Design by Contract

AMIR AKBARI

C18442284

DT228/2

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Amir Akbari

Amir Akbari

12/05/2020

Contents

Introduction

Using the *Design by Contract* engineering design technique a system that allows a *Customer*, *ARProcessor*, *Invoice* and *Receivables* account to operate together.

The aim of the assignment is to make a functional system using Design by Contract, including, classes, objects, operations, associations, post conditions, preconditions etc.

The system is constructed using Oracle's UML USE.

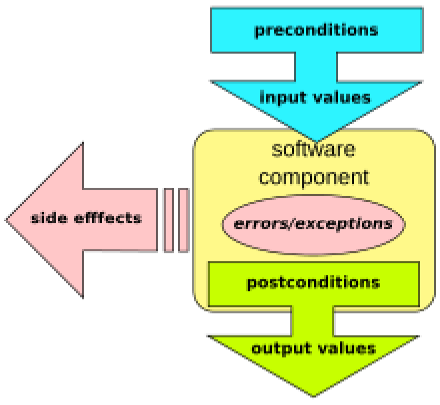
A UML representation was chosen to create a visual diagram, UML USE allows more control over creating the diagrams than other conventional software such as Argo UML as the user must code the classes, objects, associations, relationships etc, providing a greater amount of freedom over Argo UML to create more specialised diagrams.

Design by Contract

Design by contract (DbC) is a software correctness method. It uses preconditions and postconditions to record (or programmatically assert) the exchange in state caused by a bit of a program.

DBC is a software program engineering approach that specializes in reliability right from the begin of the development system. software program systems are considered as collections of speaking components whose interactions are based on exactly described contracts. Contracts specify the rights and obligations of suppliers and customers of additives via a simple declaration language carried out to operations on the component interface. utility of DBC strategies from the very beginning of a component life cycle has a profound effect on software quality.[1]

The diagram underneath indicates the fundamental technique of how of design by contract works.



[1] https://adtmag.com/articles/2001/04/01/from-uml-to-design-by-contract.aspx (Last Accessed 12/05/2020)

The precondition and postconditions are necessary for making sure that the function can execute and has executed.

There is three main concepts of this model:

- **Preconditions:** the client is obligated to meet a function's required preconditions before calling a function. If the preconditions aren't met, then the function won't perform correctly.[2]
- **Postconditions:** the function ensures that certain conditions will be met after it has finished its work. If a postcondition isn't met, then the function did not complete its work correctly.[2]
- **class invariant:** constraints that each instance of the class must satisfy. This defines the state that must hold true for the class to operate according to its design.[2]

The benefits of using Design by Contract include the following:

- ❖ A better understanding of the object-oriented method and, more generally, of software construction.[3]
- ❖ A systematic approach to building bug-free object-oriented systems.[3]
- ❖ An effective framework for debugging, testing and, more generally, quality assurance.[3]
- ❖ A method for documenting software components.[3]
- ❖ Better understanding and control of the inheritance mechanism.[3]
- ❖ A technique for dealing with abnormal cases, leading to a safe and effective language construct for exception handling.[3]

UML and OCL

A UML Operation contract identifies system state adjustments when an operation happens. effectively, it's going to outline what each system operation does. An operation is taken from a system sequence diagram. it is a single event from that diagram. a domain model can be used to help generate an operation contract

The object Constraint Language (OCL) is a formal language that expresses aspect impact-free constraints in UML models. A side effect-free operation is defined as one that will not change the state of the system whilst performed, such as attribute accesses or a true function call. In a UML model, a side effect-free operation is defined as one whose isQuery attribute equals true.[4]

[2] https://github.com/yszheda/wiki/wiki/API-Design-for-CPP-Notes (Last Accessed 12/05/2020)
[3] https://www.eiffel.com/values/design-by-contract/introduction/(Last Accessed 12/05/2020)
[4] https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6310/readings/gt-sad-ocl-specification.pdf(Last Accessed 12/05/2020)

