Software Modeling & Analysis

OOPT (Object Oriented Process with Trace)

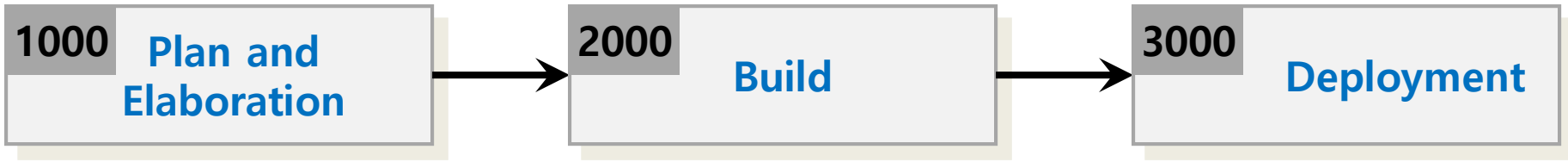
Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr

Dependable Software Laboratory

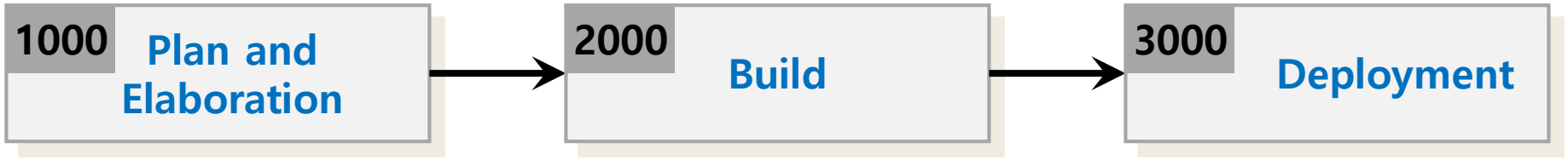
What is OOPT?

- OOPT (Object Oriented Process with Trace)
  - A software process based on RUP
  - Revision of OSP (by Tailored to SE classes in universities)

- Characteristics of OOPT
- 3 Stages
  1. Iterative : Multiple development cycles
  2. Incremental : System grows incrementally as each cycle is completed
  3. Architecture : Stage > Cycle > Phase > Activity



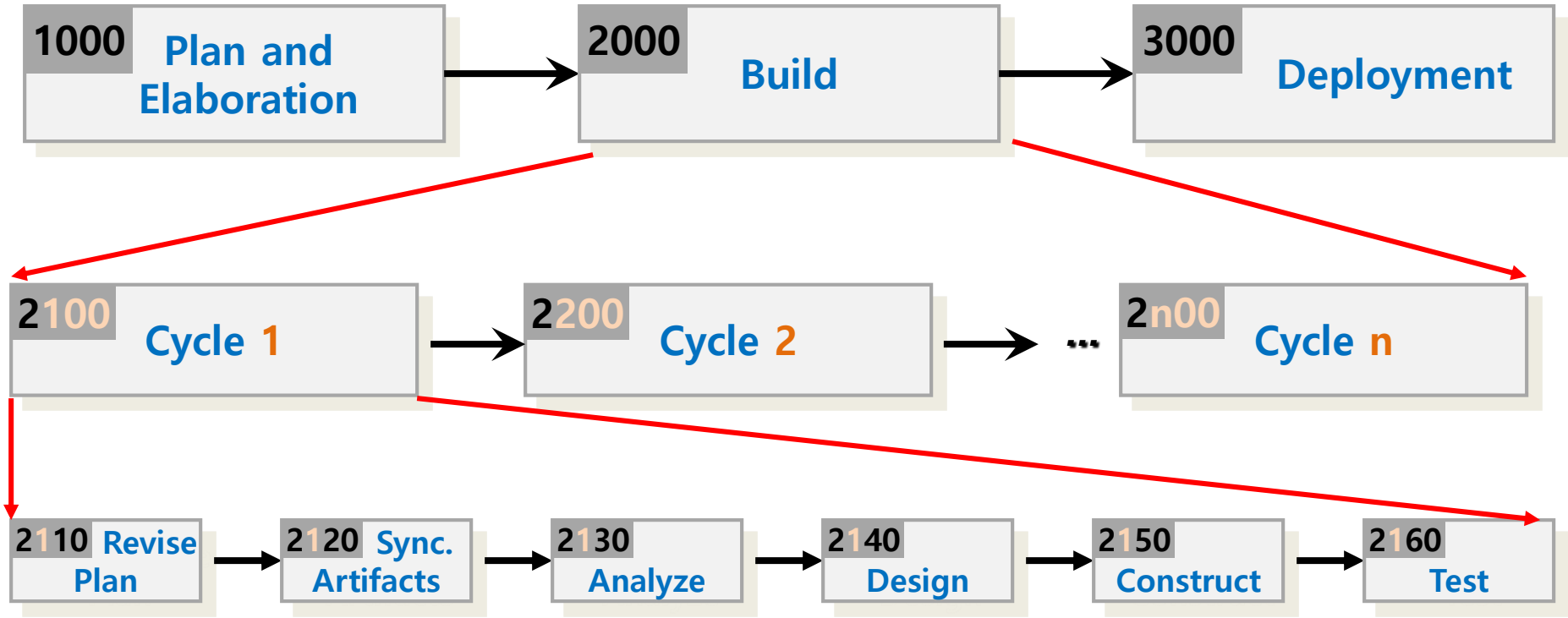
1. 3 Stages



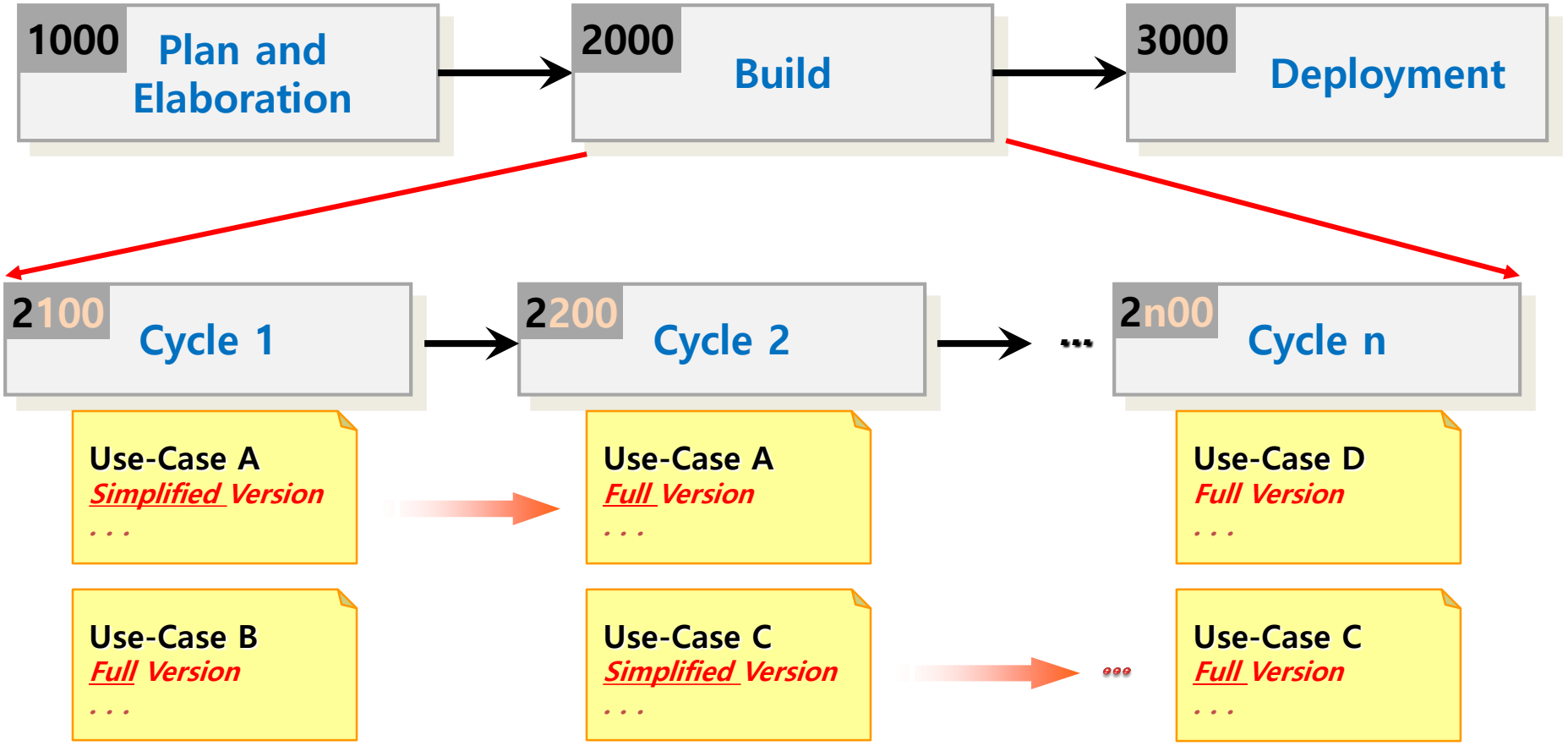
- Stage 1000 : Plan and Elaboration
  - Planning, defining requirements, building prototyping, etc
  - Corresponding to Inception/Elaboration phases in the RUP
- Stage 2000 : Build
  - Construction of the system
  - Corresponding to Construct phase in the RUP
- Stage 3000 : Deployment
  - Implementation of the system into use
  - Corresponding to Transition phase in the RUP

2. Iterative Development

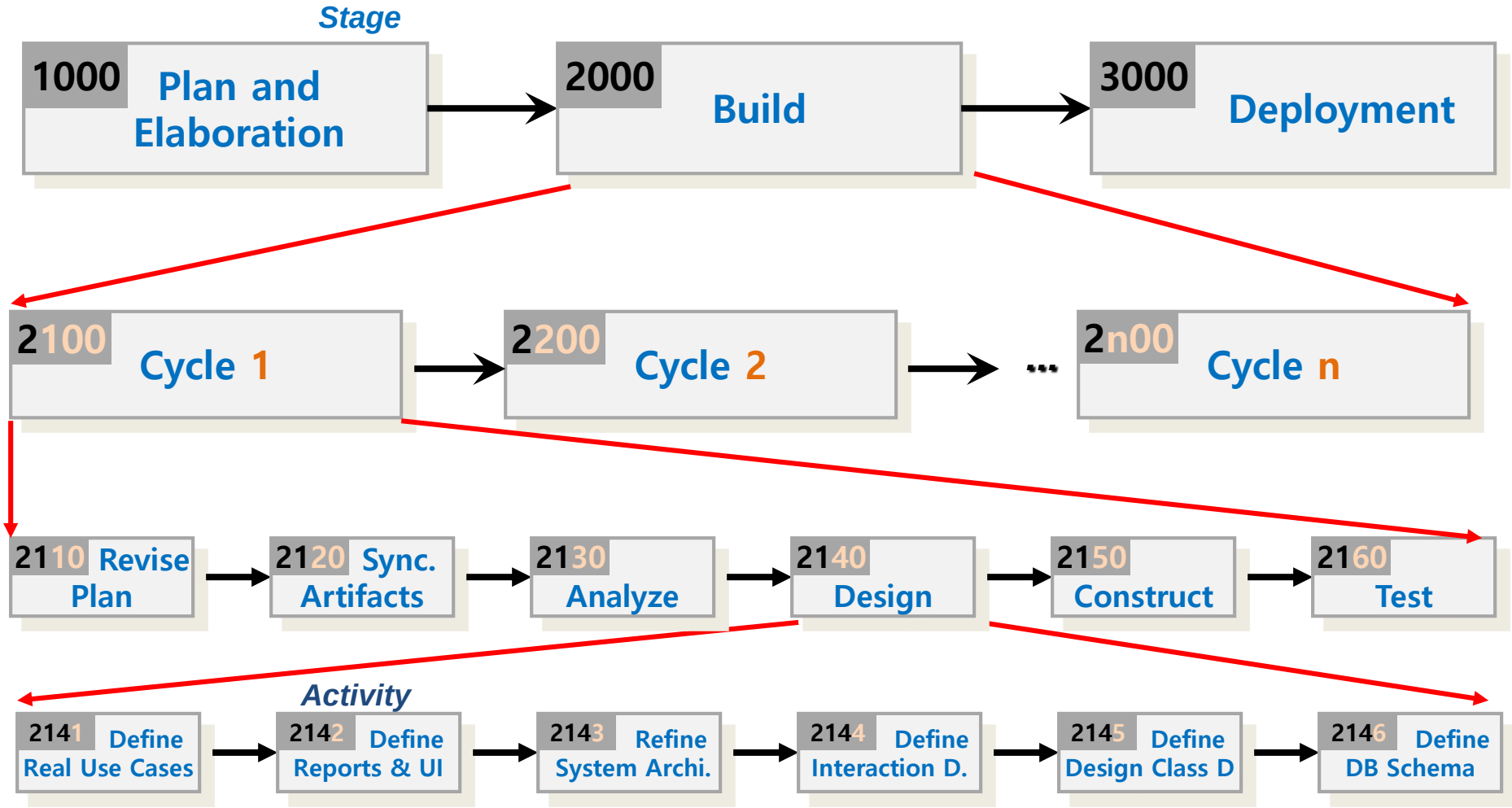
- Multiple iterations in the Build stage
- Each iteration took about 2 to 8 weeks



