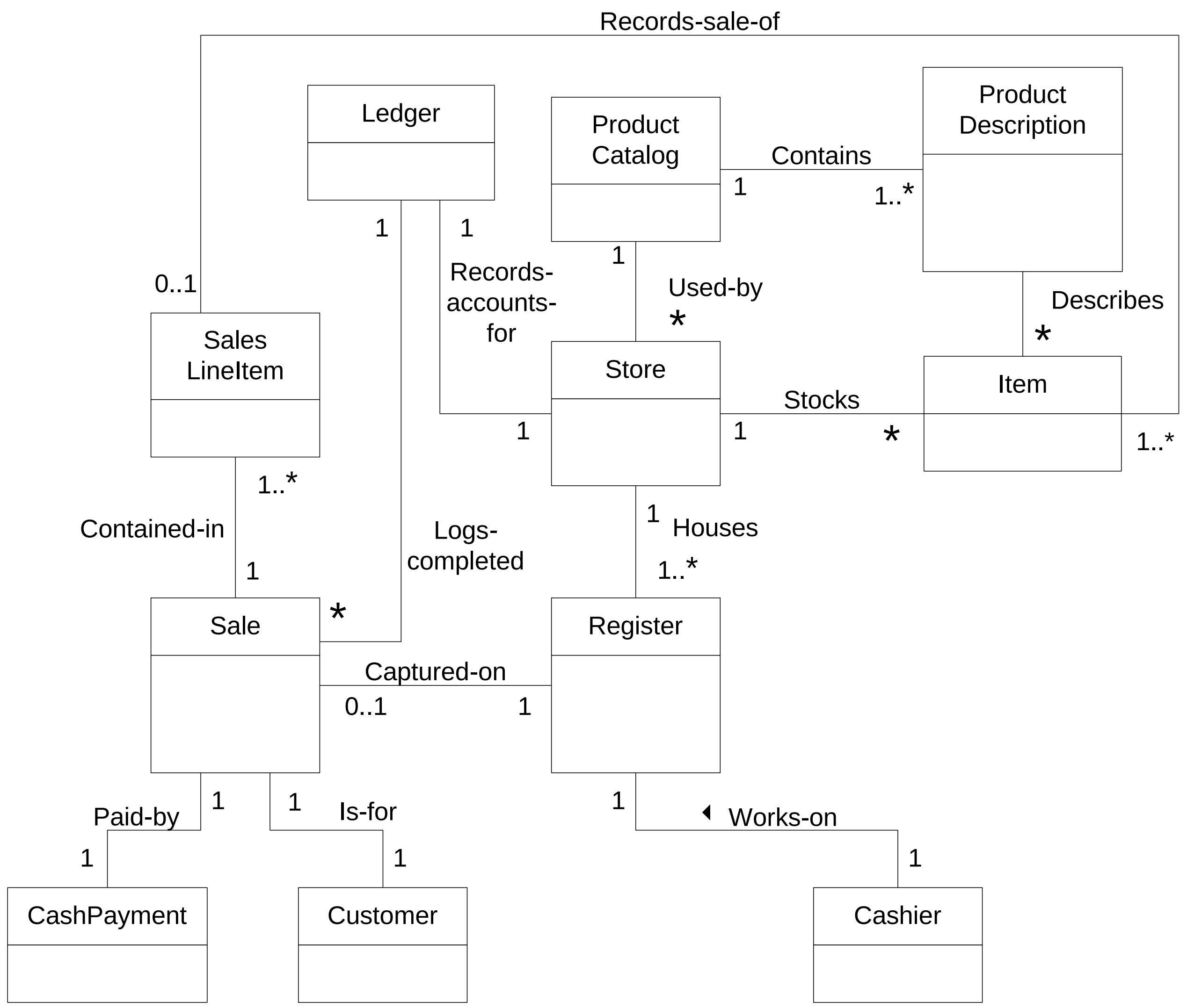
Interaction and Class Design

COMP 3700.002
Software Modeling and Design

Shehenaz Shaik

PoS: Domain Model



Use case Realization

- Describes how a particular use case is realized within the Design Model, in terms of collaborating objects
  - Use case
  - → Scenarios
  - → SSDs
  - → System operations
  - → Starting points for Domain Layer Interaction Diagrams
  - → Illustrate how objects interact to fulfil tasks

Use case Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
   - Draw Interaction Diagram (Dynamic view)
   - Expand Design Class Diagram (Static view)

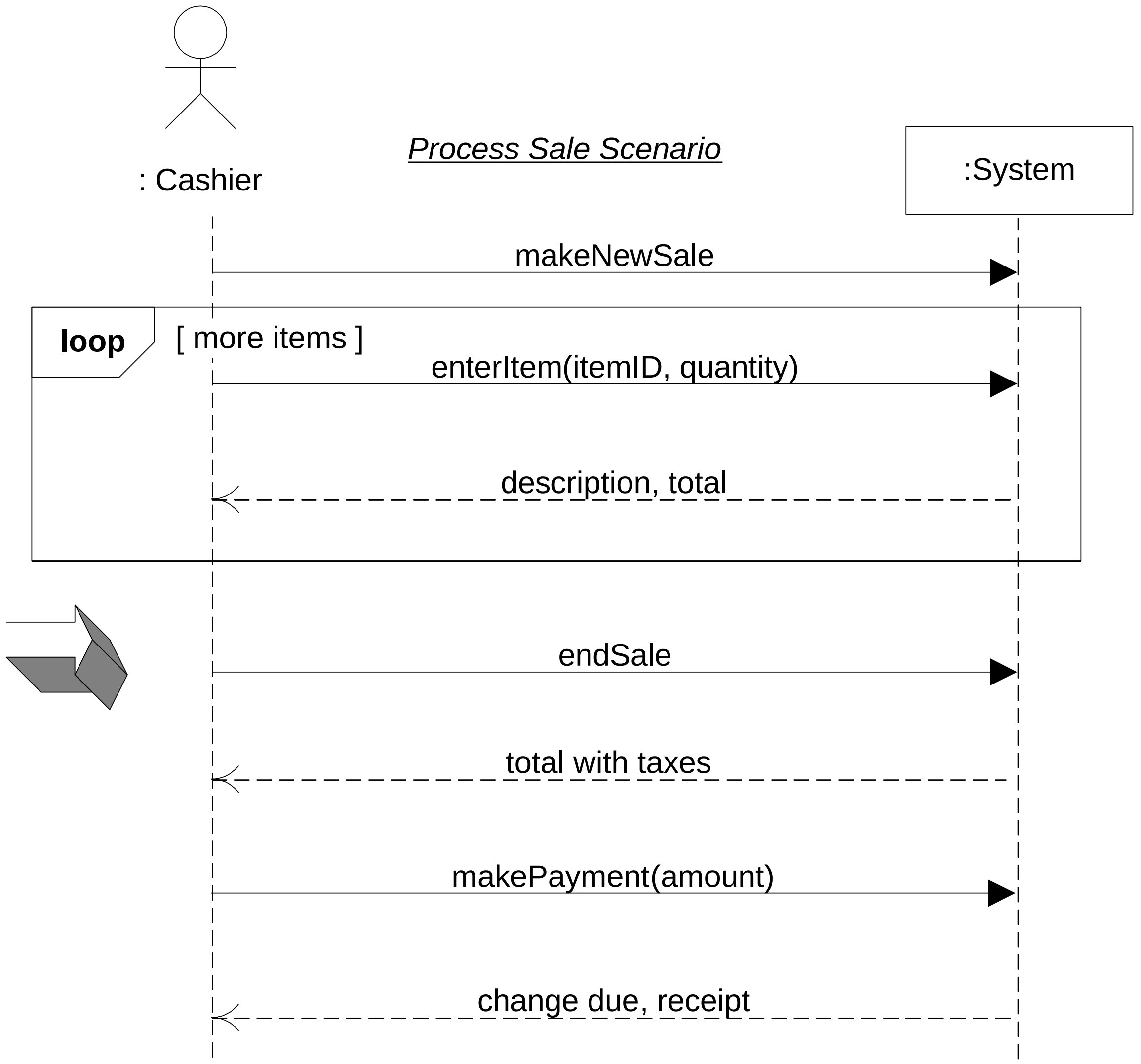
Use case Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
    - Draw Interaction Diagram (Dynamic view)
    - Expand Design Class Diagram (Static view)

PoS: Process Sale scenario: SSD

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.
...



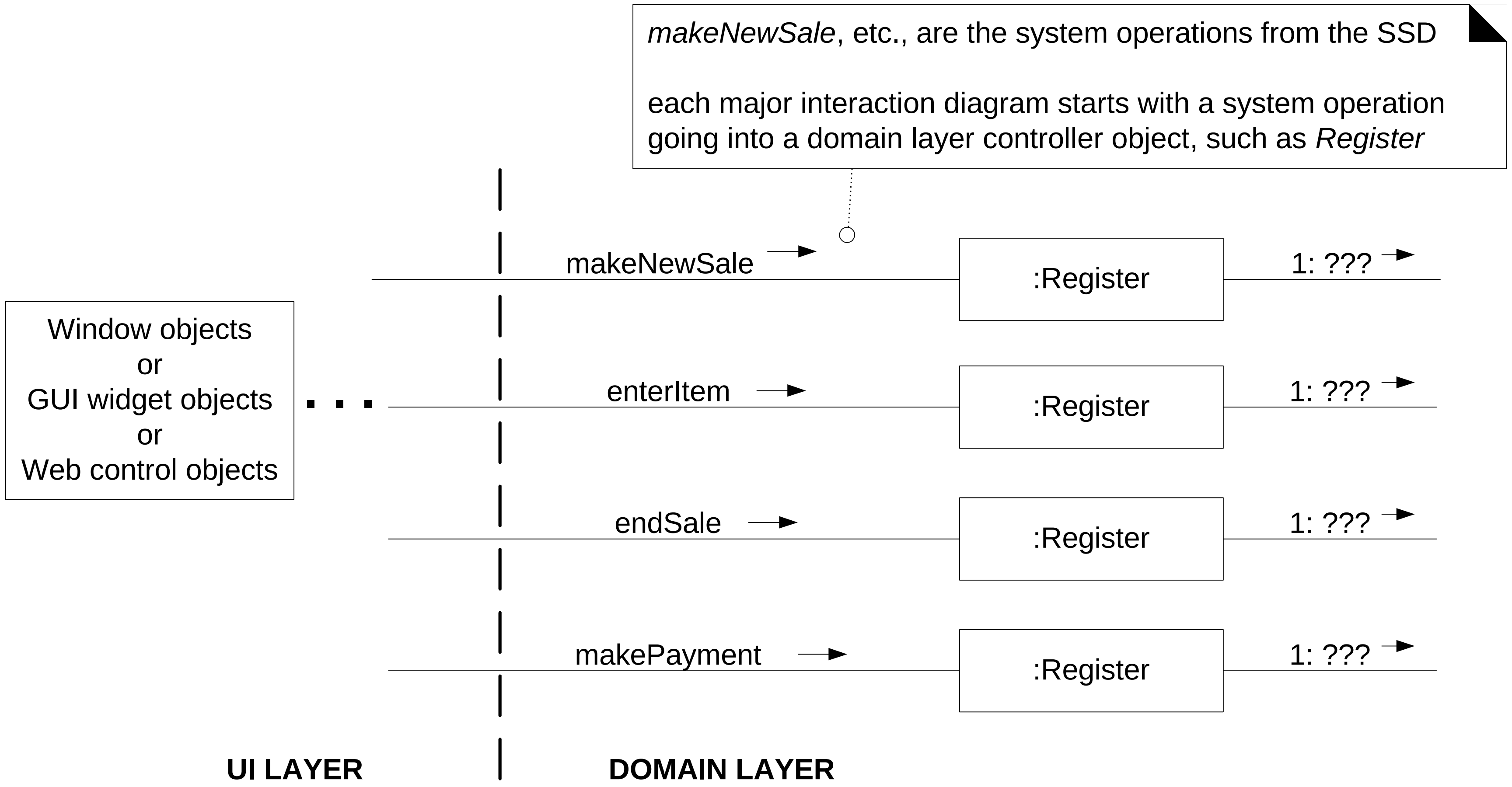
Use case Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
   - Draw Interaction Diagram (Dynamic view)
   - Expand Design Class Diagram (Static view)

