CSCI 4050/6050
Software Engineering

Object-Oriented Design

Use-Case Realization with Interaction Diagrams

Dr. Eman Saleh eman.saleh@uga.edu

Outline

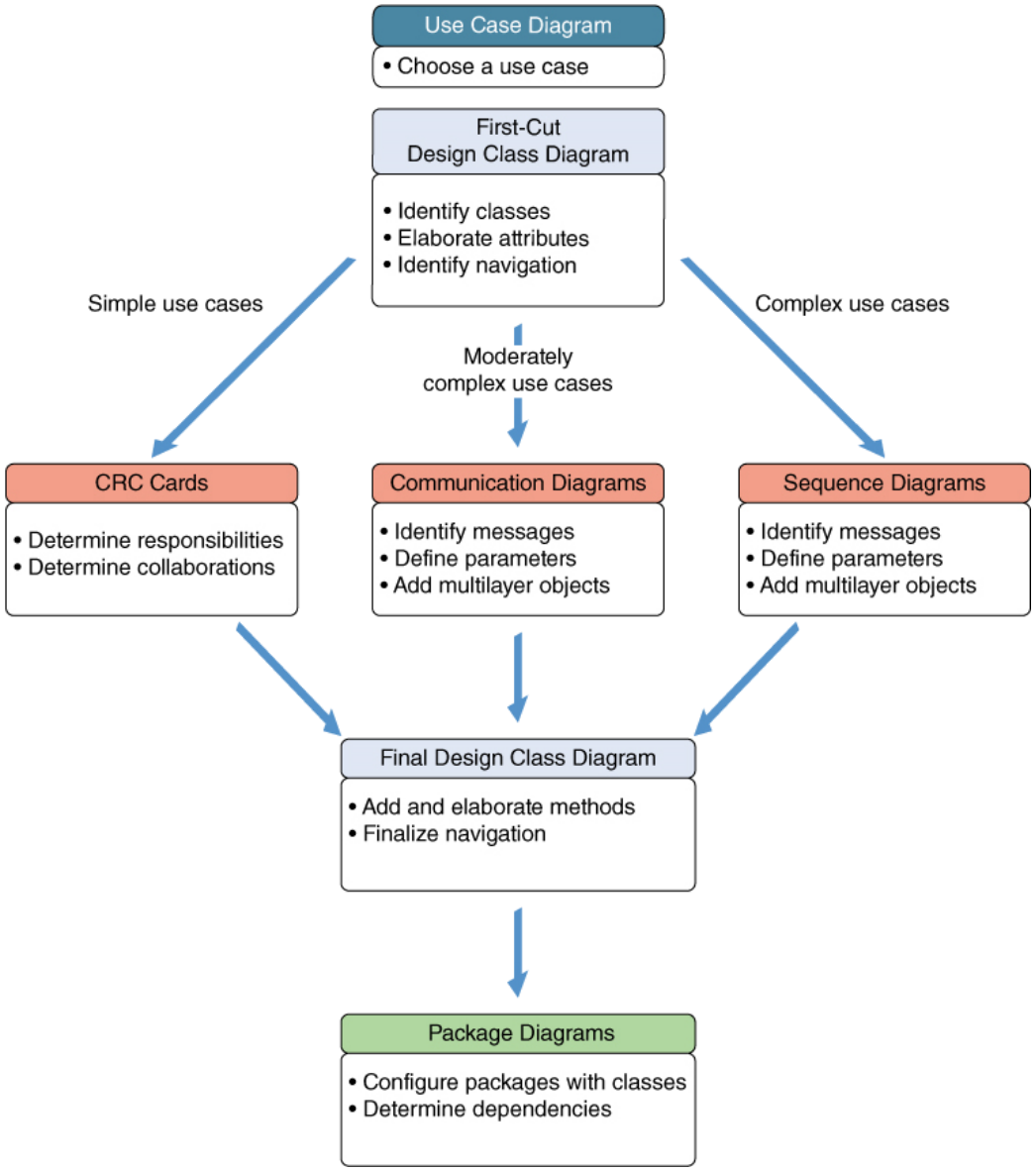
- Object-Oriented Design with Interaction Diagrams
- Use Case Realization with Sequence Diagrams
- Use Case Realization with Communication Diagrams
- Developing a Multilayer Design
- Updating and Packaging the Design Classes

Overview

- In the previous lecture introduced software design concepts for OO programs, use case realization and fundamental design principles
- This chapter continues the discussion of OO software design at a more advanced level
- Three layer design is demonstrated using communication diagrams, sequence diagrams, package diagrams, and design patterns
- Design is shown to proceed use case by use case, and within each use case, layer by layer

OOD

- First-cut DCD
- Extend use case with Communication Diagram or Sequence diagram
- Final DCD
- Package diagram



OOD with Interaction Diagrams

- Three layers include view layer, business logic/problem domain layer, and data access layer

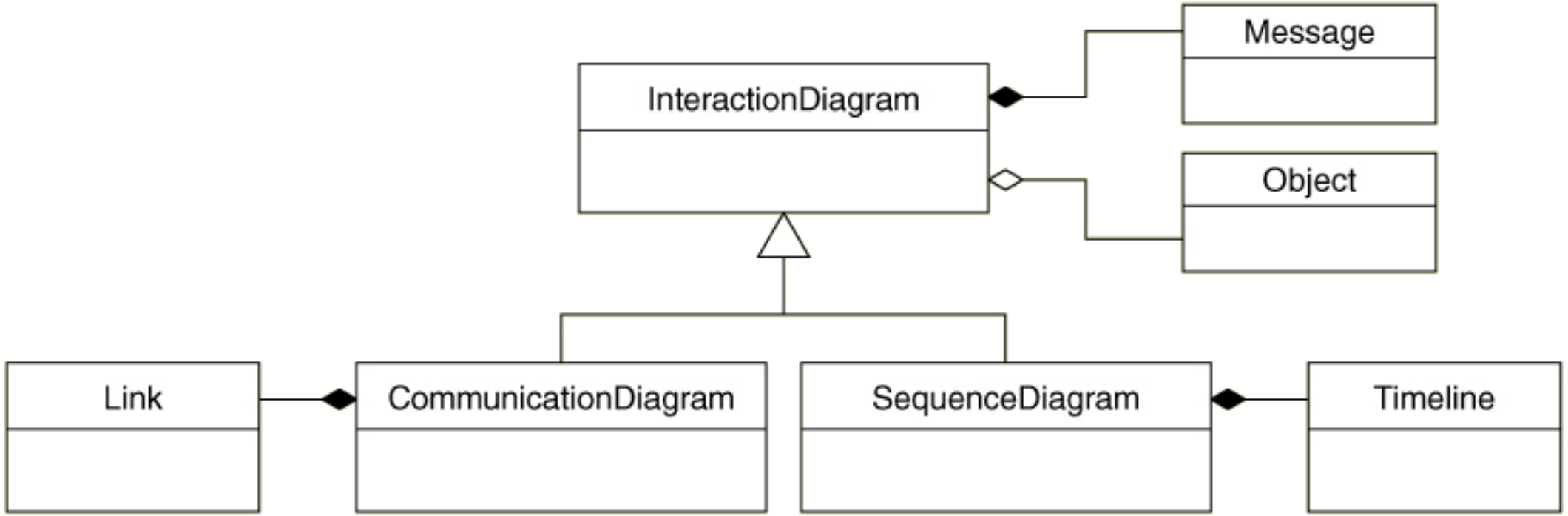
- Questions that come up include
  - How do objects get created in memory?
  - How does the user interface interact with other objects?
  - How are objects handled by the database?
  - Will other objects be necessary?
  - What is the lifespan of each object?

OOD with Interaction Diagrams

- Use case realization
  - The process of elaborating the detailed design for a particular use case using interaction diagrams
- Communication diagram
  - A type of interaction diagram which emphasizes the set of objects involved in a use case.
- Sequence diagram
  - A type of interaction diagram which emphasizes the sequence of messages involved in a use case.

Interaction diagrams

- Communication and Sequence diagrams and their components



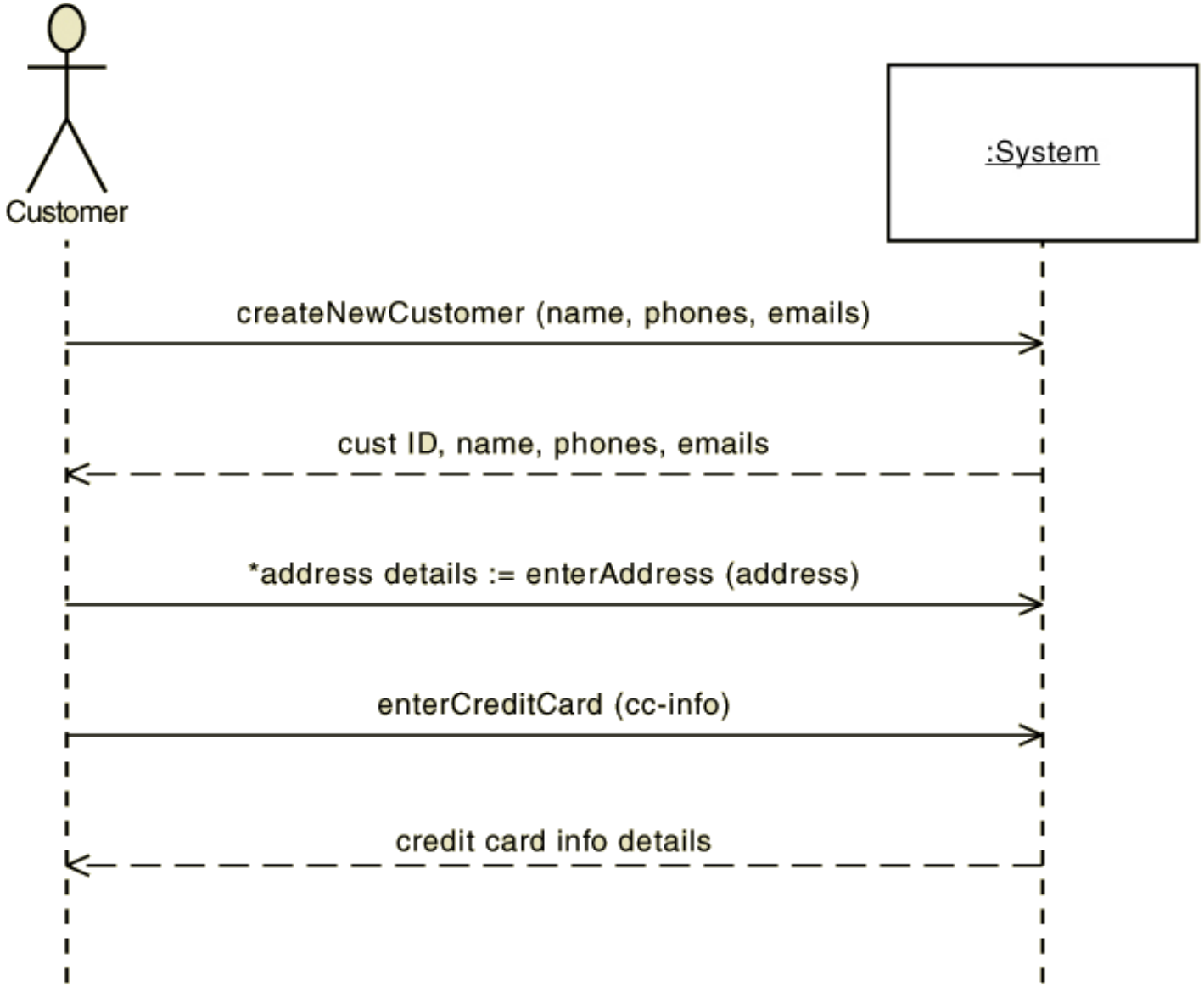
Use case realization with sequence diagrams

OOD with Sequence Diagrams

- Choose a use case
  - Input models – activity diagram, SSD, classes
- **Create first-cut DCD**
- Extend input messages
  - Add all required internal messages
  - Origin and destination objects
  - Elaborate each message
- Add other layers as desired (view, data access)
- Update DCD

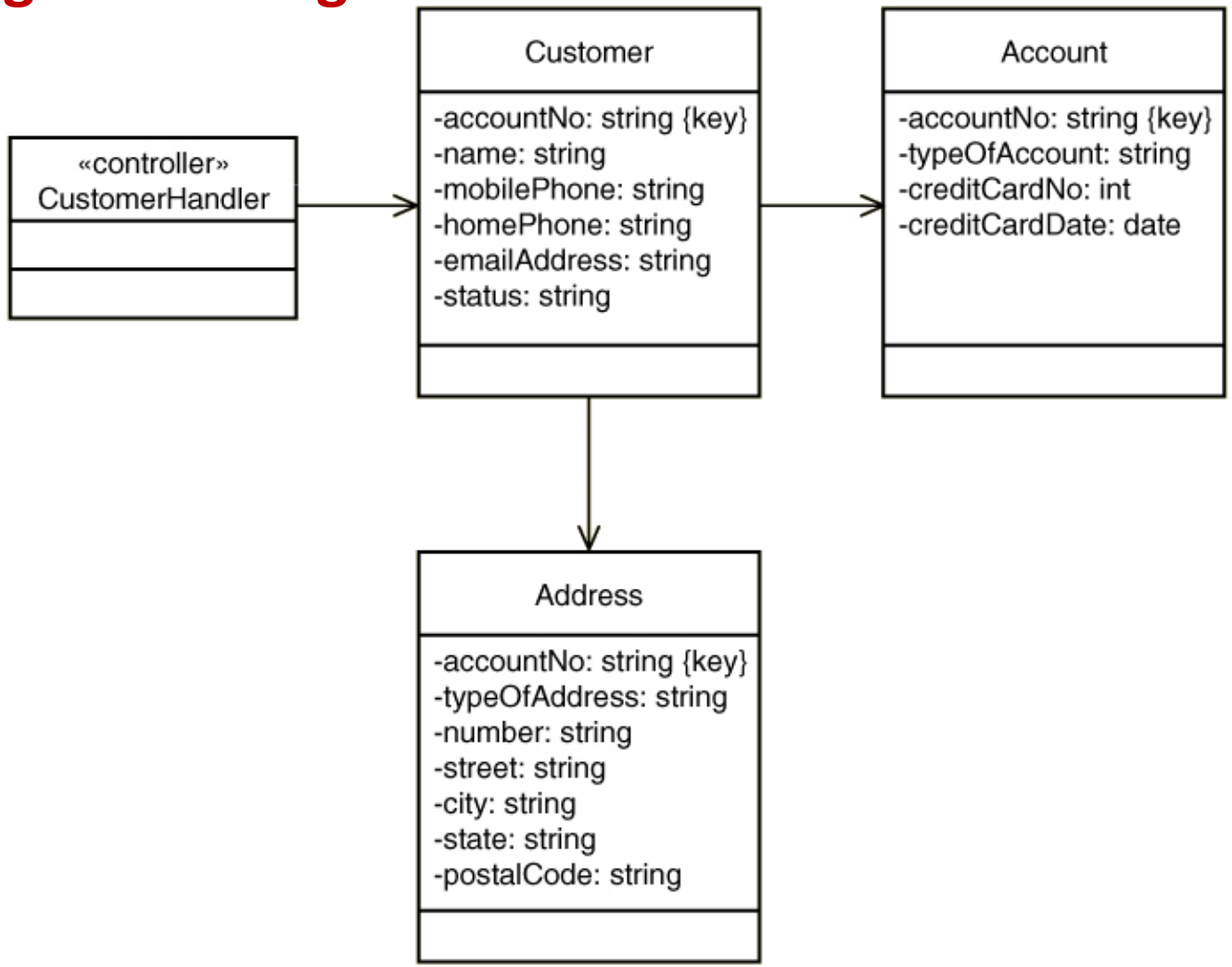
OOD for *Create customer account*

- Input model -- SSD



OOD with Sequence Diagrams: step 1
Create the first-cut design class diagram

1- Elaborate attributes
2- Add a controller
3- Add Navigation arrows



First-cut DCD for *Create customer account* use case

Use Case Controller

- Switchboard between user-interface classes and domain layer classes.
- Reduces coupling between view and domain layer
- A controller can be created for each use case, however, several controllers can be combined together for a group of related use cases
- It is a completely artificial class – an artifact

