श्रद्धावान् लभते ज्ञानम्

15CSE202 Object Oriented Programming
Lecture 6

OO Design with Sequence Diagram

Nalinadevi Kadiresan

CSE Dept.

Amrita School of Engg.

Object Oriented Development

- Phases
  - **Object-Oriented Analysis: understand the domain**
    - Define an object-based model of it
  - **Object-Oriented Design: Define an implementation**
    - Design the solution
  - **Object-Oriented Programming: Build it**

OO Design

- **design:** specifying the structure of how a software system will be written and function, without actually writing the complete implementation.
- a transition from "what" the system must do, to "how" the system will do it

How do we design classes?

- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- Use UML to represent the classes and objects

The Unified Modeling Language

- UML: pictures of an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- What is legal UML?
  - a descriptive language: rigid formal syntax (like programming)
  - a prescriptive language: shaped by usage and convention
  - it's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

Uses for UML

- as a sketch: to communicate aspects of system
  - forward design: doing UML before coding
  - backward design: doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas
- as a blueprint: a complete design to be implemented
  - sometimes done with CASE (Computer-Aided Software Engineering) tools
- as a programming language: with the right tools, code can be auto-generated and executed from UML
  - only good if this is faster than coding in a "real" language

UML

- In an effort to promote Object Oriented designs, three leading object oriented programming researchers joined ranks to combine their languages:
  - Grady Booch (BOOCH)
  - Jim Rumbaugh (OML: object modeling technique)
  - Ivar Jacobsen (OOSE: object oriented software eng)

and come up with an industry standard [mid 1990's].

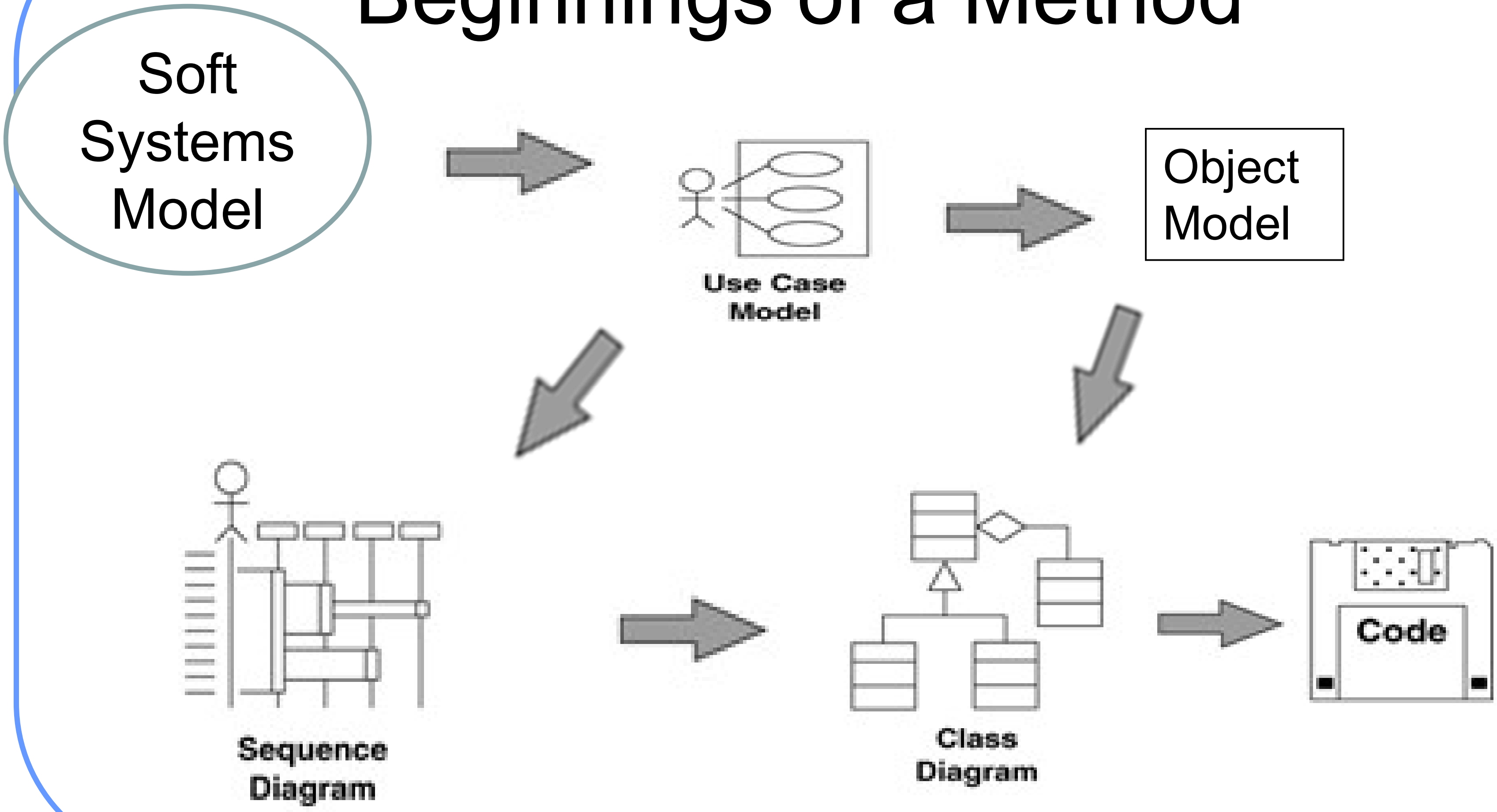
UML – Unified Modelling Language

- Union of all Modelling Languages
  - Use case diagrams
  - Class diagrams
  - Object diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams
- Very big, but a nice standard that has been embraced by the industry.

Types of UML Diagrams

- Structural Diagrams – focus on static aspects of the software system
  - Class, Object, Component, Deployment
- Behavioral Diagrams – focus on dynamic aspects of the software system
  - Use-case, Interaction, State Chart, Activity

Beginnings of a Method



Soft Systems Model

Object Model

UML - Sequence Diagrams

- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Object communication

- Conceptually, objects communicate by message passing.
- Messages
  - The name of the service requested by the calling object.
  - Copies of the information required to execute the service and the name of a holder for the result of the service.
- In practice, messages are often implemented by procedure calls
  - Name = procedure name.
  - Information = parameter list.

