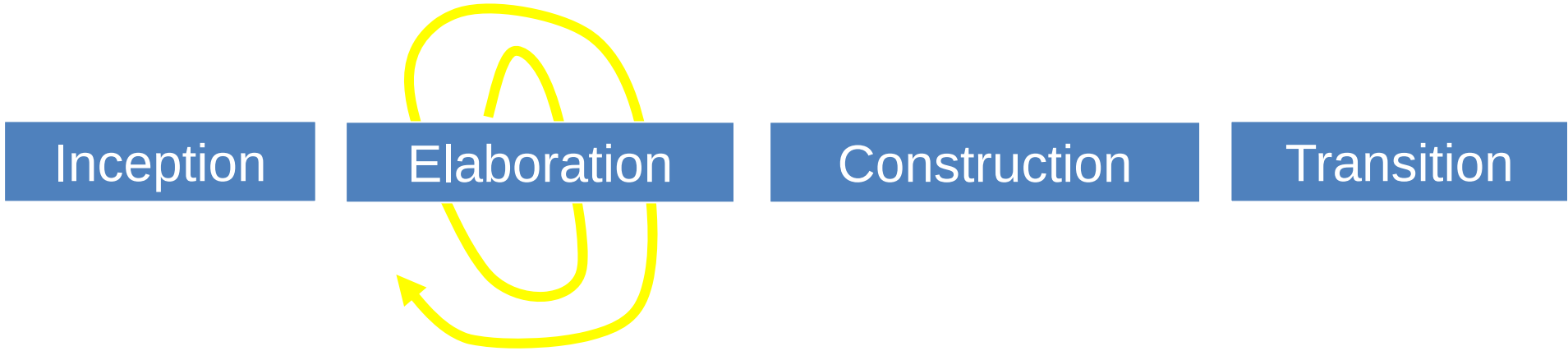
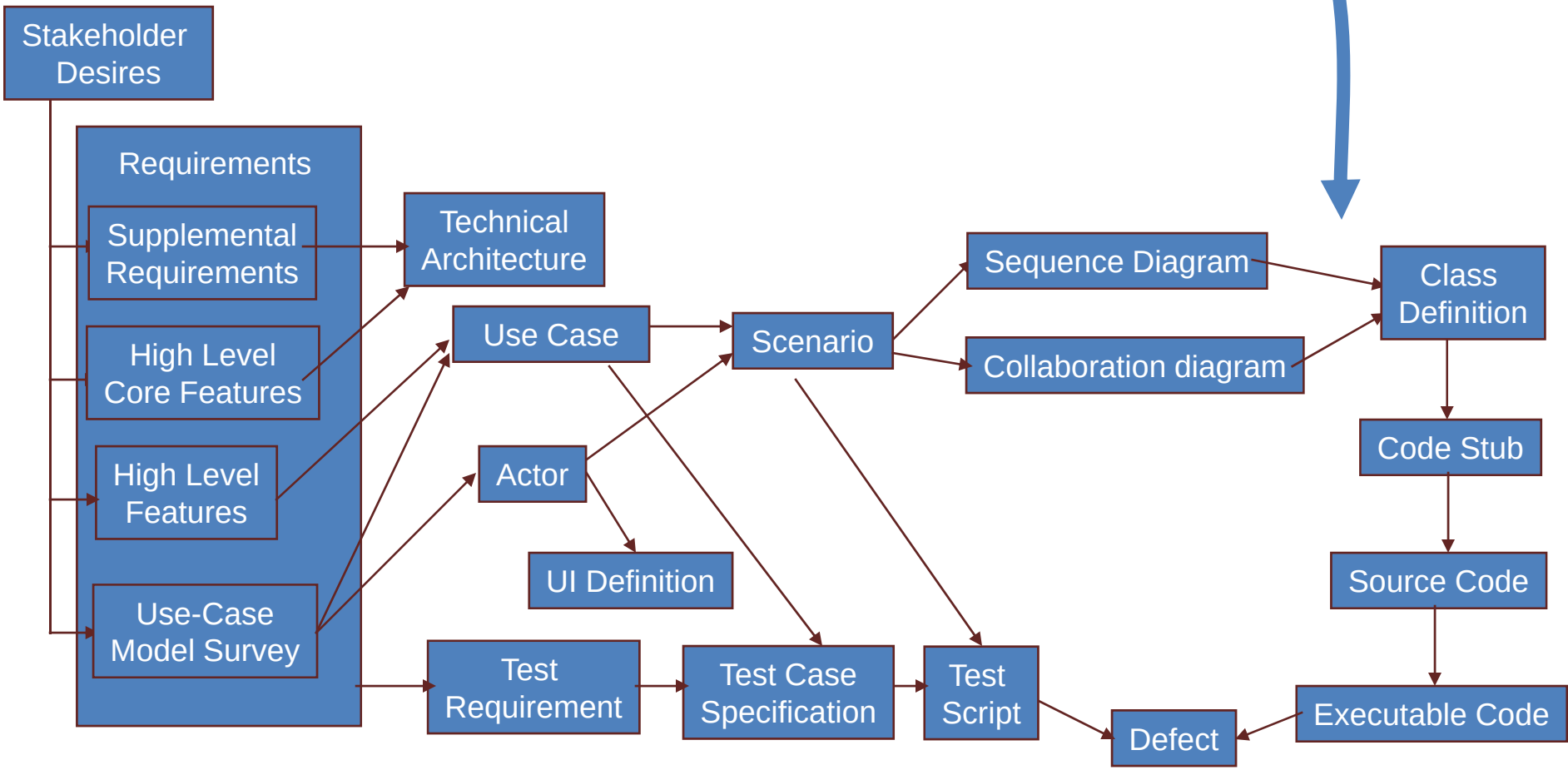


