Object oriented Analysis &Design
面向对象分析与设计

Lecture_11 Exploration to Design
设计初探
Ch18~Ch21

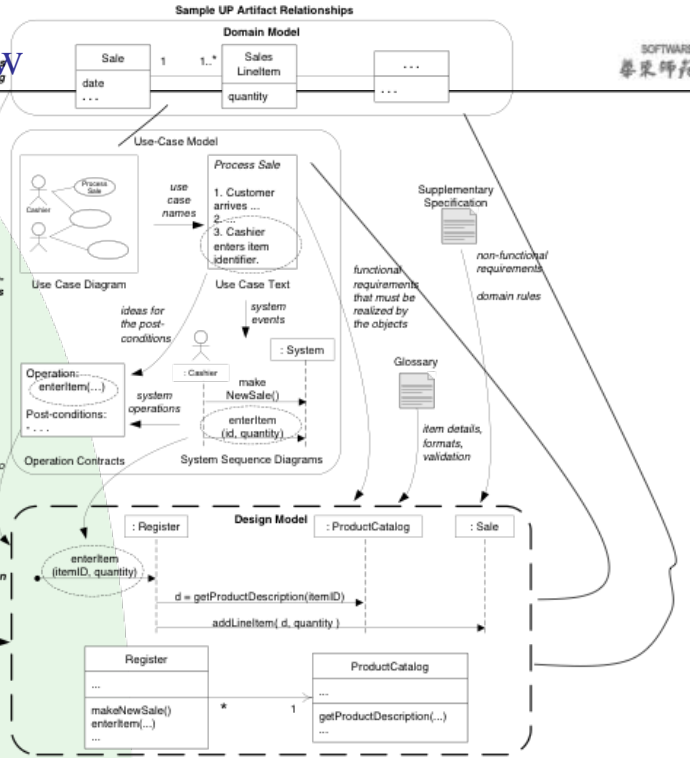
主讲: 陈小红

日期:

Review

启发了某些软件领域层对象的名称

用于设计的启动事件，以及需要被满足的详细后置条件



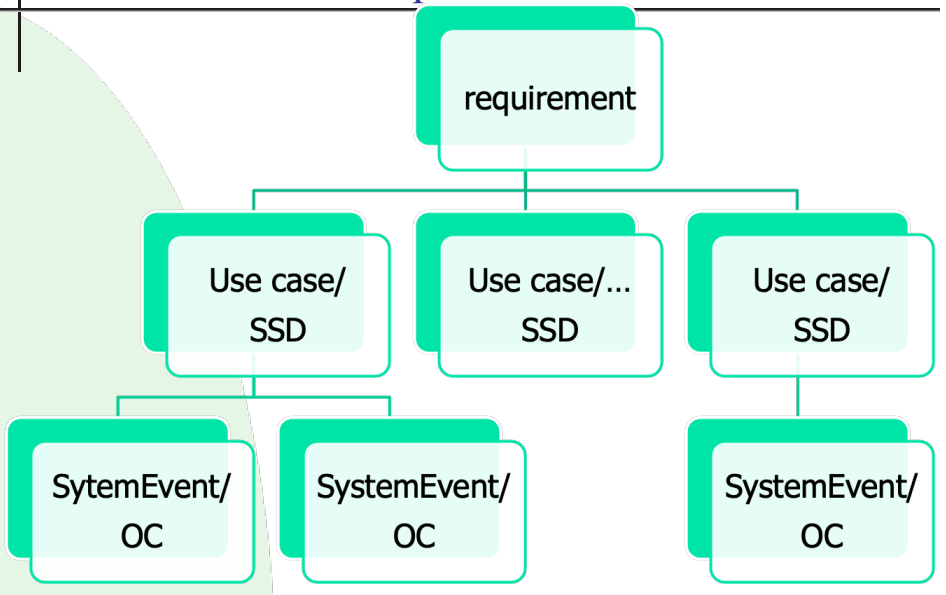
掌握要点

- 设计的切入点
- POS超市收银台系统的设计
- Monopoly游戏的设计

Chapter 18. Object Design Examples with GRASP

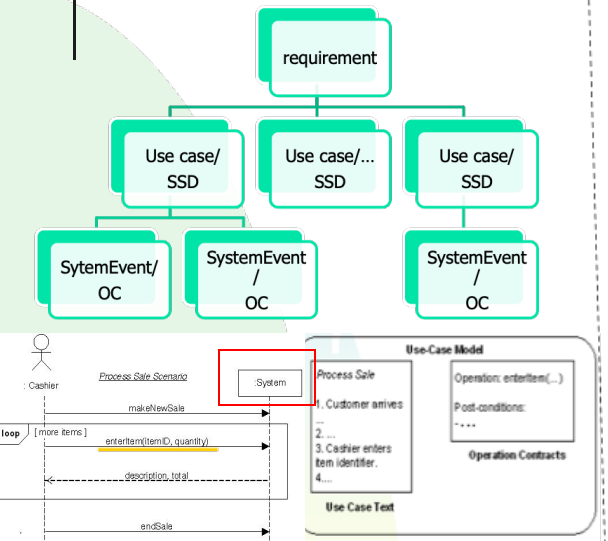
- **Objectives**
  - Design use case realizations.
  - Apply GRASP to assign responsibilities to classes.
  - Apply UML to illustrate and think through the design of objects （应用UML阐述和思考对象的设计）

Review: OOA Requirement?