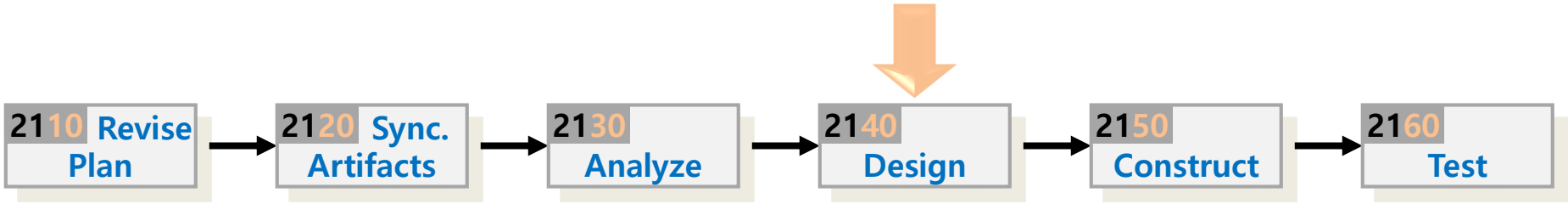
3. Incremental Development



4. Architecture of OSP



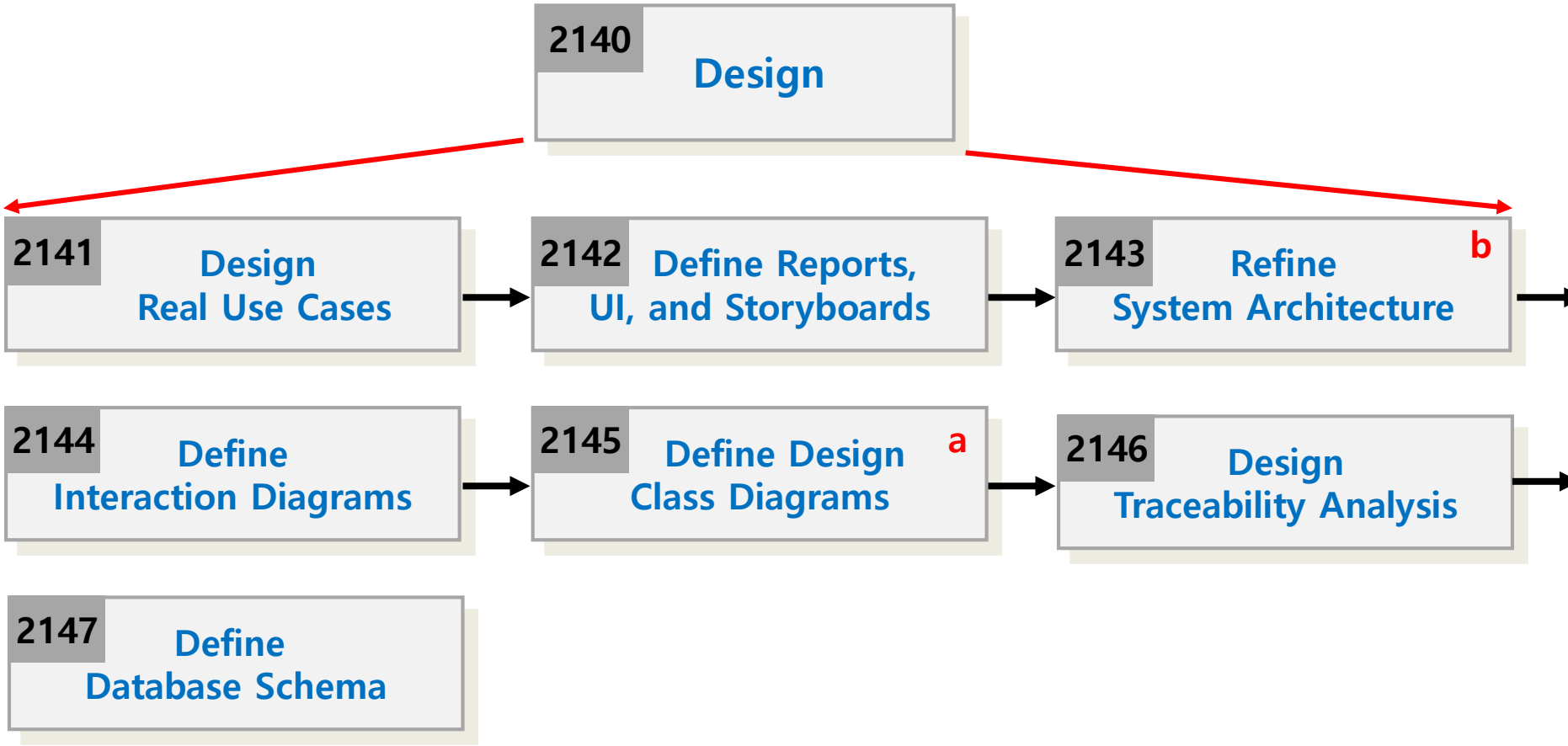
Phase 2040.
Design



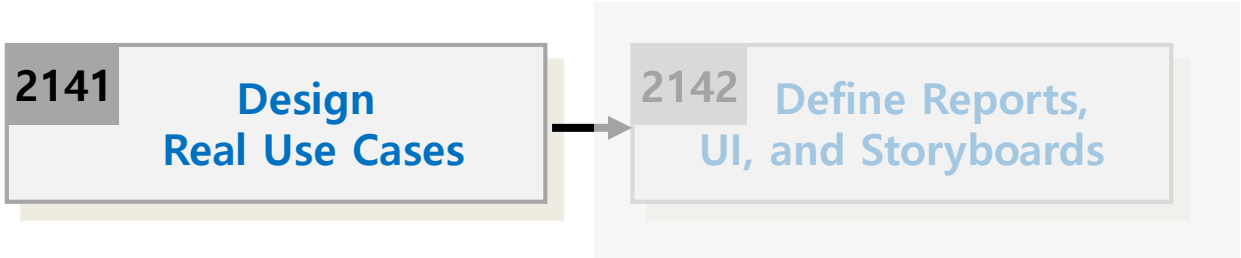
Phase 2040. Design

- Phase 2040 Activities

a. In parallel with interaction diagrams
b. Varied order



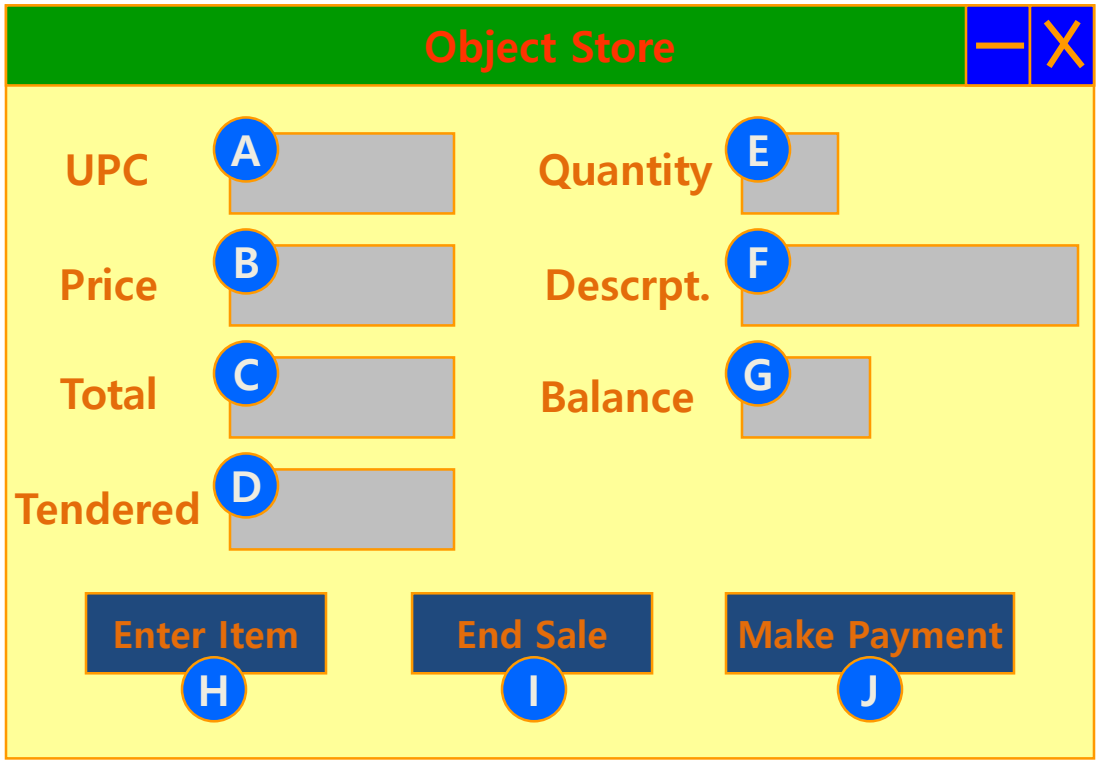
Activity 2041. Design Real Use Cases



- Description
  - It describes real/actual design of the use case in terms of concrete input and output technology and its overall implementation.
  - If a graphical user interface is involved, the real use case will include diagrams of the GUI and discussion of the low-level interactions with interface widgets.
  - Input : Essential Use Case Descriptions
  - Output : Real Use Case Descriptions

Activity 2041. Design Real Use Cases

- Steps
  1. Select each use case from essential use cases
  2. Add user interface widgets into the expanded format, and concrete implementation details into the typical courses of events



Window-1

Activity 2041. Design Real Use Cases

Use Case	Buy Items – Version 1 (Cash only)
Actor	Customer, Cashier
Purpose	Capture a sale and its cash payment
Overview	A Customer arrives at a checkout with items to purchase. The Cashier records the items and collects cash payment, which may be authorized. On completion, the Customer leaves with the items.
Type	Primary and Real
Cross Reference	Functions: R1.1, R1.2, R1.3, R1.7, R1.9, R2.1 Use Cases: Log In use case
Pre-Requisites	N/A
UI Widgets	Window-1
Typical Courses of Events	(A) : Actor, (S) : System 1. (A) This use case begins when a customer arrives at the POST to checkout with items to purchase. 2. (A) For each item, the Cashier types an UPC in A of Window-1. If there is more than one of an item, the quantity may optionally be entered in E. They press B after each item entry. (E1) 3. (S) Adds the item information to the running sales transaction. The description and price of the current item are displayed in B and F of Window1. 4. (A) The Cashier tells the customer the total.
Alternative Courses of Events	…
Exceptional Courses of Events	E1: If an invalid UPC is entered, indicate an error.

