Applying UML and Patterns

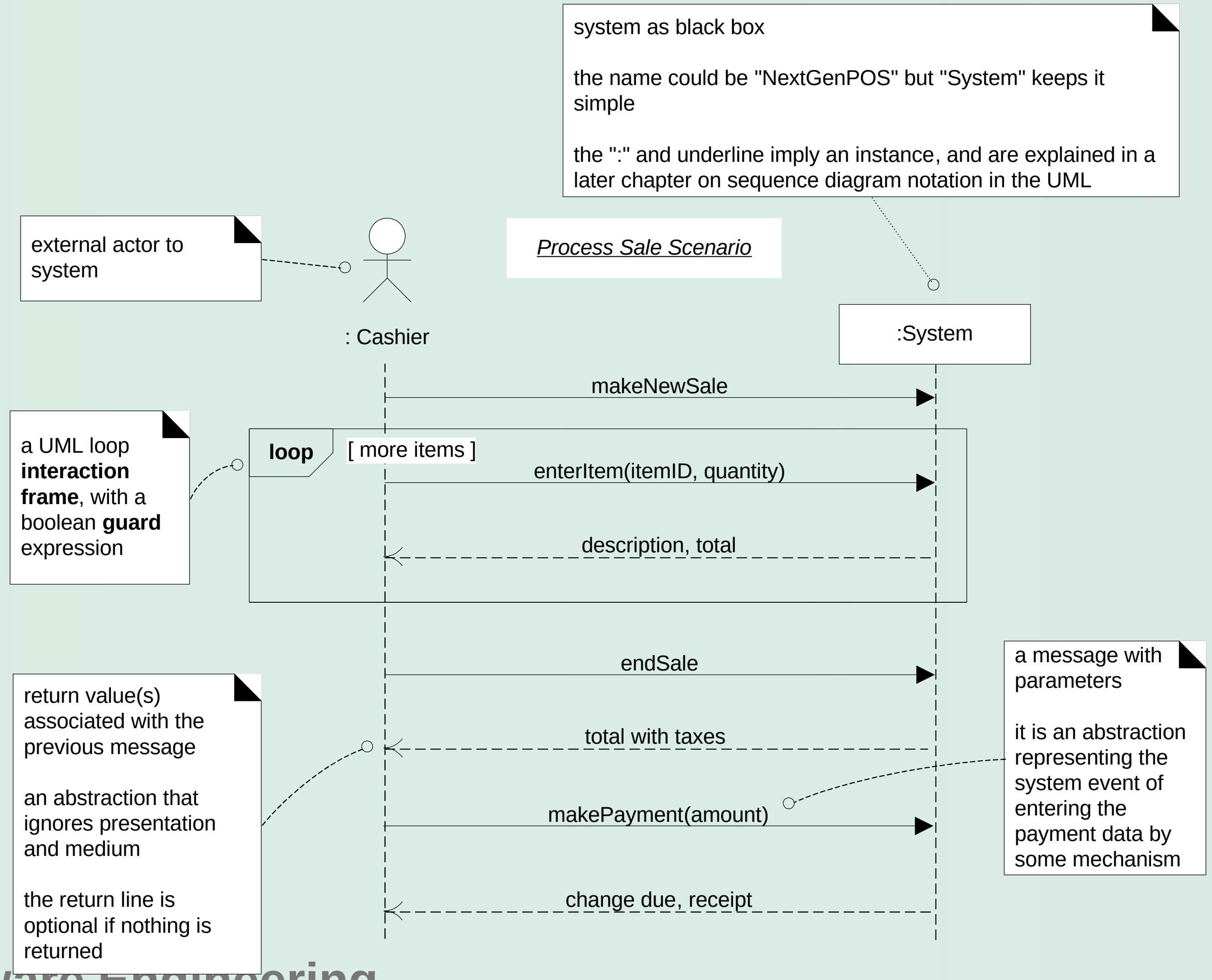
An Introduction to
Object-oriented Analysis
and Design
and Iterative Development

Part III Elaboration Iteration I – Basic[1]

