Software Engineering 2 Assignment

Accounts Receivable Processor – Design by Contract

Name: Hemant Sundarrajan

Student Number: C22440886

Table of Contents:

1. Introduction..1

2. Construction of Accounts Receivable Processor......................5

3. Diagrams...12

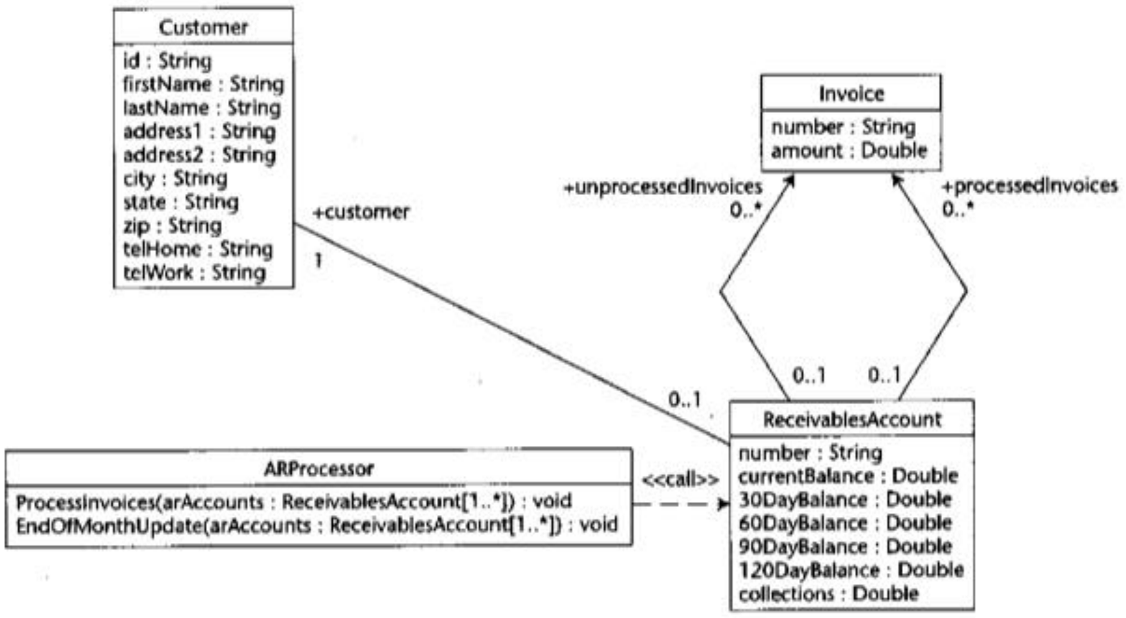
4. Testing Constraints..18

5. Conclusion..24

1. Introduction

For my software engineering assignment, there was a range of options I could choose to do. I chose to construct and test a USE model for Accounts Receivable Processor. The material is based mainly on **Model Driven Architecture** – Applying MDA to Enterprise Computing by David S. Frankel. I was given a class diagram to recreate, along with the DbC (Design by contract) constraints I had to implement. These constraints are a way to highlight how DbC approaches designing software. By utilizing these assertations, i.e. post-conditions, pre-conditions, and invariants, I will test them in the code I constructed.

Class Diagram I was required to construct:



Constraints I was required to implement:

```
-------------------------------
--ReceivablesAccount invariants
-------------------------------
--An invoice cannot be both unprocessed and processed.

context ReceivablesAccount inv:

   unprocessedInvoices->intersection(processedInvoices)->isEmpty ()

--An invoice number must be six characters in length.

context ReceivablesAccount inv:

   self.number->size () = 6

---------------------------------------------
--ARProcessor::ProcessInvoices pre-conditions
---------------------------------------------
--There must be some unprocessedInvoices.

context ARProcessor::ProcessInvoices (arAccounts : Set
(ReceivablesAccount)) pre:

   arAccounts->forAll (unprocessedInvoices->notEmpty () )

----------------------------------------------
--ARProcessor::ProcessInvoices post-conditions
----------------------------------------------

--unprocessedInvoices become processedInvoices.

context ARProcessor::ProcessInvoices (arAccounts : Set
(ReceivablesAccount)) post:

   arAccounts->forAll
   (
      unProcessedInvoices->isEmpty () and
      processedInvoices->includes (unprocessedInvoices@pre)
   )
----------------------------------------------
--ARProcessor::EndOfMonthUpdate pre-conditions
----------------------------------------------
--There are no unprocessed invoices.

context ARProcessor::EndOfMonthUpdate (arAccounts : Set
(ReceivablesAccount)) pre:

   arAccounts->forAll (unprocessedInvoices->isEmpty () )
```

```
-----------------------------------------------
--ARProcessor::EndOfMonthUpdate post-conditions
-----------------------------------------------
--For all of the ARaccounts the following holds:
   --The Collections value is its previous value plus the previous
120DayBalance and
   --the 120DayBalance is the previous 90DayBalance and
   --the 90DayBalance is the previous 60DayBalance and
   --the 60DayBalance is the previous 30DayBalance and
   --the 30DayBalance is the previous currentBalance
   --the currentBalance is 0.

context ARProcessor::EndOfMonthUpdate (arAccounts : Set
(ReceivablesAccount)) post:

   arAccounts->forAll
   (
      -- @pre modifies an identifier to refer to the value it had
      -- before the operation executed.
      currentBalance = 0 and
      30DayBalance =  currentBalance@pre and
      60DayBalance =  30DayBalance@pre and
      90DayBalance =  60DayBalance@pre and
      120DayBalance = 90DayBalance@pre and
      Collections = collections@pre + 120DayBalance@pre
   )
```

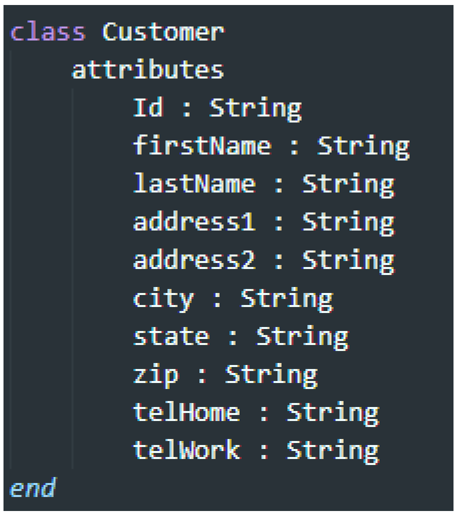
2. Construction of Accounts Receivable Processor

What was Required of me, was to construct the USE model for the ARP (Accounts Receivable model). I essentially did just this, however, it was imperative to implement a way to create invoices also, to test and understand the code being constructed. There is not much to design as with the class diagrams and constraints provided, I simply had to change a few things to be used correctly in the current model of use. I added state machines to the ARProcessor class and the ReceivablesAccount class. This allowed me to see the states of the account, and if objects have been unprocessed. Other than that, I simply implemented what was required, only adding 1 additional operation to thoroughly test the ARP.

USE Code:

Classes:

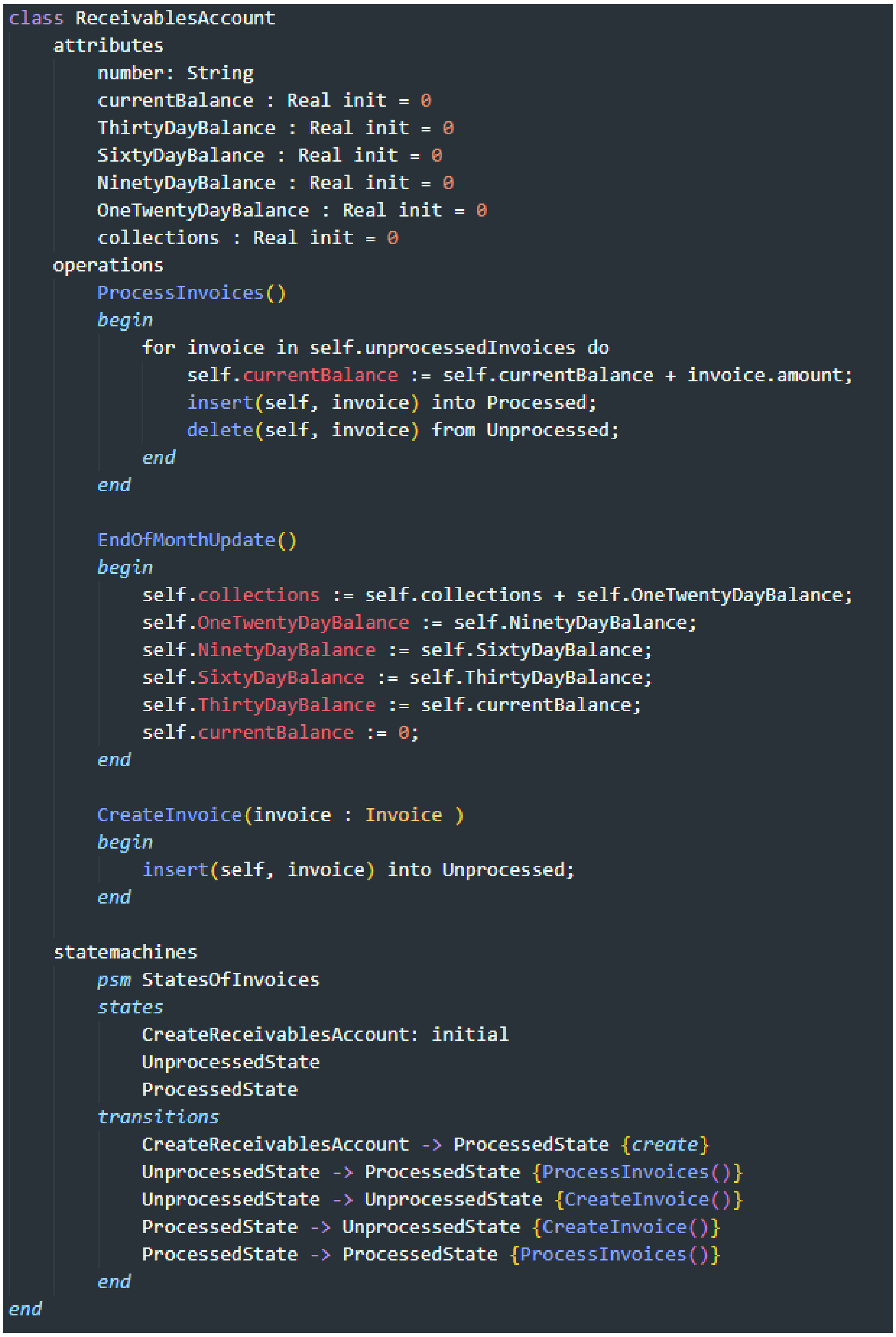
Customer Class:



```
class Customer
    attributes
        Id : String
        firstName : String
        lastName : String
        address1 : String
        address2 : String
        city : String
        state : String
        zip : String
        telHome : String
        telWork : String
end
```