Activity 2042.
Define Reports, UI, and Storyboards

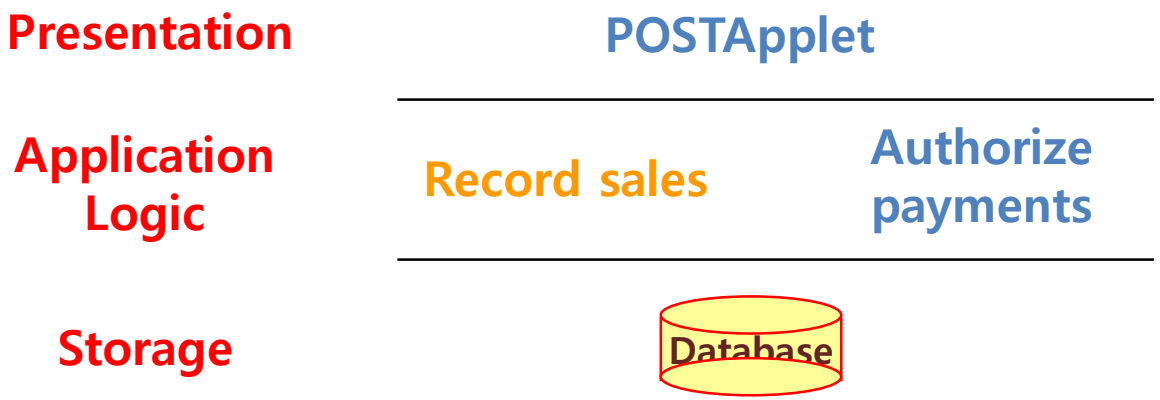
- Description
  - Design UI storyboard and UI components.
  - Input : Requirements Specification, Real Use Case Descriptions
  - Output : UI Storyboard, UI Component Design Specification

Activity 2043. Refine System Architecture

- Description
  - Refine draft system architecture developed in the plan stage
  - Input : Draft System Architecture
  - Output : A package diagram, a deployment diagram
  - Standards Applied
    - UML's Package Diagram
    - UML's Deployment Diagram

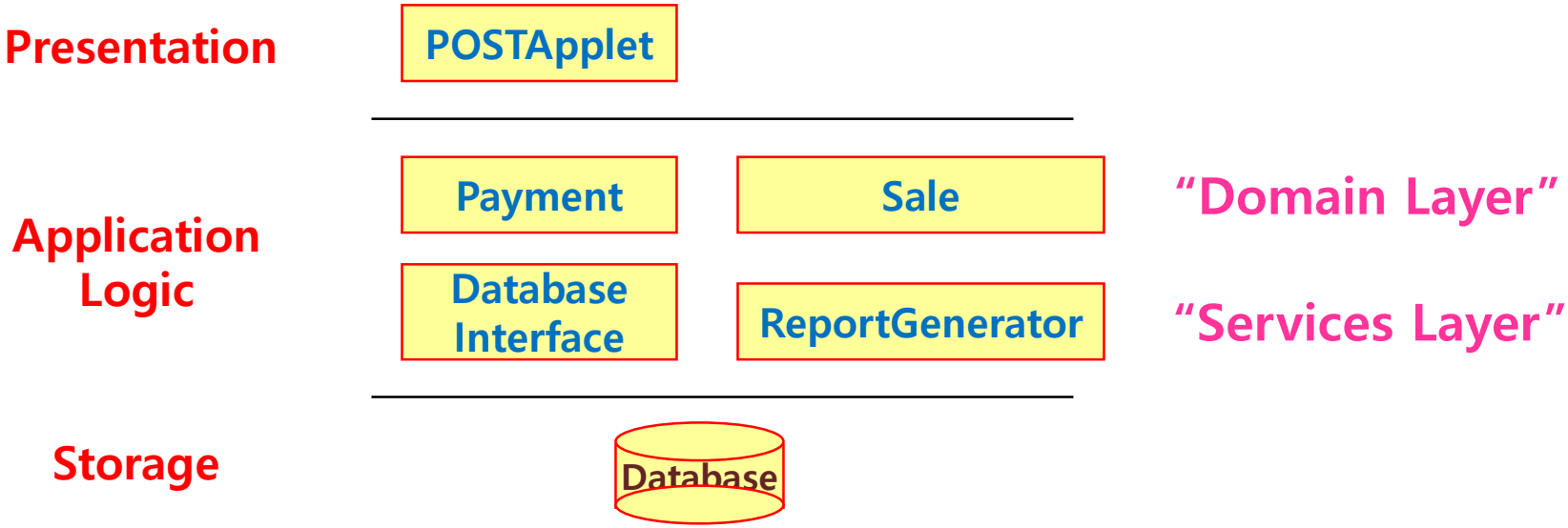
Activity 2043. Refine System Architecture

- Steps (1~3: Deployment diagram , 4~7: Package diagram)
  1. Define a 3-tier layered system architecture
     - Presentation Layer : Windows, Reports, and so on
     - Application Logic Layer : Tasks and rules that govern the process
     - Storage Layer : Persistent storage mechanism



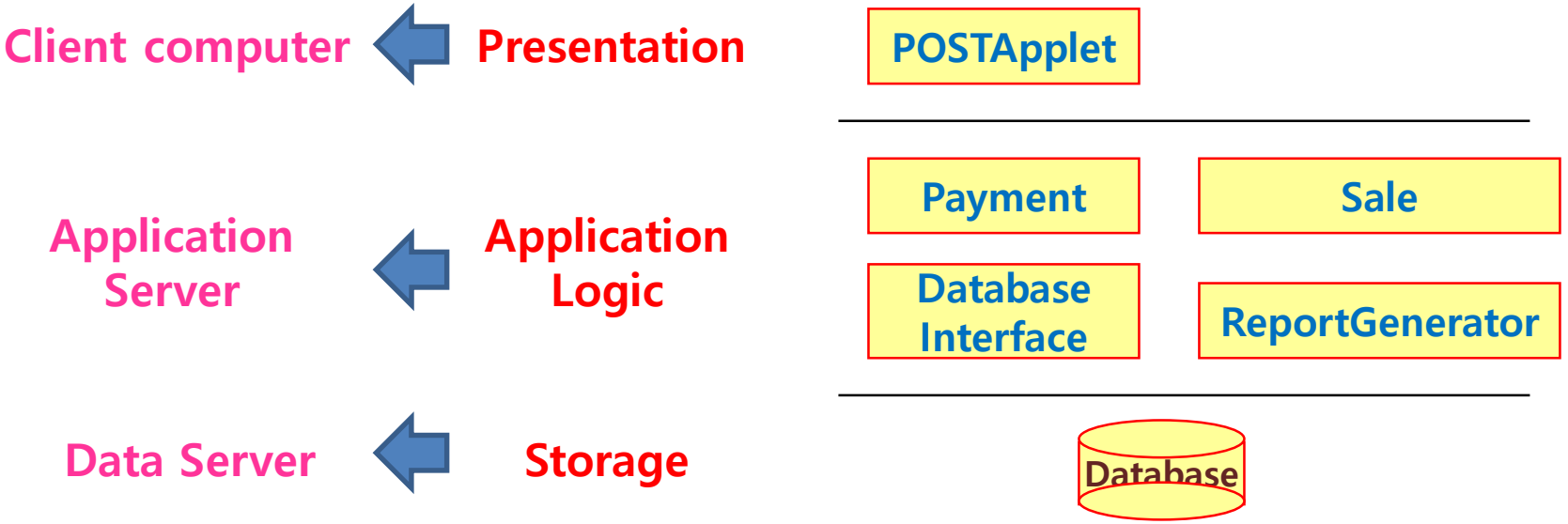
Activity 2043. Refine System Architecture

2. Decompose the application logic tier into finer layers
   - Domain object layer
     - Classes representing domain concepts
   - Service layer
     - Service objects for functions such as database interaction, reporting, communications, security, and so on



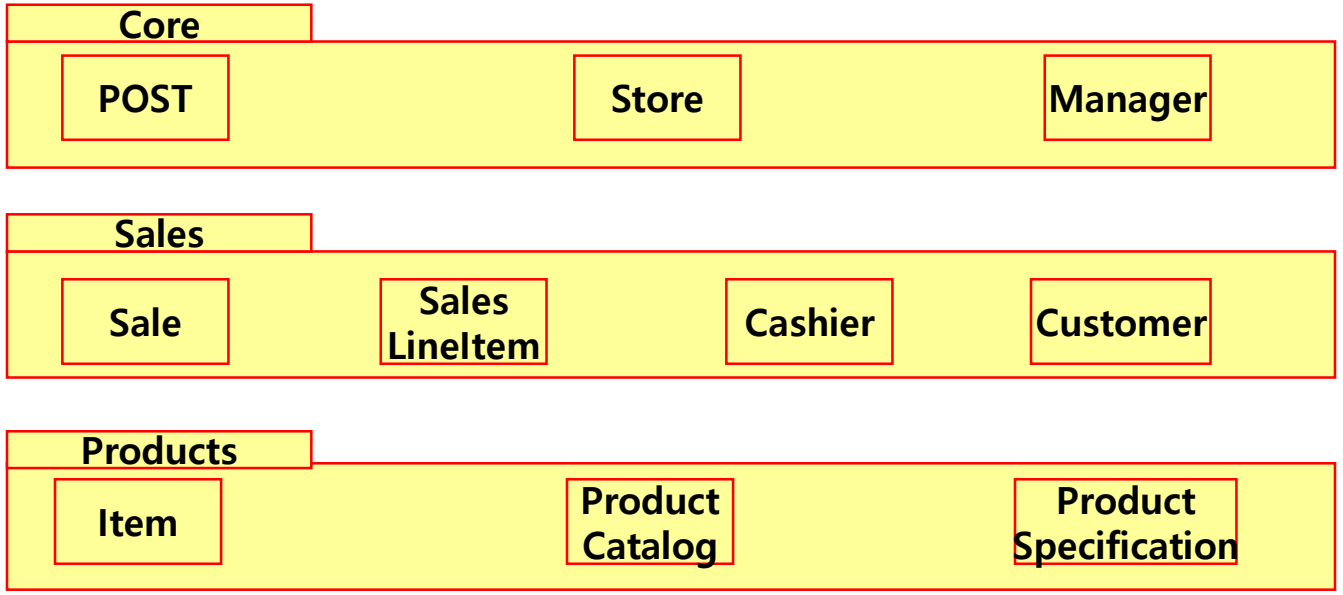
Activity 2043. Refine System Architecture

3. Assign each tier into different physical computing nodes, and/or different processes



Activity 2043. Refine System Architecture

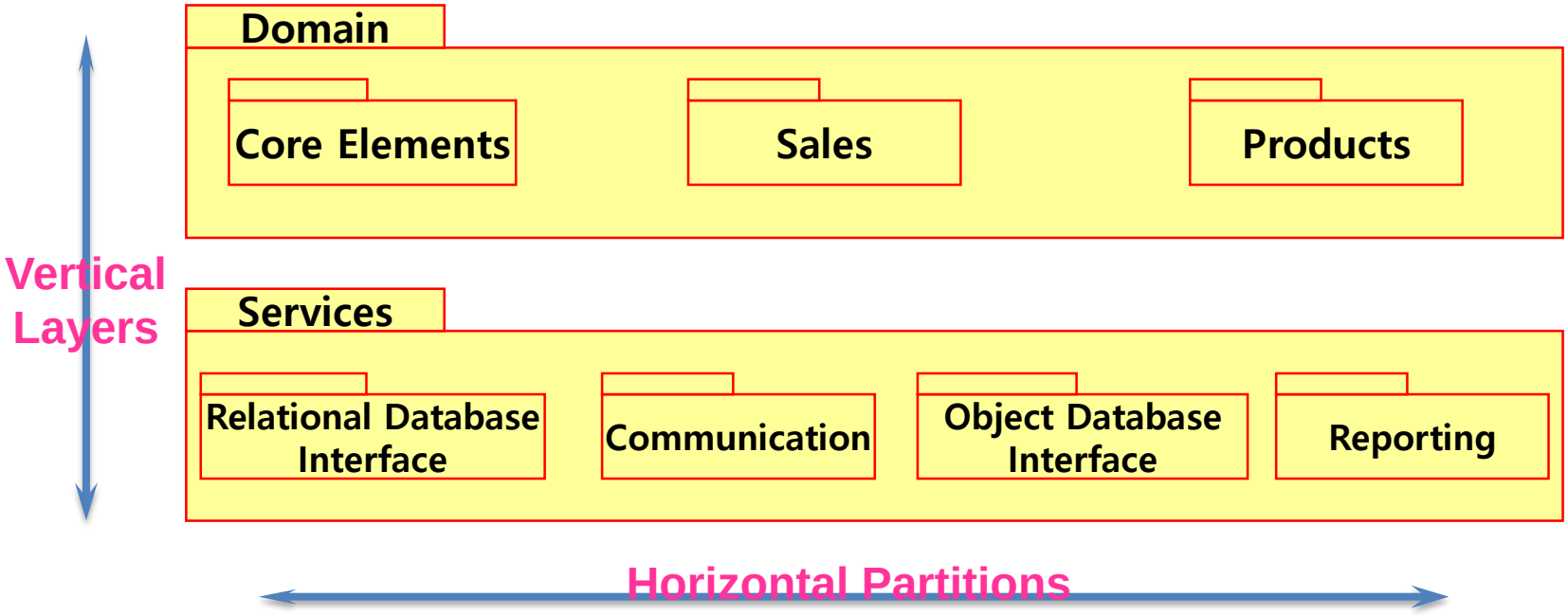
4. Identify packages
   - Place elements together
     - that are in the same subject area-closely related by concept or purpose, or that are in a type hierarchy together
     - that participate in the same use cases or
     - that are strongly associated