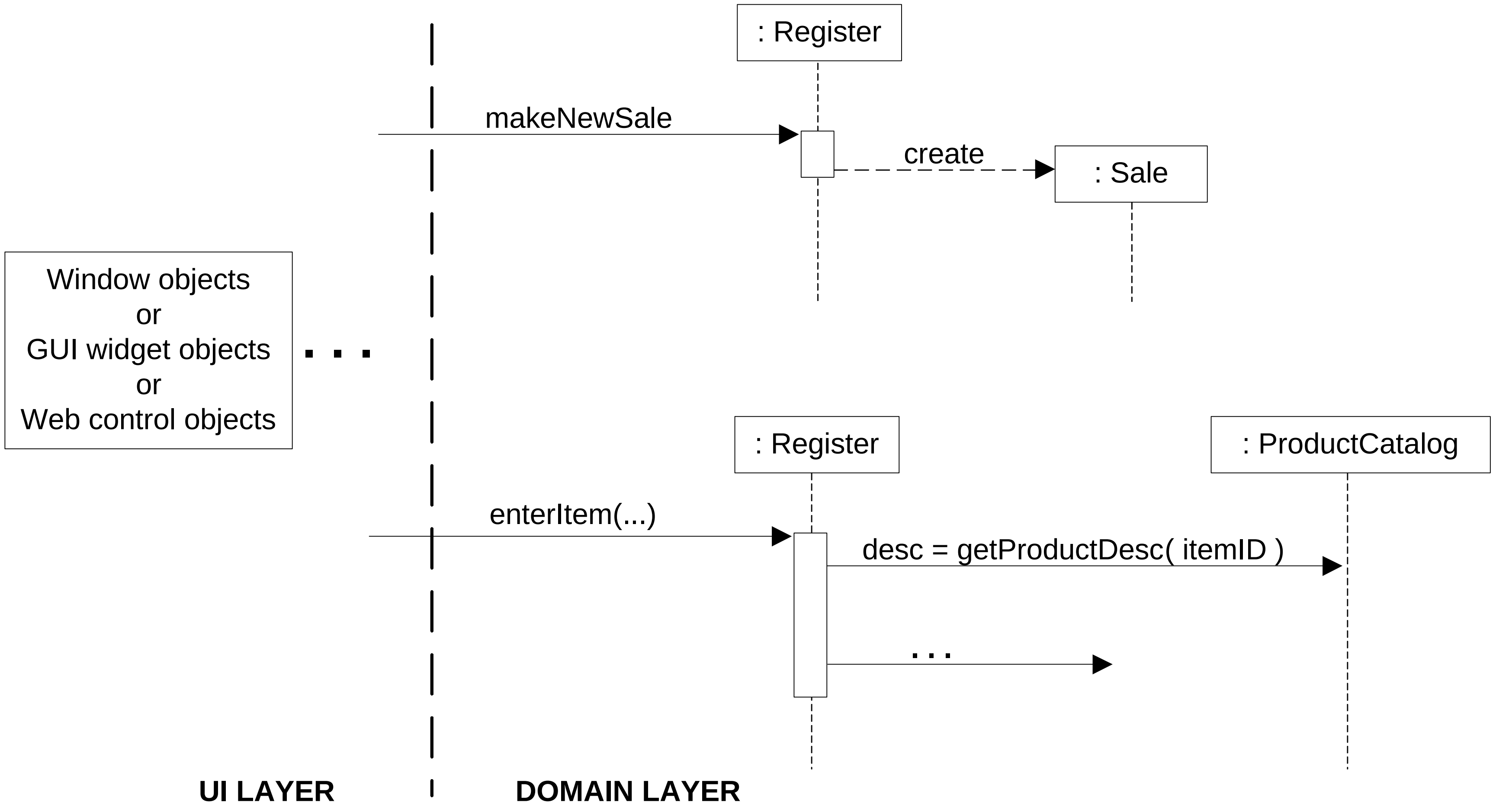
PoS: Process Sale

- System Operations
  - makeNewSale
  - enterItem
  - endSale
  - makePayment

Communication Diagrams


makeNewSale, etc., are the system operations from the SSD
each major interaction diagram starts with a system operation going into a domain layer controller object, such as *Register*
Window objects or GUI widget objects or Web control objects
makeNewSale
enterItem
endSale
makePayment
:Register
:Register
:Register
:Register
1: ???
1: ???
1: ???
1: ???
UI LAYER
DOMAIN LAYER

Sequence Diagrams



PoS: Process Sale

- System Operations
  - makeNewSale
  - enterItem
  - endSale
  - makePayment

Use case Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
   - Draw Interaction Diagram (Dynamic view)
   - Expand Design Class Diagram (Static view)

PoS: Process Sale: makeNewSale - Operation Contract

- 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 Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
   - Draw Interaction Diagram (Dynamic view)
   - Expand Design Class Diagram (Static view)

PoS: Process Sale: makeNewSale – Controller Class

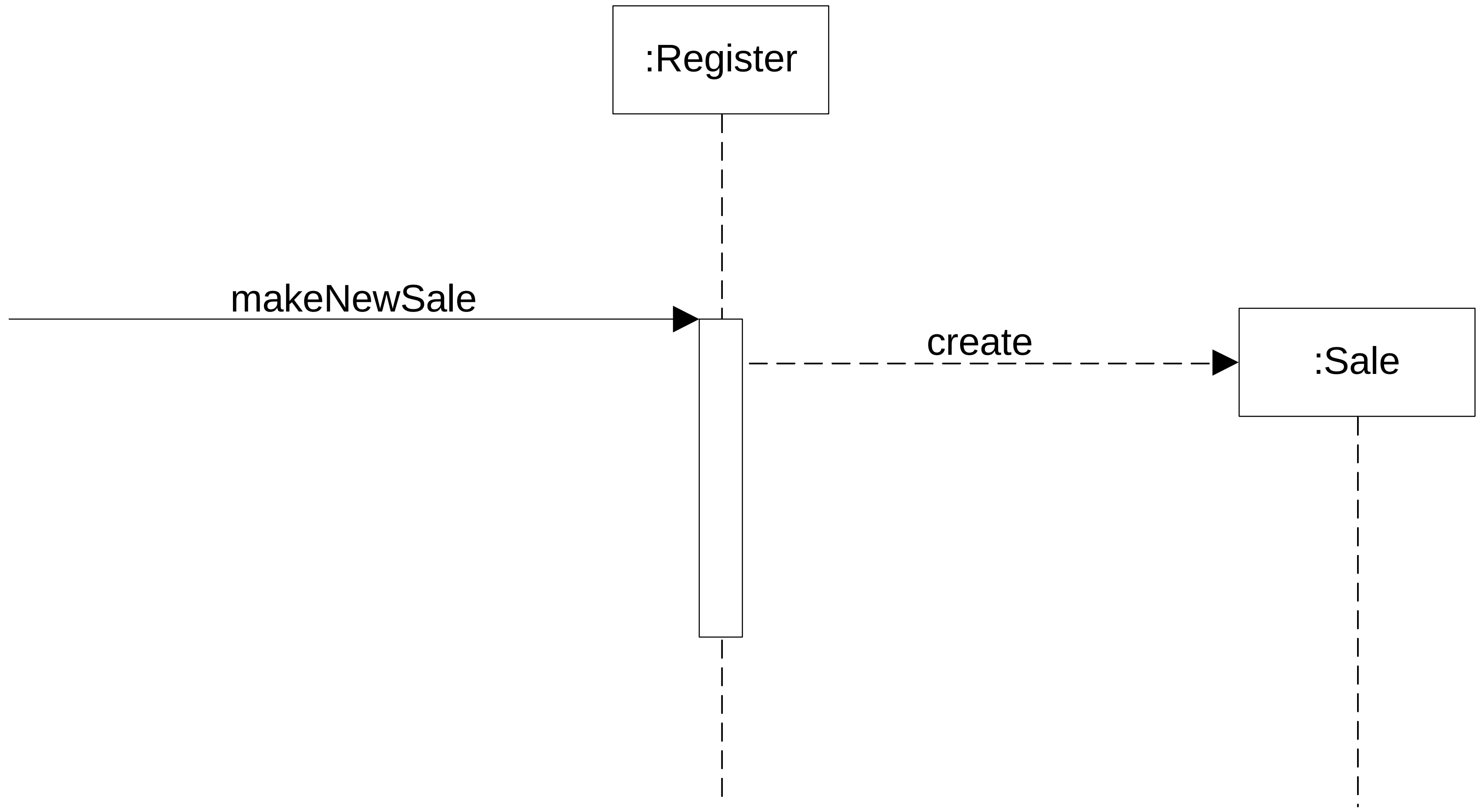
- Choices
  - Overall system / subsystem
    - Store
    - Register
    - POSSystem
  - Use case handler
    - ProcessSaleHandler
    - ProcessSaleSession

PoS: Process Sale: makeNewSale – Controller Class

- Choices
  - Overall system / subsystem
    - Store
    - Register
    - POSSystem
  - Use case handler
    - ProcessSaleHandler
    - ProcessSaleSession
- ‘Register’ selected

PoS: Process Sale: makeNewSale – Controller Class

- By GRASP Controller Principle



Use case Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
   - Draw Interaction Diagram (Dynamic view)
   - Expand Design Class Diagram (Static view)

PoS: Process Sale: makeNewSale – Identify responsibilities

- 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.