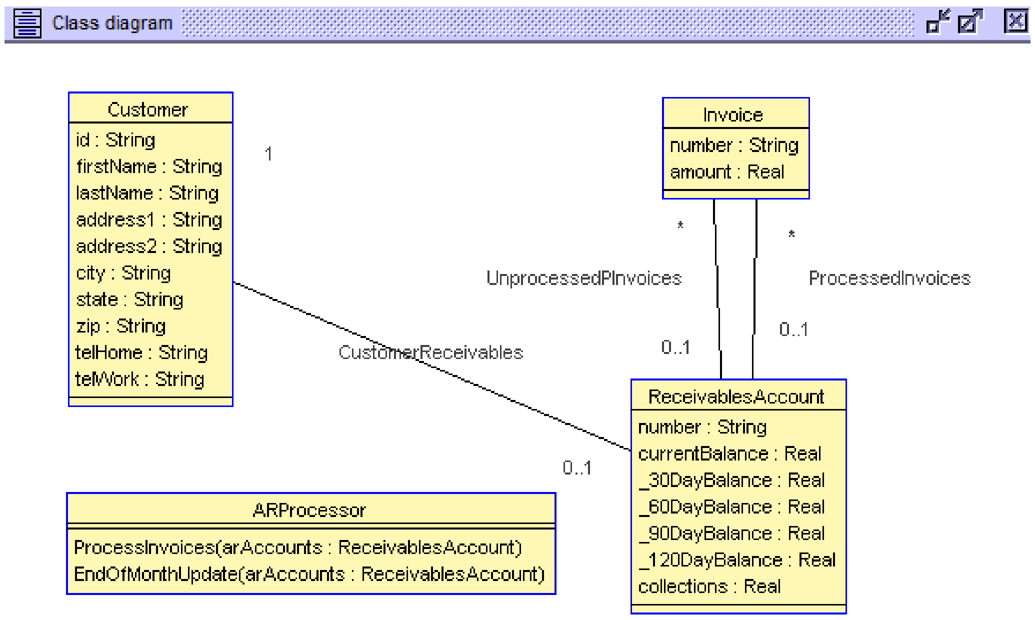
OCL expressions can be used to specify operations / actions that, whilst executed, do alter the kingdom of the system. UML modelers can use OCL to specify application-specific constraints of their models. UML modelers also can use OCL to specify queries on the UML model, which are absolutely programming language independent.[4]

UML USE Diagrams

The following diagrams are the output of my .use code system.

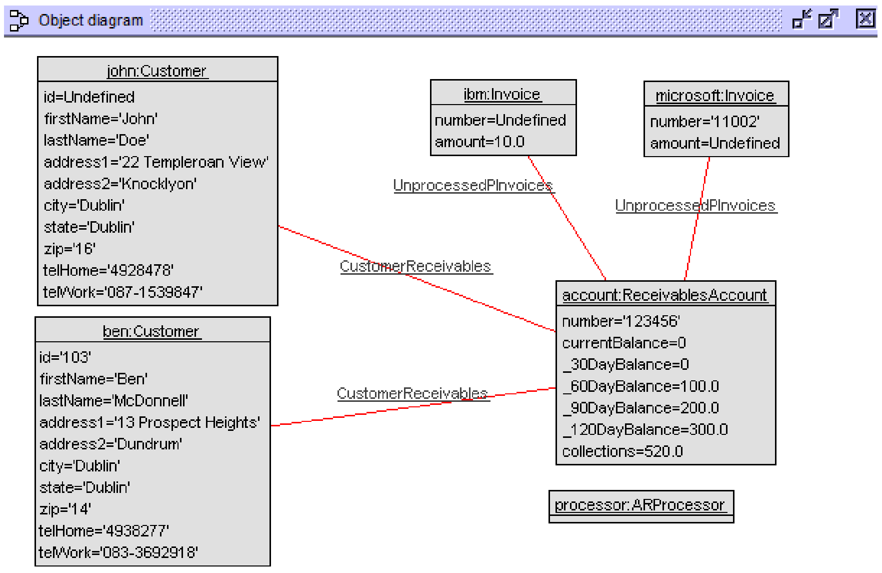
The diagram below is the class diagram of the USE system. As shown it shows the value types stored in each class and the constraints between certain classes showing the relationship types and constraint names.

