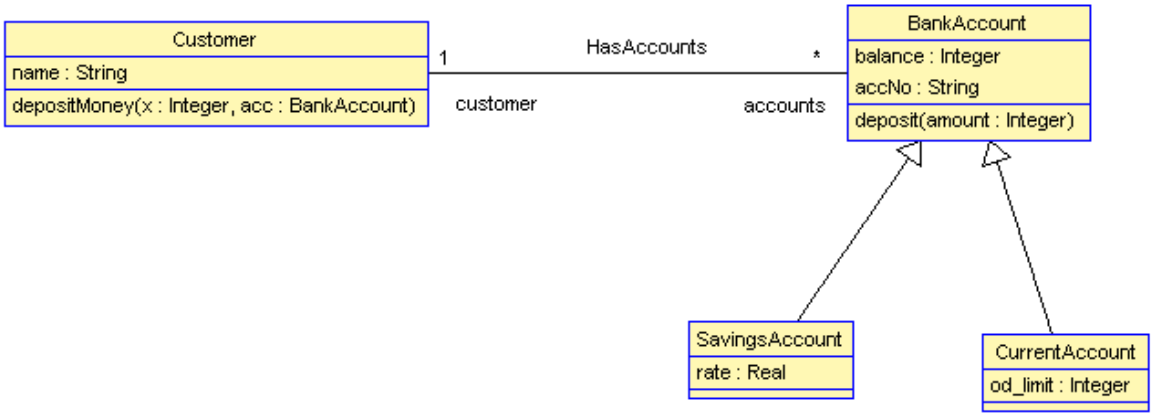
Then start USE and use the USE menu **File | Open Specification** to load your modified USE code. If successful, USE GUI will look like below. If not you have errors in your code.



To see your class diagram in USE, click on the **Create class diagram view** icon  . Class diagram like blow will then appear.



Resize this window to make it bigger and right-click on it to see context sensitive menu, select a number of the show options so that attributes, operation, multiplicities and role names are visible. Then reorganise your class diagram layout to something like:

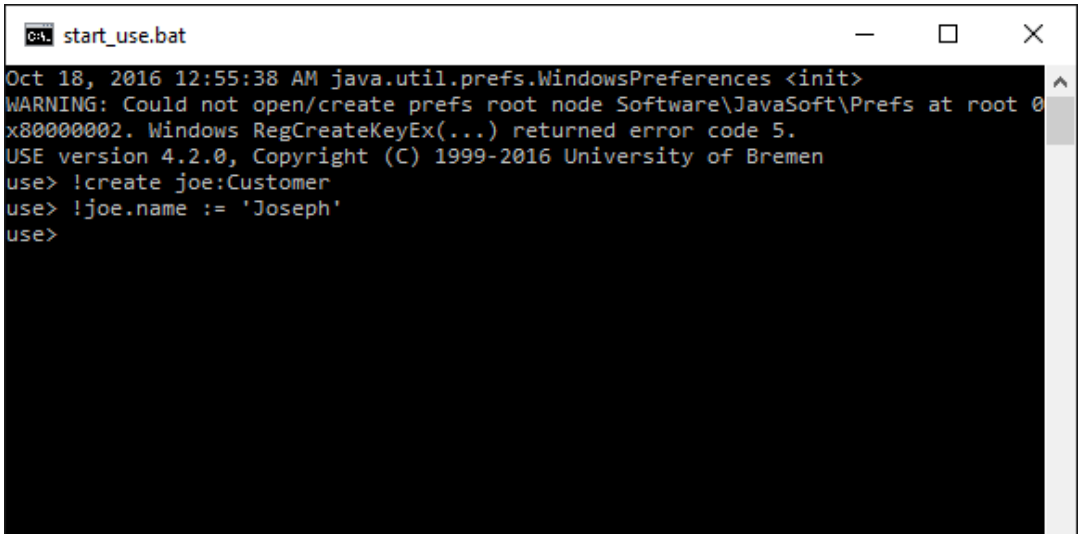


Then save this layout by right-clicking on the class diagram and selecting **Save layout**. Save it with the name bank.clt. You can reload this layout in future lab sessions.