Sequence Diagram Key Parts

- **participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

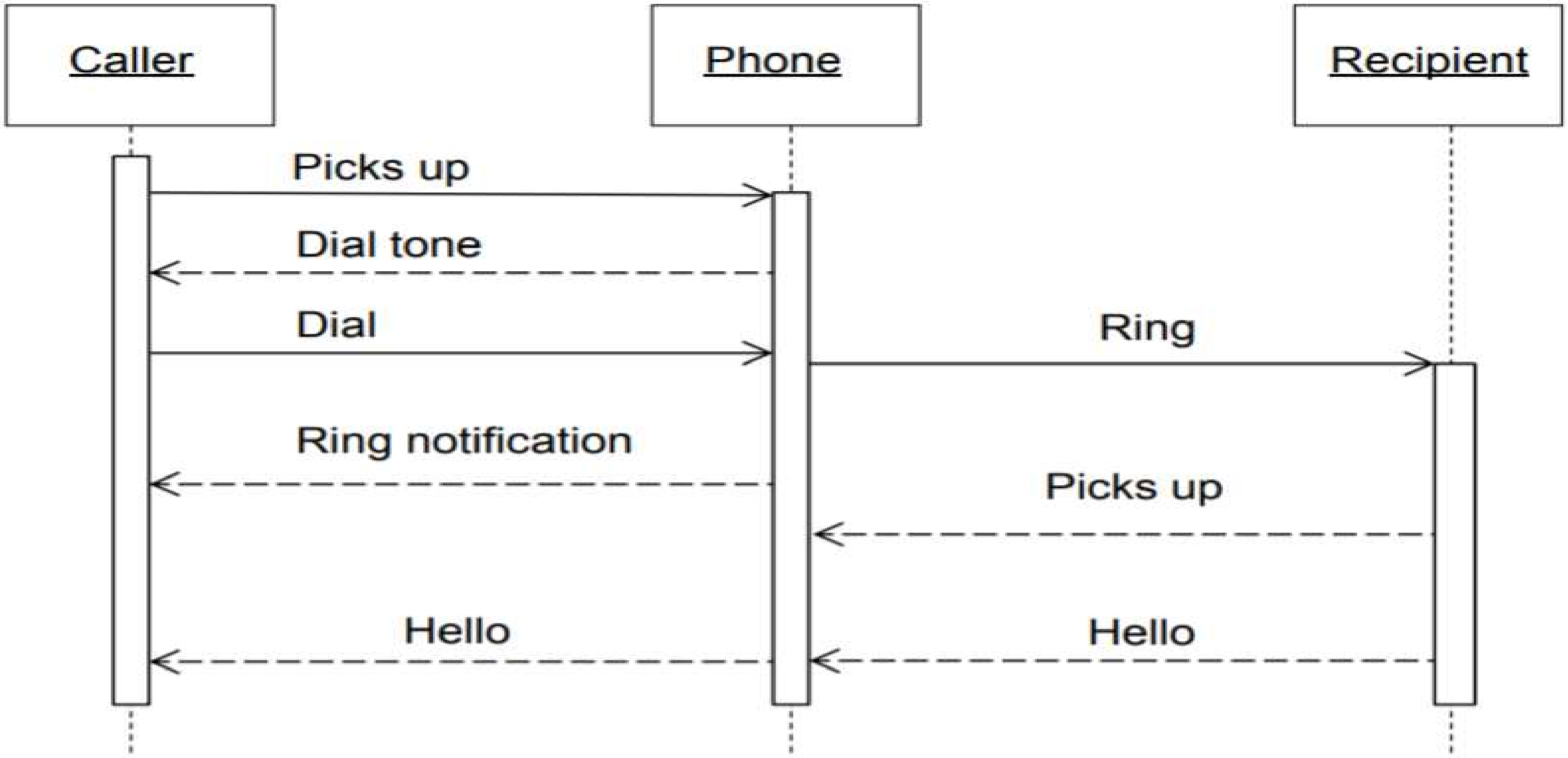
Sequence Diagrams

- X-axis is objects
  - Object that initiates interaction is left most
  - Object to the right are increasingly more subordinate
- Y-axis is time
  - Messages sent and received are ordered by time
- Object life lines represent the existence over a period of time
- Activation (double line) is the execution of the procedure.

Example: Scenario for a Phone Line

- Caller lifts receiver
- dial tone begins
- caller dials digit (5)
- dial tone ends
- caller dials digit (5)
- caller dials digit (5)
- caller dials digit (1)
- caller dials digit (2)
- caller dials digit (3)
- caller dials digit (4)
- called phone begins ringing
- ringing tone appears in calling phone
- called party answers
- called phone stop ringing
- ringing tone disappears in calling phone
- phones are connected
- called party hangs up
- phones are disconnected
- caller hangs up

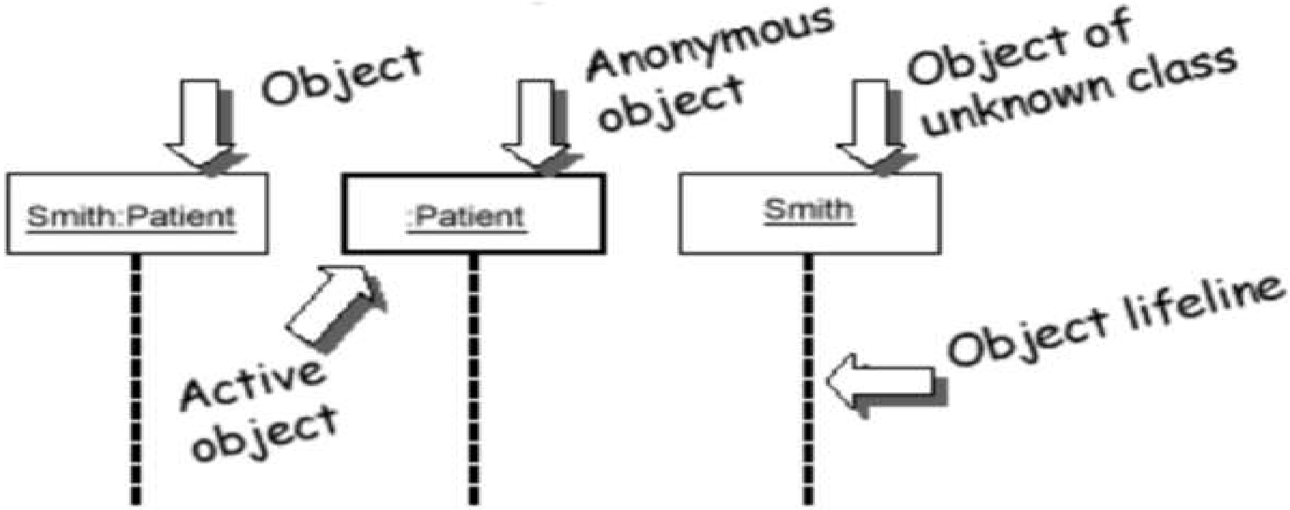
Sequence Diagram (make a phone call)



Representing Objects

Squares with object type, optionally preceded by "*name* :"

- write object's name if it clarifies the diagram
- object's "life line" represented by dashed vert. line