Activity 2043. Refine System Architecture

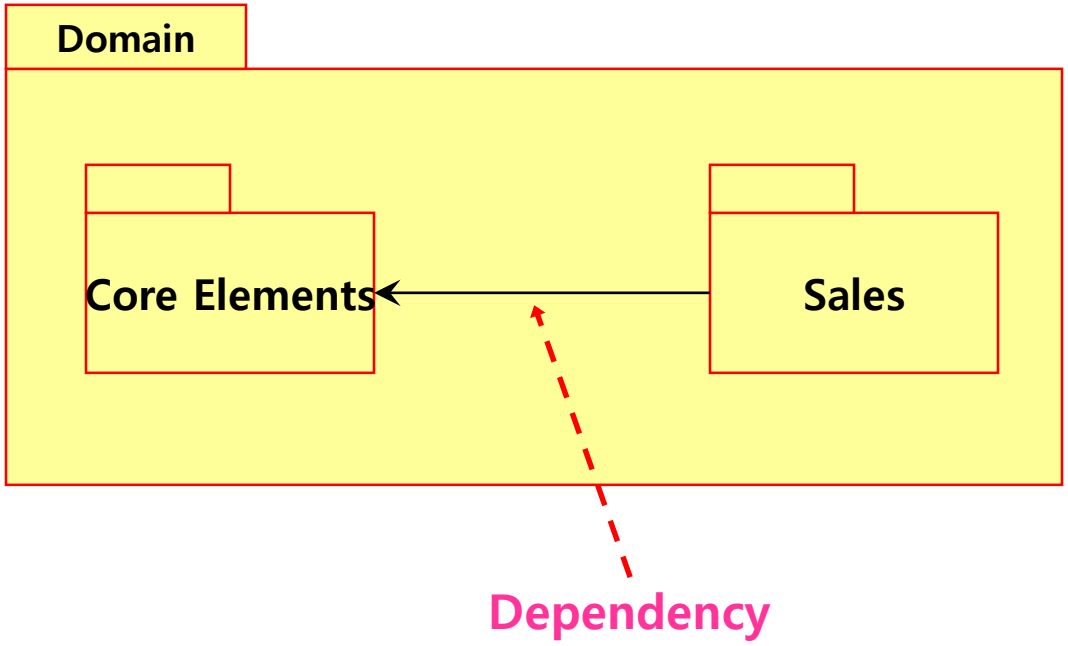
5. Layers of the architecture :
   - vertical tiers

   Partitions of the architecture :
   - horizontal division of relatively parallel subsystems



Activity 2043. Refine System Architecture

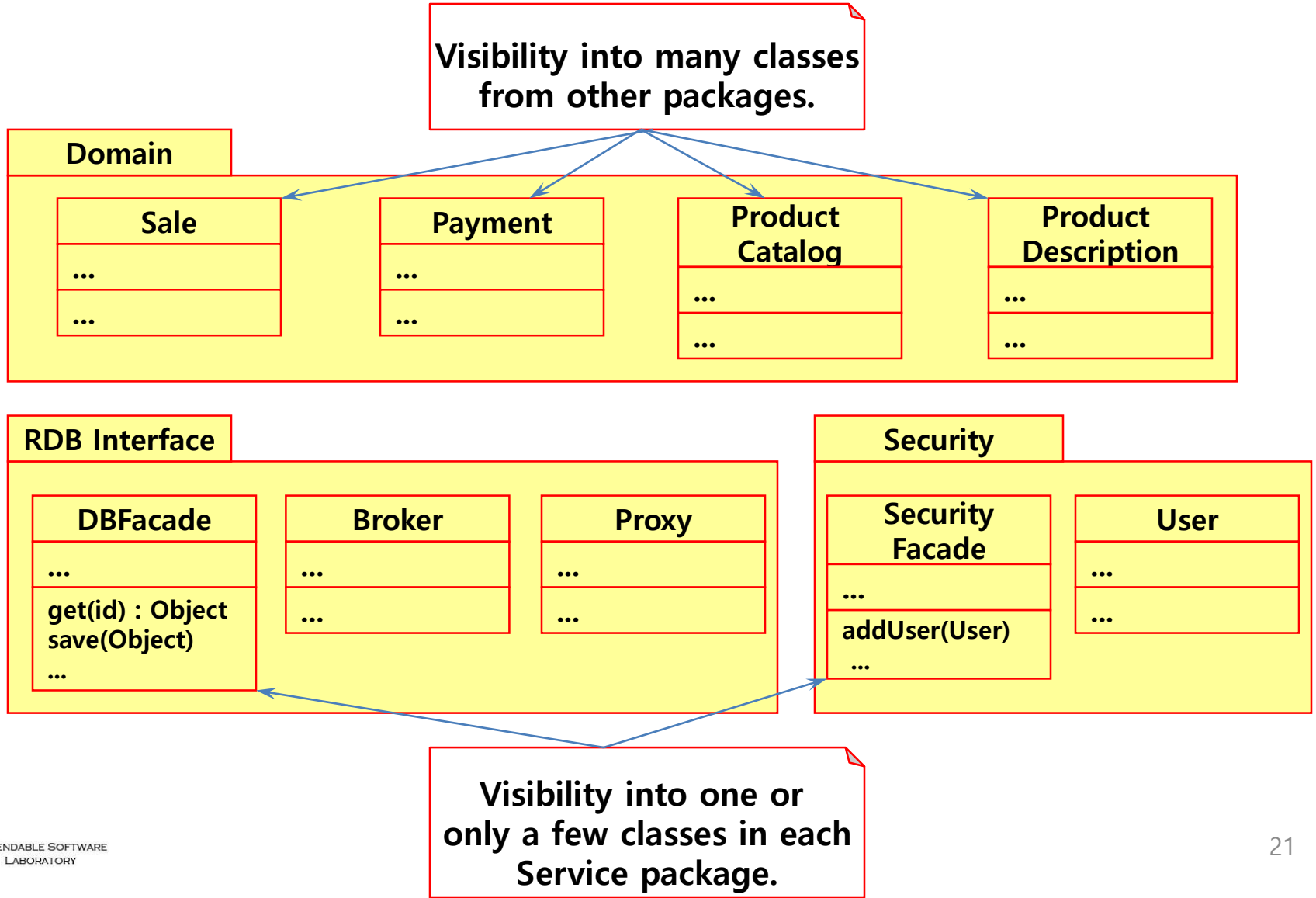
6. Determine package dependencies
   - Dependency relationships indicates coupling between packages.



Activity 2043. Refine System Architecture

7. Assign visibility between package classes.
   - Access into the Domain packages
     - Some packages, typically the presentation package, have visibility into many of the classes representing domain concepts
   - Access into the Service packages
     - Some packages, typically the Domain and Presentation packages, have visibility into only one or a very few classes in each particular Service package
   - Access into the Presentation packages
     - No other packages have direct visibility to the Presentation layer

Activity 2043. Refine System Architecture



Activity 2044. Define Interaction Diagrams

- Description
  - Collaboration diagrams illustrate object interactions in a graph or network format.
  - To illustrate how objects interactions via messages to fulfill tasks.
  - Input : Real Use Case Descriptions
  - Output : An interaction diagram
  - Standards Applied
    - UML's **Sequence Diagram** or Collaboration Diagram

Activity 2044. Define Interaction Diagrams

- Interaction diagram is a generalization of two more specialized UML diagram types:
  - Collaboration diagram
  - Sequence diagram
- The both can be used to express similar message interactions

- Collaboration Diagram
  - Illustrates object interactions in a graphs or network format
- Sequence Diagram
  - Illustrates interactions in a kind of fence format, in which each new object is added to the right.

Activity 2044. Define Interaction Diagrams

- Sequence Diagram vs. Collaboration Diagram

Type	Strengths	Weaknesses
Sequence Diagram	Clearly shows sequence or time ordering of messages	Forced to extend to the right, when adding new objects with consuming horizontal space
Collaboration Diagram	Space economical and flexible to add new objects in two dimensions Better to illustrate complex branching, iteration, and concurrent behavior	Difficult to see sequence of messages

Activity 2044. Define Interaction Diagrams

- Steps
  1. Draw up actors
  2. Deploy objects or classes participating each use case from the real use case descriptions and conceptual class diagram
  3. Design a system of interacting objects to fulfill the tasks.
     - Regard the use case description as a starting point

Activity 2044. Define Interaction Diagrams

- Illustrating Classes and Instances

Class

Instance

Named Instance

Activity 2044. Define Interaction Diagrams

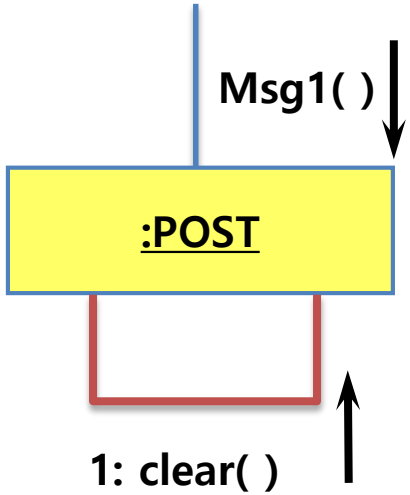
- Illustrating Links and Parameters
  - A link is a connection path between two instances.

- Illustrating a Return Value

Activity 2044. Define Interaction Diagrams

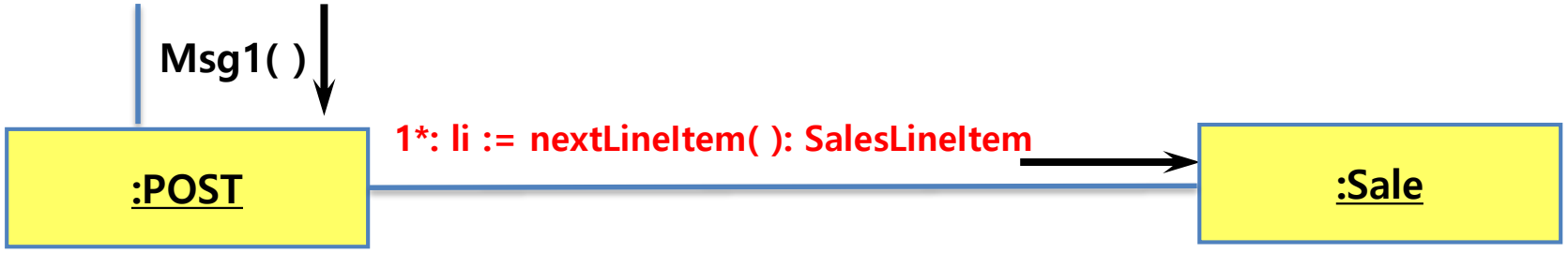
- Message Syntax
  - return := message(parameter : parameterType) : returnType
  - Standard UML message syntax

- Illustrating Messages to 'Self' ('This')