In the class diagram there is a *Customer*, *Invoice*, *ARProcessor* and *ReceivablesAccount* classes. Within those classes there are many variables with the appropriate value types. The constraints allow relationships to be established between the classes, for example in the *ReceivablesAccount* it takes in *UnprocessedInvoices* and *ProcessedInvoices* constraint operations which will process unprocessed invoices and output the processed invoices.



Below is an object diagram. All the objects created from each class is shown here. In this case there are two objects created from the Customer class (john & ben), one ReceivablesAccount object (account), two invoices (ibm, microsoft) and a ARProcessor object (processor).

In the objects associations are setup between the objects, working in a somewhat similar way than constraints. The associations are setup between the two Customers objects to the ReceivablesAccount object and the two Invoices associated with the ReceivablesAccount.



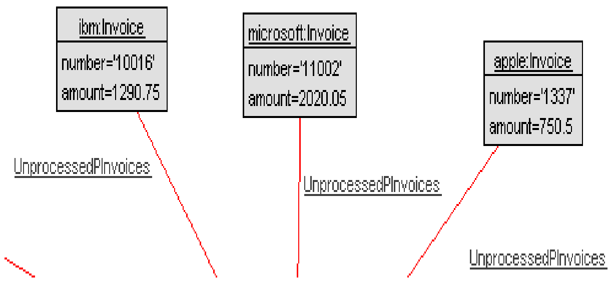
Processing Invoices

To process the unprocessed invoices, the operation in the .use to process the unprocessed invoices must be called. To do this the following .soil code is called…

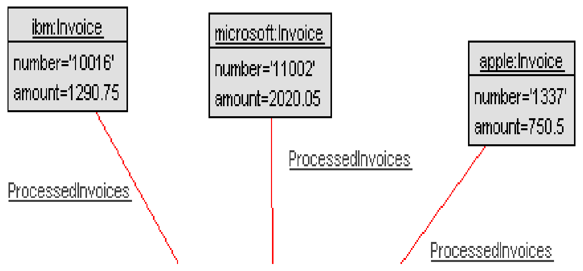
```
!openter processor EndOfMonthUpdate(Set{account})
```

The noticeable change will then take place in the objects diagram…

From:



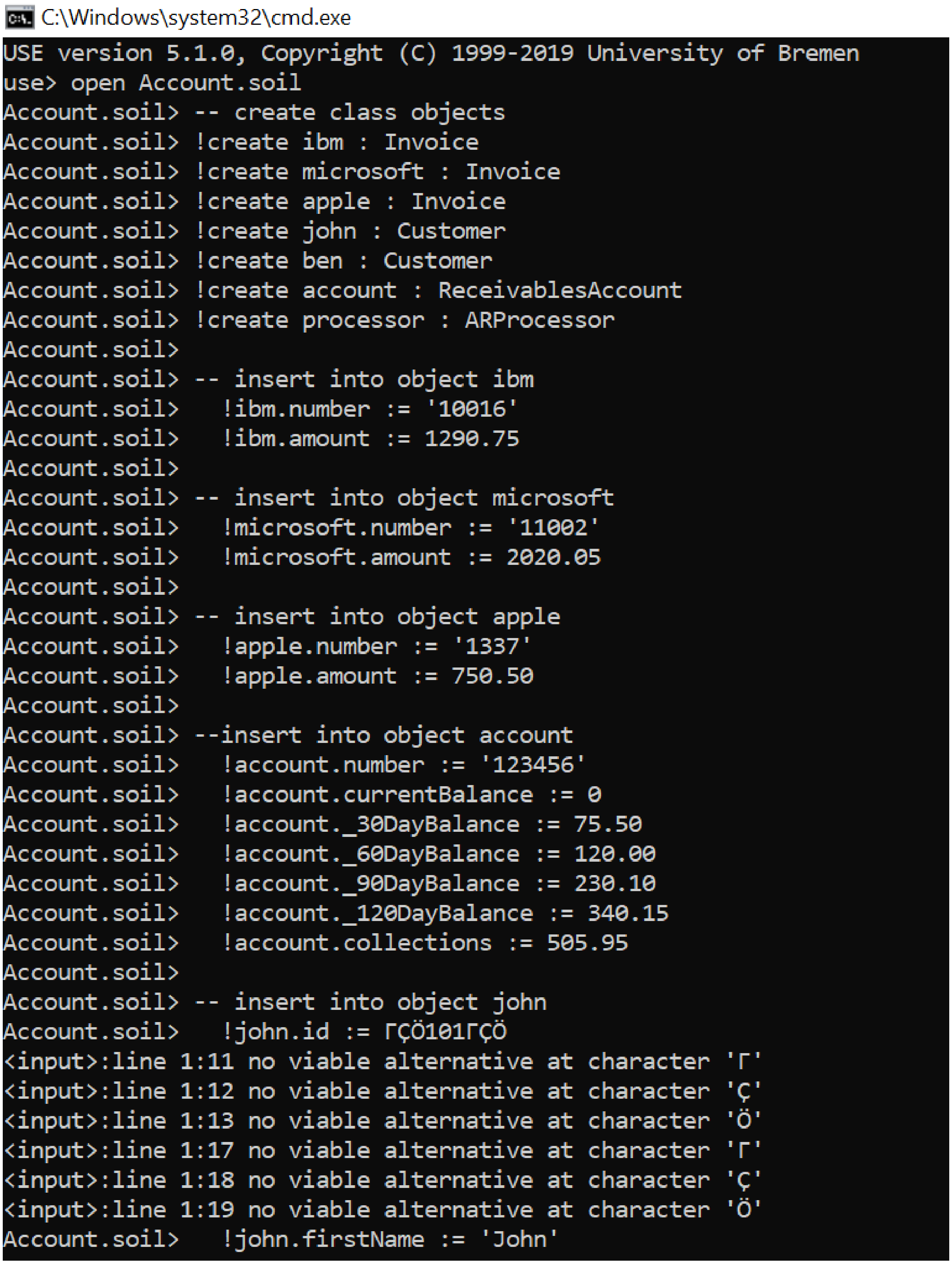
To:



Changing the state of association from Unprocessed to Processed.

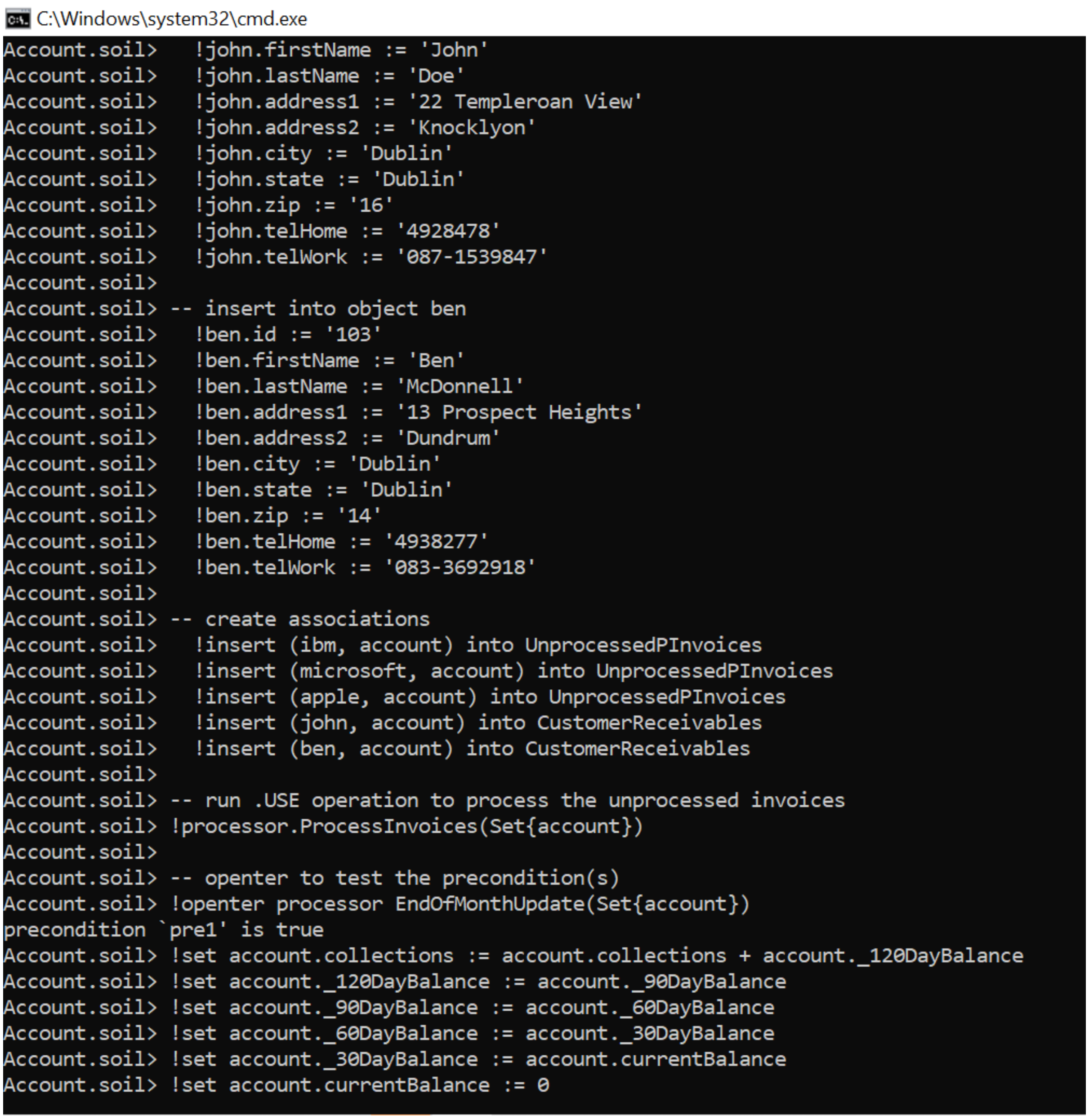
.soil Implementation

The .soil code is implemented through cmd. The following is the output when the soil code is implemented.