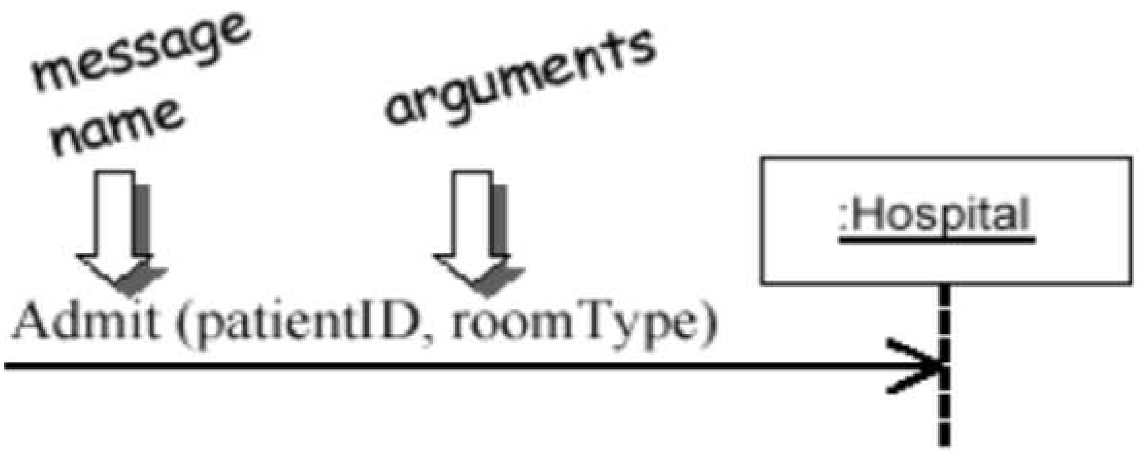


Name syntax: <objectname>:<classname>

Messages Between Objects

messages (method calls) indicated by arrow to other object

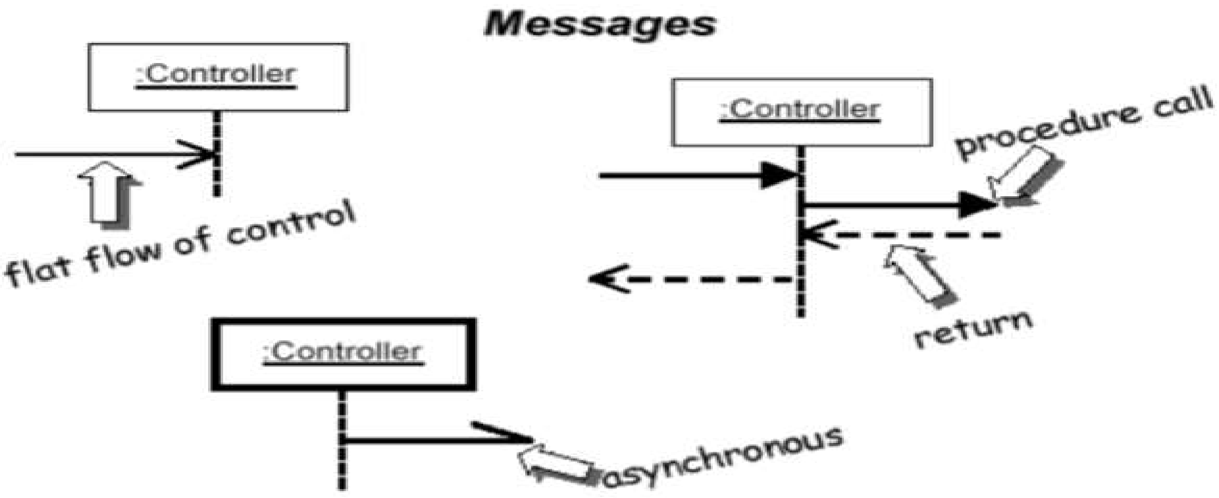
- write message name and arguments above arrow



Messages, continued

messages (method calls) indicated by arrow to other object

- dashed arrow back indicates return
- different arrowheads for normal / concurrent (asynchronous) calls



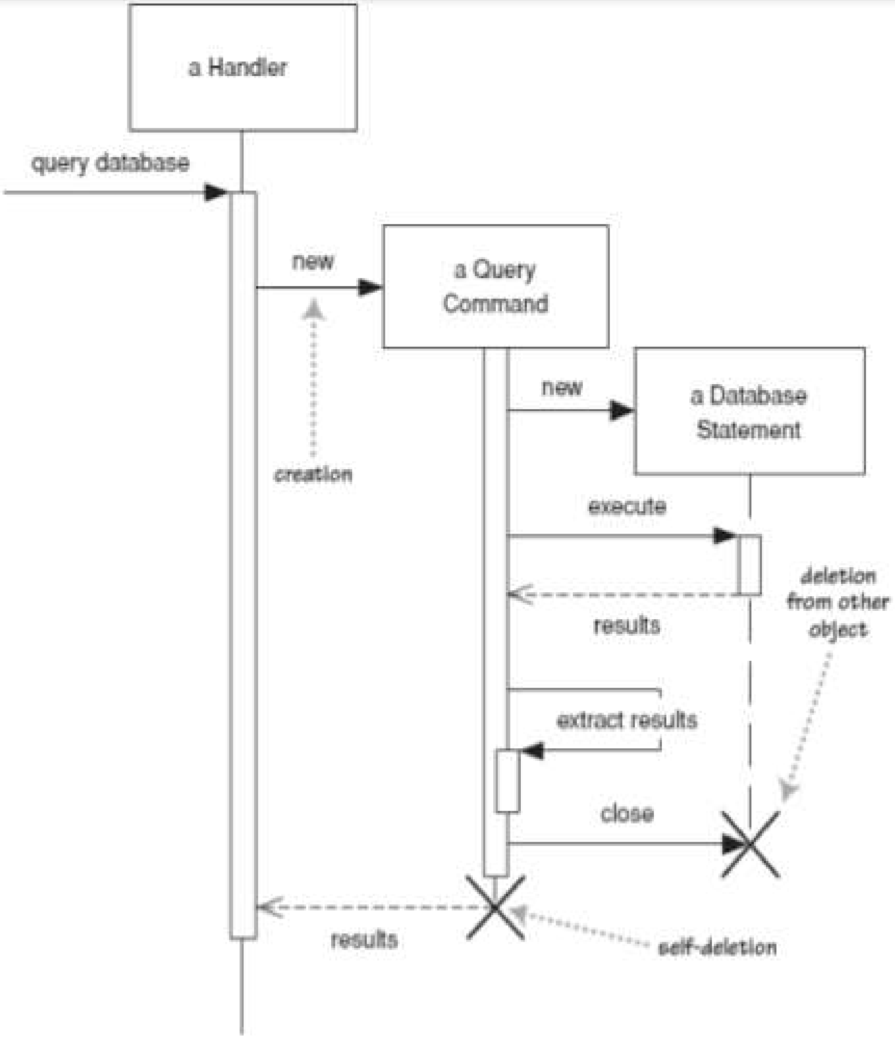
Lifetime of objects

creation: arrow with 'new' written above it

– notice that an object created after the start of the scenario appears lower than the others

