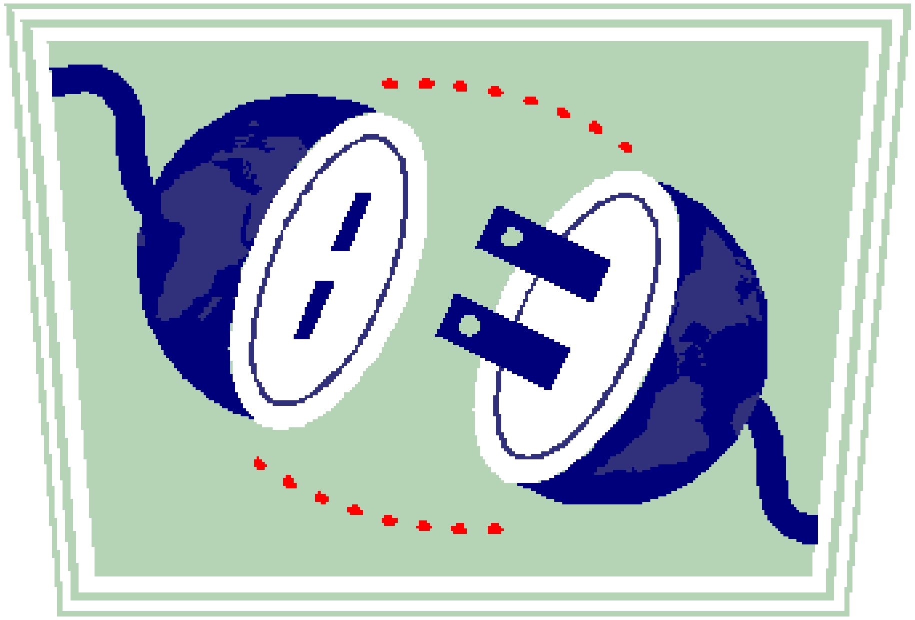
The Component Architecture Method

Exercises



Notes for Exercises

- Many lab exercises have an **Instructor Example** section, and a **Lab** section. The former is for the instructor to work through, in part or in whole, with the students. The latter is for the students to work out in teams. For long exercises with multiple sections, the instructor might work out the instructor example just one section at a time, letting students work on the corresponding section of their student labs.

- The **[estimated times]** are shown for each lab student section, not including the time for the instructor to work out the instructor example section.

- Some labs have a starting template for the solution. Please begin that lab using that template. If solutions to the previous exercises are handed out, please use that solution as the starting point for the subsequent exercise so we are always working on a converging solution.

- Work on the exercises in a **team** of 2-4 people. Work around one end of a table if possible. We recommend that you work on a single solution using just one exercise workbook; you can always make photocopies of your solution for the team at the end of each day.

- The purpose of many of these labs is to learn to use models to produce clear descriptions with reduced ambiguities and errors. In some cases there will be many ways you can define the problem itself, or the solution to the problem. Remember that our main goal is to focus on applying CAM.

Exercise 1.1 Describe Objects [30]

(a) Interacting Objects

You are given the scenario in the table below:

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. Mary arrives at the check in kiosk swipes her credit card. 2. The kiosk requests the check in system to do a check in. 3. The check in system locates Mary's reservation, asks the external credit card service to charge her card, and assigns her room 25. 4. The kiosk emits a key for room 25. 5. Mary calls desk clerk Dave to ask for some exercise equipment to be sent to her room. 6. Dave explains that he is not authorized to do that himself, but says he will take care of it. 7. Dave calls the shift manager, Max, and repeats Mary's request 8. Max sends the equipment to her room.	1. Wilma places an order with sales agent Steve for 2 hens and 3 ducks. 2. Steve enters the new order into WIC's sales system. 3. WIC's sales system internally checks inventory (no need to show inventory as a separate active object) then asks vendor HenHouse for a price for 2 hens. 4. HenHouse looks up its internal price catalog and quotes $6. (You don't need to show the catalog as a separate participating object). 5. The sales system orders the 2 hens. 6. HenHouse delivers the 2 hens to WIC. 7. Sales system checks inventory and asks DuckBroker to price 3 ducks. 8. DuckBroker does a quick scan of historical pricing at its internal reverse auction facility, and quotes $4 for the ducks. (A reverse auction is an auction driven by a buyer, in which sellers bid to sell). 9. The sales system orders the 3 ducks from DuckBroker. 10. DuckBroker starts a new reverse auction, ...

1. Show the interacting objects
   - Draw the individual objects **participating actively** in the interactions, without showing interaction. Only show objects that participate actively in the interactions (e.g. an agent or a system), not objects that are part of the information exchanged between them (e.g. a reservation). Use any icon for each object, writing the object name on or below the object. Include both physical objects and software systems. Don't show "internal" objects at your current level of description.
   - Draw the interactions involved. Use an arrow for any request from object A to object B, or to itself. Number the interactions to show sequencing, using nested numbering to distinguish trigger (3) from responses (3.1, 3.2, …), and label the interaction to name the interaction and to indicate information passed or returned in that interaction.
2. Make an informal list of the kinds of objects and their attributes that are managed by the software systems in your diagram.
   Answer: System name:
   List of objects and attributes it manages.
3. Draw a "Polymorphic" type (one with two different implementations of the same interface). List the operations on that type.

Note that you are describing two distinct kinds of objects: those that interact with others, and those that are manipulated or passed around.

(b) Find Multiple Types

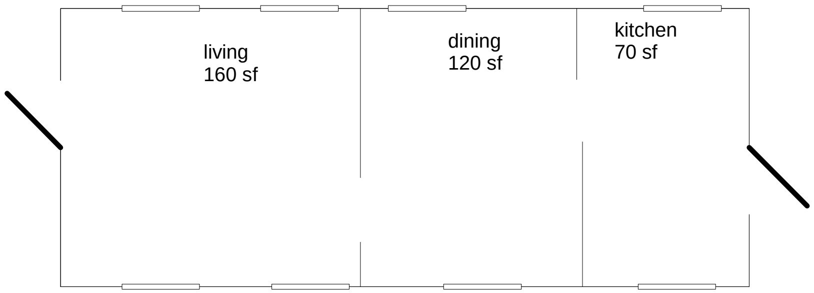
1. In the description below, find an object that can be viewed as two different types by two different applications or components. Identify some other objects that are also members of those types. Name and define those types with some operations and/or attributes.
2. Discuss a problem that might arise if an object with two types is represented in 2 different software systems as 2 disconnected objects.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
o Room 25 appears in the check-in system as an available or unavailable room, with some room rate, when Mary checks in. o Room 25 also appears in HAL's housekeeping system, together with the elevators, the hallways, etc. and the times at which they were last cleaned.	We will skip this student lab. It is included for outside class use. o The pricing system tracks unit prices of items like ducks and hens that are sold. For rental items like tractors and heavy equipment, it tracks the per-day rental price. o The maintenance system schedules maintenance of the facilities, air conditioning, delivery trucks, etc., as well as of items that are rented, rather than sold, to customers.

Exercise 3.1 Objects, Snapshots [15]

You are given the following floor plan of your house that was built in 1950 in Squirrel Hill.

o



1. Draw one object representing your residence that tells how many **exterior** doors and windows it has, when it was built, and what neighborhood it is in. Draw objects representing each room and its approximate size (ignore wall thickness).
2. Make this collection of objects into a snapshot by drawing links between the objects: Draw links between each pair of rooms that shares a wall and between each room and the residence that encloses it. Label each link.

Exercise 3.2 Information Models [15]

1. Use the snapshot from the previous lab and create an information model from it.
2. How would your snapshot change if you were to represent windows and doors as objects? Make this change to your information model (and to your snapshot if you have time).

Exercise 3.3 Snapshot Pairs [15]

Starting with the previous lab, and ignoring doors and windows as objects (simply modeling counts of doors and windows instead):

1. Draw a before and after snapshot pair for what happens when a window is added to the house.
2. Draw a before and after snapshot pair for what happens when a wall is removed to join two rooms of the house.
3. Draw a before and after snapshot pair for what happens when you remodel a bathroom (no changes to windows, doors, or room connections). Do you see any changes in your snapshots? Why or why not? Discuss whether this means your model is "wrong", and what should guide the things that should appear in a model.

Exercise 3.4 Actions [15]

1. Write an informal action specification for the two snapshot pairs from the previous exercise, underlining terms which refer to attributes in your information model. Choose parameters to the actions to correspond to some object types (or attribute types) in your model.
   - adding a window
   - removing a wall

Answer:
action add window ( )

description

pre

post

action remove wall ( )

description

pre

post

Exercise 3.5 Scenarios with text and snapshots [15]

This exercise will need new attributes or object types that are not in your earlier model and snapshots. Add these as needed by your scenario.

1. You are putting a new wing on your mansion, which consists of laying a foundation, adding walls, roofing, adding windows, adding doors, and putting down finished flooring. Windows, doors, and floors you can do yourself; for all the others you sign a contract with a contractor, and he constructs those parts before you do a final walkthrough. Write a textual scenario that describes your specific series of renovations involved in adding a wing. Be sure to follow the scenario format, which includes giving your scenario a title, initial state and final state.
2. Create snapshots corresponding to the before and after states of at least 3 steps in the scenario. What new types of objects or attributes would you need in your information model?

Scenario name:

Initial state:

1.

2.

3.

4.

5.

6.

7.

8.

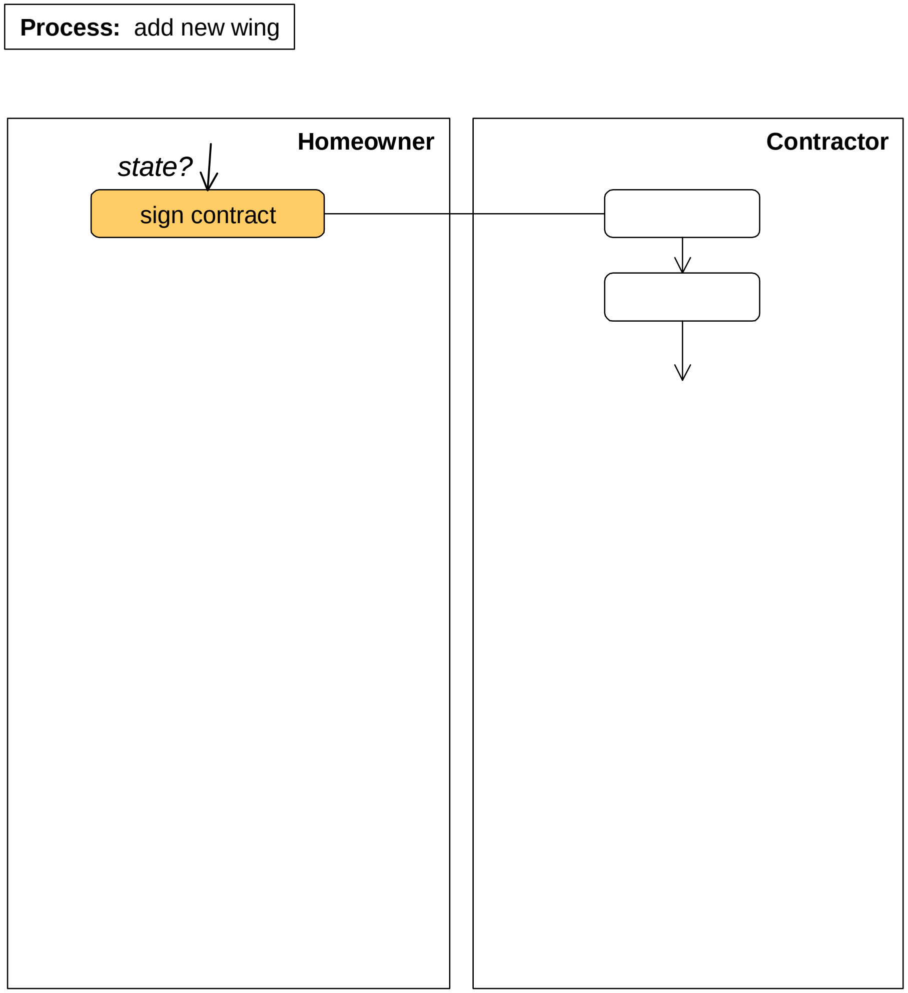
9. …

10. …

Final state:

Exercise 3.6. Activity Diagrams [15]

Using the scenario from the last exercise as a guide, draw an activity diagram that ensures that those same actions are done in an appropriate sequence and collaboration occurs when necessary.



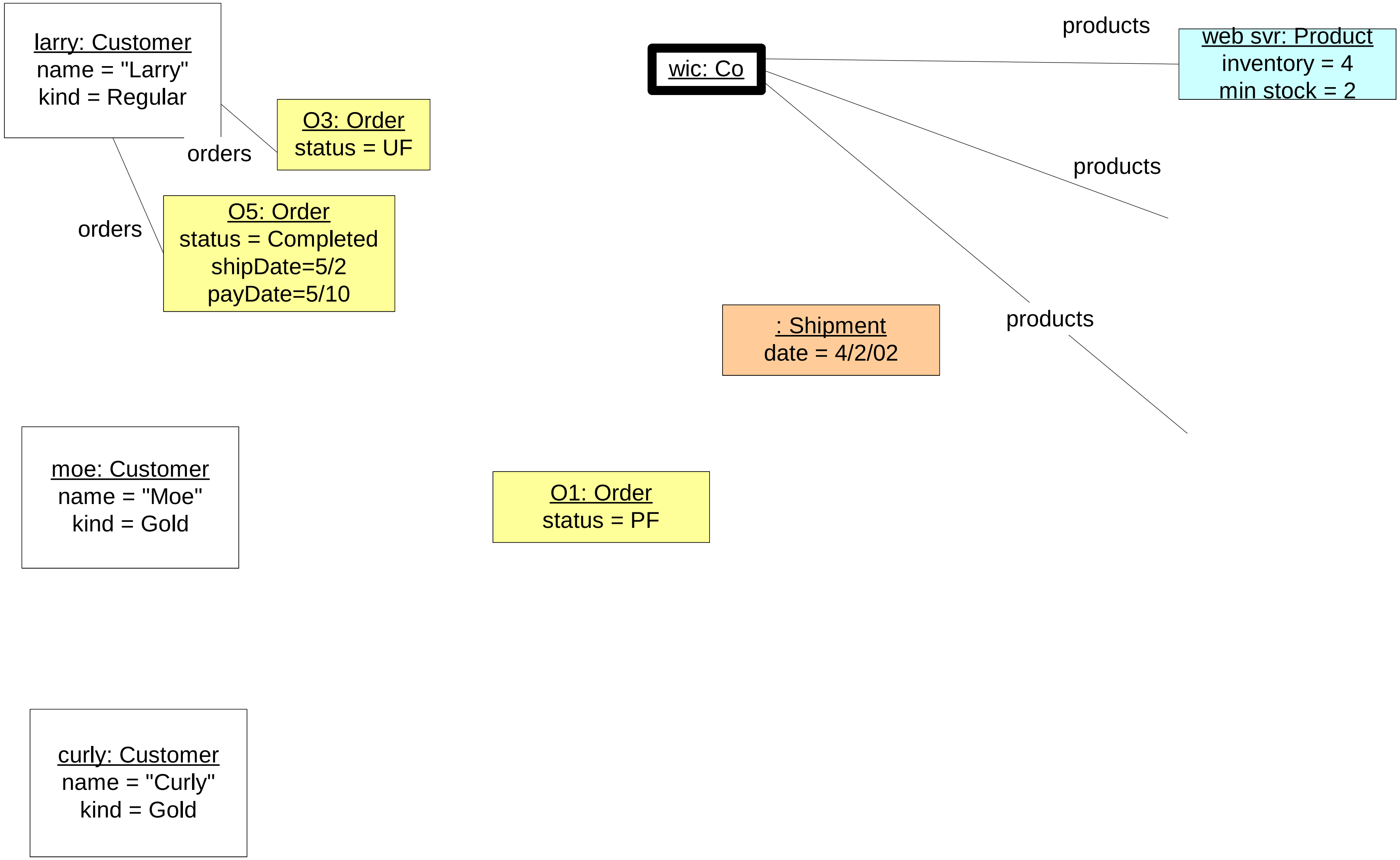
Exercise 4.1 Draw Object Snapshots, Build Information Model from Snapshots. Understand Navigation Expressions. [60 total. Instructor may interleave instructor example with corresponding student lab]

(a) Snapshots & User Interface

Given the situation described below:

1. Draw a snapshot describing the situation below. Your snapshot shows real objects, not just software records. If the links between objects gets cluttered, use attr = value as an alternative.
2. For each **type** of object, write down when a new object of that type comes into existence.
3. [Optional] Sketch either a user-interface, or a domain-specific drawing, for this snapshot.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. HAL is a **hotel chain** with 2 hotels in Chicago (Chic1 and Chic2), and 1 in New York (Man1). HAL has many customers. 2. Chic1 has 2 categories of rooms: deluxe and regular. The room rates at Chic1 are $150 for a deluxe room, and $130 for a regular room. 3. Chic1 has 2 deluxe rooms (rd1 and rd2), 1 regular room (rr1), and the following reservations: • r1 for Vic, confirmed for a deluxe room for 3/1-3/5. • r2 for Wanda, confirmed for a deluxe room, 3/3 - 3/9 • r3 for Wilma, wait-listed for a deluxe room for 3/4-3/7 4. Today is 2/23, and Chic1 has reservations occupied as follows: Stan is currently occupying rd1 from 2/23 to 2/25, Susan is occupying rd2 from 2/21 to 2/26. 5. Currently rd1 is cleaned, rd2 is not cleaned. rr1 is empty and clean.	In this case study, assume that we do care about products (e.g. monkey), but not about individual items of that product (e.g. Thelonious, a 3 year old gibbons monkey). 1. WIC's catalog contains banana, monkey, and web server. Current inventory of these products is 8, 3, and 4 respectively. Minimum stock to be maintained in inventory is 2 of each. 2. WIC's customers are Larry, Moe, and Curly. Moe and Curly are of kind "Gold" customers. All three have a balance of $50. 3. WIC has 5 orders on record: #1 - #5. 4. Orders #1 and #2 are partially fulfilled. Orders #3, #4 are not fulfilled. Order #5 is completed. 5. Order #1 is Moe's. It has 2 line items: for 3 monkeys and 2 bananas. 2 bananas were shipped on 4/2/02. (Only complete line items are shipped). 6. Order # 4 is also for Moe. Order #2 is for Curly; #3 and 5 are Larry's. 7. Larry's order #5 shipped in its entirety on 5/2/02 (i.e. if it had multiple shipments, the last shipment that completed the order was on 5/2/02). It was paid on 5/10/02.



larry: Customer
name = "Larry"
kind = Regular

orders

O3: Order
status = UF

orders

O5: Order
status = Completed
shipDate=5/2
payDate=5/10

wic: Co

products

web svr: Product
inventory = 4
min stock = 2

products

products

: Shipment
date = 4/2/02

moe: Customer
name = "Moe"
kind = Gold

O1: Order
status = PF

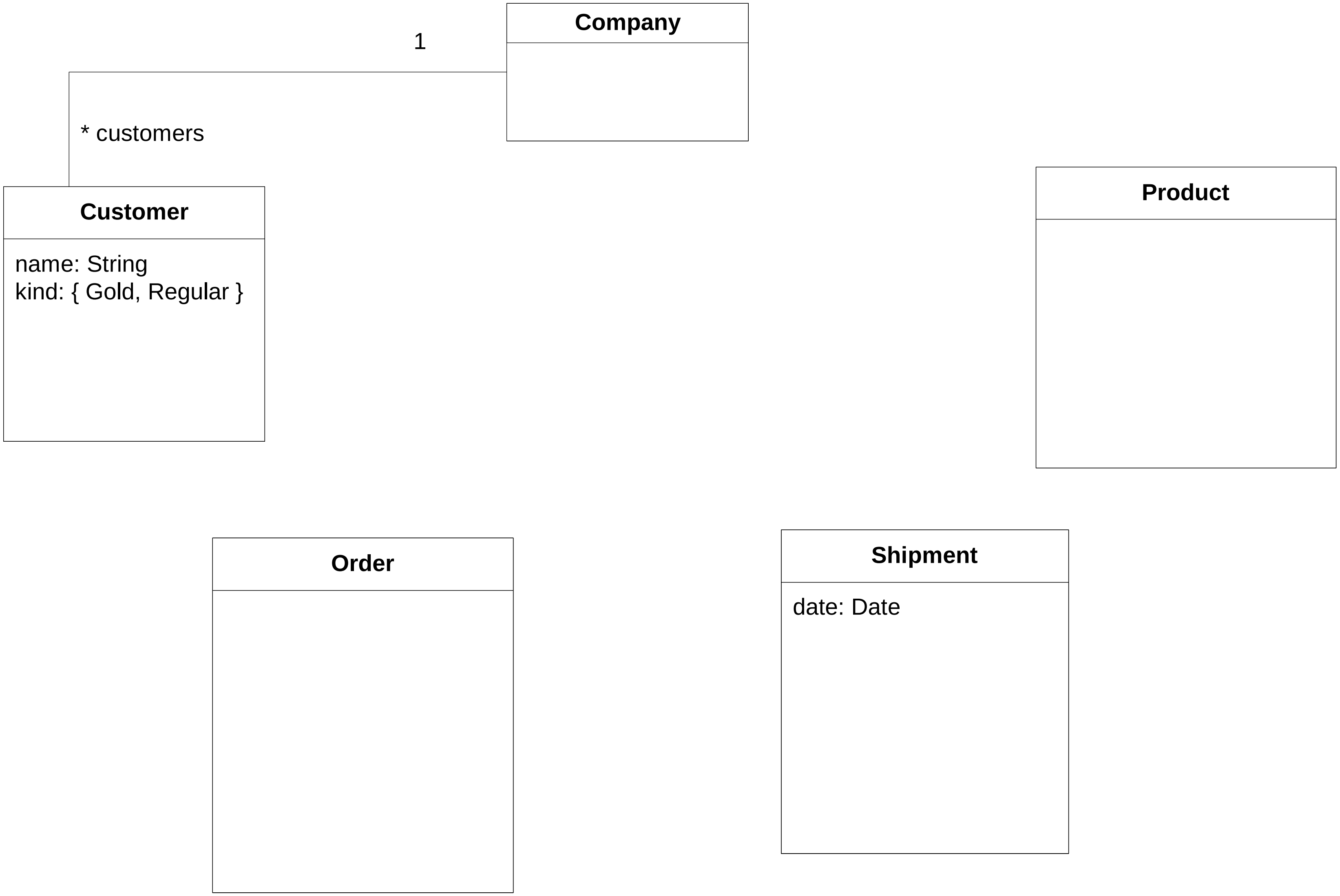
curly: Customer
name = "Curly"
kind = Gold

new Product created when …..
new Customer created when …..
new Order created when …

(b) Build Information Model from Snapshots

Build an information model from the snapshots you have developed. You may need to use your general knowledge about the domain. For each type, identify when new objects of that type get created.

Instructor Example – Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**


Company
1
* customers
Customer
name: String
kind: { Gold, Regular }
Product
Order
Shipment
date: Date

(c) Interpret Navigation Expressions

Identify the **result** sets of objects in your snapshot the following expressions refer to. (Update attributes and association names if needed). For the informal expressions, write out the **formal** version. For the formal expressions, re-state in informal narrative, underlining attribute references.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. All the hotels of HAL **Result:** { ….. } **Formal:**	1. All the customers of WIC **Result:** { } **Formal:**
2. All the hotels of HAL that are in Chicago. **Formal:**	2. All customers named "Larry" **Formal:**
3. The room categories of HAL's hotel named "Chic1". **Formal:**	3. All WIC's orders that are for the customer Larry. **Formal:**
4. Chic1's room categories whose rate is less than $140 **Formal:**	4. All unfulfilled orders for the customer Larry **Formal:**
5. HAL.hotels.roomCategories **Informal:**	5. WIC.customers.orders **à** select (o **|** o.customer.kind = "Gold") **Informal:**
6. HAL.hotels **à** select(name="Chic1"). rooms->select(r | r.unoccupied) **Informal:**	6. WIC.products **à** select (p | p.unitPrice < $25) **Informal:**

(d) Write Navigation Expressions

Write the expressions (both informally and formally) to refer to the following objects in your snapshot, starting from the top-level object. Note that this is a little bit like saying: A is the answer, what is the question? {x, y} means the set of objects x and y.

Instructor Example - Hotel Austerity Limited (HAL)	Lab - Widget International Corp (WIC)
1. { man1 }	1. {moe, curly}: Hint: both have partially fulfilled orders.
2. { r1, r2 }	2. {order#1, order#2}
3. { stan, susan }	

Exercise 4.2 Introduce Convenience Attributes. Write Invariants. Utilize Supertypes. [45]

(a) Introduce Convenience Attributes

Simply introduce a single convenience derived attribute or association into your Information Model for each of the following. Do not describe how that attribute might be derived from others (that will be part (b)). Use a suitable type for the attribute, or multiplicity or {Seq} for the association. Name the attributes or associations as you would want them to appear on a pop-up menu on an object of that type on a user-interface.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. All the cities in which a chain has a hotel. **Chain** **covered cities : Set (City)** **(part b: derivation invariant)…**	1. All the products for which the company has inventory. **Company** ***(name of attribute)***
2. If a particular room category available in a hotel for a given date range.	2. For any product, the current product need (i.e. the quantity of this product that is still unfulfilled across all orders).
3. The ratio of the number of rooms the chain has in a given city within a given price range, relative to the total number of rooms it has in that price range.	3. The percentage of orders that were fulfilled with 2 or less shipments.

(b) Write Invariants

In this exercise, you might find that requirements can be ambiguous. Working out precise definitions fleshes out many underlying questions.

<table>
<tr><th>Instructor Example - Hotel Austerity Limited (HAL)</th><th>Lab - Widget International Corp (WIC)</th></tr>
<tr><td colspan="2">1. Sketch a snapshot to understand the values of your derived attributes on a snapshot, and how you derived those values from others.
2. Write an informal (and, optionally, a formal) invariant that defines the derivation rule for each derived attribute. On the informal version, underline every term that corresponds to an attribute you have introduced in your information model. For complex cases, break them down into multiple simpler attributes that you put together.</td></tr>
<tr><td>Write a separate invariant with the following constraint: a customer cannot be in two different hotels at the same time.</td><td>Write a separate invariant with the following constraint: an order cannot have two different line items for the same product.</td></tr>
</table>

(c) Utilize Supertypes

Add supertypes and invariants to your model to describe the following by considering these questions:

1. What is it that varies? What object type encapsulates the variation?
2. What attributes can be described generically on a supertype across the variations?
3. Where can those generic attributes be used e.g. to define other derived attributes?
4. What subtypes would capture the different variations? What attributes do the subtypes need? How to they derive the supertype attributes?

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. When customers make reservations, they can elect certain preferences e.g. west-facing room, or room with a city view, or room #72. Preferences are not guaranteed. A preference will resolve to some set (possibly empty) of candidate rooms when the customer checks in. The candidate rooms for a reservation is the intersection of the sets of candidate rooms for each of its preferences. Assume that preferences are not simply "strings" to be interpreted by some human. 2. Optional: extend your model to allow combinations of preference, including “and”, “or”, and “not”.	We will skip this student lab. It is included for outside class use. 1. When a customer places an order, she can pay by credit card, cheque, or COD. The order remains unfulfilled until the payment has been approved. A credit card is considered approved once the approval code has been recorded. A cheque is considered approved when a photo ID has been verified. A COD payment is approved if that customer is not on a COD blacklist.

Exercise 4.3 Define Scenarios and Snapshot Pairs. [45]

(a) Scenarios and Snapshot Pairs

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. Name a long-lived activity that customer Wanda participates in with the hotel? When does it start, and when does it end? **Answer:** 2. List the set of finer grained actions that realize the long-lived activity. Introduce and name the long-lived object type if missing. What would be states of an object representing one instance of this type progressing through its lifecycle? **Answer:** 3. Define a scenario which represents a “normal path” through this entire long-lived activity for Wanda and r2. Let each step correspond to one finer-grained action, and pick actions that are conceptually at a consistent level of granularity or abstraction (i.e. check in Vs. arriveAtFrontDesk). Define your scenario so one of the states in the scenario is our earlier snapshot from 4.1(a). Sketch the snapshot after *Wanda checks in on r2.* 4. Sketch some other key snapshots for your scenario. 5. List some exceptions that might arise at steps along your scenario. **Answer:** At step (…) : At step (…) :	1. Name a long-lived activity that customer Moe participates in with WIC? When does it start, and when does it end? **Answer:** 2. List the set of finer grained actions that realize the long-lived activity. Introduce and name the long-lived object type if missing. What would be states of an object representing one instance of this type progressing through its lifecycle? **Answer:** 3. Define a scenario which represents a “normal path” through this entire long-lived activity for Moe and Order#1. Let each step correspond to one finer-grained action, and pick actions that are conceptually at a consistent level of granularity or abstraction. Minimize showing too many activities that are "internal" to WIC. Define your scenario so one of the states in the scenario is our earlier snapshot from 4.1(a). Sketch the snapshot after *WIC does **make shipment** to Moe for Order#1 on 5/1/02.* 4. Sketch some other key snapshots for your scenario. 5. List some exceptions that might arise at steps along your scenario. **Answer:** At step (…) : At step (…) :

(b) Specify an Action

Write an action specification for the two actions as outlined below:

1. Start with an informal description of the post-condition of the action.
2. Identify the parameters of the action, and re-write the post-condition informally, referring to the parameters and navigating from those parameters using attributes from the Information Model. To help identify parameters, look at a before snapshot and decide which object/value (or objects / values) would need to be identified for that snapshot to change. Use the underline convention to show references to attributes and to the parameters. Note that the two actions are not independent: they refer to some shared attributes.
    - Consider making the specification more concise and in terms a client might use to describe it, by introducing convenience attributes and invariants. Move parts of the pre/post into invariants, if they represent constraints on every valid snapshot (not just to the current action).
3. Add the pre-condition if any, for which that post-condition applies.
4. Formalize parts or all of your specification with formal expressions.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
makeReservation **(** **)** **pre:** **post:** checkIn **(** **)** *-- needs a room available and cleaned*	makeShipment (ord: Order): A shipment for a given order is made. The shipment includes as many of the unsent items in that order as possible i.e. if any unsent item has adequate inventory available, it should be shipped. A given line item is never split across shipments. **pre:** **post:** cancelOrder **(** **)** A customer cancels an order, affecting selected items that have not already been shipped. **pre:** **post:**

Exercise 5.1 Build Activity Diagrams from Scenarios. [30]

For the instructor lab, we will now scope our models down to a single hotel (instead of a hotel chain).

Starting with the process and activity descriptions in the table below:

1. Build an activity diagram:

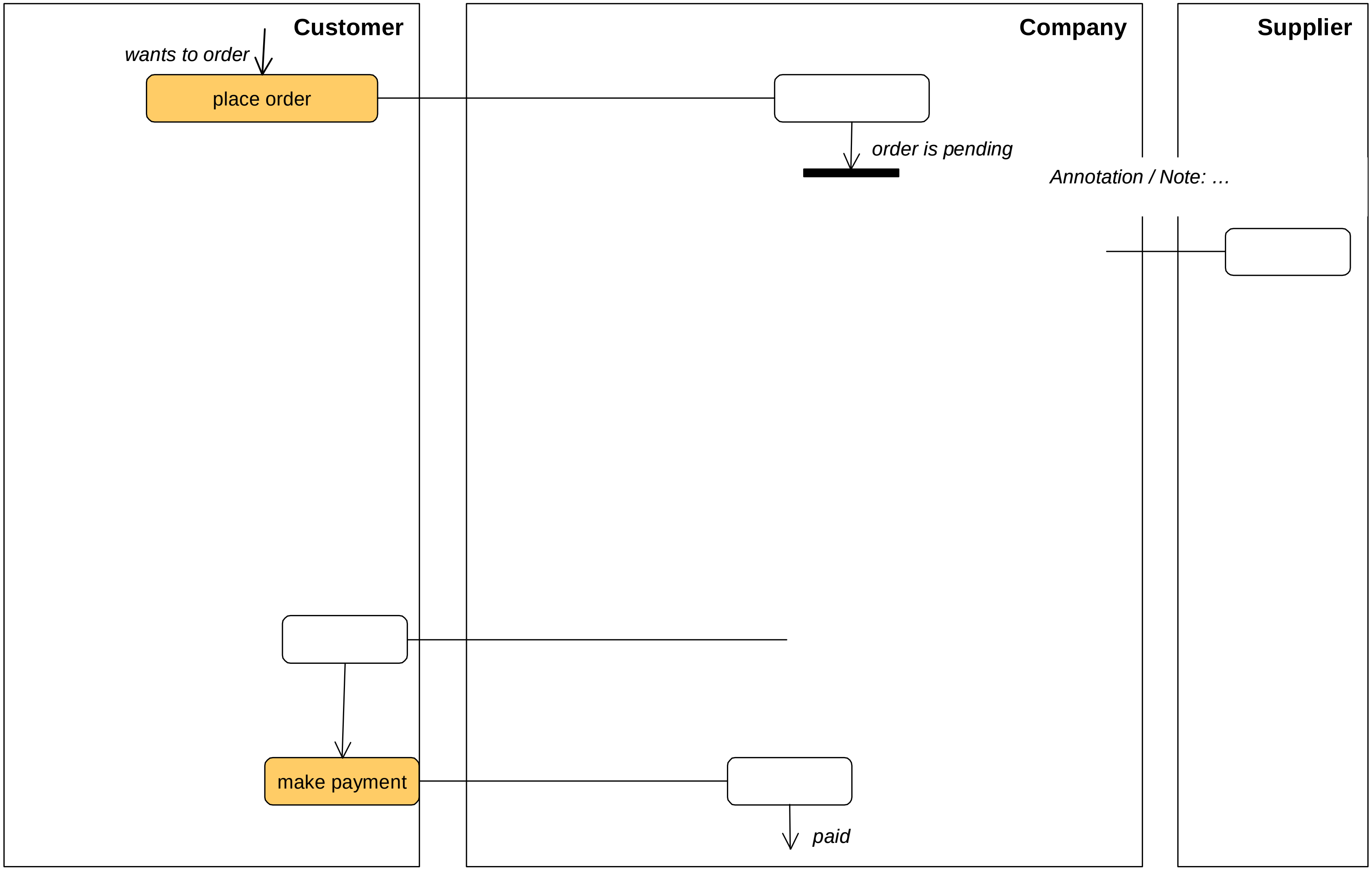
   a. First name the overall process and write a short postcondition for the overall (successful) change of state for the process.
   b. Make a list of finer-grained activities, and write down the main post condition of each. This clarifies the start/end of that activity.
   c. Place the activities on the diagram. Make this a "black-box" view i.e. minimize "internal" structure or sequencing of actions unless they are right before, after, or directly involved with external interactions. Initially do not connect the activities.
   d. Annotate the line ends in and out of the activity with the name of the state before / after the activity.
   e. Connect outgoing lines of one activity to incoming lines of another activity if the state annotations are the same. Revise state names.
   f. Add other flows and connections, including branches, forks, etc. Use annotations if the formal notation gets complex. If you get hung up on how to graphically show iteration, exceptions, etc. simply annotate the diagram with footnotes or comments e.g. *"Activities A, B, C repeated until condition X holds", or "Activities X, Y, Z are interrupted or aborted if event E happens"*
   g. Check that your process works by walking through your earlier scenario (from Exercise 4.3(a)).