Operation Contracts

COMP 3831

Larman: Chapter 11

Where are we?


Stakeholder Desires
Requirements
Supplemental Requirements
High Level Core Features
High Level Features
Use-Case Model Survey
Technical Architecture
Use Case
Actor
UI Definition
Test Requirement
Test Case Specification
Test Script
Scenario
Sequence Diagram
Collaboration diagram
Class Definition
Code Stub
Source Code
Executable Code
Defect

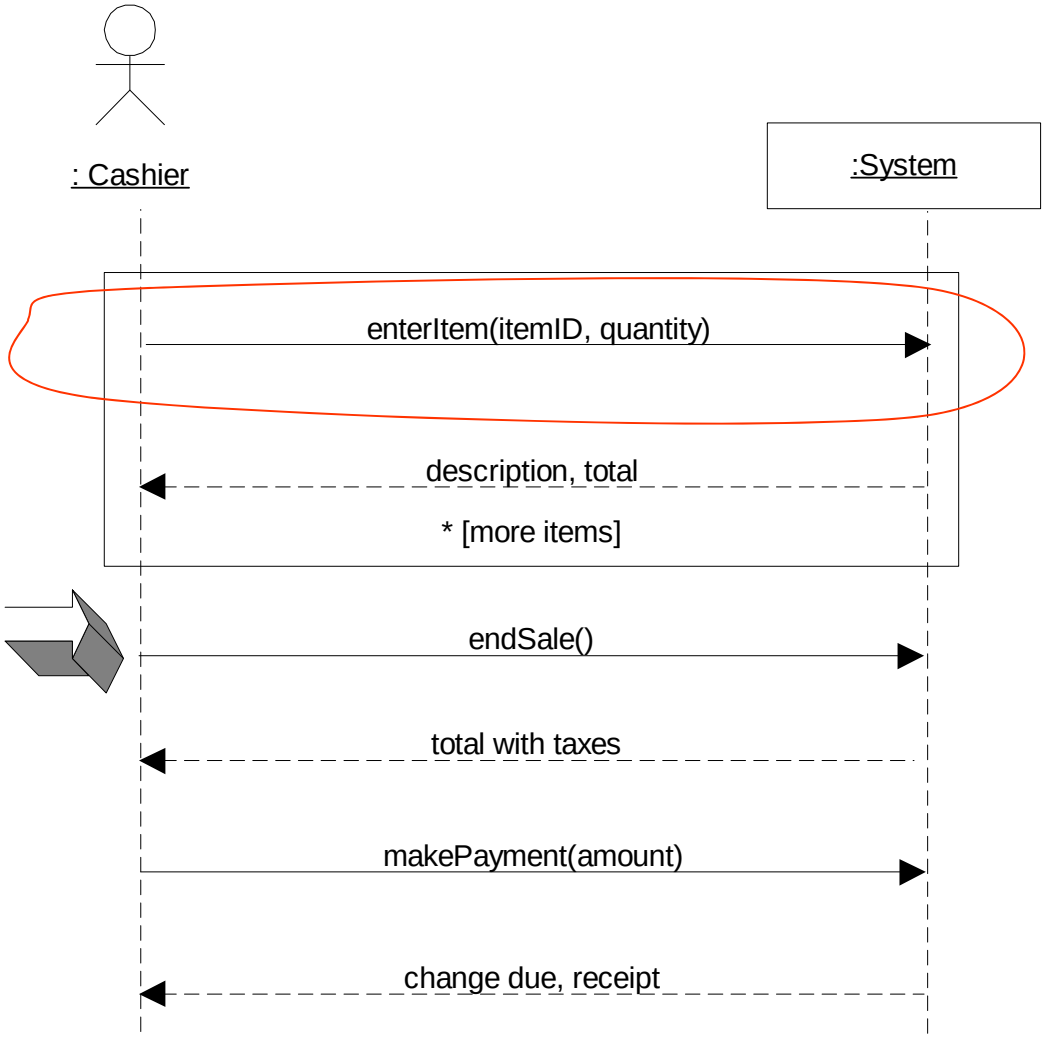
Operation Contracts

- When use-cases are insufficient for describing system behavior, contracts:
  - Help define detailed system behavior
  - Describe outcome of executing operations in terms of state changes to domain objects, after an operation has executed
- Contracts may be defined as system operations
  - System operations: operations that the system, as a black box , offers in its public interface to handle incoming system events

