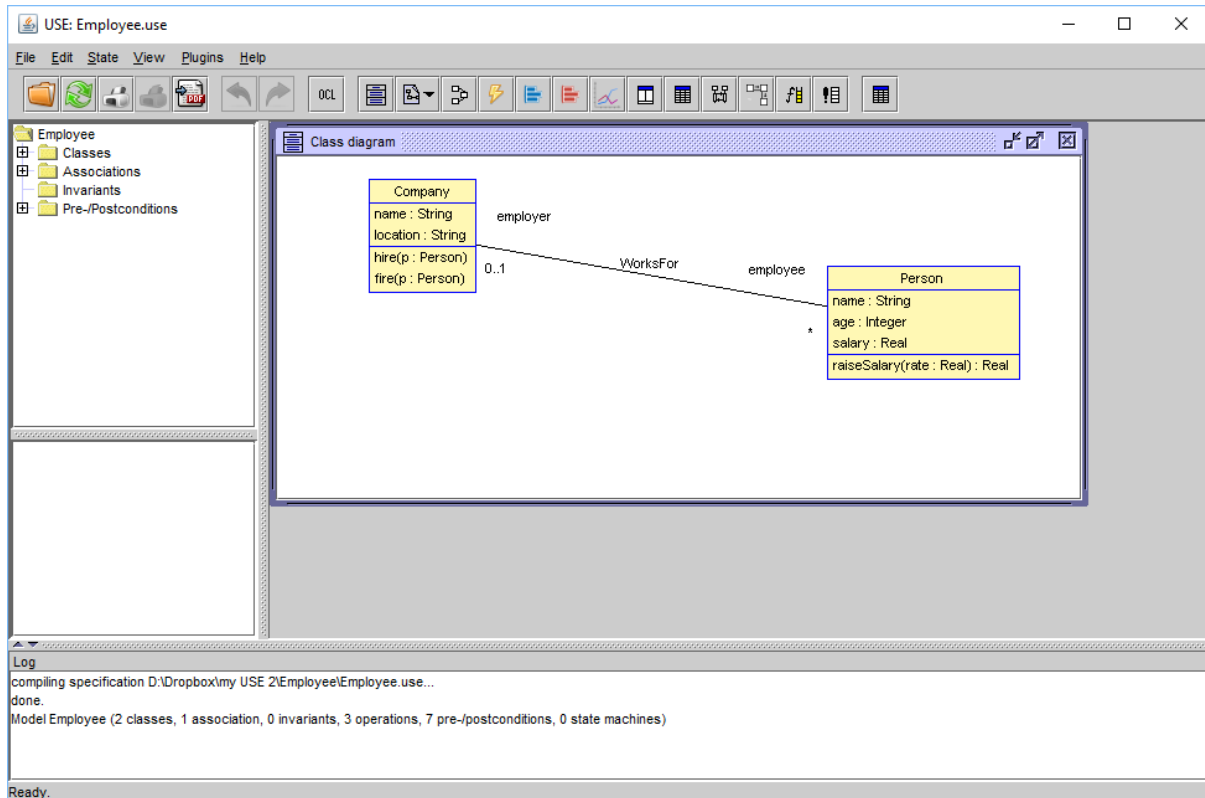


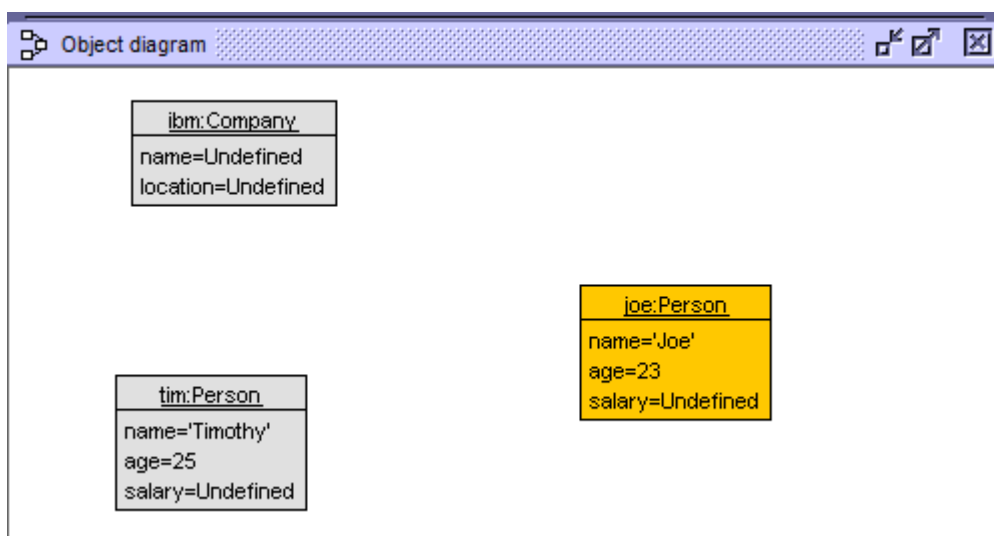
## Constraints in USE

Have a look at the constraints introduced to a simple employee specification in **employee.use**. Here we will try to test them.

Run USE and load **Employee.use** and its class layout to get.



Then create a company **ibm** and two person objects as indicated below.

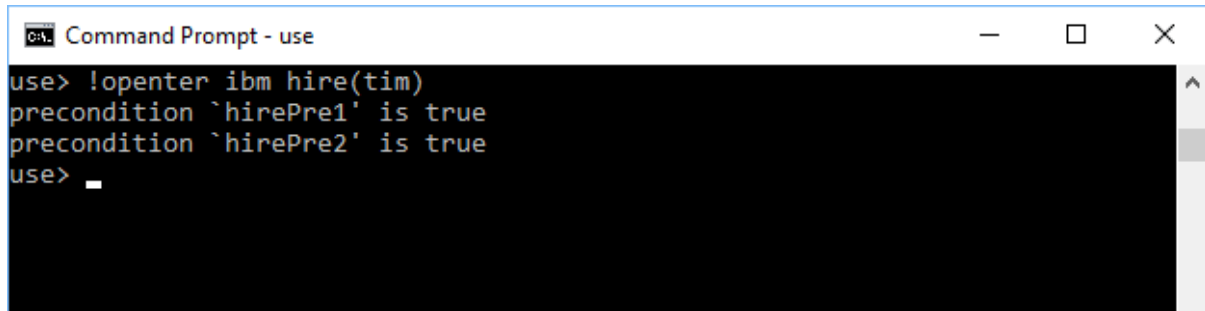