2. Write an activity specification for **make a reservation** (Instructor example) or **make a shipment** (Student lab).You can expect your specification to be very similar to the generic action spec you wrote earlier, but with an activity diagram you can also indicate which role provides each input required for the activity, and if any other roles are involved.
3. Update your information model to make sure it supports your specification. Write a short description of the types *Shipment* and *Customer*, indicating whether, in your business model, these represent types in a software system or not.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. The process of a complete hotel visit. When drawing the activity diagram, include the following main actions: • Make a reservation • Change a reservation • Confirm a wait-listed reservation. Assume the existence of an event Room Availability Increased. • Check in (or, fail to show up) • Use hotel service • Change room • Check out	1. The process of a complete purchase. When drawing the activity diagram, include the following main actions: • Place an order; order restock from supplier if necessary. • Make a shipment (including partial shipments), ordering restock from supplier if necessary. Assume the existence of a periodic event from *another* process which signals availability of some products (inventory available) so you do not have to model the checking of inventory. Why would it be a good idea to separate that process? • Send an invoice corresponding to the entire order. • Make a payment for an invoice. • Cancel an order (cancels all unshipped items). • Close an order that has been fully shipped and paid.

Answer:

Process:
Goal:



Customer

Company

Supplier

wants to order

place order

order is pending

Annotation / Note: …

make payment

paid

2.

activity: place order
roles:
inputs: from
precondition:

postcondition:

activity: make a shipment
roles:
inputs: from
precondition:

postcondition:

Exercise 5.2 Build a State Diagram and State Definition Matrix.

1. Based on the activity diagram you developed, what object has an interesting lifecycle through the entire process?
2. Sketch a state model for that type. Transitions should correspond to actions in your activity diagram, and states should include key states on the activity diagram.
3. Construct a matrix of states and actions to ensure that you have covered all state transition cases.
4. Construct a matrix showing each state and how it is derived from the attributes in your information model.

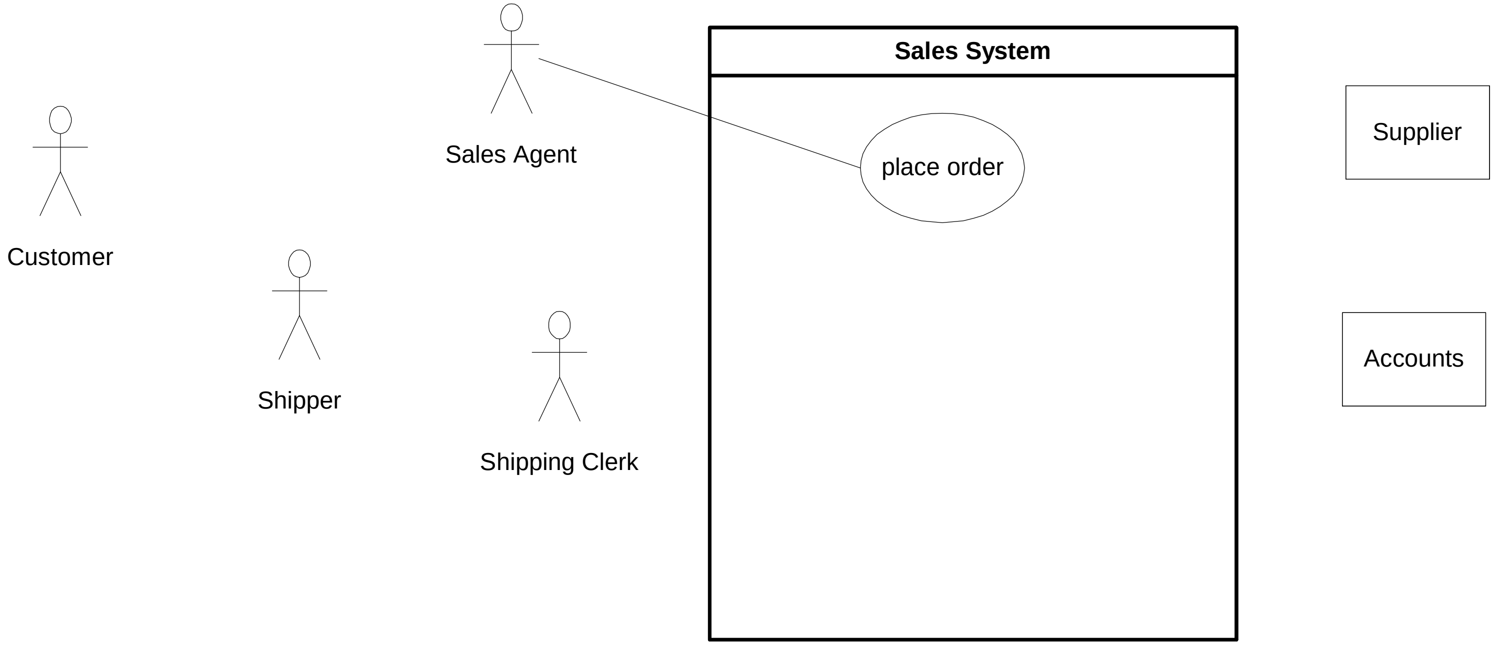
Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**

Exercise 6.1 Use Case Specification

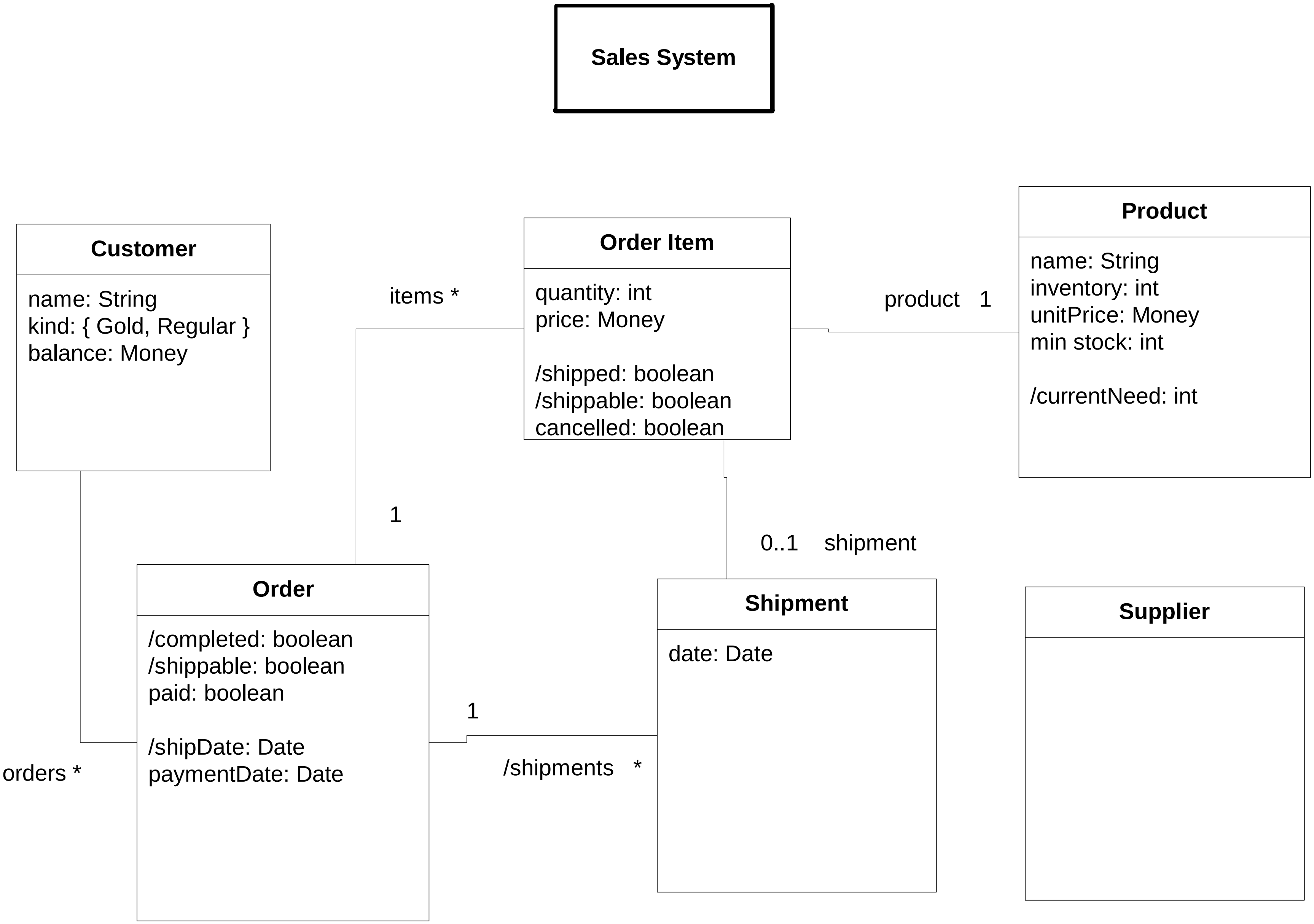
1. Draw a use case diagram. Actors include
    - **HAL**: Credit Card Services, Room Cleaning System
    - **WIC**: Customer, Sales/Shipping Agent, Supplier, Accounts; products need to be re-stocked when orders are placed or shipments are made. Accounts is an external system for invoices and payments.

    Add links from use cases to actors that immediately interact with the system in that use case. Add links (labeled by use case name) to actors that indirectly participate in a use case via other actors, but do not directly interact with the system.
2. Starting with the business information model, draw the black box information model of the "System", showing the system type and its associations to other types. Are there any parts of your business information model that are not part of the black box information model? Any parts added specifically for the black box? Any parts that are assigned to other systems or humans? Do any invariants cross systems? Mark these.
3. Specify the two use cases below. Include the net state change in the "system", as well as all communications with other actors. Reference and underline attributes or inputs or outputs, remembering that you are now talking about the state of **the system**, and inputs/outputs to that system. Your inputs and outputs can be simple values (dates, quantities) or objects (e.g. Order), without details about how those objects are identified. Understand how the use cases interact via these attributes.

1: Answer:



2: Answer (strikethrough things that are not needed in the Sales System model, add new things that are needed)



Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
Use case: **make reservation** Actors: Inputs: ___________ [**from** ____________ **via** _________] Outputs: ___________ [**to** ____________ **via** __________] Trigger: Precondition: Postcondition:	Use case: **place order** Actors: Inputs: ___________ [**from** ____________ **via** _________] Outputs: ___________ [**to** ____________ **via** ___________] Trigger: Precondition: Postcondition:
Use case: **confirm wait-listed reservation** Actors: Inputs: ___________ [**from** ____________ **via** _________] Outputs: ___________ [**to** ____________ **via** _________] Trigger: Precondition: Postcondition:	Use case: **make shipment** (assume this is initiated by the shipping agent, who selects one of the current shippable orders) Actors: Inputs: ___________ [**from** ____________ **via** ___________] Outputs: ___________ [**to** ____________ **via** ____________] Trigger: Precondition: Postcondition:

Use case: **check in**
-- *room must be available and cleaned*
Actors:
Inputs: ___________ [**from ____________ via _________**]

Outputs: ___________ [**to ____________ via _________**]

Trigger:
Precondition:

Postcondition:

Use case: **cancel order**
Actors:
Inputs: ___________ [**from ____________ via _________**]

Outputs: ___________ [**to ____________ via _________**]

Trigger:
Precondition:

Postcondition:

Exercise 6.2 Describe Steps and Alternate Paths.

1. Elaborate your use cases with steps of the main "success" scenario. Underline attributes or *include otherUseCases* in your steps.
2. Check that your use case specification abstracts the detailed step-level information exchanged via the inputs/outputs of the specification.
3. Write at least one successful alternate path description, and one failure alternate path.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
Use case: **make reservation** post: .... **Main (success) steps:** 1. 2. **Alternate paths:** #… #… **use case** <...> (if "included") post: ...	**Use case: place order** Actors: Sales Agent, Customer, (System), Supplier Inputs: Customer, products/quantities from Customer via Sales Agent Outputs: order number to Customer via Sales Agent Trigger: Customer calls to place an order (needs products). Precondition: products in company catalog Postcondition: a new order exists for the customer for quantities of products ordered; its order number is returned; A new customer exists if necessary. Re-stocking has been ordered from supplier if products need >0. **Main (success) steps:** 1. 2. 3. 4. … **Alternate paths:** #… #….

Exercise 6.3 Identify and Specify Use Case Operations

1. Look through the steps in your use case, identify (a) operations that the system provides that are invoked during that use case; (b) **<<out>>** operations that other actors provide that are invoked by the system; (c) raised **<<events>>** or other behaviors that the system needs to exhibit. You will sometimes need to consciously choose the granularity and level of abstraction you want to call "one operation" e.g. placing an entire order Vs. adding one line item at a time.
2. Identify all inputs and outputs for each such operation.
3. Specify at least 2 operations with at least its names, signature of all parameter types, and informal description of its postcondition. Reference and underline attributes in the postcondition. Remember that all the terms referred to in a postcondition should be one of (a) an input parameter provided to that operation; (b) an attribute of that input parameter; (c) an attribute that was part of the system state.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
Use case **make reservation** Use case **check in** **--** *room must be available and cleaned*	Use case: **place order** …. Main (success) steps: 1. customer calls agent to place order 2. agent: include lookup customer 3. agent looks up the product catalog on the system 4. customer informs agent of product and quantity 5. agent creates order in system a. agent enters one line item b. system confirms availability and cost c. agent agrees order item information with customer *Repeat a-c until order items complete* 6. system provides confirmation number, total cost 7. agent confirms with customer Alternate paths: #6: If not enough inventory left for product need, system asks supplier to restock products.

Exercise 6.4 User Interface Specification

Using the steps of your use case and the information model:

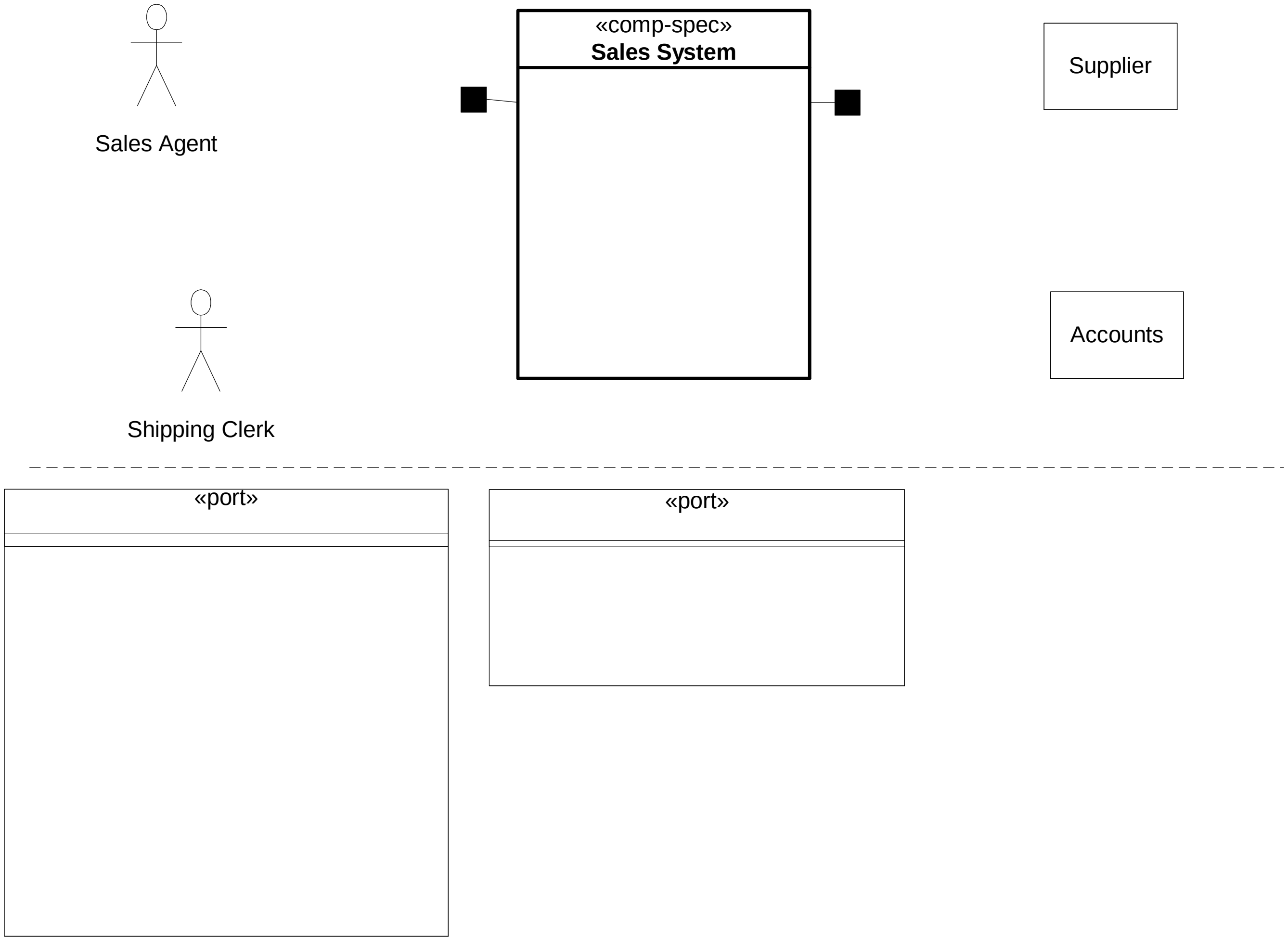
1. Draft a list of named windows, and describe each.
2. Sketch an information model with the main UI state, including the information model of each view and its relation to the core information model
3. Define any application-specific UI events if they correspond to multiple detailed UI alternatives
4. Show state diagram of main UI windows, labeled by all UI events
5. Describe operating procedures for users using terms from the information model and use cases.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
We will skip this lab. It is included for outside class use. use case **make reservation**	We will skip this student lab. It is included for outside class use. use case **make shipment**

Exercise 6.5 Specify Black Box Ports and Assembly

In this exercise we will set up our system description for consistent and recursive treatment between black box and white box views.