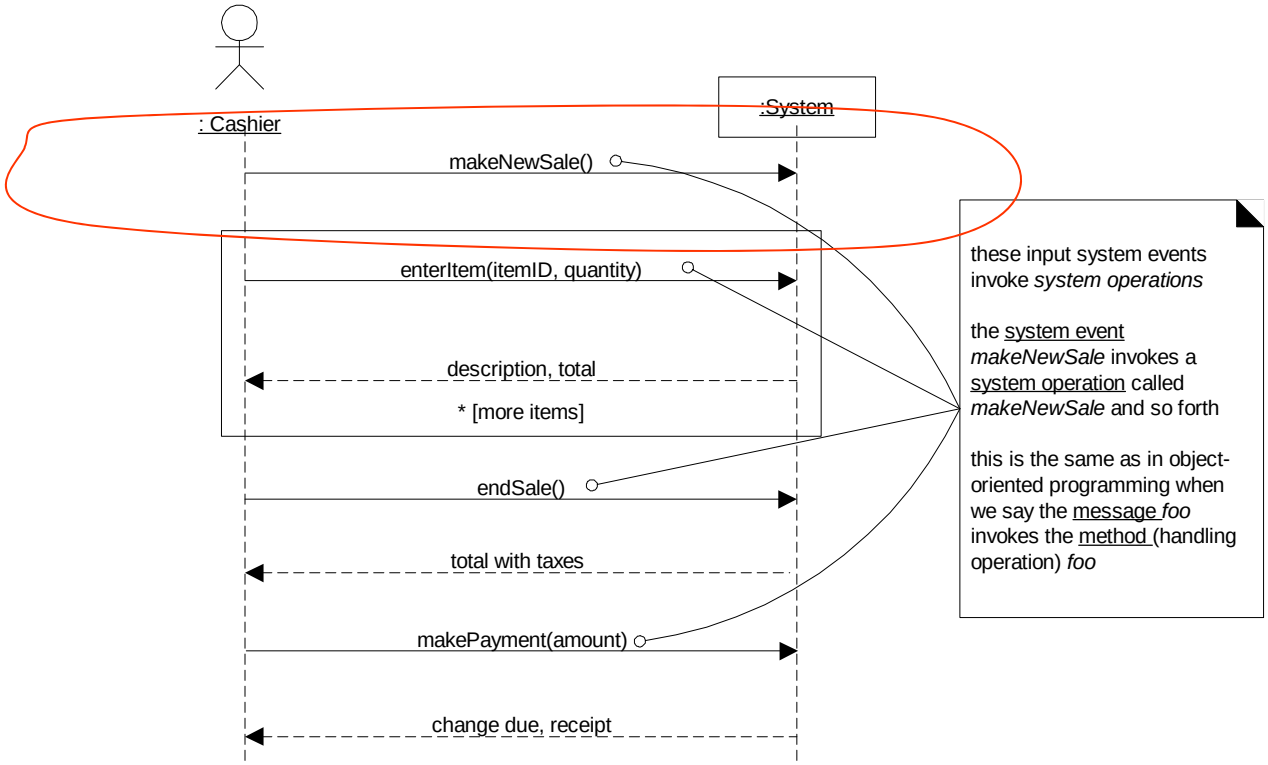
Sequence diagram derived from use-case

Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier enters item identifier and quantity, if greater than one.
3. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 2-3 until indicates done.
4. System presents total with taxes calculated.
5. Cashier tells Customer the total, and asks for payment.
6. Customer pays with cash.
7. Cashier enters cash tendered.
8. System records payment and presents change due.
7. System logs the completed sale, but does not interact with external systems.
8. System presents receipt.
9. Customer leaves with receipt and goods.



System operations handling input events



Example Contract: *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 *ProductSpecification*, based on *itemID* match (association formed)

Contract sections

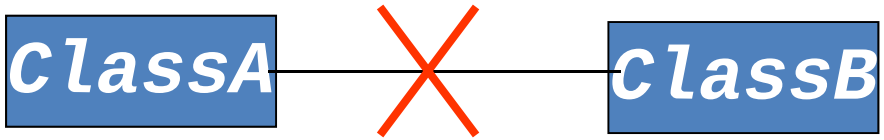
Operation:	Name of operation and parameters
Cross References:	(optional) Use cases this operation can occur within
Preconditions:	Noteworthy *assumptions* about the state of the system or objects in the Domain Model before execution of the operation. These will not be tested within the logic of this operation, are assumed to be true, and are non-trivial assumptions the reader should know were made.
Postconditions:	The state of objects in the Domain Model after completion of the operation. Discussed in detail later ….

Postconditions

- Changes in the state of objects in the Domain Model, which include:
  1. Instances created and deleted
  2. Associations (UML links) formed and broken
  3. Attributes modification
- Declarations about Domain Model objects that are true when the operation has finished – *after the smoke has cleared*.
- Expressed in a declarative fashion …

Examples of broken associations

1. When a loan is paid off …
2. When someone cancels their membership in a club
3. A country's law stipulates that seven years after a person has declared bankruptcy, all records of their bankruptcy declaration must be destroyed.



Postcondition pitfalls

- Describes state changes required of a system and not how they are going to be achieved.
- Focus analytically on what must happen, and not how it is to be accomplished
- Unnecessary and undesirable to be overly verbose and detailed
- Express postconditions in the past tense:
  - <Better> ➔ A SaleLineItem was created
  - <Worse> ➔ Create a SalesLineItem

Contract evolution

- Start with an initial best guess. Incomplete is better than deferring investigation
- Generating a complete detailed set of postconditions for system operation in not necessary during initial requirements work
- Fine details (or perhaps larger ones) will be discovered later during design work
- It is all about iterative development: discoveries during previous iteration enhances investigation and analysis of following iteration

Contracts & Domain Model updates

- It is common that during creation of contracts you discover need to record new conceptual classes, attributes, or associations in the Domain Model
- Enhance your Domain Model as you make new discoveries while thinking through contracts.

