Introduction to USE - Classes & Objects in USE

USE (UML Specification Environment) provides a simple way to animate or test-drive a UML specification (e.g. a class diagram). It allows you to create objects and get them to send messages to one another to test their interactions. It is much easier to do this in an **action language** like that available in USE, called SOIL, than in a conventional OOP language like Java. Bringing a UML model to life in this way deepens the specifiers understanding of the requirements and points the way to an initial design for the eventual software.

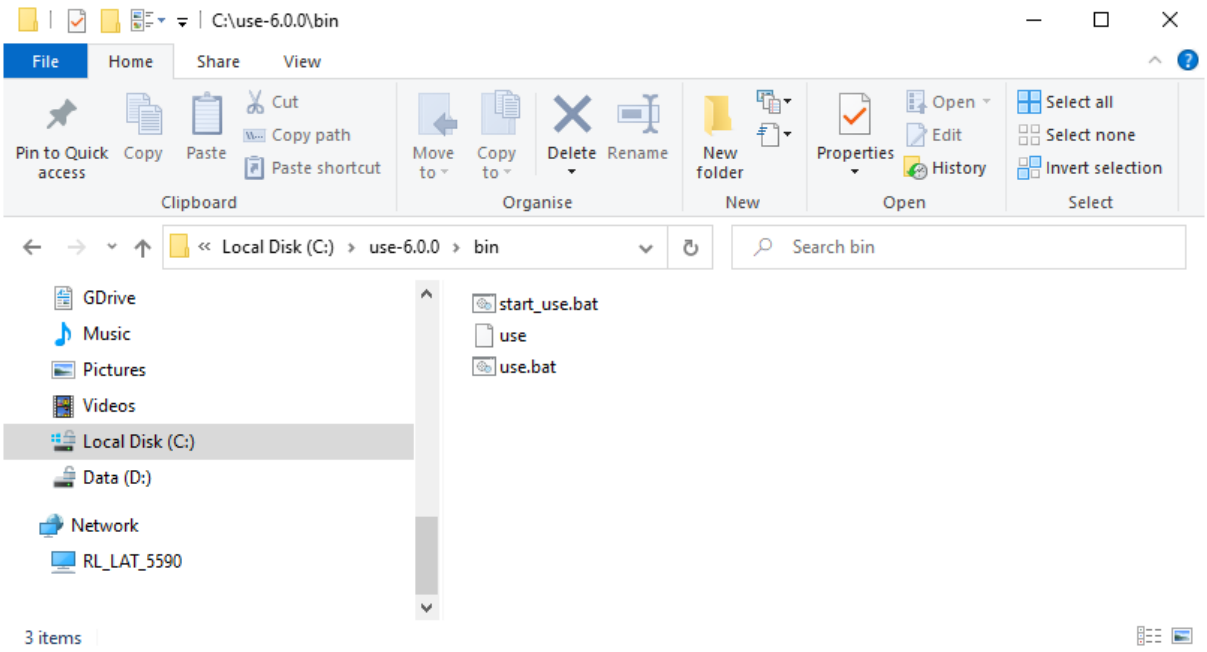
To see how you might begin to do this in USE, try the following tutorial.

Install and Start USE Program

Once you have USE extracted from the zip file, use Windows **File Explorer** to navigate to the folder that you extracted USE to.

Then navigate to the bin subfolder of the USE folder, i.e. to C:\use-6.0.0\bin for this particular version of USE.

