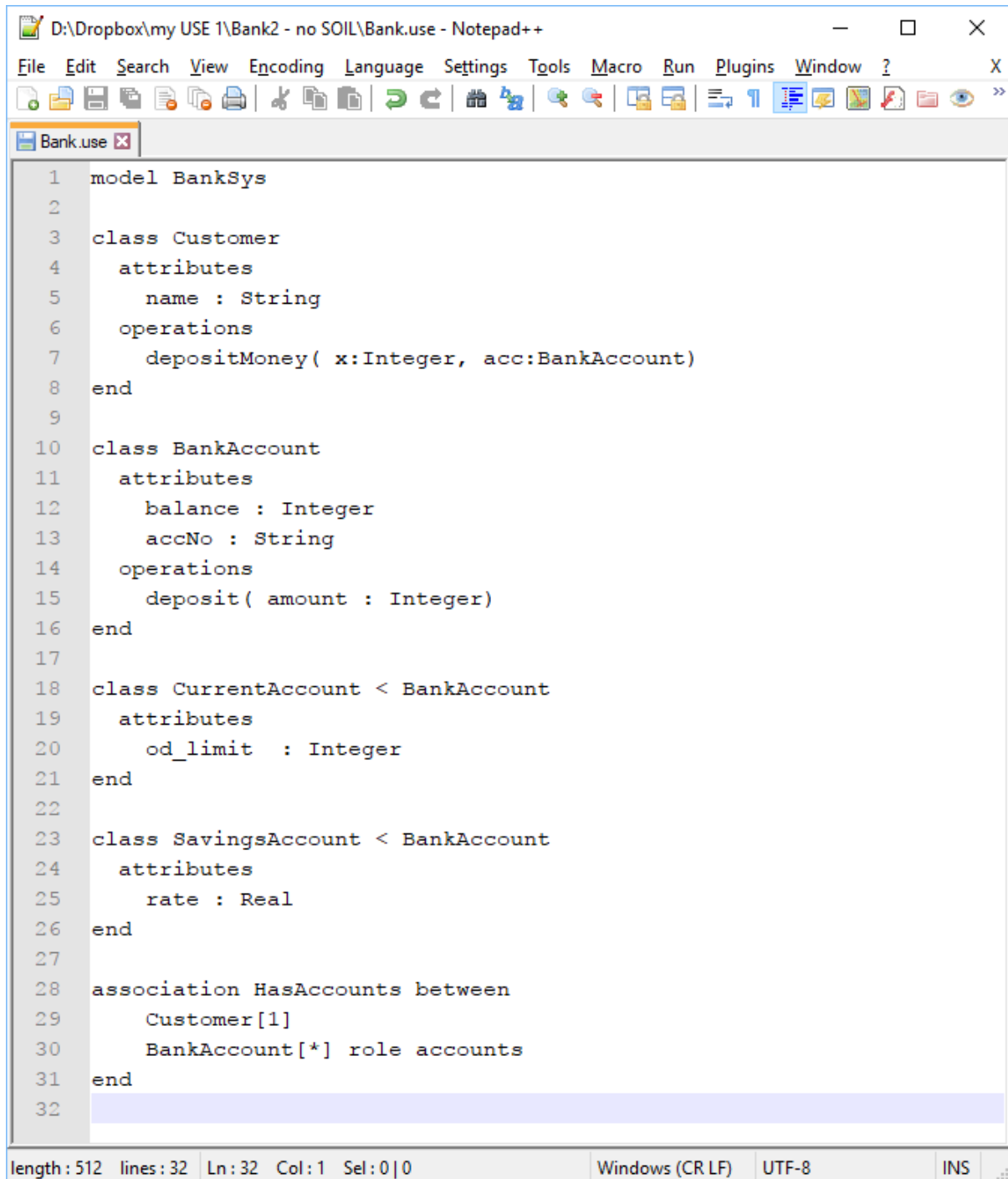


Subclasses, Associations and SOIL in USE

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.