Chapters

Chap 10
System Sequence Diagrams

POS SSD: a Process Sale Scenario


system as black box

the name could be "NextGenPOS" but "System" keeps it simple

the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML
external actor to system
Process Sale Scenario
: Cashier
:System
makeNewSale
loop
[more items]
enterItem(itemID, quantity)
description, total
a UML loop **interaction frame**, with a boolean **guard** expression
endSale
total with taxes
makePayment(amount)
change due, receipt
return value(s) associated with the previous message

an abstraction that ignores presentation and medium

the return line is optional if nothing is returned
a message with parameters

it is an abstraction representing the system event of entering the payment data by some mechanism

System Sequence Diagram 1

- System sequence diagram
  - a picture that shows, for one **particular scenario** of a use case, the events that **external actors** generate, their **order**, and **inter-system events**.
  - All systems are treated as a **black box**; the emphasis of the diagram is events that cross the **system boundary** from actors to systems.
  - During interaction between system and actor, an actor generates **system events** to a system, usually requesting some **system operation** to handle the event.
  - UML includes sequence diagrams as a notation that can illustrate actor interactions and the operations initiated by them.
- Guideline: Draw an SSD for a **main success** scenario of each use case, and **frequent** or **complex** alternative scenarios.

System Sequence Diagram 2

- SSDs are derived from use cases; they show one scenario.

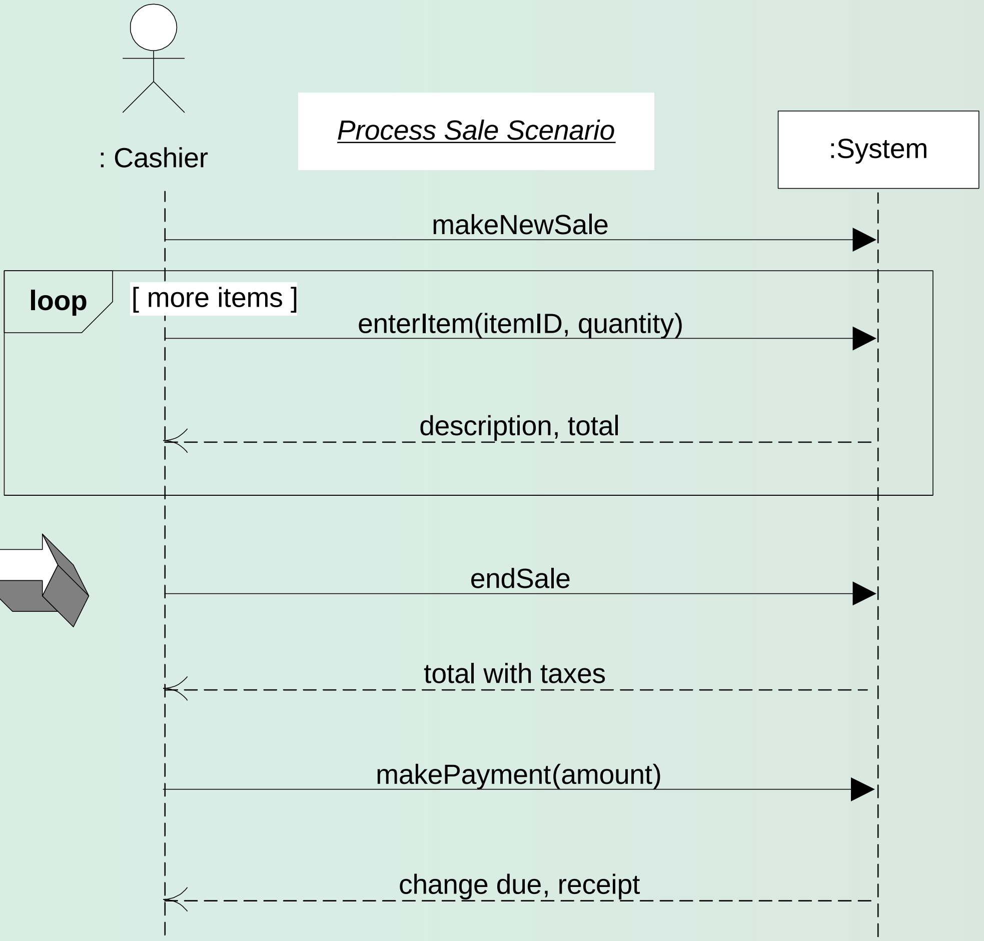
<u>Simple cash-only *Process Sale* scenario:</u>

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.