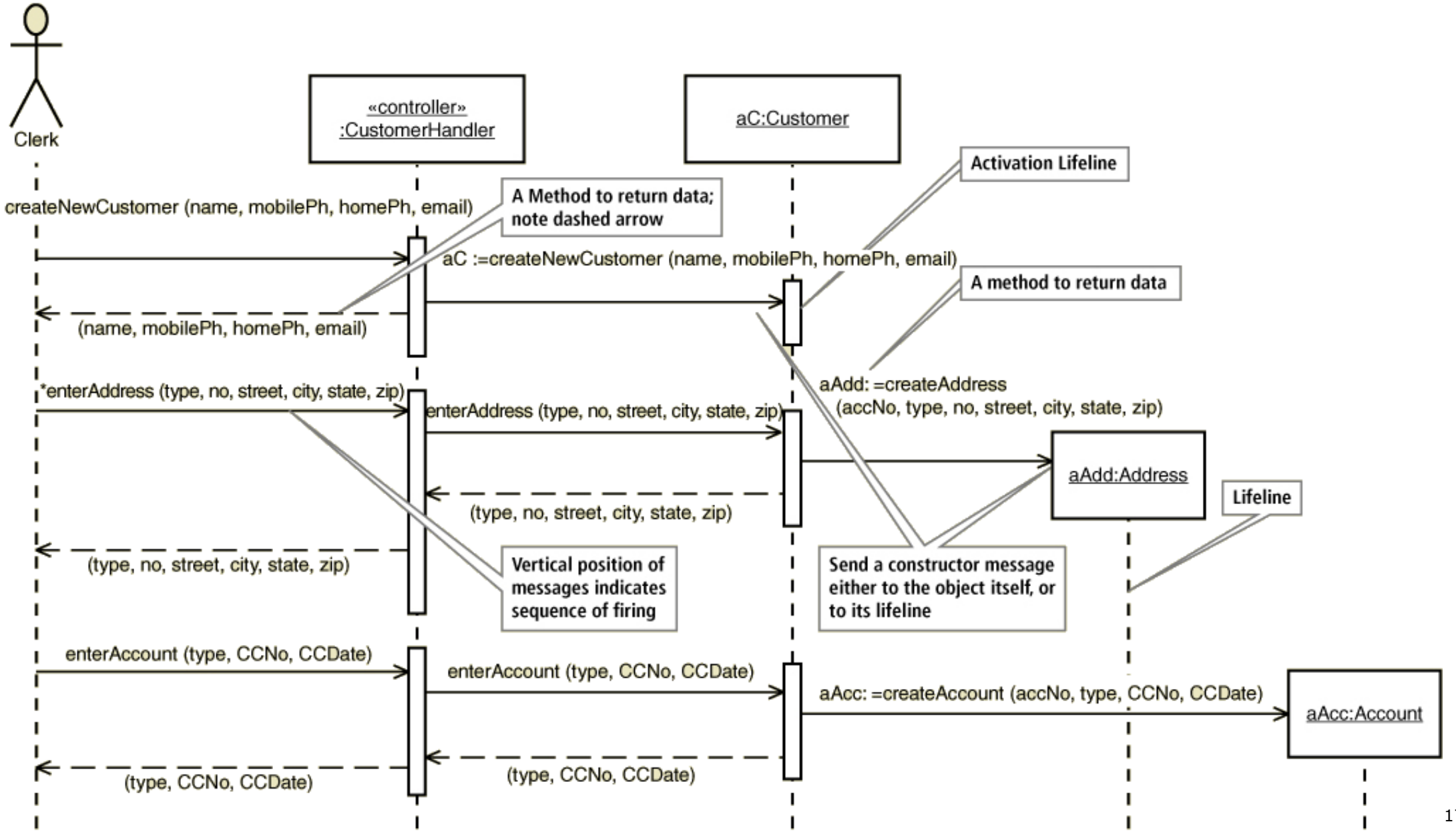
OOD with Sequence Diagrams

- Choose a use case
  - Input models – activity diagram, SSD, classes
- Create first-cut DCD
- **Extend input messages**
  - Add all required internal messages
  - Origin and destination objects
  - Elaborate each message
- Add other layers as desired (view, data access)
- Update DCD

OOD for *Create customer account*

- Extend input messages
  1. From the first cut DCD put objects on sequence diagram
  2. For each message, find primary object, ensure visibility, elaborate use case with all messages between objects
  3. Name each message and add all required message elements

Understanding Sequence Diagrams

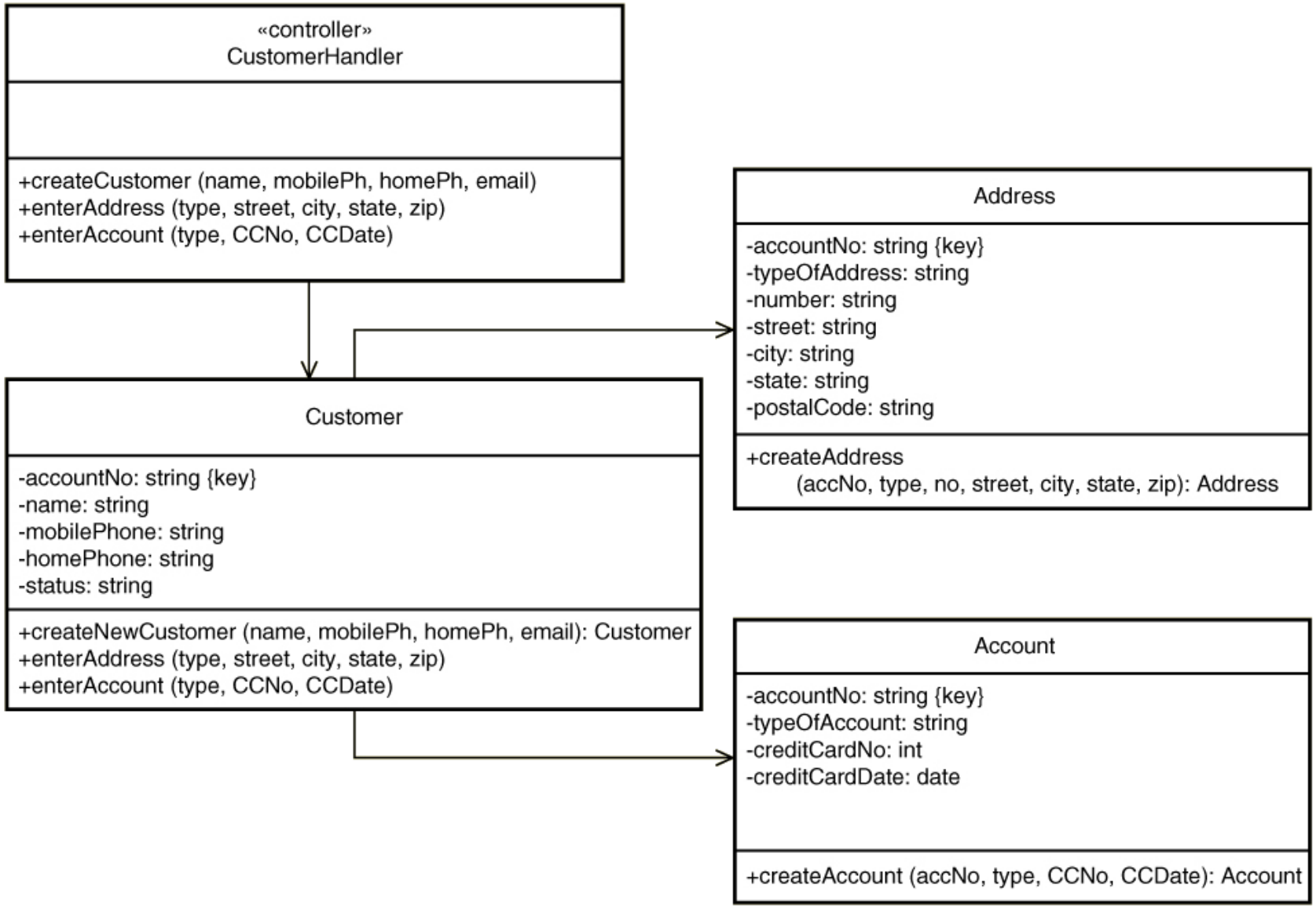


Understanding Sequence Diagrams

- Lifeline
  - The dashed line under the object which serves as an origin point and a destination point for messages
- Activation lifeline
  - The vertical box on a lifeline which indicates the time period when the object is executing based on the message
- Messages have origins and destinations
  - May be on lifeline or on object box
  - Return values may be dashed message arrow, or on same message

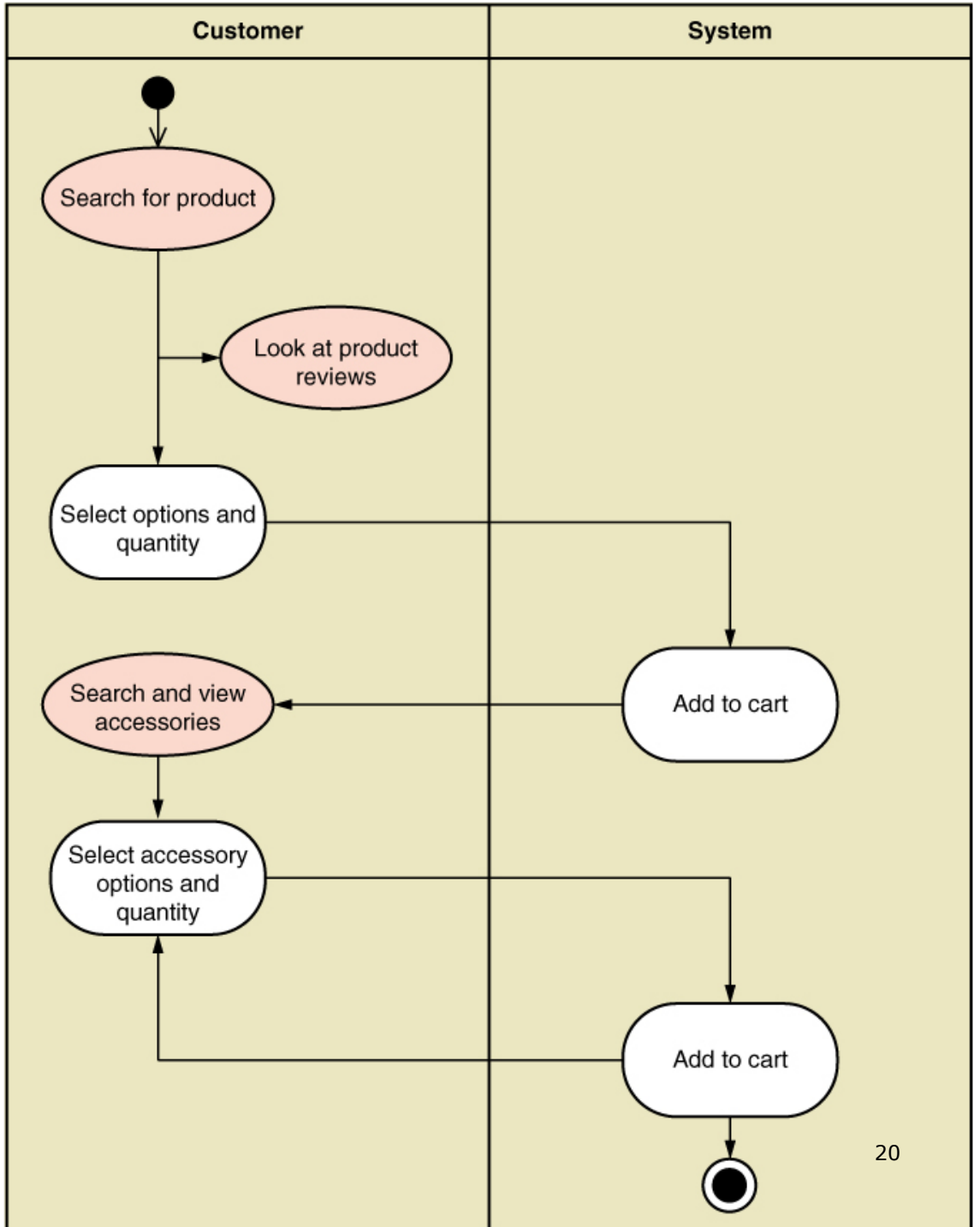
OOD with communication diagrams

- Updated DCD



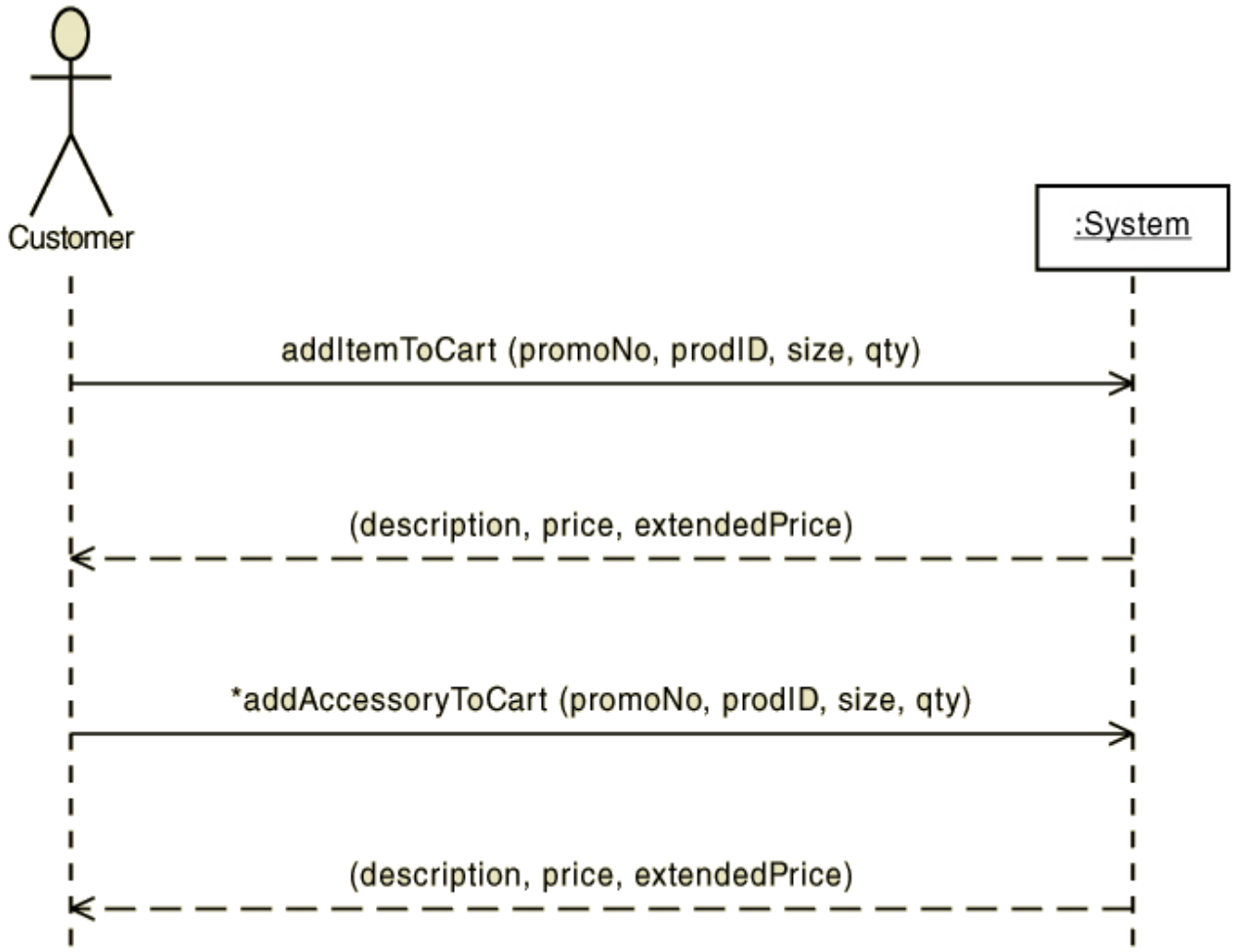
Example: *Fill Shopping Cart*

- Input activity diagram
  - Note the activity flows that cross the system boundary



Example: *Fill Shopping Cart*

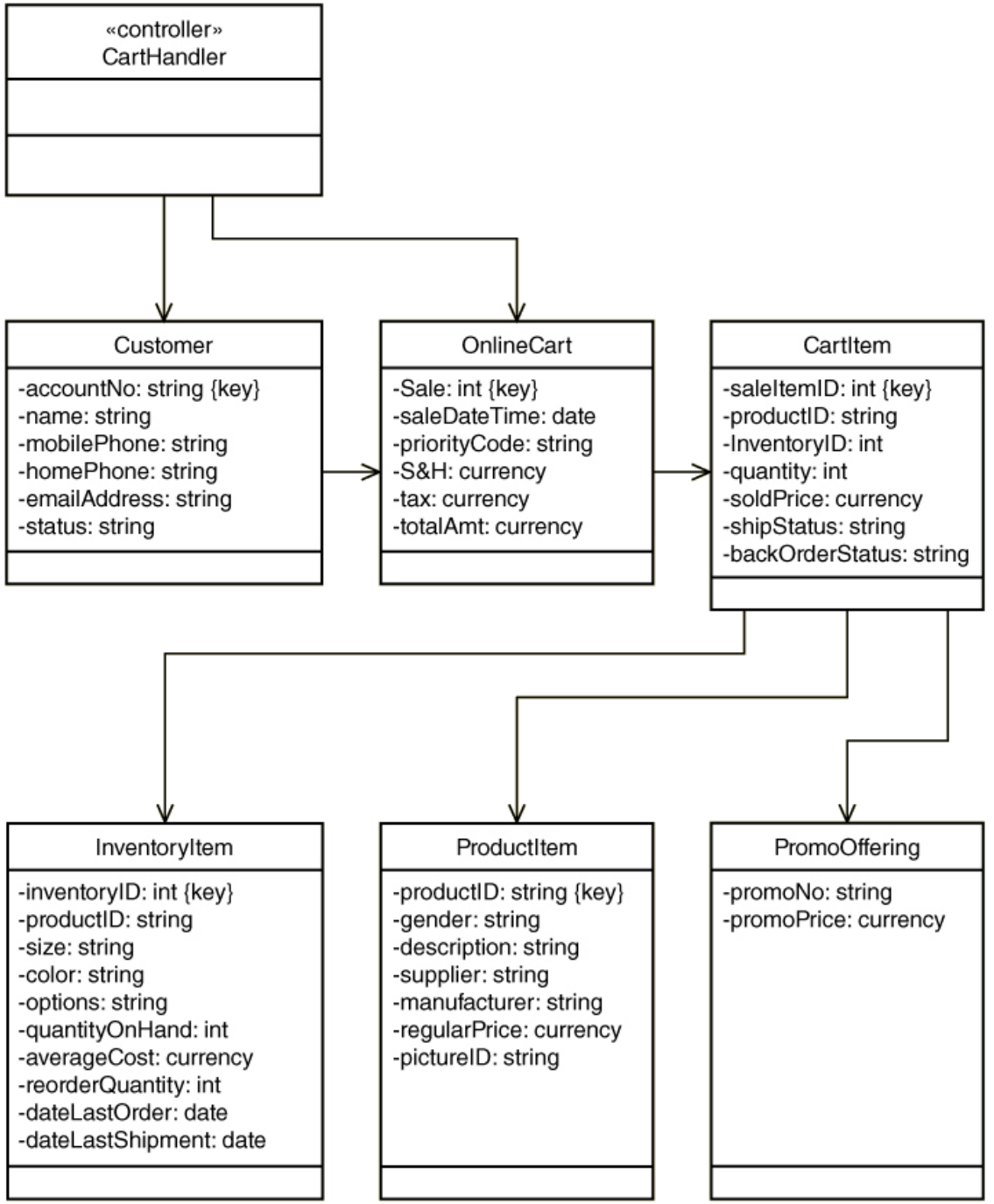
- Input: SSD
  - Note the input and return messages
  - Note the repeating message



If the item is the first item we need create a cart object. But cart does not exist Without a customer so the message should be passed to customer. The customer object is responsible of creating the cart object.