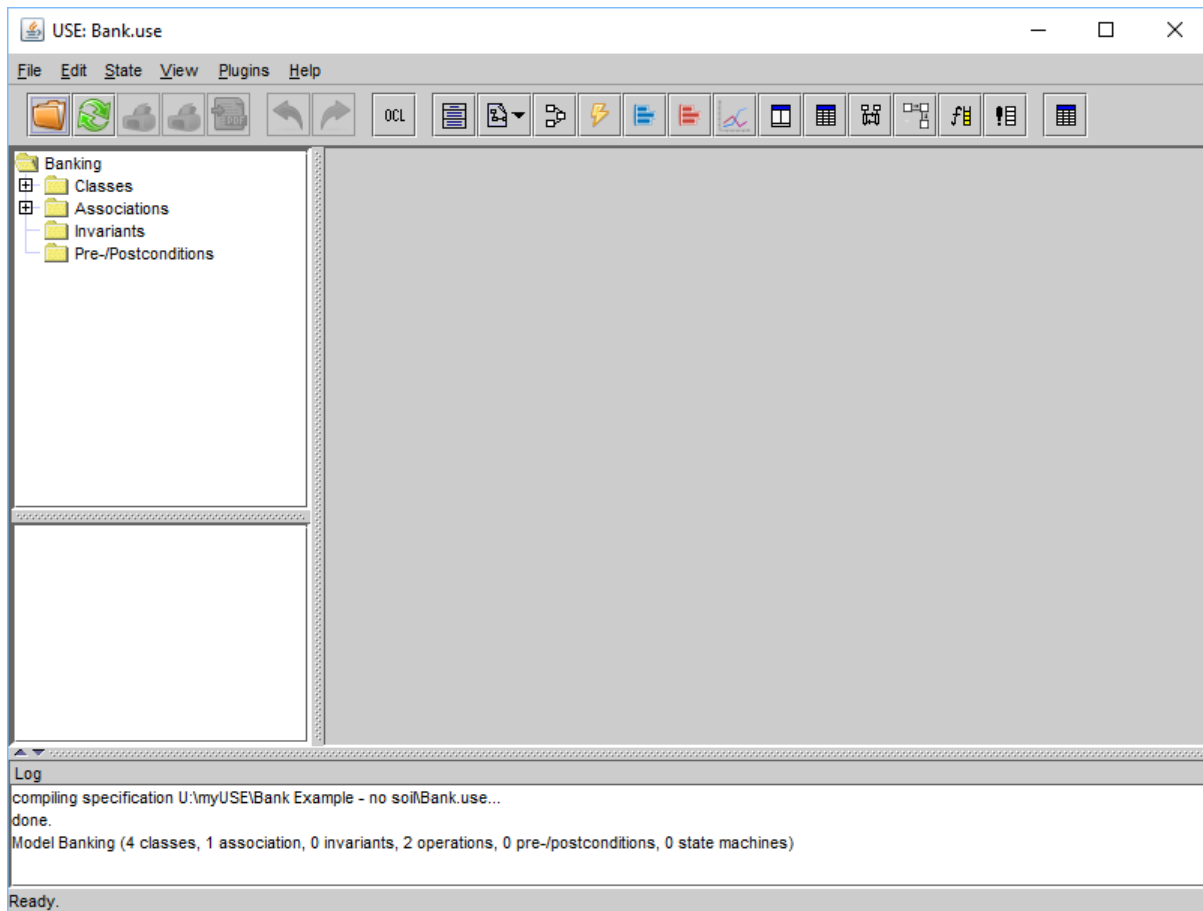
A screenshot of the Notepad++ text editor. The title bar shows the file path 'D:\Dropbox\my USE 1\Bank2 - no SOIL\Bank.use - Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The main text area shows the following UML Use Case Model code:


```
1 model BankSys
2
3 class Customer
4   attributes
5     name : String
6   operations
7     depositMoney( x:Integer, acc:BankAccount)
8 end
9
10 class BankAccount
11   attributes
12     balance : Integer
13     accNo : String
14   operations
15     deposit( amount : Integer)
16 end
17
18 class CurrentAccount < BankAccount
19   attributes
20     od_limit : Integer
21 end
22
23 class SavingsAccount < BankAccount
24   attributes
25     rate : Real
26 end
27
28 association HasAccounts between
29   Customer[1]
30   BankAccount[*] role accounts
31 end
32
```

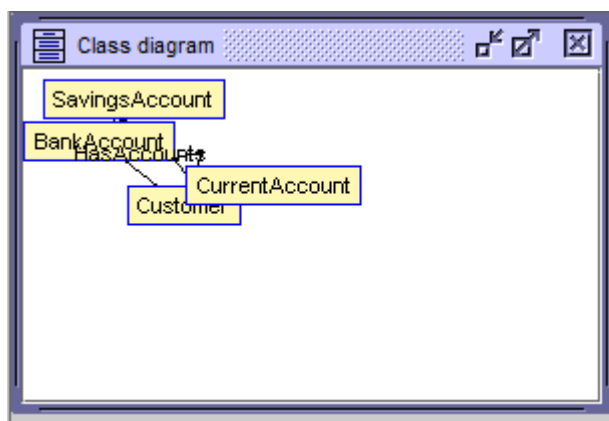
The status bar at the bottom indicates 'length: 512 lines: 32 Ln: 32 Col: 1 Sel: 0 | 0', 'Windows (CR LF)', 'UTF-8', and 'INS'.

