DESIGN MODEL

OVERVIEW

PROJECT GOAL

- The goal is to create a design and a prototype for a software system
- We will cover many different diagram types in class and talk about when and how to use them
- But not all of these diagrams and their different level of abstraction are used in one design model
  - The team usually decides which diagrams are needed and useful
  - It also decides on the level of abstraction
  - We will give you specific guidelines to help you learn a design process and will elaborate on some things a bit more than it would be done in industry but the idea is to help you understand so you are able to choose the right level of abstraction and diagrams after you are done with this course.

PROJECT GOAL

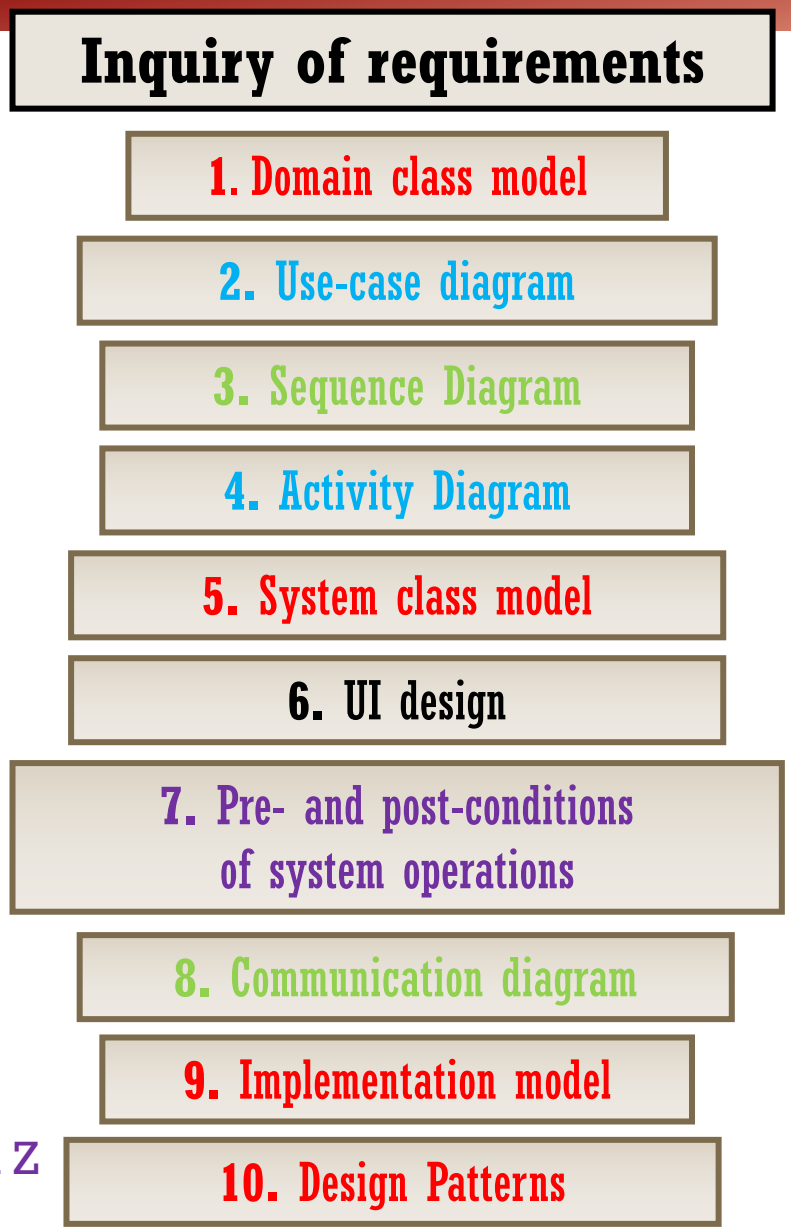
- **Design model: A design model consists of different diagrams that depend on each other and belong together and describe the software system –** create them in one Astah file (you can also use LucidChart but this tool might not give you any consistency help).