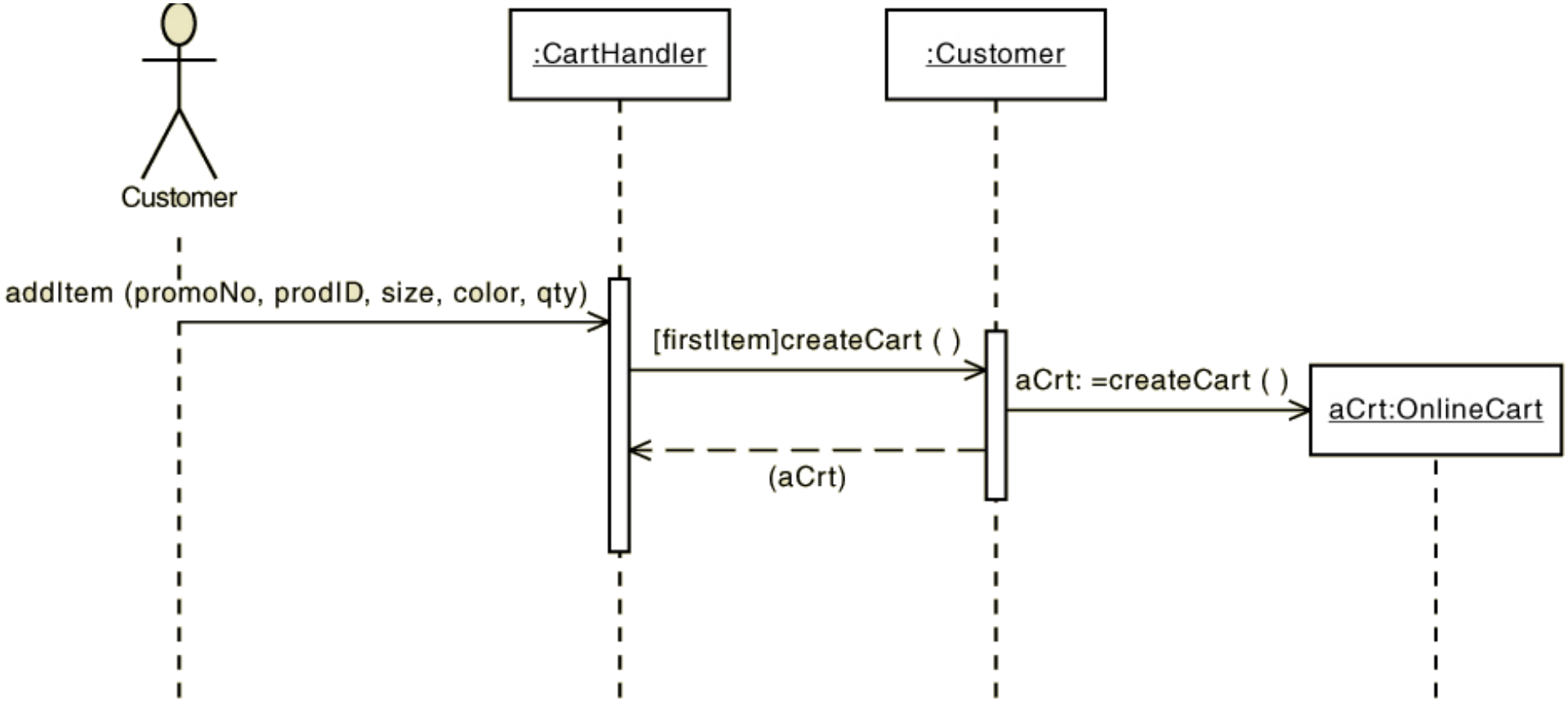
Example:
Fill Shopping Cart

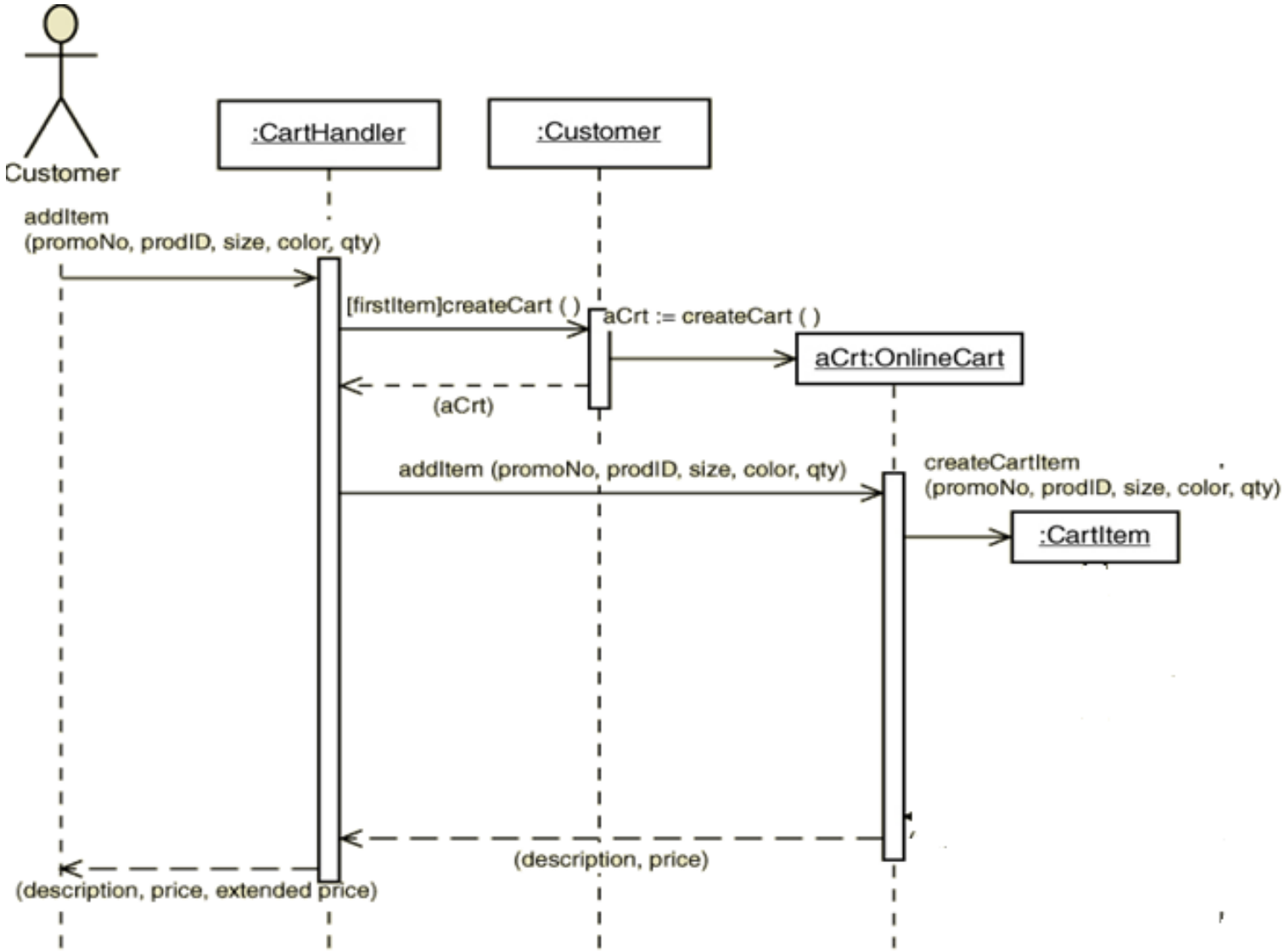
- First-Cut DCD

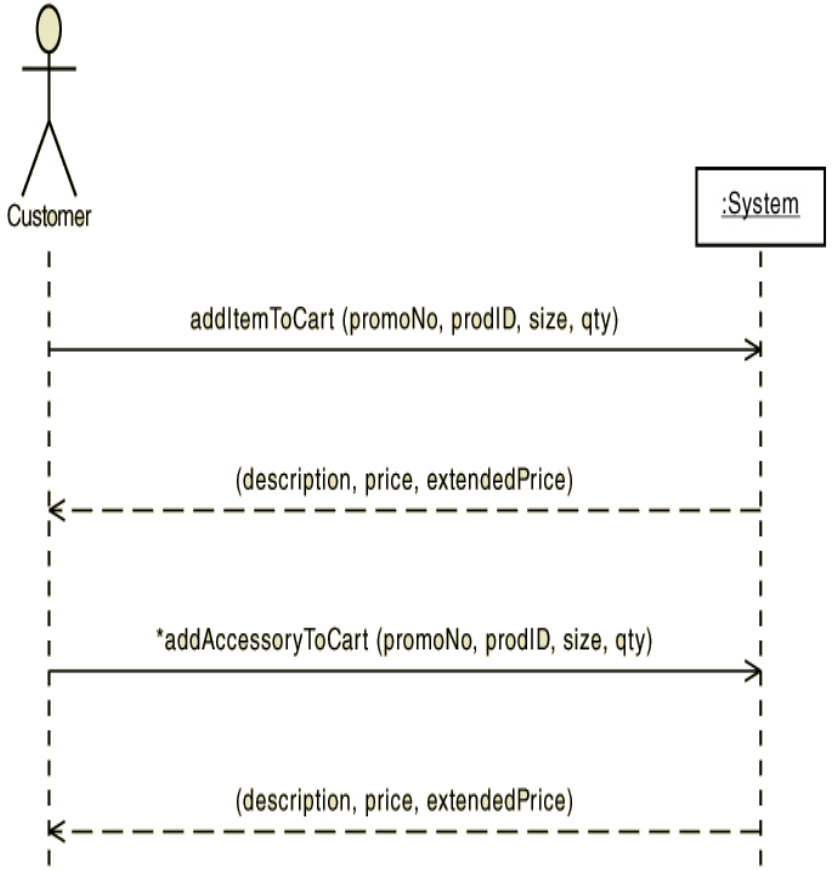


Example: *Fill Shopping Cart*

- First step in *addItem* message. Create a cart if first item in the purchase
  - Note true/false test, two types of return values, named objects, activation lifelines, message origins and destinations

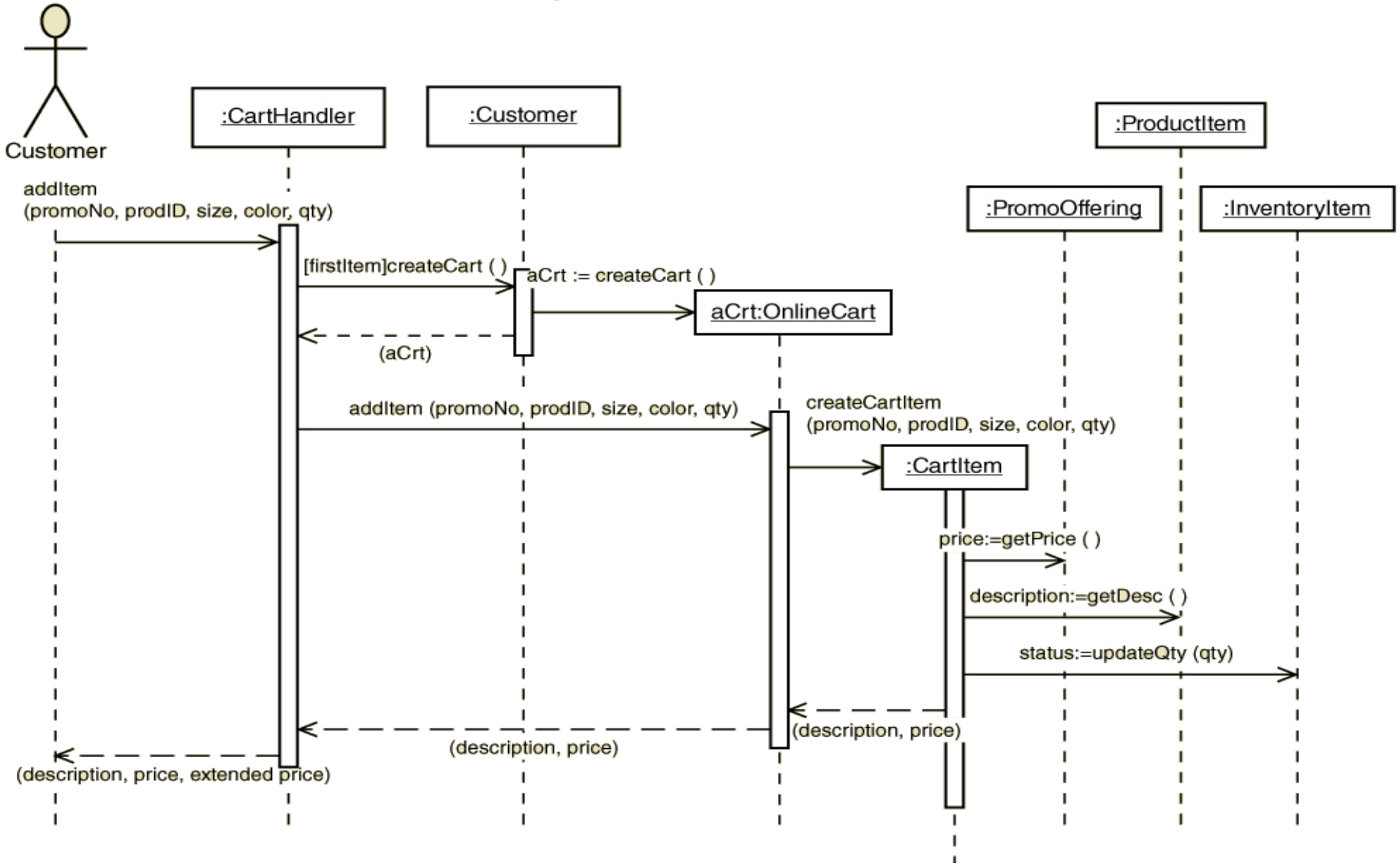



Customer
:CartHandler
:Customer
addItem
(promoNo, prodID, size, color, qty)
[firstItem]createCart ( )
aCrt := createCart ( )
aCrt:OnlineCart
(aCrt)
addItem (promoNo, prodID, size, color, qty)
createCartItem
(promoNo, prodID, size, color, qty)
:CartItem
(description, price)
(description, price, extended price)


Customer
:System
addItemToCart (promoNo, prodID, size, qty)
(description, price, extendedPrice)
*addAccessoryToCart (promoNo, prodID, size, qty)
(description, price, extendedPrice)

Example: *Fill Shopping Cart*

- Completion of *addItem* message.