The customer class is designed exactly as described in the class diagram shown above.

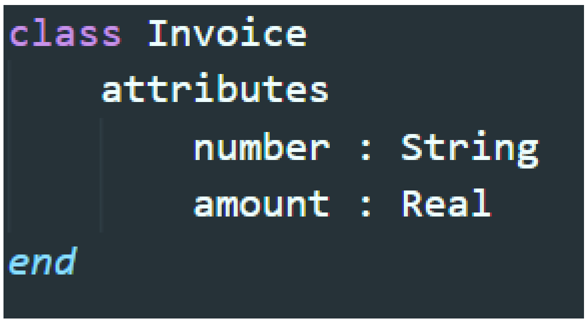
ReceivablesAccount Class:



```
class ReceivablesAccount
    attributes
        number: String
        currentBalance : Real init = 0
        ThirtyDayBalance : Real init = 0
        SixtyDayBalance : Real init = 0
        NinetyDayBalance : Real init = 0
        OneTwentyDayBalance : Real init = 0
        collections : Real init = 0
    operations
        ProcessInvoices()
        begin
            for invoice in self.unprocessedInvoices do
                self.currentBalance := self.currentBalance + invoice.amount;
                insert(self, invoice) into Processed;
                delete(self, invoice) from Unprocessed;
            end
        end

        EndOfMonthUpdate()
        begin
            self.collections := self.collections + self.OneTwentyDayBalance;
            self.OneTwentyDayBalance := self.NinetyDayBalance;
            self.NinetyDayBalance := self.SixtyDayBalance;
            self.SixtyDayBalance := self.ThirtyDayBalance;
            self.ThirtyDayBalance := self.currentBalance;
            self.currentBalance := 0;
        end

        CreateInvoice(invoice : Invoice )
        begin
            insert(self, invoice) into Unprocessed;
        end

    statemachines
        psm StatesOfInvoices
        states
            CreateReceivablesAccount: initial
            UnprocessedState
            ProcessedState
        transitions
            CreateReceivablesAccount -> ProcessedState {create}
            UnprocessedState -> ProcessedState {ProcessInvoices()}
            UnprocessedState -> UnprocessedState {CreateInvoice()}
            ProcessedState -> UnprocessedState {CreateInvoice()}
            ProcessedState -> ProcessedState {ProcessInvoices()}
        end
end
```

This class was implemented as required. A state machine was added to show the states of unprocessed and processed invoices. An operation was created to create invoices (CreateInvoice), this is linked to the InvoiceCreation operation seen later on in The ARProcessor class. Regarding using Double as a data type, in the current version of USE "Real" is what we would use.

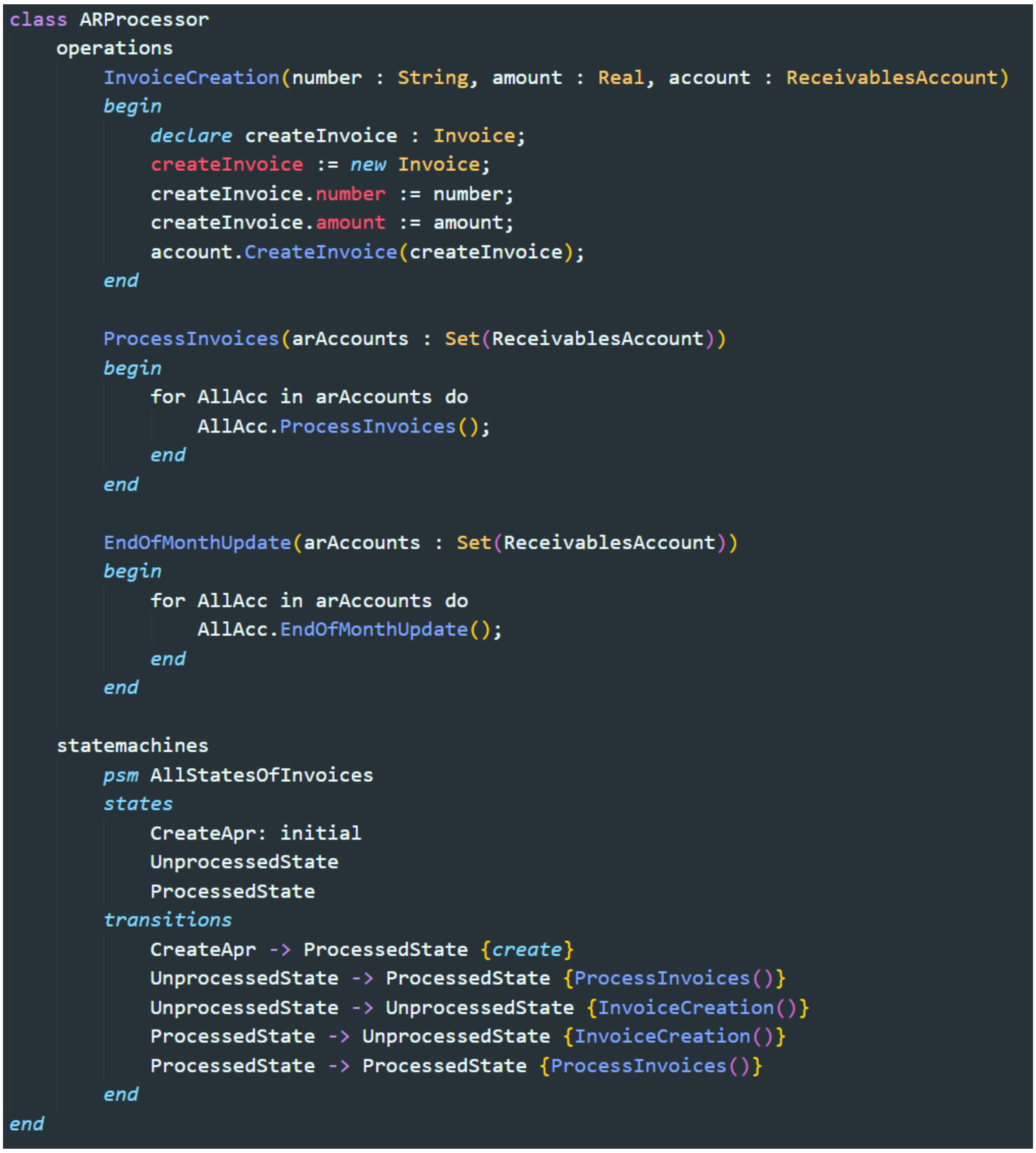
Invoice Class



```
class Invoice
    attributes
        number : String
        amount : Real
end
```

Implemented as required.