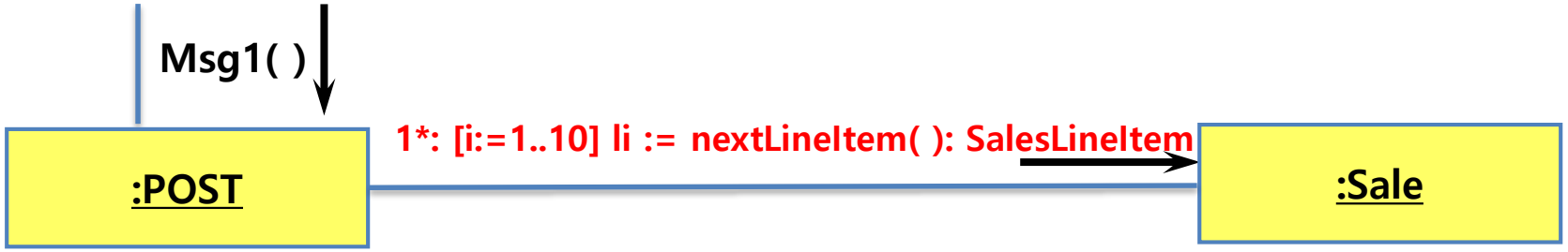


Activity 2044. Define Interaction Diagrams

- Illustrating Iterations
  - Iteration



  - Iteration Clause

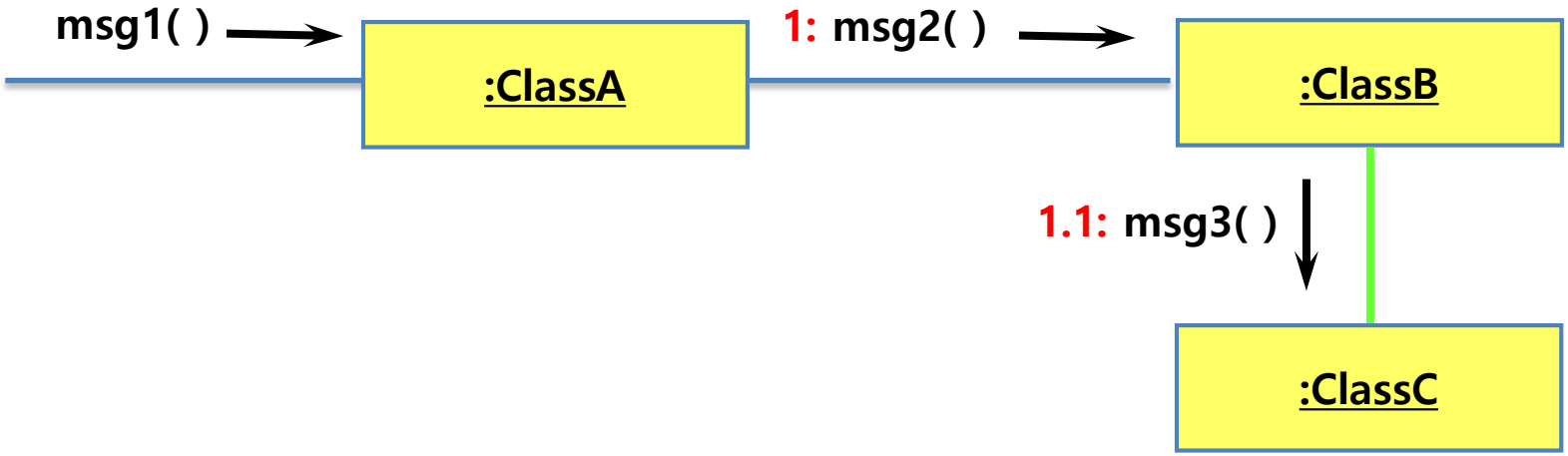


Activity 2044. Define Interaction Diagrams

- Illustrating Creation of Instances
  - Creating message with optional initializing parameters"
- Illustrating Conditional Messages

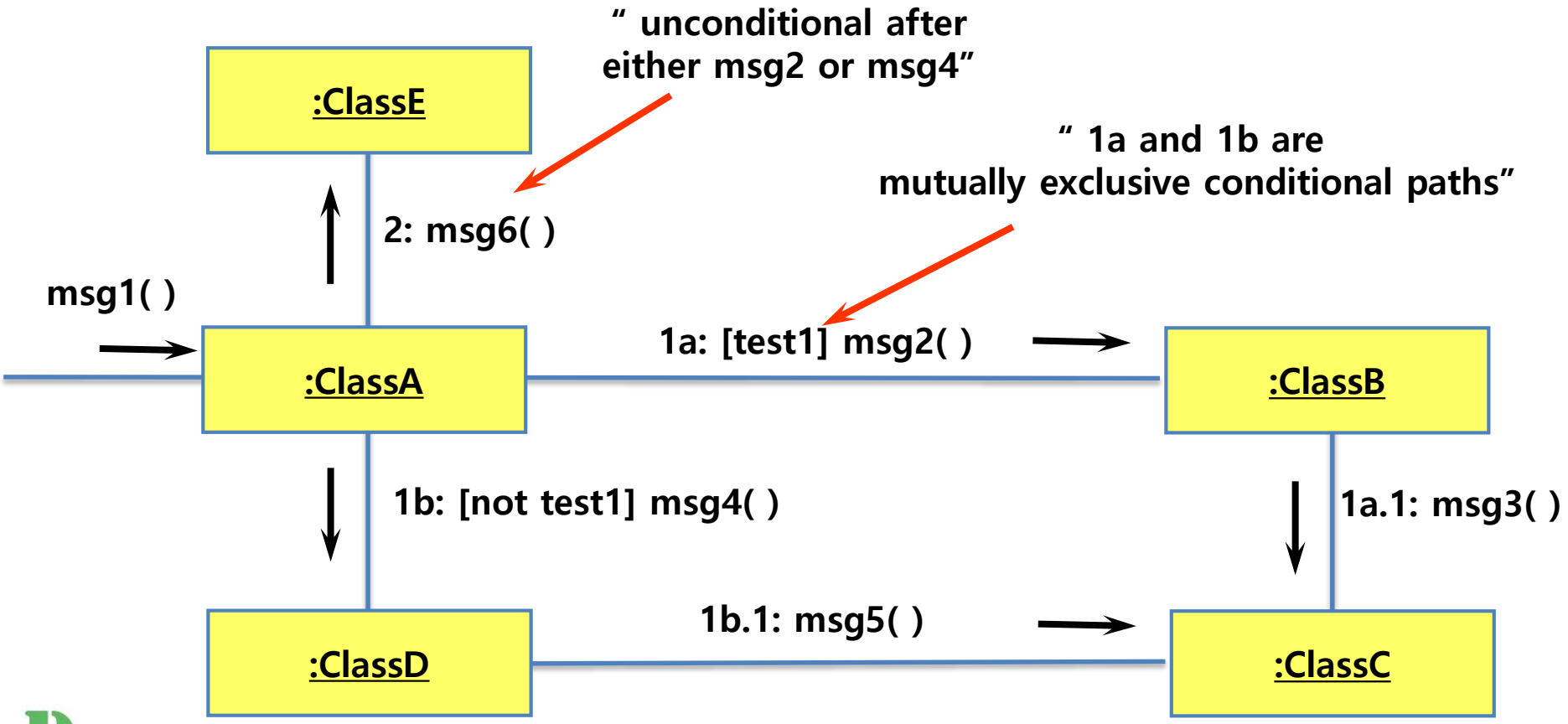
Activity 2044. Define Interaction Diagrams

- Illustrating Message Number Sequencing
  - The first message is not numbered
  - The order and nesting of subsequent messages are shown with a legal numbering scheme



Activity 2044. Define Interaction Diagrams

- Illustrating Mutually Exclusive Conditional Paths

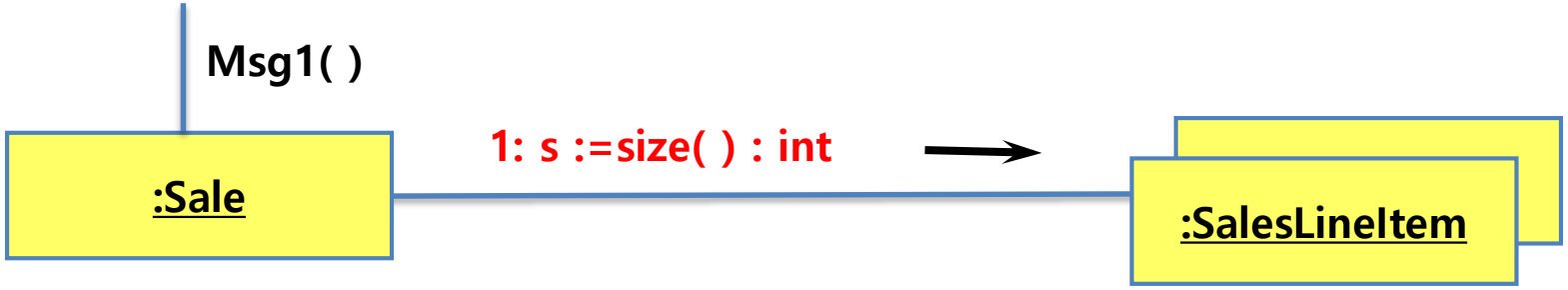


Activity 2044. Define Interaction Diagrams

- Illustrating Collections
  - A multi-object, or set of instances, may be shown with a stack icon

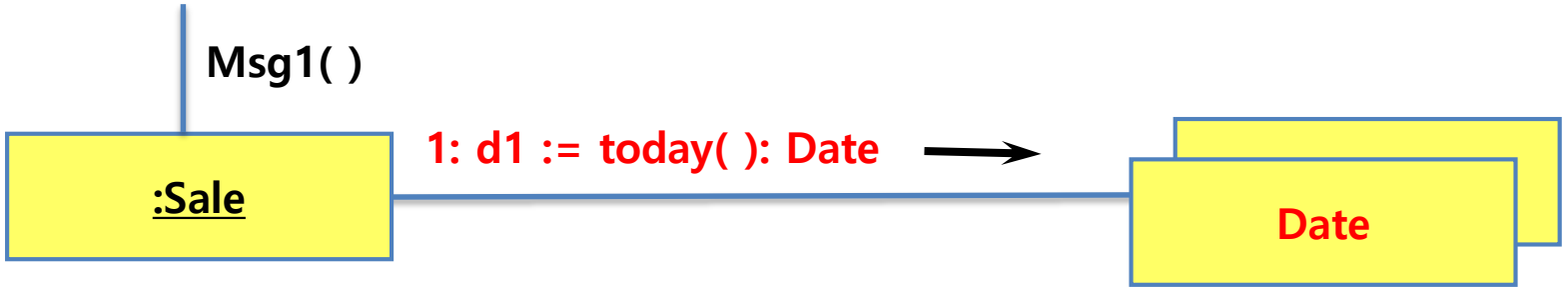
Activity 2044. Define Interaction Diagrams

- Illustrating Messages to Multi-objects
  - A message to a multi-object icon indicates that it is sent to the collection object itself



Activity 2044. Define Interaction Diagrams

- Illustrating Messages to a Class Object
  - Messages may be sent to a class itself not an instance, in order to invoke class methods



Activity 2045. Define Design Class Diagrams

- Description
  - Describes more details in conceptual class diagram
  - Add navigability, dependency, data type, operation signature, parameters, return types, and so on.
  - Input : Interaction Diagram, Conceptual Class Diagram
  - Output : A Design Class Diagram
  - Standards Applied
    - UML's Class Diagram

Activity 2045. Define Design Class Diagrams

- Steps
  1. Identity all classes
  2. Draw them in a class diagram
  3. Add attributes
  4. Add method names
  5. Add type information to the attributes and methods
  6. Add the associations
  7. Add navigability arrows
  8. Add dependency

Activity 2045. Define Design Class Diagrams

- Step 1. Identify all classes
  - by scanning all interaction diagrams
  - listing classes mentioned