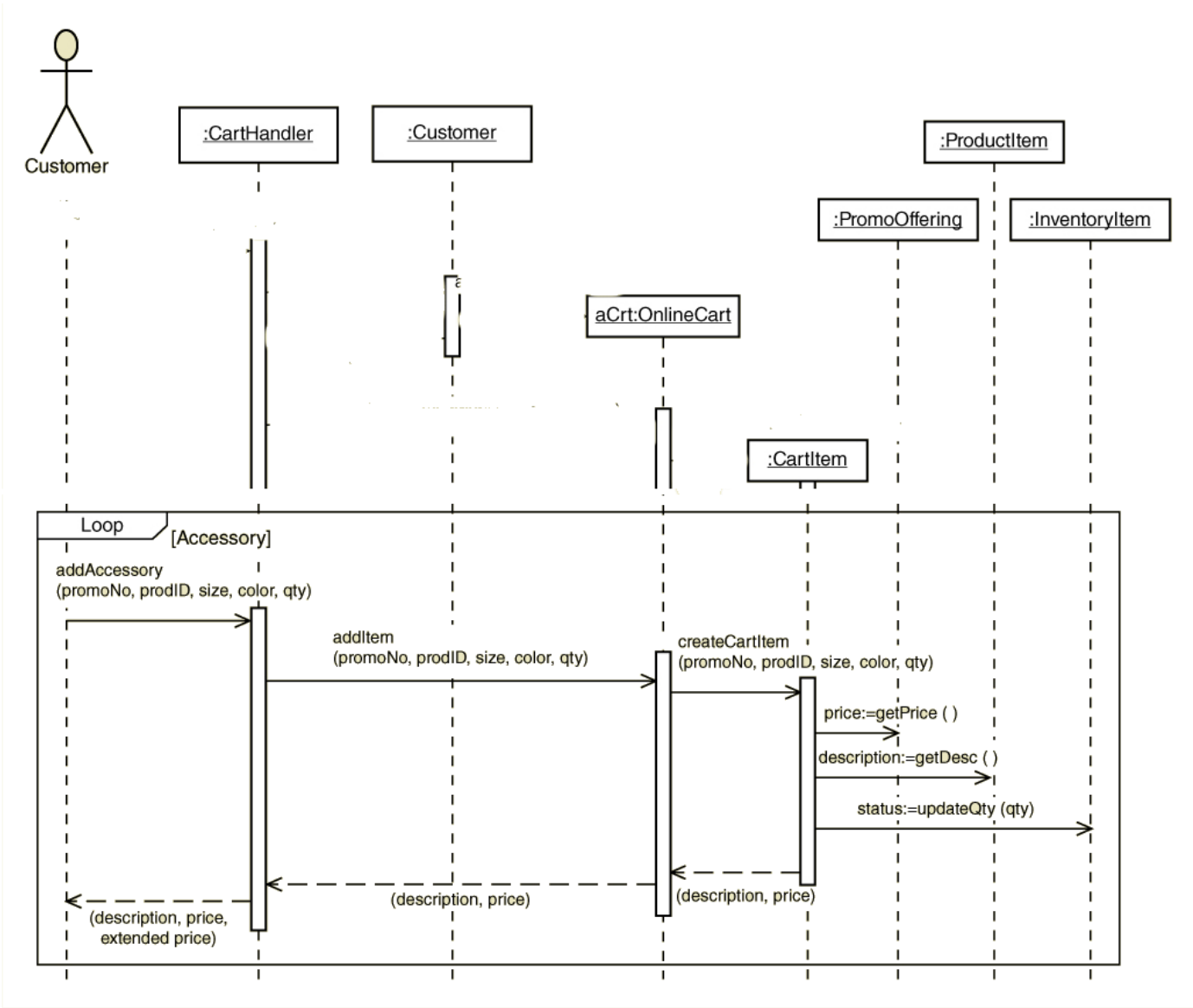


Example: *Fill Shopping Cart*

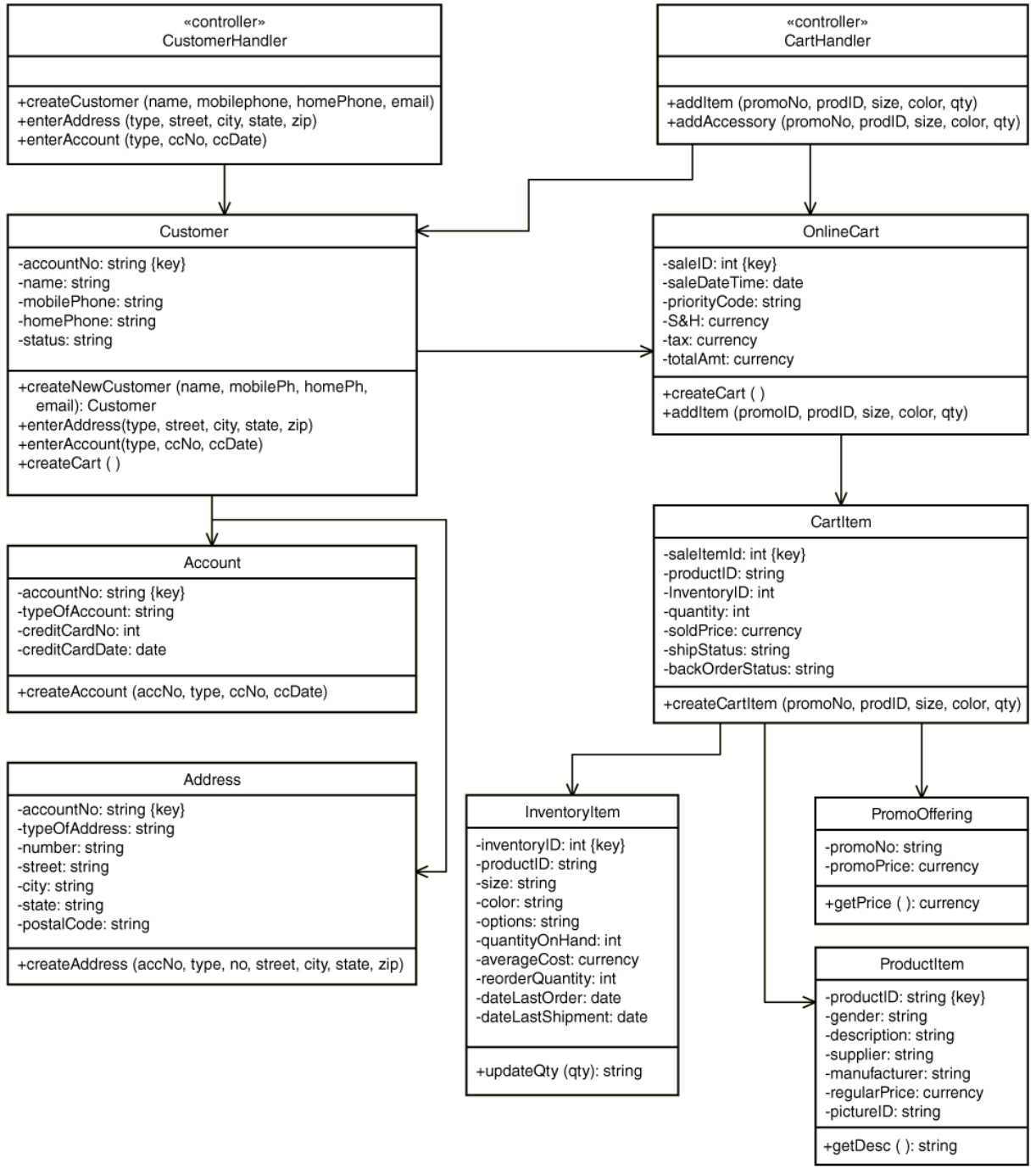
- Note origin and destination objects and visibility
  - createCart () – cart handler knows if first item
  - aCrt:=createCart() – customer owns onlineCart
  - addItem() – forwarded message
  - createCartItem() – cartItem responsible for creating itself and getting values
  - getPrice() – just returns the price
  - getDescription() – just returns description
  - updateQty(qty) – initiates updates

Example: *Fill Shopping Cart*

- *addAccesory* message (with objects and lifelines)



Updated DCD


«controller»
CustomerHandler
+createCustomer (name, mobilephone, homePhone, email)
+enterAddress (type, street, city, state, zip)
+enterAccount (type, ccNo, ccDate)
«controller»
CartHandler
+addItem (promoNo, prodID, size, color, qty)
+addAccessory (promoNo, prodID, size, color, qty)
Customer
-accountNo: string {key}
-name: string
-mobilePhone: string
-homePhone: string
-status: string
+createNewCustomer (name, mobilePh, homePh, email): Customer
+enterAddress(type, street, city, state, zip)
+enterAccount(type, ccNo, ccDate)
+createCart ( )
OnlineCart
-saleID: int {key}
-saleDateTime: date
-priorityCode: string
-S&H: currency
-tax: currency
-totalAmt: currency
+createCart ( )
+addItem (promoID, prodID, size, color, qty)
Account
-accountNo: string {key}
-typeOfAccount: string
-creditCardNo: int
-creditCardDate: date
+createAccount (accNo, type, ccNo, ccDate)
CartItem
-saleItemId: int {key}
-productID: string
-InventoryID: int
-quantity: int
-soldPrice: currency
-shipStatus: string
-backOrderStatus: string
+createCartItem (promoNo, prodID, size, color, qty)
Address
-accountNo: string {key}
-typeOfAddress: string
-number: string
-street: string
-city: string
-state: string
-postalCode: string
+createAddress (accNo, type, no, street, city, state, zip)
InventoryItem
-inventoryID: int {key}
-productID: string
-size: string
-color: string
-options: string
-quantityOnHand: int
-averageCost: currency
-reorderQuantity: int
-dateLastOrder: date
-dateLastShipment: date
+updateQty (qty): string
PromoOffering
-promoNo: string
-promoPrice: currency
+getPrice ( ): currency
ProductItem
-productID: string {key}
-gender: string
-description: string
-supplier: string
-manufacturer: string
-regularPrice: currency
-pictureID: string
+getDesc ( ): string

Guidelines

- From each input message
  - Determine all internal messages
  - Determine all objects (from classes) as origins or destinations
  - Flesh out each message with true/false conditions, parameters and returned values

First Cut Interaction Diagram Assumptions

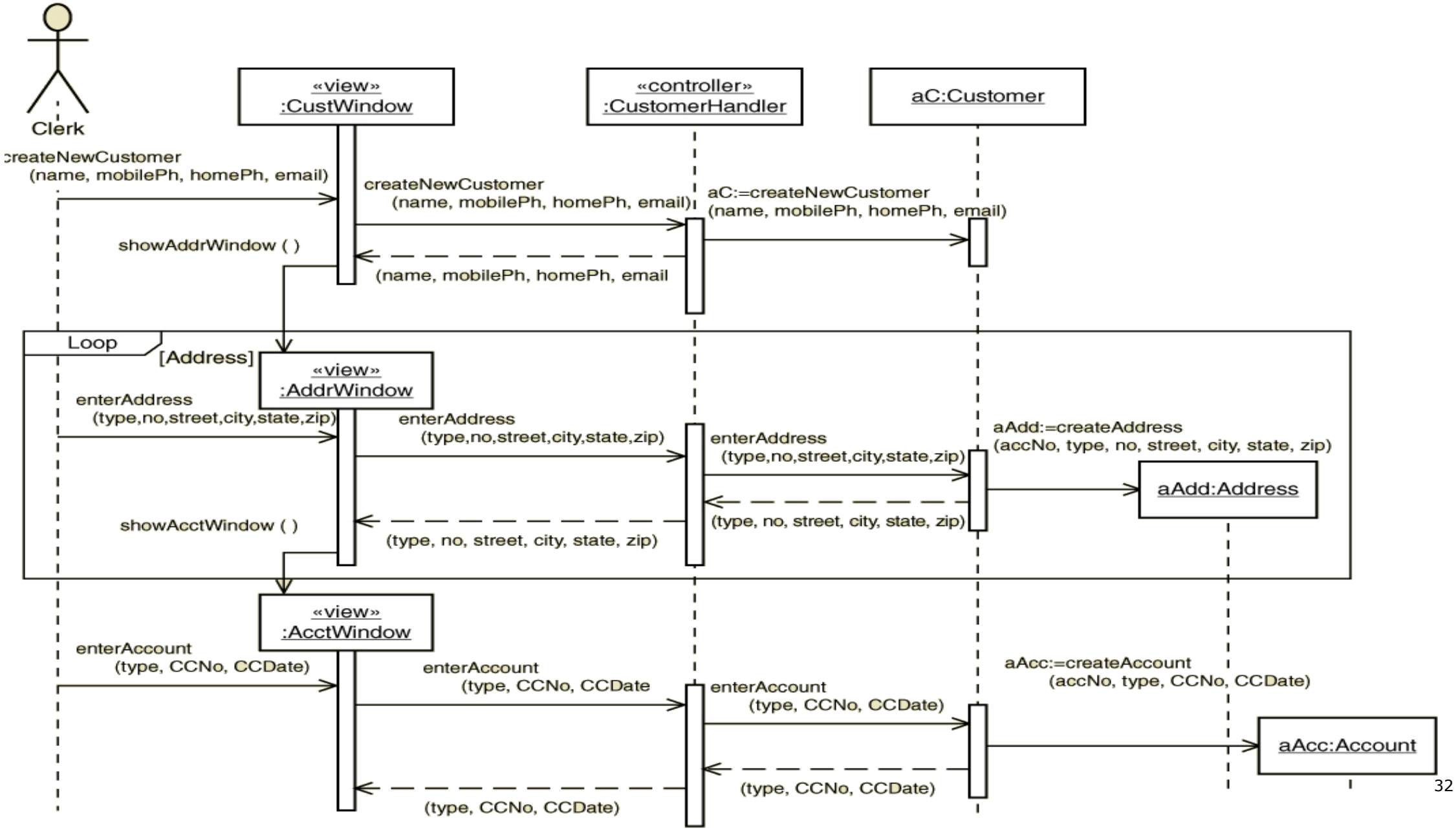
- Perfect technology assumption
  - No logon or other technical issues
- Perfect memory assumption
  - No need to read or write data
- Perfect solution assumption
  - No exception conditions, no error handling

Multilayer Sequence Diagrams

- Add the view layer for input screens to handle input messages
- Add data layer to read and write the data