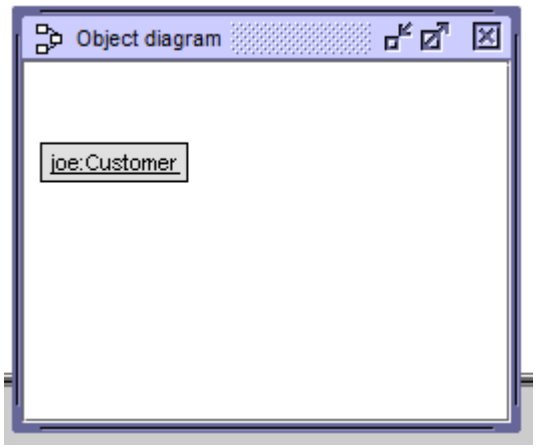
Create Objects to Animate your Specification

On the **Command window** that starts with use type in the commands

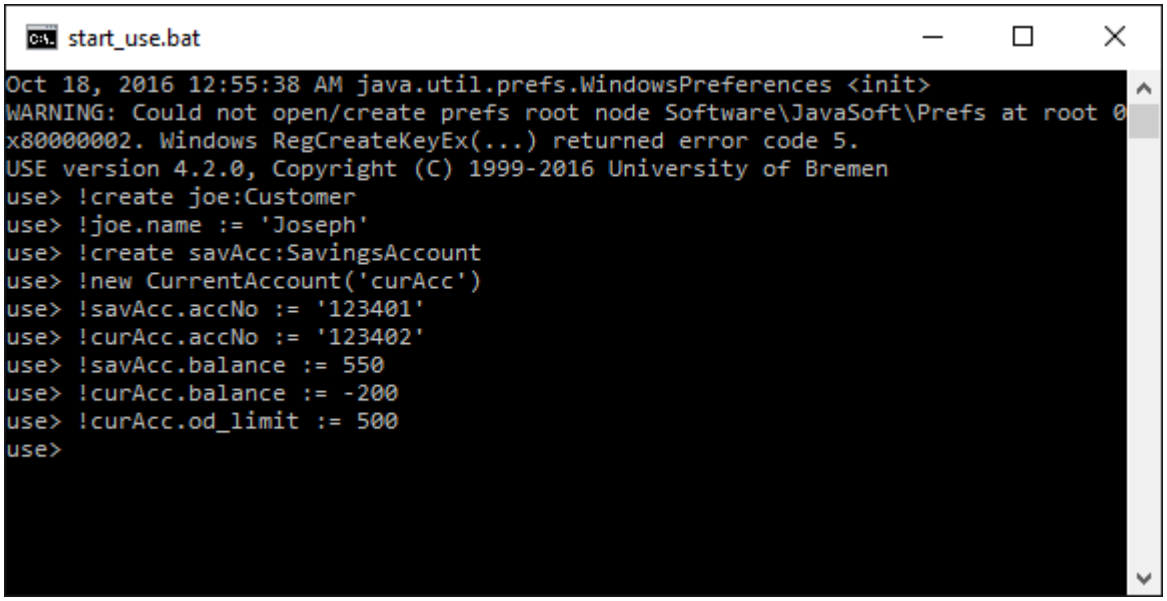
```
!create joe:Customer
!joe.name := 'Joseph'
```



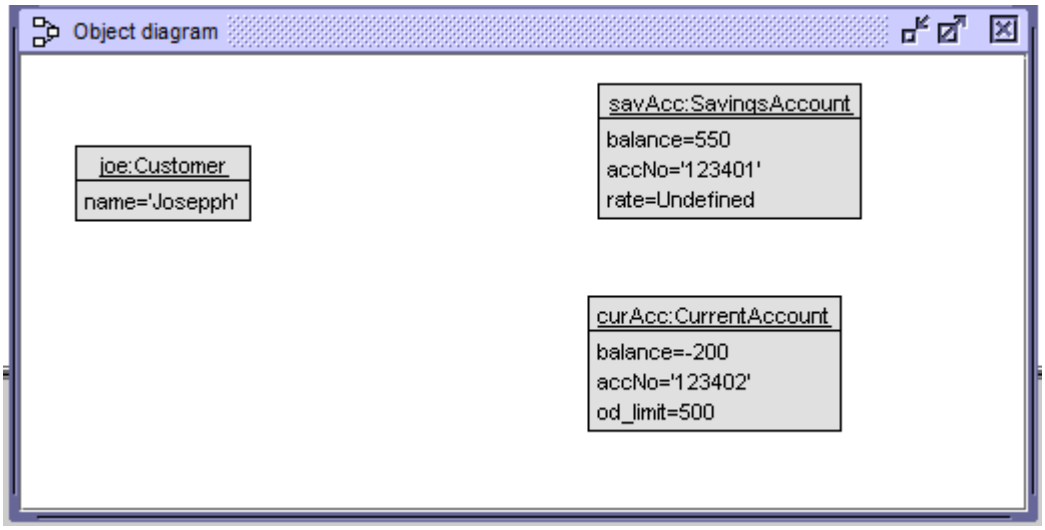
In Use click on the **Create object diagram view** icon  to get an object diagram like:



Enter more USE command line commands as shown next:



Then get the object diagram to show attributes and reorganise its layout. Right-click and select **Save layout** and save the layout with the name bank.

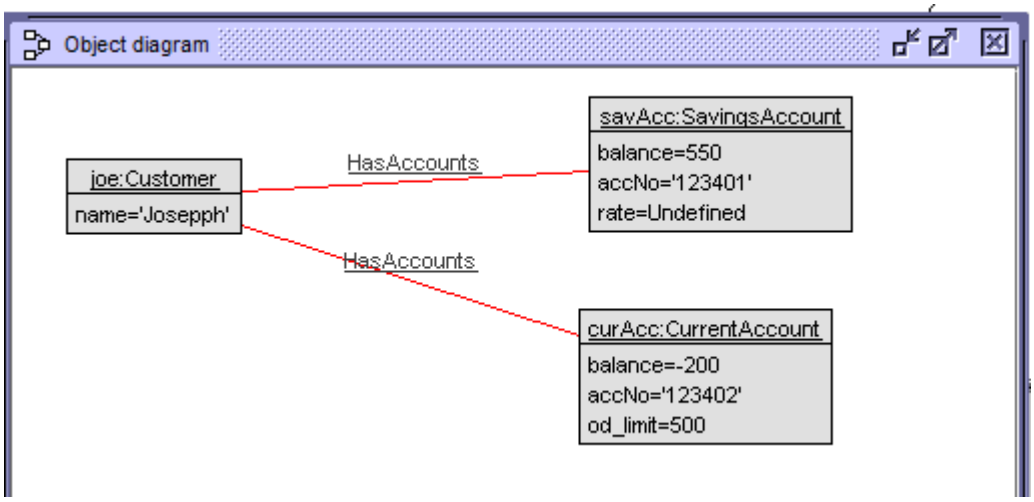


Association between Objects

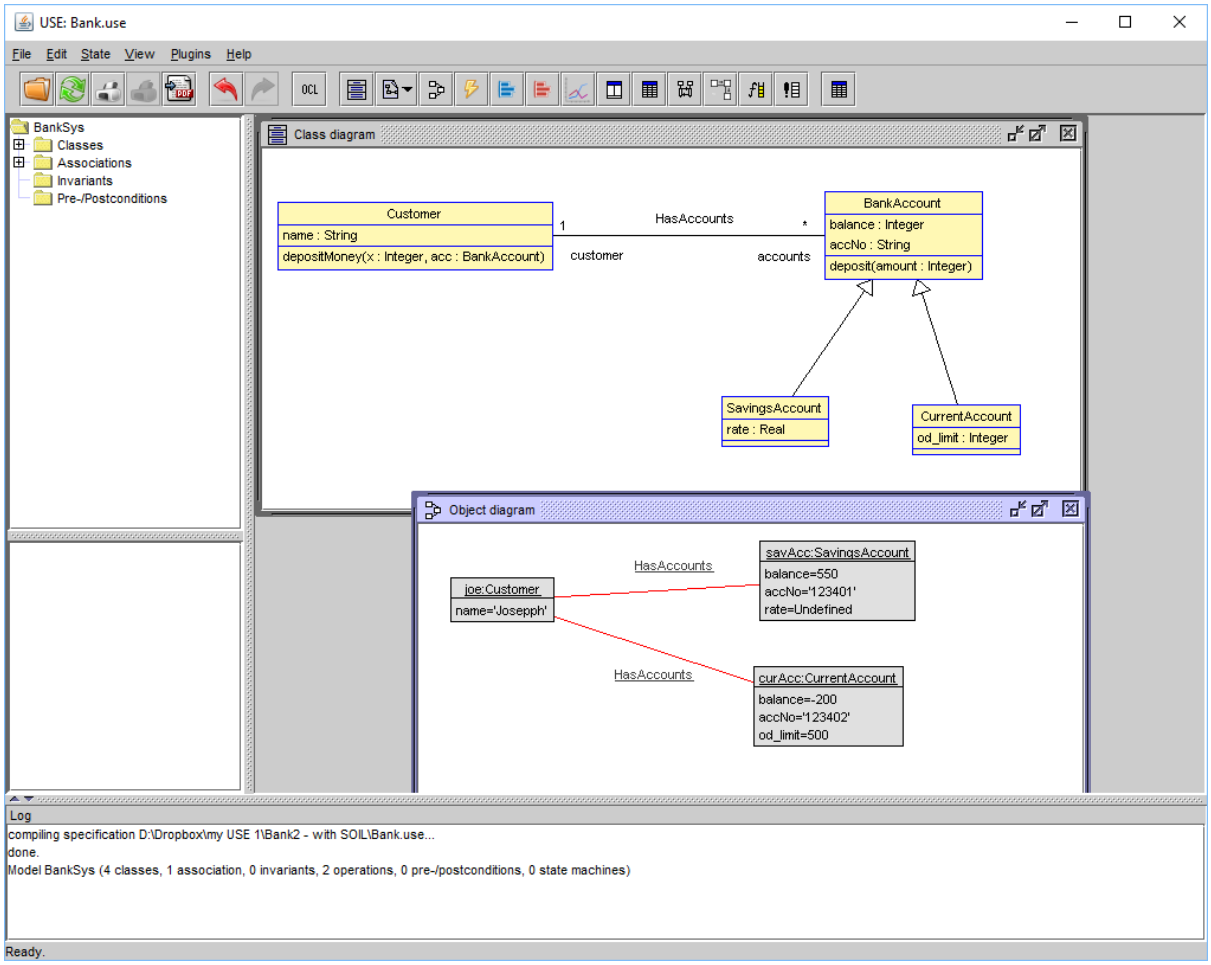
Finally we wish to link these bank account objects with joe. Use the command line to do this with:

```
!insert (joe, savAcc) into HasAccounts
!insert (joe, curAcc) into HasAccounts
```