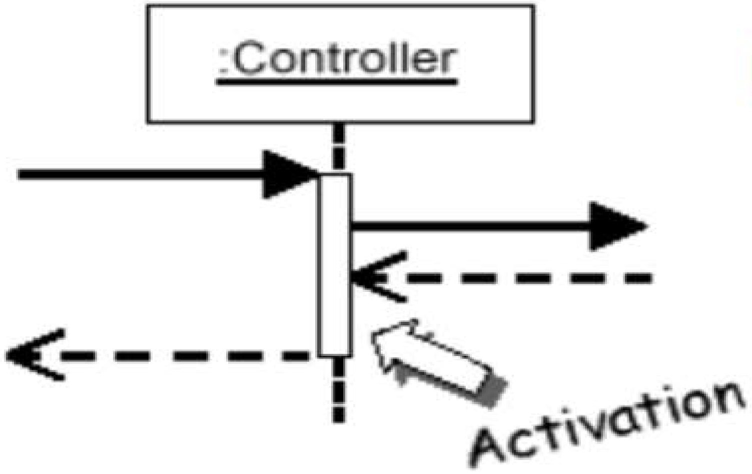
deletion: an X at bottom of object's lifeline

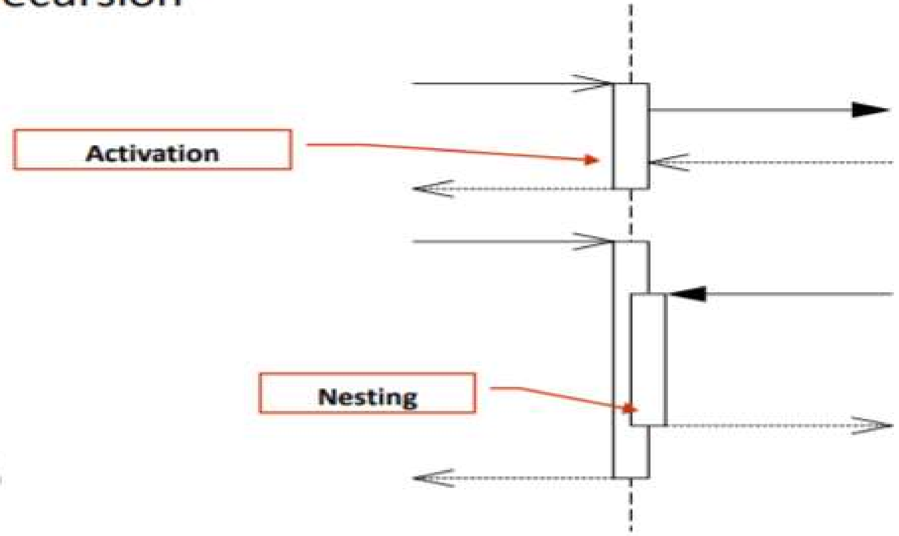
– Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected



Indicating method calls

- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code,
    or it is on the stack waiting for another object's method to finish
  - nest activations to indicate recursion

