03-60-322 Assignment 3

Part 1)

1) Yes, there are programming activities happening in the inception phase. This is because during the inception phase a team must gather just enough evidence to support a decision to continue or stop. Sometimes in order to prove feasibility early prototypes and proof-of-concepts must be done

2)a) The text book discusses use cases first before object oriented analysis and design techniques are introduced because OOA/D is normally preceded by requirements analysis. Use cases are an important part of requirement analysis. Therefore the initial lectures introduce the important topics of uses cases and requirements analysis, which are then followed by topics on OOA/D and more UML details. Uses Cases are also an input to many of the other OOAC techniques and would thus, have to be understood first.

b) 3 use cases for the DVD rental store would be

(*Assumptions: Accounting system and Register is not built into the stores information system. No money is exchanged during a DVD return, thus, accounting system, register and tax calculator are not needed.)*

1) "Returning DVDs"

Actors: -customer returning DVDs to the store
-cashier
-store information system
(Assumption: no need of register)

2)"Returning DVDs with late fees"

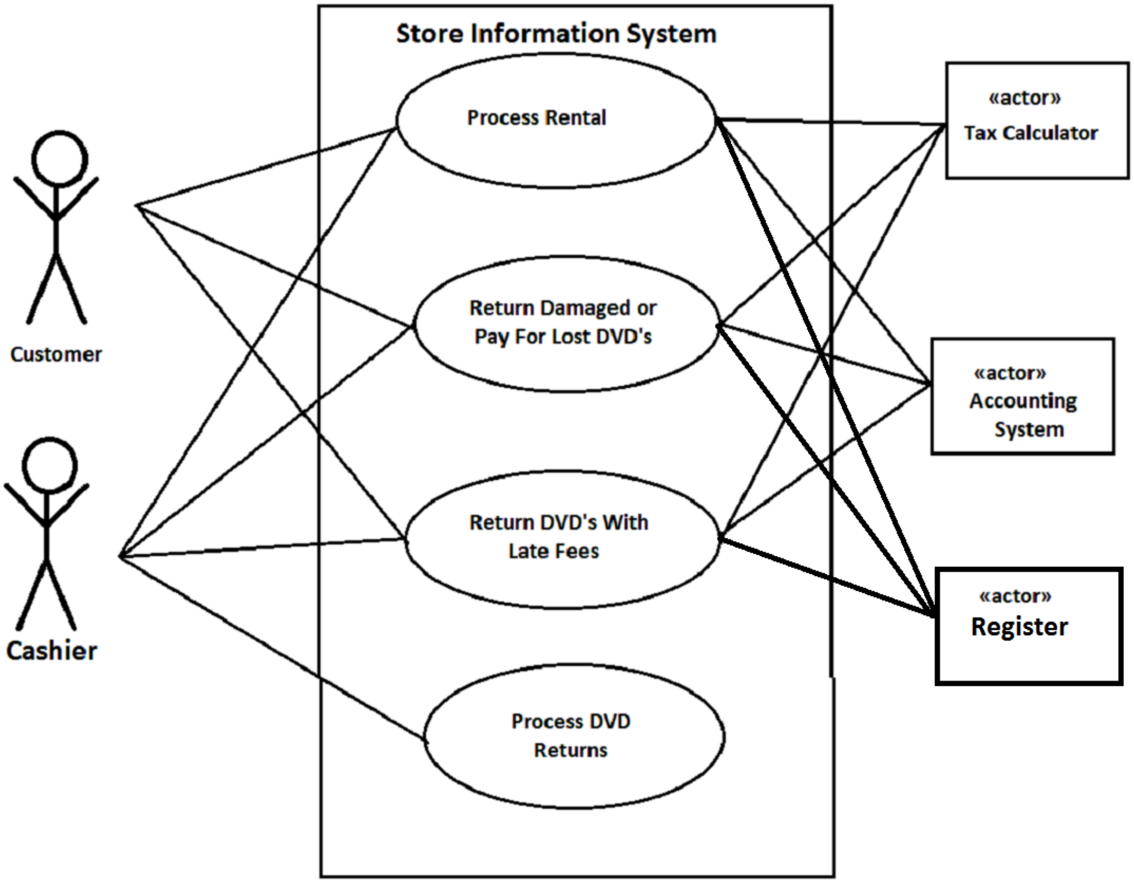
Actors: -customer returning DVDS
-cashier
-store information system
-tax calculator
-Accounting System
-register