PoS: Process Sale: makeNewSale – Identify responsibilities

- 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 Realization: Procedure

1. Select Use case
2. Select Use case scenario
3. Draw SSD
4. Identify System operations
5. Develop Operation contracts
6. Choose Controller Class
7. Identify responsibilities
8. Assign responsibilities to objects
    - Draw Interaction Diagram (Dynamic view)
    - Expand Design Class Diagram (Static view)
    - Keep track of long-term objects

PoS: Process Sale: makeNewSale – Creating a New Sale

- GRASP Creator Principle
  - Assign class B to create instance of class A, if
    - B contains or compositely aggregates A
    - B records A
    - B closely uses A
    - B has initializing data for A that will be passed to A when it is created
  - If >1 applicable, prefer class B which aggregates A

PoS: Process Sale: makeNewSale – Creating a New Sale

- GRASP Creator Principle
  - Assign class B to create instance of class A, if
    - B contains or compositely aggregates A
    - Register records Sale
    - B closely uses A
    - B has initializing data for A that will be passed to A when it is created
  - If >1 applicable, prefer class B which aggregates A
- Responsibility assigned to 'Register' class

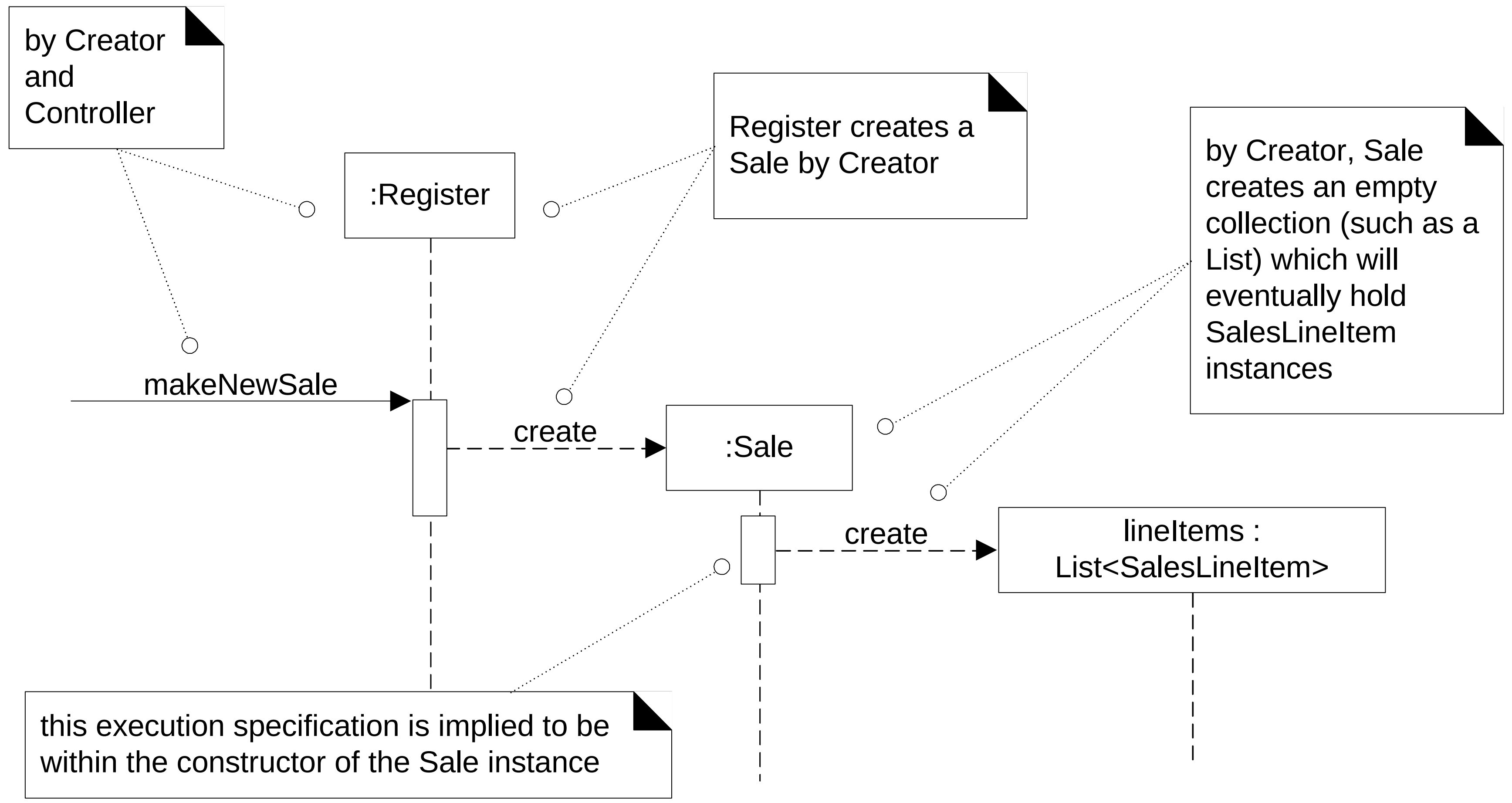
PoS: Process Sale: makeNewSale – Identify responsibilities

- 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.

PoS: Process Sale: makeNewSale – Identify responsibilities

- 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.

PoS: Process Sale: makeNewSale – Creating a New Sale


by Creator and Controller
:Register
Register creates a Sale by Creator
by Creator, Sale creates an empty collection (such as a List) which will eventually hold SalesLineItem instances
makeNewSale
create
:Sale
create
lineItems : List<SalesLineItem>
this execution specification is implied to be within the constructor of the Sale instance

PoS: Process Sale

- System Operations
  - makeNewSale
  - enterItem
  - endSale
  - makePayment

PoS: Process Sale: enterItem - Operation Contract

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

PoS: Process Sale: makeNewSale – Controller Class

- Same for all system operations of use case
- ‘Register’ selected

PoS: Process Sale: enterItem – Identify responsibilities

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

PoS: Process Sale: enterItem – Identify responsibilities

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

PoS: Process Sale: enterItem – Creating a New SalesLineItem

- GRASP Creator Principle
  - Assign class B to create instance of class A, if
    - Sale contains SalesLineItem
    - B records A
    - B closely uses A
    - Register has initializing data for SalesLineItem when it is created
  - If >1 applicable, prefer class B which aggregates A

PoS: Process Sale: enterItem – Creating a New SalesLineItem

- GRASP Creator Principle
  - Assign class B to create instance of class A, if
    - Sale contains SalesLineItem
    - B records A
    - B closely uses A
    - Register has initializing data for SalesLineItem when it is created
  - If >1 applicable, prefer class B which aggregates A
- ‘Sale’ selected

PoS: Process Sale: enterItem – Identify responsibilities

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

PoS: Process Sale: enterItem – Identify responsibilities

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

PoS: Process Sale: enterItem – Finding a ProductDescription

- Start assigning responsibilities by clearly stating the responsibility

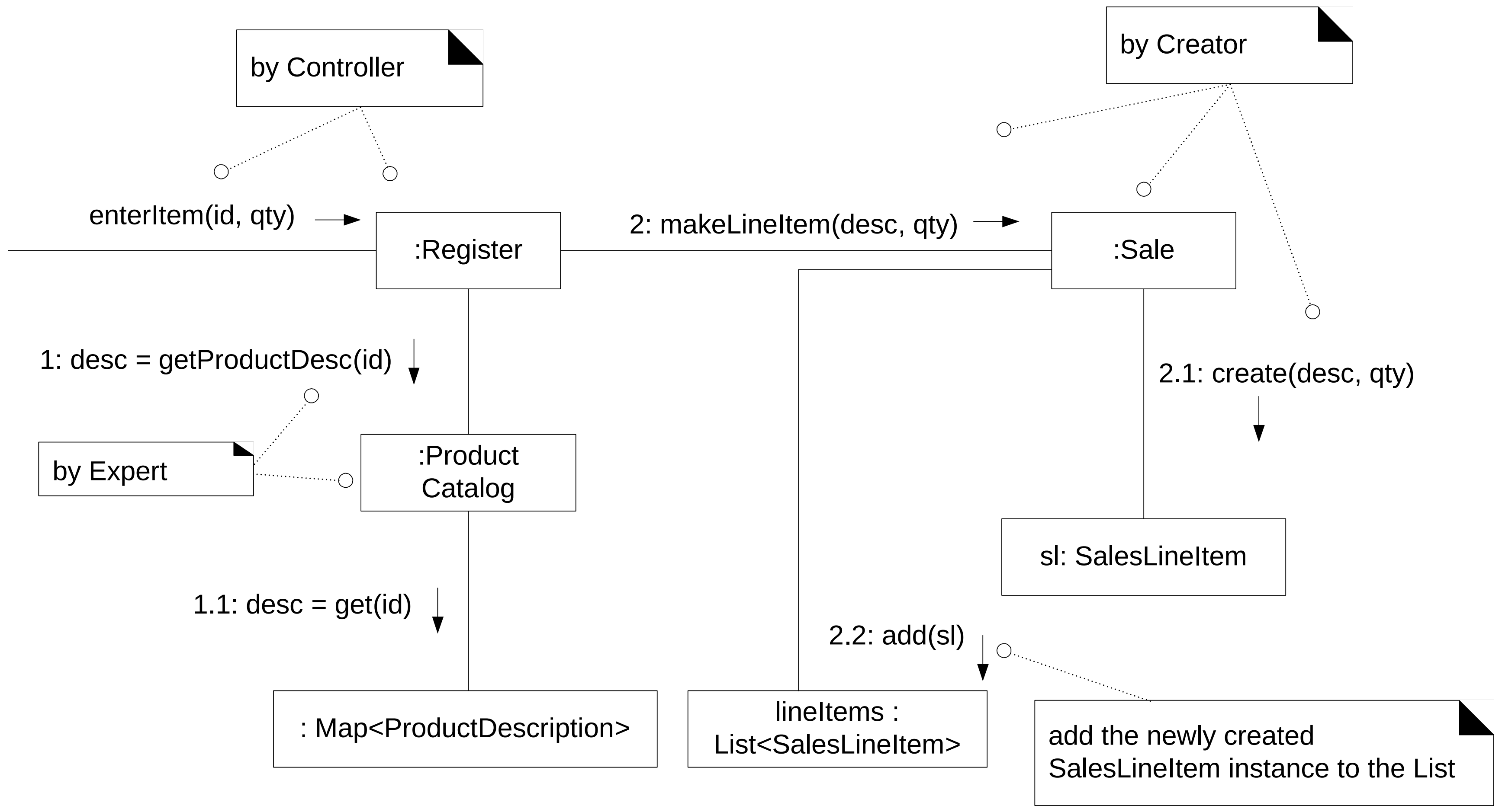
  - Who should be responsible for knowing a ProductDescription, based on an itemId match?
  - Information Expert GRASP Principle
    - ProductCatalog
      - Contains ProductDescriptions

PoS: Process Sale: enterItem – Visibility to ProductCatalog

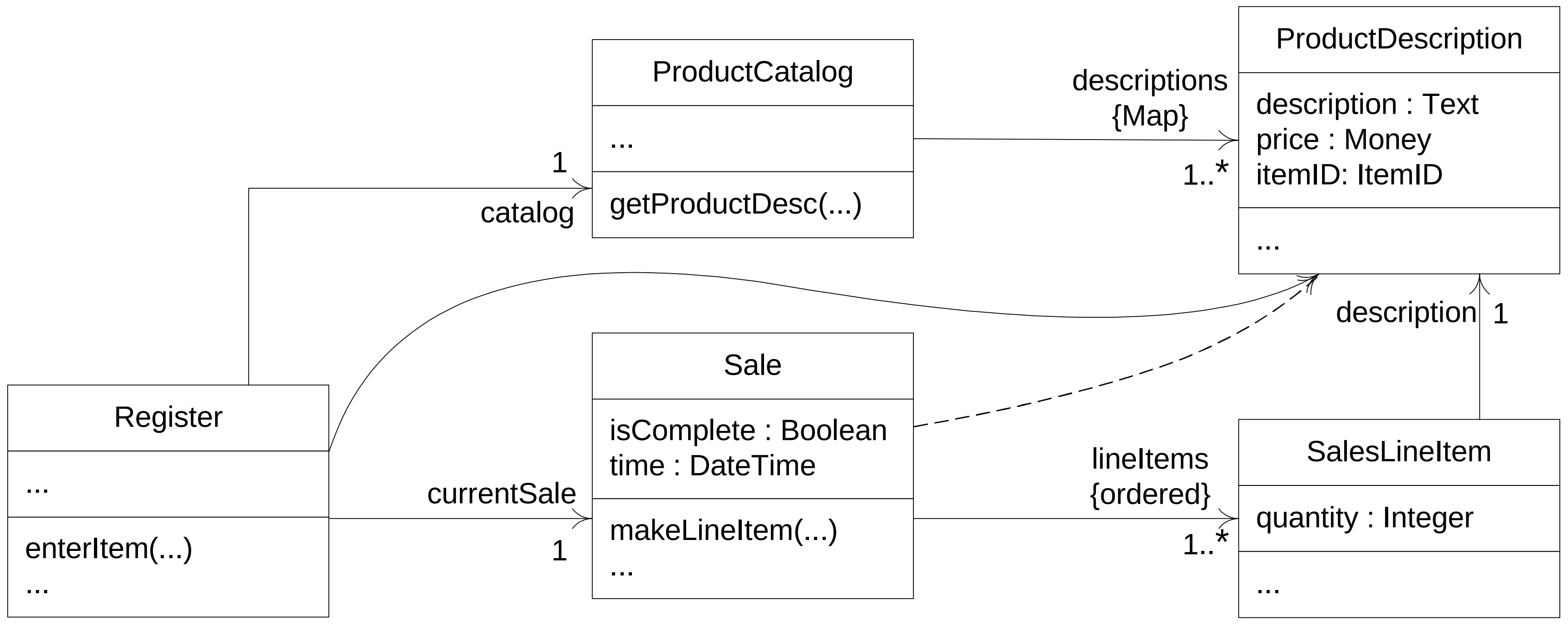
- Need a Handle / Reference
- Information Expert GRASP Principle
  - Store
  - Register

  - Sale Class?

