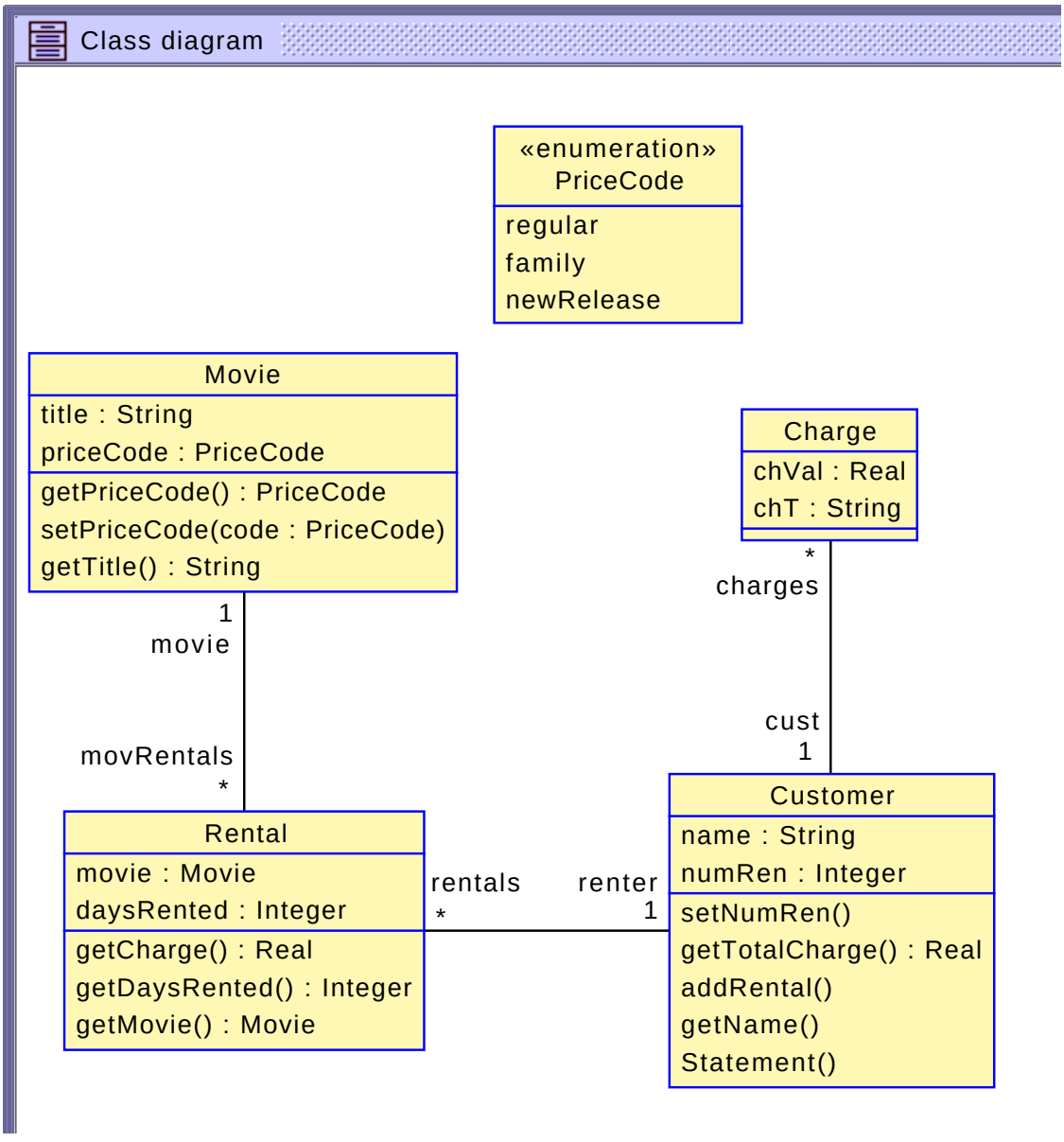
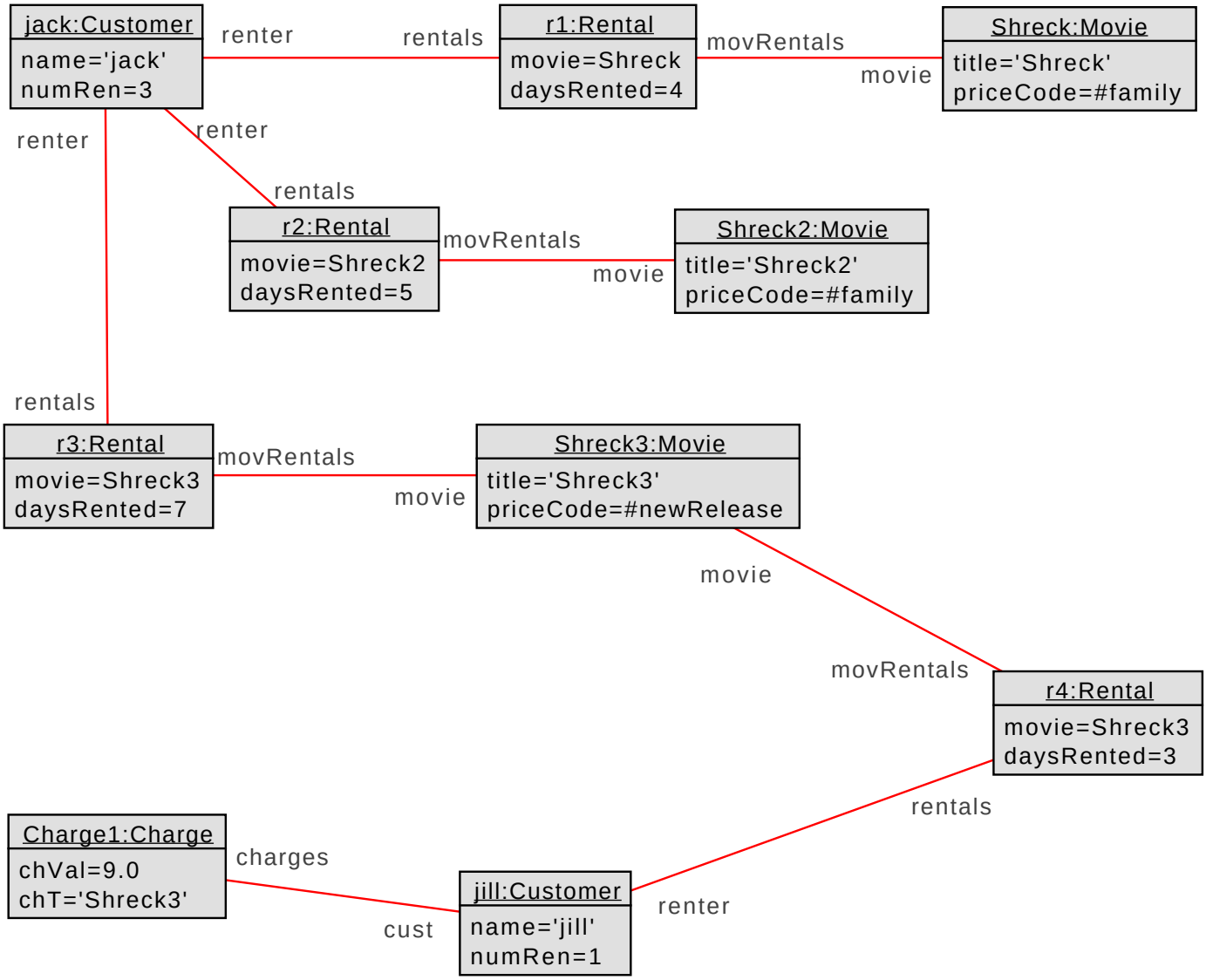
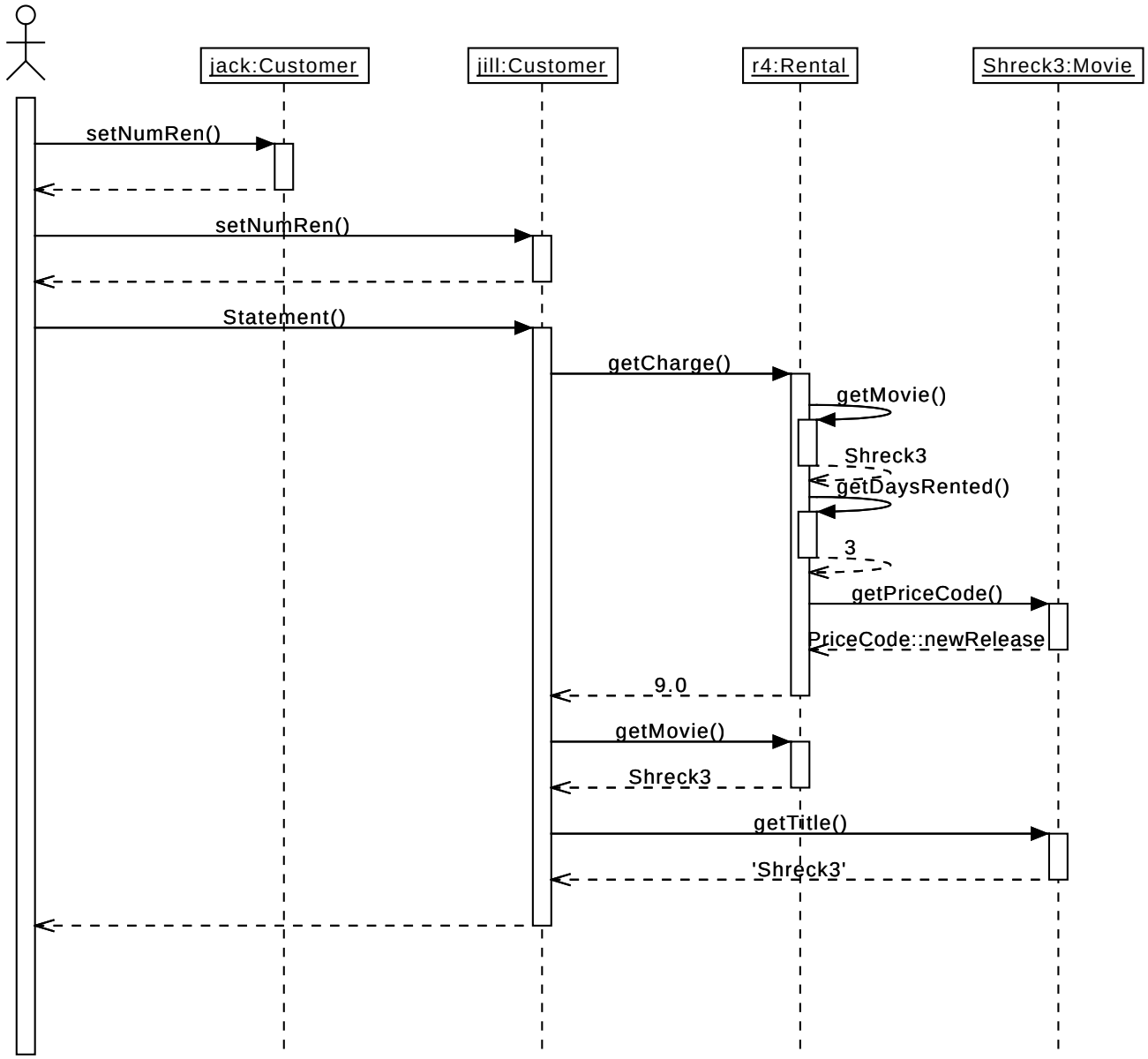
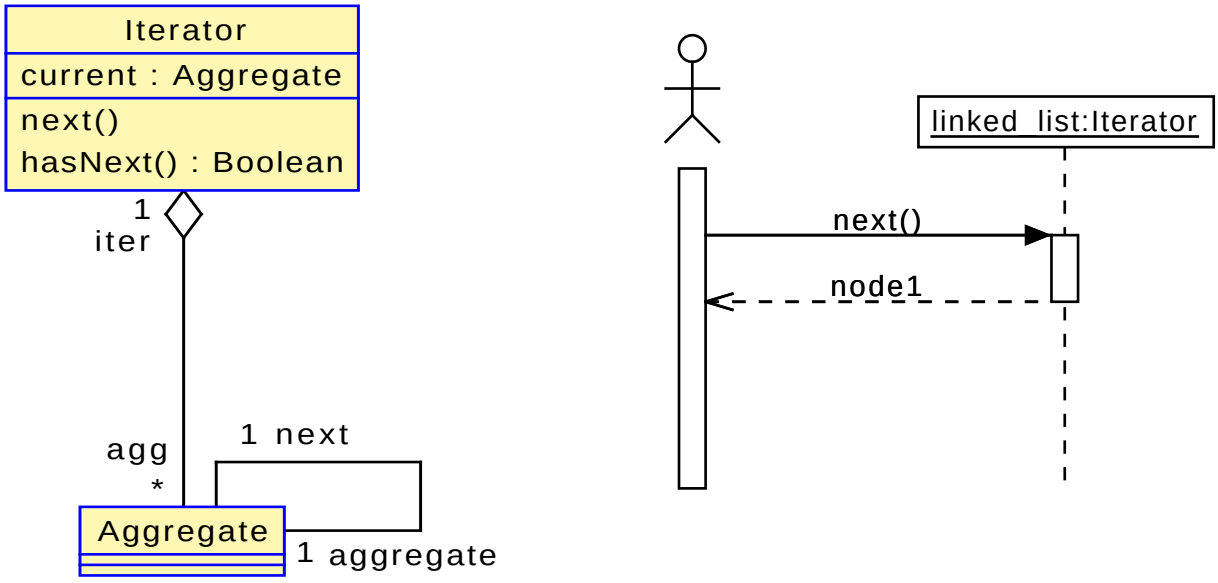
Brett Shelley
Alex Irish
ESOF 422
HW2

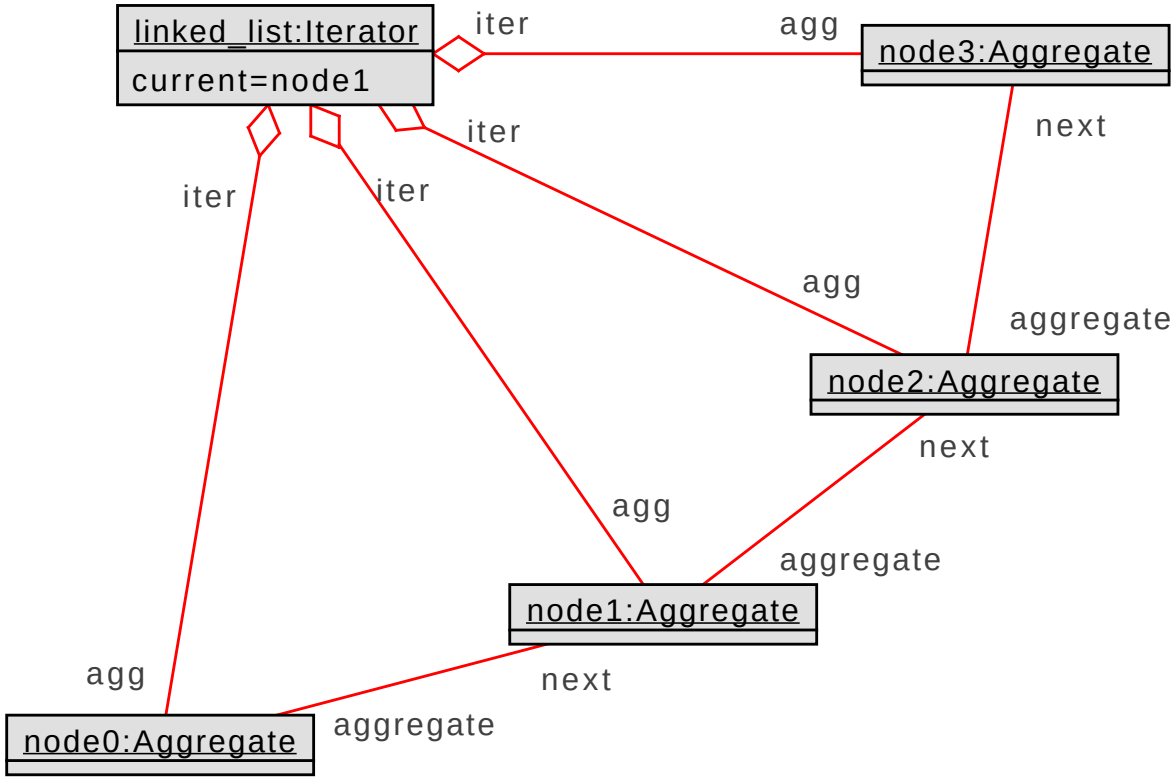



jack:Customer
name='jack'
numRen=3
renter
rentals
r1:Rental
movie=Shreck
daysRented=4
movRentals
movie
Shreck:Movie
title='Shreck'
priceCode=#family
renter
renter
rentals
r2:Rental
movie=Shreck2
daysRented=5
movRentals
movie
Shreck2:Movie
title='Shreck2'