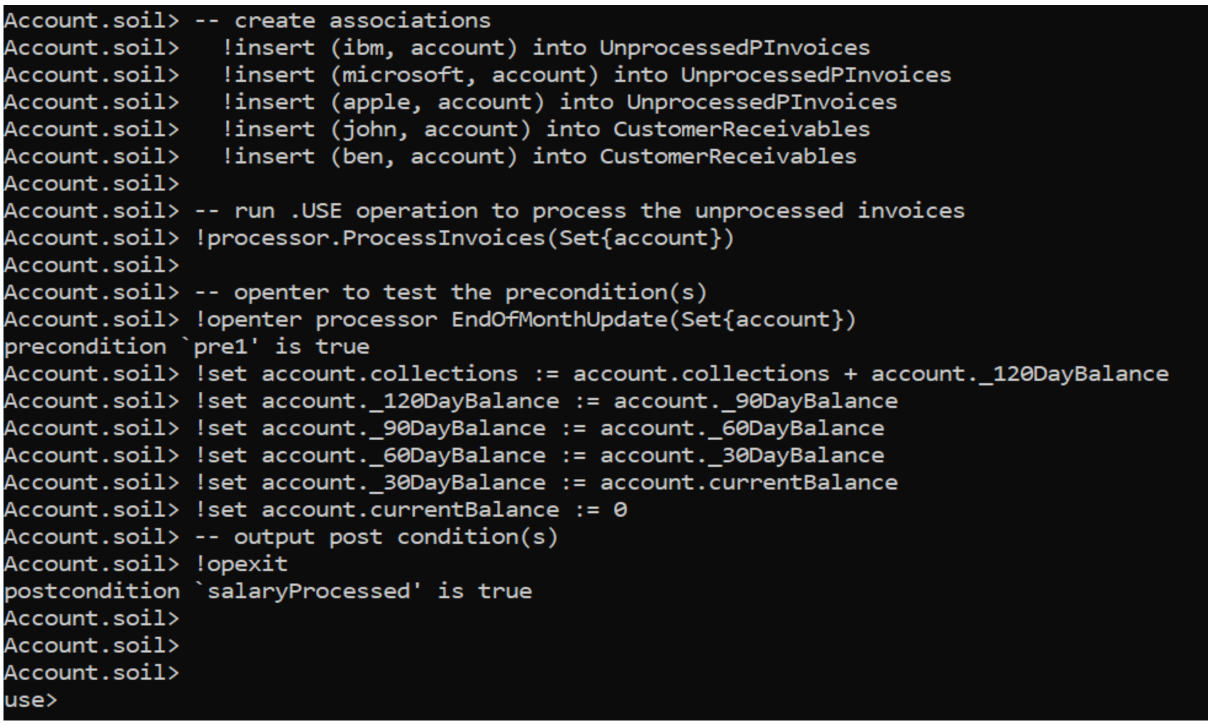


C:\Windows\system32\cmd.exe

```
USE version 5.1.0, Copyright (C) 1999-2019 University of Bremen
use> open Account.soil
Account.soil> -- create class objects
Account.soil> !create ibm : Invoice
Account.soil> !create microsoft : Invoice
Account.soil> !create apple : Invoice
Account.soil> !create john : Customer
Account.soil> !create ben : Customer
Account.soil> !create account : ReceivablesAccount
Account.soil> !create processor : ARProcessor
Account.soil>
Account.soil> -- insert into object ibm
Account.soil>   !ibm.number := '10016'
Account.soil>   !ibm.amount := 1290.75
Account.soil>
Account.soil> -- insert into object microsoft
Account.soil>   !microsoft.number := '11002'
Account.soil>   !microsoft.amount := 2020.05
Account.soil>
Account.soil> -- insert into object apple
Account.soil>   !apple.number := '1337'
Account.soil>   !apple.amount := 750.50
Account.soil>
Account.soil> --insert into object account
Account.soil>   !account.number := '123456'
Account.soil>   !account.currentBalance := 0
Account.soil>   !account._30DayBalance := 75.50
Account.soil>   !account._60DayBalance := 120.00
Account.soil>   !account._90DayBalance := 230.10
Account.soil>   !account._120DayBalance := 340.15
Account.soil>   !account.collections := 505.95
Account.soil>
Account.soil> -- insert into object john
Account.soil>   !john.id := ΓÇÖ101ΓÇÖ
<input>:line 1:11 no viable alternative at character 'Γ'
<input>:line 1:12 no viable alternative at character 'Ç'
<input>:line 1:13 no viable alternative at character 'Ö'
<input>:line 1:17 no viable alternative at character 'Γ'
<input>:line 1:18 no viable alternative at character 'Ç'
<input>:line 1:19 no viable alternative at character 'Ö'
Account.soil>   !john.firstName := 'John'
```


C:\Windows\system32\cmd.exe

```
Account.soil>   !john.firstName := 'John'
Account.soil>   !john.lastName := 'Doe'
Account.soil>   !john.address1 := '22 Templeroan View'
Account.soil>   !john.address2 := 'Knocklyon'
Account.soil>   !john.city := 'Dublin'
Account.soil>   !john.state := 'Dublin'
Account.soil>   !john.zip := '16'
Account.soil>   !john.telHome := '4928478'
Account.soil>   !john.telWork := '087-1539847'
Account.soil>
Account.soil> -- insert into object ben
Account.soil>   !ben.id := '103'
Account.soil>   !ben.firstName := 'Ben'
Account.soil>   !ben.lastName := 'McDonnell'
Account.soil>   !ben.address1 := '13 Prospect Heights'
Account.soil>   !ben.address2 := 'Dundrum'
Account.soil>   !ben.city := 'Dublin'