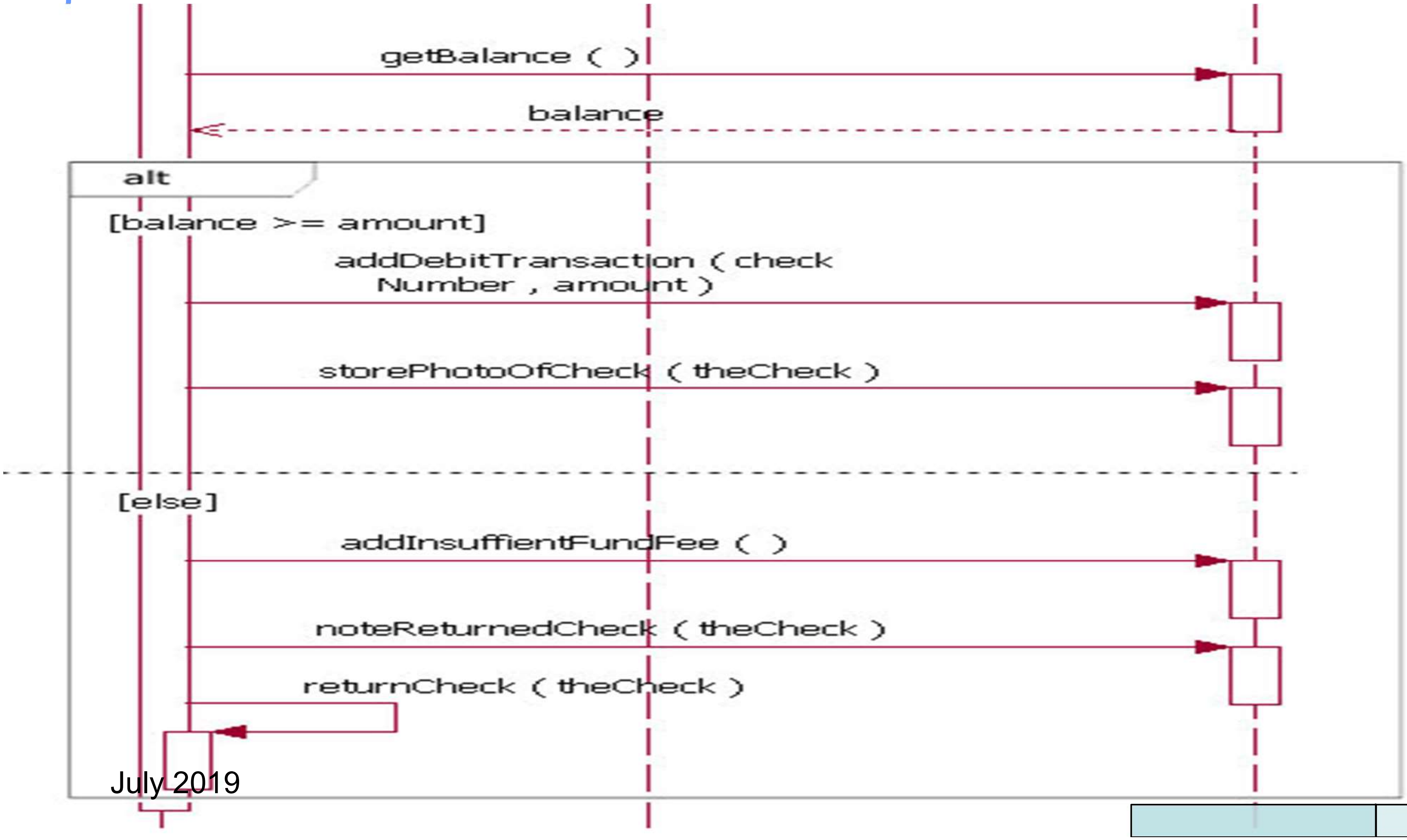




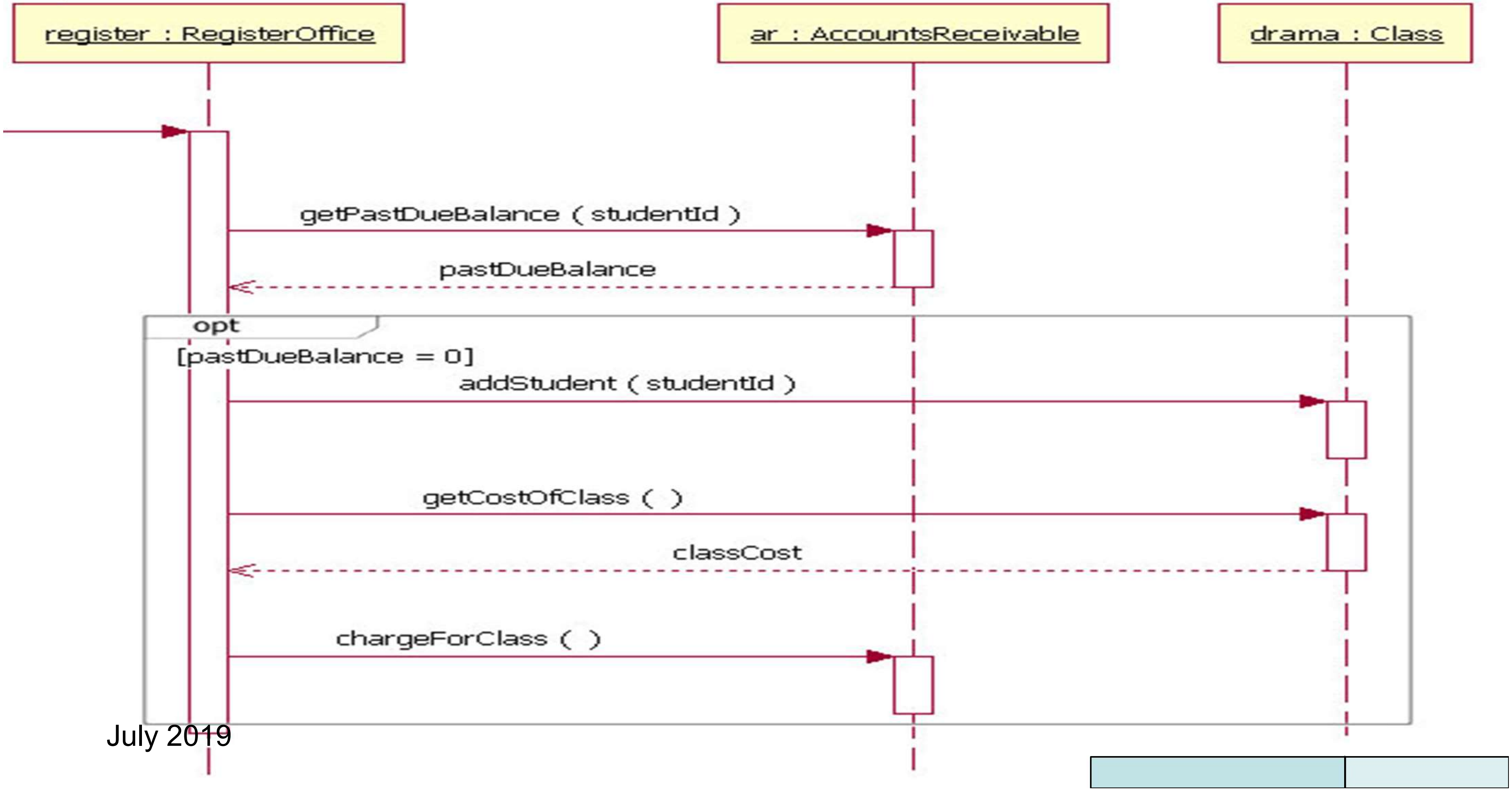
Other Components

- frame: box around part of a sequence diagram to indicate selection or loop
  - `if` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]

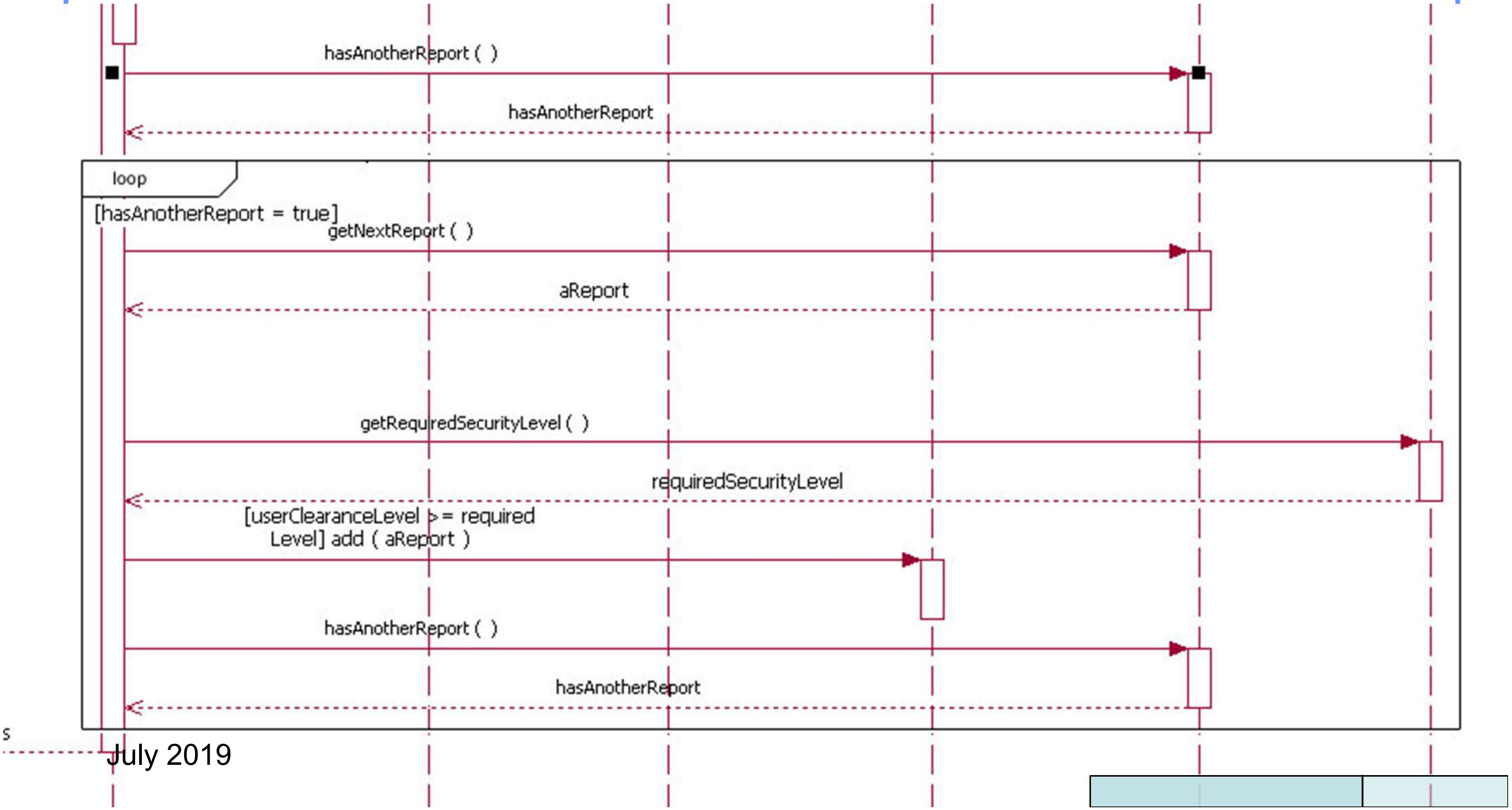
Components: alt/else


getBalance ( )
balance
alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)