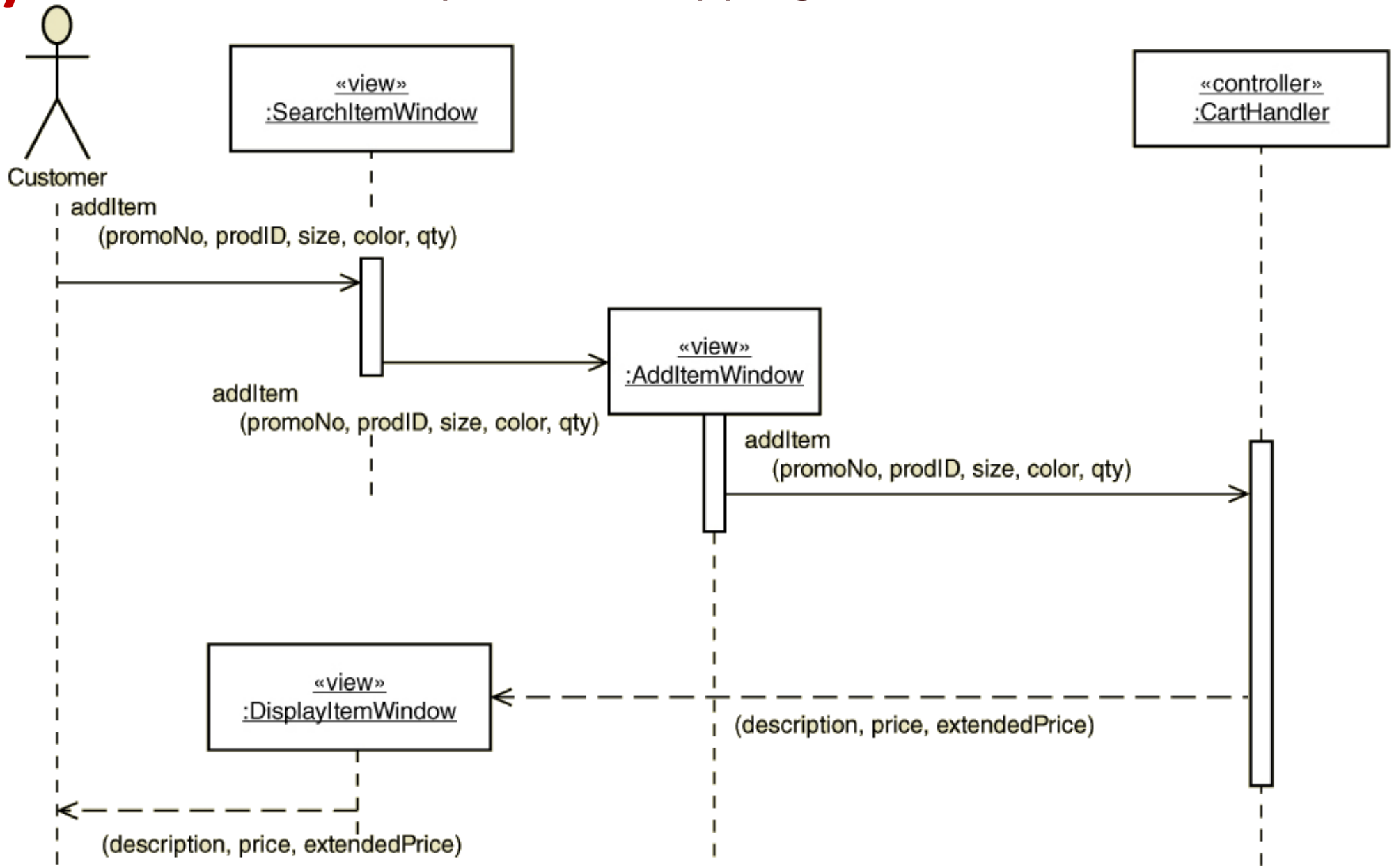
View Layer

- Example: *Add customer account* use case



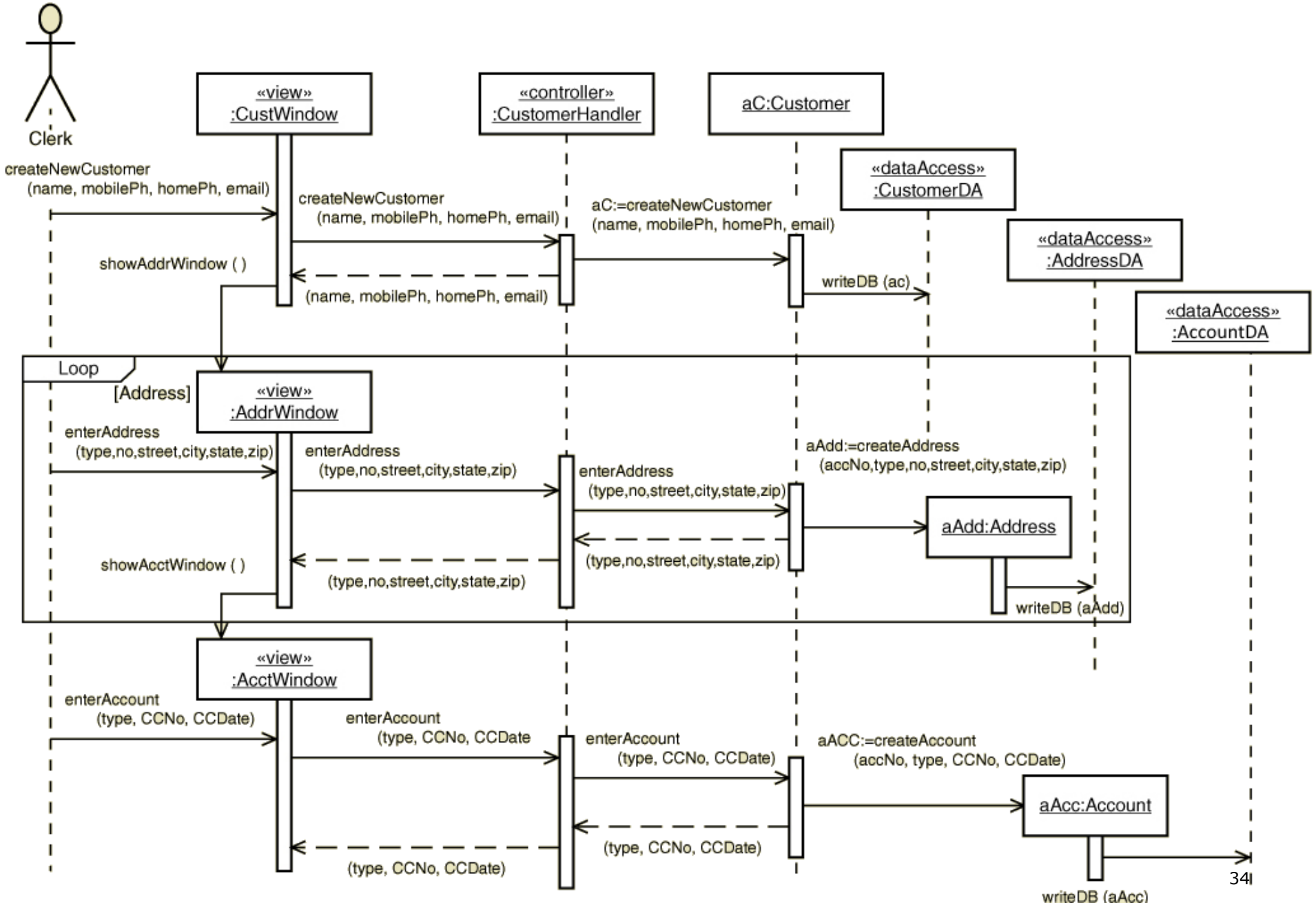
View Layer

- Partial example: *Fill shopping cart* use case



Data Layer

- Data layer for reading or updating existing persistent objects

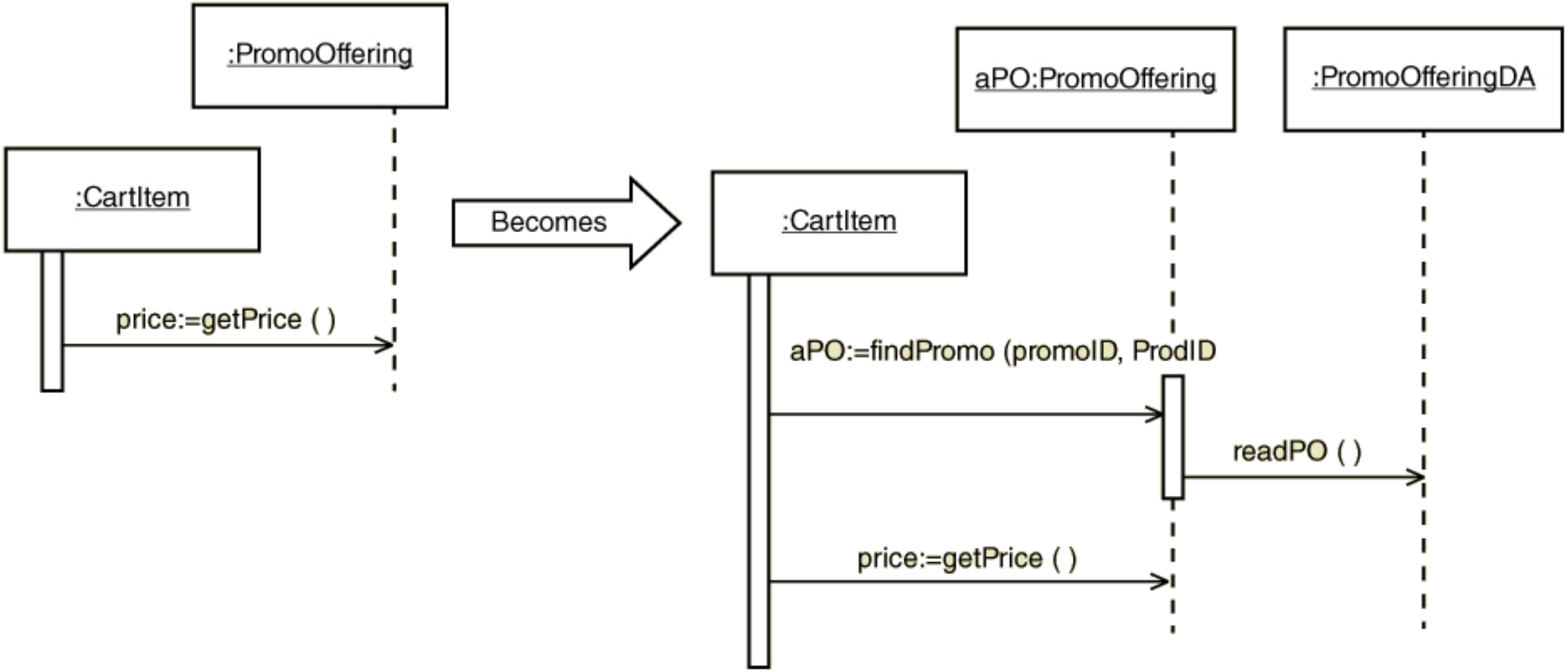


Data Layer

- Data Access to instantiate a new object – Two methods
  - Instantiate the object in memory, the object invokes the data access to get any required persistent data.
  - Send a message to the data access object, and it obtains the required persistent data and then instantiates the new object.

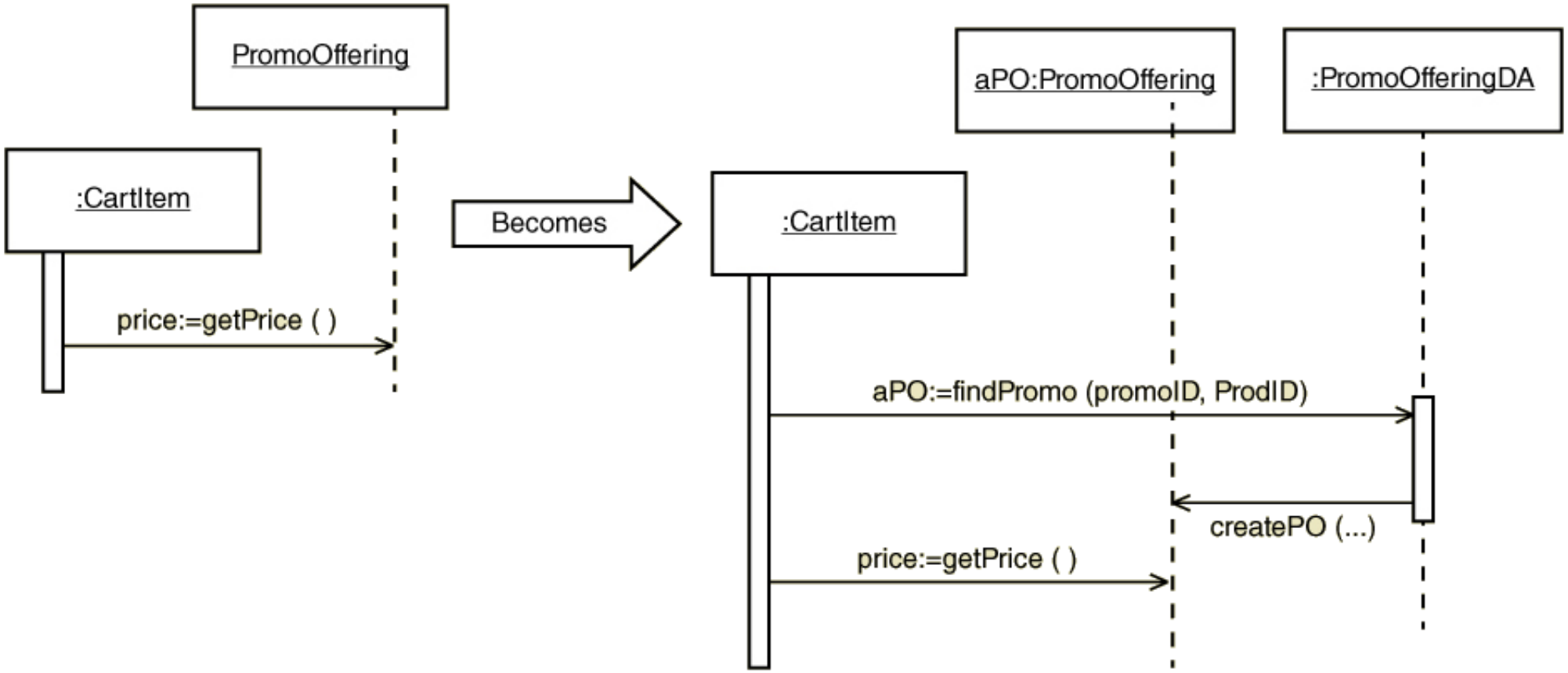
Instantiating a New Object

- Method 1. The instantiated object gets the data

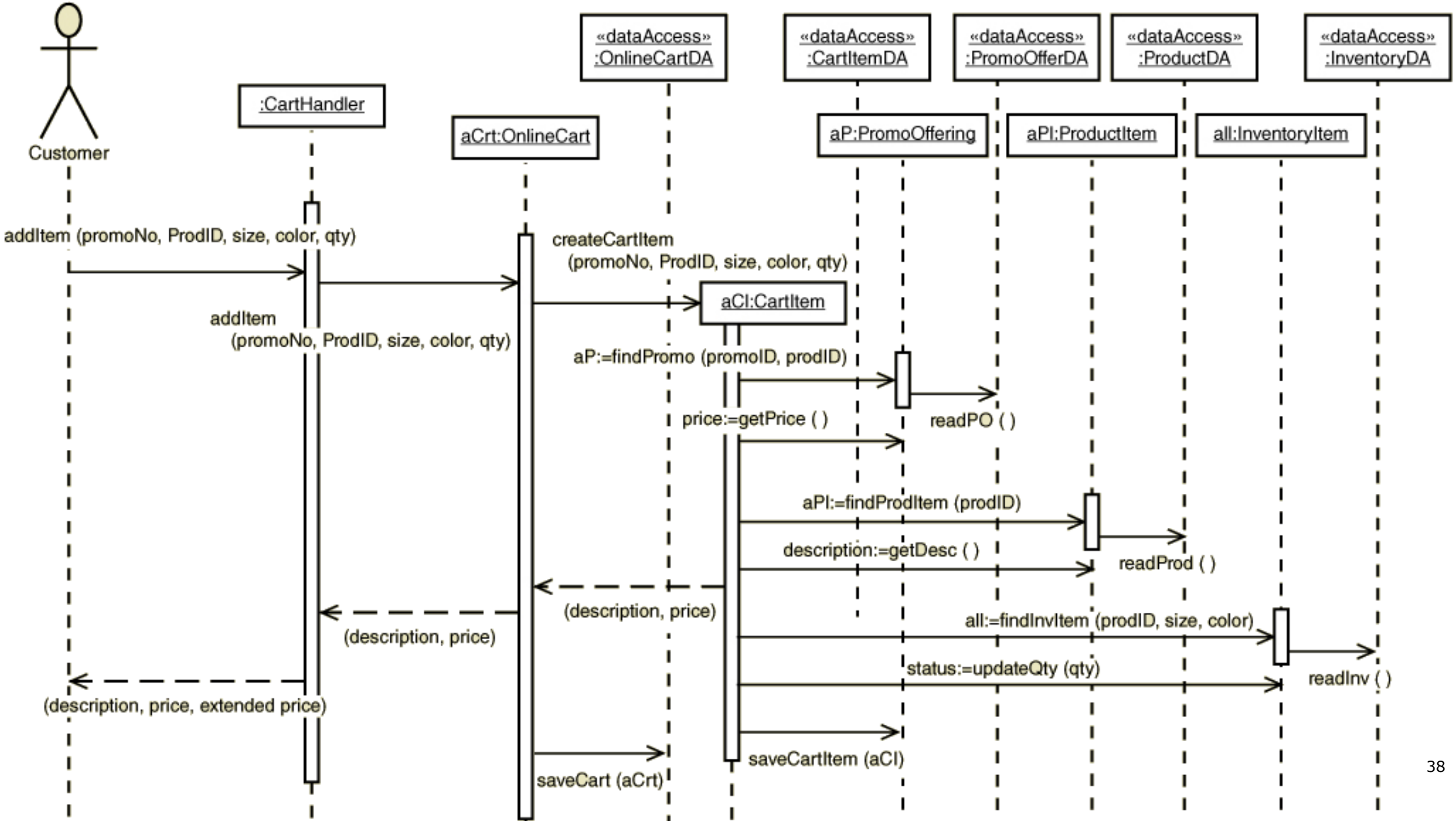


Instantiating a New Object

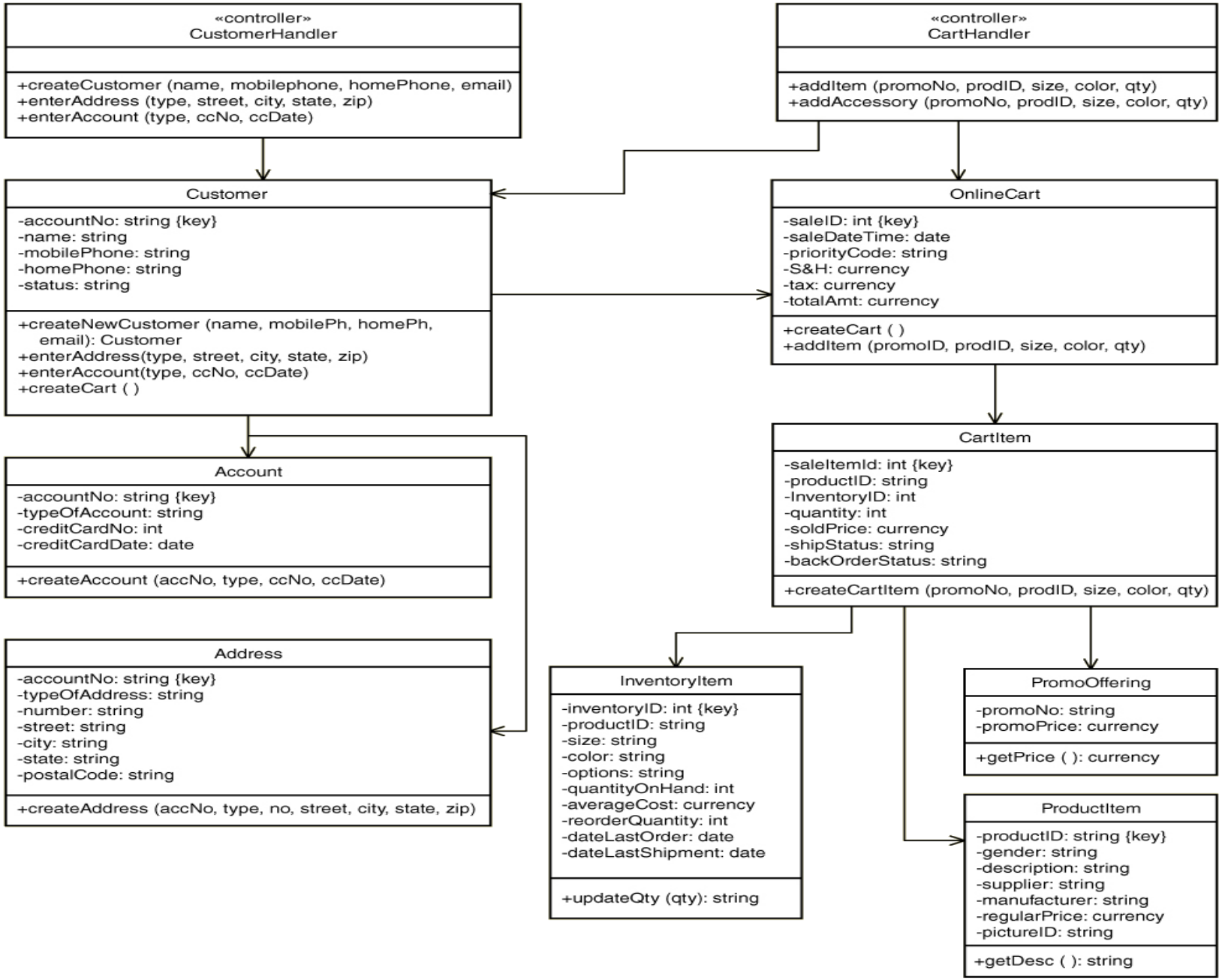
- Method 2. The data access gets the data and instantiates the object