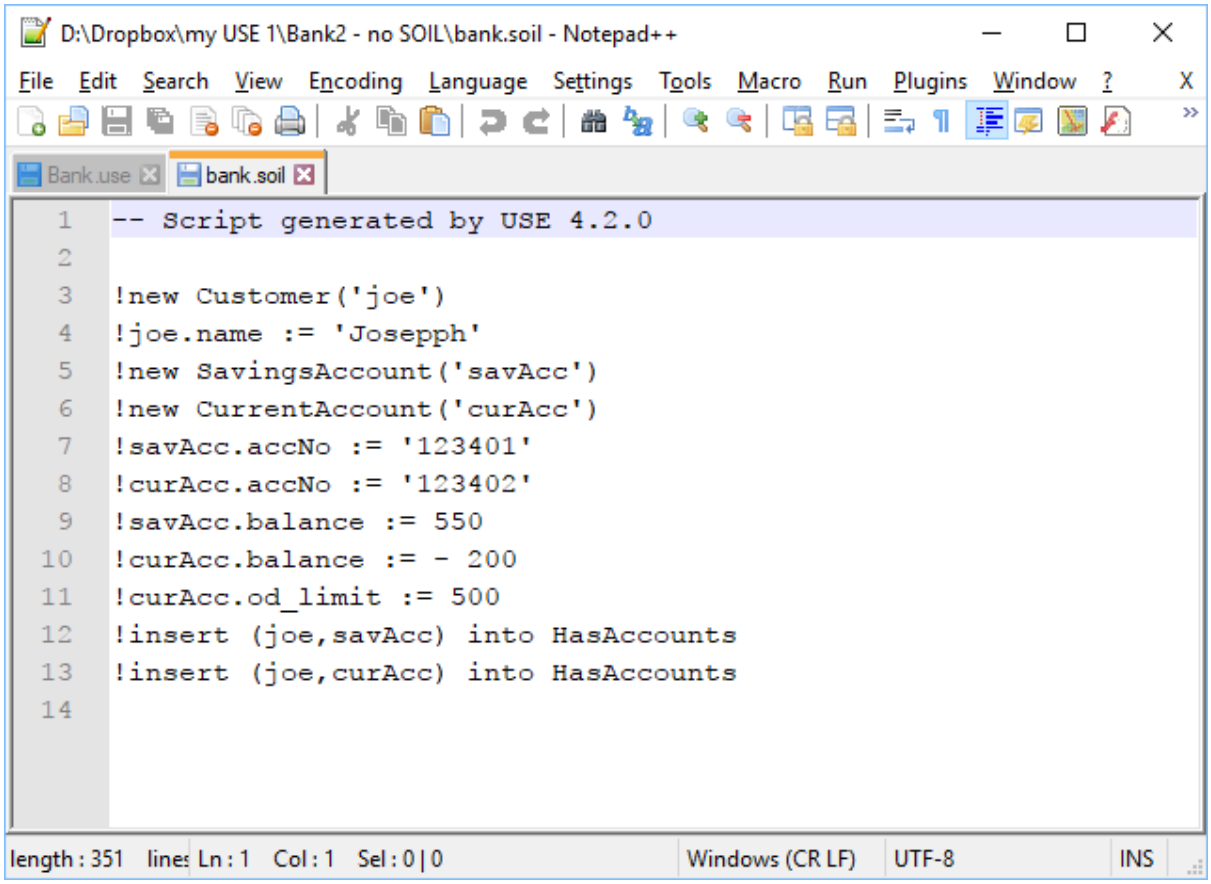
Object diagram will now look like:



USE itself may look like:



Next use the menu **File | Save script (.soil)** to save your object creation and modification commands. Save them to a file called **bank.soil**. This will allow you to relay your object at a later date. Using Notepad++ have a look inside this .soil file. Should be like:



Exercise