- This presentation gives an overview of the diagrams and level of abstraction you should use in your project and how to keep the diagrams in the design model consistent.

- I am not saying this document is complete but it hopefully helps.

- Important that you do not go for a divide and conquer approach (all diagrams have to belong together and be consistent)

Overview of Process



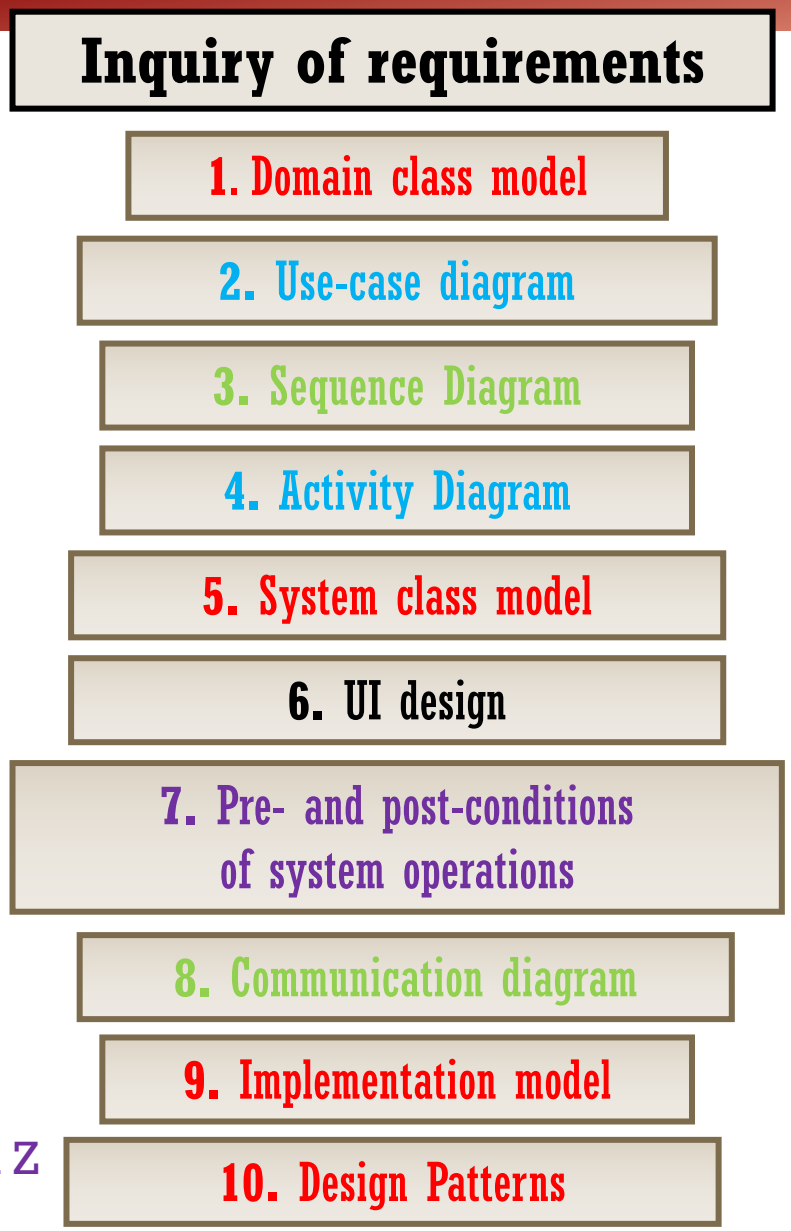
SHOP NOTES

Notes from the first requirements elicitation meeting with the shop owners and employees

- system should help to keep track of currently available products in shop
- the system should support selling products to its customers
- system should support ordering articles from warehouse, e.g. if they are sold out
- system should be able to handle customer data, e.g. add/delete customers
- each employee has to login before working with the system
- the employees of the shop should handle the customers' purchases
- customers do not interact with the system directly but are served by employees
- each product has a unique ID and is available in a certain quantity in the shop
- customers can request products in a certain quantity through an employee
  - If the product is available in the requested quantity the product is sold and the available quantity of the product is decreased correspondingly
  - If the product is not available in the requested quantity a purchase order is created for the customer and sent to the warehouse. For each customer the number of purchase orders is restricted to five.