Account.soil>   !ben.state := 'Dublin'
Account.soil>   !ben.zip := '14'
Account.soil>   !ben.telHome := '4938277'
Account.soil>   !ben.telWork := '083-3692918'
Account.soil>
Account.soil> -- create associations
Account.soil>   !insert (ibm, account) into UnprocessedPInvoices
Account.soil>   !insert (microsoft, account) into UnprocessedPInvoices
Account.soil>   !insert (apple, account) into UnprocessedPInvoices
Account.soil>   !insert (john, account) into CustomerReceivables
Account.soil>   !insert (ben, account) into CustomerReceivables
Account.soil>
Account.soil> -- run .USE operation to process the unprocessed invoices
Account.soil> !processor.ProcessInvoices(Set{account})
Account.soil>
Account.soil> -- openter to test the precondition(s)
Account.soil> !openter processor EndOfMonthUpdate(Set{account})
precondition `pre1' is true
Account.soil> !set account.collections := account.collections + account._120DayBalance
Account.soil> !set account._120DayBalance := account._90DayBalance
Account.soil> !set account._90DayBalance := account._60DayBalance
Account.soil> !set account._60DayBalance := account._30DayBalance
Account.soil> !set account._30DayBalance := account.currentBalance
Account.soil> !set account.currentBalance := 0
```



```
Account.soil> -- create associations
Account.soil>   !insert (ibm, account) into UnprocessedPInvoices
Account.soil>   !insert (microsoft, account) into UnprocessedPInvoices
Account.soil>   !insert (apple, account) into UnprocessedPInvoices
Account.soil>   !insert (john, account) into CustomerReceivables
Account.soil>   !insert (ben, account) into CustomerReceivables
Account.soil>
Account.soil> -- run .USE operation to process the unprocessed invoices
Account.soil> !processor.ProcessInvoices(Set{account})
Account.soil>
Account.soil> -- openter to test the precondition(s)
Account.soil> !openter processor EndOfMonthUpdate(Set{account})
precondition `pre1' is true
Account.soil> !set account.collections := account.collections + account._120DayBalance
Account.soil> !set account._120DayBalance := account._90DayBalance
Account.soil> !set account._90DayBalance := account._60DayBalance
Account.soil> !set account._60DayBalance := account._30DayBalance
Account.soil> !set account._30DayBalance := account.currentBalance
Account.soil> !set account.currentBalance := 0
Account.soil> -- output post condition(s)
Account.soil> !opexit
postcondition `salaryProcessed' is true
Account.soil>
Account.soil>
Account.soil>
use>
```