1. Draw the ports on your “system”, with the system having a «comp spec» stereotype and icons for each port that provide or require services from other actors. Name each port.
2. Separately show each port as a type, listing the system operations in/out of each port including incoming and <<out>> or <<event>>.
3. Show the connectors from these ports to the actors or external systems. Add a technical annotation for SOAP protocol to the Supplier actor.



Exercise 7.1 Draft Core Type Components and Use-Case Coordinator Components.

(a) Find Core Type Components

Using your black box information (adapted from the business information model) from 6.1(2):

1. Identify the «core» types. These types tend to have a unique identification in the business, 1-to-many associations to other types that can be considered "detailing" types on them, associations to some type that serves primarily to "classify" them, an association to the "top", and do not have mandatory ("1") associations to other types. Defer analysis of any derived associations you might have. Optionally, if you have objects with a complex lifecycle that is quite different from that of the core type, consider partitioning that (with its detailing types) into a separate component i.e. treat it as its own «core» type by bending the above rules a bit.
2. Assign each core type <X>, with its detailing types, to its own component. Typically name that component something like <X>Mgr, <X>Manager, or <X>Mgmt. Show assignment by the UML black-diamond (called "composition" in UML).
3. Check your model and its associations to see if all core and detailing types are unambiguously associated with just one component. You may need to associate a direction with some associations to reduce bi-directional dependencies across components. Think about likely navigation paths across components in making this decision (it will be validated when designing end-to-end collaborations across components).

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**

(b) Add Use-Case Based Components

Using your initial component partition and the use case model:

1. Introduce a component for each large-grained use case on the system, with the operations from the steps of that use case. This component will act as the primary initial "receiver" of the external operation requests (e.g. via a UI) from the steps of that use case, and will also typically act as coordinator across the «core» type components. It does not hold session-specific dialog logic or state; that is considered part of the UI/dialog tiers. Typically name component for use case <U> something like <U> Controller or <U>Coordinator, or a role-name version of the use case.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**

(c) Add UI / Dialog Components

If your system has a user interface, add a UI component (pure presentation) and a corresponding dialog component (remembers the state of the dialog, with preceding selections and the results of preceding requests).

Exercise 7.2 Design Collaborations across Sub-Components

Starting with your candidate set of sub-components, including those based on «core» types and those that support use cases:

1. Draw an instance of each sub-component, and of each external system you interact with. Use a consistent layout for those sub-components that can help you visualize the system as you design the collaborations.
2. For each “architecturally significant” operation (likely to uncover new paths/interaction), draw a collaboration diagram as follows:
   - Select the use-case component that is the "receiver" of that operation.
   - Draw the request and label it: N T: **x := request (a, b)**, where x is the result expected back (if any), and **a,b** are the objects or values passed in with the request. Note that **x, a, b** are variables, not types. Remember that a given sub-component can process a request using (a) any input parameters it received; (b) any information it receives from other components it interacts with; (c) anything it "knows" as part of its own information model.
   - Follow through the rest of the interactions that ensue, using the names **x, a, b**, names of attributes defined in the information model of each sub- component, and introducing new names ("local" variables) as needed. Note that an arrow with **1.1 x := r(a,b)** simply means that at step 1.1, the method r is executed, using parameters **a, b**; it does not mean a hard-coded invocation from one sub-component to another. You can add annotations and/or UML icons as information explanation of iteration, conditions, etc. if you need to.
   - Check that you have accomplished all of the post conditions specified for that use case, and completed all external communications.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
use cases: **make reservation, check in**	use cases: **place order, make shipment**

Exercise 7.3 Define Sub-Component Assembly

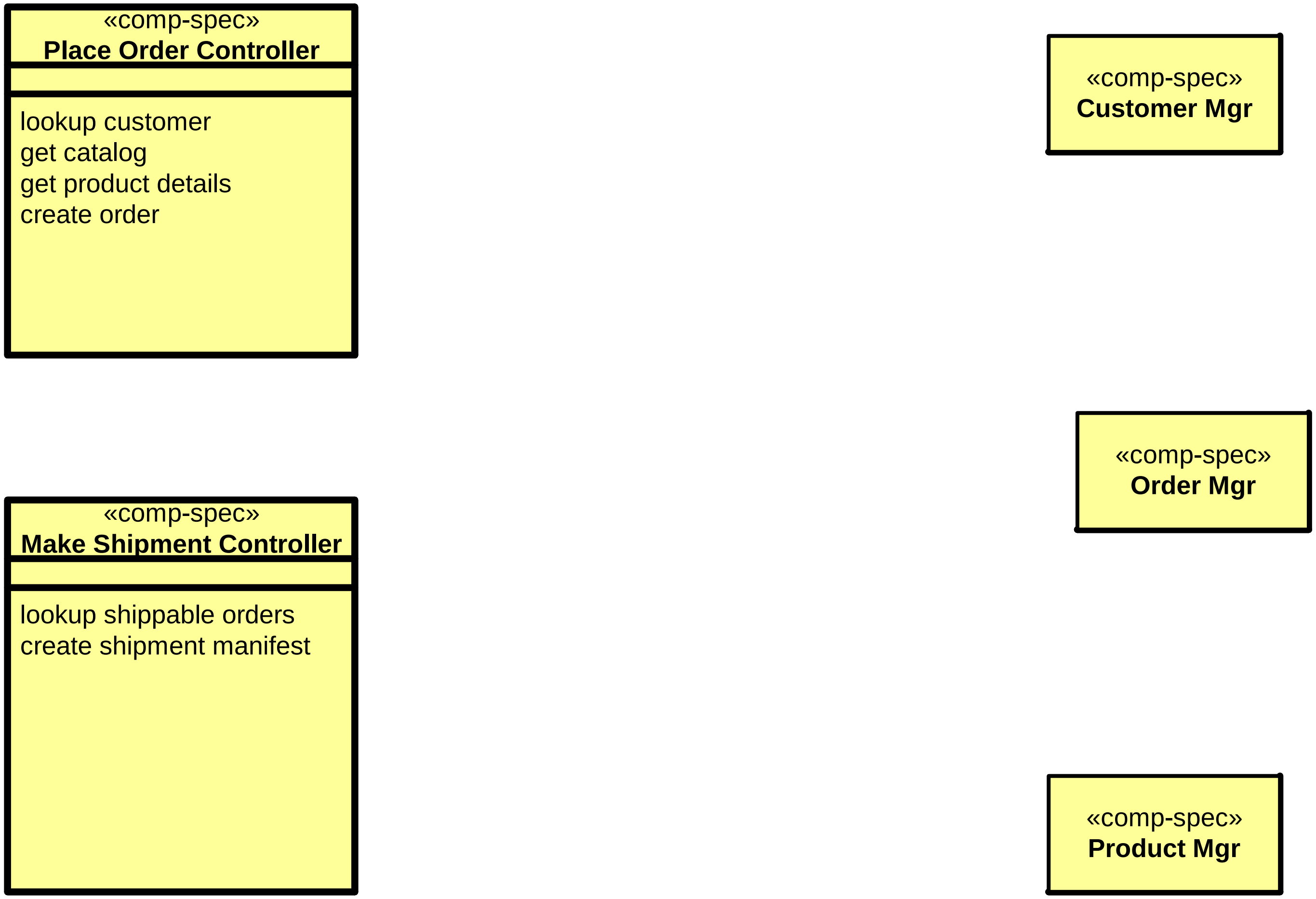
Assume that you have the following two kinds of connectors available to connect your sub-components:

1. A default “call” connector: it requires an outgoing method call from one end to exactly match the invoked method on the other end. It provides synchronous call-and-return behavior.
2. An «event» connector: it takes several pairs of:
    - an event to be raised at one end, with some name and event information type.
    - a method to be invoked at the other end, with some method parameter types.
    - a definition of the mapping from event information to method parameters
    - It provides asynchronous communication of the event, resulting in the invocation of the method.

Build your sub-component assembly as follows:

1. Draw a line connecting sub-components that interact in your collaboration diagrams, and connecting them to external actors.
2. Decide what connector type you will use and annotate the connector with «connector type», unless using “call” connector.
3. Define a «port» at each end of the connectors. Name the port ~A if it is purely a client port to a port A for a set of client-server interactions across a default “call” connector. Name the port something like <X>Events for a port that is nothing more than a source of events.
4. Define at least two ports with some of their operations, <<out>> calls, and <<events>>, including parameters and their types.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**



«comp-spec»
Place Order Controller

lookup customer
get catalog
get product details
create order

«comp-spec»
Make Shipment Controller

lookup shippable orders
create shipment manifest

«comp-spec»
Customer Mgr

«comp-spec»
Order Mgr

«comp-spec»
Product Mgr

Exercise 7.4 Specify Sub-Components

Specify each sub-component as follows:

1. Draw a «comp spec» with associations to a set of «port»s.
2. For a server port **A**, or an event port: list the incoming operations or outgoing events with names and signatures. For a client port, the name **~A** can suffice as a reference to the corresponding server port **A**, without **~A** having to list operations or events.
3. Draw the information model of the «comp spec». Use UML "composition" from the «comp spec» to show the relationship.
4. Add a «data type» class for any input or output parameters referred to in the ports. There will typically be some attributes in the persistent information model and in the data types that "link" them together.
5. Write a short post-condition specification for at least two of the operations on one of the component ports. <u>Underline</u> terms that refer to attributes in the information model.

Exercise 8.1 Extract, Define, and Apply Patterns

1. Identify a named pattern in your information model, as suggested below.
2. Define the "parameters" of that pattern as types and/or their attributes. Simply write the parameters as **<Type, attribute, ...> <Type, attribute, ...>**. Other parts of the pattern can be "generated" from those parameters. If you need to "generate" a name for an element in your model from the pattern write a compound name e.g. <Resource>_Consumer becomes Room_Consumer if <Resource> = Room.
3. Define the "expansion" of that pattern that should result when it is applied to given parameters.
4. Apply the pattern to your model by showing a dependency arrow from your package to the pattern package, annotated with **bind** [pattern parameters] i.e. the bindings of pattern parameters.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
o Reservations have a start and end date. Purchase bids might have a minimum and maximum dollar amount. Extract a pattern for a Range of <X>, including some common queries on Ranges and <X>s. o There is an underlying pattern for allocating rooms to hotel guests, or cars to rental requests. Define a pattern for resource allocation based on things like <Resource>, <Capability>, <Resource Owner>, <Job>, etc. Include the main attributes and associations, and an invariant for "no double booking".	We will skip this student lab. It is included for outside class use. Suppose each product you sold had a corresponding bill-of-materials. When you got an order, or when inventory ran low, you had to order more from suppliers based on the bill of materials. Hence, each product is comprised of parts; and some parts (excepting the "atomic" ones) are comprised of sub-parts.b Add this to your model. Then find, extract, and apply the pattern "Composite" to your model.

Exercise 8.2 Design a Connector Spec + Realization

For the connector described below:

1. Sketch one connector specification pattern. This will include enough information that someone can use it in a white-box assembly and understand its effect, without knowing about the protocol or technology detail that underlies it. Create one example of "applying" that pattern, and work out the "expanded" result of that application.
2. Sketch one possible connector realization for that same connector.
3. On your existing case study model, or inventing a small fragment for the purpose, show how you might apply the connector spec in your white-box architecture, and its realization in the technical architecture. Work out the "expanded" result of applying those patterns.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
A connector that: 1. Spec: Keeps a property at a "sink" end updated with any change in a corresponding property at the "source" end. The properties may be simple (numbers, strings), or complex (structured objects, or entire sets of objects). 2. Realization: Does the above by: • Requiring the "source" end to offer an outgoing event with data about changed values • Requiring the connector itself to offer event subscribers an interface via which different sets of published properties can be referred to by a single subscriber, in order to subscribe to their changes: **subscribe(PropertyName, Subscriber)**, with an unsubscribe() operation. • Requiring every interested "sink" end to subscribe with the connector itself, and to provide an operation to be invoked when a change has taken place.	We will skip this student lab.

Exercise 8.3 Design a Component Realization

Consider a single target technology for (some of) your components e.g. J2EE, .NET, a proprietary component platform, or a legacy environment, perhaps choosing something that is relevant to your project. Discuss any regular mappings you can find between the logical view of components and their interactions, and how those might map to the target technology e.g. names of components, how they are located by others, what the operations correspond to in the technology, how in or out parameters are communicated, etc.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
	We will skip this student lab.

Exercise 8.4 Define a Data Model

Define a pattern for dealing with one (simplified) part of an object-relational mapping. Find one place to apply this pattern to your information model Work out the expanded result of applying that pattern.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
Mapping a subtype to separate tables	We will skip this student lab. Mapping a N-M association to an associative table

Exercise 9.1 Document a Given Deployment Design

Given the deployment decisions below, build a deployment architecture with the diagrams and text descriptions.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. HAL is a **hotel chain** with 2 hotels in Chicago (Chic1 and Chic2), and 1 in New York (Man1). The IT headquarters is in Denver. 2. Each hotel has three machines for reservations and one back office server. Each reservation machine is a Macintosh G4 and the office server is an Athalon Linux server with a RAID array. 3. The IT headquarters has a round-robin load balancing router, five transaction-processing machines, and one DB2 database. The TP machines are rack mounted Athalon Linux servers and the DB2 database runs on a mainframe. 4. Each hotel has a 100mb LAN and a T1 connection to IT headquarters. There is also a backup dial-up connection. 5. The hotel machines run OS X and the Opera web browser for reservations. The back office server runs the client version of a custom J2EE application, and the IT headquarters runs the server version of the same application. 6. What happens when a reservation machine crashes? When the T1 link to the IT headquarters is down? When a transaction-processing machine fails?	Construct a plausible deployment design for WIC. Be sure to cover all of the steps in the deployment design process. 1. Identify scenarios for failure, overload, and security violation cases. Evaluate your deployment with respect to these scenarios. 2. Does your deployment assume anything not specified about the connectors? For example, guaranteed delivery of messages after outage, in-order delivery of messages, timely delivery.

Exercise 10.1 Draw Object Collaboration

1. Design the OO collaboration for one component.
2. Define operations on all your classes to support the collaboration design
3. Define the required class instance variables for all stored data, including references to other objects.

Exercise (UML Appendix 1) Choose Usage of UML Models. [ ]

From each requirement below, identify which UML diagram (class, use case, collaboration/sequence, object, state, package, deployment, patterns) might be used to document it.