PoS: Process Sale: enterItem – Interaction diagram


by Controller
by Creator
enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc = getProductDesc(id)
2.1: create(desc, qty)
by Expert
:Product Catalog
sl: SalesLineItem
1.1: desc = get(id)
2.2: add(sl)
: Map<ProductDescription>
lineItems : List<SalesLineItem>
add the newly created SalesLineItem instance to the List

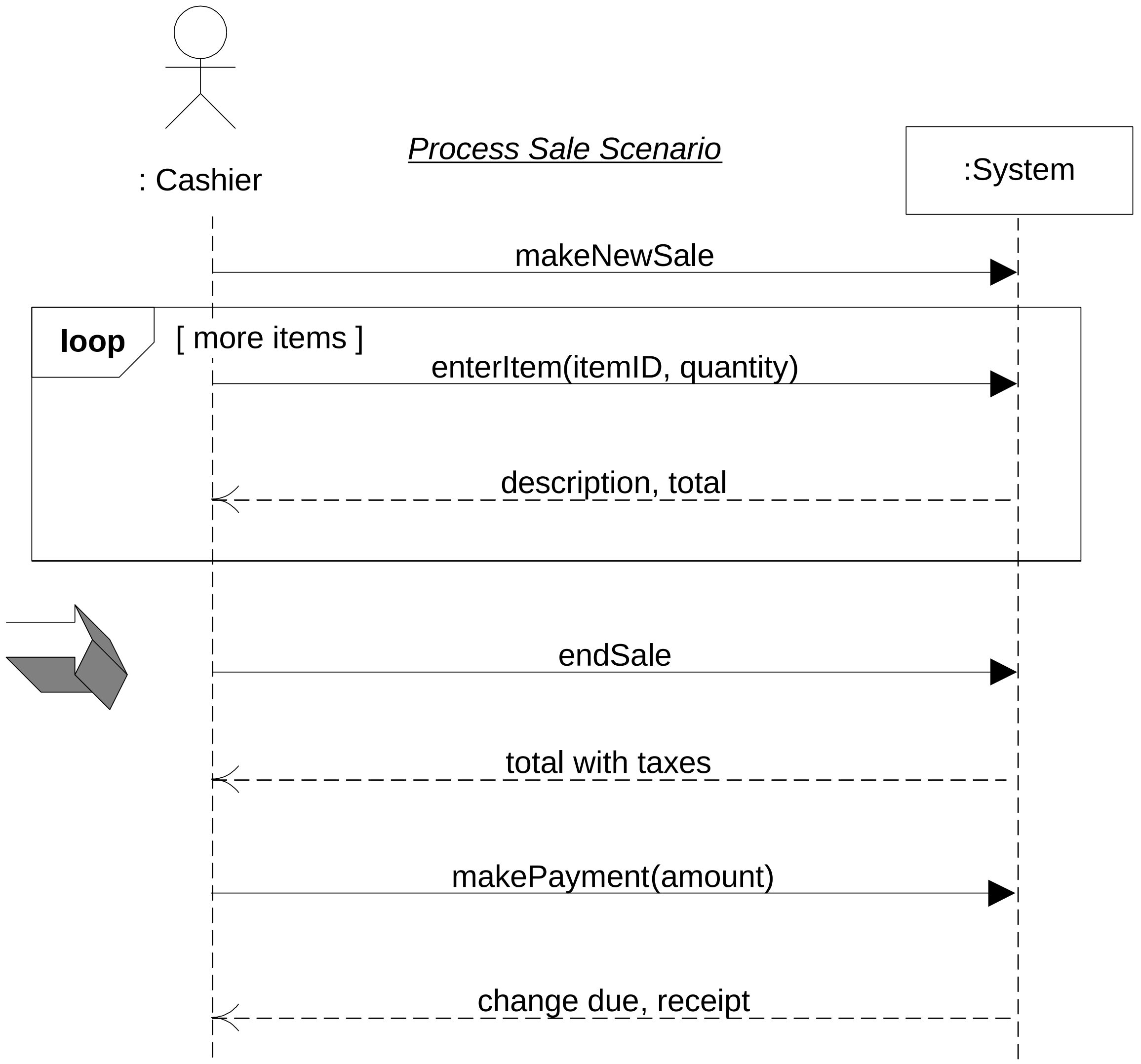
PoS: Process Sale: enterItem – Design Class Diagram (Partial)



PoS: Process Sale: enterItem – Display intermediate output

<u>Simple cash-only *Process Sale* scenario:</u>

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.
...



PoS: Process Sale

- System Operations
  - makeNewSale
  - enterItem
  - endSale
  - makePayment

PoS: Process Sale: endSale - Operation Contract

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

PoS: Process Sale: endSale – Controller Class

- Same for all system operations of use case
- ‘Register’ selected

PoS: Process Sale: endSale - Identify responsibilities

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

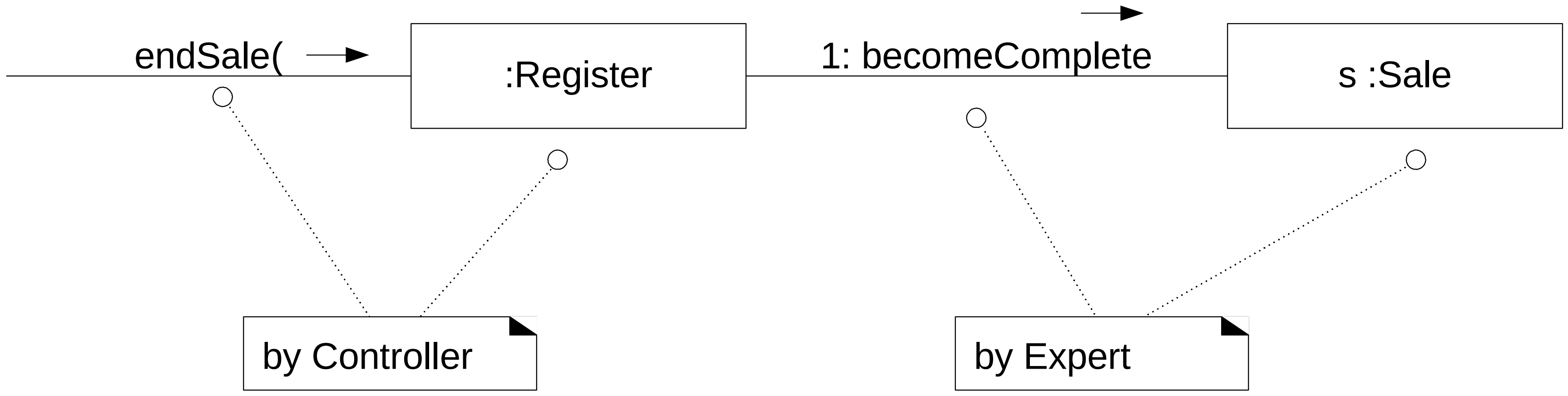
PoS: Process Sale: endSale – Setting Sale.isComplete attribute

- Start assigning responsibilities by clearly stating the responsibility
  - Who should be responsible for setting the attribute

PoS: Process Sale: endSale – Setting Sale.isComplete attribute

- Start assigning responsibilities by clearly stating the responsibility
  - Who should be responsible for setting the attribute
  - Information Expert GRASP Principle
    - Sale
      - Has attribute

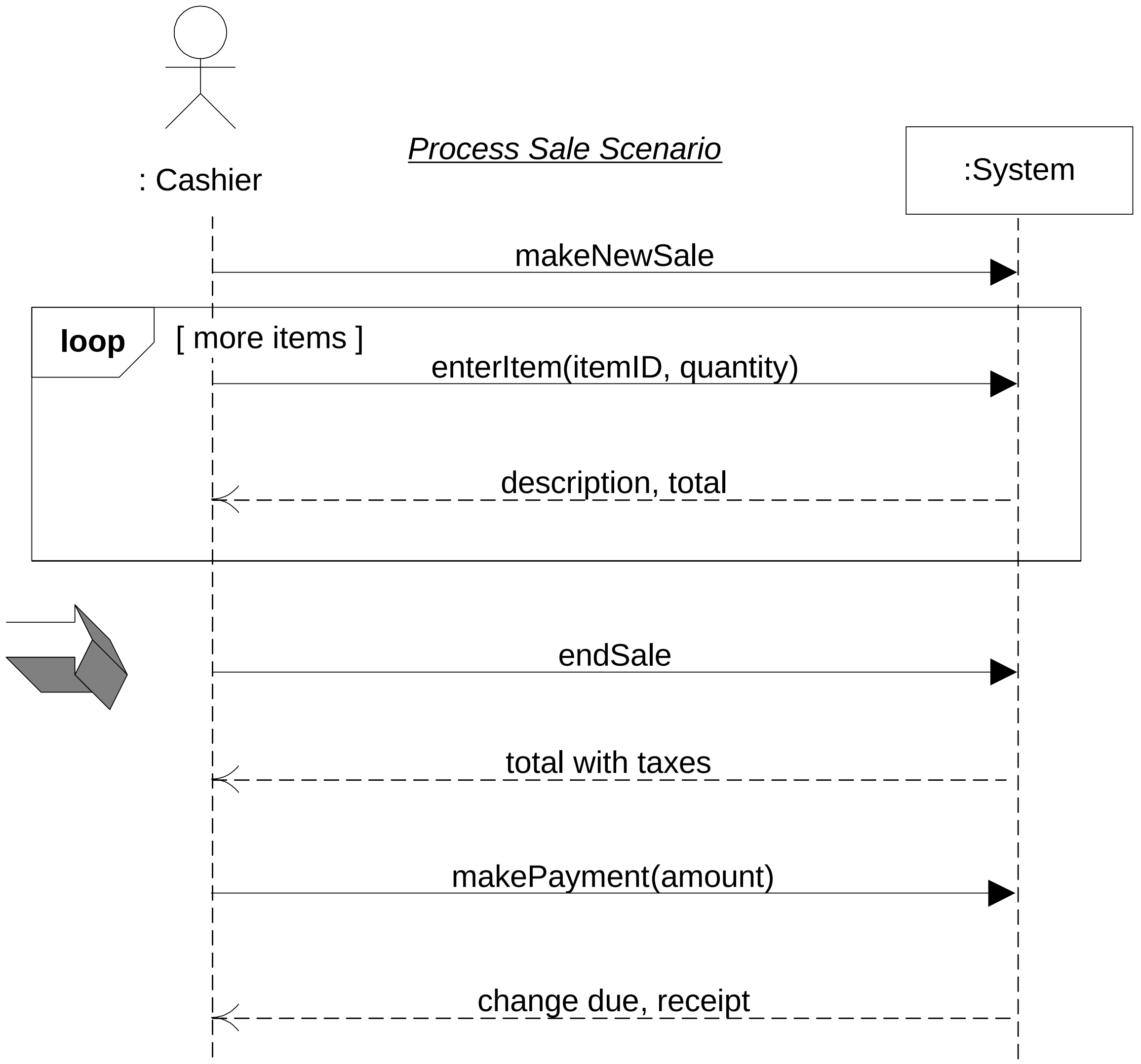
PoS: Process Sale: endSale – Interaction Diagram