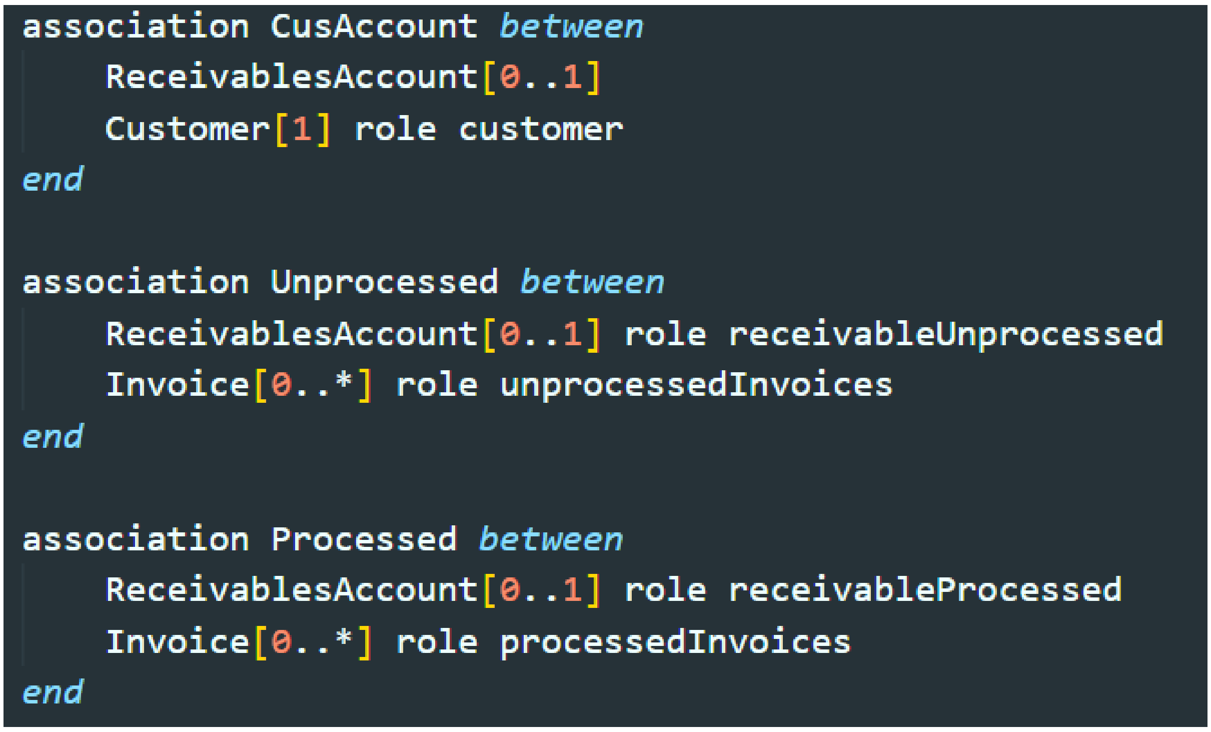
ARProcessor Class:



```
class ARProcessor
    operations
        InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)
        begin
            declare createInvoice : Invoice;
            createInvoice := new Invoice;
            createInvoice.number := number;
            createInvoice.amount := amount;
            account.CreateInvoice(createInvoice);
        end

        ProcessInvoices(arAccounts : Set(ReceivablesAccount))
        begin
            for AllAcc in arAccounts do
                AllAcc.ProcessInvoices();
            end
        end

        EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
        begin
            for AllAcc in arAccounts do
                AllAcc.EndOfMonthUpdate();
            end
        end

    statemachines
        psm AllStatesOfInvoices
        states
            CreateApr: initial
            UnprocessedState
            ProcessedState
        transitions
            CreateApr -> ProcessedState {create}
            UnprocessedState -> ProcessedState {ProcessInvoices()}
            UnprocessedState -> UnprocessedState {InvoiceCreation()}
            ProcessedState -> UnprocessedState {InvoiceCreation()}
            ProcessedState -> ProcessedState {ProcessInvoices()}
        end
end
```

The operations ProcessInvoices and EndOfMothUpdate, calls their respective operations in the ReceivablesAccount. It essentially does it for all the accounts. This is because the accounts are separate from each other. So rather than having the main operation functionality in the ARProcessor class, it calls its respective operation in the ReceivablesAccount class. I also added a similar statemachine to this class

Associations:



```
association CusAccount between
    ReceivablesAccount[0..1]
    Customer[1] role customer
end

association Unprocessed between
    ReceivablesAccount[0..1] role receivableUnprocessed
    Invoice[0..*] role unprocessedInvoices
end

association Processed between
    ReceivablesAccount[0..1] role receivableProcessed
    Invoice[0..*] role processedInvoices
end
```

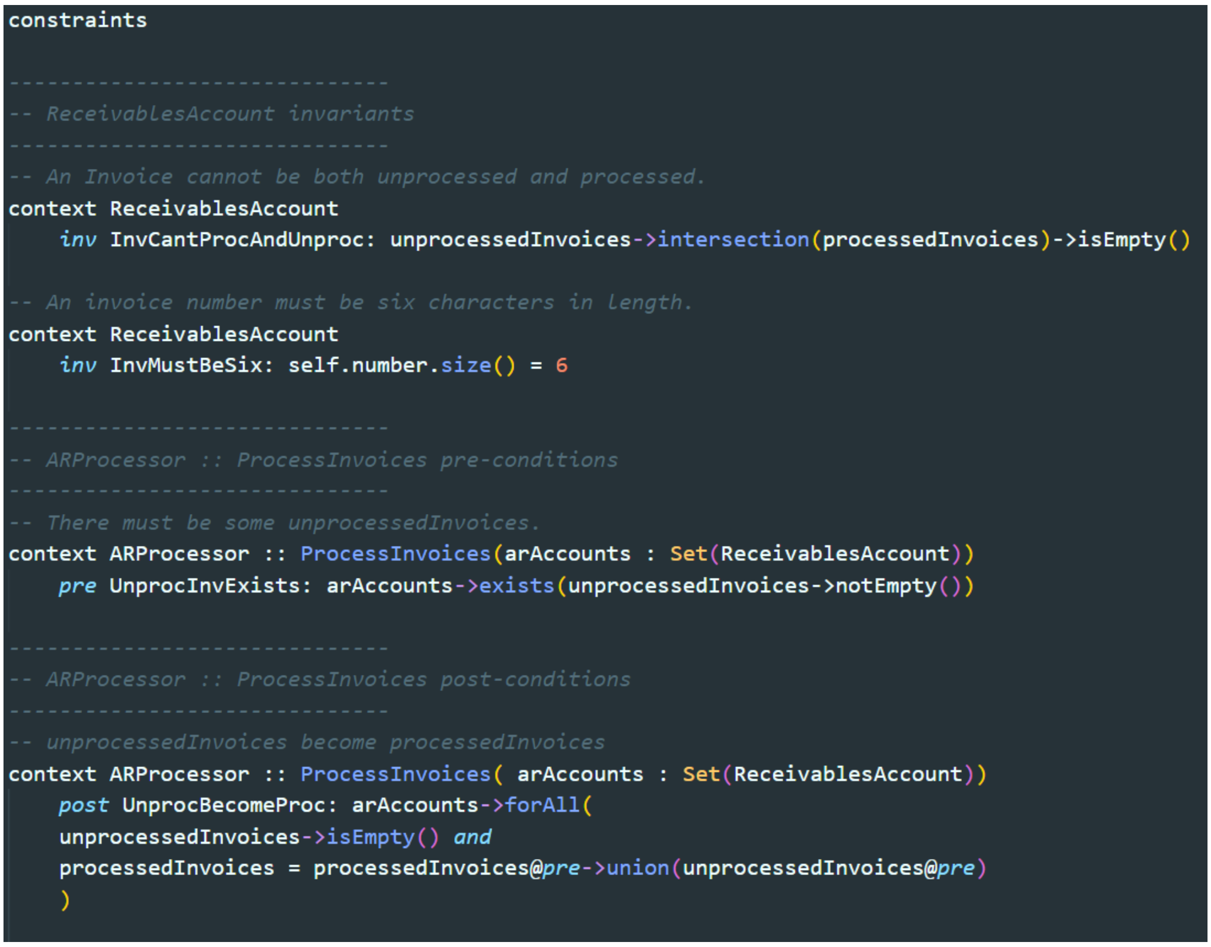
This is the association between the Customer and ReceivablesAccount.

The relationship between the ReceivablesAccount and the Unprocessed Invoices.

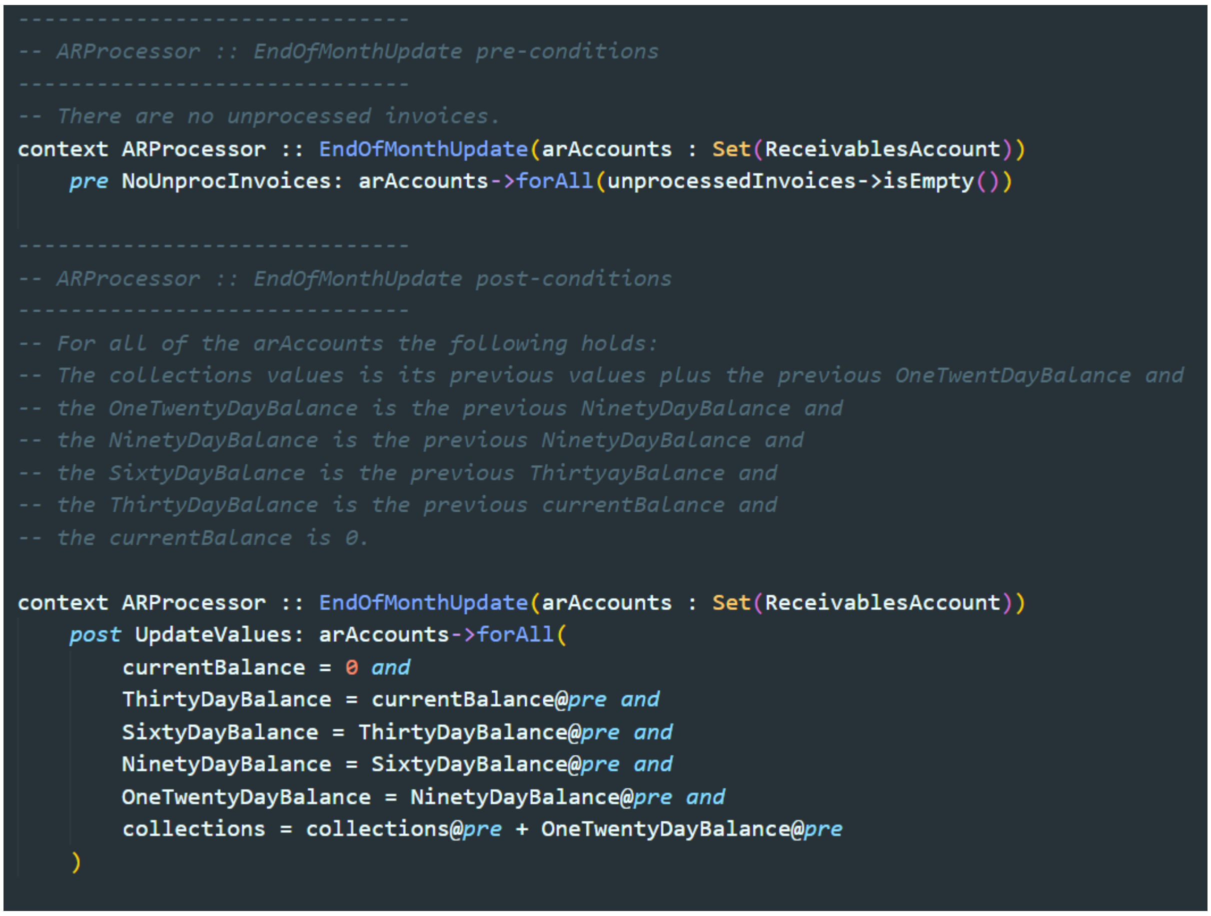
The relationship between the ReceivablesAccount and the Processed Invoices.