Instructor Example - Hotel Austerity Limited (HAL)	**Lab - Widget International Corp (WIC)**
1. As of 6 pm yesterday, HAL had 4 reservations for arrival that day. Res # 1, 2, 3 have arrived and checked into rooms #25, 26, 32. 2. Every room has a room number, and belongs to one room category. The room rate is the same as the rate for the room category. 3. Check out involves the guest, desk clerk, and the system. The result is that the occupancy is ended, and all charges paid. The guest first asks the clerk to check out, providing the room number. The clerk looks up .... 4. A no-show at the hotel. 1. The timer sends an "end-of-day" message to the reservation system. 2. The reservations system sends a "no-shows" message to the billing system, with the list of reservations for that day that were not used up. 3. The billing system sends the credit card service a series of "charge" messages, with the credit card number, the charge amount, and a description of the charge. 5. A reservation goes through the following lifecycle: confirmed, checked-in or no-show, checked- out. A confirmed reservation can become a no-show on its check-in date. 6. Let us structure our models for parallel development. Group A will work out the models for room pricing. Group B will do the in-room services. Group C, the reservations UI 7. The room rate and discount calculator will run on the back office server. It is used by the billing system that runs on the sales office server. Customer access to reservations is from any browser via WIC's web server and firewall. 8. We expect many places in our design to use remote notification of some event. Here is our generic design for remote notification: .... Group A uses it to notify others about room rate changes. Group B uses it to notify others when maintenance takes a room out of service.	No student lab for this exercise.

CAM Lite: Case Study A – Internet Portal

You work at a large internet portal company, Yippee. Yippee has a large number of services offered on the portal but because you are a dot-com, nothing was documented as it was built. The current services offered include:

- **email**: members can send and receive emails: **send** (from, to, XML body), and **receive** (user), based on SMTP and POP3 mail protocol.
- **calendar**: a calendar of events for each member: **add event** (member Id, event info), **update event** (event Id, event info)**, delete** (event Id)
- **maps**: provides maps and driving directions: **get map** (location, scale), **get directions** (from location, to location)
- **membership**: maintains information about all registered members, names, emails
- **contacts**: an address book for each registered member, including member's own physical address(es)

You are asked to create a new service, **Invites**. Invites will use the other services already built rather than recreate their functions.

Invites works like this: A member who is logged on to your portal can go to the Invites section, create a scheduled event, and invite others. Invitations are sent out via email to other guests, who are not required to be members of your portal. The email invitation contains a link to the page for this event on your portal. From there, the guest can say Yes, No, or Maybe to the invitation and leave a note about the event. Before the event date, reminders are sent out to every guest who responded Yes or Maybe. Each invitation can optionally have an address and provide directions specifically for the guest. If the guest is a member of your portal, integrated services should be offered. For example, if the invitation is accepted it appears on the guest's calendar and driving directions are provided from the guest's list of saved locations.

Instructions:

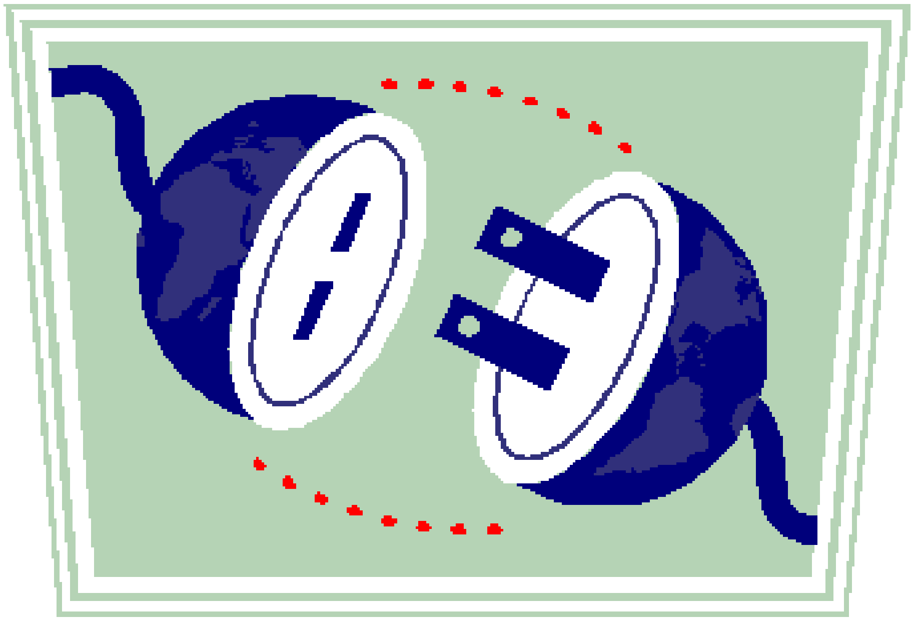
Follow the CAM method and build appropriate models to describe the system. You will need to sketch black box descriptions of the other services that already exist. You will do this in a "list" or "draft" version of CAM, rather than "fully dressed" CAM. Use white-board and flip-charts.

1. Construct the **Business Architecture** for the **Invites service**. In defining this business architecture, it could be helpful to treat the entire portal company Yippee as one single object, and each of the "flesh-and-blood" people as objects. Do steps b-d in a tight iteration.
   a. Write a concrete "story" of an end-to-end scenario of an invitation. Use specific names of participants, and define your initial state.
   b. Determine the actions involved. Cross check that your scenario can be expressed only using these actions.
   c. Sketch out snapshots at key points in the scenario e.g. before or after key steps.
   d. Sketch your information model and sketch the post-condition of each action. Which of your object types are "flesh-and-blood"?
2. Construct the **Black Box Architecture** of the **Invites service**. The context will include users, invitees, and other services (as external actors)
   a. Draw the context showing Invites and each external actor.
   b. List the types and attributes that are managed by each component. If you find that a "link" between a "flesh-and-blood" object needs to be managed by a software component, add an object type that represents what the software component manages separately from the object type which represents the "flesh-and-blood" object.

   c. List the use cases on Invites. Each use case should be of a granularity typically conducted in a single “session”, leading to an end state that meets some goal of the primary actor in that use case. List the trigger, actors, and post condition of each use case, where the post-condition includes the change in state of Invites, and any interactions it has to have the external actors.
   d. Draw the black-box ports, and list the incoming operations,<<out>> operations, and events on each port.
   e. Informally write out the post-condition of the operations Invites requires from the other components, and check that they can be (roughly) specified in terms of the attributes “managed” by that component.
3. [Optional] Add technical concern annotations as overlays to your black box, its ports, and its connectors. Consider the principles, styles, or patterns that underlie your annotations.
   a. communication protocols
   b. data encoding
   c. UI specifics
4. Construct the **White Box Architecture** of the **Invites service**.
   a. Using the core types heuristic and the use case heuristic, determine a set of components to implement the black box. Look also for generic components that might be useful if you introduce new object types.
   b. Design the collaboration between the components and specify each as a black box.
   c. Add technical annotations to your white box
5. Map the white box components into an appropriate web service architecture (for example, J2EE or .NET).
6. Describe the deployment architecture of your system.

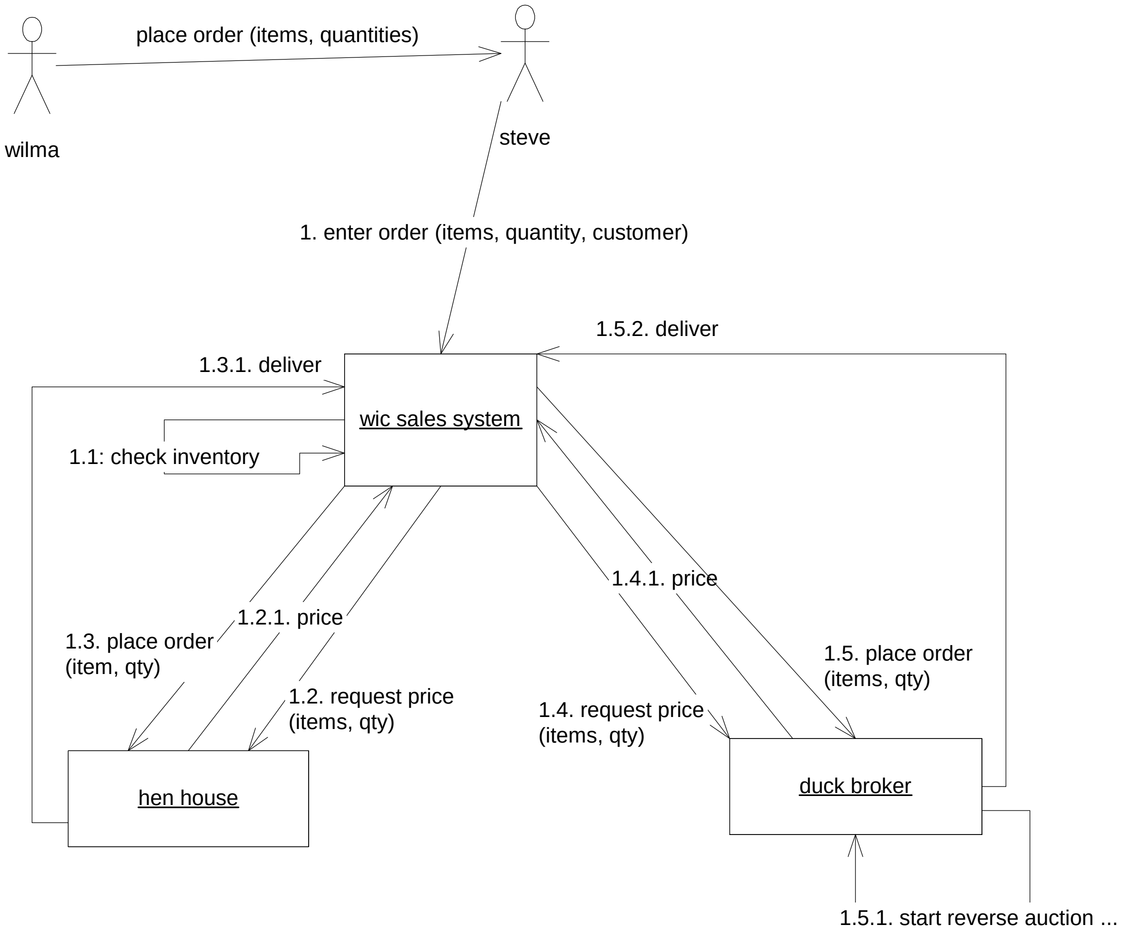
Component Architecture Method

Possible Solutions to Exercises



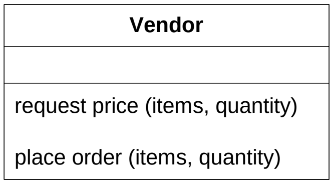
1.1 Describe Objects

a) Interacting Objects

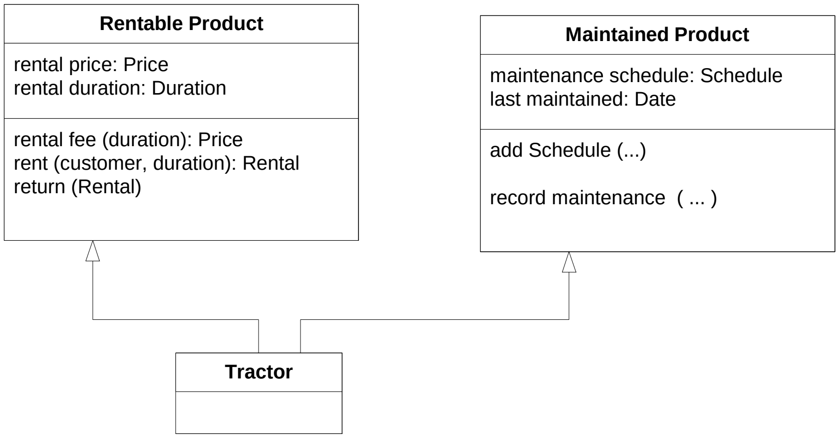

place order (items, quantities)
wilma
steve
1. enter order (items, quantity, customer)
1.5.2. deliver
1.3.1. deliver
wic sales system
1.1: check inventory
1.4.1. price
1.2.1. price
1.3. place order (item, qty)
1.5. place order (items, qty)
1.2. request price (items, qty)
1.4. request price (items, qty)
hen house
duck broker
1.5.1. start reverse auction ...

2. Kinds of objects and attributes managed:
Sales System: customers, orders, order items (with quantity), product catalog, vendor requests (with bids, deliveries)
Duck Broker: reverse auction site, historical pricing per product
Hen House: catalog of products (with price of each).

3. Polymorphic Type
- implemented differently by hen house and duck broker

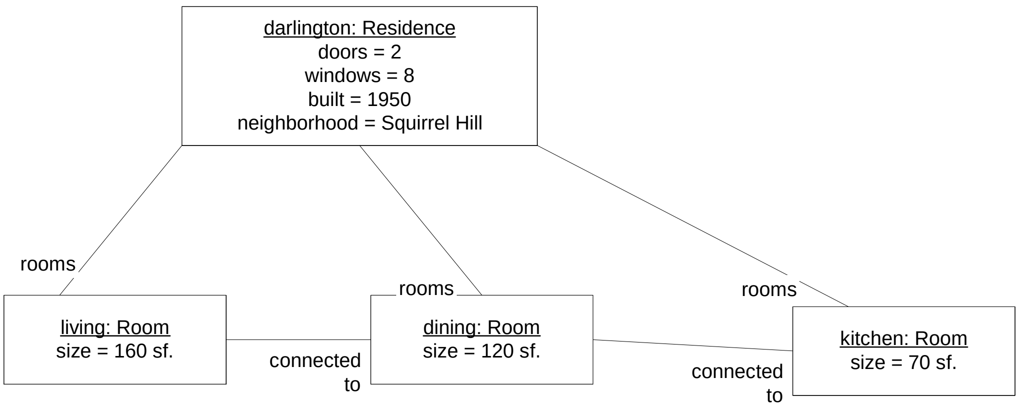

Vendor
request price (items, quantity)
place order (items, quantity)

b) Find Multiple Types

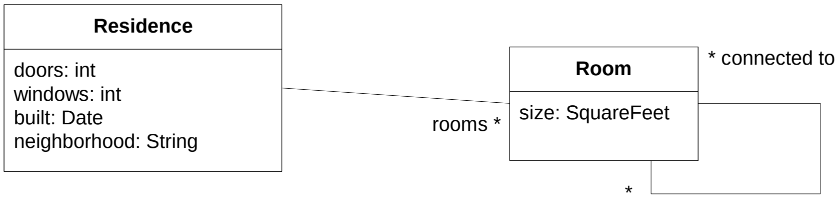

Rentable Product
rental price: Price
rental duration: Duration
rental fee (duration): Price
rent (customer, duration): Rental
return (Rental)
Maintained Product
maintenance schedule: Schedule
last maintained: Date
add Schedule (...)
record maintenance (...)
Tractor

If an object with 2 types is represented in 2 different software systems as 2 disconnected objects, keeping them in synch needs to be addressed/dealt with.

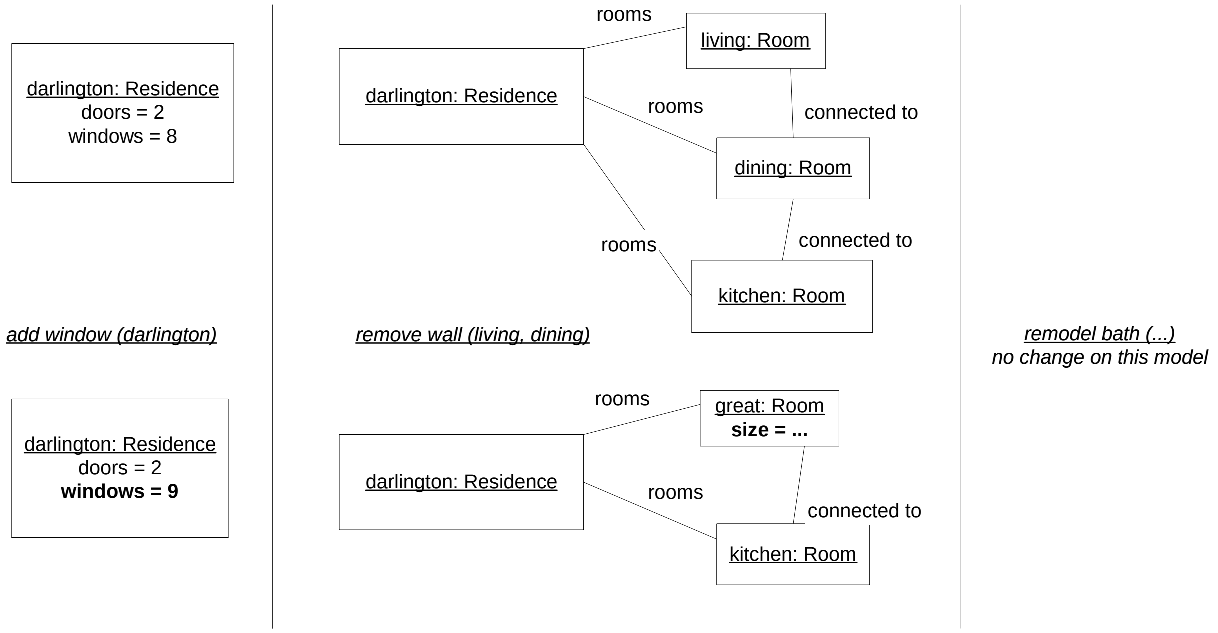
3.1 Objects, Snapshots



3.2 Information Model



3.3 Snapshot Pairs



3.4 Actions

action add Window (r : Residence)
description: the house is remodeled to add a new window to a residence
precondition: none
postcondition: the number of windows in the residence r is one greater than before

action remove Wall (r1: Room, r2: Room)
description: the house is remodeled to join two contiguous rooms into one
precondition: the two rooms are connected to each other
postcondition: the rooms r1, r2 are gone. A new room, r3, is created and added to house rooms.
r3 is connected to all the rooms that r1 and r2 were connected to.
the size of r3 is the sum of the sizes of r1 and r2.