POST

ProductCatalog

Store

Payment

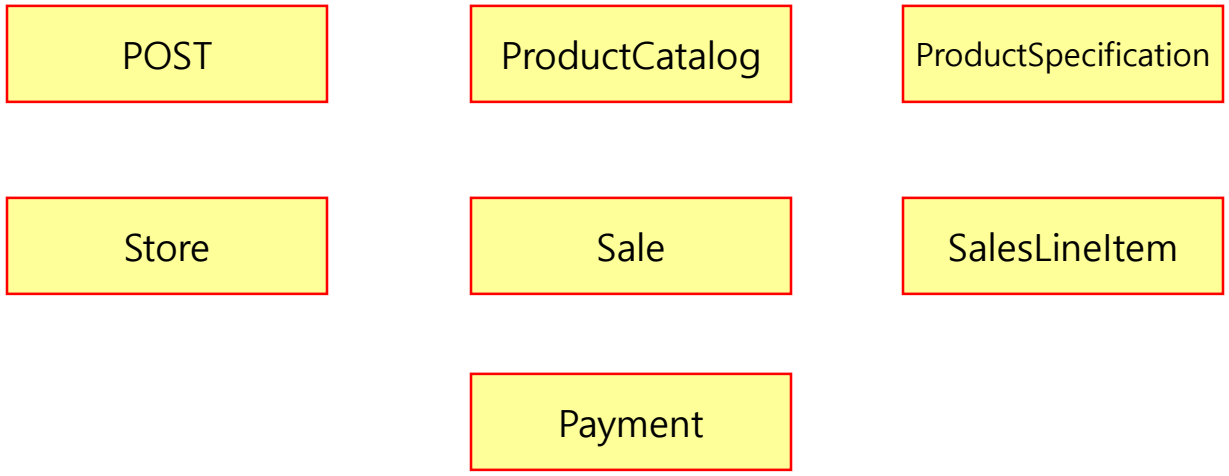
Sale

ProductSpecification

SalesLineItem

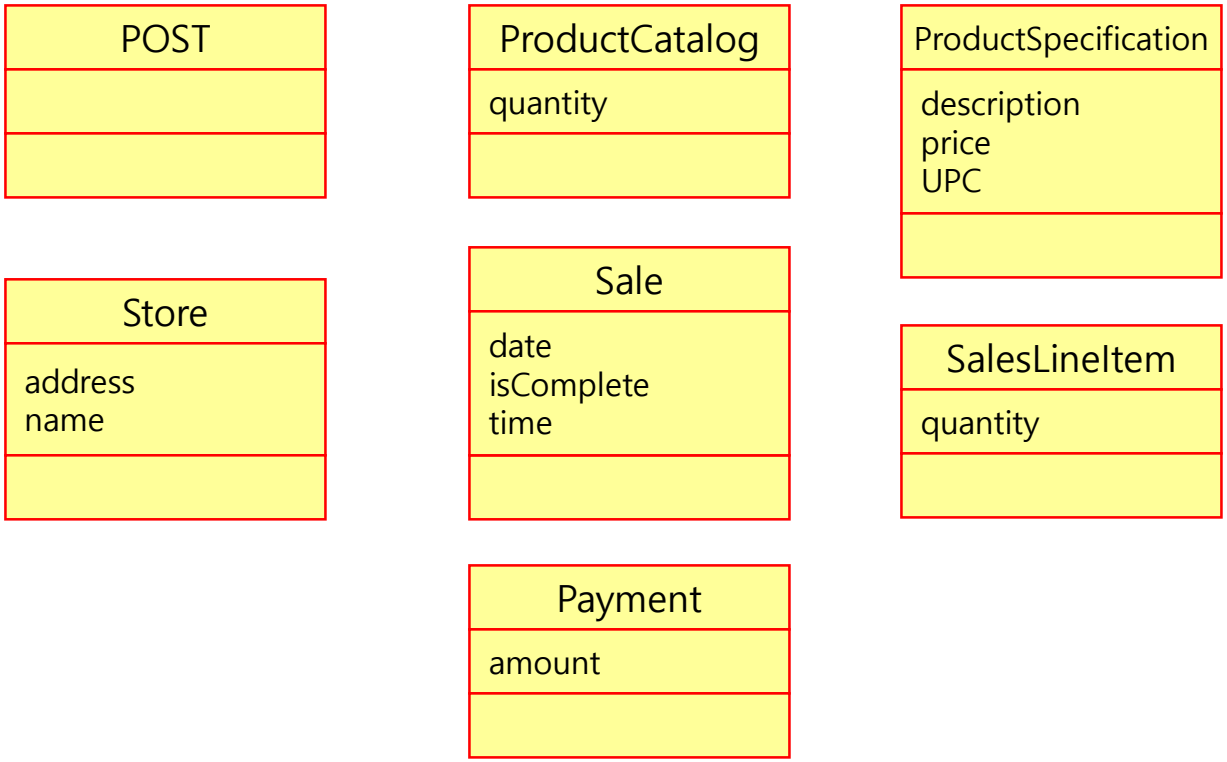
Activity 2045. Define Design Class Diagrams

- Step 2. Draw a class diagram
  - includes classes found in Step 1



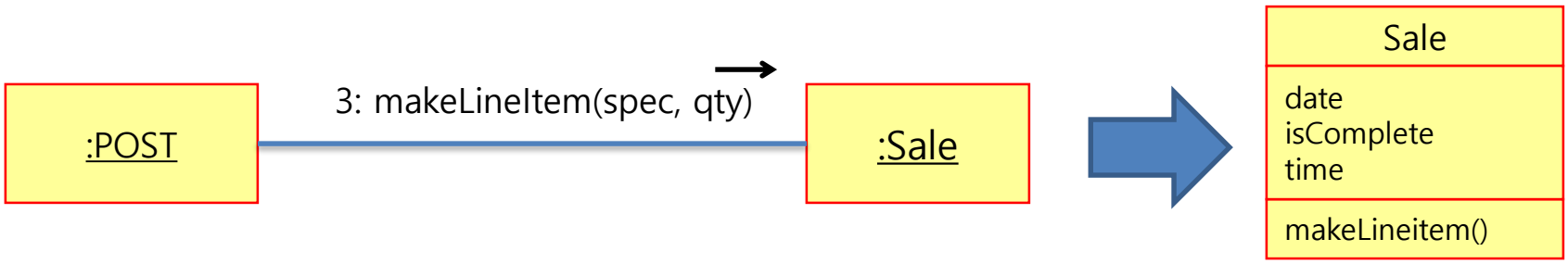
Activity 2045. Define Design Class Diagrams

- Step 3. Add attributes
  - Include the attributes previously identified in the conceptual class diagram that are also used in the design

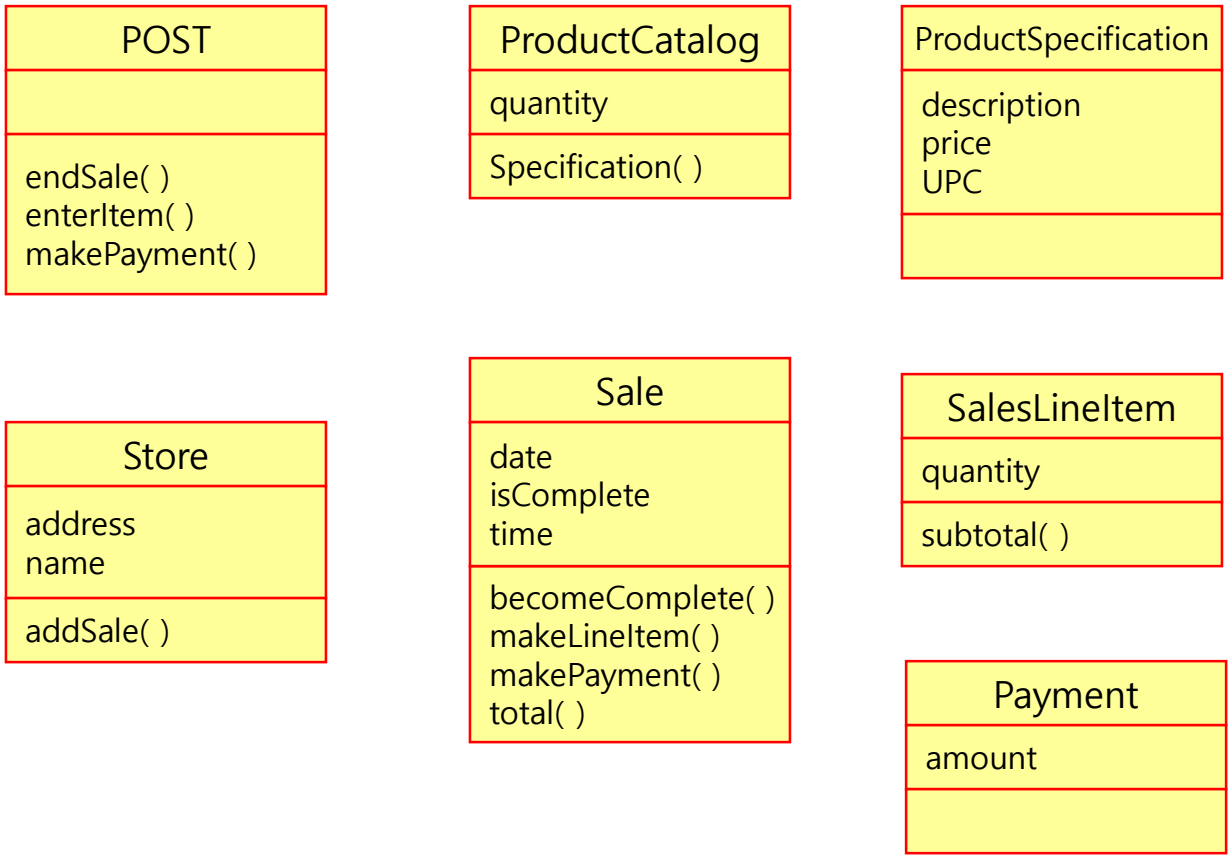


Activity 2045. Define Design Class Diagrams

- Step 4. Add method names
  - Identify method of each class by scanning the interaction diagrams
  - The messages sent to a class in interaction diagrams must be defined in the class
  - Don't add
    - creation methods and constructors
    - accessing methods
    - messages to a multiobject



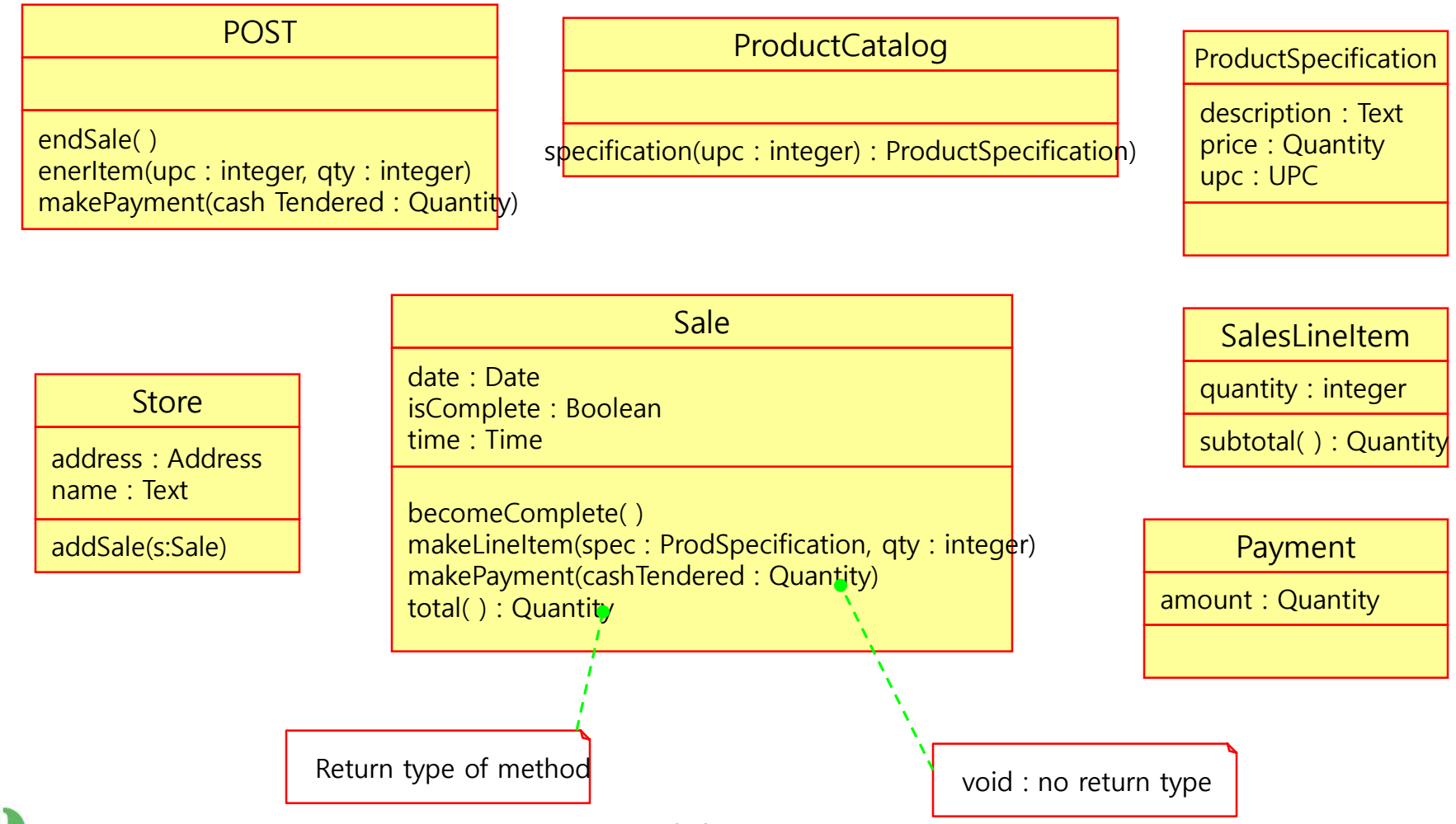
Activity 2045. Define Design Class Diagrams



Activity 2045. Define Design Class Diagrams

- Step 5. Add type information
  - Show types of attributes, method parameters, and method return values optionally.
  - Determine whether to show type information or not
    - When using a CASE tool with automatic code generation, exhaustive details are necessary
    - If it is being created for software developers to read, exhaustive detail may adversely effect the noise-to-value ratio