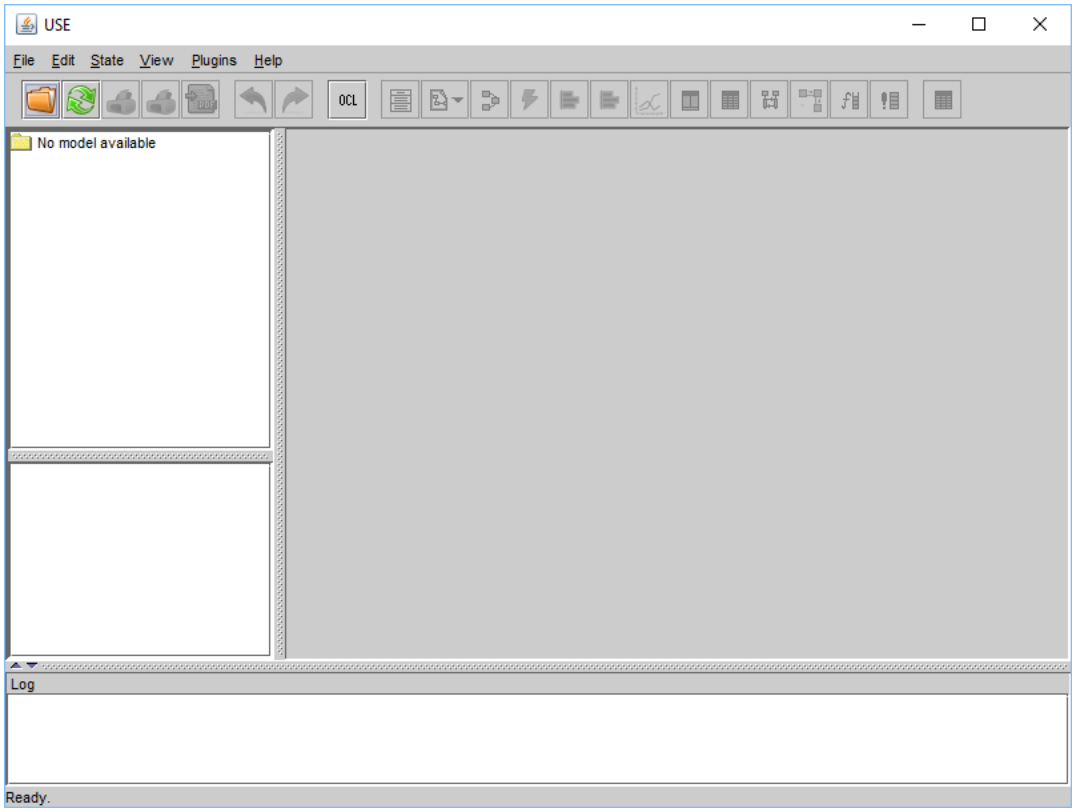
You will see something like:



Double click on **start_use.bat**.

For this to work, the Java **bin** folder (e.g. `C:\Program Files\Java\jdk1.7.0_51\bin`) must be listed on the **Path** environment variable. Settings in Windows will allow you to do this.

If using OS X, click on **use** in the **bin** folder. The USE GUI should initially look like:



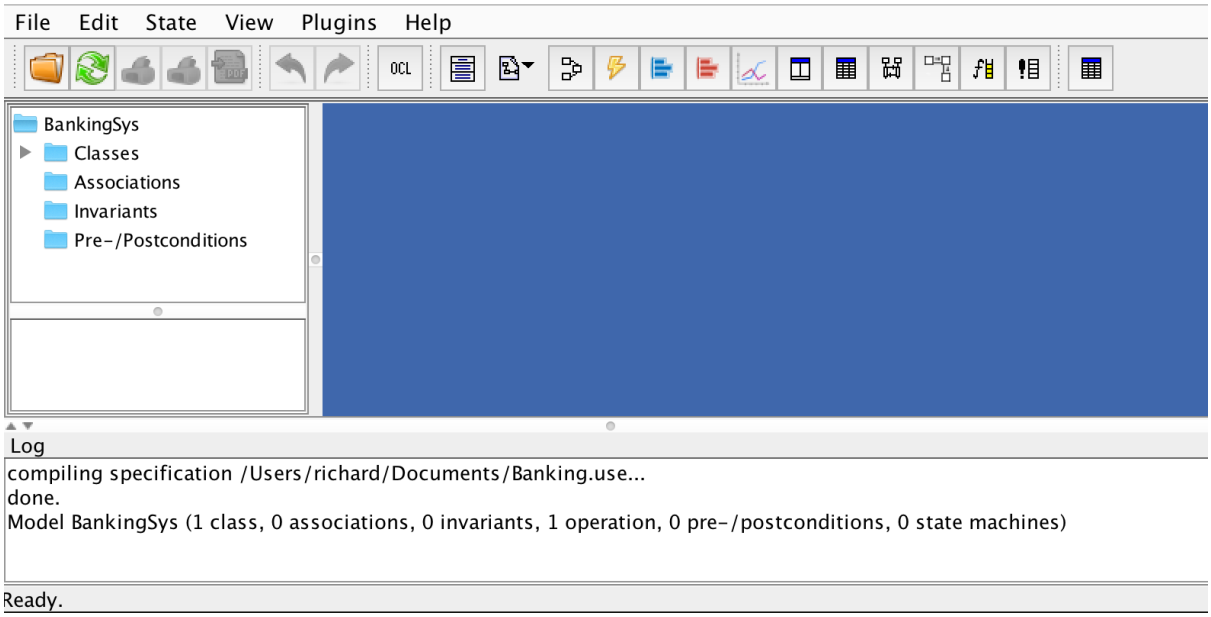
Coding BankAccount Class in USE

Use a text editor to write the following USE code and save it to a file called **Banking.use** in a a folder called Bank. Later on put any related files from the tutorial in this folder.

```
model BankingSys

class BankAccount
   attributes
     balance : Integer
     accountNo : String
   operations
     deposit(amount : Integer)

end
```

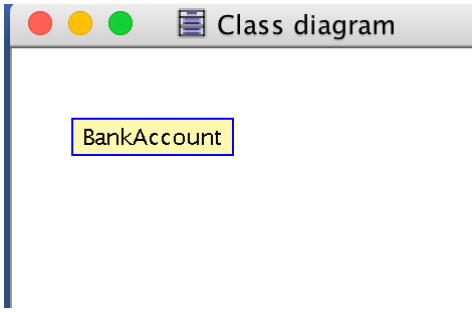
Then, assuming you have USE running, use the icon  to locate and load your class definition into USE.

If you typed the USE code correctly, you will see something like:

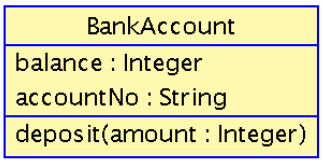


Class Diagram View

To see the class diagram for your USE code use the icon  or the menu **View | Create View | Class diagram**.



Here all that can be seen is the class name, so right-click on the class diagram and select both **show attributes** and **show operations** to get:

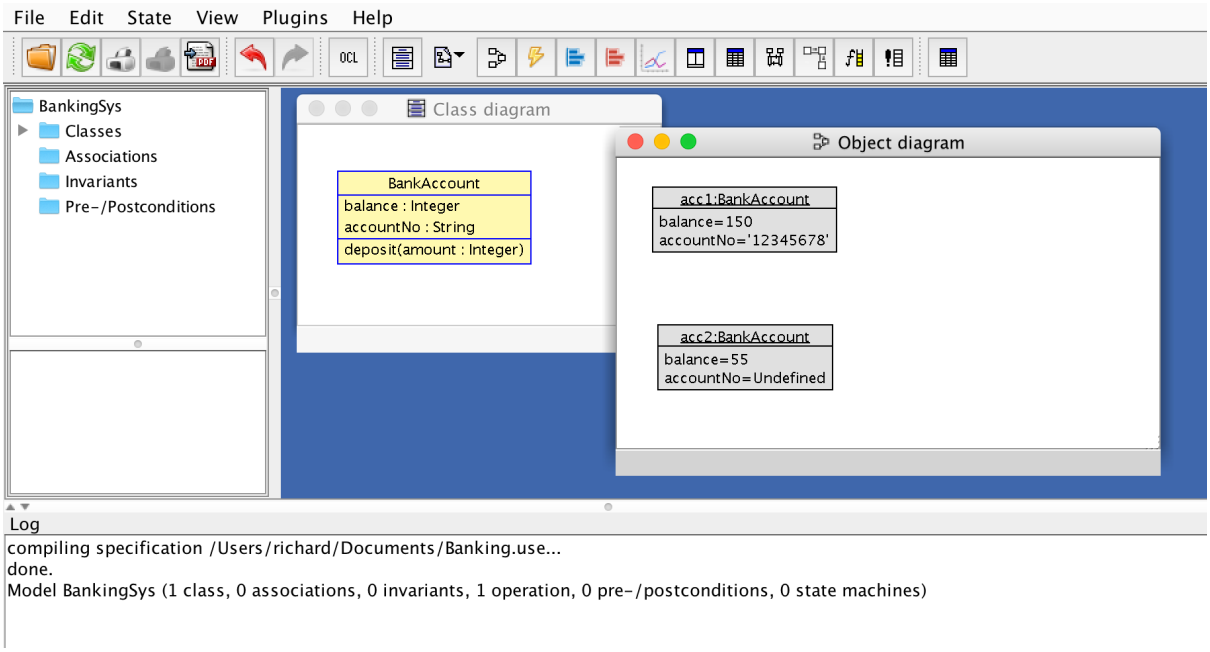


Creating BankAccount Objects

When you start USE a new terminal window (in Linux or OS X) or command prompt window (in Windows) will start alongside the USE GUI. You can enter object creation or modification commands from this terminal window as show below.

```
use> !create acc1:BankAccount
use> !create acc2:BankAccount
use> !acc1.balance := 150
use> !acc2.balance := 55
use> !acc1.accountNo := '12345678'
```