Solution: OOD?

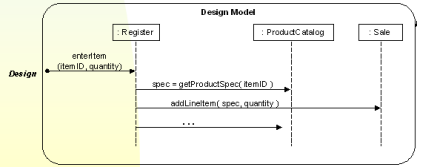
UI



system

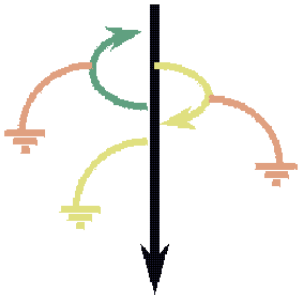
'Start Up' Use Case

Use Case Realization



What is a Use Case Realization?

- RUP define:
  - A use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects
- a designer can describe the design of one or more scenarios of a use case; each of these is called a use case realization (also called a scenario realization)



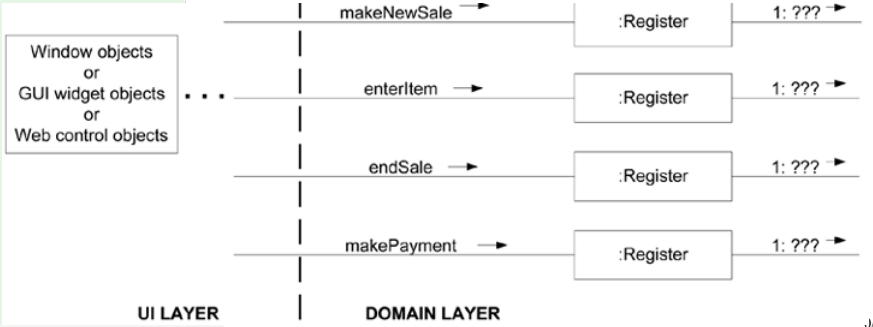
Use Case Realization

- How a particular use case is realized within the Design Model in terms of collaborating objects
  - Use case suggests the system operations
  - System operations becoming starting messages entering the controllers for domain layer interaction diagrams
  - Domain layer interaction diagrams illustrate the use case realization
  - Domain model “inspires” classes but not one-to-one
  - Use post-conditions from contract to design detailed message interactions
- Customer needs to be involved in the process throughout

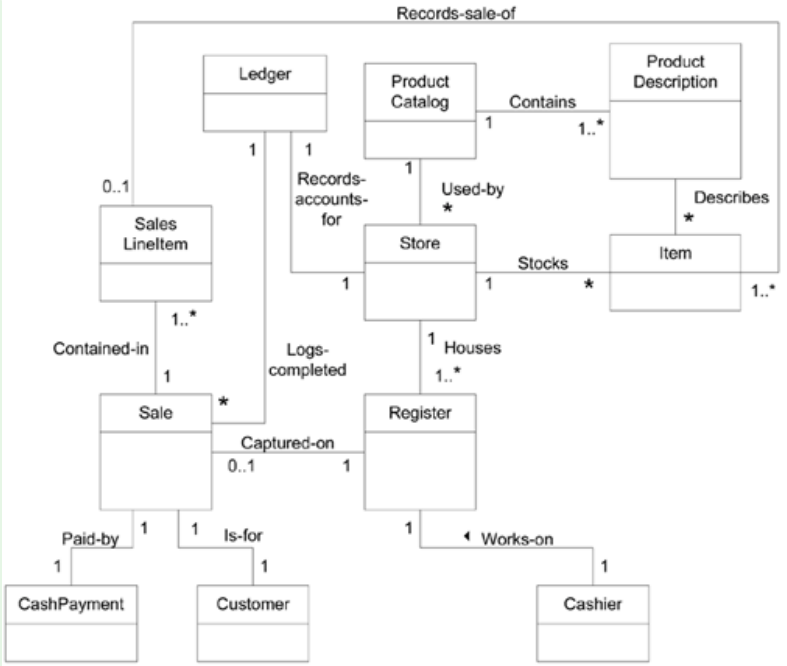
Usecase-realization Example

- Process Sale use case □ system operation
  - makeNewSale
  - enterItem
  - endSale
  - makePayment
- How to handle system operation?

MakeNewSale is the system operation from SSD. Each main interaction diagram begins from controller objects where system operations entering domain layer. makeNewSale等是来自SSD的系统操作 每个主要交互图都从进入领域层控制器对象（如Register)的系统操作开始