3)"Returning damaged or lost DVDs"

Actors: -customer
-cashier
-store information system
-tax calculator
-Accounting System
-register

I chose these three use cases because to me they are critical to the operation of the DVD Rental store and therefore have high business value.

Use Case Diagram



3)

Use Case 1. Rent DVDs

Actors:Customer, Cashier, SuD, register, accounting system, and tax calculator

Basic Flow

1. Customer arrives at checkout with DVDs he wishes to rent
2. Cashier starts a new rental
3. Cashier asks for Customers phone number to let the system know which customer account the rental belongs to
4. Cashier enters DVD identifier
5. System records DVD item and presents item description, price, and running total.
   Cashier repeats steps 4-5 until done
6. System presents total with taxes calculated
7. Cashier ask customer for payment
8. Customer pays and store information system handles payment
9. System logs completed rental in a ledger
10. System updates status of rented DVDs in inventory to "rented" and changes their due date to a week after time of rental
11. System adds DVDs to currently rented list on customer's membership profile
13. System prints receipt with information such a the DVDs rented, price, date and time of

rental, and due date.

14. Customer leaves with receipt and DVDs

Alternative Flow

3a. Customer is not a member

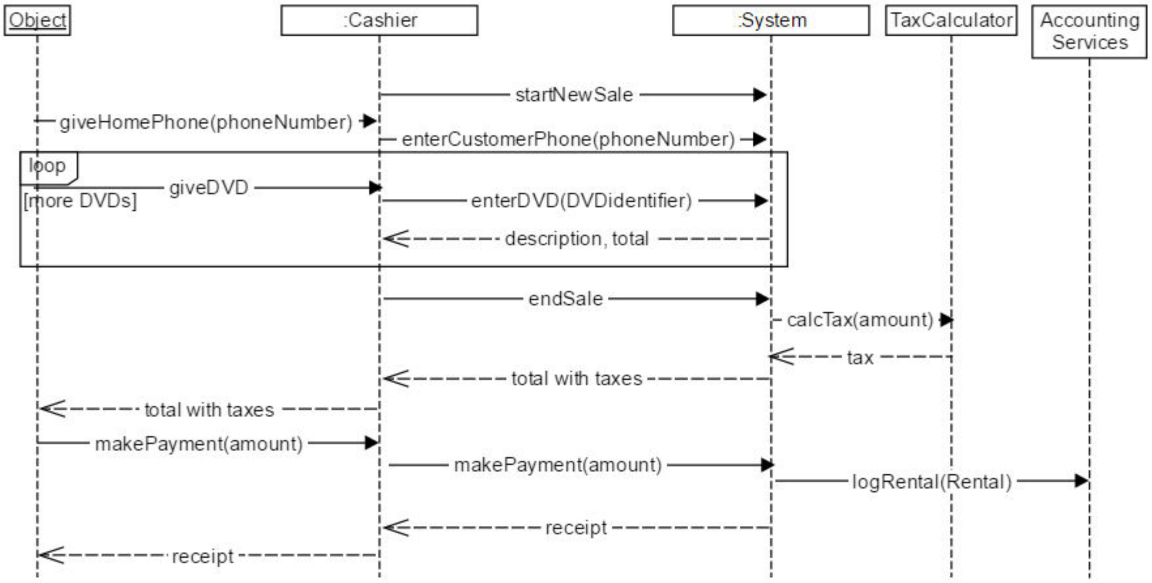
1)cashier asks for customer information and inputs it to the system