AFTER WEEK 2

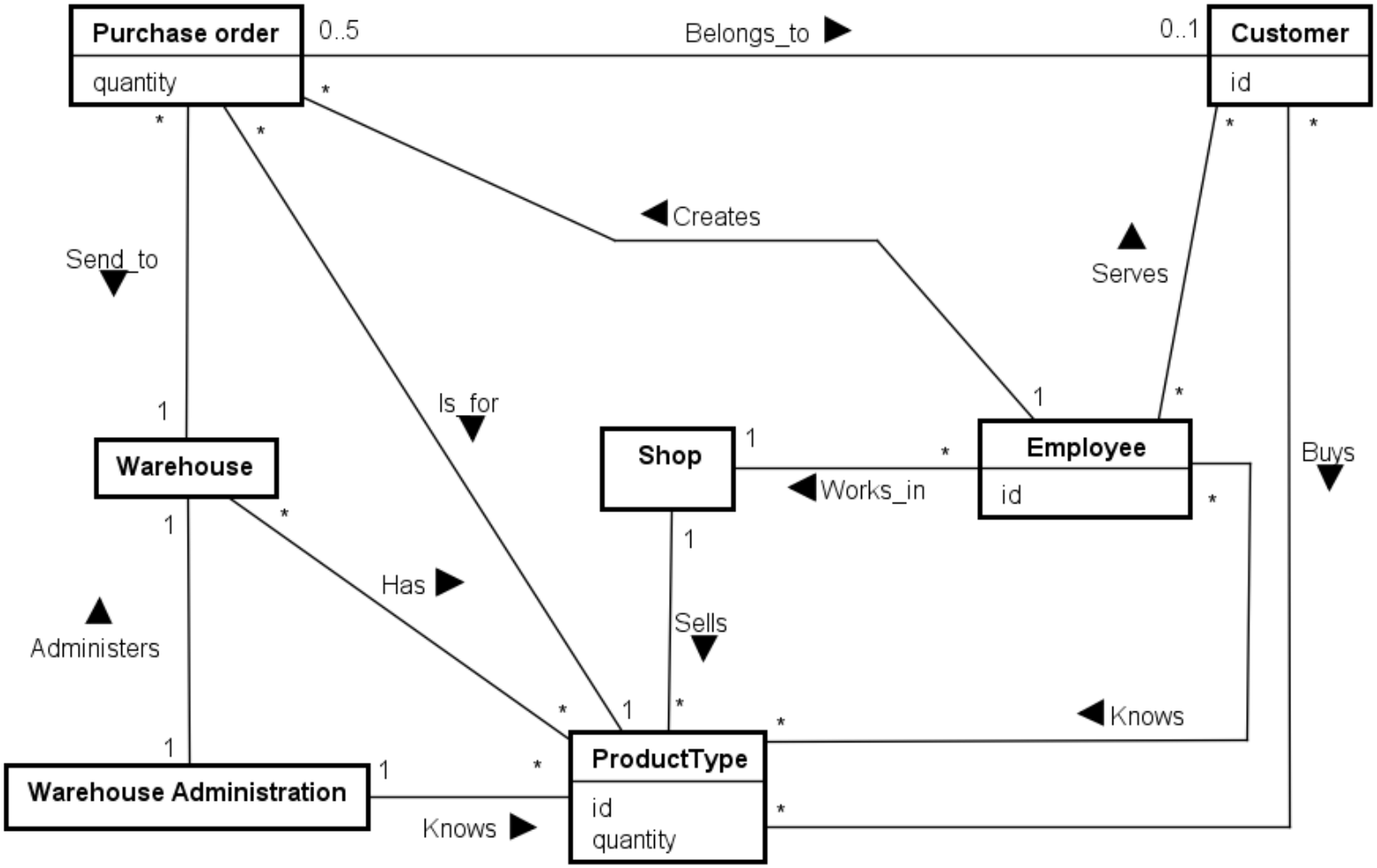
Overview of Process



DOMAIN CLASS MODEL

- Classes are all the things in the system that are relevant
- Relation between things are specified as associations (aggregation, composition as well)
- Generalization is also possible
- Model includes
  - Association names
  - Class names
  - Multiplicities
  - Basic attributes
- Classes in the diagram do not have to relate to a class that will be implemented later (but should be relevant to the system)
- A class can be a person, an object, a dataset etc.
- Always consider what is important for the system
- Do not model processes but the static view of the system
- It makes everything explicit