PoS: Process Sale: enterItem – Display intermediate output

Simple cash-only *Process Sale* scenario:

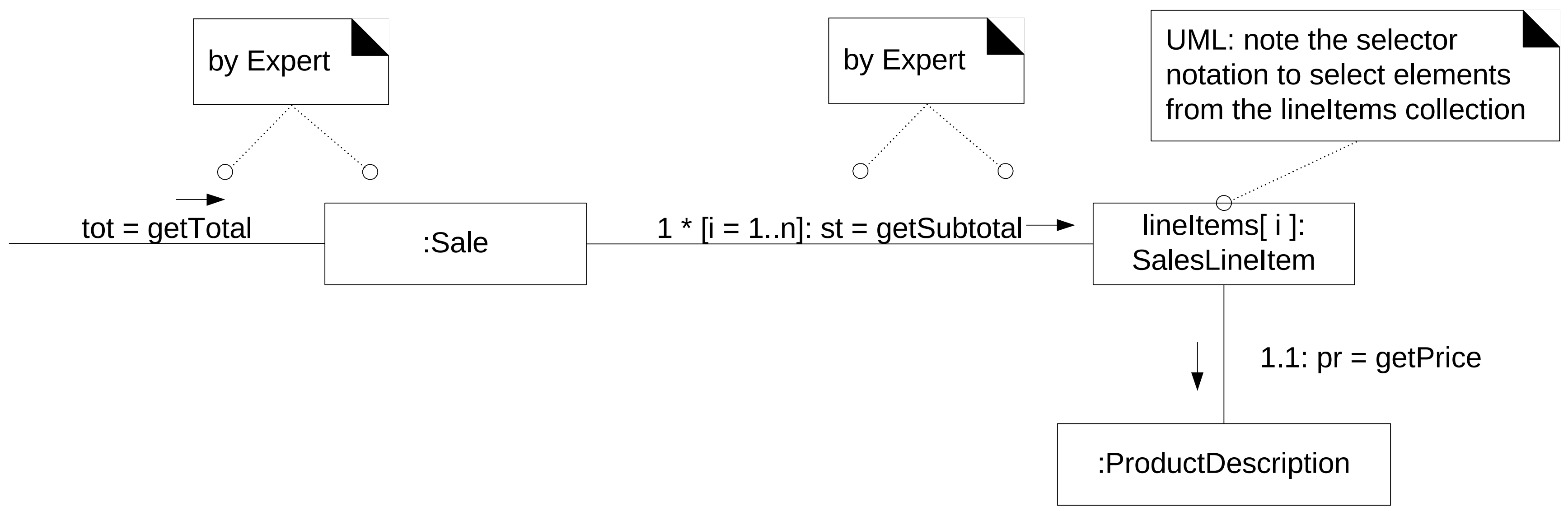
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total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



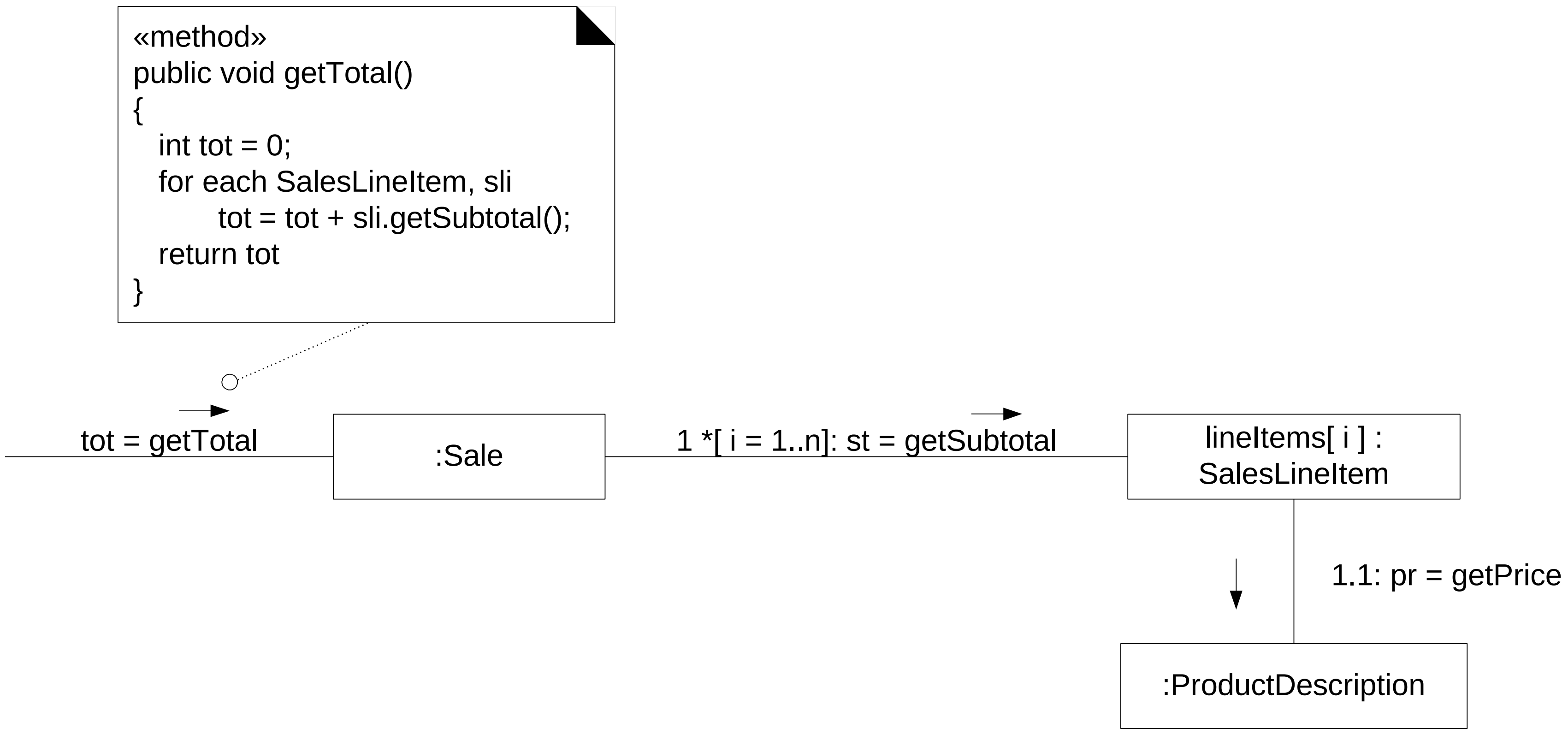
PoS: Process Sale: endSale – Calculating Sale total

- Start assigning responsibilities by clearly stating the responsibility
  - Who should be responsible for setting the attribute
  - Information Expert GRASP Principle
    - Sale
      - Has / can request required information

PoS: Process Sale: endSale – Calculating Sale total – Interaction Diagram


by Expert
by Expert
UML: note the selector notation to select elements from the lineItems collection
tot = getTotal
:Sale
1 * [i = 1..n]: st = getSubtotal
lineItems[i]:
SalesLineItem
1.1: pr = getPrice
:ProductDescription

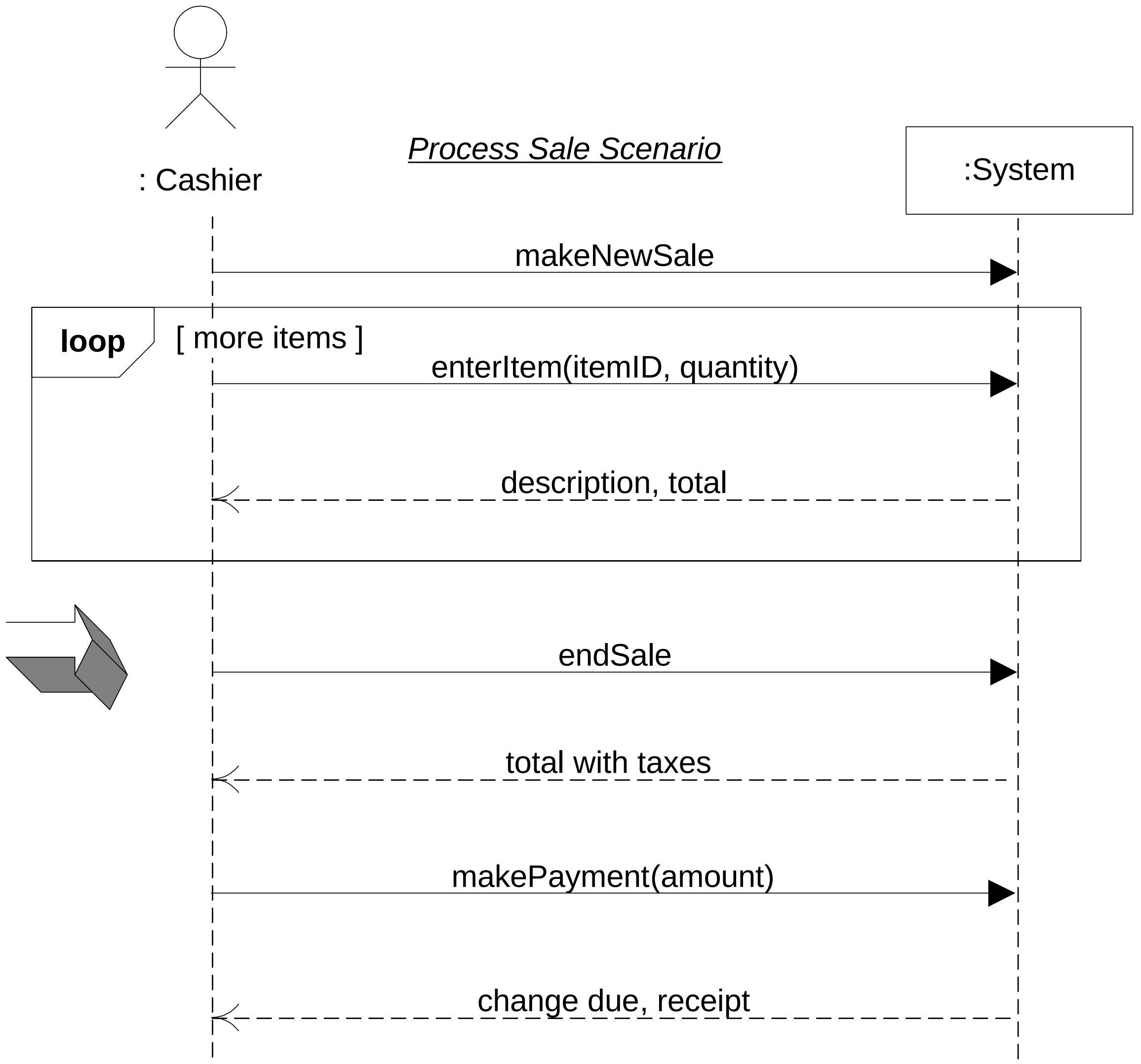
UML Notation: Showing a method in a note symbol


«method»
public void getTotal()
{
int tot = 0;
for each SalesLineItem, sli
tot = tot + sli.getSubtotal();
return tot
}
tot = getTotal
:Sale
1 *[i = 1..n]: st = getSubtotal
lineItems[i] : SalesLineItem
1.1: pr = getPrice
:ProductDescription

PoS: Process Sale: enterItem – Display intermediate output

<u>Simple cash-only *Process Sale* scenario:</u>

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.
...