priceCode=#family
rentals
r3:Rental
movie=Shreck3
daysRented=7
movRentals
movie
Shreck3:Movie
title='Shreck3'
priceCode=#newRelease
movie
movRentals
r4:Rental
movie=Shreck3
daysRented=3
rentals
Charge1:Charge
chVal=9.0
chT='Shreck3'
charges
cust
jill:Customer
name='jill'
numRen=1
renter


jack:Customer
jill:Customer
r4:Rental
Shreck3:Movie
setNumRen()
setNumRen()
Statement()
getCharge()
getMovie()
Shreck3
getDaysRented()
3
getPriceCode()
PriceCode::newRelease
9.0
getMovie()
Shreck3
getTitle()
'Shreck3'

Iterator Pattern: An iterator class is used to look through the elements of of a collection by looking at each element sequentially by looking at an element followed by the element designated by the next pointer in the current element.





Iterator.use

```
-- USE model for an Iterator --

model Iterator

-- classes
class Aggregate
attributes
operations
end

class Iterator
attributes
  current:Aggregate
operations
  next():Aggregate
    begin
     self.current := self.current.next;
     result := self.current;
    end

  hasNext():Boolean
    begin
      result := self.current.next->notEmpty;
    end
end

-- associations

aggregation creates between
  Iterator [1] role iter
  Aggregate [0..*] role agg
end

association next between
  Aggregate [1]
  Aggregate [1] role next
end

-- constraints
constraints
context Iterator::next():Aggregate
  pre nextPre: current.next->notEmpty
```

Iterator.x

```
-- Object models commands for Iterator

-- create objects
!create linked_list:Iterator
!create node0:Aggregate
!create node1:Aggregate
!create node2:Aggregate
!create node3:Aggregate

-- create associations
!insert (linked_list,node0) into creates
!insert (linked_list,node1) into creates
!insert (linked_list,node2) into creates
!insert (linked_list,node3) into creates
!insert (node0,node1) into next
!insert (node1,node2) into next
!insert (node2,node3) into next

-- assign varibles
!set linked_list.current := node0

-- call next
! linked_list.next()
```