面向对象程序的分析与设计
Object-Oriented Analysis and Design

Lecture 10
Use Case Realization II

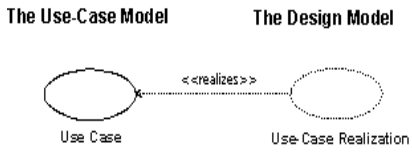
Prof. S. Xu

Contents

- Use-case realization (continued)

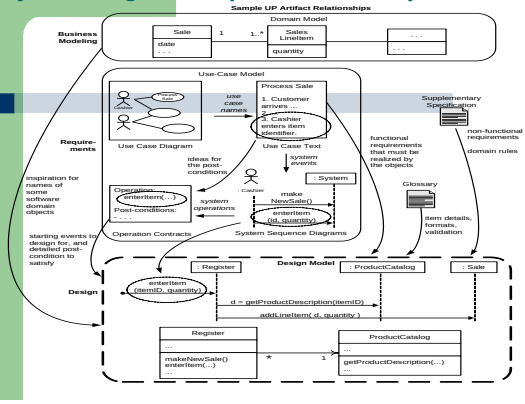
Object Design Examples With Grasp

- ***use case realizations***.
  - What it is?

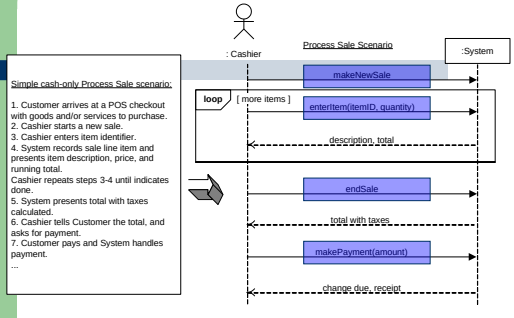


3

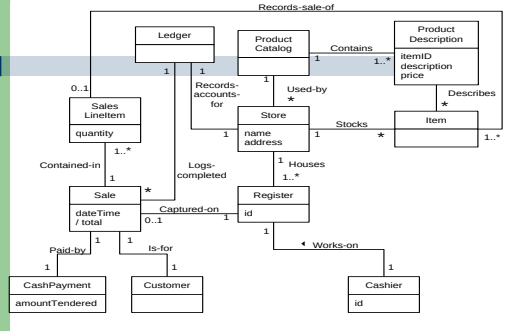
Object Design Examples With Grasp



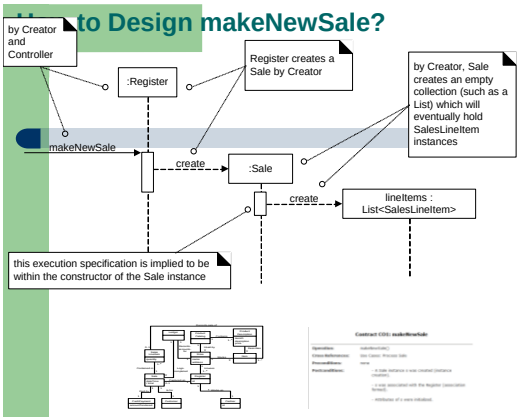
Use Case Realization for NextGen POS



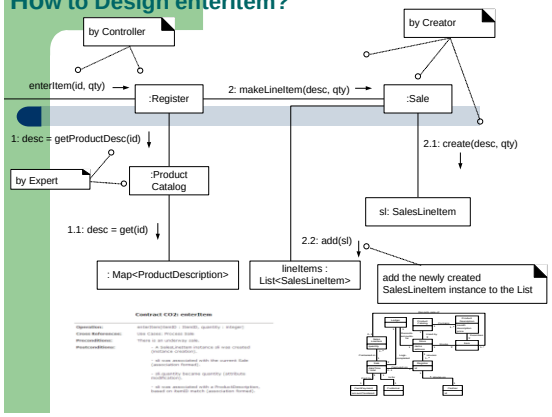
Use Case Realization for NextGen POS



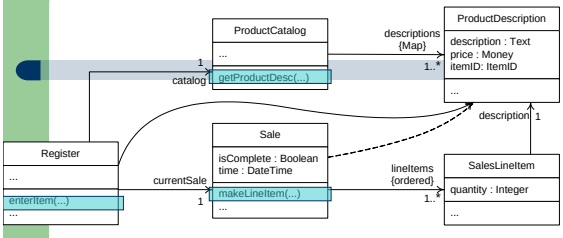
How to Design makeNewSale?