Components: option



Components: loop



Rules of thumb

- Rarely use options, loops, alt/else
  - These constructs complicate a diagram and make them hard to read/interpret.
  - Frequently it is better to create multiple simple diagrams
- Create sequence diagrams for use cases when it helps clarify and visualize a complex flow
- Remember: the goal of UML is communication and understanding

Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

Steps for Building a Sequence Diagram

1. Set the context
2. Identify which objects and actors will participate
3. Set the lifeline for each object/actor’
4. Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent
5. Add the focus of control for each object’s or actor’s lifeline
6. Validate the sequence diagram

Steps for Building a Sequence Diagram

1. Set the context.
   a) Select a use case.
      – Typically each use case includes a primary scenario (or main course of events) and zero or more secondary scenarios that are alternative courses of events to the primary scenario.
      – In RUP (Rational Unified Process), user requirements are captured as use cases that are refined into scenarios.
      – Then: A scenario is one path or flow through a use case that describes a sequence of events that occurs during one particular execution of a system.
   b) Decide the initiating actor

Example-1 Car Rental Application Use case

- **Use Case Name**: Release a Vehicle (to a Customer)
- **Actors**:
  1. Front-Desk Clerk
  2. Customer
- **Preconditions**: Vehicle has been assigned to the customer

Basic Flow ("Sunny Day Scenario"):

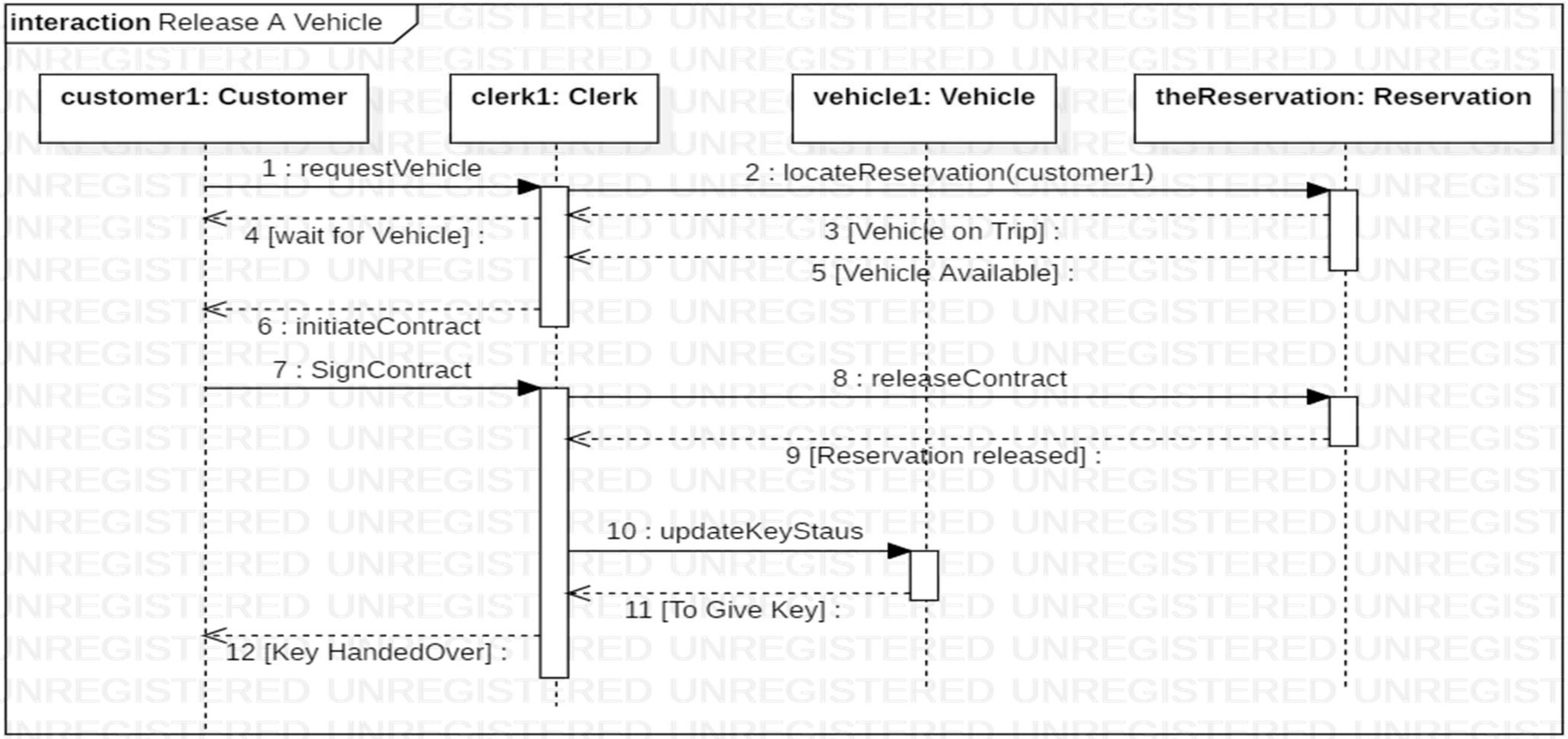
1. A **customer** comes to the **office** to acquire a **vehicle**.
2. The **clerk** locates the **vehicle reservation contract** by means of the **reservation number** and/or **customer name**. [Exception: Required vehicle type is not available due to late arrivals.]
3. The **customer** signs the **contract** and the **clerk** gives the **keys** to the **vehicle**.
4. The **clerk** then marks the **contract** active by entering the **vehicle release date** (today's date) onto **the vehicle reservation contract**.
5. The use case terminates at this point.