Load Specification into USE

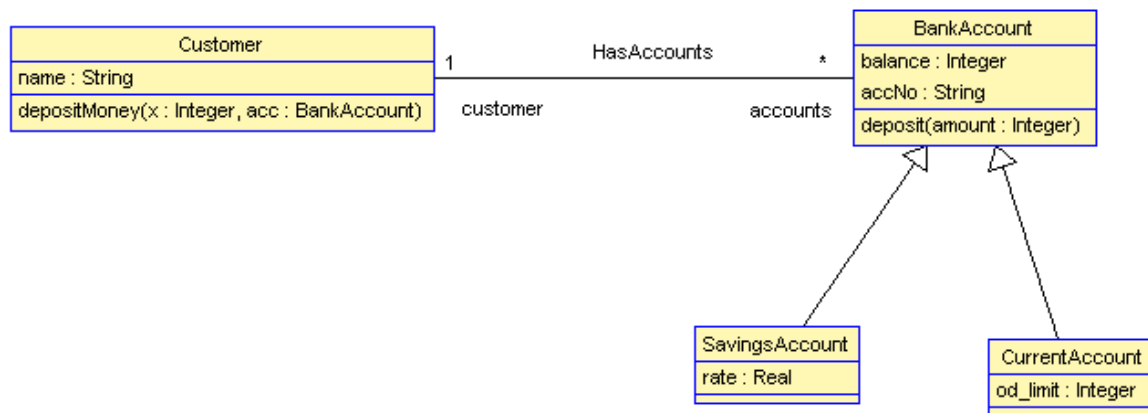
Then start USE and use the USE menu **File | Open Specification** to load your modified USE code. If successful, USE GUI will look like below. If not you have errors in your code.



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 and selecting **Save layout**. Save it with the name **bank**. 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

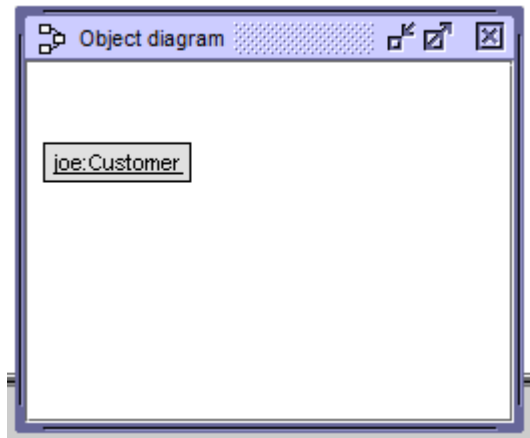
```
!create joe:Customer
!joe.name := 'Joseph'
```

The screenshot shows a command window titled `start_use.bat`. The output text is as follows:

```

Oct 18, 2016 12:55:38 AM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0
x80000002. Windows RegCreateKeyEx(...) returned error code 5.
USE version 4.2.0, Copyright (C) 1999-2016 University of Bremen
use> !create joe:Customer
use> !joe.name := 'Joseph'
use>
  
```

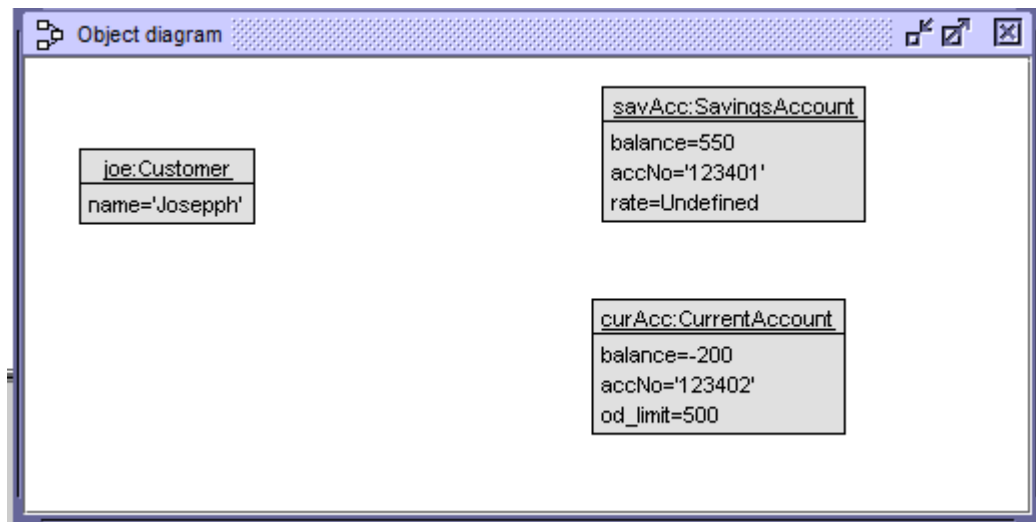
In Use click on the **Create object diagram view** icon  to get an object diagram like:



Enter more USE command line commands as shown next:

```
Oct 18, 2016 12:55:38 AM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0
x80000002. Windows RegCreateKeyEx(...) returned error code 5.
USE version 4.2.0, Copyright (C) 1999-2016 University of Bremen
use> !create joe:Customer
use> !joe.name := 'Joseph'
use> !create savAcc:SavingsAccount
use> !new CurrentAccount('curAcc')
use> !savAcc.accNo := '123401'
use> !curAcc.accNo := '123402'
use> !savAcc.balance := 550
use> !curAcc.balance := -200
use> !curAcc.od_limit := 500
use>
```

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**.

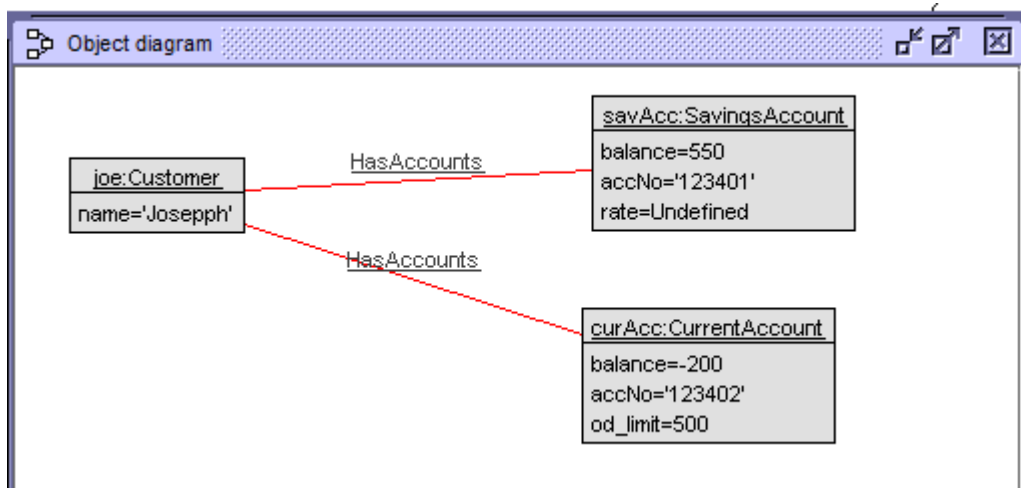


Association between Objects

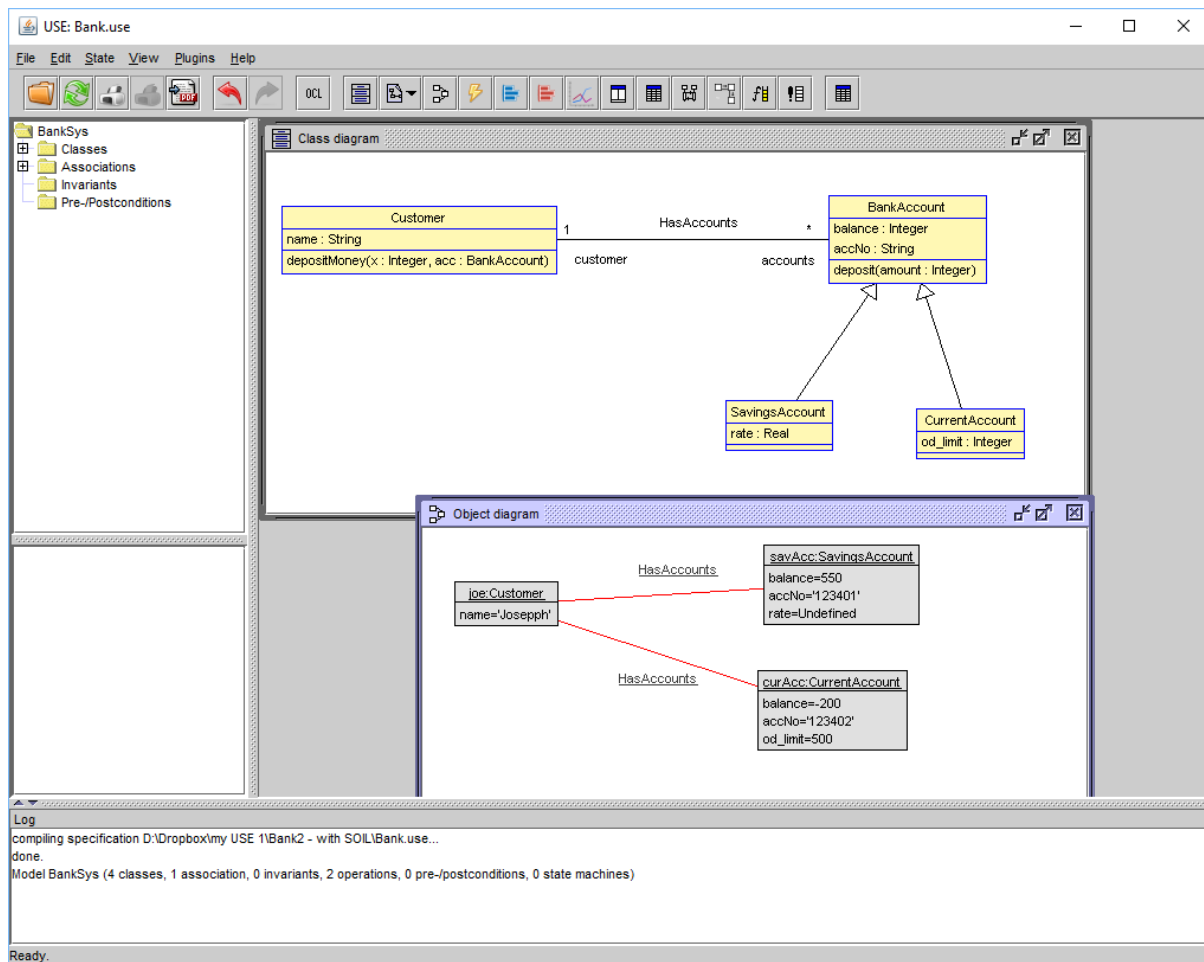
Finally we wish to link these bank account objects with joe. Use the command line to do this with:

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

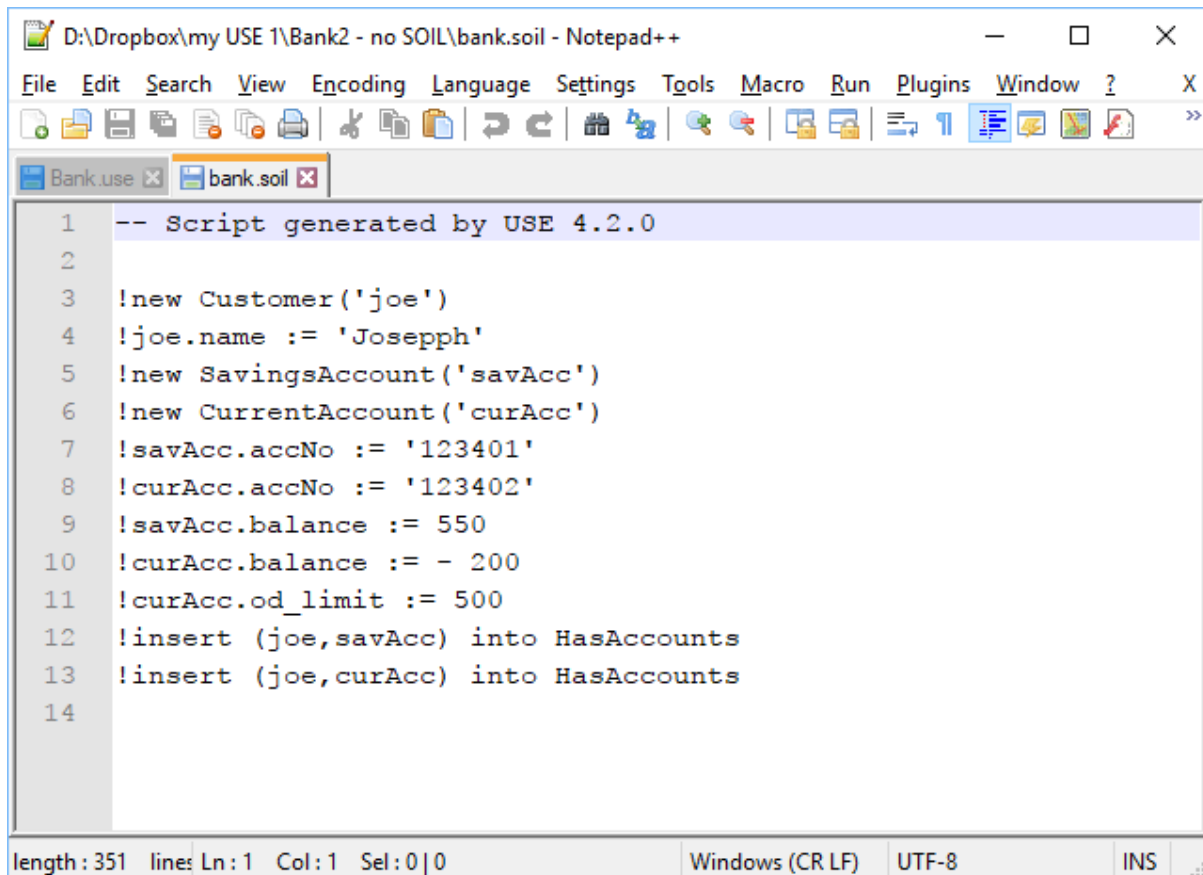
Object diagram will now look like:



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:

A screenshot of the Notepad++ application window. The title bar reads "D:\Dropbox\my USE 1\Bank2 - no SOIL\bank.soil - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The tab bar shows two tabs: "Bank.use" and "bank.soil", with "bank.soil" being the active tab. The text area contains 14 lines of code. Line 1 is a comment: "-- Script generated by USE 4.2.0". Lines 3-13 are commands to create and configure objects: !new Customer('joe'), !joe.name := 'Josephph', !new SavingsAccount('savAcc'), !new CurrentAccount('curAcc'), !savAcc.accNo := '123401', !curAcc.accNo := '123402', !savAcc.balance := 550, !curAcc.balance := - 200, !curAcc.od_limit := 500, !insert (joe,savAcc) into HasAccounts, and !insert (joe,curAcc) into HasAccounts. Line 14 is empty. The status bar at the bottom shows "length : 351 lines Ln : 1 Col : 1 Sel : 0 | 0", "Windows (CR LF)", "UTF-8", and "INS".

```
1  -- Script generated by USE 4.2.0
2
3  !new Customer('joe')
4  !joe.name := 'Josephph'
5  !new SavingsAccount('savAcc')
6  !new CurrentAccount('curAcc')
7  !savAcc.accNo := '123401'
8  !curAcc.accNo := '123402'
9  !savAcc.balance := 550
10 !curAcc.balance := - 200
11 !curAcc.od_limit := 500
12 !insert (joe,savAcc) into HasAccounts
13 !insert (joe,curAcc) into HasAccounts
14
```

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

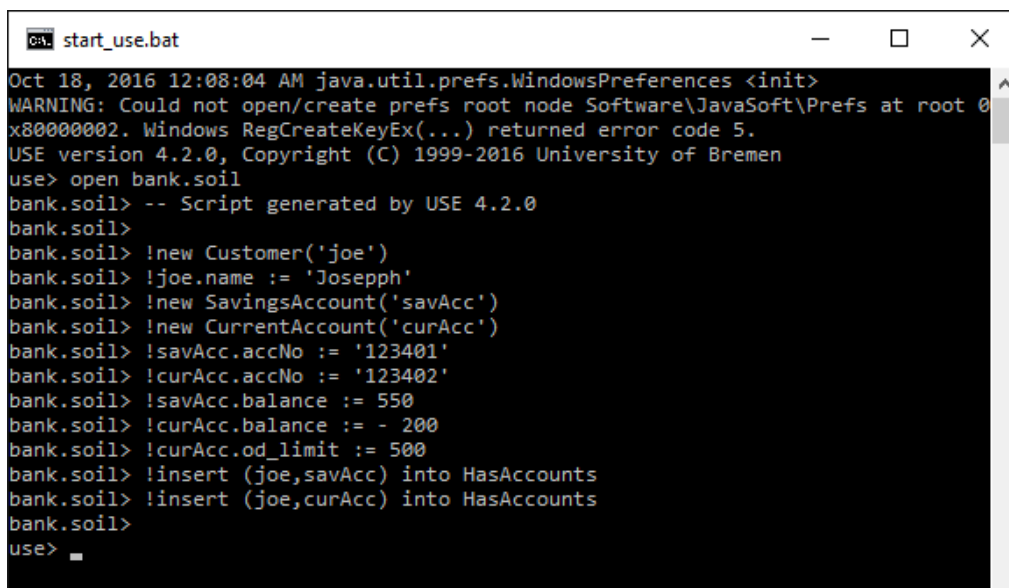
SOIL is a simple and unspectacular but complete imperative language that can be used to operationally specify UML models (i. e., to program or animate UML models). SOIL is rather lightweight and does not aim to compete against general purpose languages such as Java or C#.