PoS: Process Sale

- System Operations
  - makeNewSale
  - enterItem
  - endSale
  - makePayment

PoS: Process Sale: makePayment - Operation Contract

- Operation: makePayment()
- 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).

PoS: Process Sale: makePayment - Operation Contract

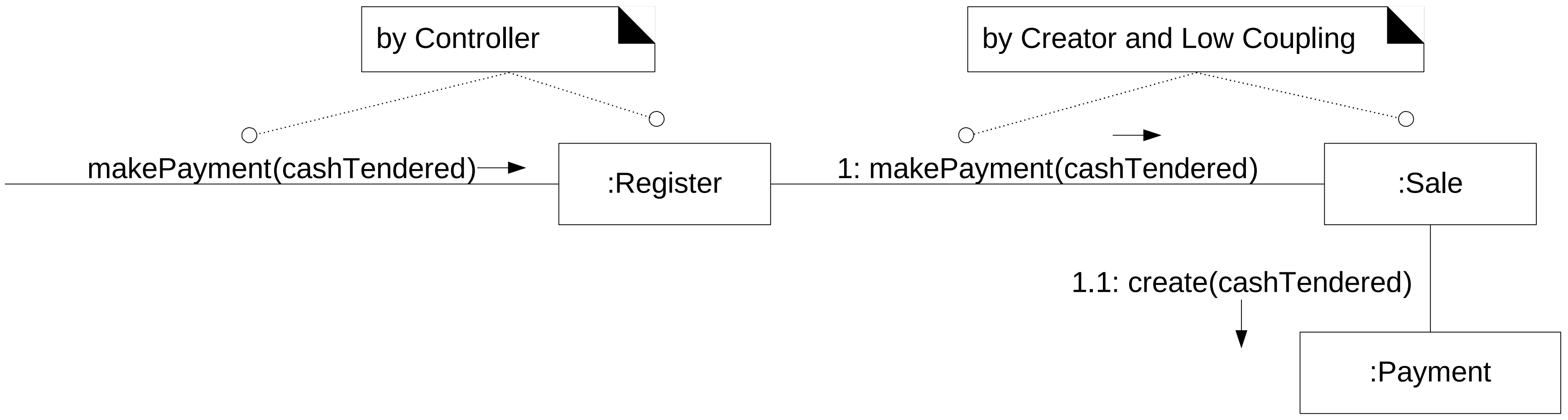
- Operation: makePayment()
- Cross References: Use Cases: Process Sale
- Preconditions: There is an underway sale
- Postconditions:
  - A Payment instance p was created (instance creation).
  - p.amountTendered became amount (attribute modification)
  - p was associated with the current Sale (association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

PoS: Process Sale: makePayment – Creating the Payment

- Choices
  - Creator
    - Register records Payment
    - Sale closely uses Payment
  - Expert
    - Register has initialization data
- Evaluate alternate design choices by
  - Coupling
  - Cohesion

PoS: Process Sale: makePayment – Creating the Payment - Interaction Diagram

- Sale creates Payment instance


by Controller
by Creator and Low Coupling
makePayment(cashTendered)
:Register
1: makePayment(cashTendered)
:Sale
1.1: create(cashTendered)
:Payment

PoS: Process Sale: makePayment - Operation Contract

- Operation: makePayment()
- 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).

PoS: Process Sale: makePayment - Operation Contract

- Operation: makePayment()
- 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).

PoS: Process Sale: makePayment - Operation Contract

- Operation: makePayment()
- 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).

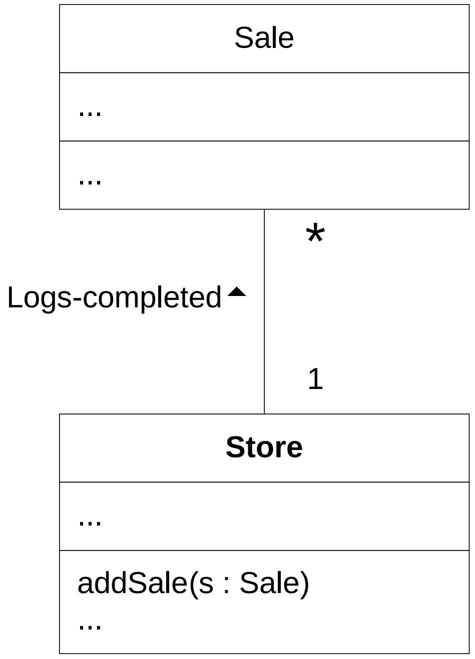
PoS: Process Sale: makePayment – Logging a Sale

- Who is responsible for knowing all the logged sales and doing the logging?

PoS: Process Sale: makePayment – Logging a Sale

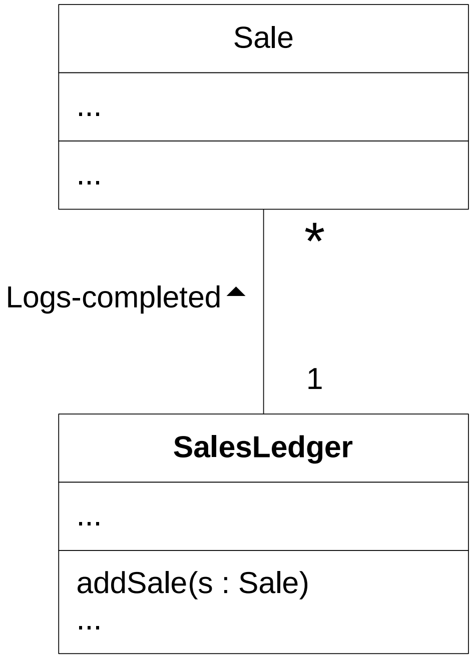
- Who is responsible for knowing all the logged sales and doing the logging?
- Choices
  - Expert
    - Register
    - Store
    - SaleLedger

PoS: Process Sale: makePayment – Logging a Sale – Options



Store is responsible for knowing and adding completed Sales.

Acceptable in early development cycles if the Store has few responsibilities.



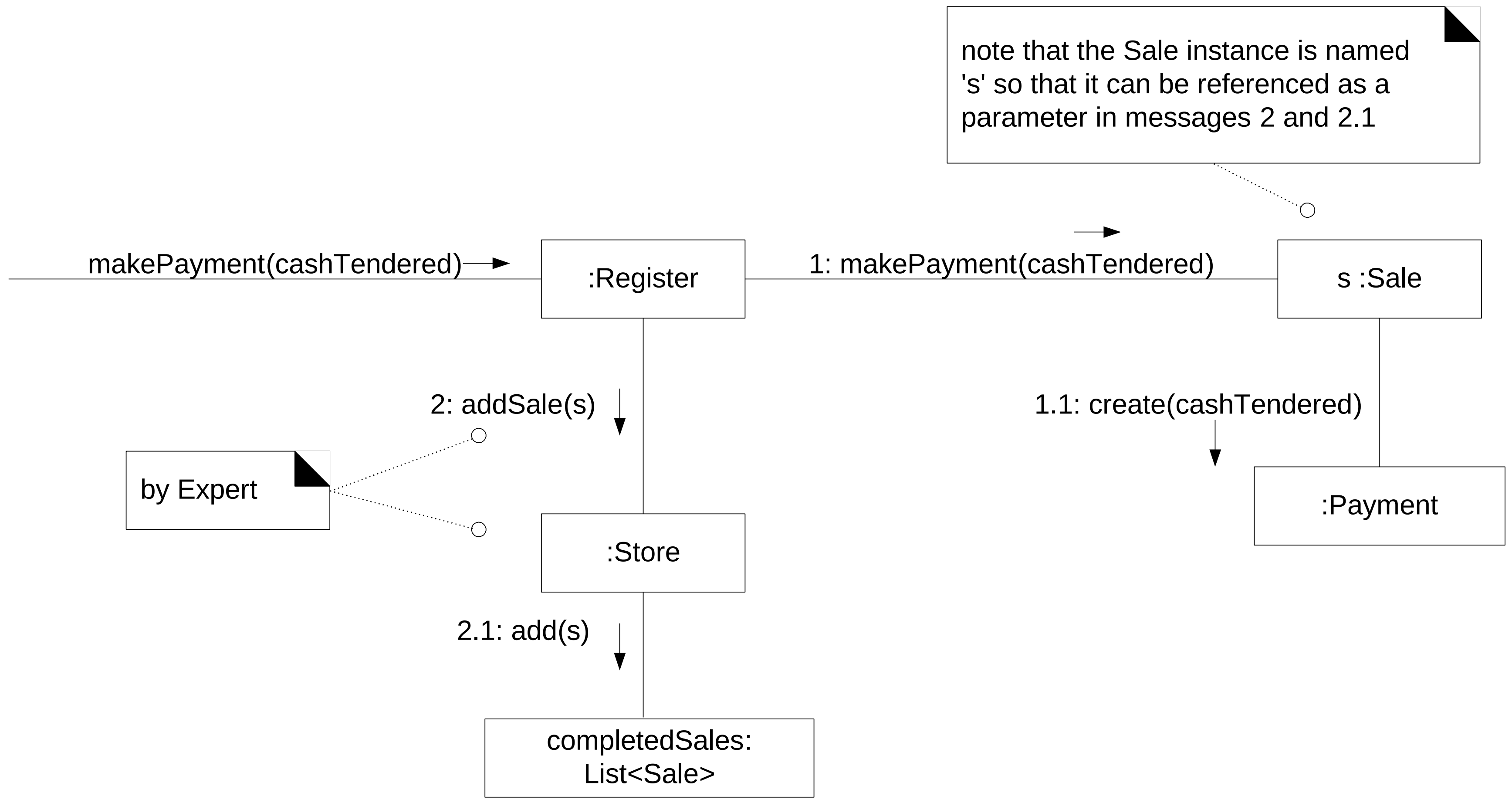
SalesLedger is responsible for knowing and adding completed Sales.

Suitable when the design grows and the Store becomes uncohesive.