Create another customer called anne and a savings account for her with a balance of 1100. Link the two objects and rearrange the Object diagram.

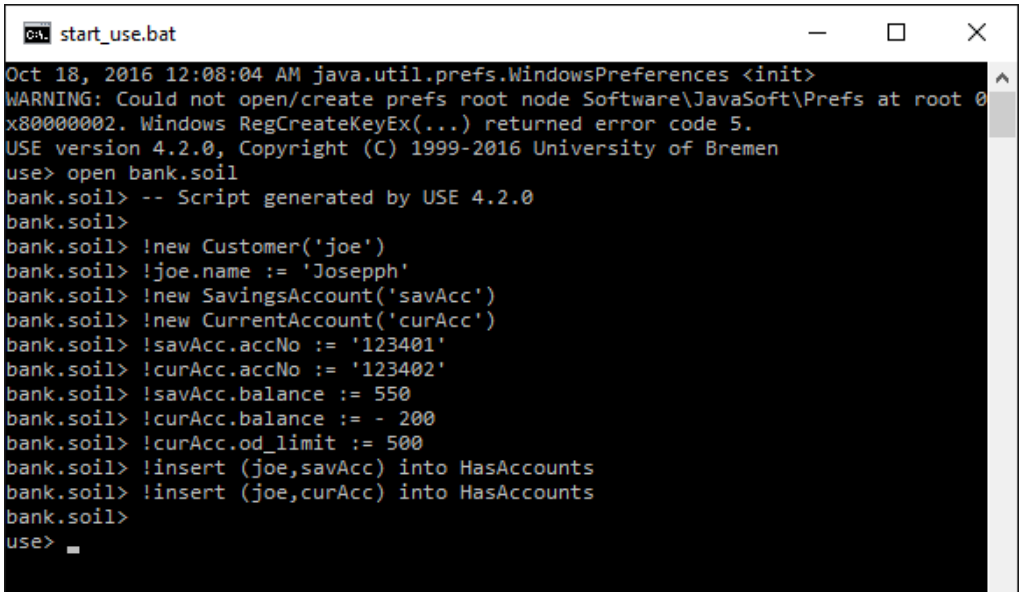
Add SOIL code to USE Spec

SOIL is a simple and unspectacular but complete imperative language that can be used to operationally specify UML models (i. e., to program or animate UML models). SOIL is rather lightweight and does not aim to compete against general purpose languages such as Java or C#.