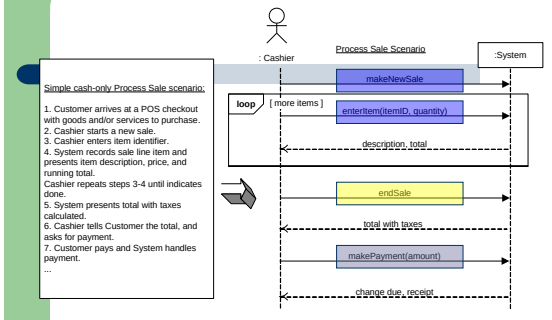
How to Design enterItem?



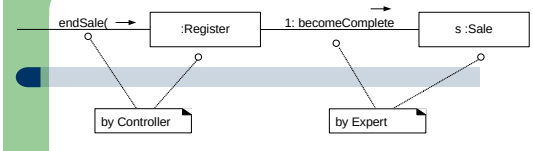
How to Design enterItem?



Use Case Realization for NextGen POS



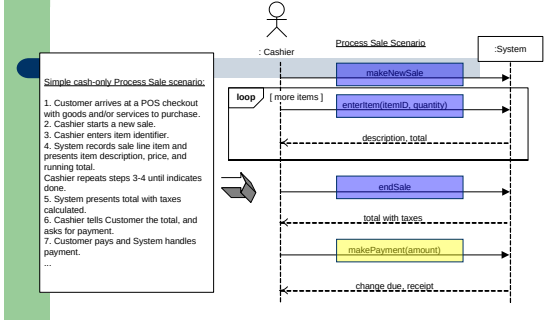
How to Design endSale



Contract CO3: endSale

Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	- Sale.isComplete became true (attribute modification).

Use Case Realization for NextGen POS

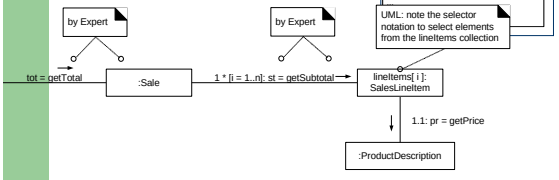


How to Design getTotal()?

Information Required for Sale Total	Information Expert
ProductDescription.price	ProductDescription
SalesLineItem.quantity	SalesLineItem
all the SalesLineItems in the current Sale	Sale

Simple cash-only Process Sale scenario:

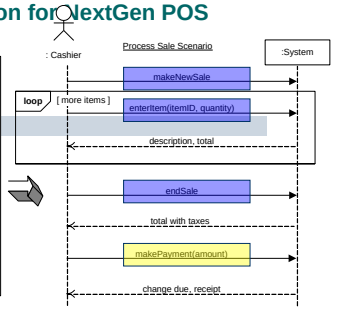
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

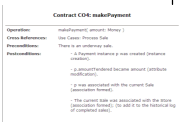


Use Case Realization for NextGen POS

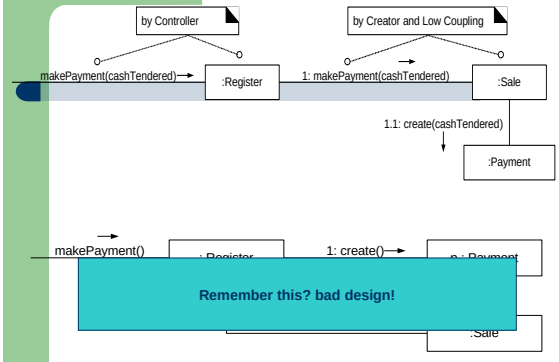
Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...





How to Design makePayment?



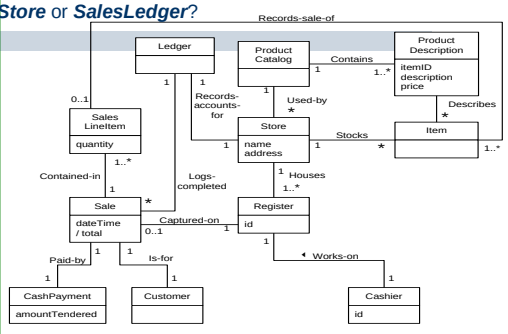
Logging a Sale - How to Design makePayment?

Contract CO4: makePayment

Operation:	makePayment(amount: Money)
Cross References:	Use Cases: Process Sale
Preconditions:	There is an underway sale.
Postconditions:	- A Payment instance p was created (instance creation). - p.amountTendered became amount (attribute modification). - p was associated with the current Sale (association formed). - The current Sale was associated with the Store (association formed); (to add it to the historical log of completed sales).