Activity 2045. Define Design Class Diagrams

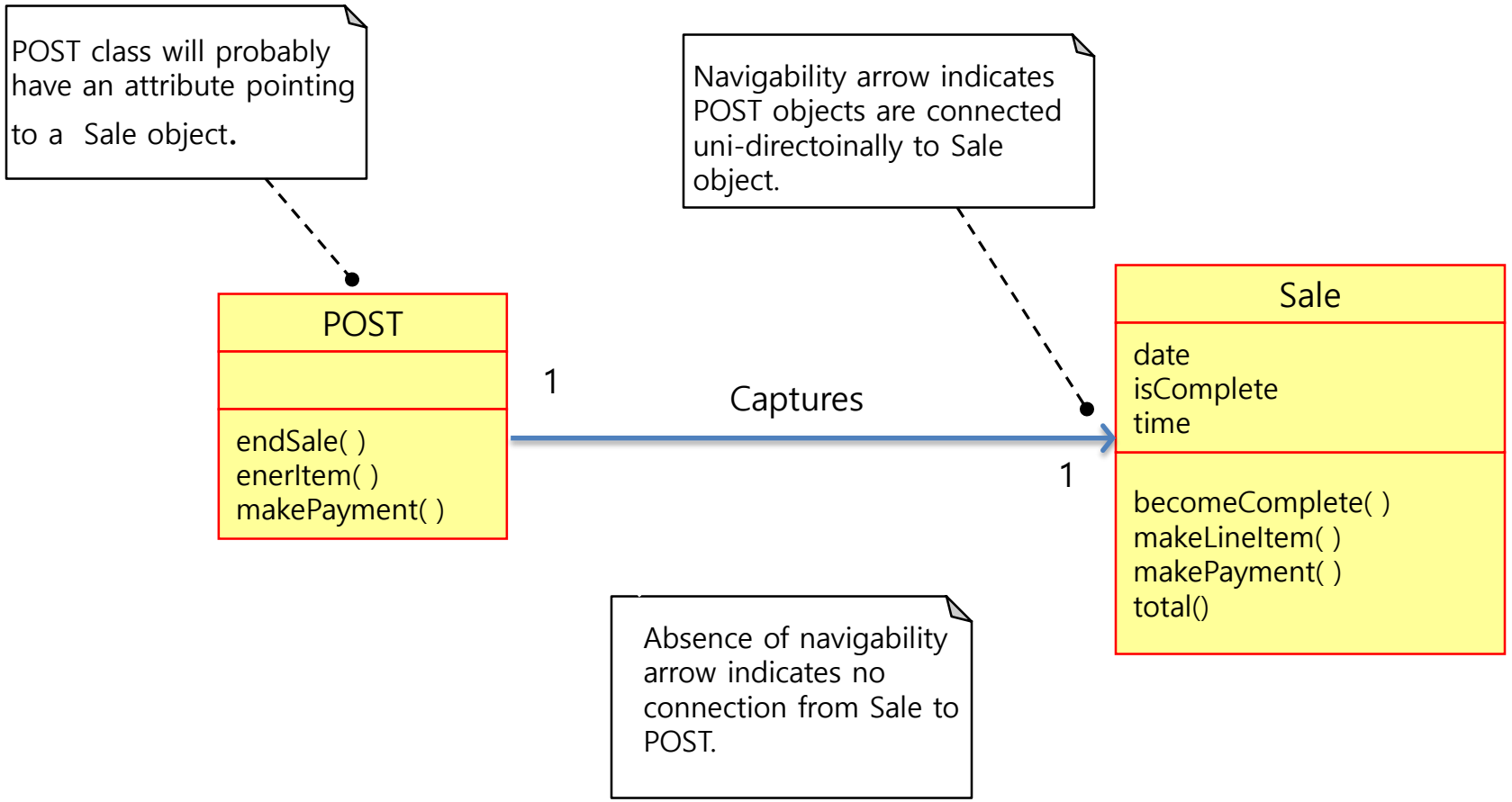


Activity 2045. Define Design Class Diagrams

- Step 6. Add associations
  - Choose associations by software-oriented need-to-know criterion from the interaction diagrams

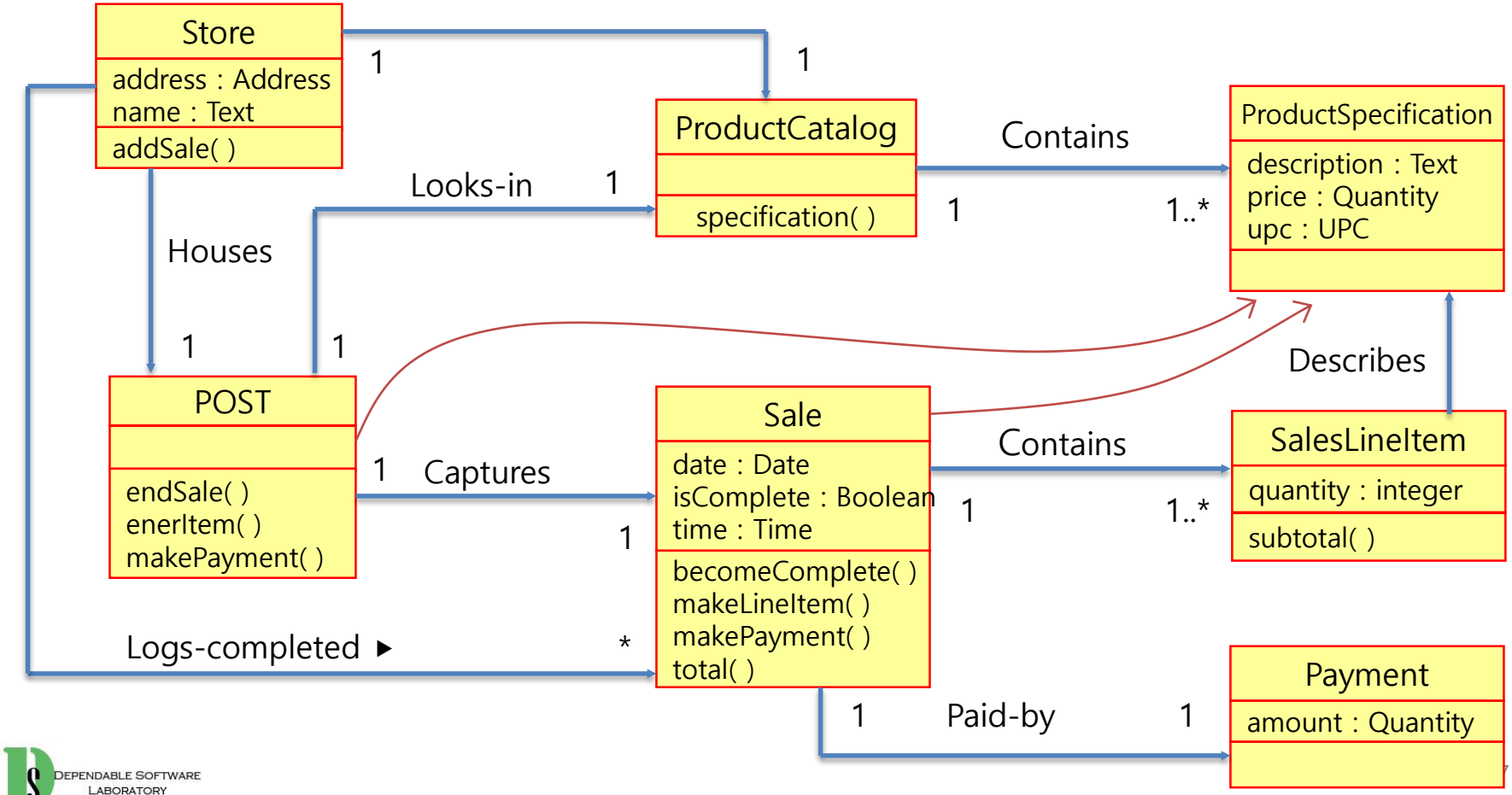
- Step 7. Add navigability arrows
  - According to the interaction diagram
  - Common situations to define navigability
    - A sends a message to B
    - A creates an instance B
    - A needs to maintain a connection to B

Activity 2045. Define Design Class Diagrams

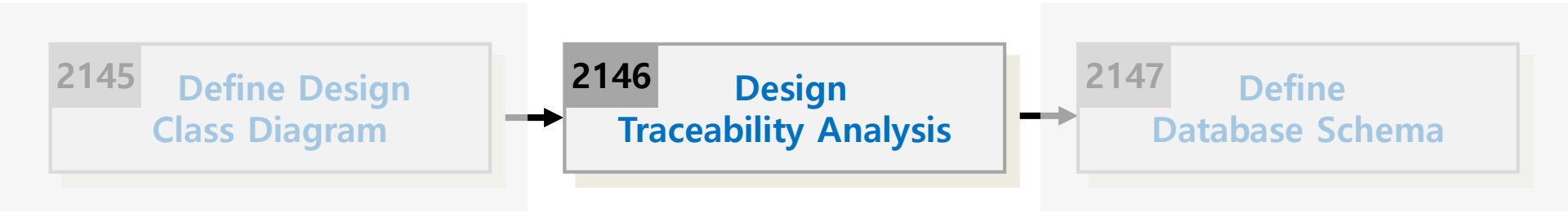


Activity 2045. Define Design Class Diagrams

- Step 8. Add dependency relationship
  - when there is non-attribute visibility between classes
  - Non-attribute visibility : parameter, global, or locally declared visibility



Activity 2046. Design Traceability Analysis



- Description
  - Analysis the connection of results which are the results of analyze and design step
    - Identify the connection of use cases and class, methods and test cases
  - Express the traces about requirements to test cases

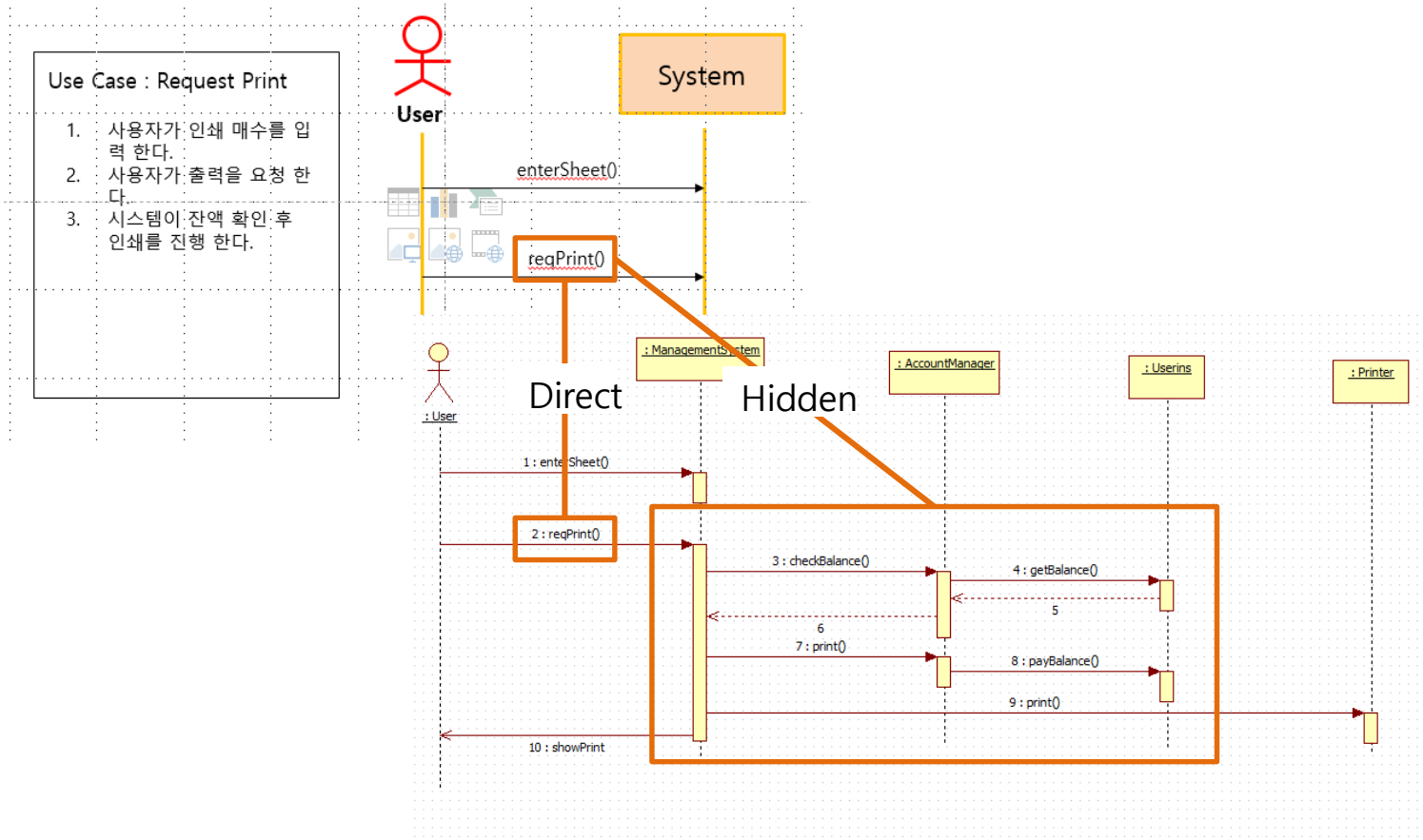
  - Input : Real use case description, design class diagram, functional requirements, System test cases
  - Output : Traceability analysis result