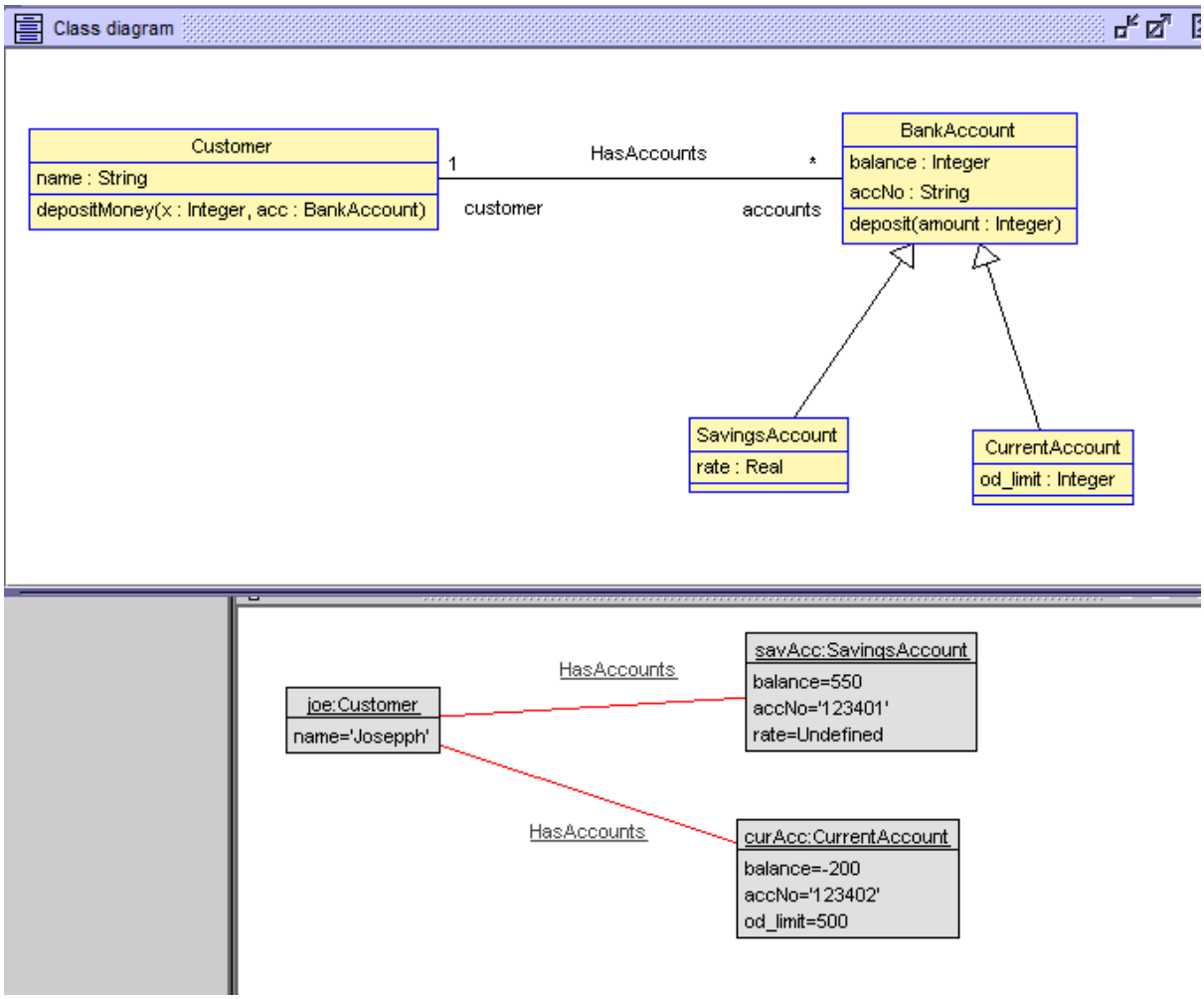
Here we provide a SOIL implementation of the operations/methods depositMoney() and deposit() in the Customer and BankAccount classes. Use Notepad++ to open **bank.use** which you created previously. Then modify the Customer and BankAccount classes as shown next.

```
6   class Customer
7     attributes
8       name : String
9     operations
10      depositMoney( x:Integer, acc:BankAccount)
11      begin
12          acc.deposit(x)
13      end
14  end
15
16  class BankAccount
17    attributes
18      balance : Integer
19      accNo : String
20    operations
21      deposit( amount : Integer)
22      begin
23          self.balance := self.balance + amount
24      end
25
26      getBalance() : Integer
27      begin
28          result := self.balance;
29      end
30  end
```