These associations were implemented as intended, only adding certain role and association names, for clarity, and to ensure it would work with USE.

Constraints:



```
constraints

------------------------------
-- ReceivablesAccount invariants
------------------------------
-- An Invoice cannot be both unprocessed and processed.
context ReceivablesAccount
    inv InvCantProcAndUnproc: unprocessedInvoices->intersection(processedInvoices)->isEmpty()

-- An invoice number must be six characters in length.
context ReceivablesAccount
    inv InvMustBeSix: self.number.size() = 6

------------------------------
-- ARProcessor :: ProcessInvoices pre-conditions
------------------------------
-- There must be some unprocessedInvoices.
context ARProcessor :: ProcessInvoices(arAccounts : Set(ReceivablesAccount))
    pre UnprocInvExists: arAccounts->exists(unprocessedInvoices->notEmpty())

------------------------------
-- ARProcessor :: ProcessInvoices post-conditions
------------------------------
-- unprocessedInvoices become processedInvoices
context ARProcessor :: ProcessInvoices( arAccounts : Set(ReceivablesAccount))
    post UnprocBecomeProc: arAccounts->forAll(
    unprocessedInvoices->isEmpty() and
    processedInvoices = processedInvoices@pre->union(unprocessedInvoices@pre)
    )
```



```
------------------------------
-- ARProcessor :: EndOfMonthUpdate pre-conditions
------------------------------
-- There are no unprocessed invoices.
context ARProcessor :: EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
    pre NoUnprocInvoices: arAccounts->forAll(unprocessedInvoices->isEmpty())

------------------------------
-- ARProcessor :: EndOfMonthUpdate post-conditions
------------------------------
-- For all of the arAccounts the following holds:
-- The collections values is its previous values plus the previous OneTwentDayBalance and
-- the OneTwentyDayBalance is the previous NinetyDayBalance and
-- the NinetyDayBalance is the previous NinetyDayBalance and
-- the SixtyDayBalance is the previous ThirtyayBalance and
-- the ThirtyDayBalance is the previous currentBalance and
-- the currentBalance is 0.

context ARProcessor :: EndOfMonthUpdate(arAccounts : Set(ReceivablesAccount))
    post UpdateValues: arAccounts->forAll(
        currentBalance = 0 and
        ThirtyDayBalance = currentBalance@pre and
        SixtyDayBalance = ThirtyDayBalance@pre and
        NinetyDayBalance = SixtyDayBalance@pre and
        OneTwentyDayBalance = NinetyDayBalance@pre and
        collections = collections@pre + OneTwentyDayBalance@pre
    )
```

These were the constraints/assertations I was required to implement. I did this by using invariants, preconditions, and post conditions.

What I implemented was:

<u>ReceivablesAccount invariants</u>

1. An invoice cannot be both unprocessed and processed

2. An invoice number must be six characters in length

<u>ARProcessor ProcessInvoices() – pre & post conditions</u>

1. There must be some unprocessed invoices

2. Unprocessed invoices become Processed invoices

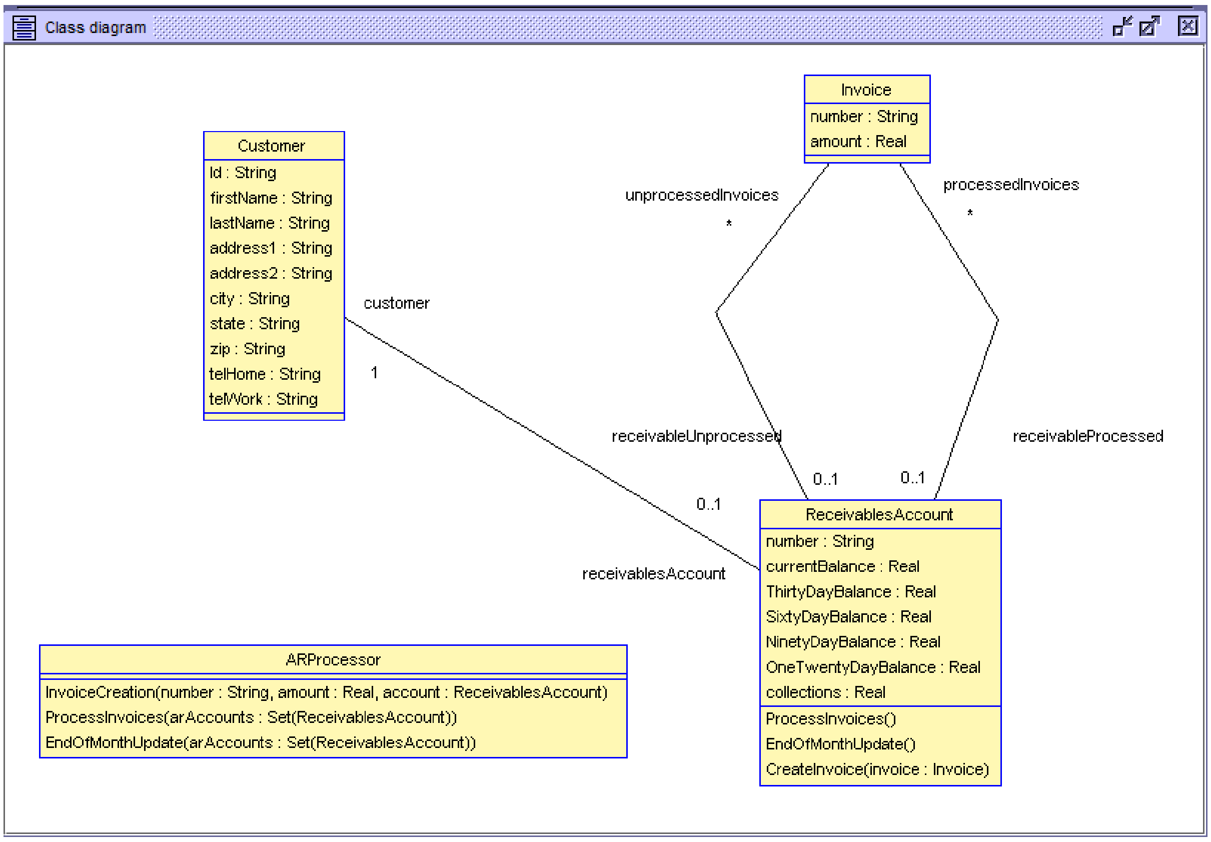
<u>ARProcessor EndOfMonthUpdate() – pre & post conditions</u>

1. There are no unprocessed invoices

2. The collections value is its previous values plus the previous OneTwentyDayBalance, and so on.

3. Diagrams

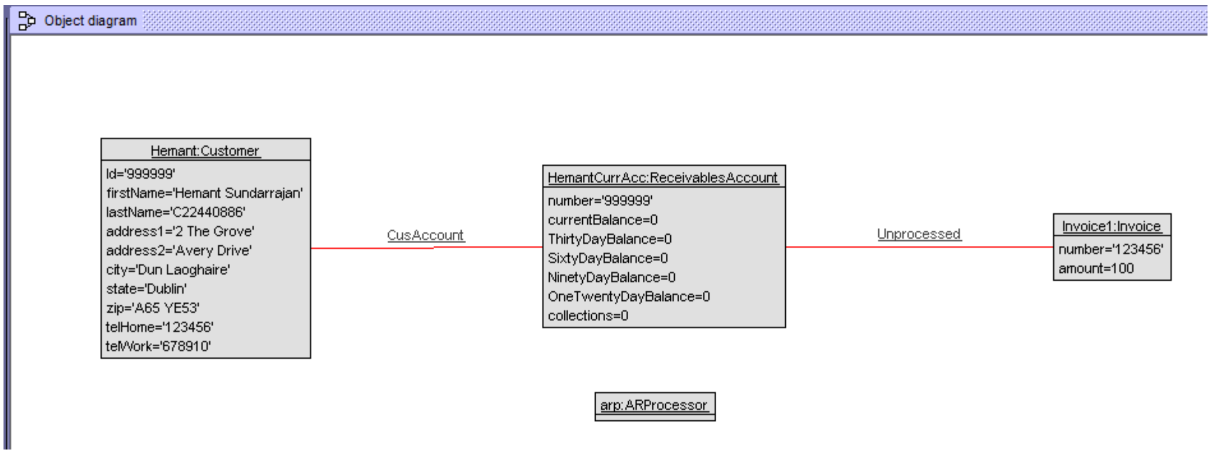
Class Diagram:



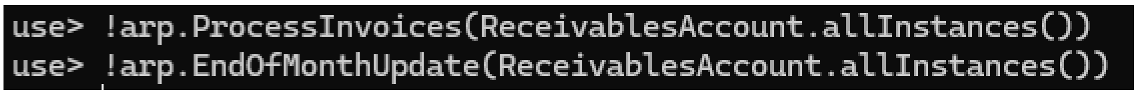
Implemented exactly as intended, only difference is the addition of necessary role names, and operations.