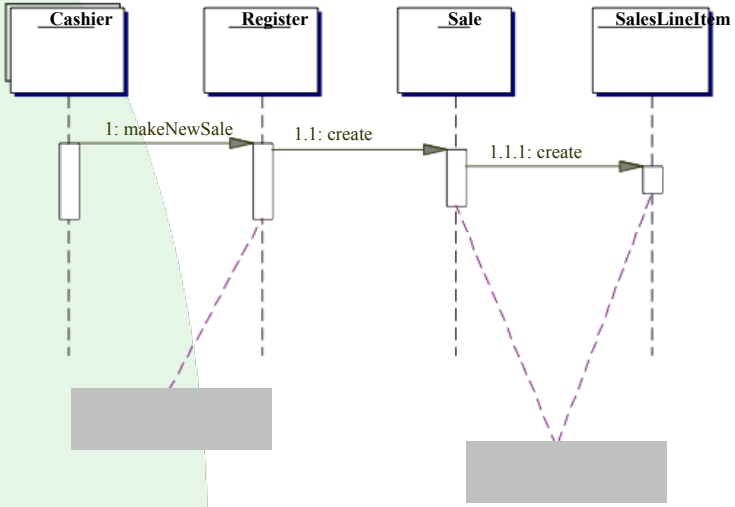


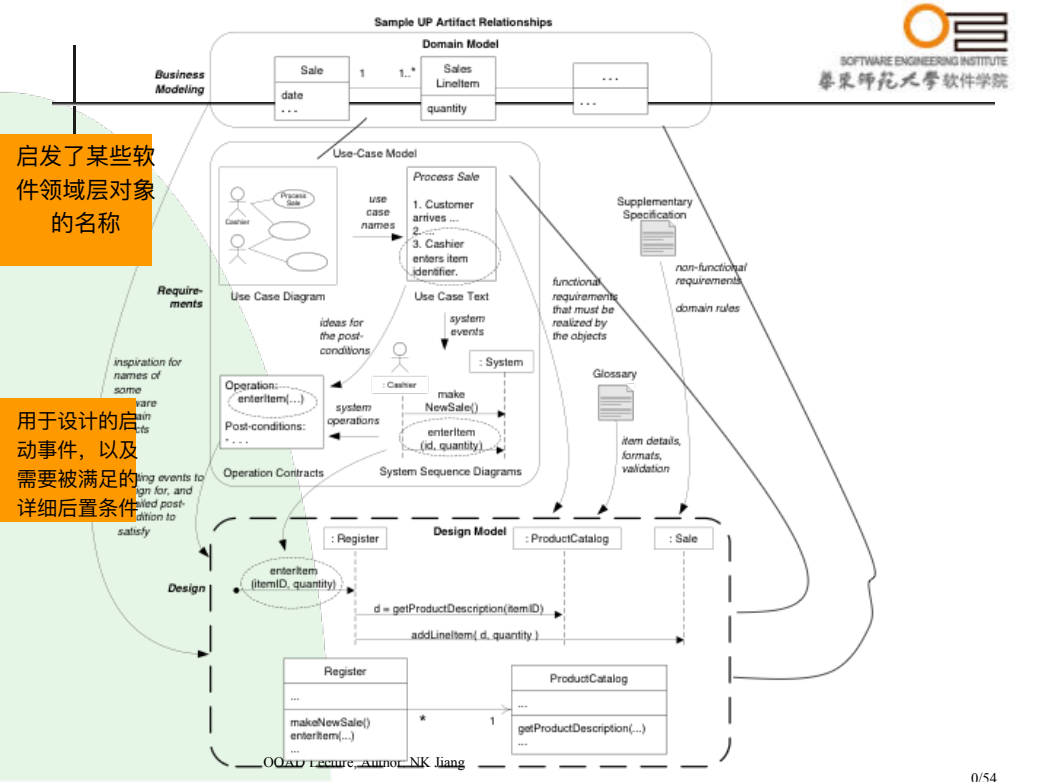
NextGen POS partial domain model



Use case realization (presented by SD)

- makeNewSale





启发了某些软件领域层对象的名称

用于设计的启动事件，以及需要被满足的详细后置条件

- Use case realization for operation contract
  - Ch18.4
    - Please be attentive to read and understand. 请同学们仔细地去看、体会

Use Case Realizations for the NextGen Iteration

- 1. Guideline
  - Initialization and the 'Start Up' Use Case
  - When coding, program at least some Start Up initialization first.
  - But during OO design modeling, consider the Start Up initialization design last, after you have discovered what really needs to be created and initialized.
  - Then, design the initialization to support the needs of other use case realizations

Use Case Realizations for the NextGen Iteration

- 2. How to Design makeNewSale?
- 2.1 Contract CO1: makeNewSale

Contract CO1: makeNewSale	
Operation:	makeNewSale()
Cross References:	Use Cases: Process Sale
Preconditions:	none
Postconditions:	- A Sale instance s was created (instance creation). - s was associated with the Register (association formed). - Attributes of s were initialized.