The *precondition* for the *EndOfMonthUpdate* is testing if the function can be executed if there are *unprocessed* invoices. The *postcondition* is testing if the function has run correctly, in this case it’s returning ‘true’ meaning that it has successfully executed and the *unprocessed invoices* have been processed.

Conclusion

I have achieved creating a flexible system which implement Design by Contract concepts, pre and post conditions, achieving all that I have set out to do. The system accepts users into an account which can process invoices and handle end of month balances.

The postcondition returns ‘true’ which means the precondition has successfully executed, and the invoices are processed meaning that the system works as intended.

What I have learned

- Vastly improved knowledge and knowhow of using USE UML.
- Learned how .USE, .SOIL, .OLT, .CLT work.
- Improved knowledge of design by contract.
- Improved knowledge of post and preconditions and how they work and are executed.
- How to write a specific system using design by contract.

Sample Code

The following is the .use code implemented:

```
-- model Name =...
model Account
-- declare and populate classes with variables and value types...
--Invoice
class Invoice
attributes
        number : String
        amount : Real
end
--ReceivablesAccount
class ReceivablesAccount
attributes
        number : String
        currentBalance : Real
        _30DayBalance : Real
        _60DayBalance : Real
        _90DayBalance : Real
        _120DayBalance : Real
        collections : Real
end
--Customer
class Customer
attributes
        id : String
        firstName : String
        lastName : String
        address1 : String
        address2 : String
        city : String
        state : String
        zip : String
        telHome : String
        telWork : String
end
--ARProcessor
class ARProcessor
operations
ProcessInvoices(arAccounts : Set(ReceivablesAccount))
        -- begin operation
        begin

                -- create forloop to process unprocessed invoices, is later called in .SOIL code
                for arAccount in arAccounts do

                        for invoice in arAccount.unProcessedInvoices do
                                insert (invoice, arAccount) into ProcessedInvoices;
                                delete (invoice, arAccount) from UnprocessedPInvoices;
                        end
                end
        -- end operation
        end
-- update at end of the month
EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
end
-- create associations between classes
-- Associations
        -- association between unProcessedInvoices & unProcessedReceivables
        association UnprocessedPInvoices between
                Invoice[0..*] role unProcessedInvoices
                ReceivablesAccount[0..1] role unProcessedReceivables
        end
        -- association between ProcessInvoices & processedReceivables
        association ProcessedInvoices between
                Invoice[0..*] role processedInvoices
                ReceivablesAccount[0..1] role processedReceivables
        end
        -- association between customer & receivables
        association CustomerReceivables between
                Customer[1] role customer
                ReceivablesAccount[0..1] role receivables
        end
constraints
 --ReceivablesAccount invariants
context ReceivablesAccount
        inv invoiceNumberLength:
                self.number.size() = 6
context ReceivablesAccount
        inv inv2:
                unProcessedInvoices->intersection(processedInvoices)->isEmpty()
--ARProcessor::ProcessInvoices pre-conditions
context ARProcessor::ProcessInvoices(arAccounts : Set(ReceivablesAccount))
        pre pre1:
                arAccounts->forAll(unProcessedInvoices->notEmpty())
--ARProcessor::ProcessInvoices post-conditions
--unProcessedInvoices become processedInvoices.
context ARProcessor::ProcessInvoices(arAccounts : Set(ReceivablesAccount))
```

```
        post post1:
                        arAccounts->forAll(unProcessedInvoices->isEmpty() and
                        processedInvoices->includesAll(unProcessedInvoices@pre))
--ARProcessor::EndOfMonthUpdate pre-conditions
--There are no unprocessed invoices
context ARProcessor::EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
        pre pre1:
        --arAccounts.currentBalance = 100.00
                arAccounts->forAll(unProcessedInvoices->isEmpty())
--ARProcessor::EndOfMonthUpdate post-conditions
--context ARProcessor::EndOfMonthUpdate (arAccounts : ReceivablesAccount)
        --post post1 salaryProcessed:
        context ARProcessor::EndOfMonthUpdate (arAccounts : Set(ReceivablesAccount))
        post salaryProcessed:
                        arAccounts->forAll(
                                currentBalance = 0 and
                                _30DayBalance = currentBalance@pre and
                                _60DayBalance = _30DayBalance@pre and
                                _90DayBalance = _60DayBalance@pre and
                                _120DayBalance = _90DayBalance@pre and
                                collections = collections@pre + _120DayBalance@pre
                        )
```