Partial Fill shopping cart with Data Access



Updated DCD


«controller»
CustomerHandler
+createCustomer (name, mobilephone, homePhone, email)
+enterAddress (type, street, city, state, zip)
+enterAccount (type, ccNo, ccDate)
«controller»
CartHandler
+addItem (promoNo, prodID, size, color, qty)
+addAccessory (promoNo, prodID, size, color, qty)
Customer
-accountNo: string {key}
-name: string
-mobilePhone: string
-homePhone: string
-status: string
+createNewCustomer (name, mobilePh, homePh, email): Customer
+enterAddress(type, street, city, state, zip)
+enterAccount(type, ccNo, ccDate)
+createCart ( )
OnlineCart
-saleID: int {key}
-saleDateTime: date
-priorityCode: string
-S&H: currency
-tax: currency
-totalAmt: currency
+createCart ( )
+addItem (promoID, prodID, size, color, qty)
Account
-accountNo: string {key}
-typeOfAccount: string
-creditCardNo: int
-creditCardDate: date
+createAccount (accNo, type, ccNo, ccDate)
CartItem
-saleItemId: int {key}
-productID: string
-InventoryID: int
-quantity: int
-soldPrice: currency
-shipStatus: string
-backOrderStatus: string
+createCartItem (promoNo, prodID, size, color, qty)
Address
-accountNo: string {key}
-typeOfAddress: string
-number: string
-street: string
-city: string
-state: string
-postalCode: string
+createAddress (accNo, type, no, street, city, state, zip)
InventoryItem
-inventoryID: int {key}
-productID: string
-size: string
-color: string
-options: string
-quantityOnHand: int
-averageCost: currency
-reorderQuantity: int
-dateLastOrder: date
-dateLastShipment: date
+updateQty (qty): string
PromoOffering
-promoNo: string
-promoPrice: currency
+getPrice ( ): currency
ProductItem
-productID: string {key}
-gender: string
-description: string
-supplier: string
-manufacturer: string
-regularPrice: currency
-pictureID: string
+getDesc ( ): string

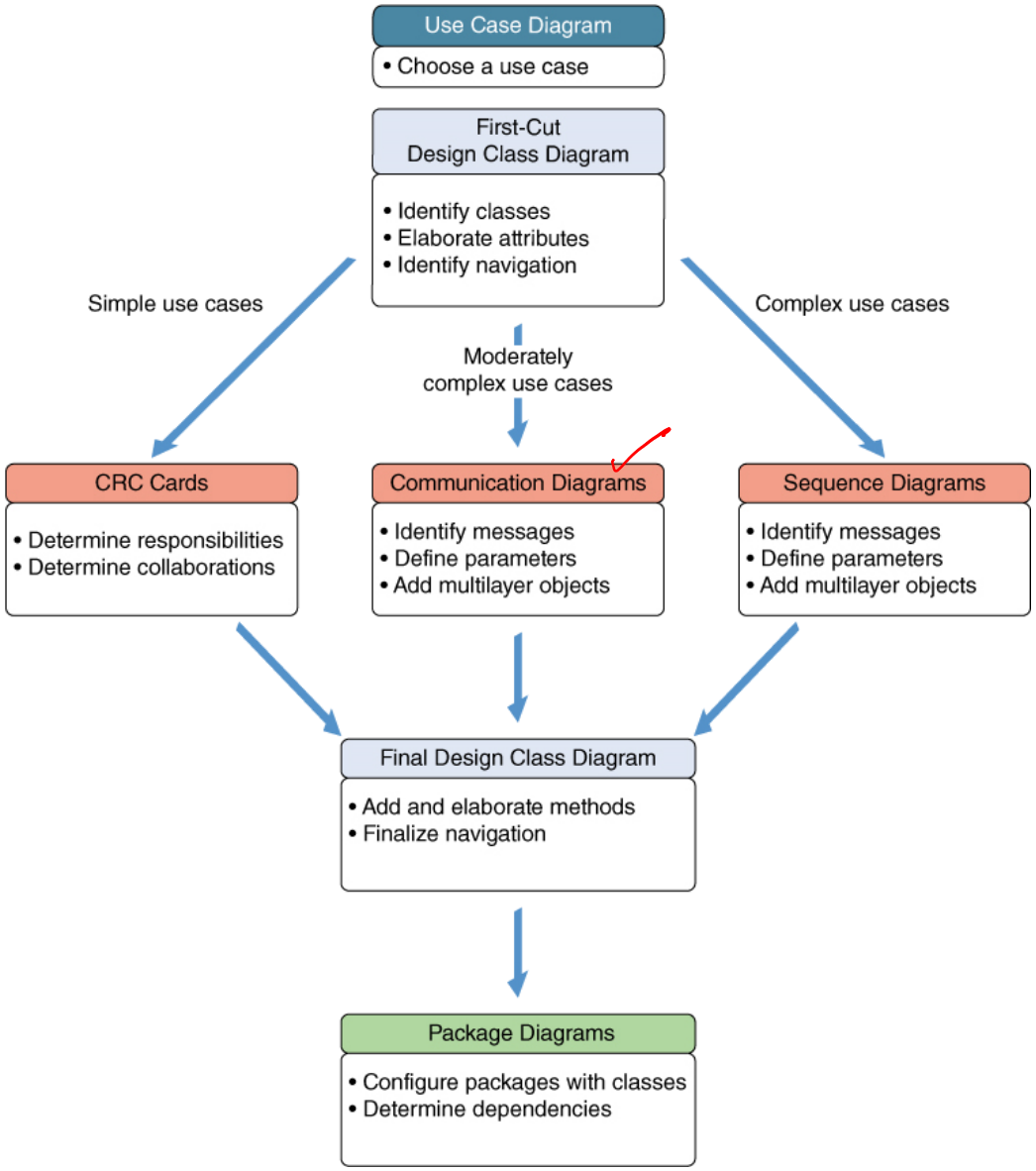
Object-Oriented Design

Use-Case Realization with Communication Diagrams

Dr. Eman Saleh eman.saleh@uga.edu

OOD

- First-cut DCD
- Extend use case with Communication Diagram or Sequence diagram
- Final DCD
- Package diagram

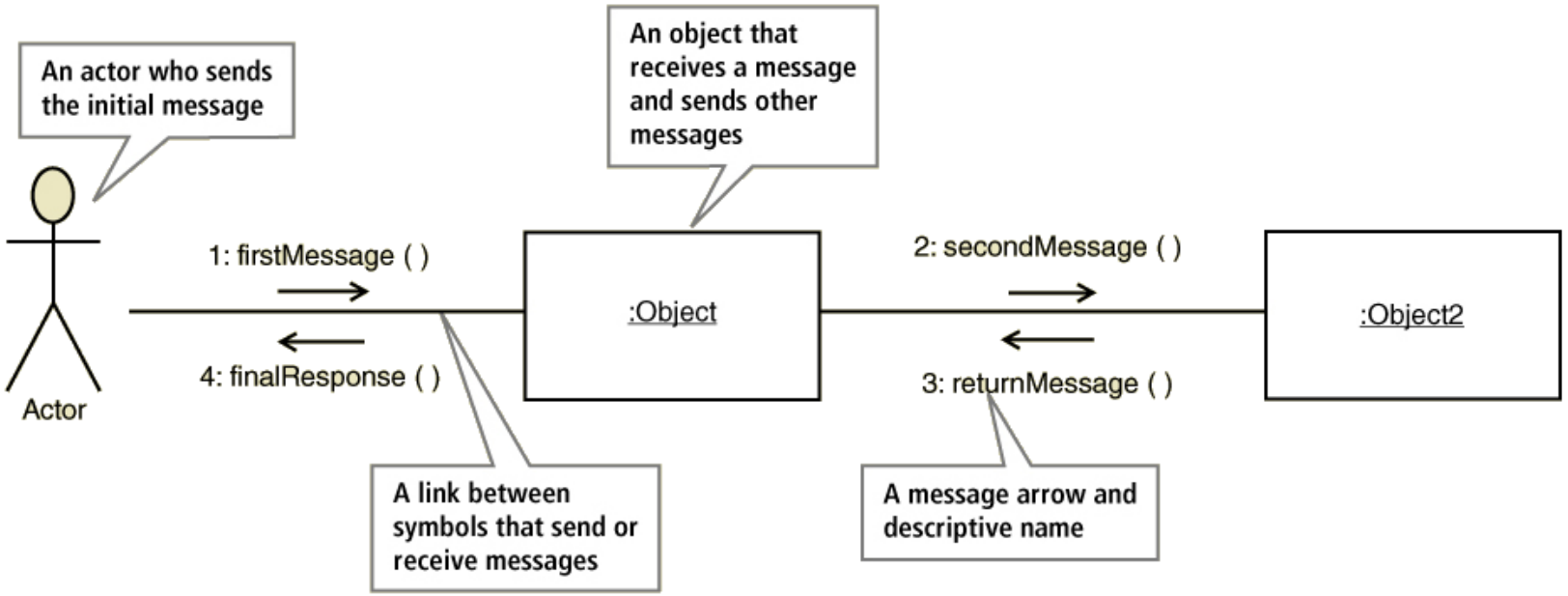


Understanding Communication Diagrams

- Actor – the external role of the person or thing that initiates the use case. Sends messages.
- Object – the instantiated class objects that perform the actions (methods) to execute the use case. They receive messages and process messages.
- Link – simply connectors between objects to carry the messages.
- Message – the requests for service with an originating actor or object and a destination object, which performs the requested service

Understanding Communication Diagrams

- Symbols used in a communication diagram:

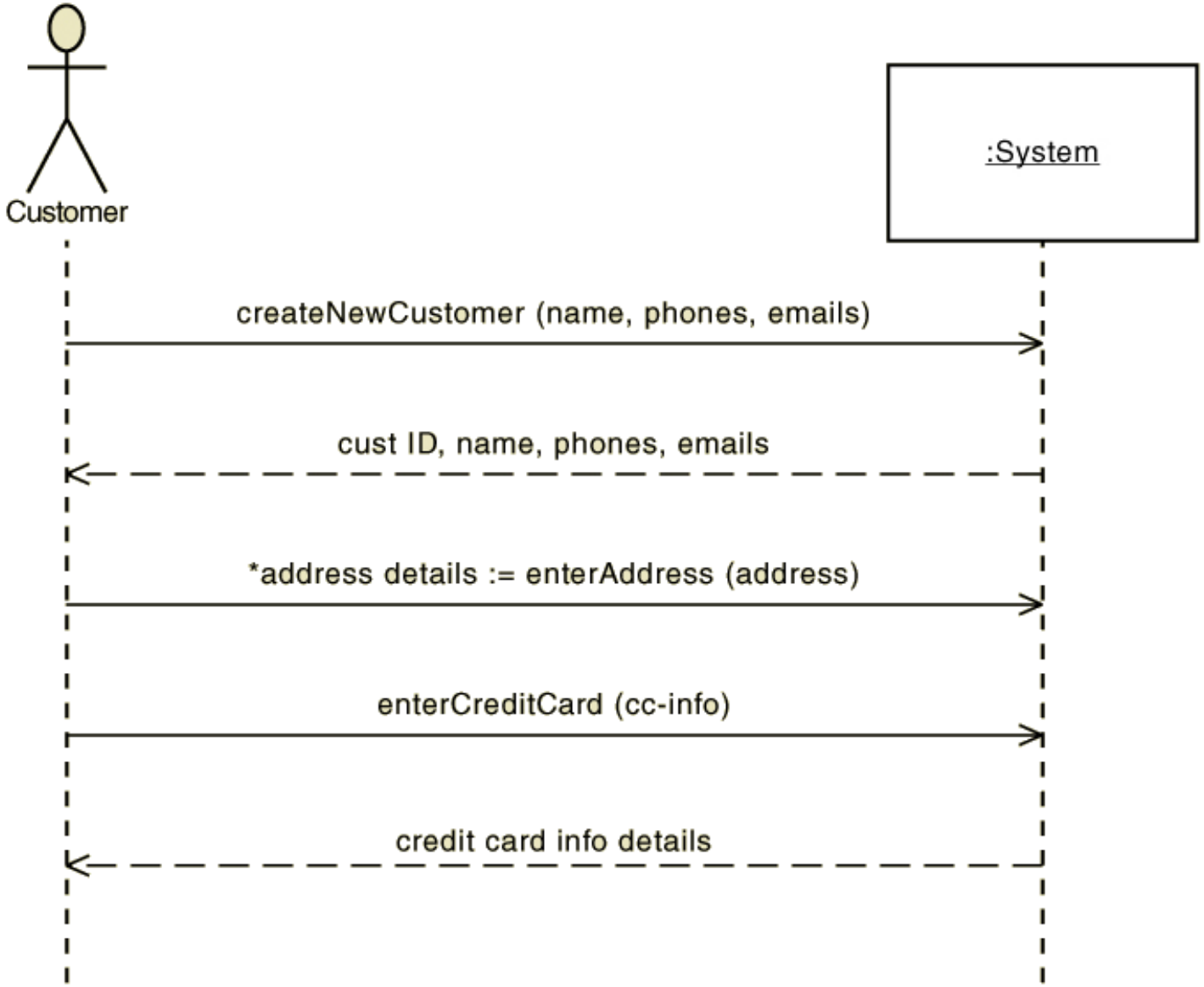


Message Syntax

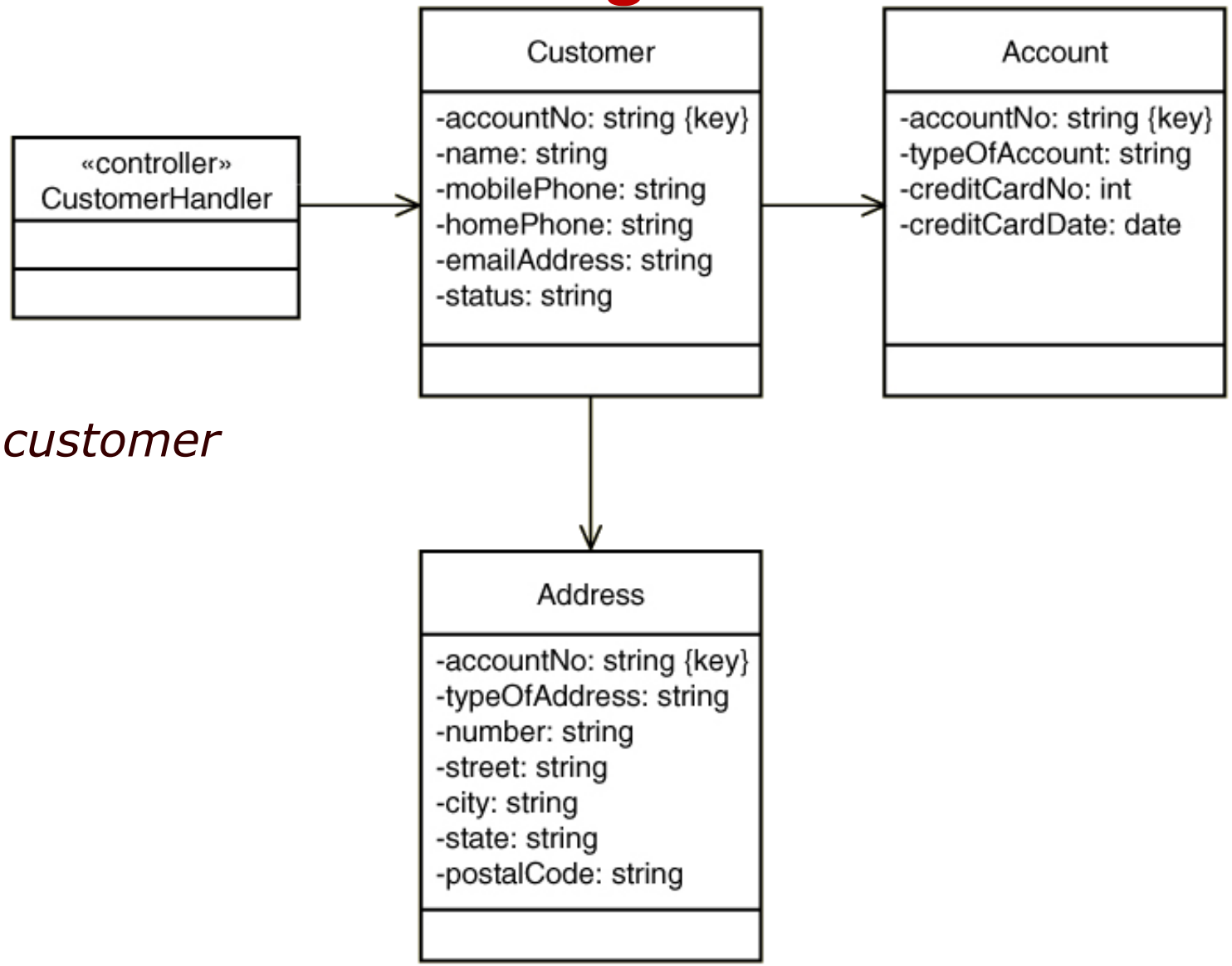
- ***[true/false condition] sequence-number: return-value: = message-name (parameter-list)***
  - true/false condition – determines if message is sent
  - sequence number – notes the order of the messages
  - return-value – value coming back to origin object from the destination object
  - message-name – camelCase name identifier. Reflects the requested service
  - parameter-list – the arguments being passed to the destination object
- Every element of a message is optional