Use Case Realizations for the NextGen Iteration

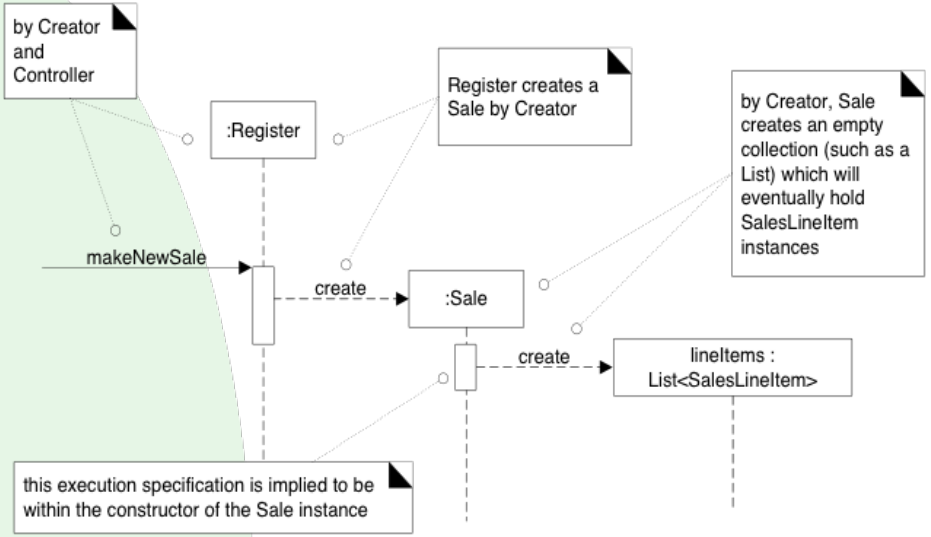
2.2 Choosing the Controller Class

- Candidates
  - Façade controller
    - Store a kind of root object because we think of most of the other domain objects as "within" the Store.
    - Register a specialized device that the software runs on; also called a POSTerminal.
    - POSSystem a name suggesting the overall system
  - Usecase controller
    - ProcessSaleHandler constructed from the pattern <use-case-name> "Handler" or "Session"
    - ProcessSaleSession
- Principle
  - a few system operations and if the facade controller is not taking on too many responsibilities
- Result
  - Register
  - Note: this Register is a software object in the Design Model. It isn't a physical register

Use Case Realizations for the NextGen Iteration

- 2.3 Creating software ‘Sale’ object
  - Creator + Doamin Model + LRG
    - Register
- 2.4 Create an empty collection to record all the future ‘SalesLineItem’ instances

POS: makeNewSale Interactions



Use Case Realizations for the NextGen Iteration

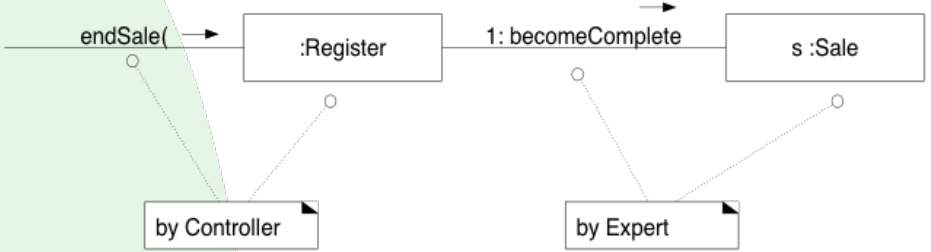
- 3. How to Design endSale?
- 3.1 Contract

Contract CO3: endSale

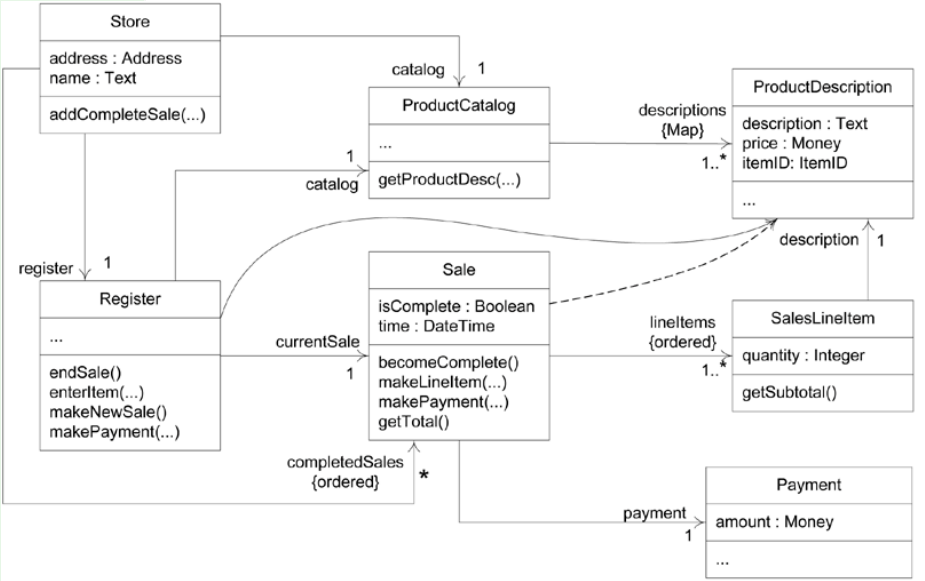
Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is an underway sale.
Postconditions:	Sale.isComplete became true (attribute modification).