## Testing USE Constraints

Here we use the objects created to test the constraint pre and post conditions which form a contract for the operation **hire()**. We will try to “execute” `ibm.hire(tim)` but recall that `hire()` has not been implemented in SOIL yet.

Try

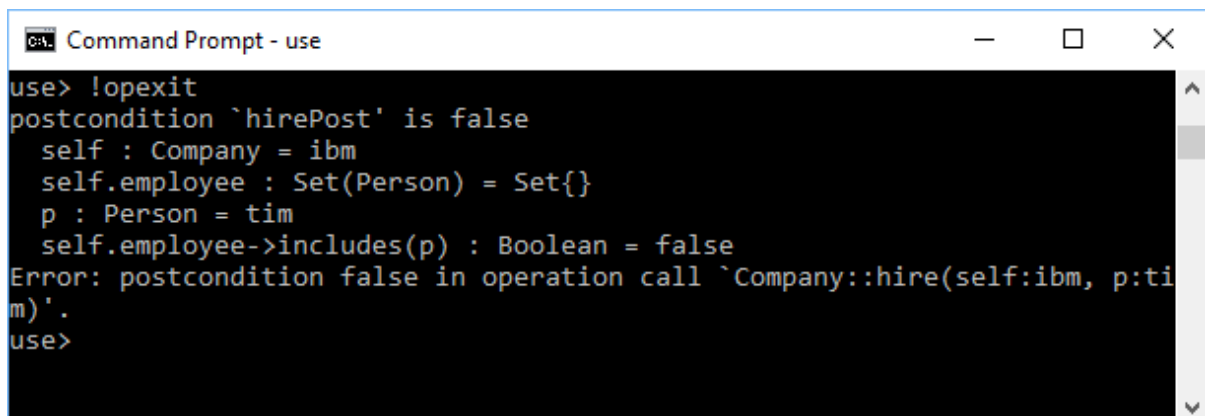
```
!openter ibm hire(tim)
```



```
Command Prompt - use
use> !openter ibm hire(tim)
precondition `hirePre1' is true
precondition `hirePre2' is true
use> _
```