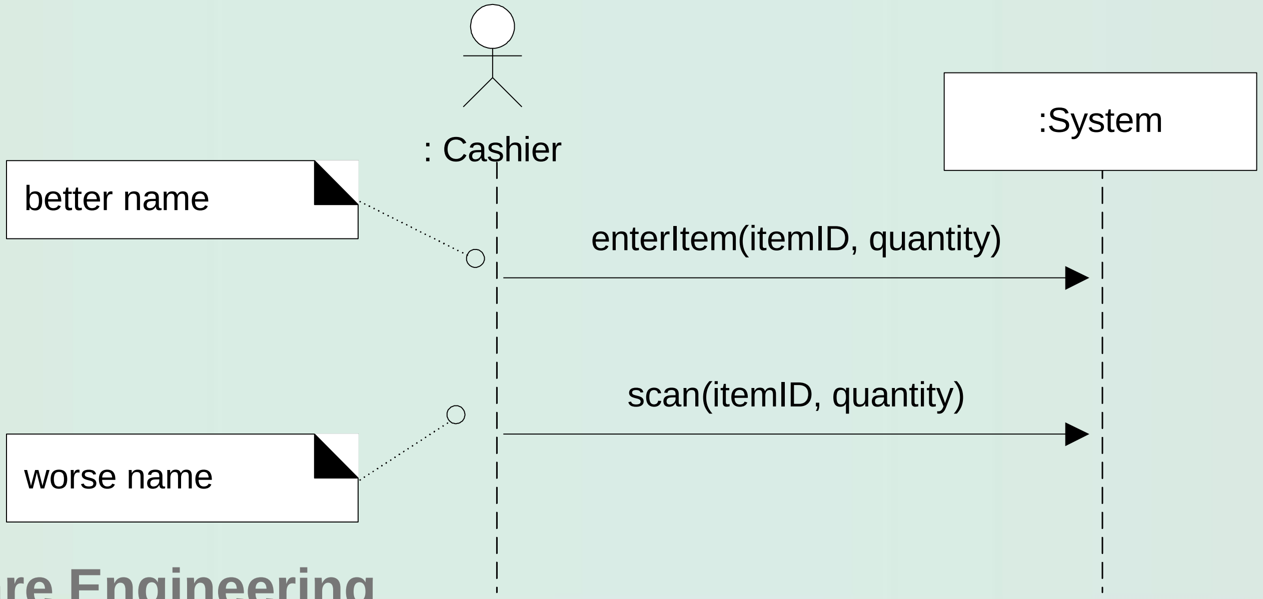
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

...



System Sequence Diagram 3

- System events should be expressed at the **abstract level of intention** rather than in terms of the physical input device.
  - "enterItem" is better than "scan" (laser scan) because it captures the abstract intent of the operation.
  - design choices about what interface is used to capture the system event (laser scanner, keyboard, voice input ..)