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.

```
class Customer
  attributes
    name : String
  operations
    depositMoney( x:Integer, acc:BankAccount)
    begin
      .....
      acc.deposit(x)
    end
end

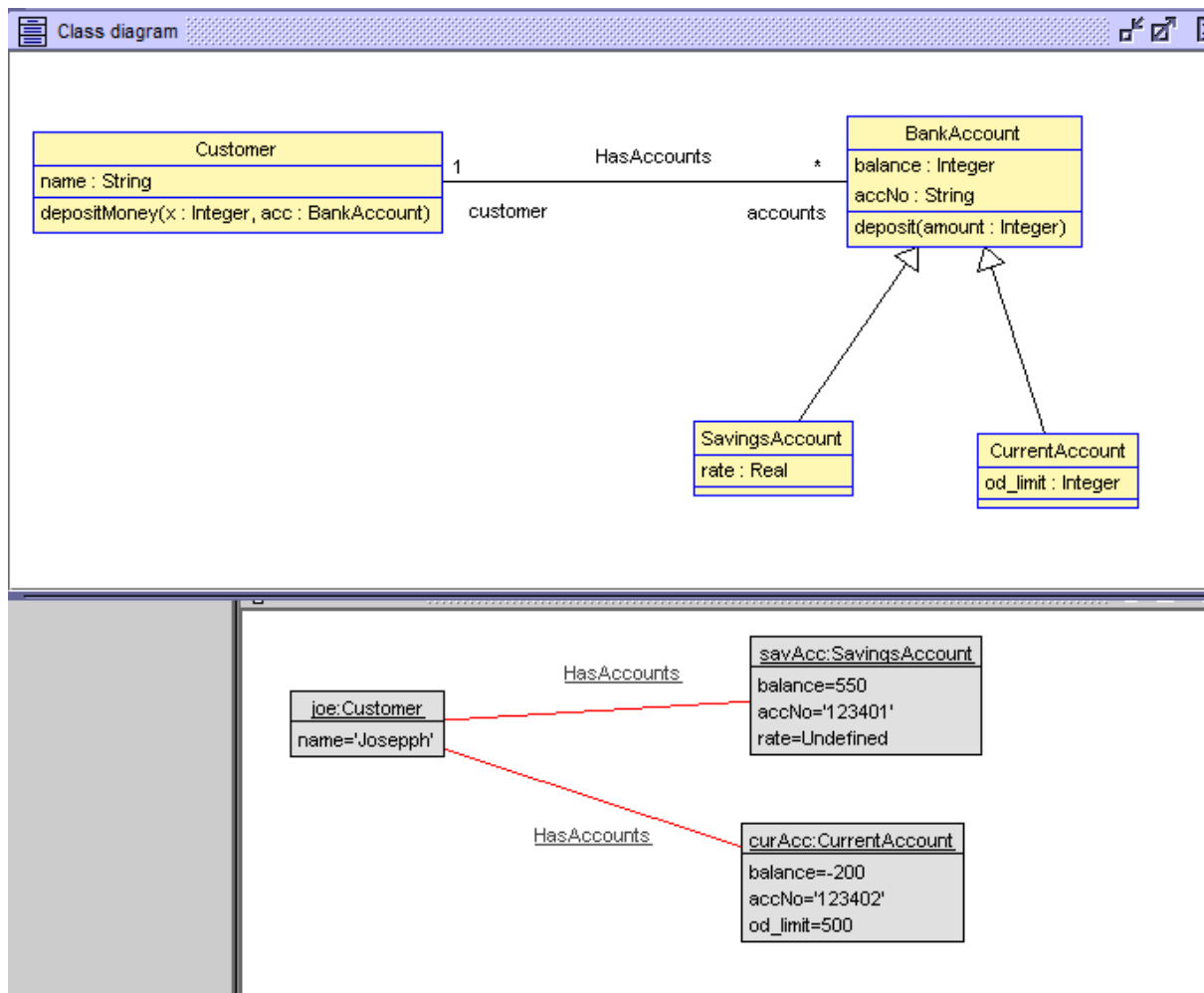
class BankAccount
  attributes
    balance : Integer
    accNo : String
  operations
    deposit( amount : Integer)
    begin
      .....
      self.balance := self.balance + amount
    end
end
```

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



```
Oct 18, 2016 12:08:04 AM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0
x80000002. Windows RegCreateKeyEx(...) returned error code 5.
USE version 4.2.0, Copyright (C) 1999-2016 University of Bremen
use> open bank.soil
bank.soil> -- Script generated by USE 4.2.0
bank.soil>
bank.soil> !new Customer('joe')
bank.soil> !joe.name := 'Josepph'
bank.soil> !new SavingsAccount('savAcc')
bank.soil> !new CurrentAccount('curAcc')
bank.soil> !savAcc.accNo := '123401'
bank.soil> !curAcc.accNo := '123402'
bank.soil> !savAcc.balance := 550
bank.soil> !curAcc.balance := - 200
bank.soil> !curAcc.od_limit := 500
bank.soil> !insert (joe,savAcc) into HasAccounts
bank.soil> !insert (joe,curAcc) into HasAccounts
bank.soil>
use> _
```