Reload **bank.use**. At the command or terminal window, type: `open bank.soil` as show below. This loads the object commands used and saved previously for creating and modifying objects.



Then open a class diagram view and an object diagram view and load the corresponding layout files saved in last lab session. USE GUI should look something like:

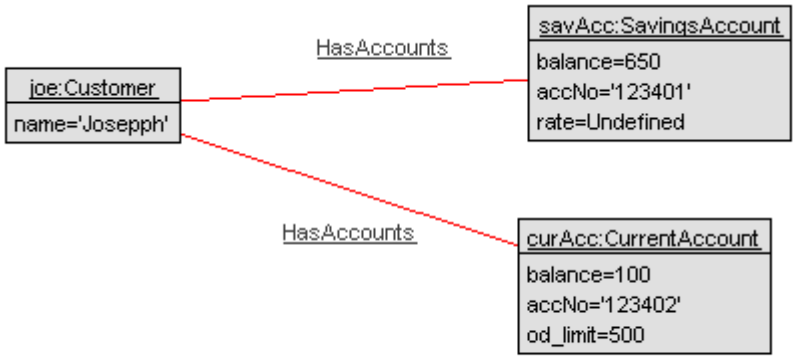


Execute SOIL Operations

At the command window try

```
use> !joe.depositMoney(100, savAcc)
use> !joe.depositMoney(300, curAcc)
use>
```

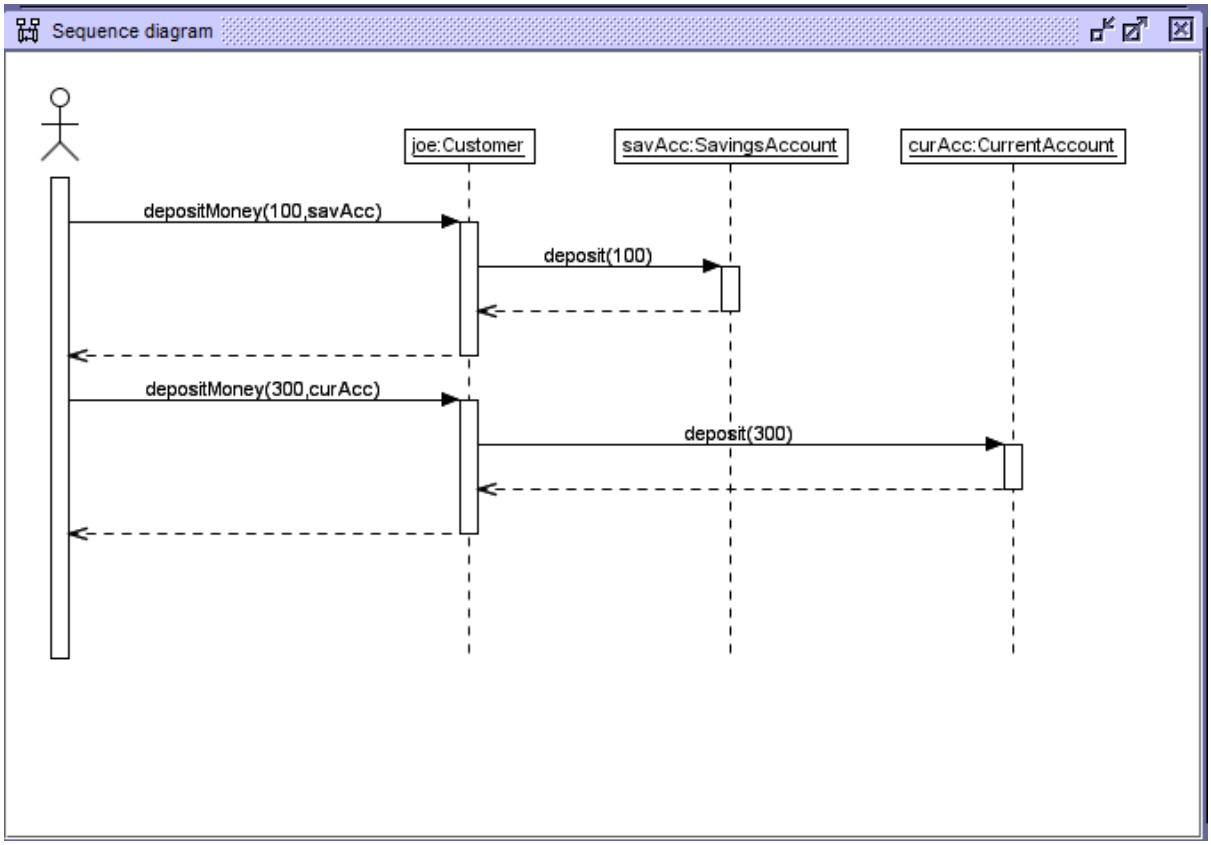
Look at your object diagram. Has Joe's savings account balance been updated? Should now be:



Create a Sequence Diagram

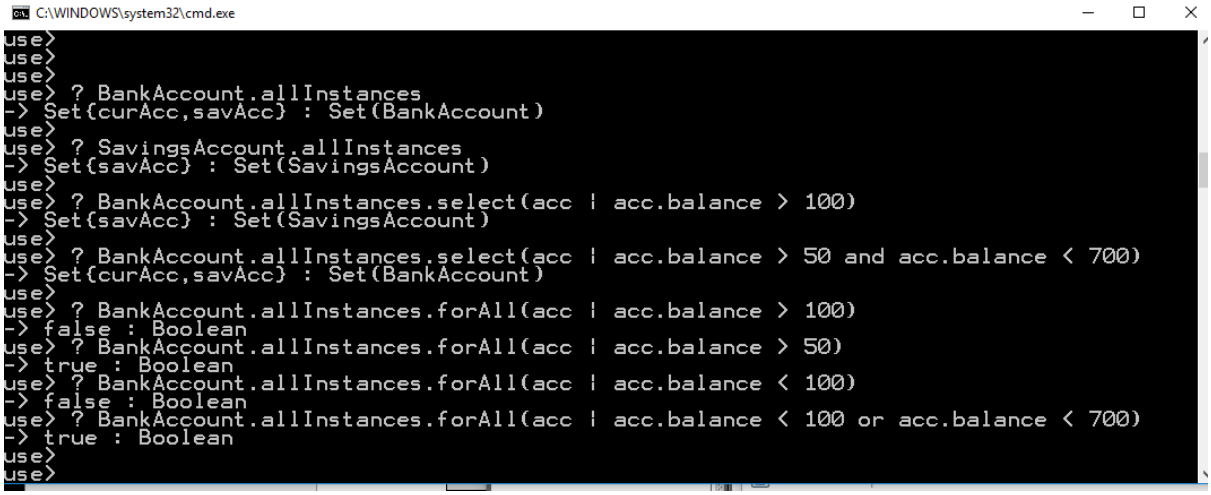
We now create a sequence diagram that shows the object interactions that we cause to be executed with joe.depositMoney(100, savAcc) and joe.depositMoney(300, curAcc).

Use the menu **View | Create view | Sequence diagram**. Should get:



OCL Queries

In the USE command window you can execute OCL queries and they always begin with ?. These do not change the state of the system. Some examples are shown below. Try them.



Exercise

Declare, implement and test the following operations

- Customer class: withdrawMoney(m : Integer, a : BankAccount)
- BankAccount class: withdraw(amount : Integer)

Open a sequence diagram view. Any differences?