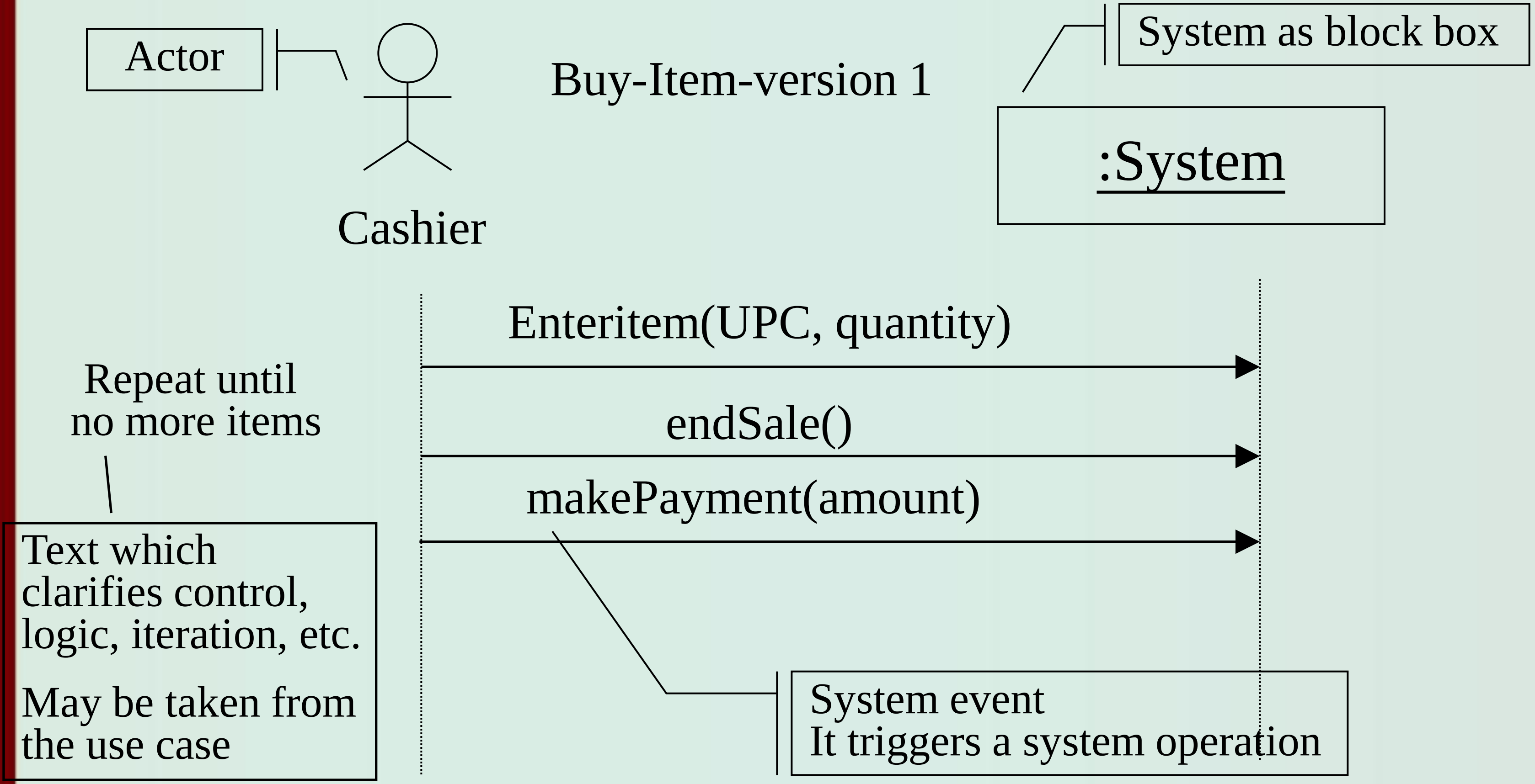


System Sequence Diagram 4

- **Guideline**
  - show details of SSD in the Glossary.
  - The elements shown in SSDs (operation name, parameters, return data) are terse.

- Iterative and Evolutionary SSDs - **UP Phases**
  - Inception phase: SSDs are not usually motivated in inception, unless you are doing rough estimating (don't expect inception estimating to be reliable) **- function points or COCOMO II.**
  - Elaboration phase: Most SSDs are created during elaboration, when it is useful to identify the details of the system events to clarify what major operations, write system operation contracts, and possibly to support estimation.

System Sequence Diagram - *Buy-Item*


Actor
Buy-Item-version 1
System as block box
:System
Cashier
Enteritem(UPC, quantity)
Repeat until no more items
endSale()
makePayment(amount)
Text which clarifies control, logic, iteration, etc.
May be taken from the use case
System event
It triggers a system operation

Chap 11
Operation Contracts

POS Operation Contract

- **Contract CO2: enterItem**
  - 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 ProductDescription, based on itemID match (association formed).