- 3.2 Choosing the Controller Class
  - Register
- 3.3 Setting the Sale.isComplete Attribute
  - Who do it?
    - 'Sale' itself after receiving message from 'Register'

POS: endSale Interaction



- Figure 18.17. A more complete DCD reflecting most design decisions



- 4. How to Connect the UI Layer to the Domain Layer?
  - 4.1 UI layer obtain visibility to objects in the domain layer
    - 1) initializer object

E.g.

```
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 );
  ...
} }
```

    - 2) from a well-known source

Use Case Realizations for the NextGen Iteration

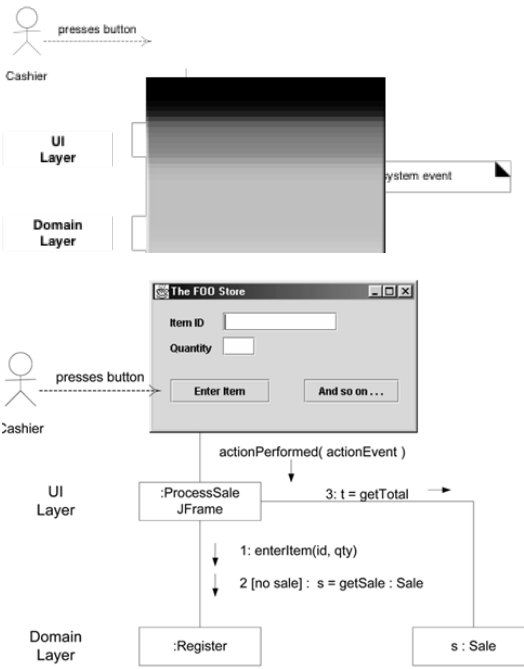
4 2. we want the window to show the running total after each entry

S1: UI -> Register -> Sale

expand the interface of the Register object, making it less cohesive

S2: UI -> get reference of Sale -> send msg to Sale directly

higher coupling in and of itself is not a problem; rather, coupling to unstable things is a real problem



5. Initialization and the 'Start Up' Use Case

5.1 When to Create the Initialization Design?

Guideline

Do the initialization design last.

5.2

initial domain object： Main()

Main will create other objects

Objects will create its sub-objects

Guideline

Choose as an initial domain object a class at or near the root of the containment or aggregation hierarchy of domain objects(选择位于或接近于领域对象包含或聚合层次的根类，作为初始领域对象).

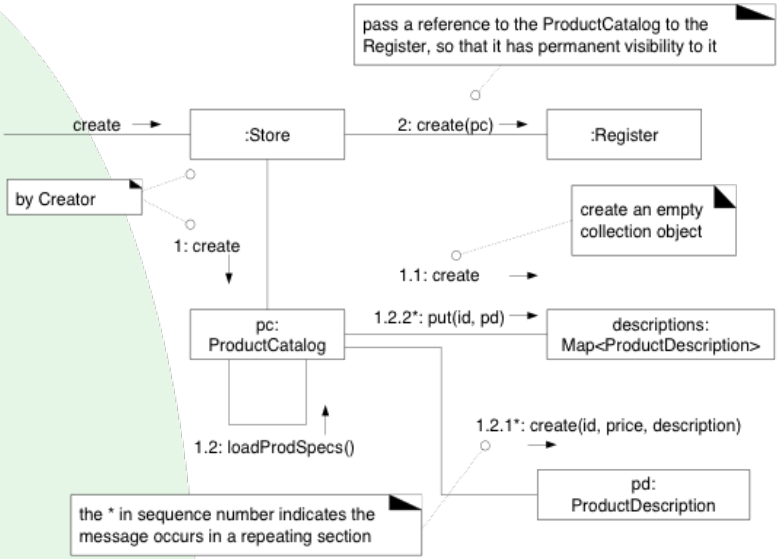
This may be a facade controller, such as Register,

or some other object considered to contain all or most other objects, such as a Store.

Use Case Realizations for the NextGen Iteration

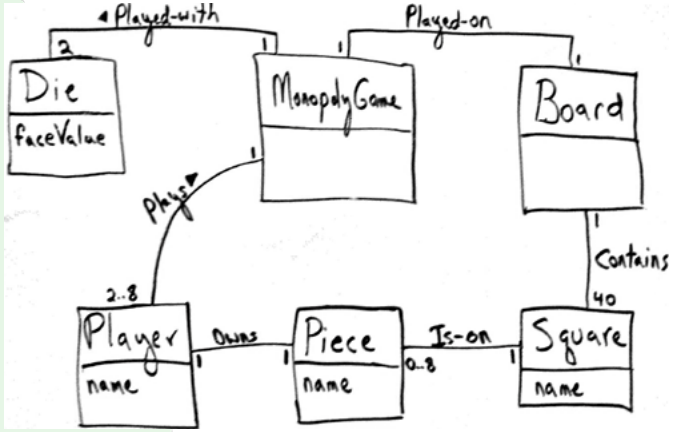
- 5.3. Store.create()
- identify the following initialization work
  - Create a Store, Register, ProductCatalog, and ProductDescriptions.
  - Associate the ProductCatalog with ProductDescriptions.
  - Associate Store with ProductCatalog.
  - Associate Store with Register.
  - Associate Register with ProductCatalog

POS: StartUp


pass a reference to the ProductCatalog to the Register, so that it has permanent visibility to it
create
:Store
2: create(pc)
:Register
by Creator
1: create
create an empty collection object
1.1: create
pc: ProductCatalog
1.2.2*: put(id, pd)
descriptions: Map<ProductDescription>
1.2: loadProdSpecs()
1.2.1*: create(id, price, description)
pd: ProductDescription
the * in sequence number indicates the message occurs in a repeating section