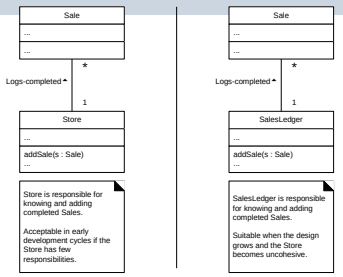
Use Case Realization for NextGen POS

- **Who is responsible for knowing all the logged sales and doing the logging?**
- ***Store*** or ***SalesLedger***?



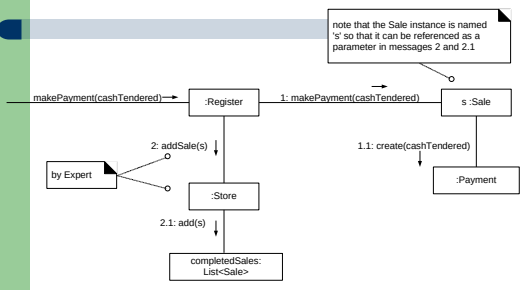
Logging a Sale - How to Design makePayment?

- **Who is responsible for knowing all the logged sales and doing the logging?**
- ***Store*** or ***SalesLedger***?



Logging a Sale - How to Design makePayment?

- **Stick with *Store*** for now, may change to ***SalesLedger*** later during designing?



Calculating balance

- **The Process Sale use case implies that the balance due from a payment be printed on a receipt and displayed somehow.**

we must ensure that it is known.

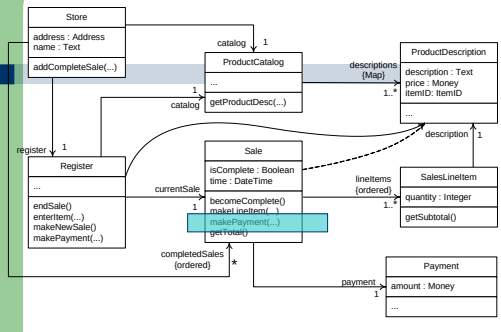
- **No class currently knows the balance, how to know the balance?**

Calculating balance – the Reasoning

To calculate the balance, we need the sale total and payment cash tendered. Therefore, *Sale* and *Payment* are partial Experts on solving this problem.

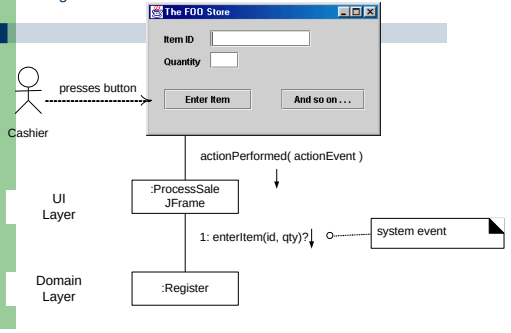
Should Payment or Sale class calculate the balance?

The Final NextGen DCD for Iteration -1



How to Connect the UI Layer to the Domain Layer?

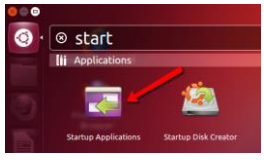
- Once the UI object has a connection to the ***Register*** instance , it can forward system event messages, such as the ***enterItem*** and ***endSale*** message



Initialization and the 'Start Up' Use Case

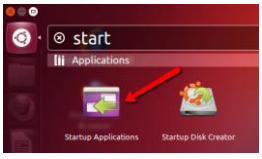
- Most systems have either an implicit or explicit ***Start Up*** use case and some initial system operation related to the *starting up of the application*.

- Although abstractly, a startUp system operation is the earliest one to execute, *delay the development of an interaction diagram for it until after all other system operations have been considered.*



How do Applications Start Up?

- a common design: create an ***initial domain object*** or ***a set of peer initial domain objects*** that are the first software "domain" objects created.
  - This creation may happen explicitly in the starting main method.



How do Applications Start Up?

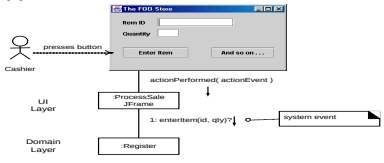
- Often, the initial domain object, once created, is responsible for the creation of its direct child domain objects.
  - For example, a Store chosen as the initial domain object may be responsible for the creation of a Register object.

```
public class Main
{

public static void main( String[] args )
{
   // Store is the initial domain object.
   // The Store creates some other domain objects.

   Store store = new Store();

   Register register = store.getRegister();

   ProcessSaleJFrame frame = new ProcessSaleJFrame( register );
   ...
}

}
```