Activity 2046. Design Traceability Analysis

- Steps:
  1. Identify the related information of essential and real use cases
     - 1:1 or more
  2. Identify the traces between operation contracts(2036) and operations in interaction diagram(2044)
     - Express the direct contacts or hidden contacts
  3. Identify the relations of the results of step 2 and class diagram (class, method)
  4. Writing the results of the analysis
  5. If the operations which are not expressed directly in this step (e.g. GUI related operation like text input), they should be written by 2053

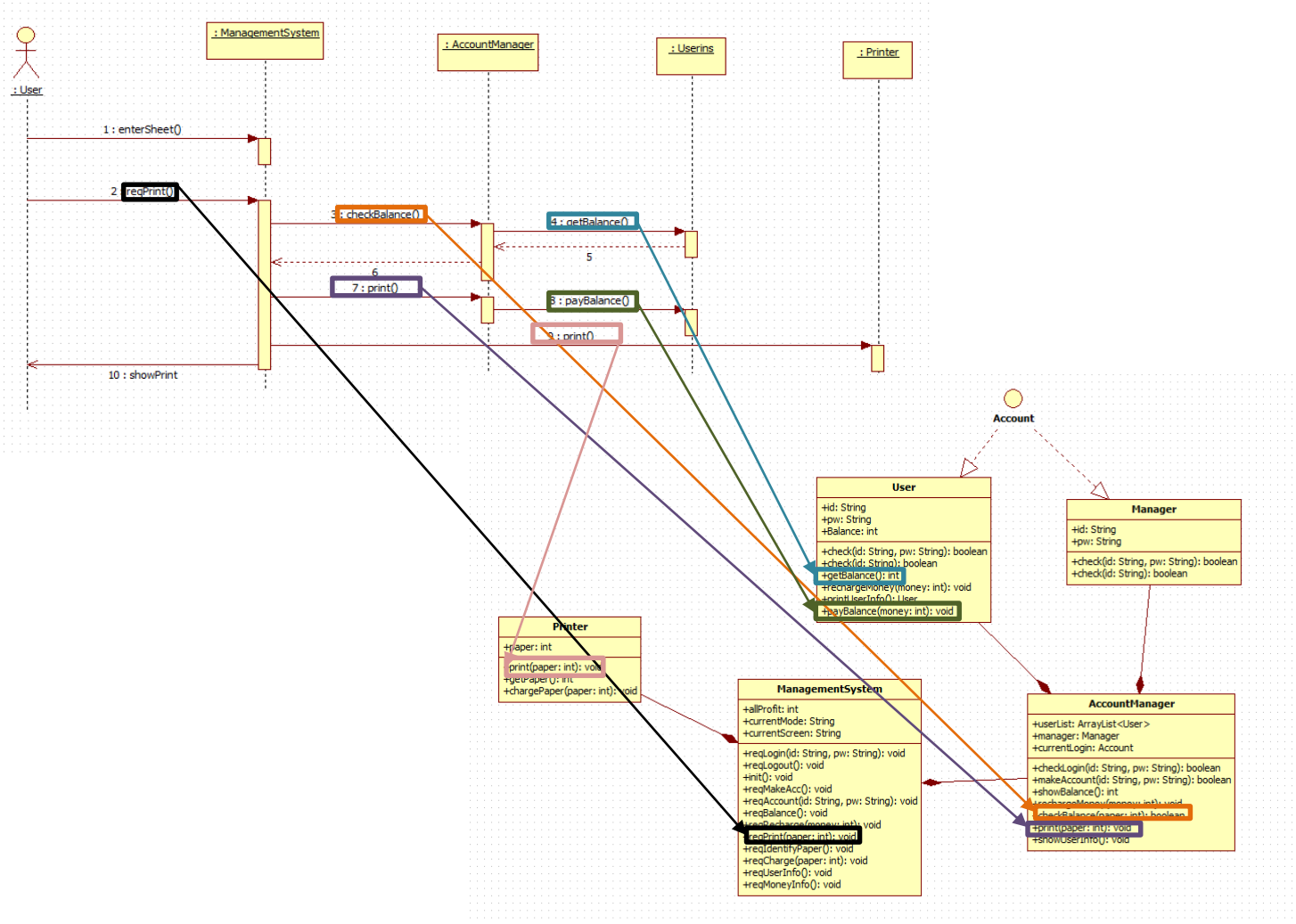
Activity 2046. Design Traceability Analysis

- Draw up the traces between operation contracts(2036) and operations in interaction diagram(2044)
  - Direct : Operation which is connected directly
  - Hidden : Operations which are used to operate the function invisibly

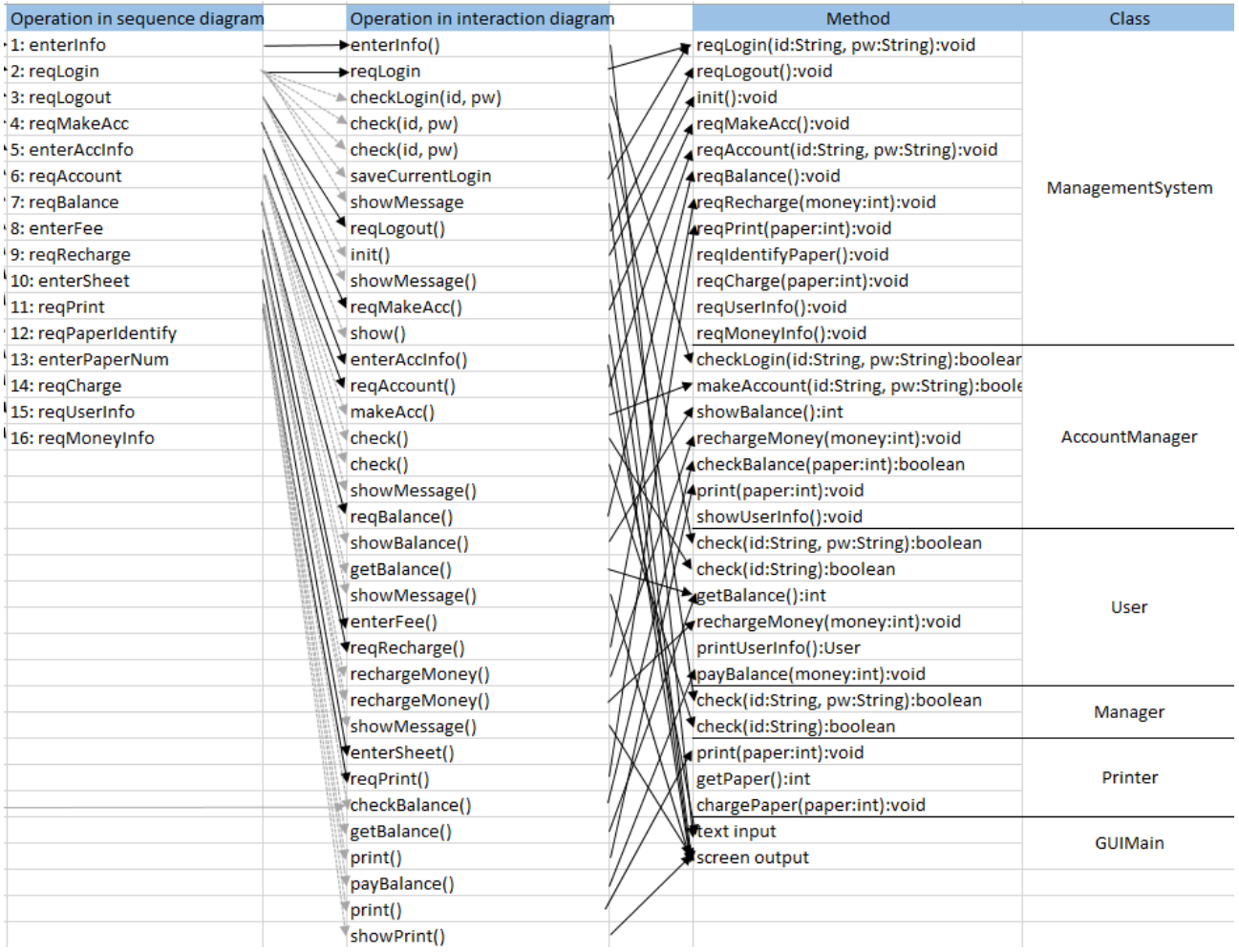


Activity 2046. Design Traceability Analysis

- The results of step 3 and 4

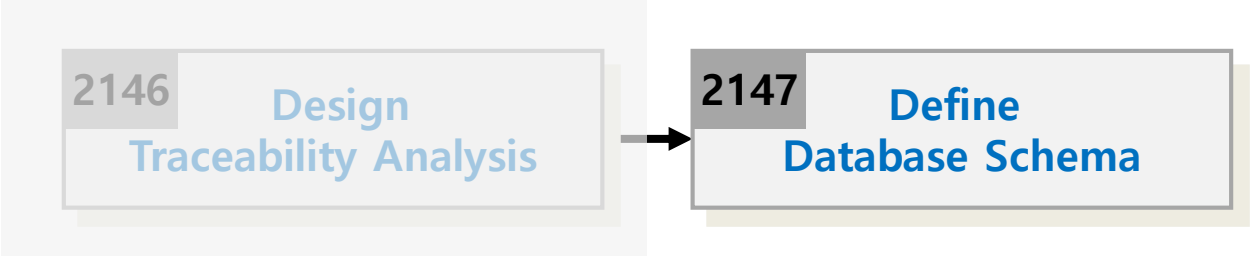


Activity 2046. Design Traceability Analysis



Operation in sequence diagram	Operation in interaction diagram	Method	Class
1: enterInfo	enterInfo()	reqLogin(id:String, pw:String):void	ManagementSystem
2: reqLogin	reqLogin	reqLogout():void	
3: reqLogout	checkLogin(id, pw)	init():void	
4: reqMakeAcc	check(id, pw)	reqMakeAcc():void	
5: enterAccInfo	check(id, pw)	reqAccount(id:String, pw:String):void	
6: reqAccount	saveCurrentLogin	reqBalance():void	
7: reqBalance	showMessage	reqRecharge(money:int):void	
8: enterFee	reqLogout()	reqPrint(paper:int):void	
9: reqRecharge	init()	reqIdentifyPaper():void	
10: enterSheet	showMessage()	reqCharge(paper:int):void	
11: reqPrint	reqMakeAcc()	reqUserInfo():void	
12: reqPaperIdentify	show()	reqMoneyInfo():void	
13: enterPaperNum	enterAccInfo()	checkLogin(id:String, pw:String):boolean	AccountManager
14: reqCharge	reqAccount()	makeAccount(id:String, pw:String):boole	
15: reqUserInfo	makeAcc()	showBalance():int	
16: reqMoneyInfo	check()	rechargeMoney(money:int):void	
	check()	checkBalance(paper:int):boolean	
	showMessage()	print(paper:int):void	
	reqBalance()	showUserInfo():void	
	showBalance()	check(id:String, pw:String):boolean	User
	getBalance()	check(id:String):boolean	
	showMessage()	getBalance():int	
	enterFee()	rechargeMoney(money:int):void	
	reqRecharge()	printUserInfo():User	
	rechargeMoney()	payBalance(money:int):void	
	rechargeMoney()	check(id:String, pw:String):boolean	Manager
	showMessage()	check(id:String):boolean	
	enterSheet()	print(paper:int):void	Printer
	reqPrint()	getPaper():int	
	checkBalance()	chargePaper(paper:int):void	
	getBalance()	text input	GUIMain
	print()	screen output	
	payBalance()		
	print()		
	showPrint()		

Activity 2047. Define Database Schema



- Description
  - Design database, table, and records
  - Map classes into tables
  - Input : Design Class Diagram
  - Output : A Database Schema
- Steps:
  1. Map classes into tables
  2. Map relationships between classes into relations between tables
  3. Map attributes into fields of tables
  4. Design Schema