Use Case Realizations for the Monopoly Iteration

- Figure 18.21. Iteration-1 Domain Model for Monopoly



- System Operation
  - playGame

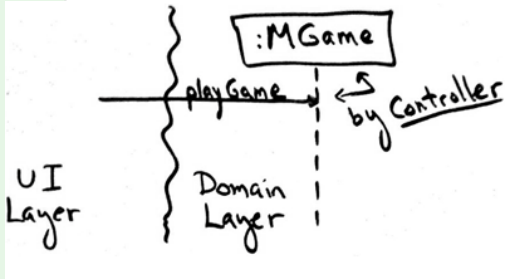
Use Case Realizations for the Monopoly Iteration

- 6.1 Choosing the Controller Class

Represents the overall "system," "root object," a specialized device, or a major subsystem.	MonopolyGame a kind of root object: We think of most of the other domain objects as "contained within" the MonopolyGame. Abbreviated MGame in most of the UML sketches. MonopolyGameSystem a name suggesting the overall system
Represents a receiver or handler of all system events of a use case scenario.	PlayMonopolyGameHandler constructed from the pattern <use-case-name> "Handler" PlayMonopolyGameSession

- We select MonopolyGame (abv. MGame)

Figure 18.22. Applying Controller to the playGame system operation

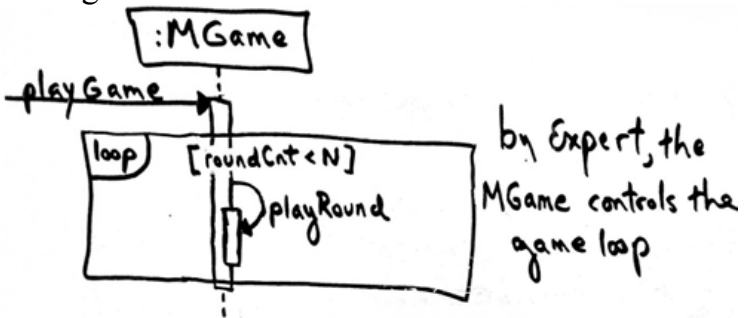


- 6.2 Who is Responsible for Controlling the Game Loop?
  - Turn: a player rolling the dice and moving the piece
  - Round: all the players taking one turn
  - What information is needed for the responsibility?

Use Case Realizations for the Monopoly Iteration

Information Needed	Who Has the Information?
the current round count(当前回合数)	No object has it yet, but by LRG, assigning this to the MonopolyGame object is justifiable.
all the players (so that each can be used in taking a turn)	Taking inspiration from the domain model, MonopolyGame is a good candidate.

MonopolyGame is a good candidate



Use Case Realizations for the Monopoly Iteration

- 6.3 Who Takes a Turn?
  - What information is needed for the responsibility?

Information Needed	Who Has the Information?
current location of the player (to know the starting point of a move)	Taking inspiration from the domain model, a Piece knows its Square and a Player knows its Piece. Therefore, a Player software object could know its location by LRG.
the two Die objects (to roll them and calculate their total)	Taking inspiration from the domain model, MonopolyGame is a candidate since we think of the dice as being part of the game.
all the squares the square organization (to be able to move to the correct new square)	By LRG, Board is a good

three partial information experts for the "take a turn" responsibility: Player, MonopolyGame, and Board.

Solution

- Guideline1: When there are multiple partial information experts to choose from, place the responsibility in the dominant information expert the object with the majority of the information.
  - This tends to best support Low Coupling.
- Guideline2: When there are alternative design choices, consider the coupling and cohesion impact of each, and choose the best.
  - This can be applied.
  - MonopolyGame is already doing some work, so giving it more work impacts its cohesion, especially when contrasted with a Player and Board object, which are not doing anything yet.
  - But we still have a two-way tie with these objects.
- Guideline3: When there is no clear winner from the alternatives other guidelines, consider probable future evolution of the software objects and the impact in terms of Information Expert, cohesion, and coupling