Contracts vs. Use Cases

- Contracts may not be needed if use cases provide most or all of the detail necessary for design
- Contracts, on the other hand, are handy where details and complexity of required state changes are awkward to capture in use cases

Contract pitfalls

- Do not make contracts for every system operation of every use case
- If you find yourself creating too many contracts, then project may be suffering from the following ailments:
  1. Use cases are poorly done
  2. Not enough ongoing collaboration or access to a subject matter expert
  3. Your team is doing too much unnecessary documentation

Contract pitfalls (continued …)

- Remember to establish a memory between existing objects and those newly created by defining the forming of an association

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)

It is necessary to define the forming of a new association

Contracts: guidelines

To write contracts:

1. Identify system operations from the SSDs (system sequence diagrams)
2. Construct contracts for operations that are complex, subtle in their results, or not clear in the use case
3. Use following categories to describe postconditions:
   i. Instance creation and deletion
   ii. Attribute modification
   iii. Associations formed and broken

POS example

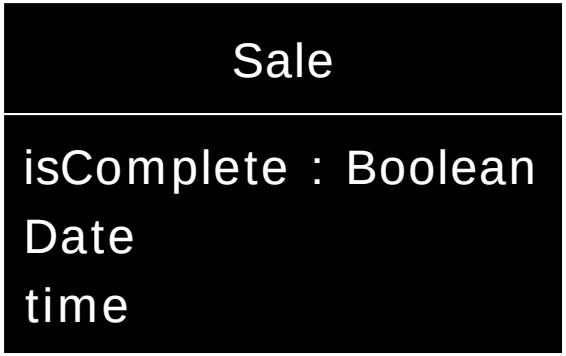
Contract C01: makeNewSale	
Operation:	makeNewSale()
Cross References:	Use Cases: Process Sale
Preconditions:	none
Postconditions:	✹ A *Sale* instance *s* was created (instance creation) ✹ *s* was associated with the *Register* (association formed) ✹ Attributes of *s* were initialized

Contract C02: 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 *ProductSpecification*, based on *itemID* match (association formed)

POS example

Contract C03: endSale	
Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway
Postconditions:	✸ *Sale.isComplete* became *true* (attribute modification)