Object Diagrams:

This is my object diagram after creating an invoice. As you can see it is currently Unprocessed.

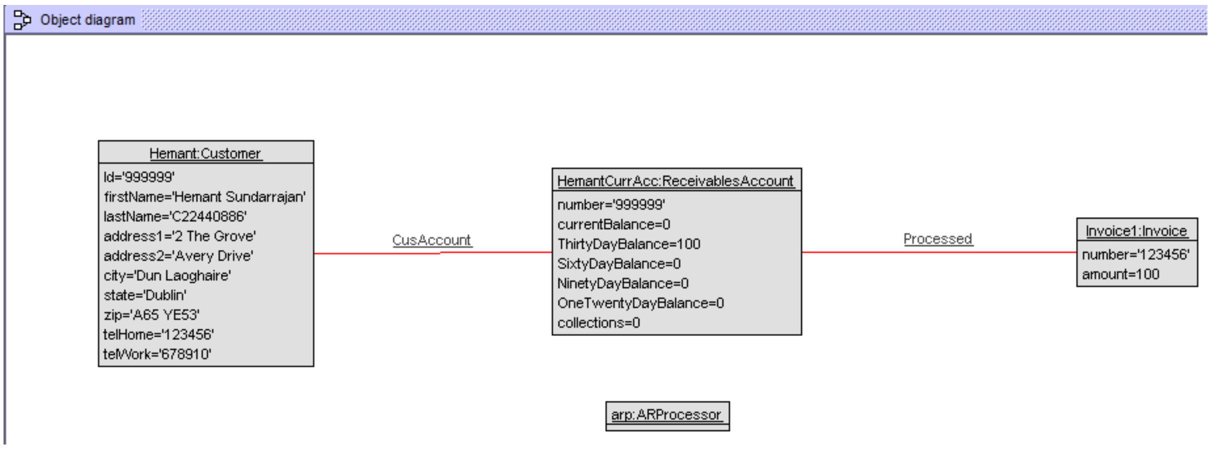


After calling these two operations:



```
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
use> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
```

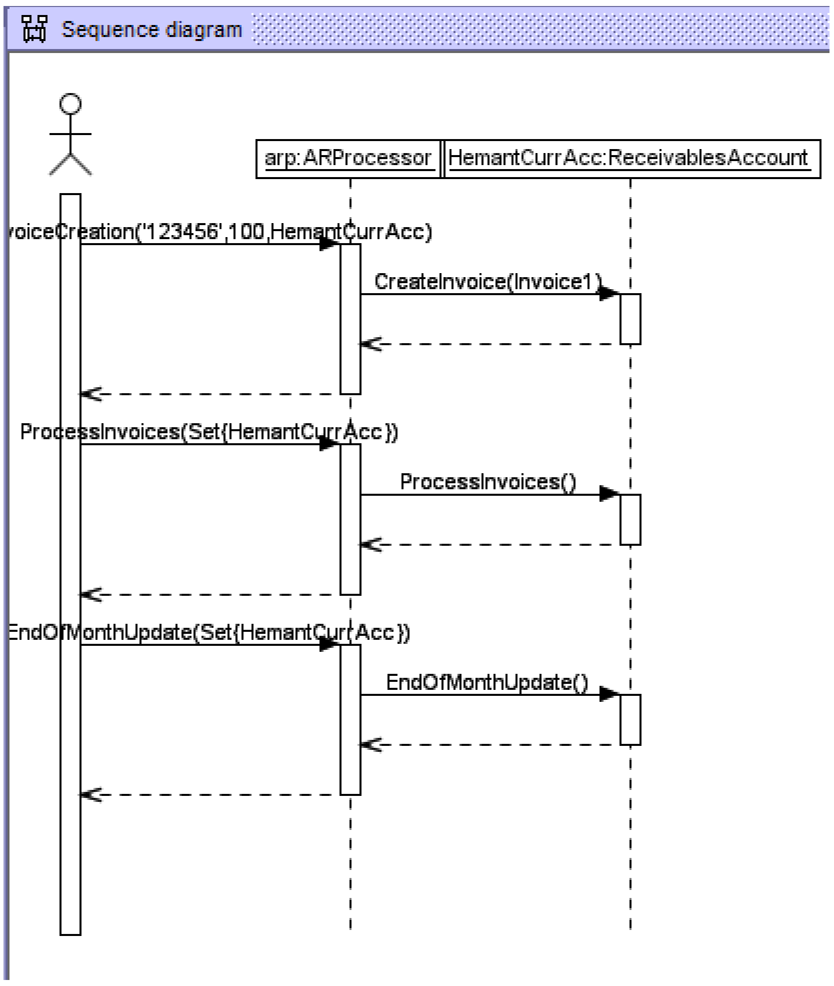
My object diagram looks like this:



As you can see, the invoice is now correctly processed, and the EndOfMonthUpdate worked as intended, as the ThirtyDayBalance is now the amount of the invoice that was processed.

Sequence Diagram:

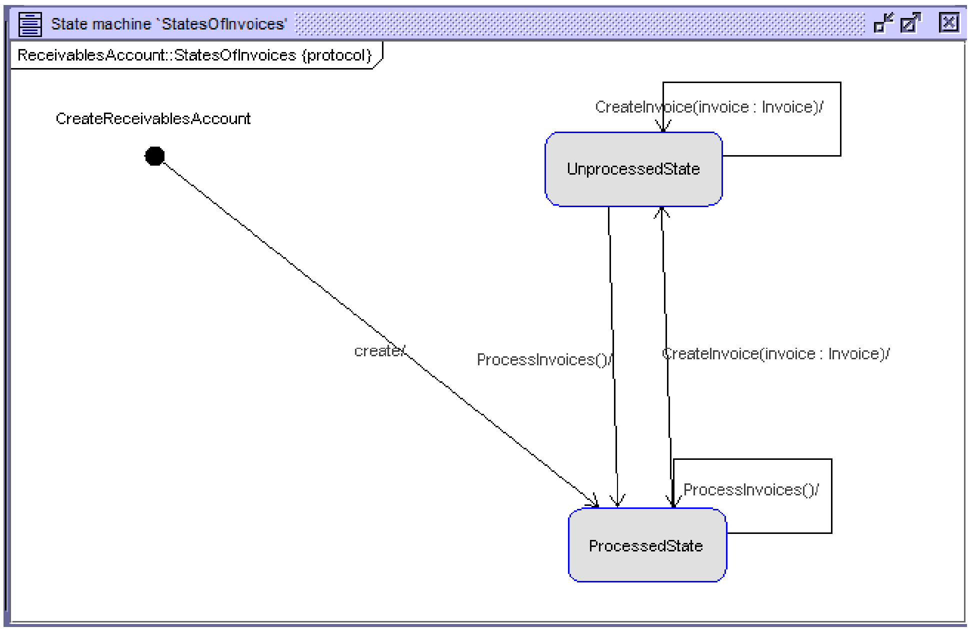
Sequence Diagram of the three operations being called.



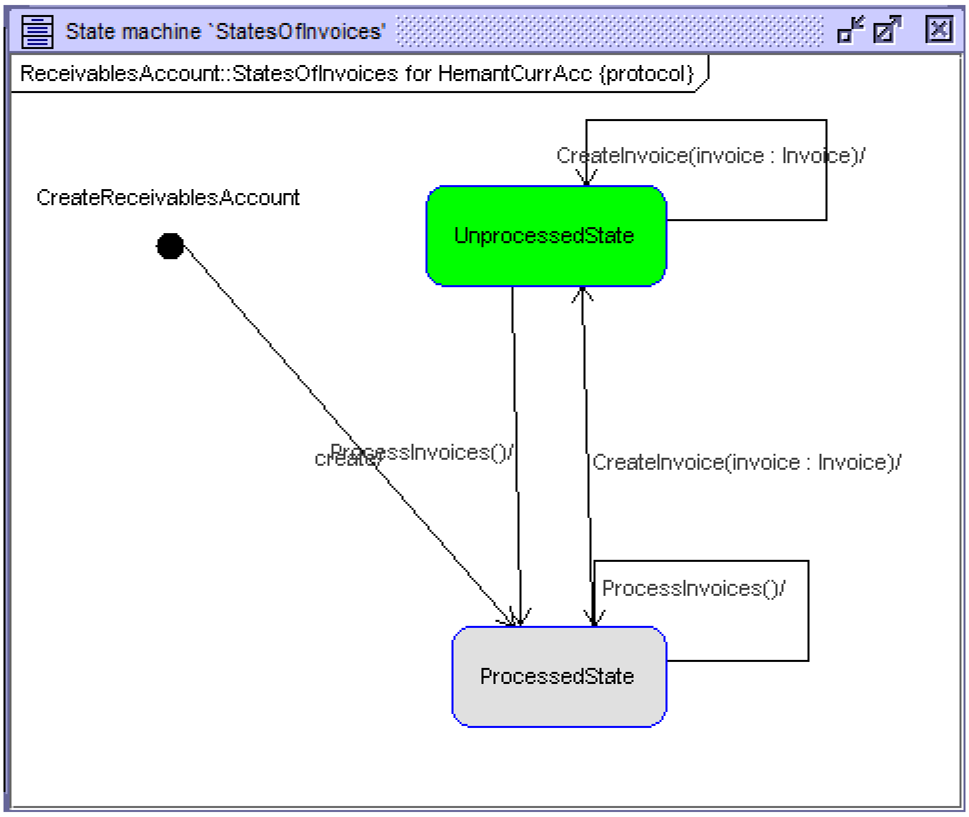
Sequence diagram
arp:ARProcessor
HemantCurrAcc:ReceivablesAccount
voiceCreation('123456',100,HemantCurrAcc)
CreateInvoice(Invoice1)
ProcessInvoices(Set{HemantCurrAcc})
ProcessInvoices()
EndOfMonthUpdate(Set{HemantCurrAcc})
EndOfMonthUpdate()