- So , Who Takes a Turn?
  - Player is most suitable

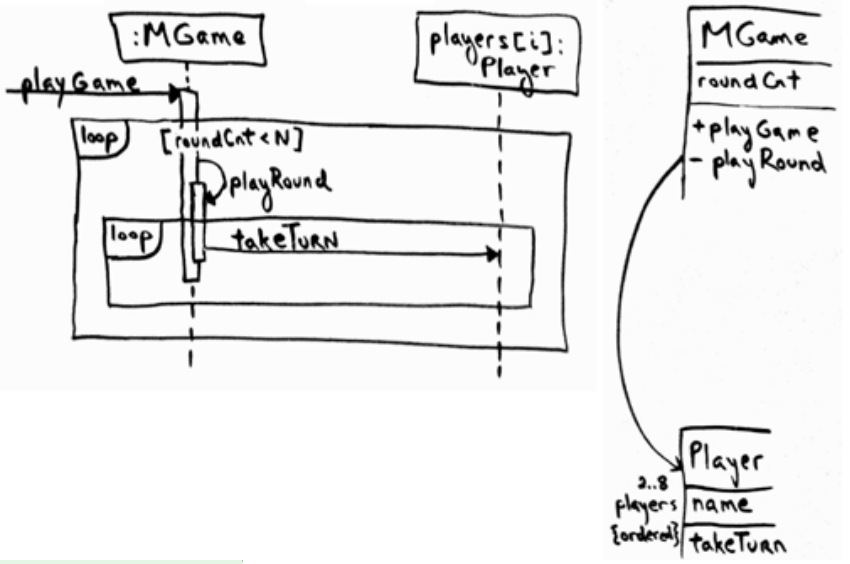
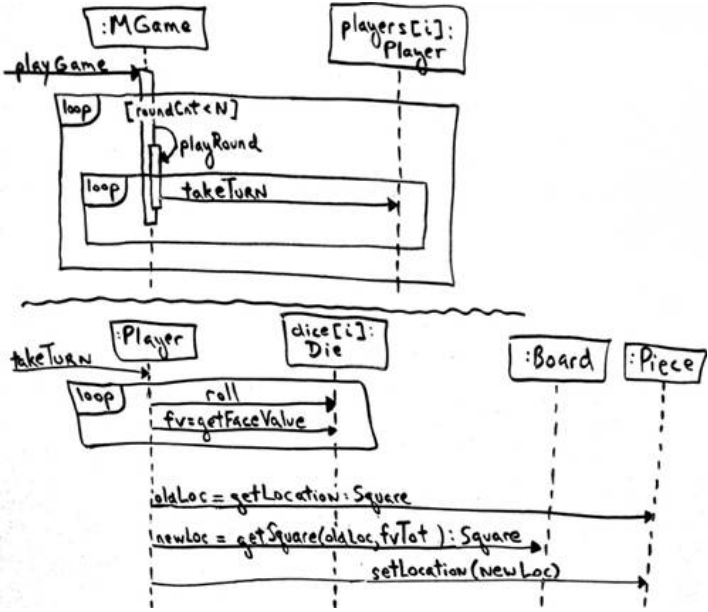


Figure 18.24. Player takes a turn by Expert

- 6.4 Taking a Turn
  - calculating a random number total between 2 and 12 (the range of two dice)
  - calculating the new square location
  - moving the player's piece from an old location to a new square location
- 6.5 Who Coordinates All This?
  - Player
- 6.6 The Problem of Visibility
  - Player will need visibility to the Die, Board, and Piece objects each and every turn
- 6.7 The Final Design of playGame

- Figure 18.25. Dynamic design for playGame



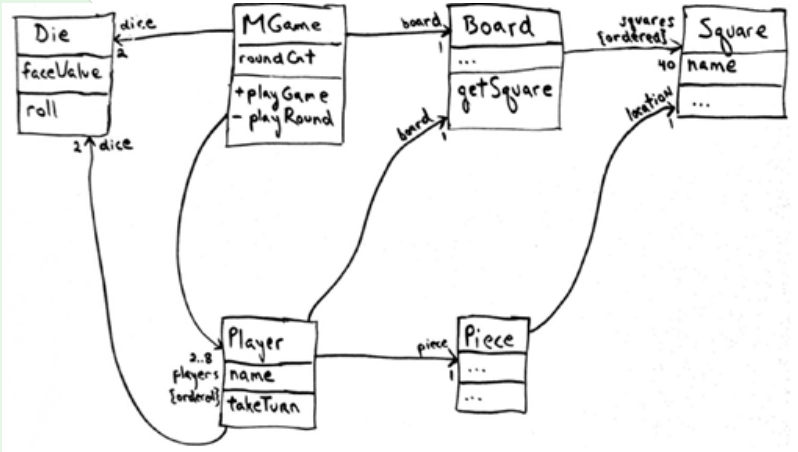
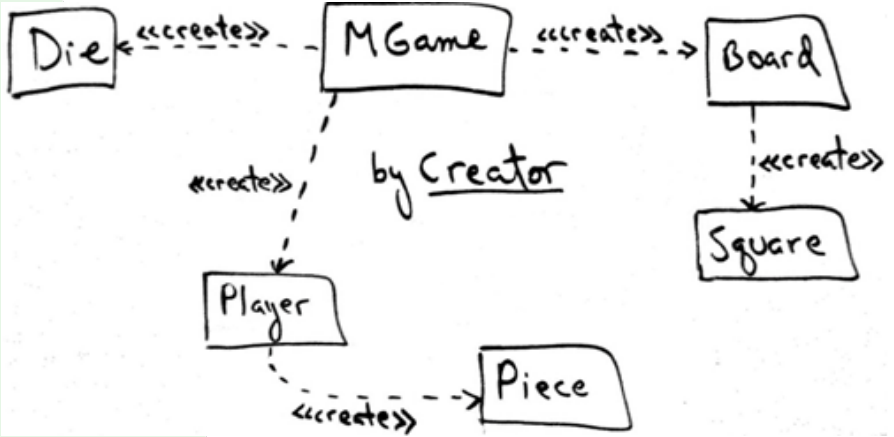


Figure 18.26. Static design for playGame

- 6.8 Initialization and the 'Start Up' Use Case



end

- 后续是选修内容

- 作业：（手写）
  - 整理MonopolyGame游戏，从需求到设计的全过程。
  - （高级选修作业）分析设计的任务从哪里来？是否需要在需求模型中增加新的模型？需要什么样的模型，才能提供这些信息呢？
  - 作业提交时间： 12月1日