Example-1 Car Rental Applicat
Use case

Each color represents a conceptual class

- **Noun-Phrases Identified**: Customer, Customer name, Clerk, Office, Vehicle, keys, Vehicle reservation contract, Reservation number, Vehicle release date
- **Noun as conceptual class:**
  - Customer, Clerk, Vehicle, Reservation

Example-1 Car Rental Application Sequence Diagram


interaction Release A Vehicle

customer1: Customer

clerk1: Clerk

vehicle1: Vehicle

theReservation: Reservation

1 : requestVehicle

2 : locateReservation(customer1)

3 [Vehicle on Trip] :

4 [wait for Vehicle] :

5 [Vehicle Available] :

6 : initiateContract

7 : SignContract

8 : releaseContract

9 [Reservation released] :

10 : updateKeyStaus

11 [To Give Key] :

12 [Key HandedOver] :

Example-2 Coin Flip

Use Case Name: Coin Flip

Actor:

1. A **player** who makes a prediction
2. A **player** who gets the other option
3. **Coin**
4. **Coin game**

Pre-condition:

1. **Two players are available**
2. **A coin is available**

Basic Flow:

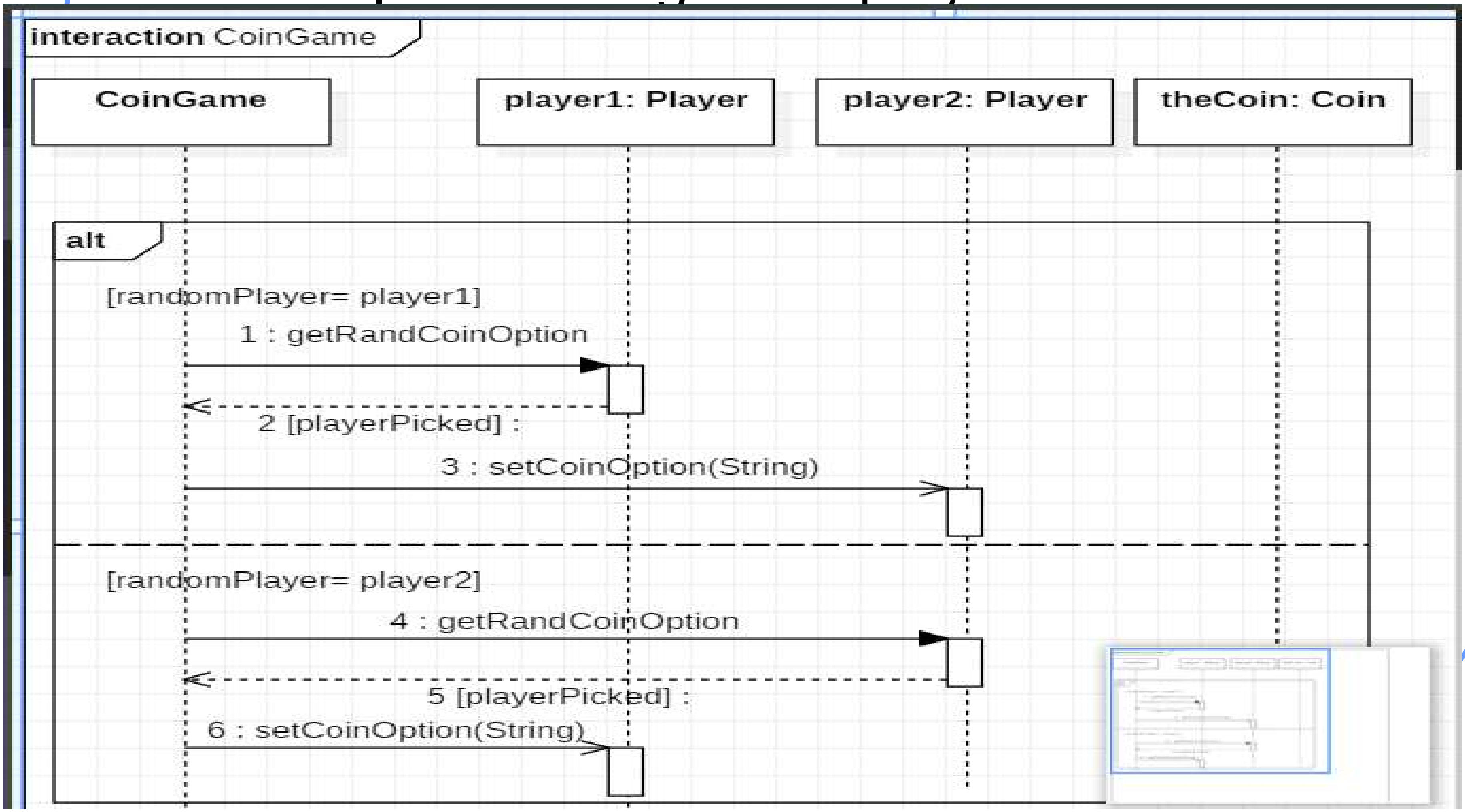
1. A **player** at random is picked to predict the **coin** flip
2. The **player** picked offers a prediction of **coin** flip
3. The other **player** gets the other **coin** flip option
4. The **coin** is flipped and **result** is provided
5. A **winner** and **loser** is picked
6. Offer to try again