3.5 Scenarios with text and snapshots

Scenario name: new wing on mansion

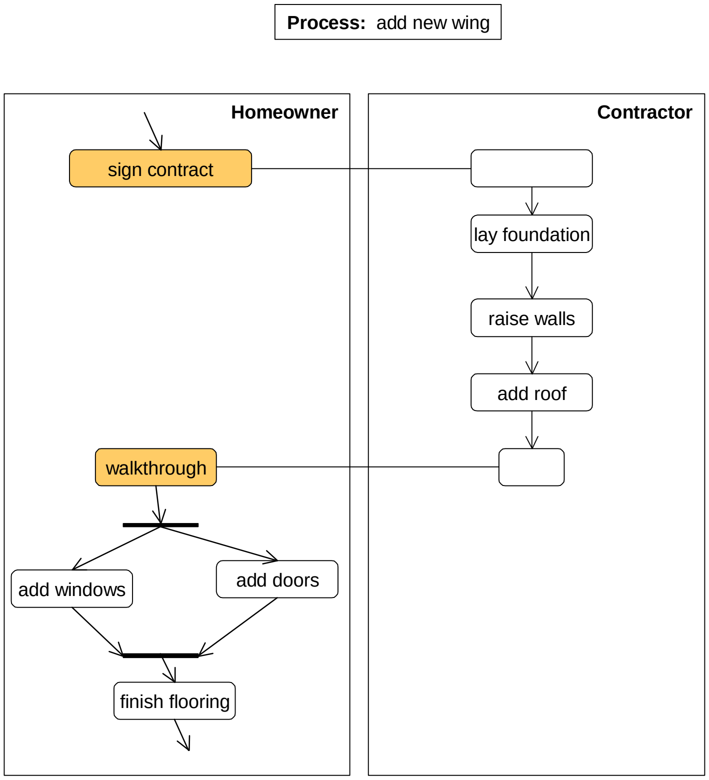
Initial state: Titus is a contractor, Jake the homeowner

1. titus and jake sign the contract
2. titus lays the foundation
3. titus raises the walls
4. titus adds the roof
5. titus and jake do the walkthrough
6. jake adds the windows
7. jake adds the doors
8. jake adds the finished flooring

Final state: the new wing is complete

Snapshots: this would probably require new types of objects for Wing, Foundation, Roof, Wall.

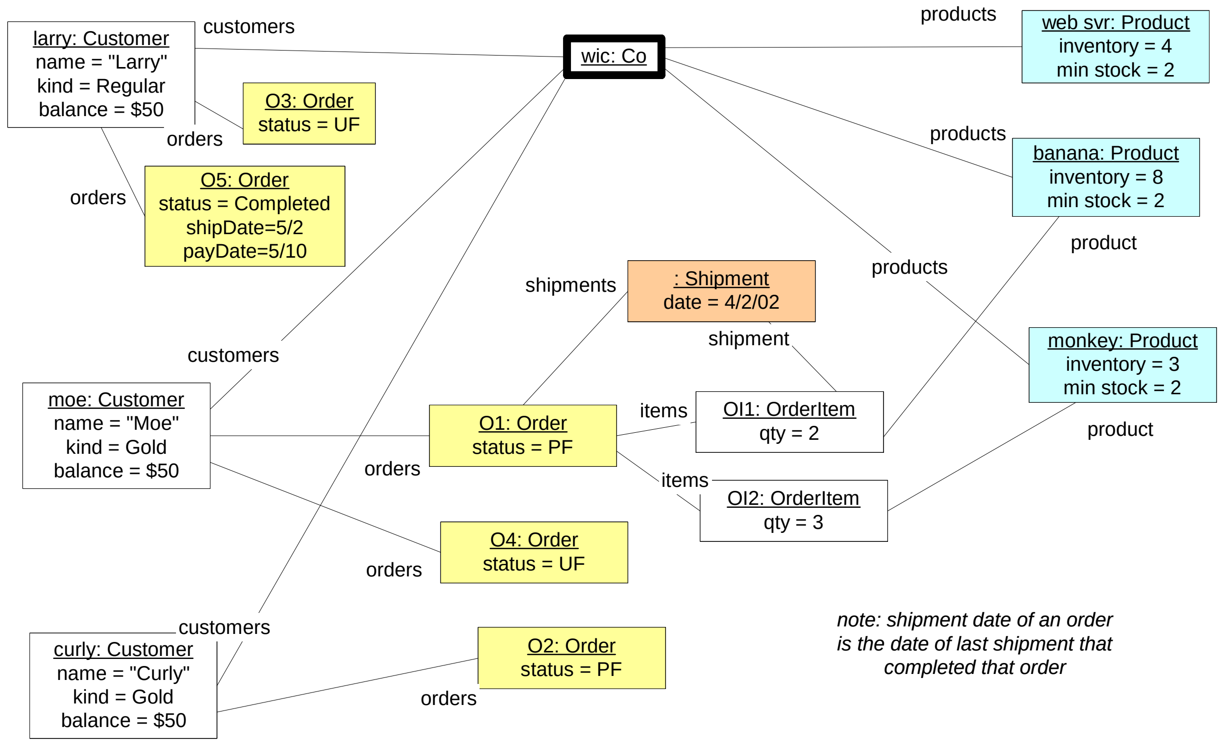
3.6 Activity Diagrams



4.1 Draw Object Snapshots, Build Information Model from Snapshots. Understand Navigation Expressions.

(a) Snapshots & User Interface

Note: information not shown in snapshot is simply omitted, rather than lost.

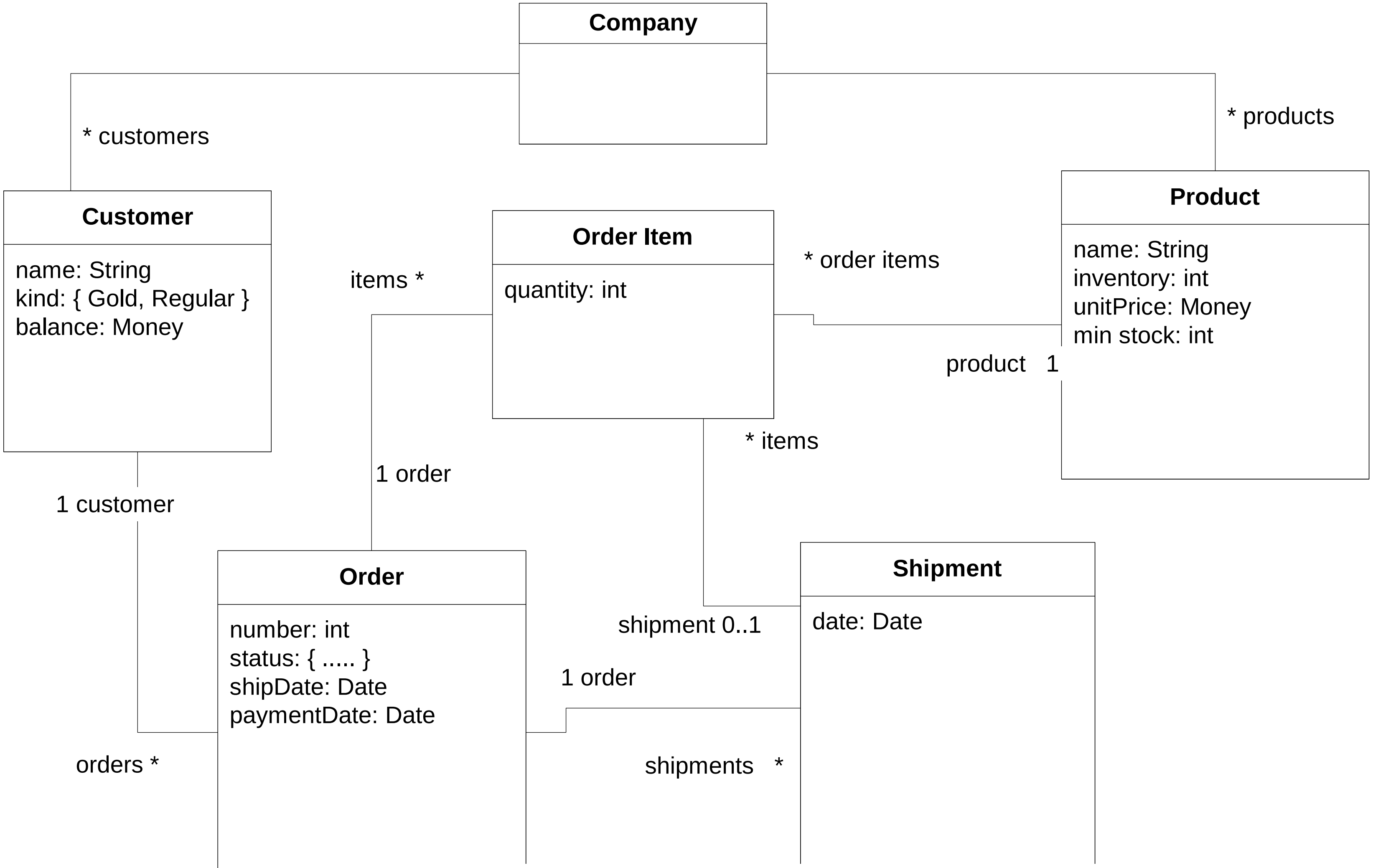


note: shipment date of an order is the date of last shipment that completed that order

Creation:

- Customer: when the company gets a new customer
- Order: when a customer places an order
- Order Item: when a new item is added to an order (as part of when a customer places an order)
- Product: when a new product is added to the catalog
- Shipment: when a shipment is made for some order items in the order

(b) Build Information Model from Snapshots



notes:
status = fulfilled if (and only if) all order items have a shipment
shipment Date defined only if status = fulfilled; shipment Date = date of last shipment
status = partially fullfilled if (and only if) at least one (but not all) order item have a shipment

(c) Interpret Navigation Expressions

1. wic.customers = { larry, moe, curly }
2. wic.customer **à** select(cust | cust . name = “larry”) = larry
3. larry.orders = *[any other way to refer to larry]*.orders = { o3, o5 }
4. larry.orders **à** select (ord | ord . status = Unfilled) = { o3 }
5. all wic orders placed by gold customers = { o1, o2, o4 }
6. all products which cost less than $25

(d) Write Navigation Expressions

1. wic.customers.orders **à** select (o | o.status = PartiallyFulfilled).customer
2. wic.orders **à** select (o | o.status = PartiallyFulfilled)

4.2 Convenience Attributes. Invariants. Supertypes.

a) Introduce convenience attributes

1. all products with inventory
   Company
   inStockProducts : Set(Product)
2. the current need for a product
   Product
   currentNeed : integer
3. percentage of orders with 2 or less shipments
   Company
   totalCompletedOrders : integer
   totalEfficientCompletedOrders: integer
   efficientlyShippedOrders : Percentage

b) write invariants

1. Company:: inStockProducts : Set(Product)
   all products whose inventory is greater than 0
   products à select (p | p.inventory > 0)
2. Product:: currentNeed : integer
   the sum of the ordered quantity for a given product's orderItems that have not been shipped.
   orderItems à select (item | not item.shipped) à sum(item | item.quantity)

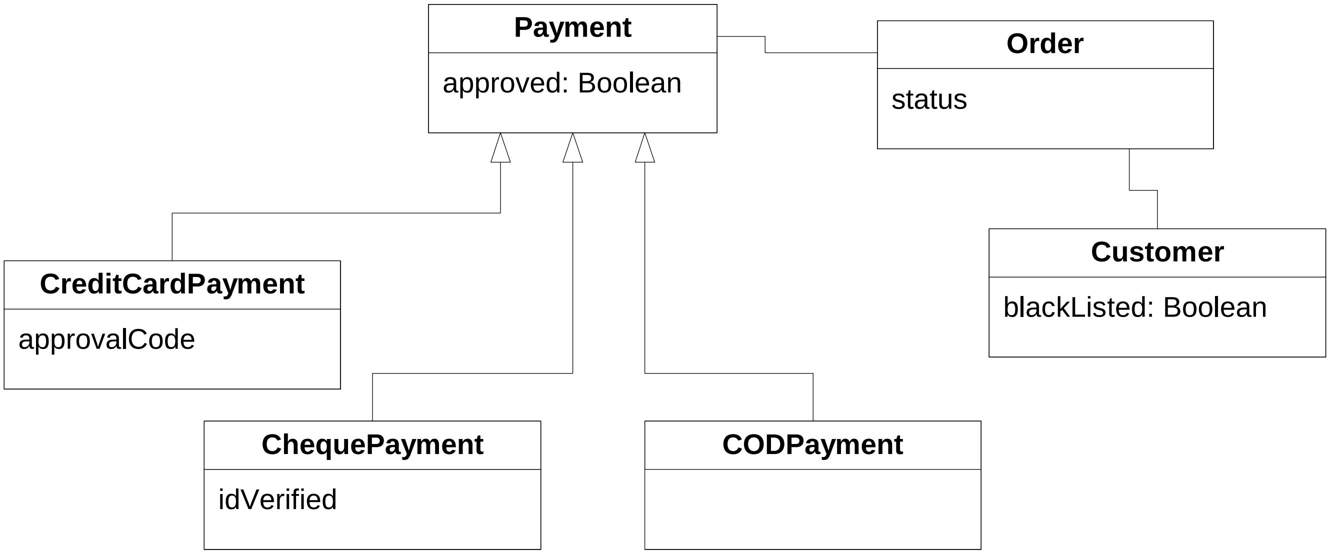
   (where item.shipped could be defined as, or replaced by, item.shipment <> null)
3. Company:: totalCompletedOrders: integer
   the count of the orders which have been completed.
   customers . orders à count (o | o.completed)

   Company:: totalEfficientCompletedOrders: integer
   the count of the orders which have been completed with 2 or less shipments.
   customers . orders à select (o | o.completed and o.shipments à size <= 2)

   (where o.completed could be defined as, or replaced by, o.shipDate <> null)

   Company:: shippingEfficiency : Percentage
   the ratio of totalEfficientCompletedOrders to totalCompletedOrders
   100 * totalEfficientCompletedOrders / totalCompletedOrders
4. for any order, there are never two or more orderItems which have the same product.

c) utilize supertypes



Invariant on Order: if payment is not approved then status is not fulfilled

there are different types of payments: by credit card, by cheque, by COD.

supertype is Payment

generic attribute – approved : boolean – is used to determine if an order can be fulfilled

subtypes and attributes needed

> CreditCardPayment – approvalCode on this type
> approved means approvalCode not null
>
> ChequePayment - idVerified on this type
> approved means idVerified = true
>
> CODPayment – blacklisted on Customer type
> approved means customer.blacklisted = false
>
> (where customer may be defined as order.customer)

4.3 Define Scenarios and Snapshot Pairs.

(a) Scenarios and Snapshot Pairs

1. the long-lived activity could be: **buy products**

   starts – when moe places an order with WIC.

   ends – when moe has received order in full and paid for it

2. the finer grained actions could be: place order, make shipment, pay for order.

   the type could be: Order.

   the states could be: unfulfilled, partiallyFulfilled, completed, paid, cancelled.

3. scenario: moe buys monkeys and bananas

   initial state: moe is an existing customer of wic. wic has 3 monkeys and 10 bananas in stock.

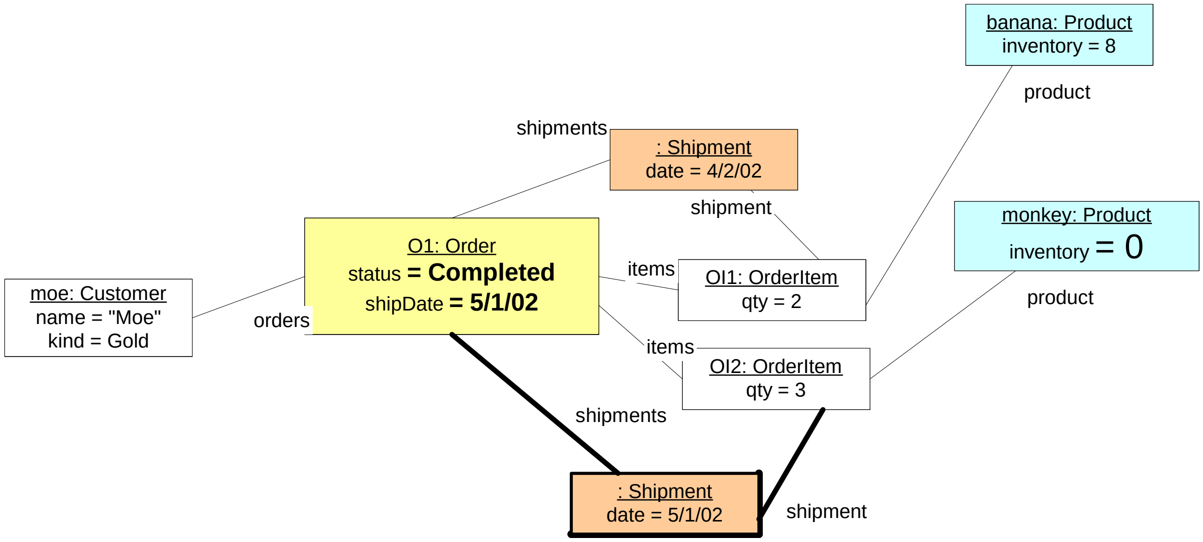
   scenario steps

   1. moe calls wic and places an order for 3 monkeys and 2 bananas.
   result - a new order exists for moe and has the items ordered.