Choosing the Initial Domain Object

Guideline

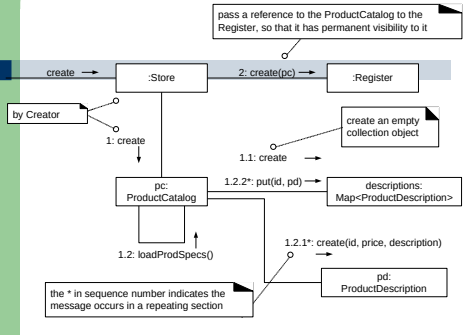
- Choose as an initial domain object a class at or near the root of the containment or aggregation hierarchy of domain objects. This may be a facade controller, such as Register, or some other object considered to contain all or most other objects, such as a Store.
- In this application, we chose the **Store** as the initial object.



Choosing the Initial Domain Object

- Therefore, we identify the following initialization work:
  - Create a Store, Register, ProductCatalog, and ProductDescriptions.
  - Associate the ProductCatalog with ProductDescriptions
  - Associate Store with ProductCatalog.
  - Associate Store with Register.
  - Associate Register with ProductCatalog.

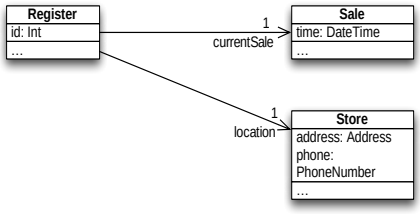
Choosing the Initial Domain Object



- More Mobile Computing

Review

- How many attributes does Register have according to this DCD? ____



Review

- In the model-view separation principle, "model" is a synonym for the _____ layer and "view" is a synonym for the _____ layer.

Review

- In a typical layered architecture, where do system operations from SSDs show up as messages?

Exercise 1

Our problem domain is an information system for a video rental store. Simplifying assumptions and details:

- It is a stand-alone store, not part of a larger organization.
- Rents only videos, not computer games or other items.
- A "video" can be in any medium: tape, DVD, and so on.
- The rental charge may vary by medium. For example, DVD rentals are more expensive than tapes.
- The store does not sell anything. For example, there are no sales of videos or food.
- All transactions are rentals.
- The input medium by which membership and video rentals are captured is not important.
- Cash-only payments
- On completion of a rental, the customer receives a transaction report with 'typical' information on—use your judgement.
- Each renter has a separate membership.

Exercise 1 - Use Case: Rent Videos

Actor Actions	System Response
1. This use case begins when a Customer arrives at a checkout with videos to rent.	
2. The Customer presents their membership identification to the Clerk, who enters it into the system.	3. Presents membership information, and status of loans (usually nothing on loan, and no outstanding fines).
4. For each video, the Clerk records the item identification into the system.	5. Presents accumulating list of rental video titles, due dates, and total rental fee.
6. Clerk informs Customer of total rental fee, and asks for payment.	
7. Customer pays cash to Clerk.	
8. Clerk records payment into system.	9. Generates receipt and loan report.
10 Clerk gives receipt and loan report to Customer, who then leaves with the rental items.	

Exercise 1

Write confirmMemberShip Operation contract

Exercise 1

Write confirmMemberShip Operation contract

- **Name:** confirmMembership (membershipID)
- **Responsibilities:** Confirm that the membership is valid—that it exists, and that the membership is not suspended. Present confirmation. Present any overdue rentals and charges owed.
- **Cross References:** Use Case: Renting Videos
- **Preconditions:** System does not have a rental transaction underway.
- **Postconditions:**
  - If the membership was valid:
  - A *RentalTransaction txn* was created and initialized.
  - The *Membership* (with *membershipID*) was associated with *txn*.

Exercise 1

Using the *confirmMembership* operation contract as a starting hint, complete the UML communication (collaboration) diagram. Annotate every message with the GRASP (Expert, Creator, and so on) and/or other pattern that justifies it

Exercise 1

Draw a partial design class diagram, only for the *VideoStore* and *Membership* classes. Show all simple attributes, design-phase associations (with navigability) between these two classes only, and method signatures.

Exercise 2

The following sentences describe information used in a car loan system. Cars may be owned by persons, companies, or banks. A car loan from a bank may be involved in the purchase of a car. An owner can own several cars. A car can have several loans against it. Banks lend money to persons, companies, or other banks.

Draw a UML design class diagram for the car loan system .

Exercise 3

Geographic Information System (GIS) contains information about countries of the world. A country may border any number of other countries or oceans. Each country has only one capitol city but many regular cities. A country has a unique name. Cities have names and populations. The population of a country is derived by summing the populations of all its cities. Each country is only on one continent.

Draw a UML design class diagram for this system

- Internet of Things