PoS: Process Sale: makePayment – Logging a Sale

- Who is responsible for knowing all the logged sales and doing the logging?
- Choices
  - Expert
    - Register
    - Store
    - SaleLedger
- Store Class selected

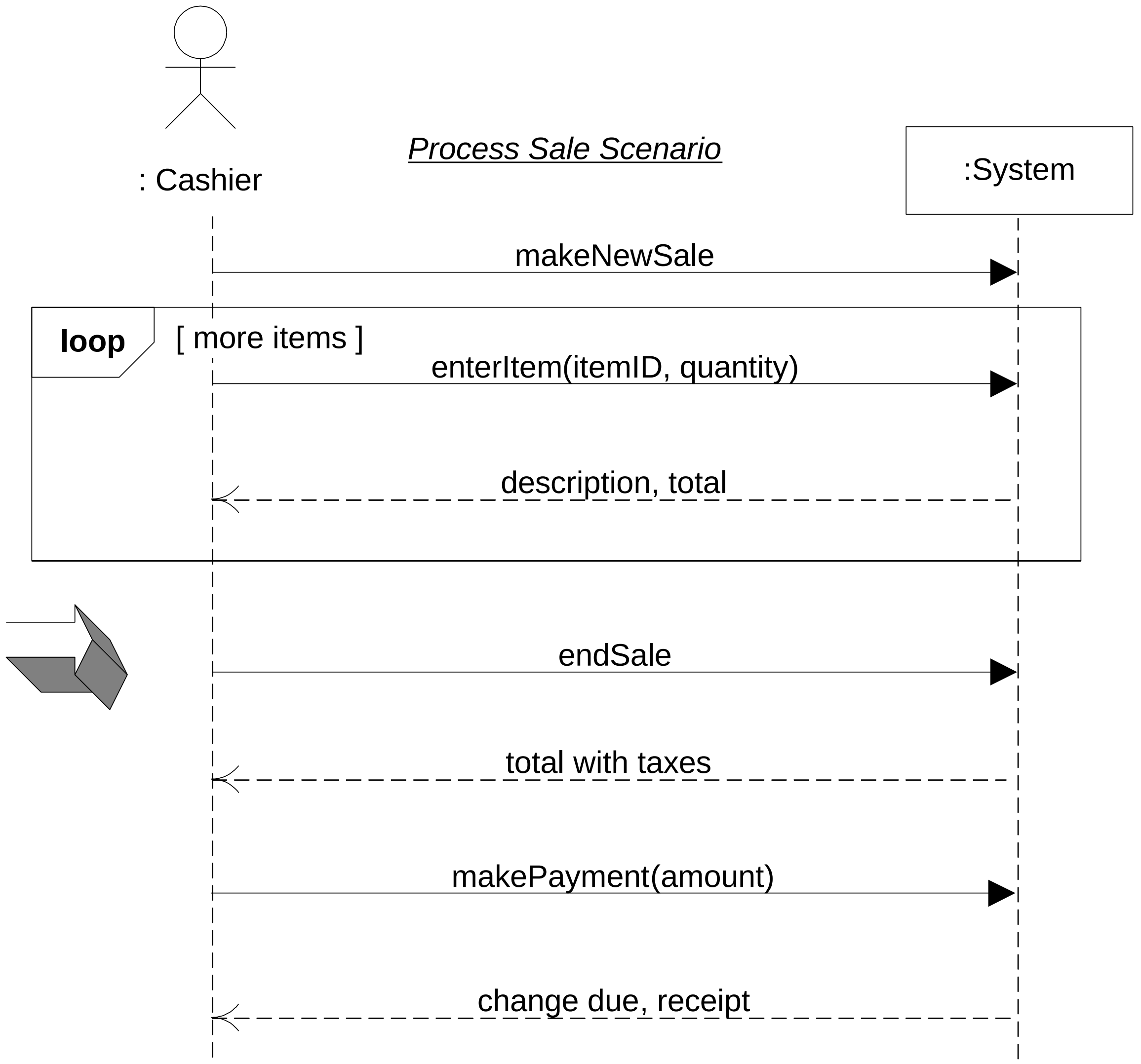
PoS: Process Sale: makePayment – Logging a completed sale


note that the Sale instance is named 's' so that it can be referenced as a parameter in messages 2 and 2.1
makePayment(cashTendered)
:Register
1: makePayment(cashTendered)
s :Sale
2: addSale(s)
1.1: create(cashTendered)
by Expert
:Payment
:Store
2.1: add(s)
completedSales: List<Sale>

PoS: Process Sale: enterItem – Display intermediate output

<u>Simple cash-only *Process Sale* scenario:</u>

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.
...



PoS: Process Sale: makePayment – Calculating the Balance

- Who is responsible for knowing the balance?
- Choices
  - Expert
    - Sale
    - Payment

PoS: Process Sale: makePayment – Calculating the Balance

- Who is responsible for knowing the balance?
- Choices
  - Expert
    - Sale
    - Payment

- Consider visibility
- Evaluate Coupling and Cohesion

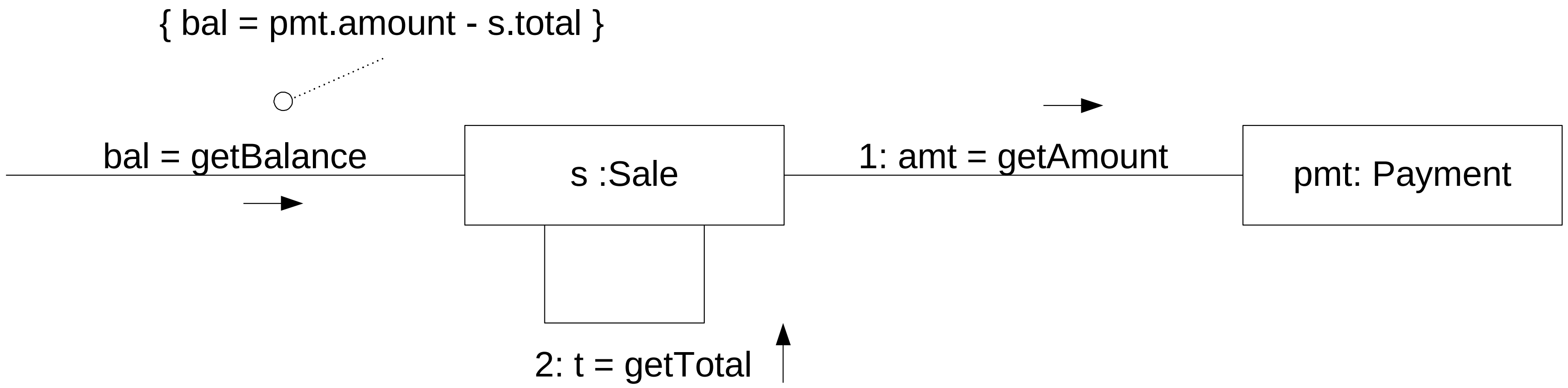
PoS: Process Sale: makePayment – Calculating the Balance

- Who is responsible for knowing the balance?
- Choices
  - Expert
    - Sale
    - Payment

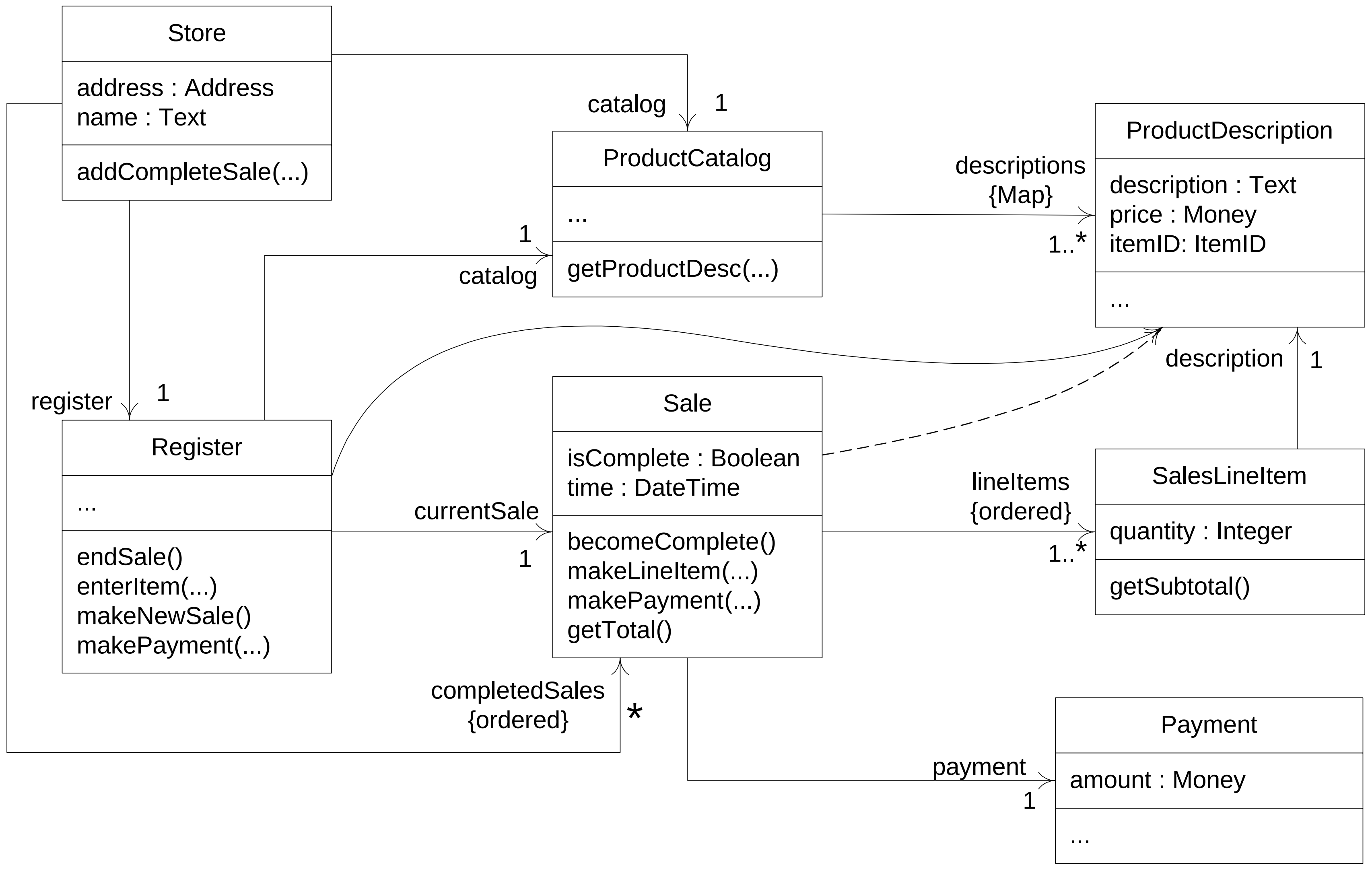
- Consider visibility
- Evaluate Coupling and Cohesion