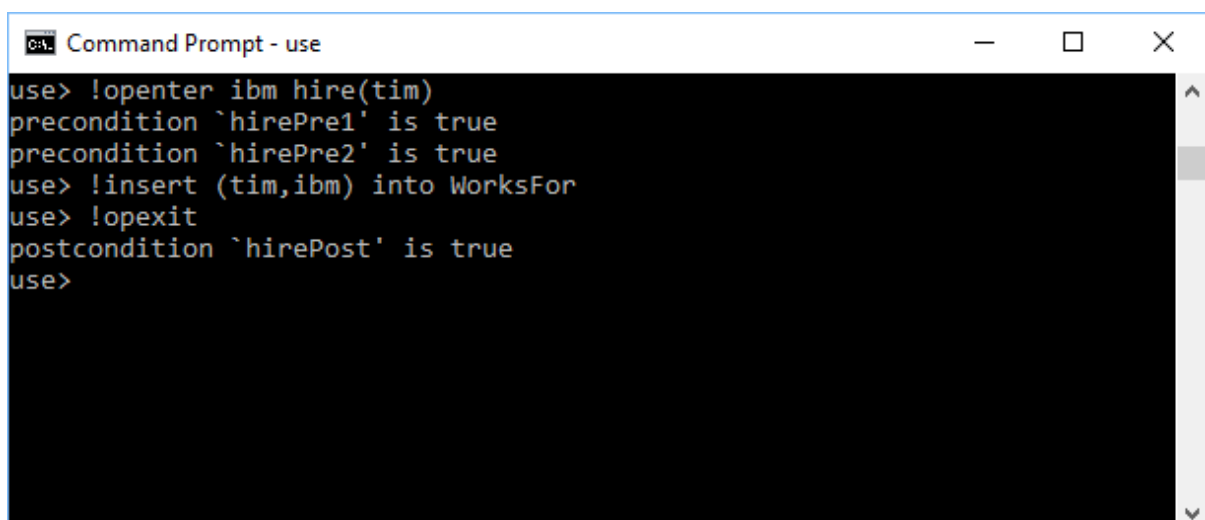
You can see the two preconditions are satisfied or true. And then try

```
!opexit
```



```
Command Prompt - use
use> !opexit
postcondition `hirePost' is false
  self : Company = ibm
  self.employee : Set(Person) = Set{}
  p : Person = tim
  self.employee->includes(p) : Boolean = false
Error: postcondition false in operation call `Company::hire(self:ibm, p:tim)'.
use>
```

Here the postcondition fails as **tim** has not been added to the **employee** set of **ibm**. There was no SOIL code to implement `hire()`. So we will try again as follows:

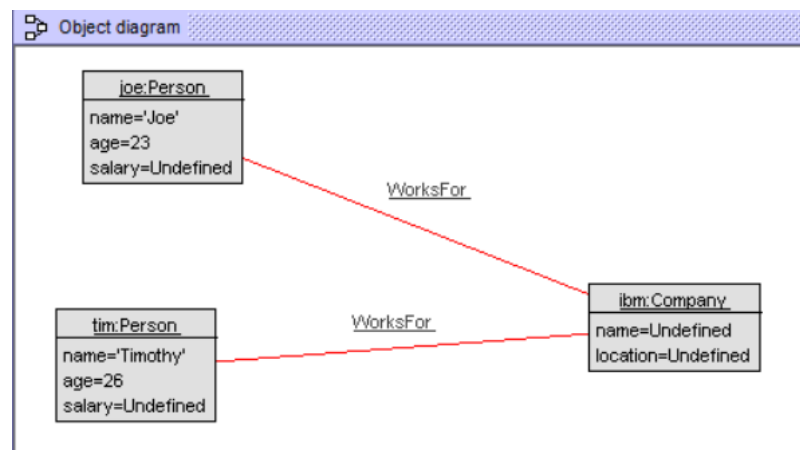


```
Command Prompt - use
use> !openter ibm hire(tim)
precondition `hirePre1' is true
precondition `hirePre2' is true
use> !insert (tim,ibm) into WorksFor
use> !opexit
postcondition `hirePost' is true
use>
```