The following is the .soil code implemented:

```
-- create class objects
!create ibm : Invoice
!create microsoft : Invoice
!create apple : Invoice
!create john : Customer
!create ben : Customer
!create account : ReceivablesAccount
!create processor : ARProcessor

-- insert into object ibm
        !ibm.number := '10016'
        !ibm.amount := 1290.75

-- insert into object microsoft
        !microsoft.number := '11002'
        !microsoft.amount := 2020.05

-- insert into object apple
        !apple.number := '1337'
        !apple.amount := 750.50

--insert into object account
        !account.number := '123456'
        !account.currentBalance := 0
        !account._30DayBalance := 75.50
        !account._60DayBalance := 120.00
        !account._90DayBalance := 230.10
        !account._120DayBalance := 340.15
        !account.collections := 505.95

-- insert into object john
        !john.id := ’101’
        !john.firstName := 'John'
        !john.lastName := 'Doe'
        !john.address1 := '22 Templeroan View'
        !john.address2 := 'Knocklyon'
        !john.city := 'Dublin'
        !john.state := 'Dublin'
        !john.zip := '16'
        !john.telHome := '4928478'
        !john.telWork := '087-1539847'

-- insert into object ben
        !ben.id := '103'
        !ben.firstName := 'Ben'
        !ben.lastName := 'McDonnell'
        !ben.address1 := '13 Prospect Heights'
```

```
            !ben.address2 := 'Dundrum'
            !ben.city := 'Dublin'
            !ben.state := 'Dublin'
            !ben.zip := '14'
            !ben.telHome := '4938277'
            !ben.telWork := '083-3692918'

-- create associations
            !insert (ibm, account) into UnprocessedPInvoices
            !insert (microsoft, account) into UnprocessedPInvoices
            !insert (apple, account) into UnprocessedPInvoices
            !insert (john, account) into CustomerReceivables
            !insert (ben, account) into CustomerReceivables

-- run .USE operation to process the unprocessed invoices
!processor.ProcessInvoices(Set{account})

-- openter to test the precondition(s)
!openter processor EndOfMonthUpdate(Set{account})
!set account.collections := account.collections + account._120DayBalance
!set account._120DayBalance := account._90DayBalance
!set account._90DayBalance := account._60DayBalance
!set account._60DayBalance := account._30DayBalance
!set account._30DayBalance := account.currentBalance
!set account.currentBalance := 0
-- output post condition(s)
!opexit
```