Sections of a Contract

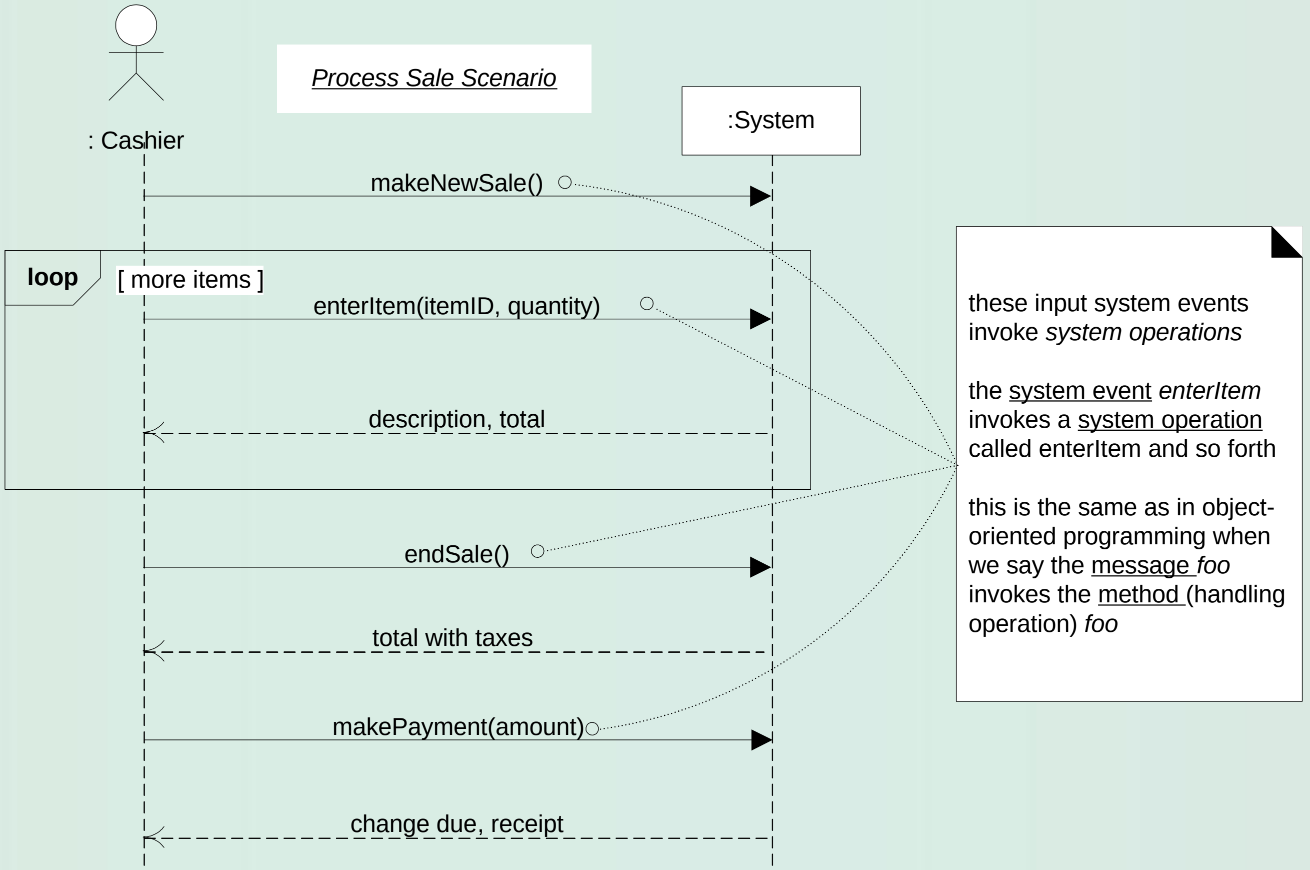
- ❑ Operation: Name of operation, and parameters
- ❑ Cross References: Use cases this operation can occur within
- ❑ Preconditions: Noteworthy assumptions about the state of the system or objects in the Domain Model before execution of the operation.
- ❑ Postconditions: The state of objects in the Domain Model after completion of the operation.

System Operations 1

- System operations
  - Define as Operation contracts
  - the system as a black box component offers in its public interface.
  - can be identified while sketching SSDs
  - SSDs show **system events or I/O messages** relative to the system.

System Operations 2

- SSD. System operations handle input system events


: Cashier
Process Sale Scenario
:System
makeNewSale()
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale()
total with taxes
makePayment(amount)
change due, receipt
these input system events invoke system operations
the system event enterItem invokes a system operation called enterItem and so forth
this is the same as in object-oriented programming when we say the message foo invokes the method (handling operation) foo

Postconditions 1

- Postconditions
  - describe changes in the state of objects in the domain model.
  - Domain model state changes include instances created, associations formed or broken, and attributes changed.
  - Postconditions are not actions to be performed.
- Postconditions fall into these categories:
  - Instance creation and deletion.
  - Attribute change of value.
  - Associations (UML links) formed and broken.
- Postconditions Related to the Domain Model
  - What instances can be created; What associations can be formed in the Domain Model.
- Motivation: Why Postconditions?
  - Postconditions support **fine-grained detail and precision in declaring what the outcome of the operation** must be.