Here we used a SOIL command, `insert (tim,ibm) into WorksFor`, to implement the operation `hire()` and on !opexit we see that the postcondition is true. In other words the contract for `hire()` has been satisfied. The implementation simply consisted of inserting `tim` into the set `employee`, and afterwards opexit checks that he is in fact in this set. In Java or C# the `tim` object would be inserted into possibly an array or a more complicated collection data structure.

### Evaluating OCL Expression

Next, hire joe and then use the USE menu **State | Evaluate OCL expression** to evaluate various OCL expressions as shown below.



**Evaluate OCL expression**

Enter OCL expression:

ibm.employee

**Result:**

Set{joe,tim} : Set(Person)

Buttons: Evaluate, Browser, Clear, Close

**Evaluate OCL expression**


Enter OCL expression:

ibm.employee->includes(tim)

**Result:**

true : Boolean

Buttons: Evaluate, Browser, Clear, Close

 Evaluate OCL expression ×

Enter OCL expression:

ibm.employee->excludes(tim)

Result:


false : Boolean

Evaluate

Browser

Clear

Close

 Evaluate OCL expression ×

Enter OCL expression:

ibm.employee->size()

Result:


2 : Integer

Evaluate

Browser

Clear

Close

 Evaluate OCL expression ×

Enter OCL expression:

tim.age

Result:

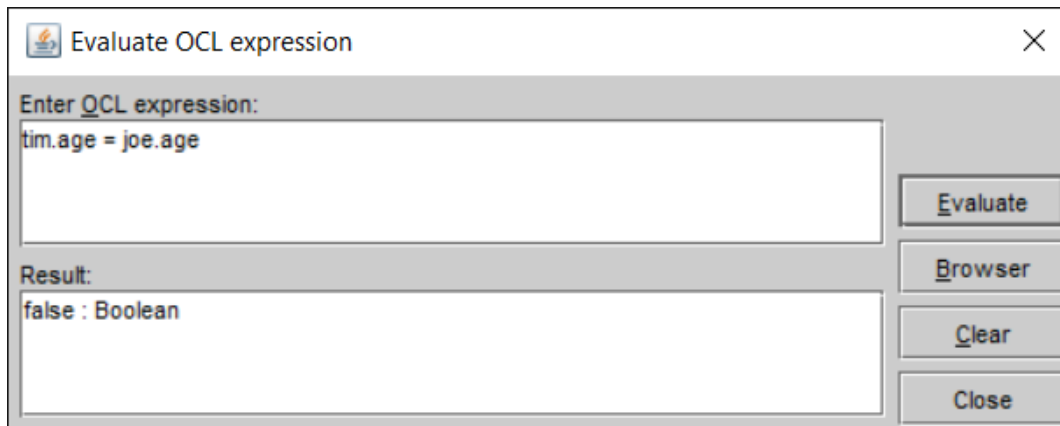
26 : Integer

Evaluate

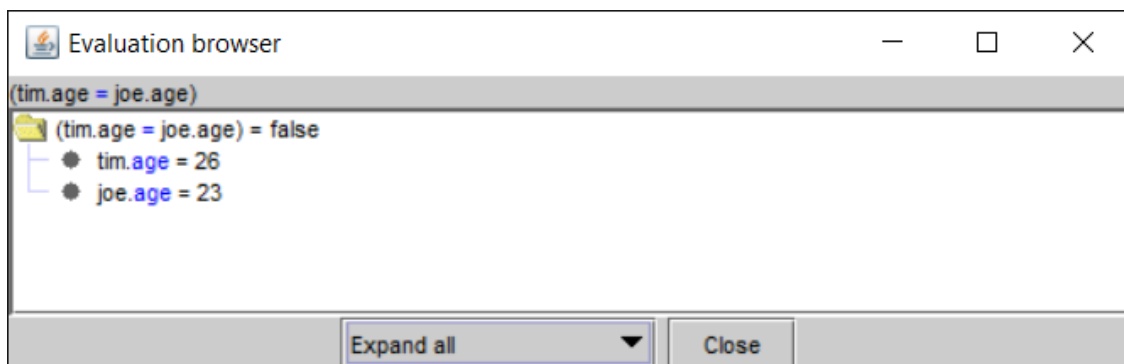
Browser

Clear

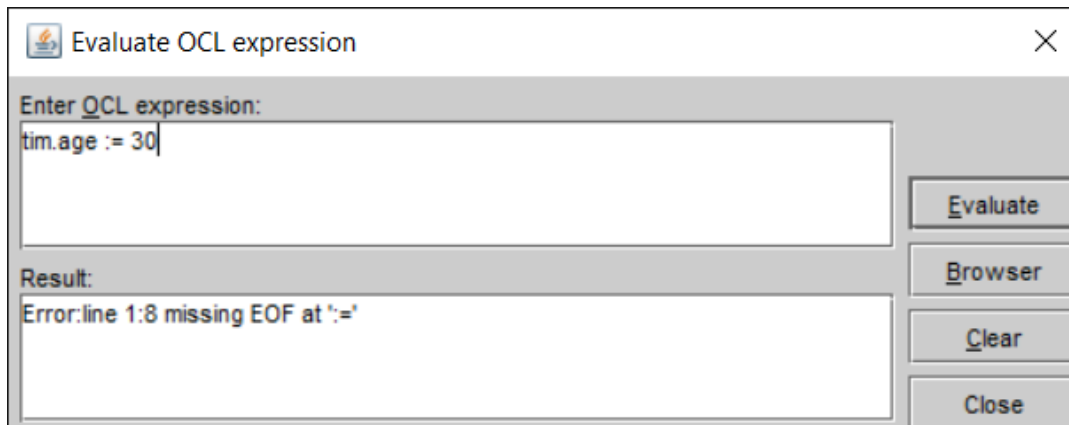
Close

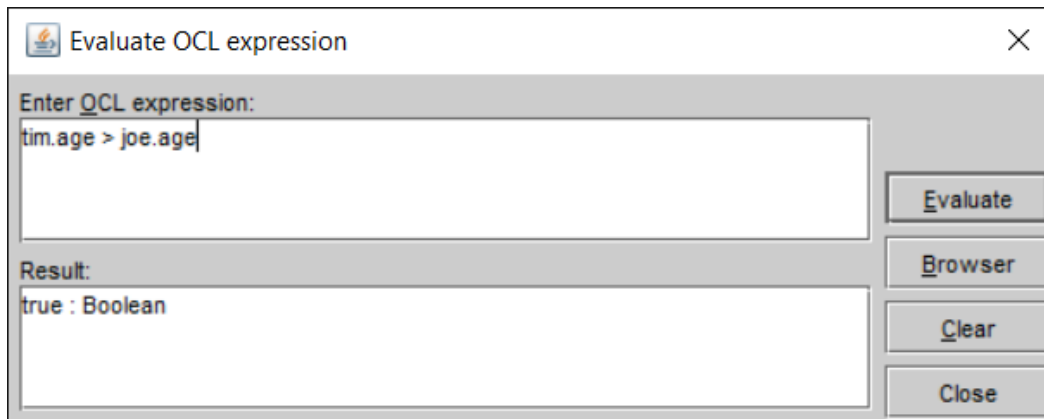


On this, press the **Browser** button to see why the OCL expression is false. This can be useful in debugging OCL expressions.



Note you can't run a SOIL command from within the OCL Evaluator as in:





Press **Browser** to get:



## Exercises

1. Try to fire tim, are all the pre and post conditions satisfied?
2. If you succeeded in firing tim, try again. What happens?