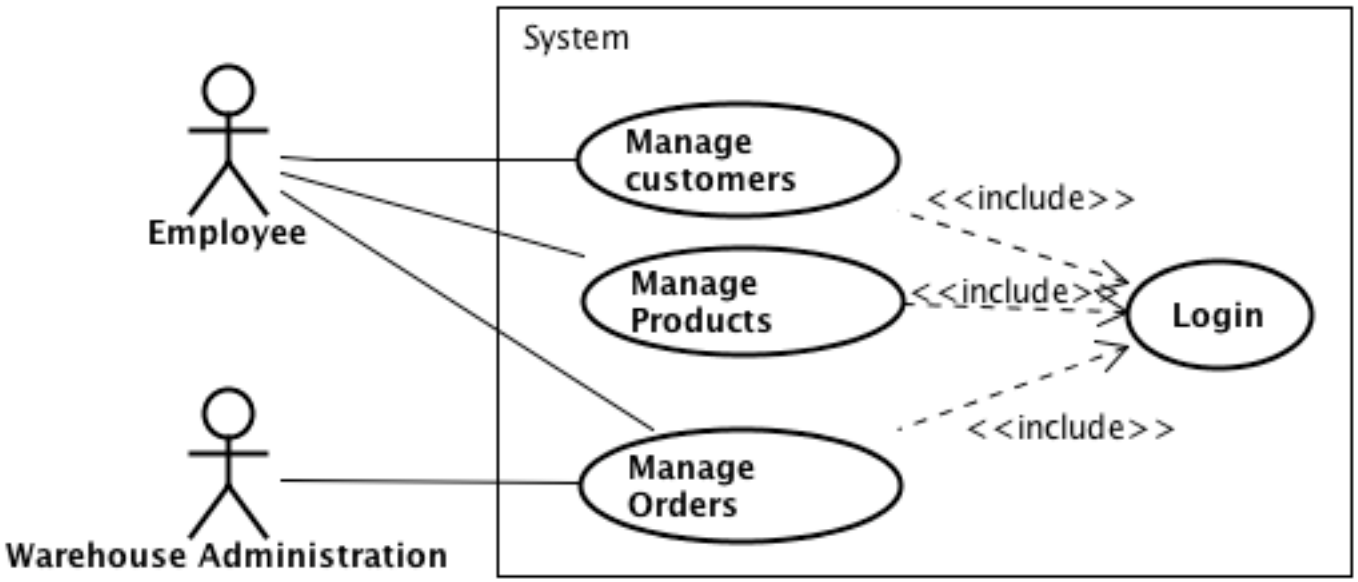
SHOP FROM LECTURE



USE-CASE DIAGRAM

- Create one Use Case model (Use Case diagram + Documentation)
- You need 1 Use Case diagram with 3-8 Use Cases for your project
- The Use Cases together should describe the functionality of the whole system
- Each Use-Case can have 1-n Scenarios (system requests/system operations)
  - Write down which Scenarios belong to each Use Case (Use Case documentation)
- Use Case Diagram has all the systems actors in it
  - An actor has to be in the domain class model as class if it is not it is either not an actor or your domain class model is wrong!
  - An Actor is someone who interacts with the system