   2. charley, a wic employee, gets 2 bananas from stock and ships them to moe.
   result - a new shipment associated with the bananas orderItem for moe’s order; inventory of bananas is reduced by 2.

   3. moe receives the bananas.

   4. sandra, a wic employee, gets 3 monkeys from stock and ships them to moe.
   result - a new shipment associated with the monkeys orderItem; monkey inventory reduced; order completed.



   5. moe receives the monkeys.

6. moe sends payment to wic for his monkeys and bananas.

final state: wic has fewer bananas and monkeys, more money. Moe has bananas and monkeys. Moe's order is completed and paid.

4. after step 1 – o1, oi1, oi2 and links between them exist; link from moe to o1 exists.
   after step 2 – sh1 and link to oi2 exists. banana.quantity goes to 6 from 8.
   after step 4 – sh2 and link to oi1 exists. monkey.quantity goes to 0 from 3.
   o1.shipdate and status updated.

5. possible interference or exceptions
   moe attempts to order items that are out of stock and cannot be obtained
   the shipments get lost or sent to the wrong location (will require changes to model)
   moe does not pay for the products he receives (might require time-based event and changes to model)

(b) Specify an Action

1. informal description for post-conditions

   makeShipment

   postcondition: the order has a new shipment with today's date and is associated with each shippable orderItem.

   each shippable orderItem is shipped.

   each shippable orderItem's product's inventory is reduced by the orderItem's quantity.

   if all orderItems are shipped, the order is completed and it's shipDate is today. otherwise, it is partiallyFulfilled (this could be moved to an invariant in the information model)

   cancelOrder

   postcondition: all unshipped orderItems are cancelled.

2. add parameters and clean up description

   makeShipment (o:Order)

   postcondition: the order has a new shipment; its date is today.

   each shippable orderItem is shipped and its shipment is the new shipment.

   each shippable orderItem's product's inventory is reduced by the orderItem's quantity.

   if all orderItems are shipped, the order status is completed and it's shipDate is today. otherwise, its status is partiallyFulfilled and shipDate is not defined.

   cancelOrder (o:Order)

   postcondition: all orderItems not shipped are cancelled.

3. add pre-conditions

   makeShipment

   precondition: order is not completed.

   there is at least 1 shippable orderItem.

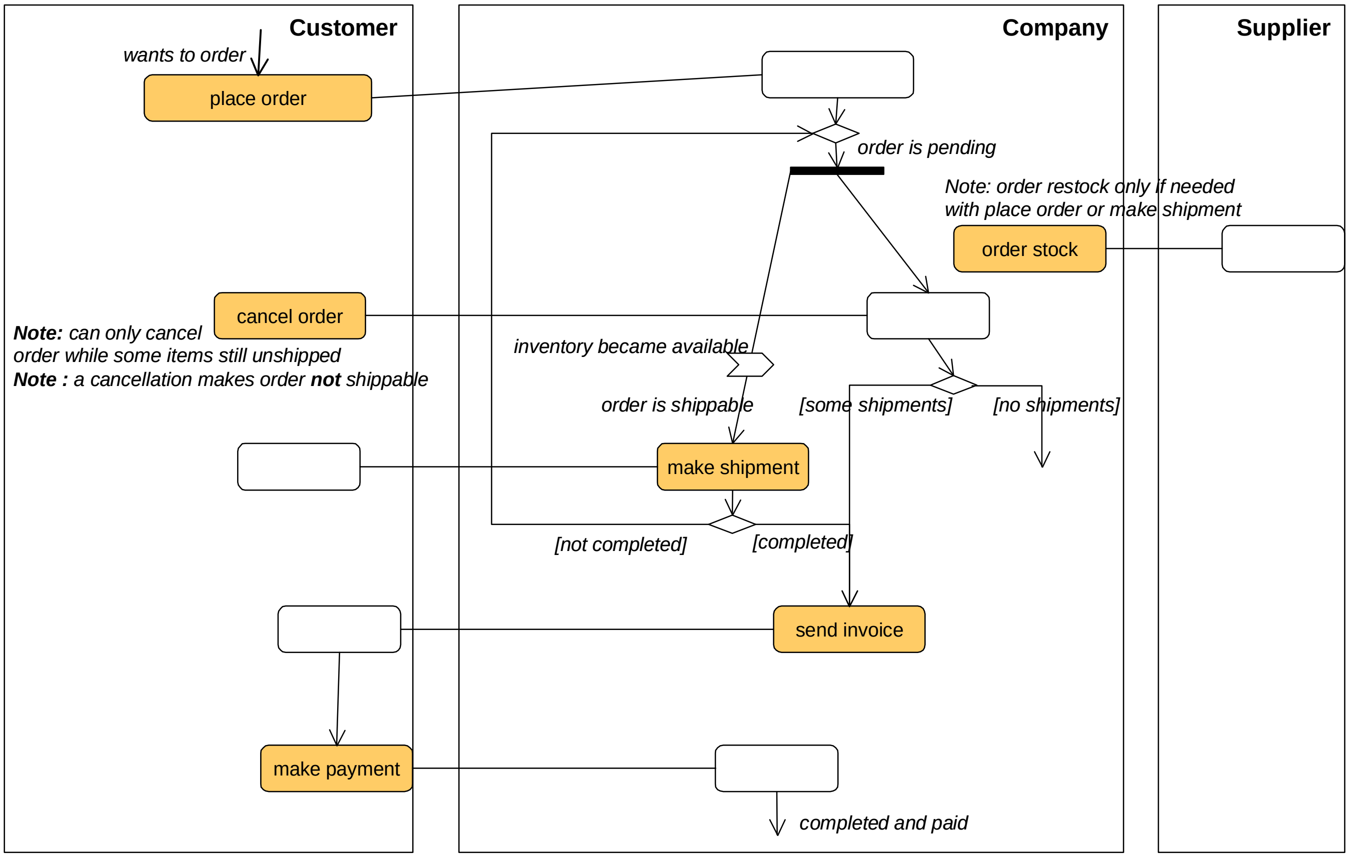
   cancelOrder

   precondition: order is not completed.

5.1 Build Activity Diagram

1.

Process: buy products
Goal: customer has received all items ordered and had paid in full; inventory maintained



Notes: the activity diagram above simply focuses on main flow, and makes some simplifications.
Note:

- order cancellation only accepted while there are still some unshipped items in the order.
- a customer may receive an invoice before the associated shipment.
- a customer may receive one or more shipments before receiving the order invoice.
- we have simply annotated supplier interactions for re-stocking requests and deliveries.

2. Activity Specifications

activity: make a shipment

roles: Company, Customer

inputs: order: Order from Company -- *company decides which order to ship*

outputs: shipment: Shipment to Customer

precondition: the order is not completed.

and there is at least 1 shippable orderItem for the order.

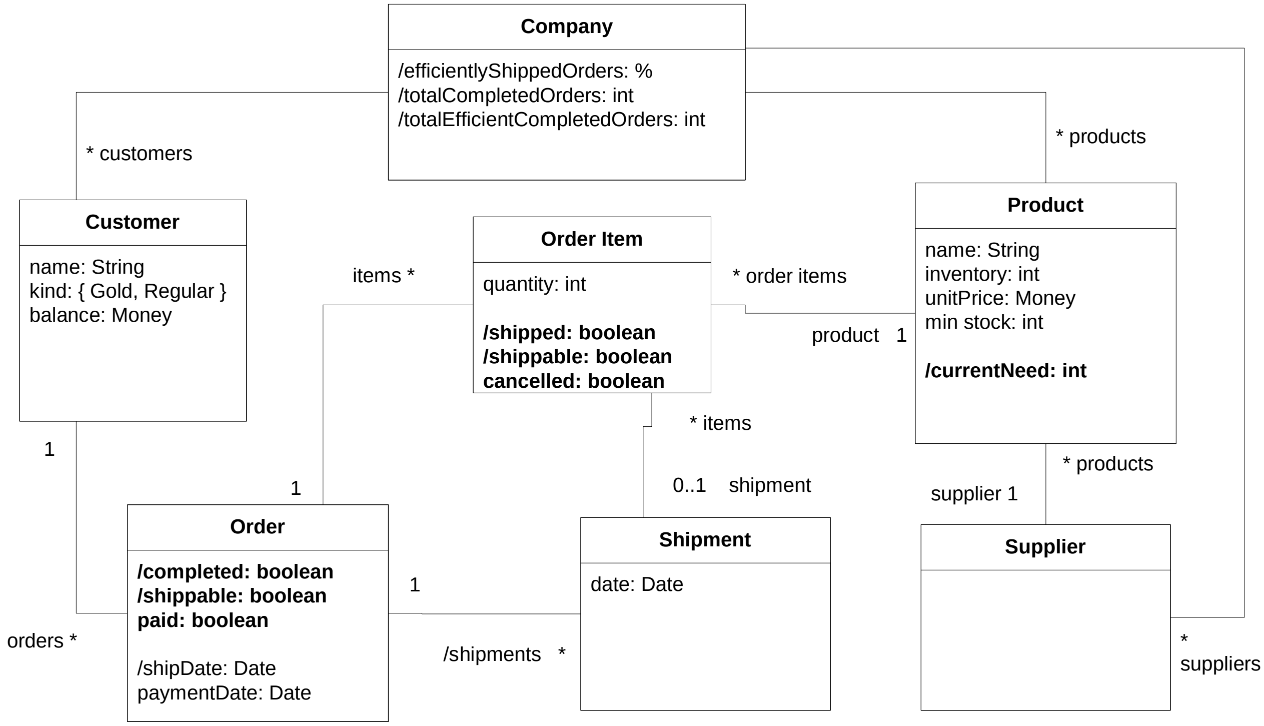
postcondition: all orderItems for the order that were shippable are included in a shipment.

Re-stocking ordered from provider of the shipped products if necessary.

3. Information Model

The types and attributes have business meaning and are not specific to a software system.

Customer	*The party for whom orders are placed*
Shipment	*A package of ordered items sent by the Company to the Customer in part or total fulfillment of the items in an order.*
Supplier	*The party who restocks some products for the Company*

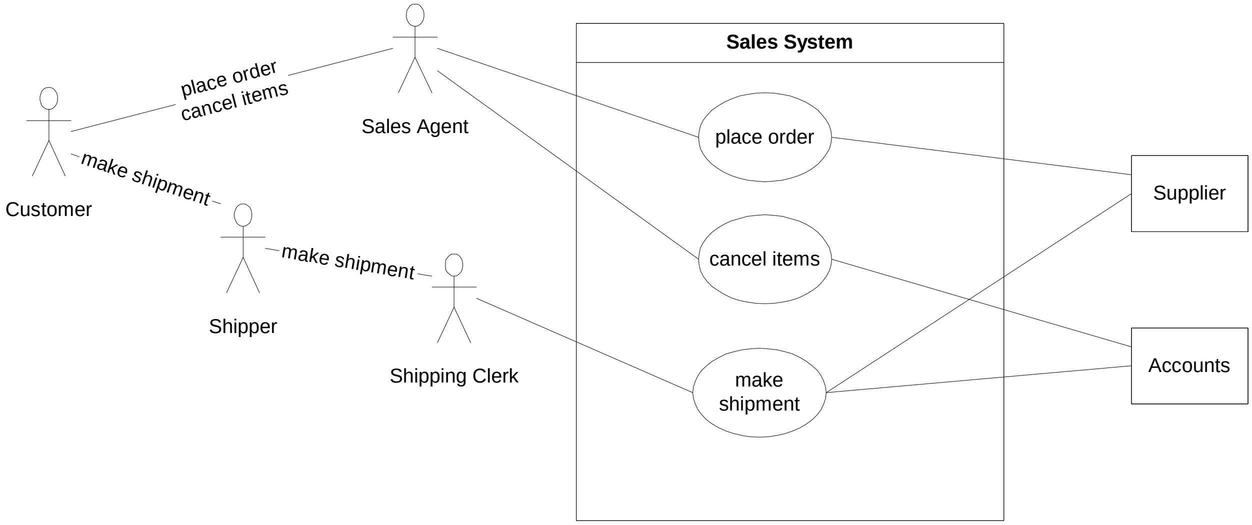


5.2 Build a State Diagram & Definition Matrix

include figure here

6.1 Use Case Specification

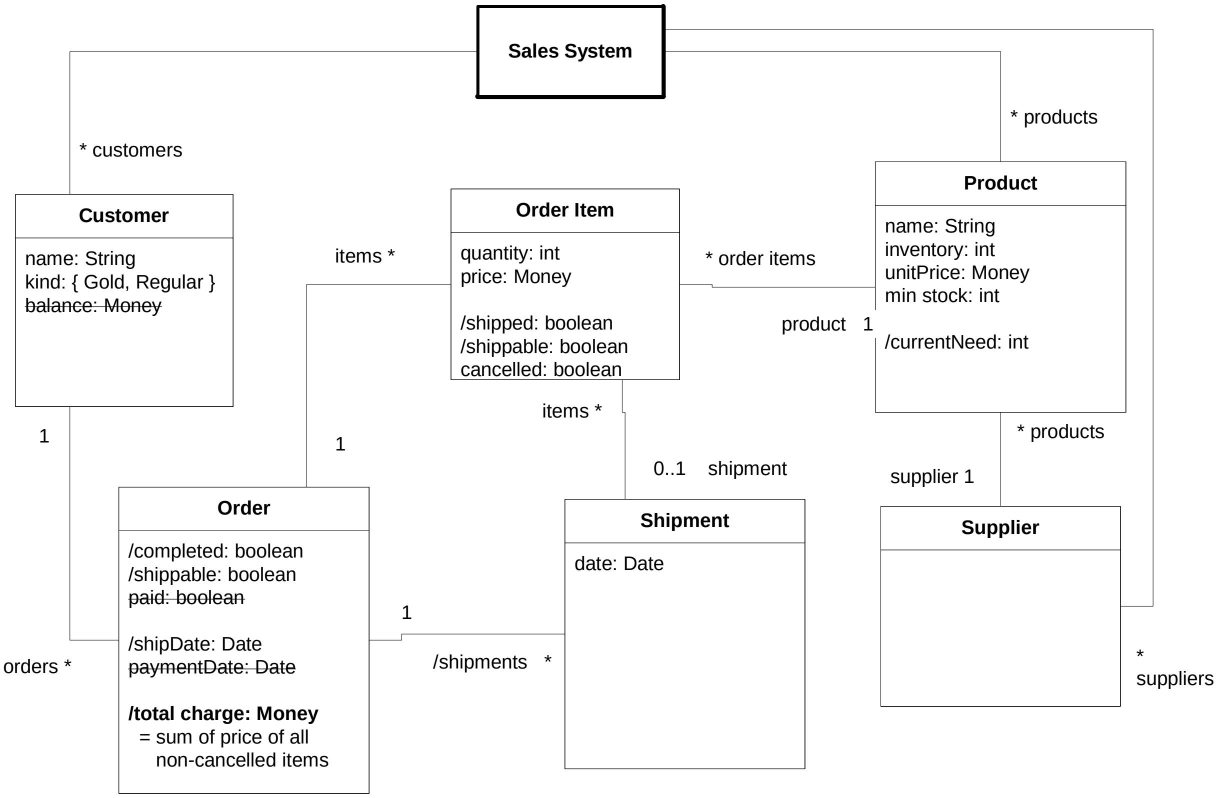
1) use case diagram



Note: We could have separated out two use cases, and **included** them in the main uses cases.

- *restock*
- *charge account*

2) Black Box Information Model



3) use case specifications

Use case: place order

Actors: Sales Agent, Customer, Supplier

Inputs: Customer, list of products with quantities **from** Customer **via** Sales Agent

Outputs: order number **to** Customer **via** Sales Agent

Trigger: Customer calls to place an order.

Precondition: products in company catalog

Postcondition: a new order exists for the customer for quantities of products ordered.

A new customer added to sales system’s customers if necessary.

Re-stocking has been ordered from supplier if necessary (could formalize in terms of product currentNeed attribute being greater than inventory)

Note: it could be useful to model a CustomerQuery type as the input type; and perhaps split out an IdentifyCustomer (sub) use case.

Use case: make shipment

Actors: Shipment Clerk, Customer, System, Supplier

Inputs: order **from** Shipment Clerk

Outputs: shipment (manifest) **to** Shipment Clerk

Trigger: Shipment Clerk selects an order to make a shipment for.

Precondition: order is not completed.
there is at least 1 shippable order item for the order.

Postcondition: a new shipment exists and contains all shippable order items for the order.

Re-stocking has been ordered from Supplier if necessary (could formalize in terms of product min stock and inventory attributes)

If order completed then Accounts System has been notified of order total charge

Use case: cancel order

Actors: Sales Agent, Customer, System

Inputs: order **from** Customer **via** Sales Agent

Outputs: cancellation number **to** Customer **via** Sales Agent

Trigger: Customer calls in decision to cancel all the remaining order items.