OOD for *Create customer account*

- Input model -- SSD



OOD with Communication Diagrams



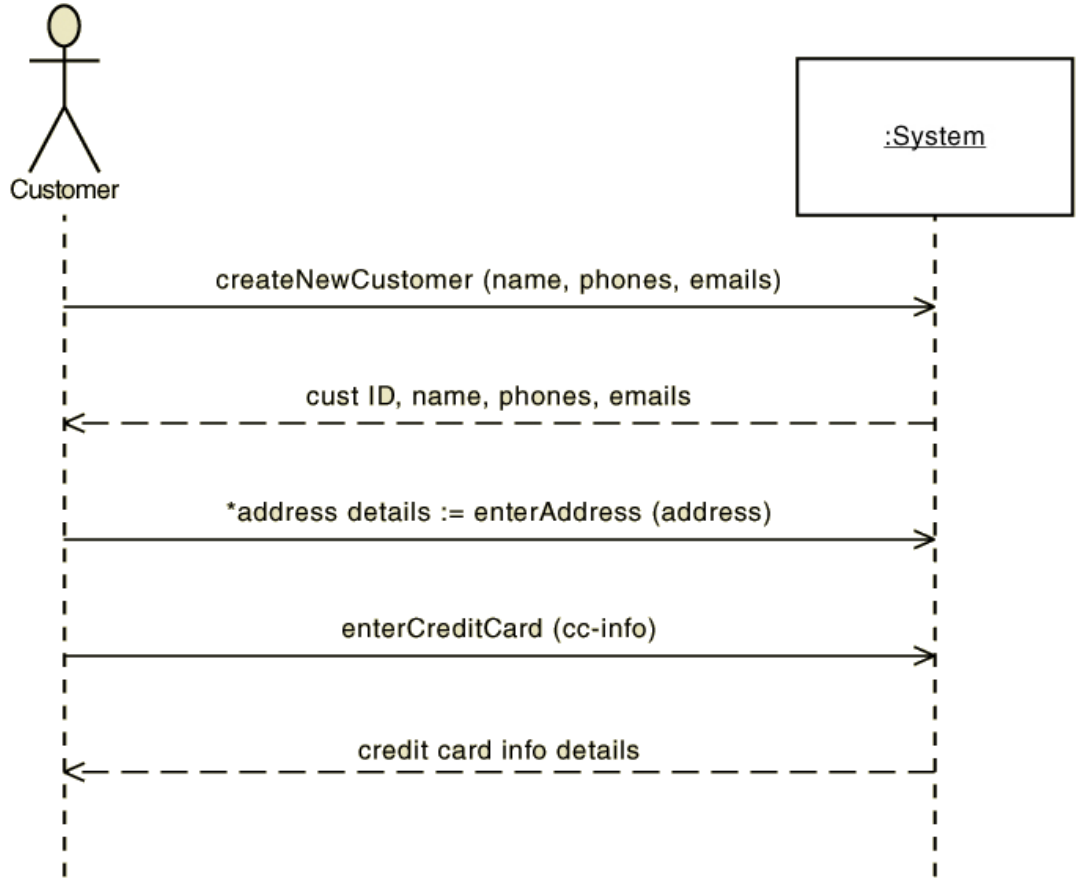
- First-cut DCD for *Create customer account* use case

OOD for *Create customer account*

- Extend input messages
  1. From the DCD put objects on communication diagram
  2. For each message, find primary object, ensure visibility, elaborate use case with all messages between objects
  3. Name each message and add all required message elements

Developing the communication diagram for the use case *Create customer account*

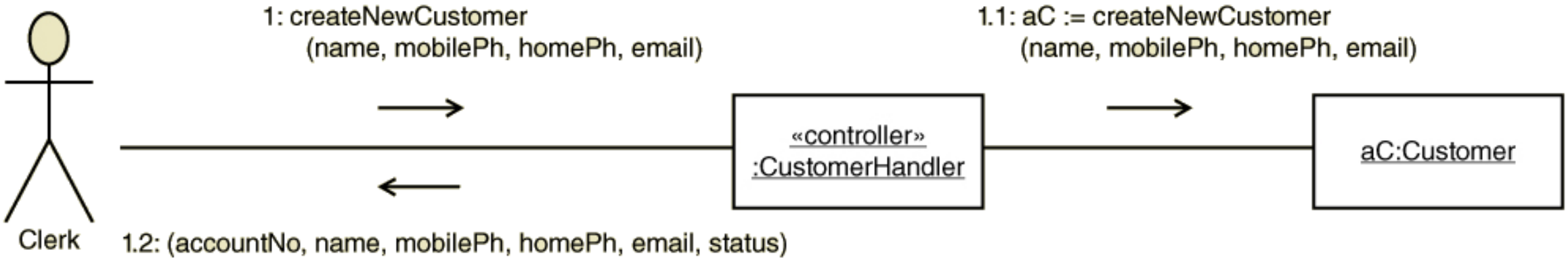
The input is a use case description, the activity diagram, or the system sequence diagram of the use case



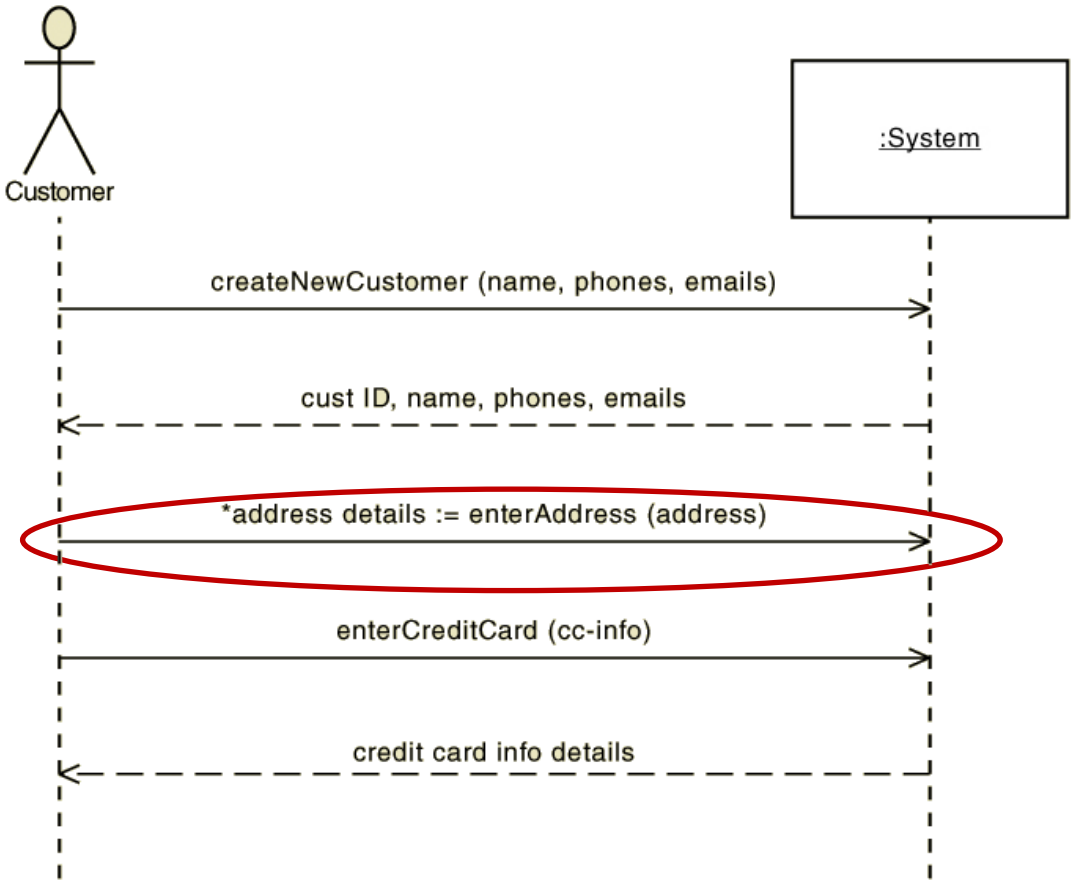
- *In the next three slides, I will draw the communication diagram messages one by one, each on a diagram, for illustration purposes.*

OOD with communication diagrams

- *createNewCustomer* message extended to all objects



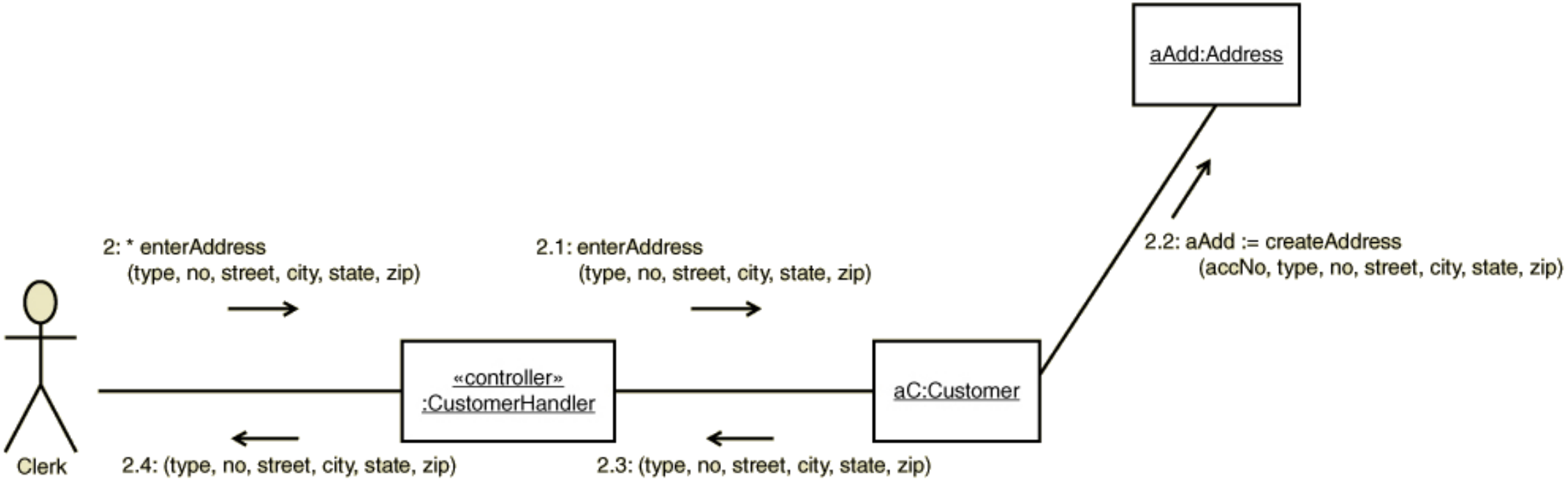
Developing the communication diagram for the use case *Create customer account*



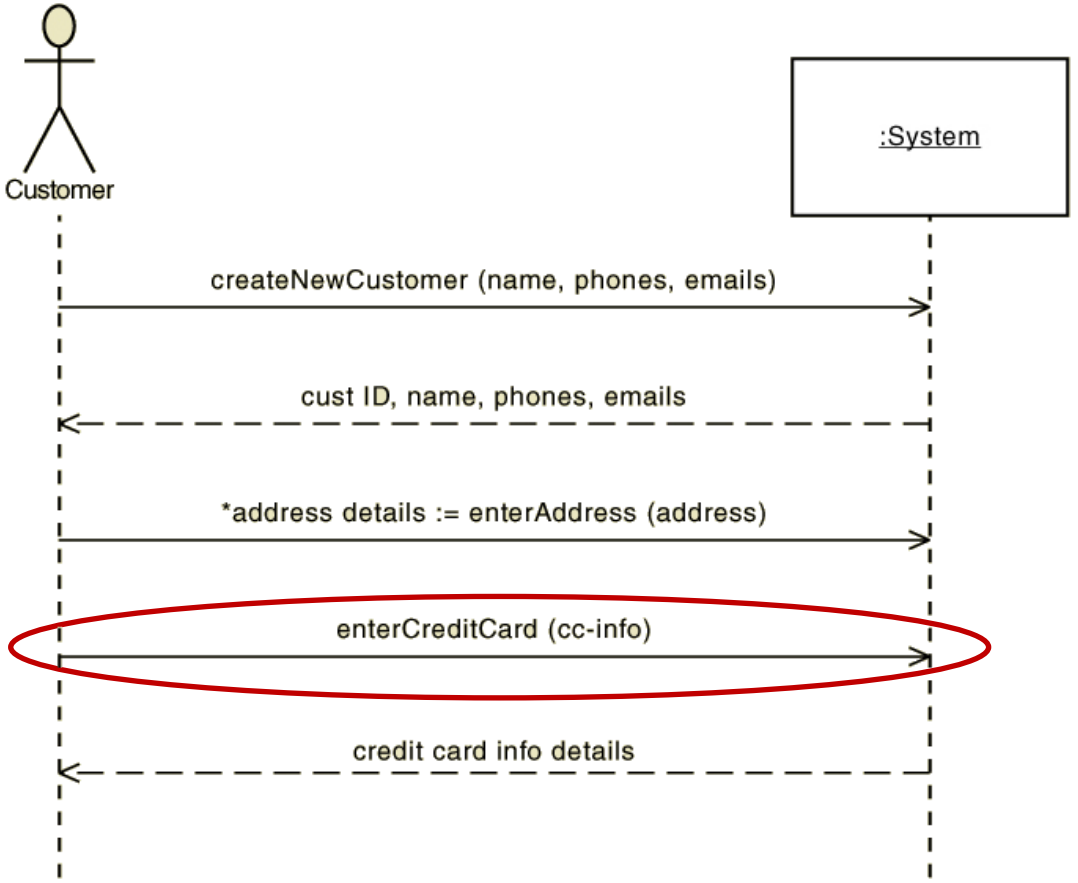
- *The second message*

OOD with communication diagrams

- *enterAddress* message extended to all objects



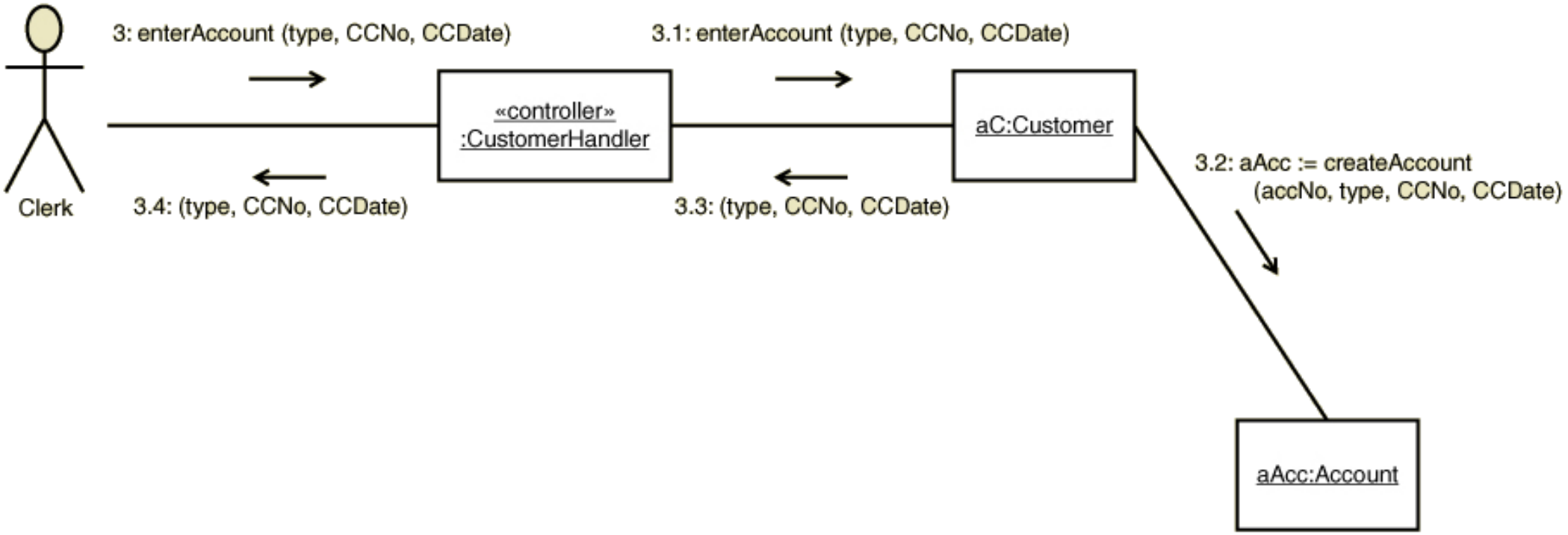
Developing the communication diagram for the use case *Create customer account*