Data item suggested in contract that is not yet represented in domain model



Operations in the eyes of UML

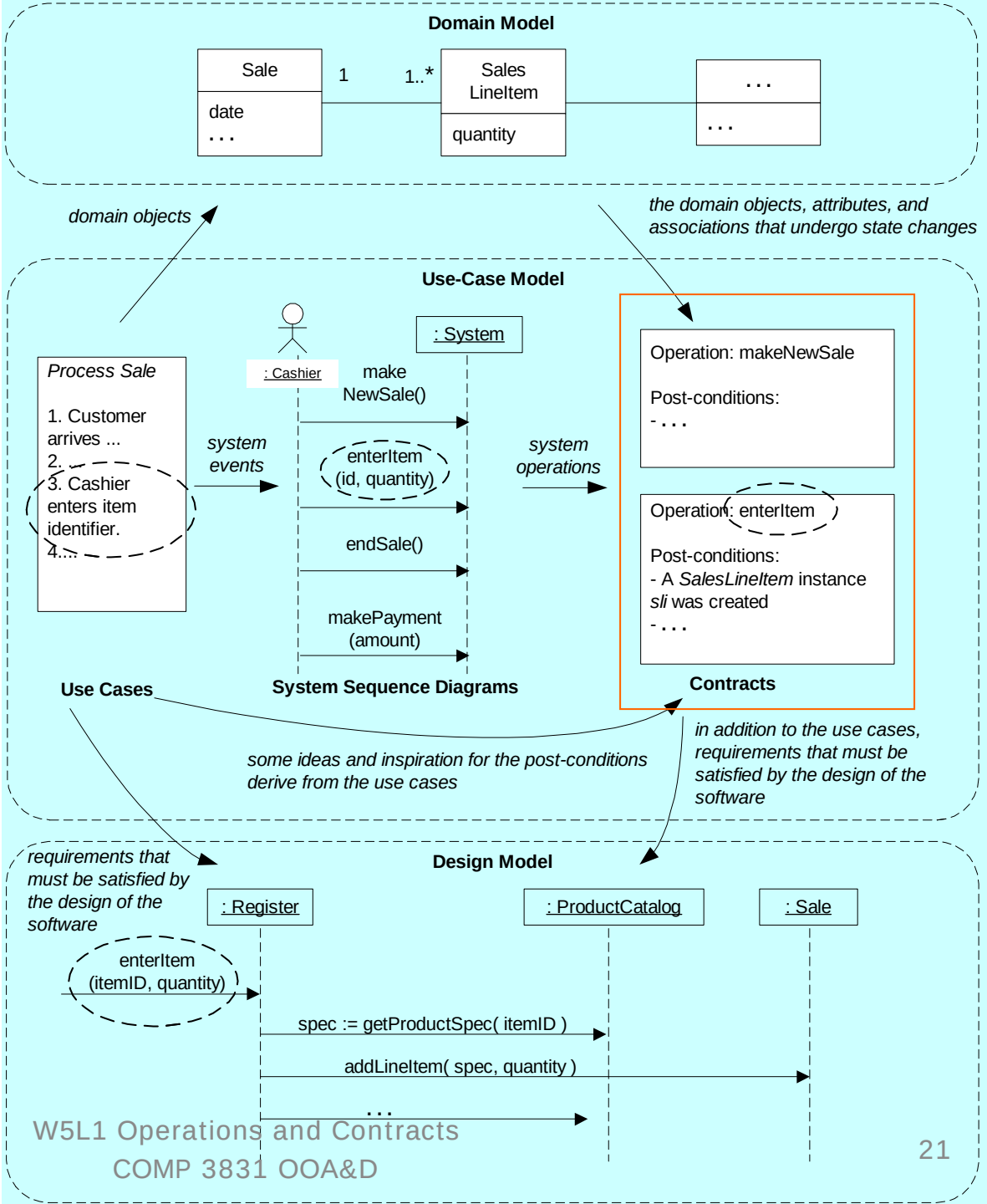
- An operation is a specification of a transformation or query that an object may be called to execute.
  - Example: elements of an interface are operations
- An operation is an abstraction, not an implementation.
  - By contrast, a method (in the UML) is an implementation of an operation
- A UML operation has a signature (name & parameters) and an operation specification (postconditions)

UP Phases and Contracts

Inception – contracts are not motivated during inception because they are too detailed

Elaboration – If used at all, most contracts will be written during elaboration

Contract relationship to other artifacts



Questions

Inception | Elaboration | Construction | Transition

Requirements to Design - Iteratively

COMP 3831

Larman: Chapter 12

Requirements and OOA

- Focused on learning to *do the right thing*
  - understand the goals and related rules and constraints
- Design stresses *do the thing right*
- In iterative development, a transition from requirements to design will occur in each iteration
- By the end of elaboration, ~80% of the requirements are reliably defined in detail.

Object Design

- It is here that a logical O-O solution is developed.
- Interaction Diagrams are a key artifact.
  - illustrate how objects collaborate to fulfill requirements
- Class Diagrams may also be used
- These artifacts help to make up the Design Model

Object Design Skill Vs. UML Notation Skill

- What is important is knowing how to think and design in objects
- This is quite a different skill, and much more important skill than knowing UML diagramming notation
- Object design requires knowledge of:
  - principles of responsibility assignment
  - design patterns

Questions and Conclusions