- ‘Sale’ Class selected

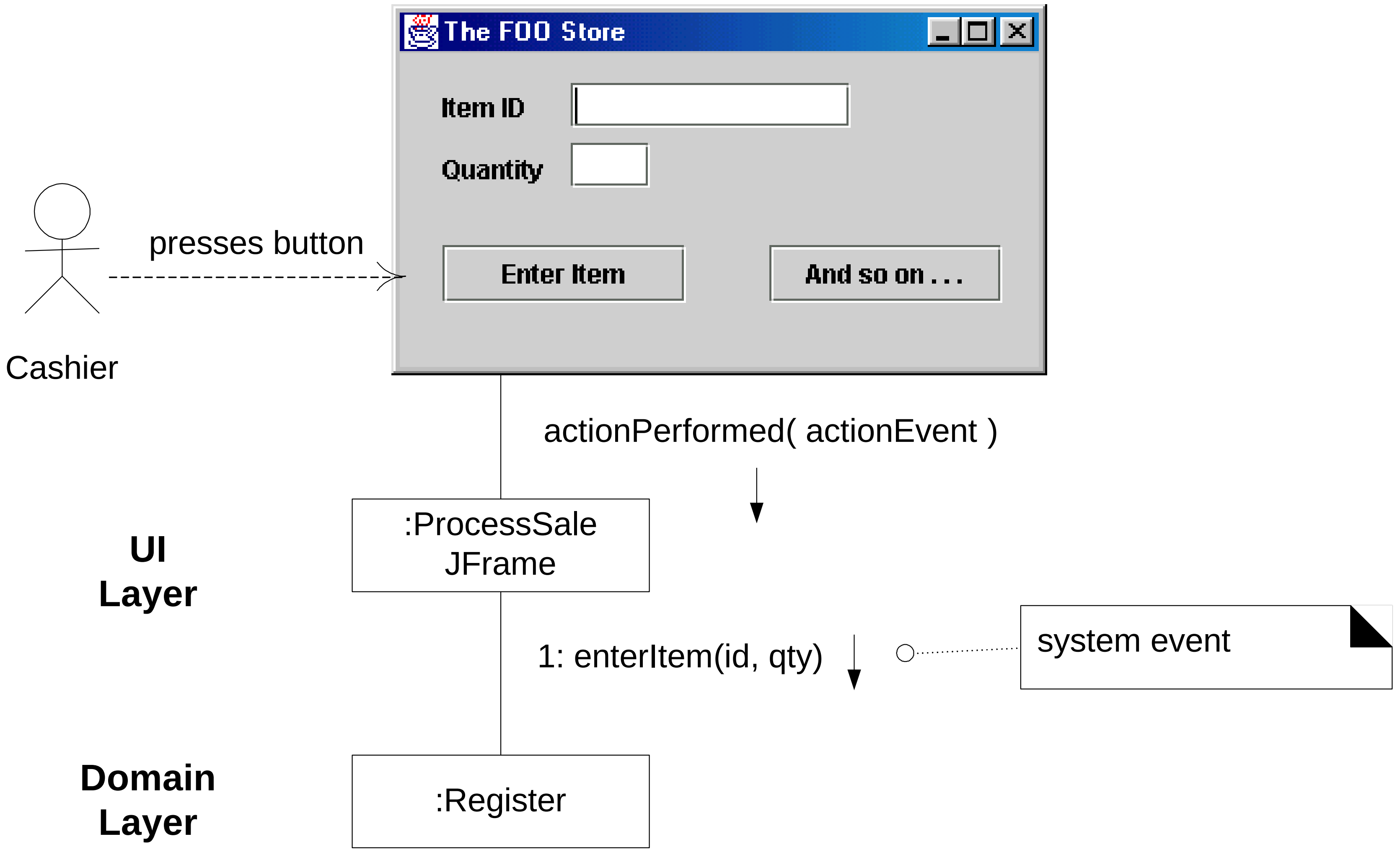
PoS: Process Sale: makePayment – Calculating the Balance – Interaction Diagram



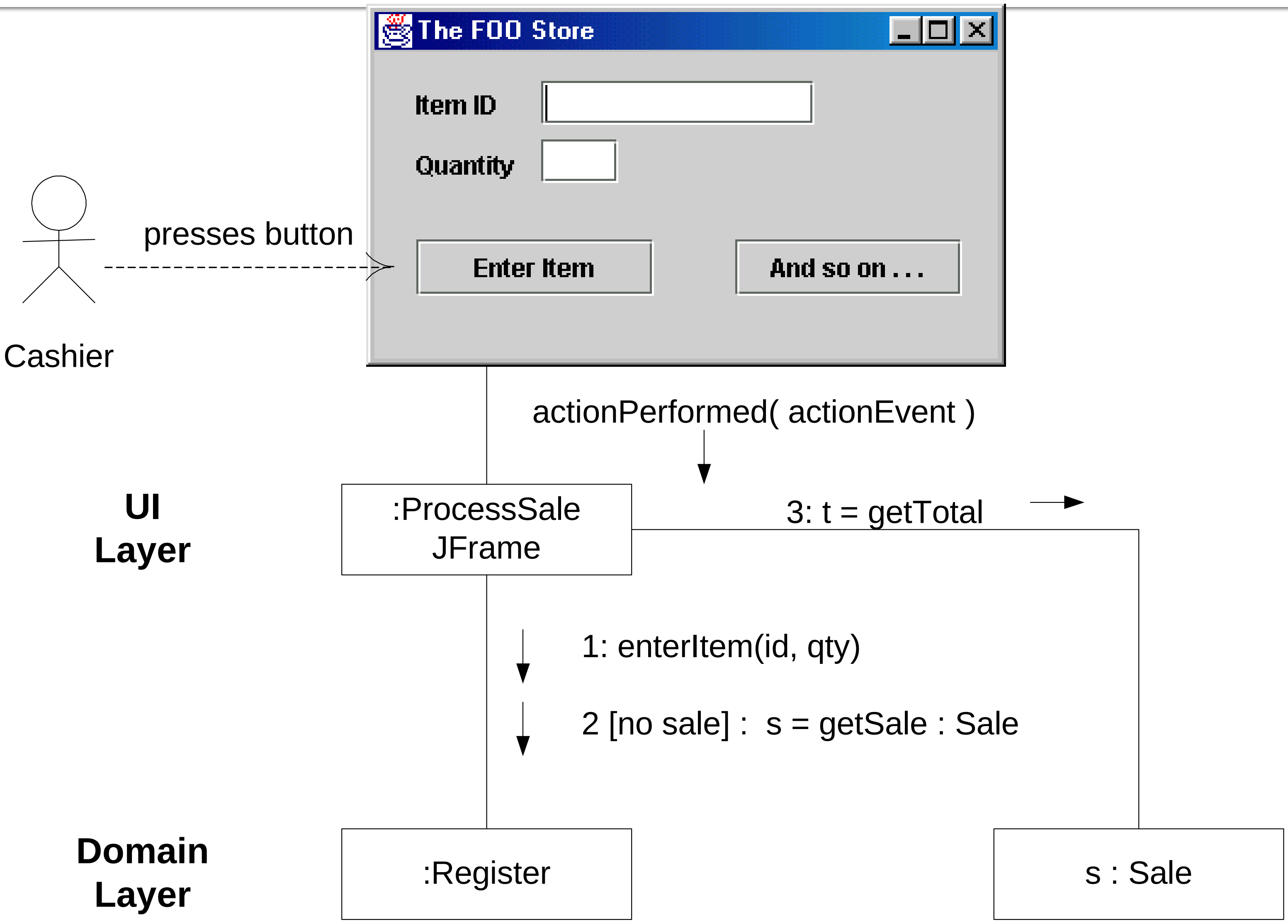
PoS: Process Sale: Design Class Diagram (DCD)



Connecting UI and Domain layers



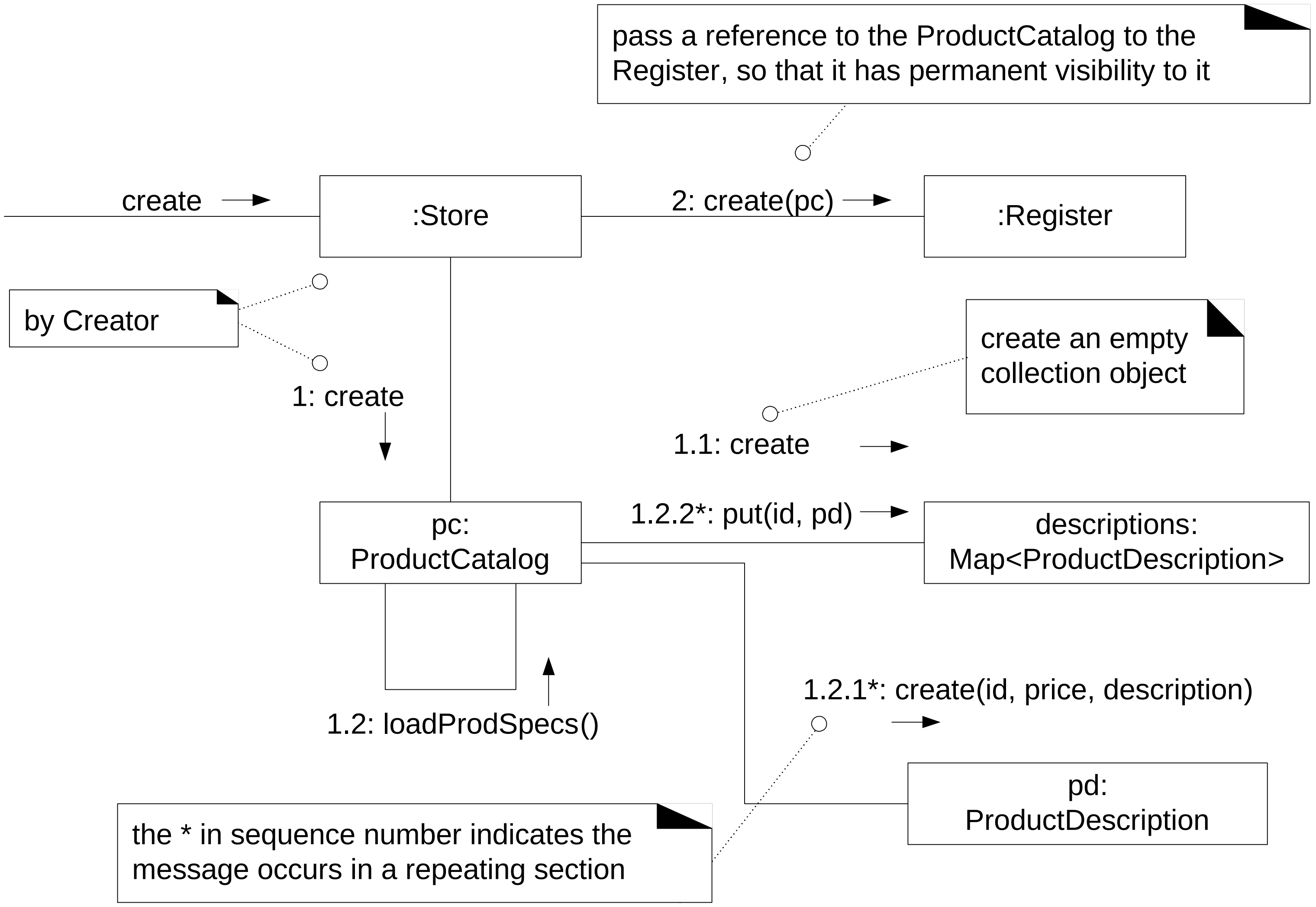
Connecting UI and Domain layers



PoS: Initialization and Startup Use case

- Do the initialization design last
  - Create >=1 initial / peer domain objects
  - Choose Class near the root of aggregation hierarchy of domain objects
  - Choices
    - Register
    - Store
  - Evaluate by Coupling and Cohesion
  - Store Class selected
  - Store.Create
    - Create Store, Register, ProductCatalog, ProductDescriptions
    - Create Associations

PoS: Creation of initial domain object and subsequent objects


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
create an empty collection object
1: create
1.1: create
pc: ProductCatalog
1.2.2*: put(id, pd)
descriptions: Map<ProductDescription>
1.2.1*: create(id, price, description)
1.2: loadProdSpecs()
pd: ProductDescription
the * in sequence number indicates the message occurs in a repeating section

Next sessions…

- GoF Design Patterns

Reading assignment

- Reference Book
  - Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition, Craig Larman, 2004
    - Chapter 18: Object Design Examples with GRASP: Pages 321-350.

GRASP Principles: Summary

General Responsibility Assignment Software Patterns

1. Creator
2. Information Expert
3. Controller
4. Low Coupling
5. High Cohesion
6. Polymorphism
7. Pure Fabrication
8. Indirection
9. Protected Variations