2)system creates new member and starts a sale for that member

3-8a. Customer asks Cashier to cancel rental

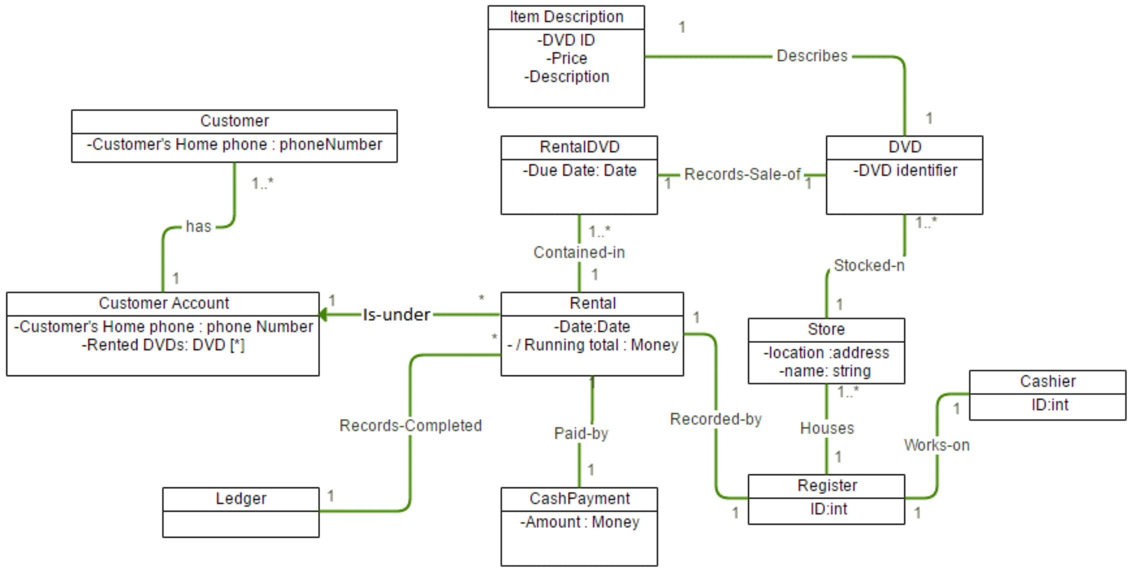
1)Cashier cancels rental on system

4) SDD between Cashier and SuD



System Operations are import in OO design because they are help identify what will later become the methods of the classes in our software. For example the system event enterDVD triggers the system operation enterDVD. In implementation of this system, enterDVD would be easily conceptualized as a method in Java or in other OO languages.

5)

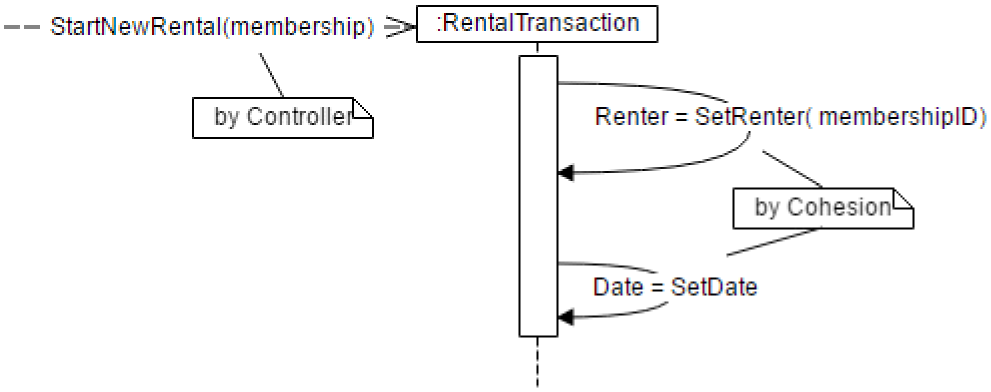


Part 2)
1)
Operation: StartNewRental(membershipID)

Cross References: Use Cases: Rent Bikes

Preconditions: There is no rentalTransaction underway, and a customer is waiting to rent something