Precondition: order is not completed.

Postcondition: all order items not shipped are now cancelled.

If order completed then Accounts System has been notified of order total charge

6.2 Describe Steps and Alternate Paths

Use case: place order

Actors: Sales Agent, Customer, System

Inputs: Customer, list of products with quantities from Customer via Sales Agent

Outputs: order number to Customer via Sales Agent

Trigger: Customer calls to place an order (needs products).

Precondition: products in company catalog

Postcondition: a new order exists for the customer for quantities of products ordered.

A new customer exists if necessary.

Main (success) steps:

1. customer calls agent to place order
2. agent: include lookup customer
3. agent looks up the product catalog on the system, describes product options to customer
4. customer informs agent of product and quantity
5. agent creates order in system
   - i. agent enters one order item
   - ii. system confirms availability, delivery time, and cost for order item
   - iii. agent agrees order item information with customer
     Repeat I-iii until order items complete
6. system confirms order provides order number, total cost
7. agent confirms with customer

Alternate paths:

#2. *Customer not found:*
1. agent: *include Create Customer*
2. resume at 2

#5: *Not enough inventory:*
1. Sales System detects that restocking is required based on product need
2. Sales System requests restock from provider for those products

6.3 Identify & Specify Use Case Operations

Use case: place order

Operations from steps

1. lookup customer (customer info)
2. get catalog ( ) *(presents catalog to agent e.g. for discussion with customer)*
3. get product details (product)
4. …
5. create order (customer, products and quantities)
6. … *(note: you need to fill exactly 11 steps)*
7. …
8. …
9. …
10. …
11. <<out>> re-stock (product, quantity) *(request to a supplier)*

Use case: make shipment

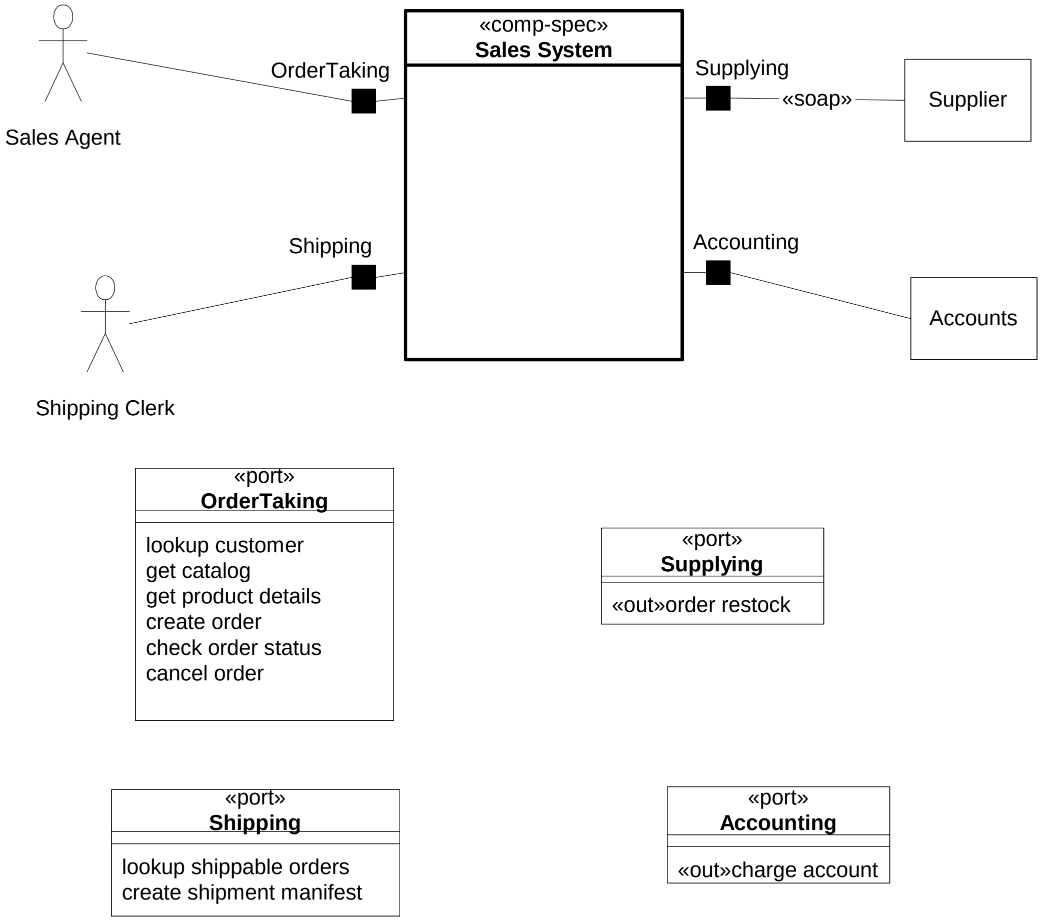
Operations from steps

… lookup shippable orders

… create shipment manifest

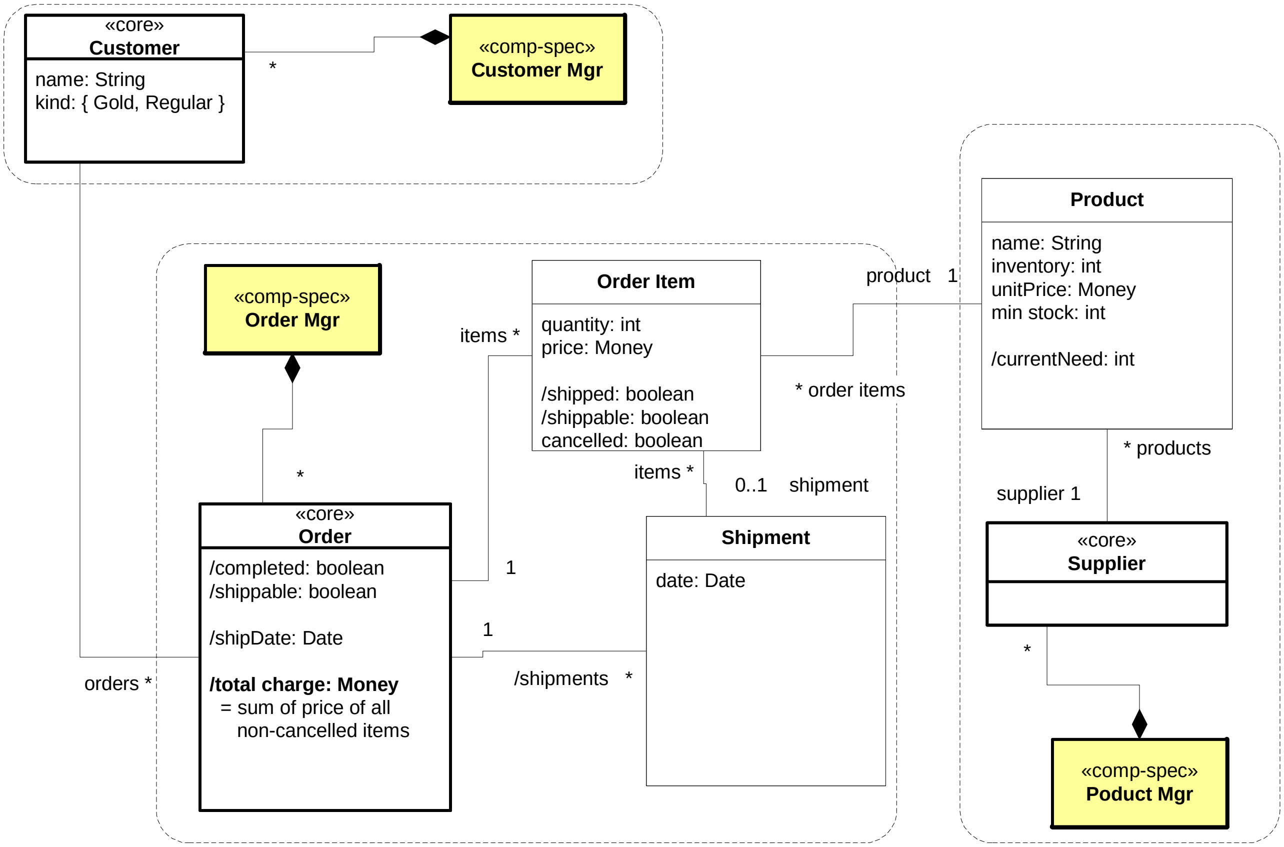
…

6.4 Black Box Ports and Assembly



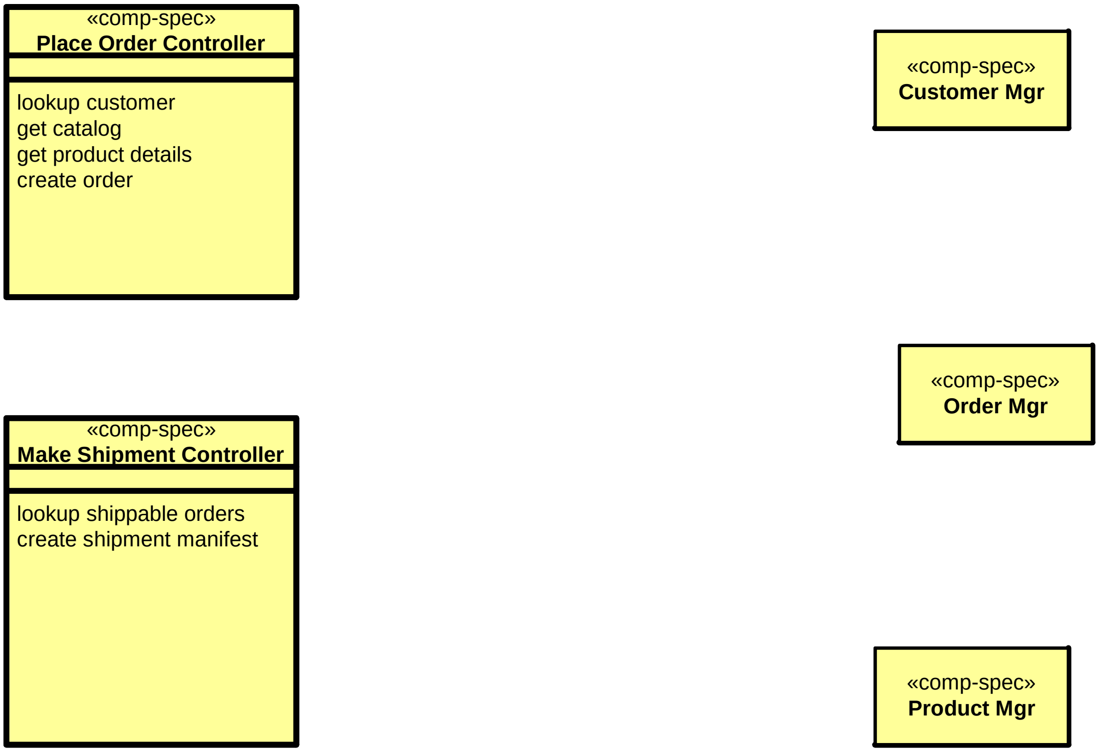
7.1 Draft Core Type & Use-Case Based Components

a) find core type components



b) ***add use-case based components***

Note: These can be separate interfaces of a single use-case component, particularly if using a generic workflow-like co-ordinator.



«comp-spec»
Place Order Controller

lookup customer
get catalog
get product details
create order

«comp-spec»
Make Shipment Controller

lookup shippable orders
create shipment manifest

«comp-spec»
Customer Mgr

«comp-spec»
Order Mgr

«comp-spec»
Product Mgr

Appendix Exercise 1: UML Models

Instructor Example

1. object snapshot – describes some objects, relationship values, attribute values
2. class diagram – describes structure & constraints
3. use case diagram – describes interactions between actors, systems
4. sequence or collaboration diagram – describes interactions between objects
5. state diagram – describes states & changes the object undergoes
6. package diagram – describes structuring of models and work
7. deployment diagram – describes nodes, applications, programs, etc.
8. pattern – describes recurring design and its application

Internet Portal Case Study – "CAM Light"

Business Architecture

Note: this business architecture shows a bit more system-oriented properties than normal.

a) Types	Attributes, description
Invite	Invitees, event,
Calendar	Collection of events (do events overlap?)
Email message	Relevant because invite is delivered via email message. Attributes: email address, subject, content, attachment
Email system	Inbox, outbox
InviteResponse	Yes, No, Maybe + message, Invitee
Invitee	Is this really a kind of contact?
Host	Member of the portal
Location	Postal address
Directions	Text and graphical description of how to get from some location to another location (perhaps contact.address to event.location)
Event	Description, date/time range, location
AddressBook	Collection of contacts
Contact	Email address, postal address, phone numbers, notes, isPortalMember
SavedLocations	
PortalAccount	SavedLocations, Contacts, Calendar, Invites, Email system

Scenario

Name: Celebrate USA even still playing in World Cup party

Description: Party at The Hogs Head with all students to celebrate Bruce Arena

Initial state: George is a member of MegaPortal. Martin and Ian are too, but Chris and Sean are not. Martin and Chris are in George's contact list but the others are not. The Hogs Head is on George's list of locations.

1. George opens the MegaPortal page for Invites on Monday.
2. George creates a new Invite, specifying that it will take place at the Hogs Head on Friday at noon. He adds Martin and Chris to the list of attendees from his contacts list. He also adds Sean and Ian to the invite, causing them to be added to the contact list.

3. The system sends out an email to all the invitees giving the description of the event, a link to the MegaPortal page for this invite, and an attachment with map of the event location.
4. Chris receives the email, follows the link, and issues his "No" answer, noting that he will be having a liver transplant that day.
5. Ian receives the email, follows the link, and accepts the invite. Since he is a member of MegaPortal, this event is put onto his personal calendar.
6. Sean receives the email, follows the link, and reports "Maybe". He notes that he has a potentially reschedulable dentist appointment.
7. Martin receives the email, follows the link, and reports "Yes". Since he is a member of the MegaPortal, the event is added to his calendar. He requests customized directions from his saved work address to the event.
8. George checks the status of the Invite and finds two Yes, one No, and one Maybe.
9. On Friday morning the system sends out reminder emails to the Yes and Maybe invitees (not Chris).
10. On Friday, the party happens and the Invite is marked as complete by the system.

Final state: The Invite is complete, emails have been sent to all participants and responses have been recorded by the system.

Action	**Description**
Create invite	List of attendees is collected, event is scheduled, invitees are optionally added to the contact list, and email is sent to all invitees
Respond to invite	Response of Yes, No, or Maybe plus message is recorded. If respondee is a member then his calendar is updated with this event. An email confirming is sent to the event host.
Check invite status	Boring
Send reminder email	By default, the day of the event, the system sends reminder emails to all invitees who have responded Yes and Maybe.
Deactivate invite	The invite is marked as complete after the end of the event

Black Box Architecture

Use Case Name	**Business Trigger and Source**	**External Actors**	**End/Success Goal**
Create Invite	Host/PortalMember request	Non-Member Invitees, Member Invitees, Calendar	Invite sent to all invitees via email, Portal page for this

		system, Email System, Contact System, Location System, Memberships	invite, event added to host's calendar
Respond to Invite	Invitee visits website via link	None	Status of the invite updated with InviteResponse
Check invite status	Host visits via link	None	View current invite status
Deactivate Invite	Time of event passes	Calendar System,	
Cancel event	Host visits via link	All invitees, Calendar system, Email System	Status of invite updated to canceled, invitees notified of cancellation via email, event removed from member calendars
Send reminder	Time of event is near	Email system, all invitees, host, calendar system	Email sent to all invitees with reminder

Port	**Operations / Description**
Host	Create invite, add contact, add location, add event, cancel event, check invite status, CRUD
Invitee	Respond to invite, check invite status
~Calendar	Add event, remove event, change event status
~Email	Send email
~Contact	Add contact, list contacts
~Location	Add location, list locations, get directions
~Membership	Lookup members for invitation to event and for integrated services

White Box Architecture

SubComponent	Description
Create Invite	Use case component, works with Invite manager and Reminder

	manager, ~calendar, ~contacts, and ~location (to pre-populate lists and possibly to add to each of these)
Respond to Invite	Use case component, works with Invite manager
Cancel Invite	Use case component, works with Invite manager and ~calendar and Reminder manager
Invite Manager	Stores invites, invite responses, references to host, invitees, location. Works with ~Calendar, ~Contact, and ~Location, but only references entities in these systems and does not modify their entities. Also works with ~Email.
Reminder Manager	Keeps list of reminder events, sends email to the remindees at set time before the reminder event. This component could be used generally across the portal, but we are assuming for now that it had not yet been created. Works with ~Email.

Technical Architecture

Try mapping into J2EE Architecture

Component	**Mapping to J2EE**
Create Invite	Split between UI parts in JSP/Servlet and core business function in a stateless session bean.
Respond to Invite	Split between UI parts in JSP/Servlet and core business function in a stateless session bean.
Cancel Invite	Split between UI parts in JSP/Servlet and core business function in a stateless session bean.
Invite Manager	Stateless session beans, with their data stored in the db directly.
Reminder Manager	Business function maps to Stateless session bean. Since EJB's cannot spontaneously generate events, it is also mapped to an external timer that sends messages every minute.