To see these two objects, use the menu **View | Create View | Object diagram** or use the icon  . Right-click on resulting object diagram to see the object attributes. USE should now look like:



We would next like to send a message i.e. invoke an operation on these account objects. To do this the USE code has to be modified and reloaded, so it's better to first save the object creation commands to a text file. This will help you avoid creating them from scratch again once the model is reloaded.

Use the menu **File | Save script (.soil)** and name it **Banking.soil**. Look at the contents of this file using a text editor.

Implementation an Operation in SOIL

Here we will provide a SOIL (Simple OCL-like Imperative Language) "implementation" of the operation **deposit()**.

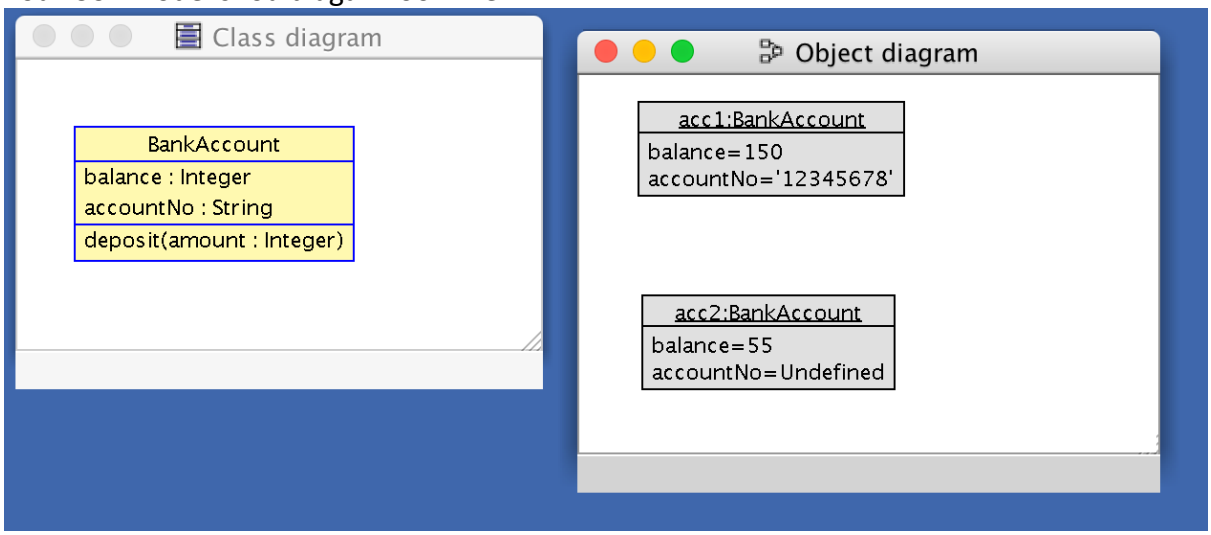
Modify your original USE code as shown below and reload in USE using the icon  .

```
model BankingSys

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

end
```

Then open a class diagram view and an object diagram view. The object diagram will be empty, so re-create you previous objects by typing the command **open Banking.soil** shown below at the terminal window (or command prompt):

```
use> open Banking.soil
Banking.soil> -- Script generated by USE 4.2.0
Banking.soil>
Banking.soil> !new BankAccount('acc1')
Banking.soil> !new BankAccount('acc2')
Banking.soil> !acc1.balance := 150
Banking.soil> !acc2.balance := 55
Banking.soil> !acc1.accountNo := '12345678'
Banking.soil>
```

Your USE model should again look like:

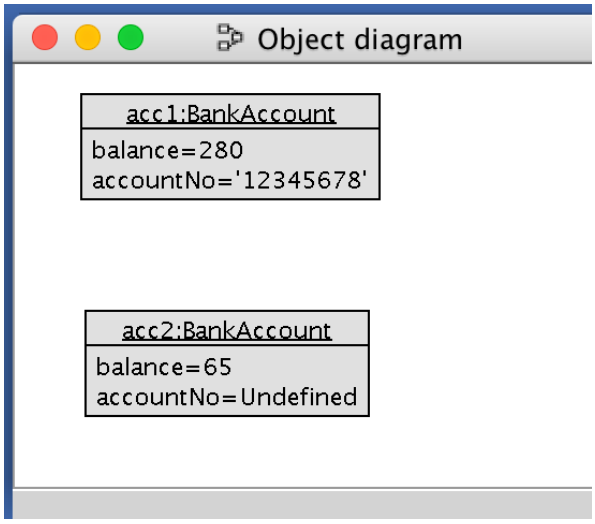


Invoking an Operation in USE

We now have two account objects, let us call execute the operation **deposit()** 3 times as follows:

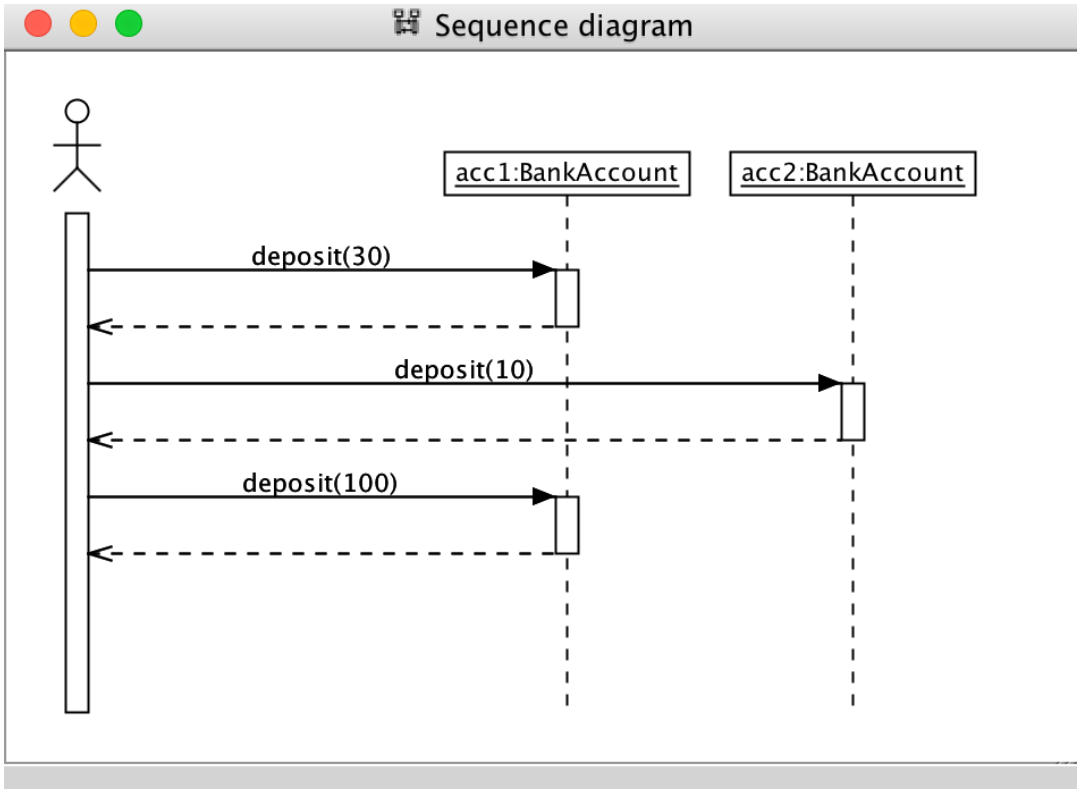
```
use> !acc1.deposit(30)
use> !acc2.deposit(10)
use> !acc1.deposit(100)
use>
```

Take a look at the object diagram. Have the object balances changed?



Sequence Diagram View

To see a diagram of the operation being called on the objects (or the objects receiving a message in OO parlance) click on the icon  or use the menu **View | Create View | Sequence diagram** to get:



Exercise

1. Add a class definition for **Customer** to your USE code and reload it.
2. Reload previous object creation commands and using the command prompt, create a customer object with your own name as the name attribute of the object. Save these object commands as before to **Banking.soil**.