The Command-Query Separation Principle : CQS原则

- It is important.

```
// style #1; used in the official solution public
void roll() {

 faceValue = // random num eneration

}

public int getFaceValue() {

 return faceValue;

}
```

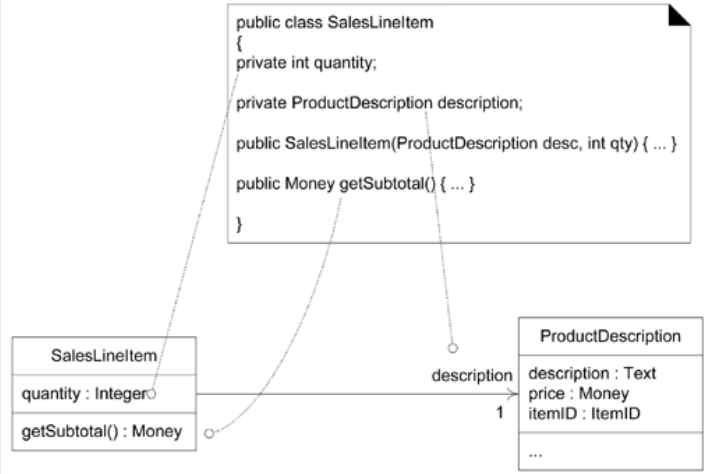
```
// style #2; why is this poor?
public int roll()
{
  faceValue = // random num generation
  return faceValue;
}
```

Chapter 20. Mapping Designs to Code

- The creation of code in an OO language,such as Java or C#is not part of OOA/D, it's an end goal
  - Creativity and Change During Implementation
- Implementation in an object-oriented language requires writing source code for:
  - class and interface definitions
  - method definitions
- Coding by artifacts of Design Model
  - DCD
  - SD
  - Collection Classes in Code
  - Exceptions and Error Handling

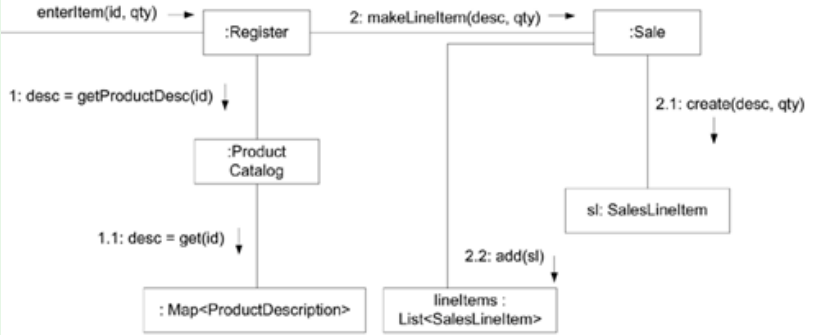
DCD to code

Figure 20.1. SalesLineItem in Java.



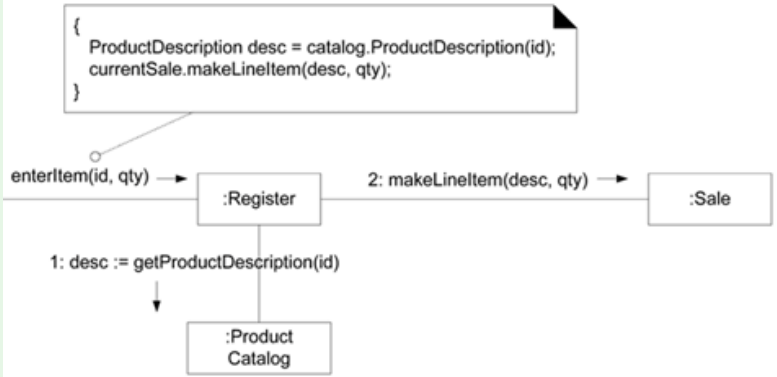
from Interaction Diagrams

Figure 20.2. The enterItem interaction diagram.



- The enterItem message is sent to a Register instance; therefore, the enterItem method is defined in class Register.
  - public void enterItem(ItemID itemID, int qty)
- Message 1: A getProductDescription message is sent to the ProductCatalog to retrieve a ProductDescription.
  - ProductDescription desc = catalog.getProductDescription(itemID);
- Message 2: The makeLineItem message is sent to the Sale.
  - currentSale.makeLineItem(desc, qty);

Figure 20.4. The enterItem method.



- 请同学们自学ch20.11, 代码，体会 面向对象的编码：
- Please read ch20.11 and the code by yourself, and understand object-oriented programming.
  - 类之间的关系如何在代码中体现 how to reflect the association between classes in code.
  - 和书本前面章节的各种设计图 对照，体会设计和编码之间的联系
  - Compared with the various design diagram in previous chapter, understand the connection between designing and programming.

Ch21 Refactoring (重构)

- Refactoring
  - **Structured method to rewrite code maintaining external behavior while applying small internal changes (transformations)**
  - **Continuously tested with unit tests and regression tests**
- **Mostly for beautification**

Conclusion

- The end