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:

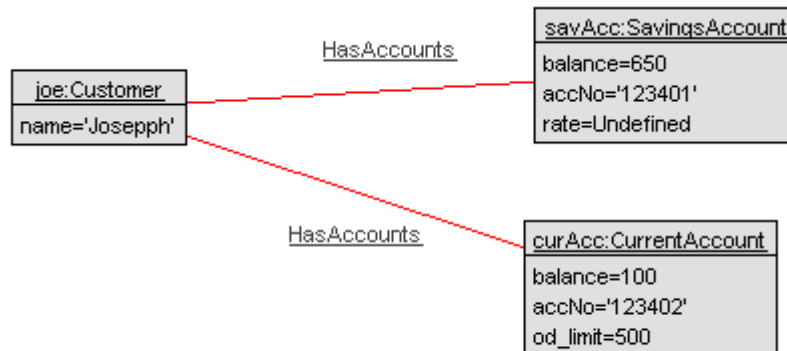


Execute SOIL Operations

At the command window try

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

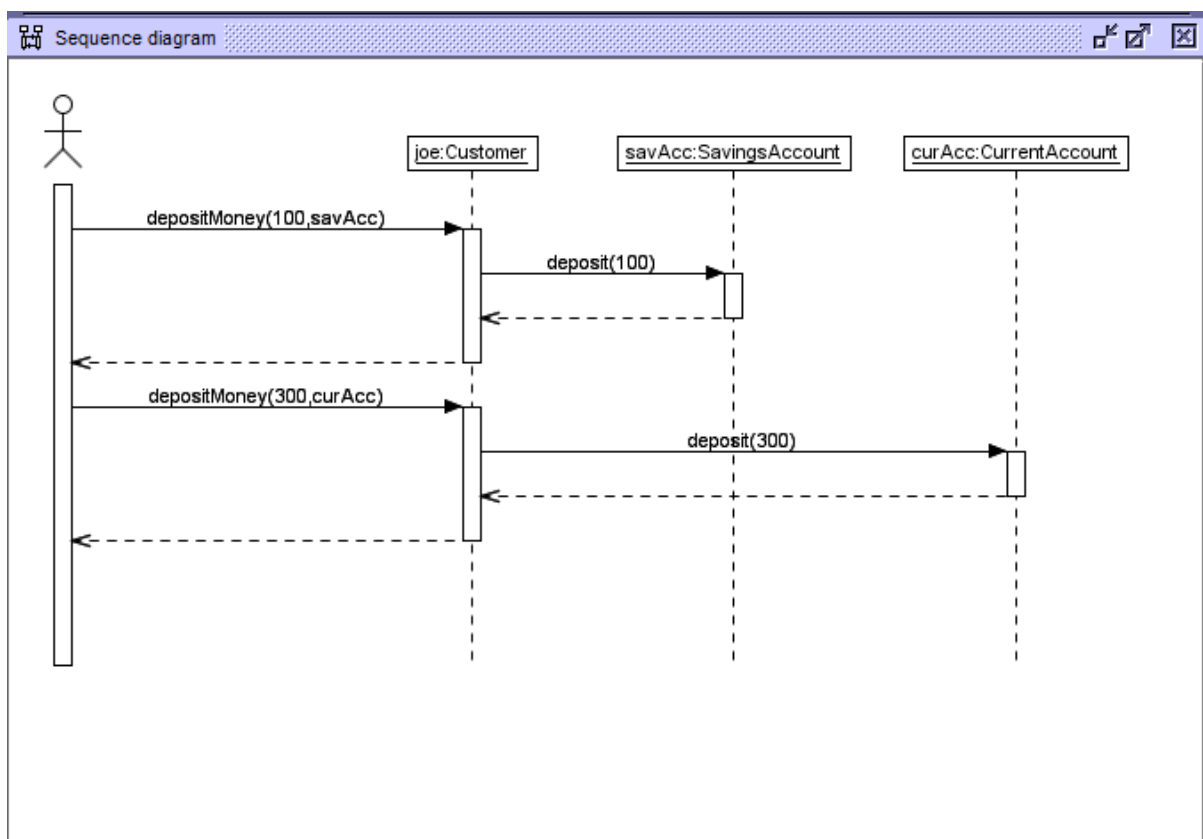
Look at your object diagram. Has Joe's savings account balance been updated? Should now be:



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:



Exercise

Declare, implement and test operations

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

Open a sequence diagram view. Any differences?