|  |
| --- |
| Software Engineering 2  UML Report  Donncha Cassidy-Hand  D14123580 |

**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.

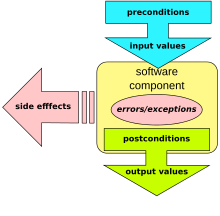
The system is constructed using Oracle’s UML USE.

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 methodology. It uses preconditions and *postconditions* to document (or programmatically assert) the change in state caused by a piece of a program.

The diagram below shows the basic process of how of design by contract works.



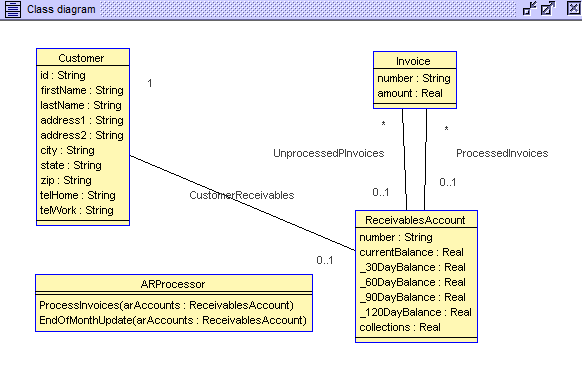
The precondition and postconditions are necessary for making sure that the function can execute and has executed.

**UML USE Diagrams**

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

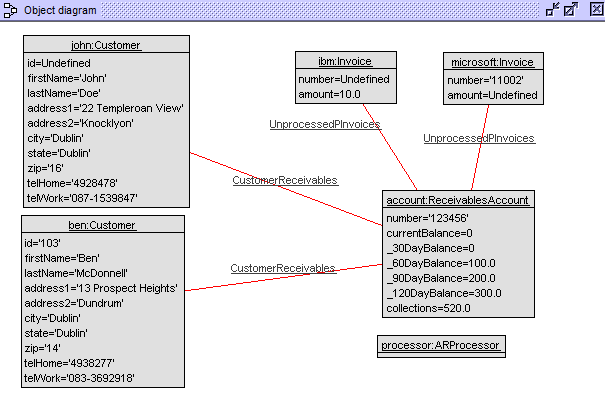
The diagram below is the class diagram of the USE system. As shown it shows the value types stored in each class and the constraints between certain classes showing the relationship types and constraint names.

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.



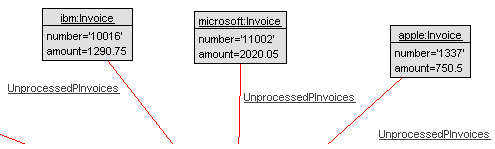
**Processing Invoices**

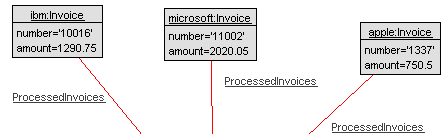
To process the unprocessed invoices, the operation in the .use to process the unprocessed invoices must be called. To do this the following .soil code is called…



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

**From:**   **To:**

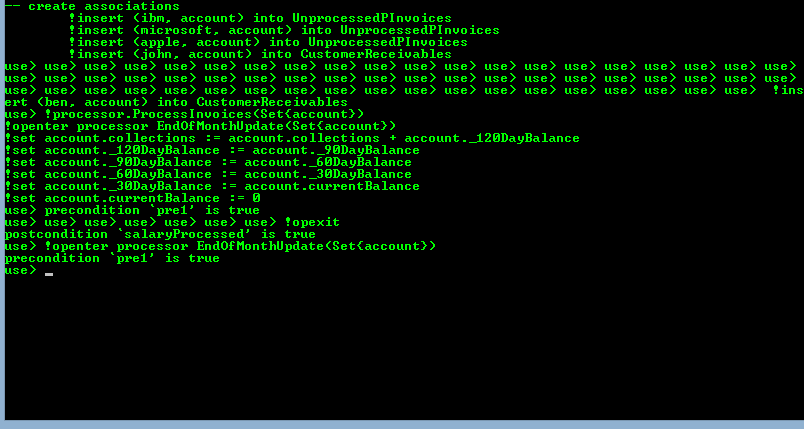
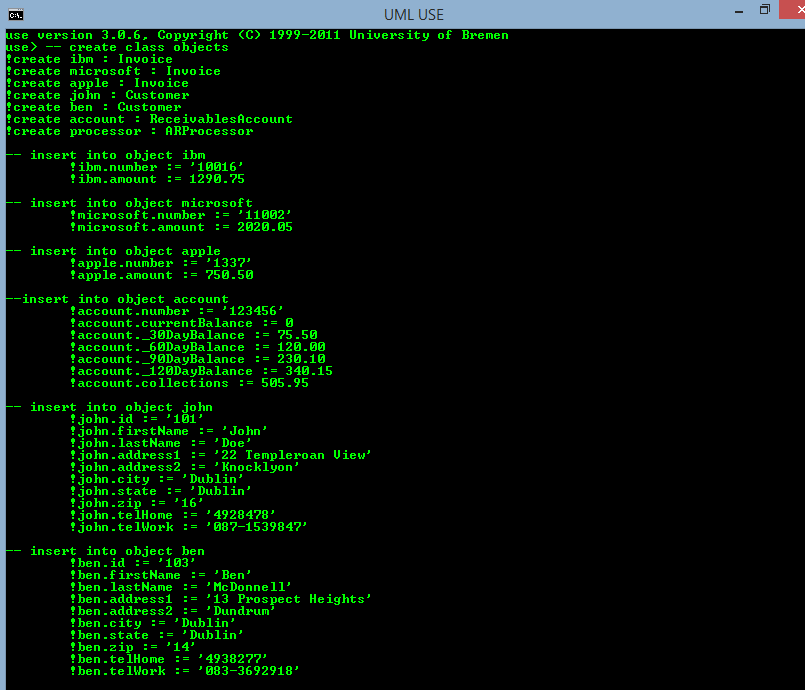




Changing the state of association from Unprocessed to Processed.

**.soil Implementation**

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



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 |