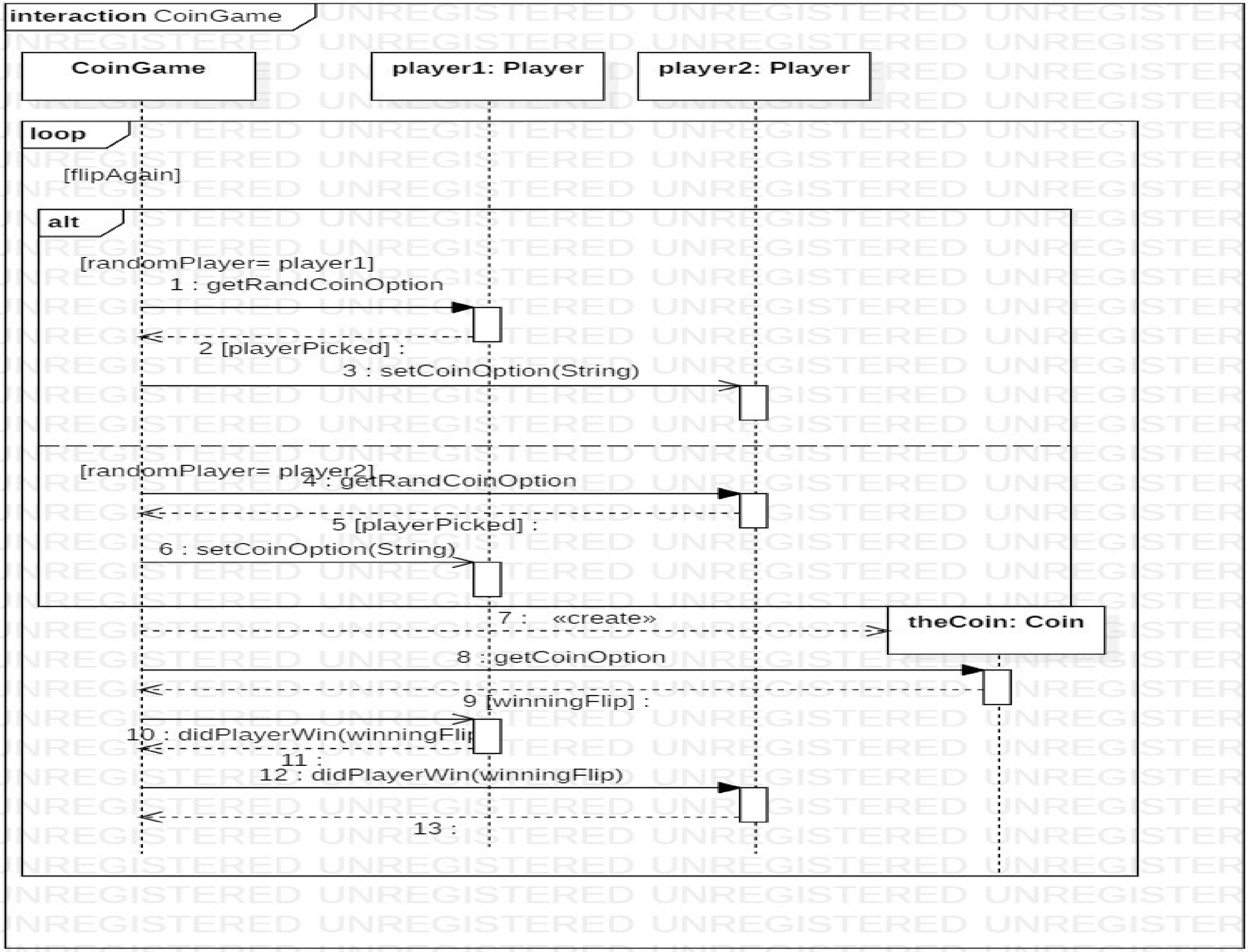
Same as Player

Noun Phrases: Player, Coin, Coin game, winner, loser

Example-2 Coin Flip

Sequence Diagram – player selection




interaction CoinGame
CoinGame
player1: Player
player2: Player
loop
[flipAgain]
alt
[randomPlayer= player1]
1 : getRandCoinOption
2 [playerPicked] :
3 : setCoinOption(String)
[randomPlayer= player2]
4 : getRandCoinOption
5 [playerPicked] :
6 : setCoinOption(String)
7 : «create»
theCoin: Coin
8 : getCoinOption
9 [winningFlip] :
10 : didPlayerWin(winningFlip
11 :
12 : didPlayerWin(winningFlip)
13 :

Example-3 Process Sale

- **Brief:** A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items

Example-3 Process Sale Use Case

- Primary Actor: Cashier
- Main Success Scenario (or Basic Flow):

1. Customer arrives at 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. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

Example-3 Process Sale Use Case

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Example-3 Process Sale
Conceptual Classes

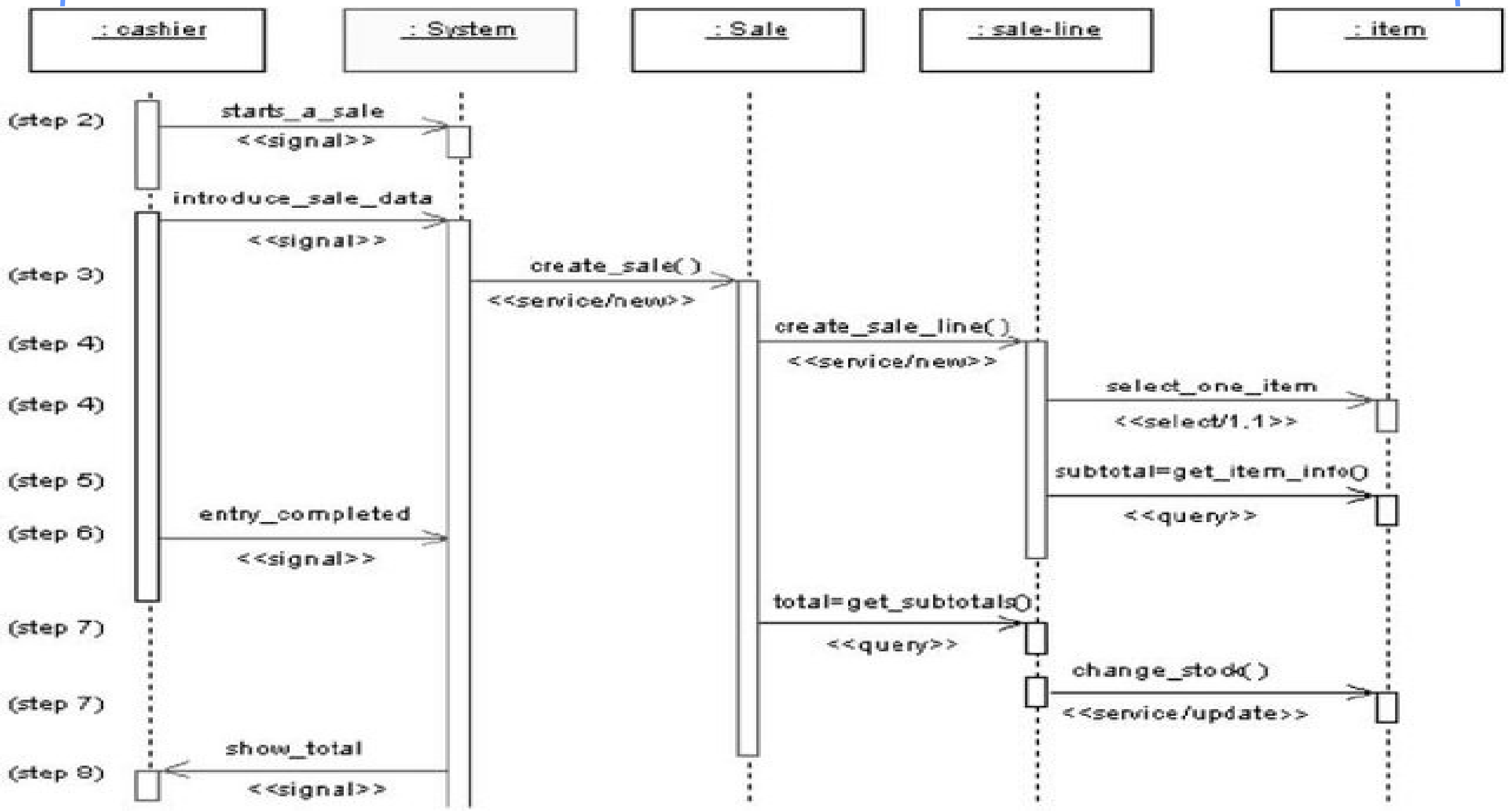
Noun Phrases	**Conceptual Class**
Cashier	**Cashier**
Customer	**Customer**
POS Checkout	**Sale**
Sale, Receipt	
Sale Line item, Goods/Services, taxes, running total	**Sale Line Item, Item**
Item identifier, item description, price	**ProductSpec**
System	**Store**
Payment	**Payment**

15CSE202 Object oriented Programming

Example-3 Process Sale

Sequence Diagram *(sample for a related use case)*

Try it yourself !!!



Sequence diagram exercise 1

- Let's do a sequence diagram for the following casual use case, *Start New Poker Round* :

  The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

  All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies an ante bet of the proper amount. Next each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player, then the second card.

  If the player left of the dealer doesn't have enough money to ante, he/she is removed from the game, and the next player supplies the ante. If that player also cannot afford the ante, this cycle continues until such a player is found or all players are removed.

Sequence diagram exercise 2

- Let's do a sequence diagram for the following casual use case, *Add Calendar Appointment* :

The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

Next Session will be

Object Oriented Design with Class Diagram