Postconditions 2

- Guideline: To Write a Postcondition
  - to emphasize state changes that arose from an operation, not an action to happen.
  - (better) A SalesLineItem was created.
  - (worse) Create a SalesLineItem, or, A SalesLineItem is created.
- Analogy: The Spirit of Postconditions: The Stage and Curtain
  - The system and its objects are presented on a theatre stage.
    - 1.Before the operation, take a picture of the stage.
    - 2.Close the curtains on the stage, and apply the system operation
    - 3.Open the curtains and take a second picture.
    - 4.Compare the before and after pictures, and express as postconditions the changes in the state of the stage
- Guideline: How Complete Should Postconditions Be?
  - generating a complete and detailed set of postconditions for all system operations is not likely or necessary.

Example: enterItem Postconditions

- Postconditions of the enterItem system operation.
  - Instance Creation and Deletion
    - After the itemID and quantity of an item have been entered, what new object should have been created? A SalesLineItem. Thus:
    - A SalesLineItem instance sli was created (instance creation).
  - Attribute Modification
    - After the itemID and quantity of an item have been entered by the cashier, what attributes of new or existing objects should have been modified? The quantity of the SalesLineItem should have become equal to the quantity parameter. Thus:
    - sli.quantity became quantity (attribute modification).
  - Associations Formed and Broken
    - After the itemID and quantity of an item have been entered by the cashier, what associations between new or existing objects should have been formed or broken?
    - The new SalesLineItem should have been related to its Sale, and related to its ProductDescription. Thus:
    - sli was associated with the current Sale (association formed).
    - sli was associated with a ProductDescription, based on itemID match (association formed).

Guideline: Update the Domain Model

- It's common during the creation of the contracts to discover the need to record new conceptual classes, attributes, or associations in the domain model.
- Enhance it as you make new discoveries while thinking through operation contracts.

Guideline: When Are Contracts Useful

- Consider an airline reservation system
  - the system operation addNewReservation: the complexity is very high regarding all the domain objects that must be changed, created, and associated.
  - These fine-grained details can be written up in the use case, but it will make it extremely detailed (e.g., noting each attribute in all the objects that must change).
  - the postcondition offers and encourages a very precise, analytical language.
- If developers can comfortably understand what to do without them, then avoid writing contracts.

Guideline: Create and Write Contracts

- Identify system operations from the SSDs.
- For system operations that are complex and perhaps subtle in their results, or which are not clear in the use case, construct a contract.
- To describe the postconditions, use the following categories:
  - instance creation and deletion
  - attribute modification
  - associations formed and broken
- **Writing Contracts**
  - write the postconditions in a declarative, passive past form (was ..) to emphasize the observation of a change rather than how it is going to be achieved.
  - (better) A SalesLineItem was created.
  - (worse) Create a SalesLineItem.
- To establish an association between existing objects or those newly created.
  - After the **enterItem** operation is complete, the newly created instance was associated with Sale; thus:
  - The SalesLineItem was associated with the Sale.
- The most common problem is forgetting to include the forming of associations. Particularly when new instances are created.

Example: NextGen POS Contracts 1

- **System Operations of the Process Sale Use Case**
  - **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 a Register (association formed).
    - Attributes of s were initialized.
- Keep it as light as possible, and avoid all artifacts unless they really add value.
- **Contract CO2: enterItem**
  - 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 ProductDescription, based on itemID match (association formed).

Example: NextGen POS Contracts 2

- Contract CO3: endSale
    - Operation: endSale()
    - Cross References: Use Cases: Process Sale
    - Preconditions: There is a sale underway.
    - Postconditions:
        - Sale.isComplete became true (attribute modification).
- Contract CO4: makePayment
    - Operation: makePayment(amount: Money)
    - Cross References: Use Cases: Process Sale
    - Preconditions:There is a sale underway.
    - 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)

Process: Operation Contracts Within the UP

- In the UML, operations exists at many levels, from System down to fine-grained classes, such as Sale.
  - Operation contracts for the System level are part of the Use-Case Model.
- Inception phase
  - Contracts are not motivated during inception, they are too detailed.
- Elaboration phase
  - If used at all, most contracts will be written during elaboration, when most use cases are written. Only write contracts for the most complex and subtle system operations.