StateMachine Diagrams:

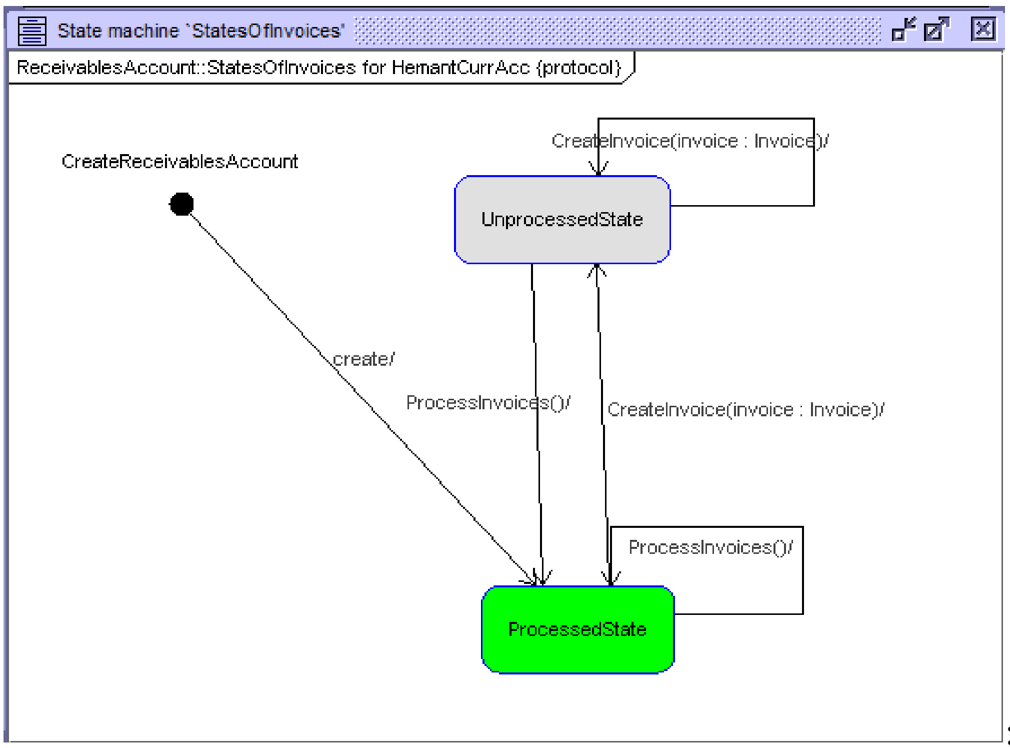
ReceivablesAccount Statemachine before any processes:


State machine `StatesOfInvoices'
ReceivablesAccount::StatesOfInvoices {protocol}
CreateReceivablesAccount
create/
CreateInvoice(invoice : Invoice)/
UnprocessedState
ProcessInvoices()/
CreateInvoice(invoice : Invoice)/
ProcessInvoices()/
ProcessedState

ReceivablesAccount Statemachine After Invoice creation:

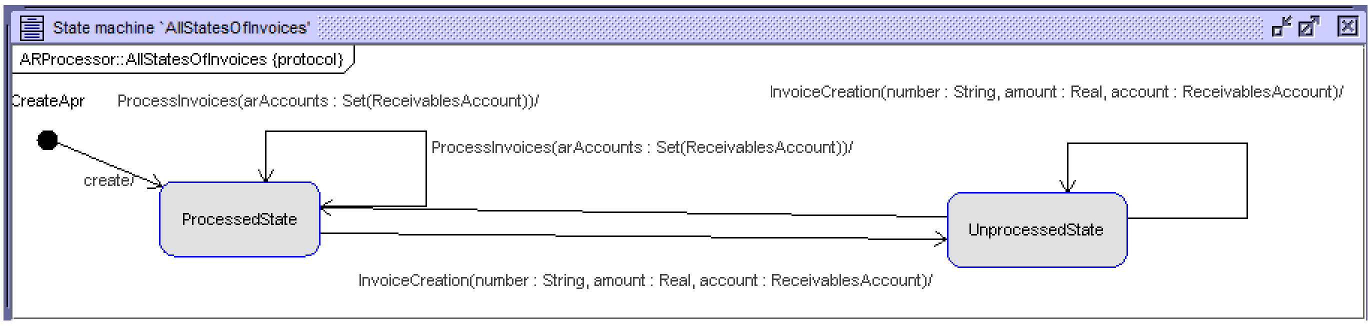

State machine `StatesOfInvoices'
ReceivablesAccount::StatesOfInvoices for HemantCurrAcc {protocol}
CreateInvoice(invoice : Invoice)/
CreateReceivablesAccount
UnprocessedState
create/
ProcessInvoices()/
CreateInvoice(invoice : Invoice)/
ProcessInvoices()/
ProcessedState

ReceivablesAccount Statemachine after creation/when invoice processed:

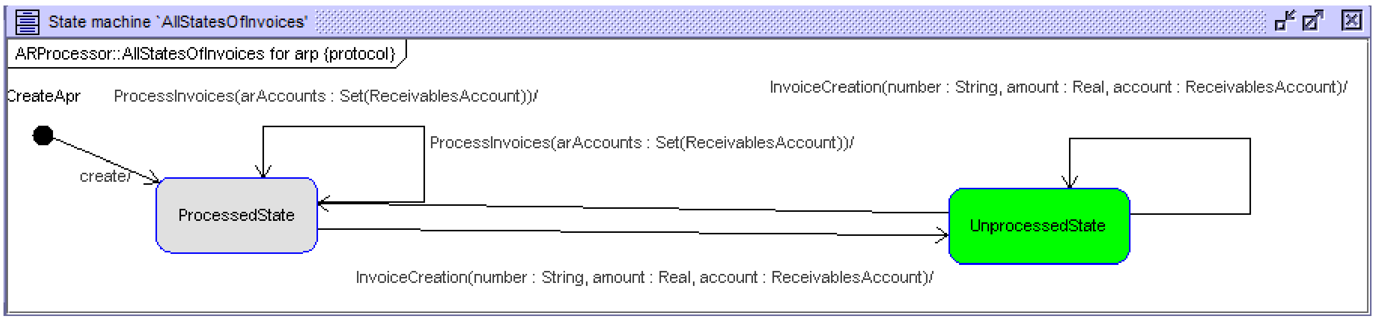

State machine `StatesOfInvoices'
ReceivablesAccount::StatesOfInvoices for HemantCurrAcc {protocol}
CreateReceivablesAccount
create/
CreateInvoice(invoice : Invoice)/
UnprocessedState
ProcessInvoices()/
CreateInvoice(invoice : Invoice)/
ProcessInvoices()/
ProcessedState

:

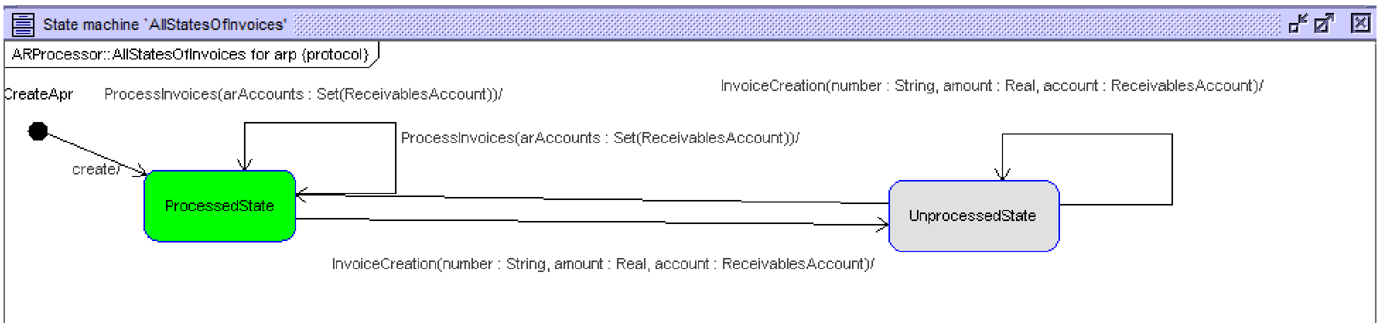
ARProcessor Statemachine before processes occur:


State machine `AllStatesOfInvoices'
ARProcessor::AllStatesOfInvoices {protocol}
CreateApr
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
create/
ProcessedState
UnprocessedState
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/

ARProcessor Statemachine after invoice is created:


State machine `AllStatesOfInvoices'
ARProcessor::AllStatesOfInvoices for arp {protocol}
CreateApr
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
create/
ProcessedState
UnprocessedState
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/

ARProcessor Statemachine after creation/when invoice is processed:

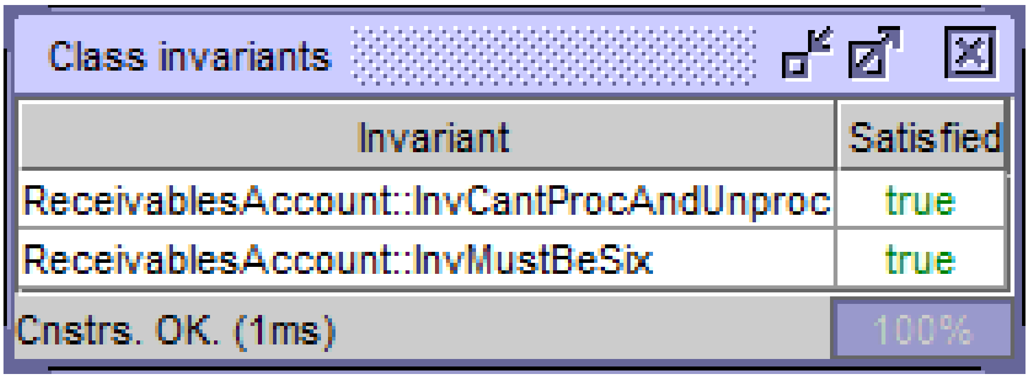

State machine `AllStatesOfInvoices'
ARProcessor::AllStatesOfInvoices for arp {protocol}
CreateApr
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/
ProcessInvoices(arAccounts : Set(ReceivablesAccount))/
create/
ProcessedState
UnprocessedState
InvoiceCreation(number : String, amount : Real, account : ReceivablesAccount)/

4.Testing Constraints

ReceivableAccount invariants

1. An invoice cannot be both unprocessed and processed

Before:

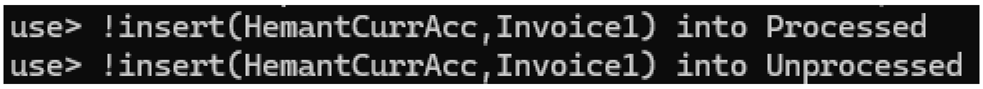


Class invariants

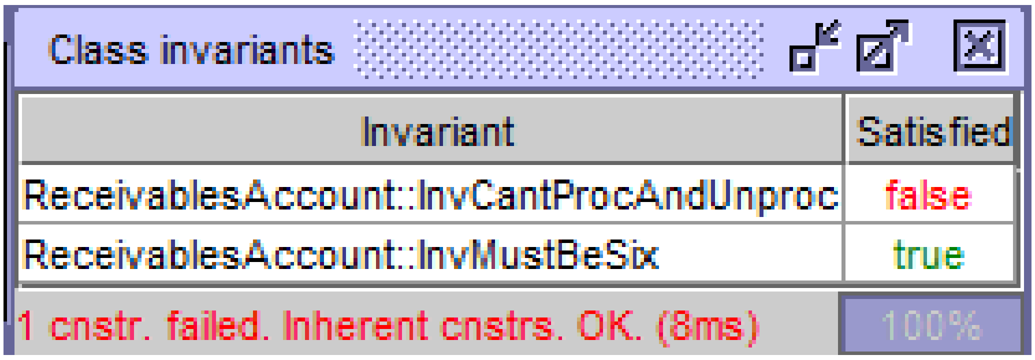
Invariant	Satisfied
ReceivablesAccount::InvCantProcAndUnproc	true
ReceivablesAccount::InvMustBeSix	true

Cnstrs. OK. (1ms) 100%

After calling these two lines of soil (attempting to process and unprocess):



```
use> !insert(HemantCurrAcc,Invoice1) into Processed
use> !insert(HemantCurrAcc,Invoice1) into Unprocessed
```



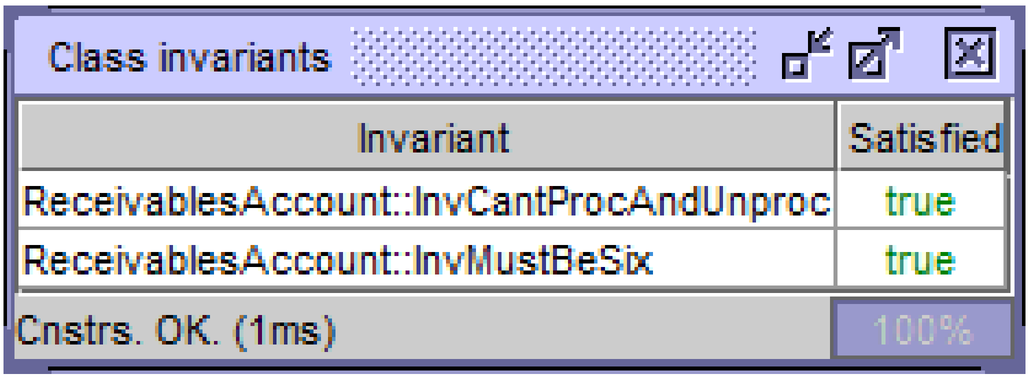
Class invariants

Invariant	Satisfied
ReceivablesAccount::InvCantProcAndUnproc	false
ReceivablesAccount::InvMustBeSix	true

1 cnstr. failed. Inherent cnstrs. OK. (8ms) 100%

2. An invoice number must be six characters in length

Before:

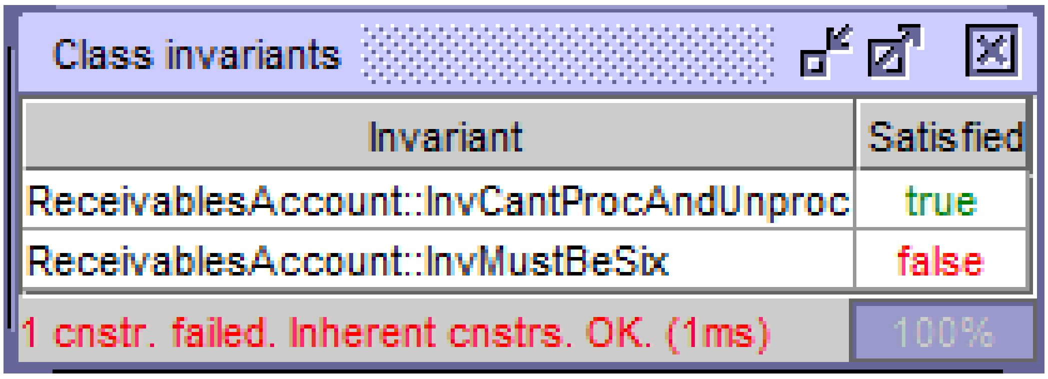


Invariant	Satisfied
ReceivablesAccount::InvCantProcAndUnproc	true
ReceivablesAccount::InvMustBeSix	true

Cnstrs. OK. (1ms) 100%

After attempting to make an invoice number 7 characters in length:

```
use> !HemantCurrAcc.number := '1234567'
```



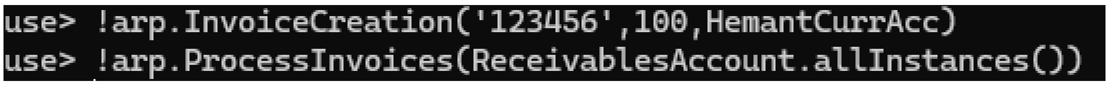
Invariant	Satisfied
ReceivablesAccount::InvCantProcAndUnproc	true
ReceivablesAccount::InvMustBeSix	false

1 cnstr. failed. Inherent cnstrs. OK. (1ms) 100%

ARProcessor ProcessInvoices() PRE-condition

1. There must be some unprocessedInvoices.

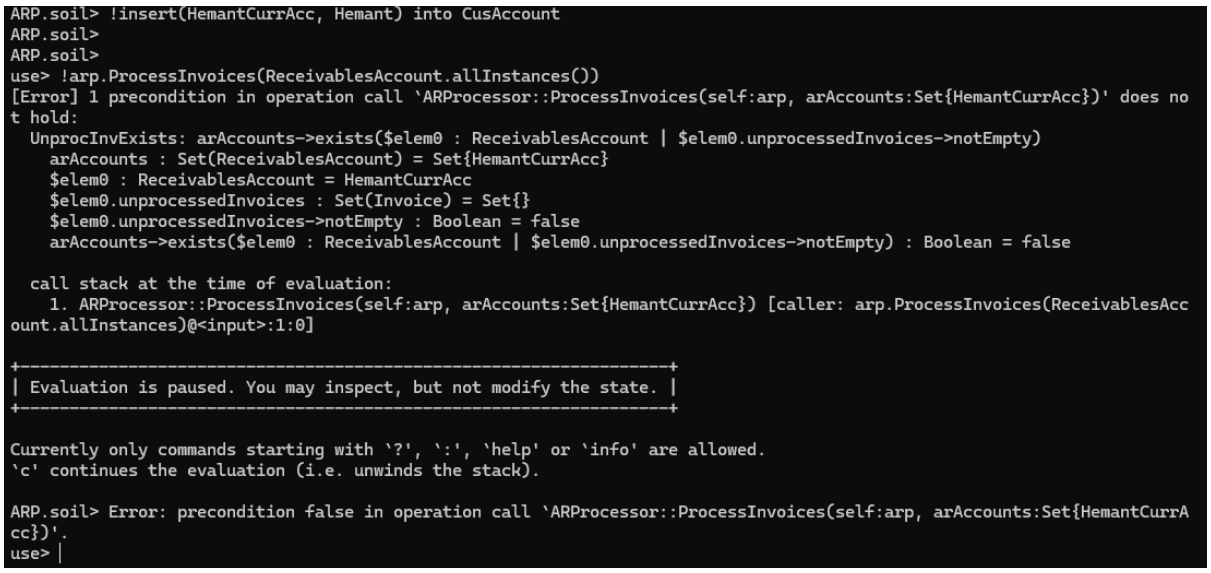
Before:



```
use> !arp.InvoiceCreation('123456',100,HemantCurrAcc)
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
```

Works as intended, as there is a invoice created (which is unprocessed), so when ProcessInvoices() is called, the pre-condition isn't false, and the code runs correctly runs correctly.