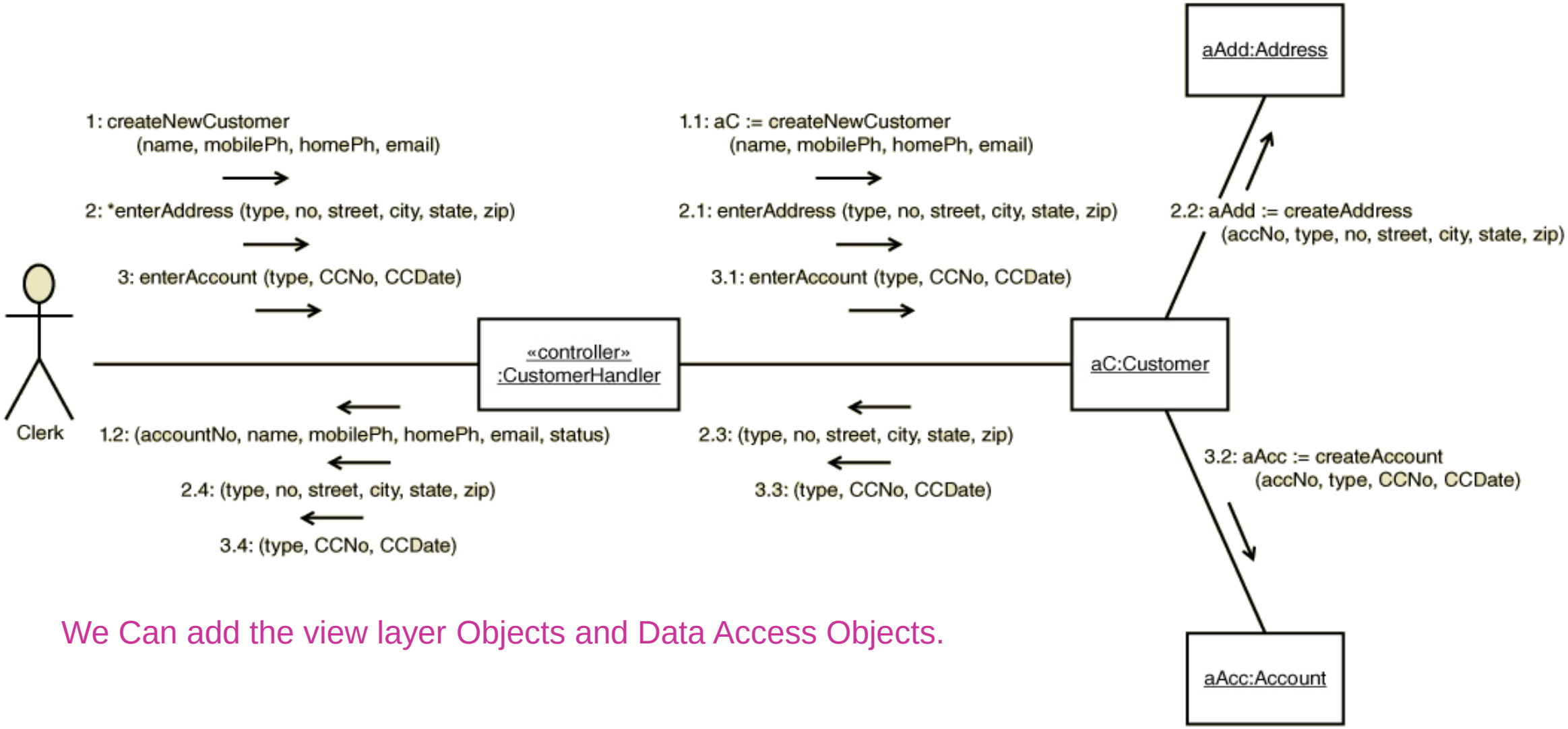
- *The third message*

OOD with communication diagrams

- *enterAccount* message extended to all objects



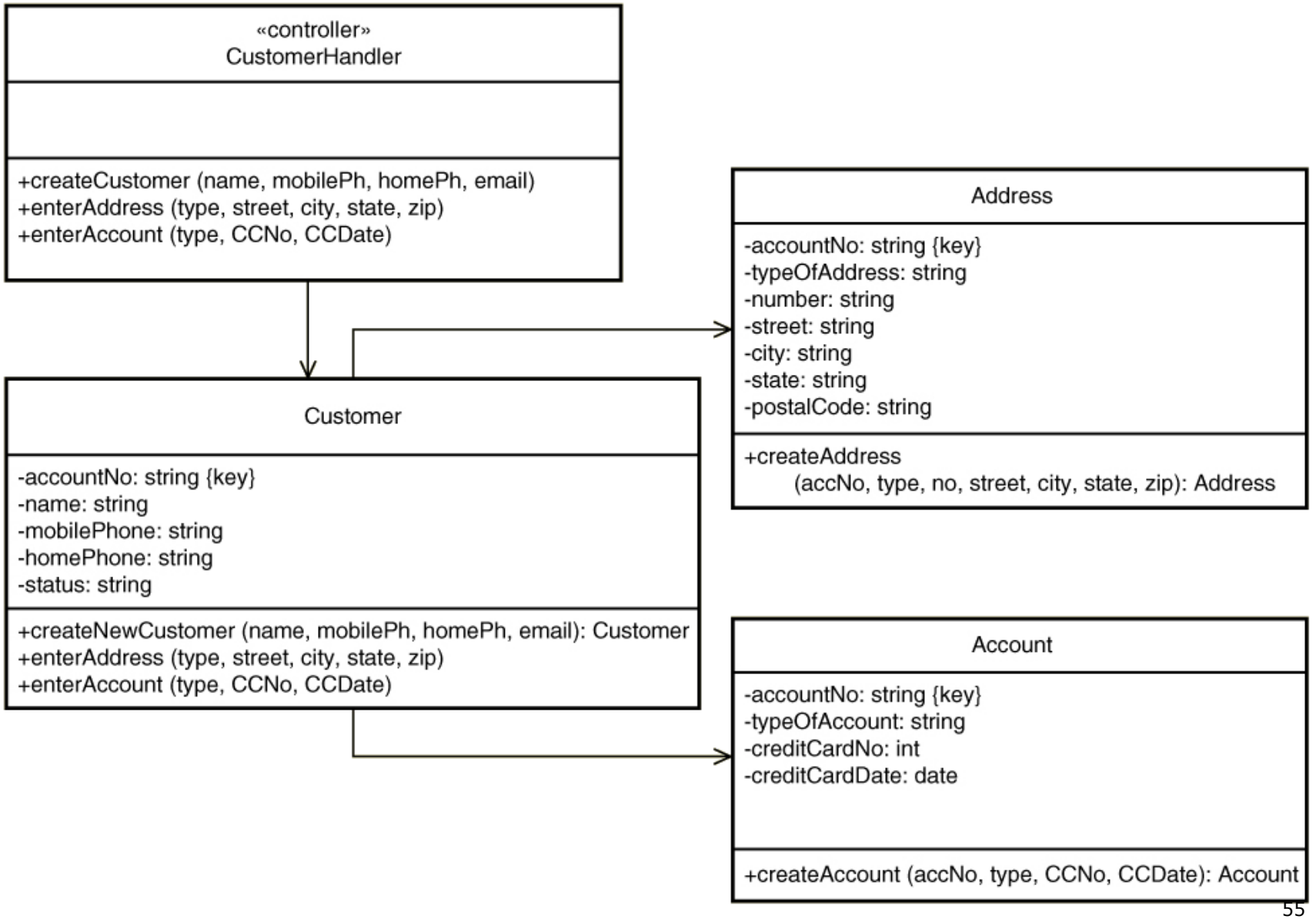
- Final communication diagram with all messages



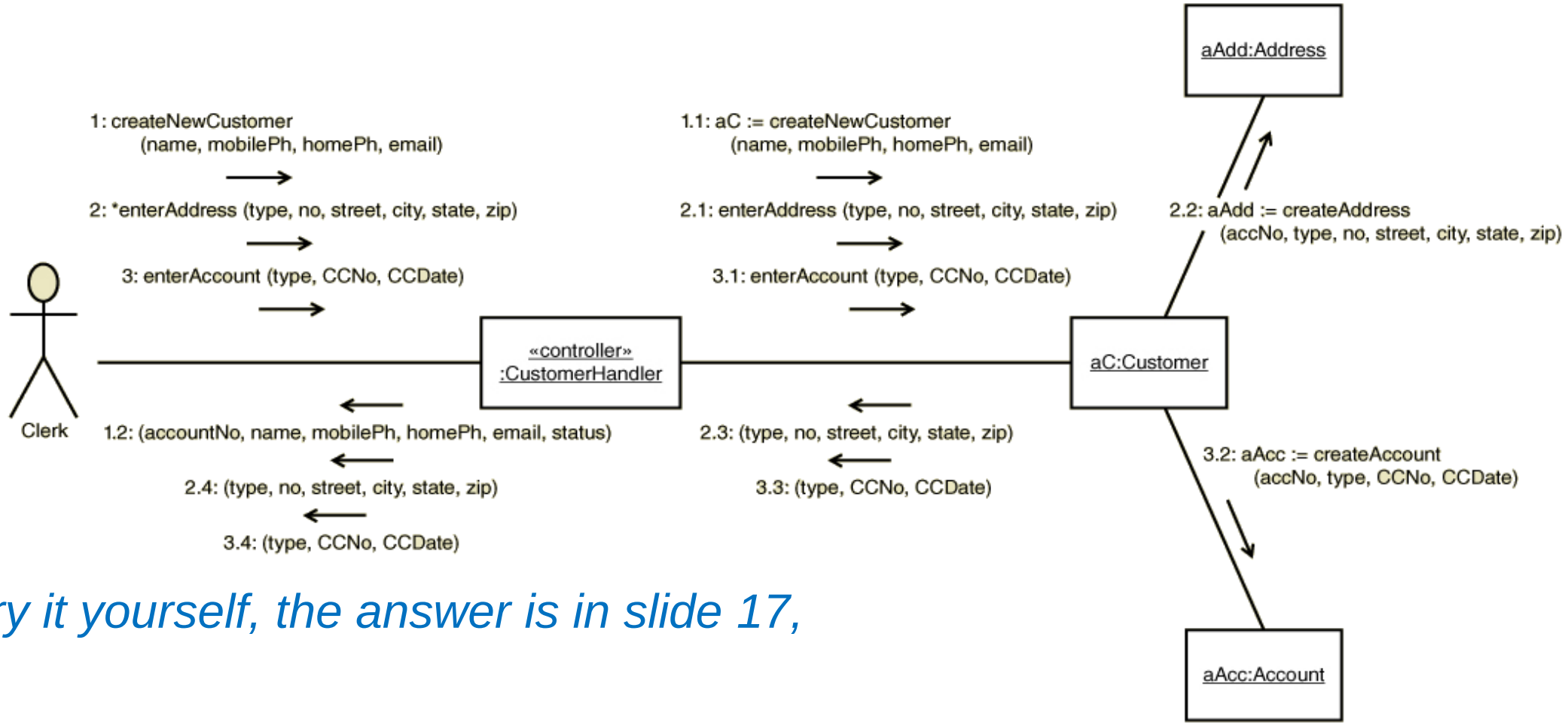
We Can add the view layer Objects and Data Access Objects.

OOD with communication diagrams

- **Updated DCD**



Convert this communication diagram to a sequence Diagram



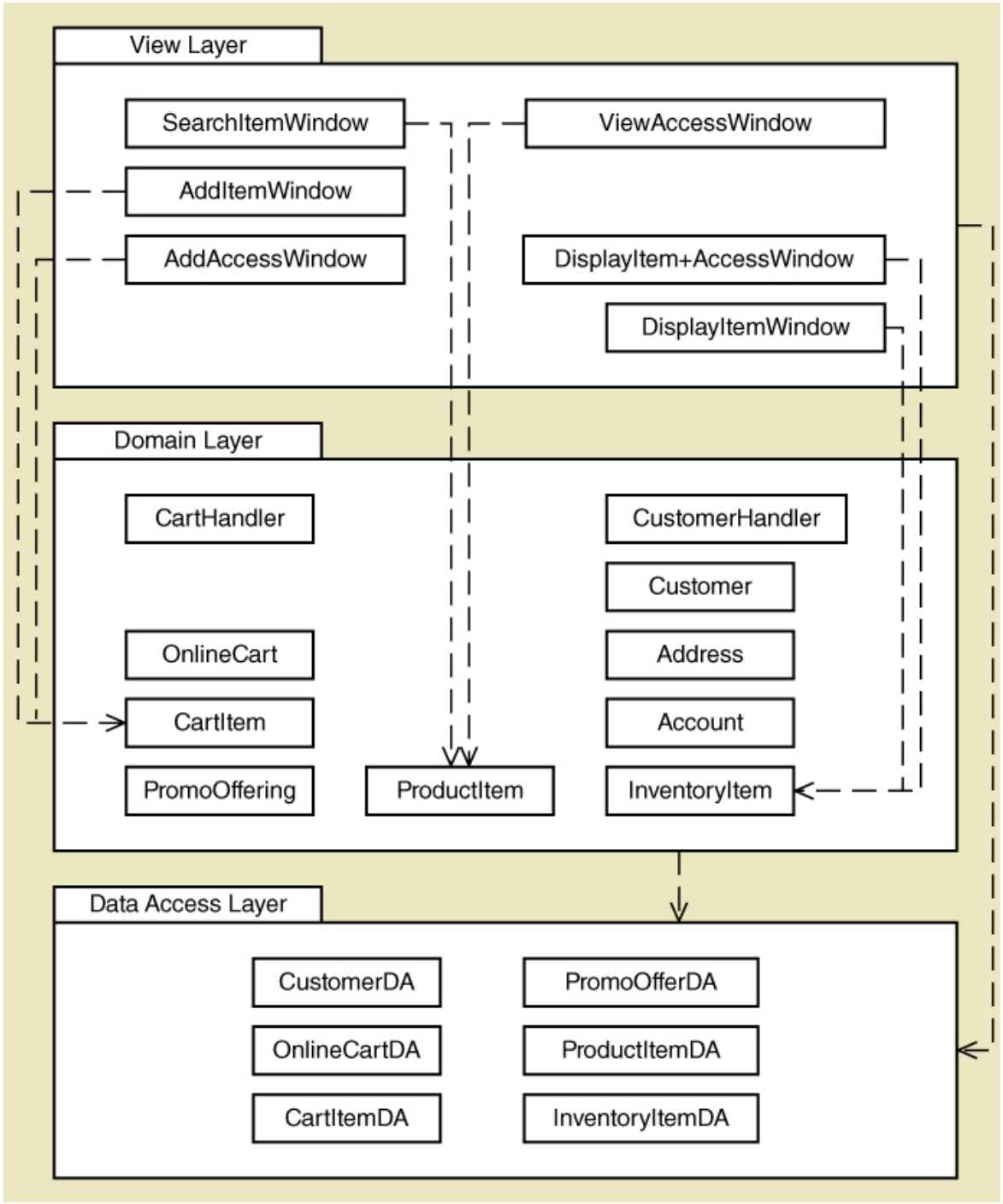
Try it yourself, the answer is in slide 17,

Package Diagrams

- Can be used to define formal packages such as subsystems
- Can be used informally to group classes together for understanding
- Dependency relationship
  - A relationship between packages or classes within a package in which a change of the independent component may require a change in the dependent component. Indicated by dashed line. Read in the direction of the arrow, i.e. A→B is A depends on B.

Package Diagram

- Three-layer package diagram of classes Dependencies
  - View layer depends on Data Access layer
  - SearchItemWindow depends on ProductItem
  - etc.



Implementation Issues

Three Layer Design

- View Layer Class Responsibilities
  - Display electronic forms and reports.
  - Capture such input events as clicks, rollovers, and key entries.
  - Display data fields.
  - Accept input data.
  - Edit and validate input data.
  - Forward input data to the domain layer classes.
  - Start and shut down the system.

Implementation Issues
Three Layer Design

- Domain Layer Class Responsibilities
  - Create problem domain (persistent) classes.
  - Process all business rules with appropriate logic.
  - Prepare persistent classes for storage to the database.

- Data Access Layer Class Responsibilities
  - Establish and maintain connections to the database.
  - Contain all SQL statements.
  - Process result sets (the results of SQL executions) into appropriate domain objects.
  - Disconnect gracefully from the database.

Summary

- This chapter went into more detail about use case realization and three layer design to extend the design techniques.

- Three layer design is an architectural design pattern, part of the movement toward the use of design principles and patterns.

- Use case realization is the design of a use case, done with a design class diagram and interaction diagrams. Using interaction diagrams allows greater depth and precision than using CRC cards.

- Use case realization proceeds use case by use case (use case driven) and then for each use case, it proceeds layer by layer

Summary (continued)

- Starting with the business logic/domain layer, domain classes are selected and an initial design class diagram is drawn.
- The systems sequence diagram (SSD) from analysis is expanded by adding a use case controller and then the domain classes for the use case.
- Messages and returns are added to the interaction diagram as responsibilities are assigned to each class.

The design class diagram is then updated by adding methods to the classes based on messages they receive and by updating navigation visibility.

- Simple use case might be left with two layers if the domain classes are responsible for database access. More complex systems add a data access layer as a third layer to handle database access

Summary (continued)

- The view layer can also be added to the sequence diagram to show how multiple pages or forms interact with the use case controller.

- The UML package diagram is used to structure the classes into packages, usually one package per layer. The package diagram can also be used to package layers into subsystems.