Post Conditions: -A RentalTransaction instance rental was created
-rental was associated with a Membership, based on membershipID match
-rental.date became the current date



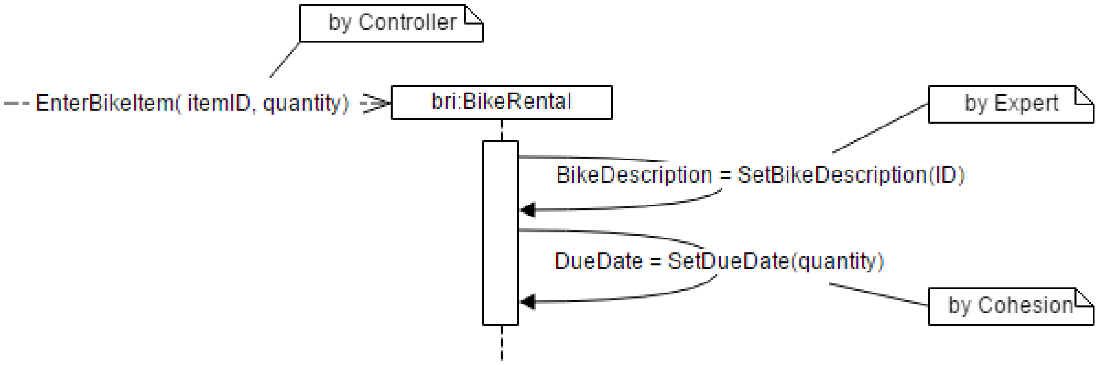
2)
Operation: enterBikeItem(itemID, quantity)

Cross References: Use cases: Rent Bikes

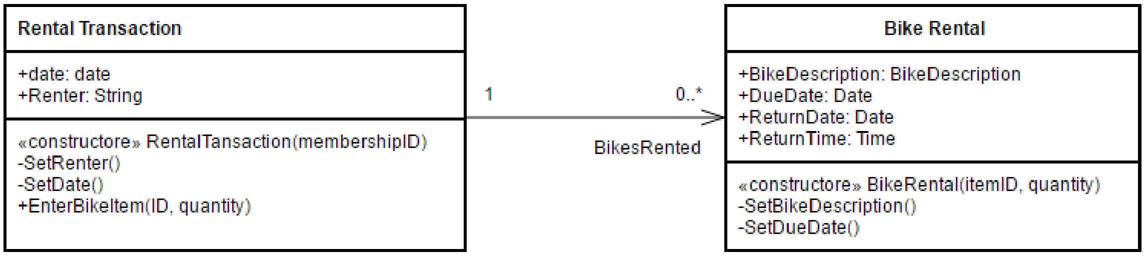
Preconditions: There is a rental underway

Post Conditions: -A BikeRental instance bri was created
-bri was associated with the current RentalTransaction
-bri was associated with a BikeDescription, based on itemID match
-bri.dueDate became the current RentalTransaction.date plus the number of days specified in the quantity argument

(*Assumption: since there is no quantity attribute in your domain model but there is however a quantity argument in you "enterBikeItem" operation, I am assuming that the quantity argument is meant to be the number of days the customer wants to rents the bikes*
Assumption: I am assuming that returnDate and returnTime are set to when the customer actually returns the Bike, and therefore will not be modified until then)


by Controller
EnterBikeItem(itemID, quantity)
bri:BikeRental
by Expert
BikeDescription = SetBikeDescription(ID)
DueDate = SetDueDate(quantity)
by Cohesion

3)


Rental Transaction
+date: date
+Renter: String
«constructore» RentalTansaction(membershipID)
-SetRenter()
-SetDate()
+EnterBikeItem(ID, quantity)
1
0..*
BikesRented
Bike Rental
+BikeDescription: BikeDescription
+DueDate: Date
+ReturnDate: Date
+ReturnTime: Time
«constructore» BikeRental(itemID, quantity)
-SetBikeDescription()
-SetDueDate()