After:



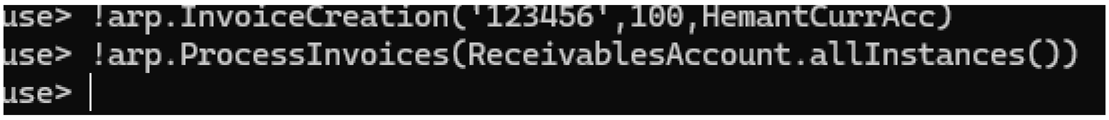
```
ARP.soil> !insert(HemantCurrAcc, Hemant) into CusAccount
ARP.soil>
ARP.soil>
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
[Error] 1 precondition in operation call `ARProcessor::ProcessInvoices(self:arp, arAccounts:Set{HemantCurrAcc})' does no
t hold:
  UnprocInvExists: arAccounts->exists($elem0 : ReceivablesAccount | $elem0.unprocessedInvoices->notEmpty)
    arAccounts : Set(ReceivablesAccount) = Set{HemantCurrAcc}
    $elem0 : ReceivablesAccount = HemantCurrAcc
    $elem0.unprocessedInvoices : Set(Invoice) = Set{}
    $elem0.unprocessedInvoices->notEmpty : Boolean = false
    arAccounts->exists($elem0 : ReceivablesAccount | $elem0.unprocessedInvoices->notEmpty) : Boolean = false

  call stack at the time of evaluation:
    1. ARProcessor::ProcessInvoices(self:arp, arAccounts:Set{HemantCurrAcc}) [caller: arp.ProcessInvoices(ReceivablesAcc
ount.allInstances)@<input>:1:0]

+------------------------------------------------------------------+
| Evaluation is paused. You may inspect, but not modify the state. |
+------------------------------------------------------------------+

Currently only commands starting with `?', `:', `help' or `info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).

ARP.soil> Error: precondition false in operation call `ARProcessor::ProcessInvoices(self:arp, arAccounts:Set{HemantCurrA
cc})'.
use> |
```

As I didn't create any invoices, the pre-condition fails when I attempt to process the invoices.

ARProcessor ProcessInvoices() POST-condition

1. Unprocessed invoices become Processed invoices

For this post condition, I can't show it failing. It simply deletes all associations for unprocessed invoices and inserts them into processed invoices. The soil commands I use to do this are as follows:



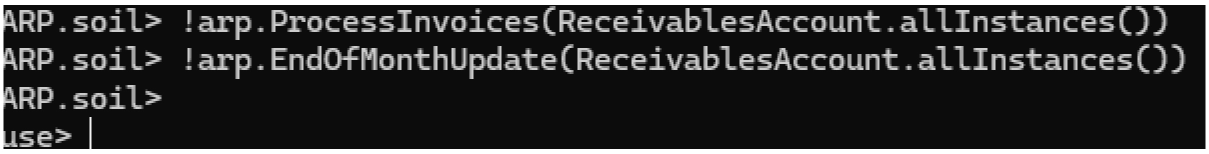
```
use> !arp.InvoiceCreation('123456',100,HemantCurrAcc)
use> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
use> |
```

If you look at the state machine diagrams above, you can see how this occurs, with the state changing from unprocessed to processed when this operation is called.

ARProcessor EndOfMonthUpdate() PRE-condition

1. There are no unprocessed invoices

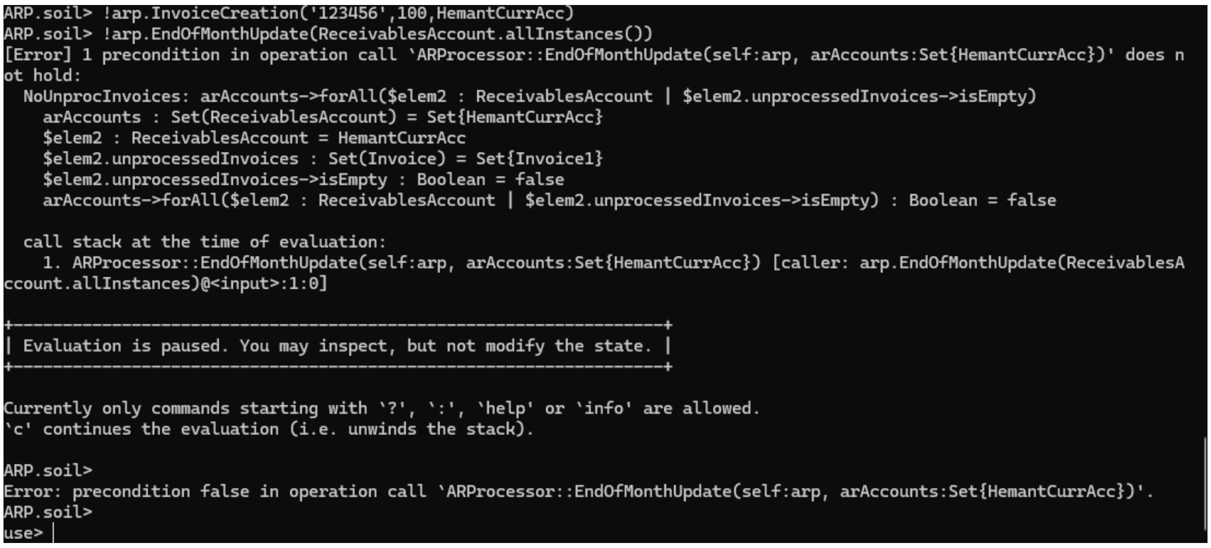
Before:



```
ARP.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
ARP.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
ARP.soil>
use> |
```

Works as intended if you call ProcessInvoices() operation prior to calling EndOfMonthUpdate() operation.

After:



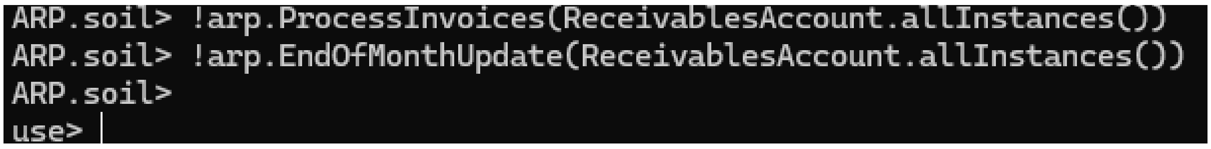
```
ARP.soil> !arp.InvoiceCreation('123456',100,HemantCurrAcc)
ARP.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
[Error] 1 precondition in operation call `ARProcessor::EndOfMonthUpdate(self:arp, arAccounts:Set{HemantCurrAcc})' does n
ot hold:
  NoUnprocInvoices: arAccounts->forAll($elem2 : ReceivablesAccount | $elem2.unprocessedInvoices->isEmpty)
    arAccounts : Set(ReceivablesAccount) = Set{HemantCurrAcc}
    $elem2 : ReceivablesAccount = HemantCurrAcc
    $elem2.unprocessedInvoices : Set(Invoice) = Set{Invoice1}
    $elem2.unprocessedInvoices->isEmpty : Boolean = false
    arAccounts->forAll($elem2 : ReceivablesAccount | $elem2.unprocessedInvoices->isEmpty) : Boolean = false

  call stack at the time of evaluation:
    1. ARProcessor::EndOfMonthUpdate(self:arp, arAccounts:Set{HemantCurrAcc}) [caller: arp.EndOfMonthUpdate(ReceivablesA
ccount.allInstances)@<input>:1:0]

+---------------------------------------------------------------+
| Evaluation is paused. You may inspect, but not modify the state. |
+---------------------------------------------------------------+

Currently only commands starting with `?', `:', `help' or `info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).

ARP.soil>
Error: precondition false in operation call `ARProcessor::EndOfMonthUpdate(self:arp, arAccounts:Set{HemantCurrAcc})'.
ARP.soil>
use> |
```

As you can see we get a pre-condition error. You can’t call this operation unless all invoices are processed prior to it being called with the ProcessInvoices() operation.

ARProcessor EndOfMonthUpdate() POST-condition

1. Where the collections values are the previous values plus the previous 120-day balance. Where the 120-day balance is the previous 90-day balance and so on. Collections->120->90->60->30->current balance. The current balance is reset to 0.

Post-condition working as intended:



```
ARP.soil> !arp.ProcessInvoices(ReceivablesAccount.allInstances())
ARP.soil> !arp.EndOfMonthUpdate(ReceivablesAccount.allInstances())
ARP.soil>
use>
```

I can't display it not working, as when the operation is called, it works instantly. Other conditions would stop me from calling this operation if an invoice isn't created. The post-condition works as intended.

5. Conclusion

The assignment has been completed in its fullest form. I have constructed the ARP and DbC as required. This involved constructing and implementing classes, associations, and constraints to achieve the completed ARP. By completing this assignment, I gained a better understanding into software engineering principles. I learnt in depth how invariants, post, and preconditions work, and how they impact software design. I showed how the operations work in-depth using state machine diagrams, sequence diagrams, class diagrams and object diagrams. I displayed a high level of proficiency in my software designing skills, and I made alterations where required with the USE syntax to ensure the ARP was constructed as intended.