EXAMPLE



Manage customers:
- Add customer
- Delete customer

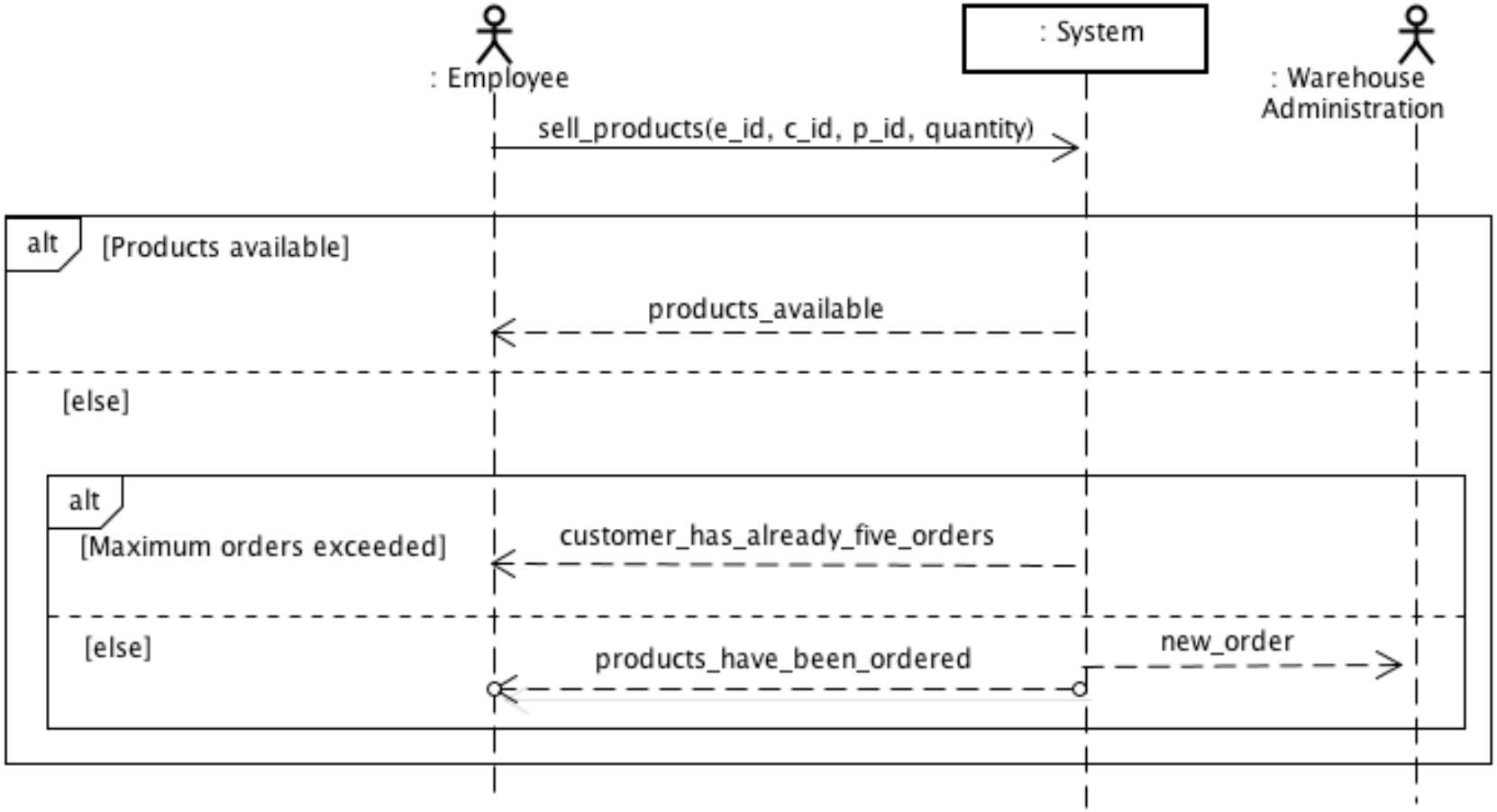
Manage orders:
- sell products
- create purchase order

Manage products:
- list products
- add/delete products

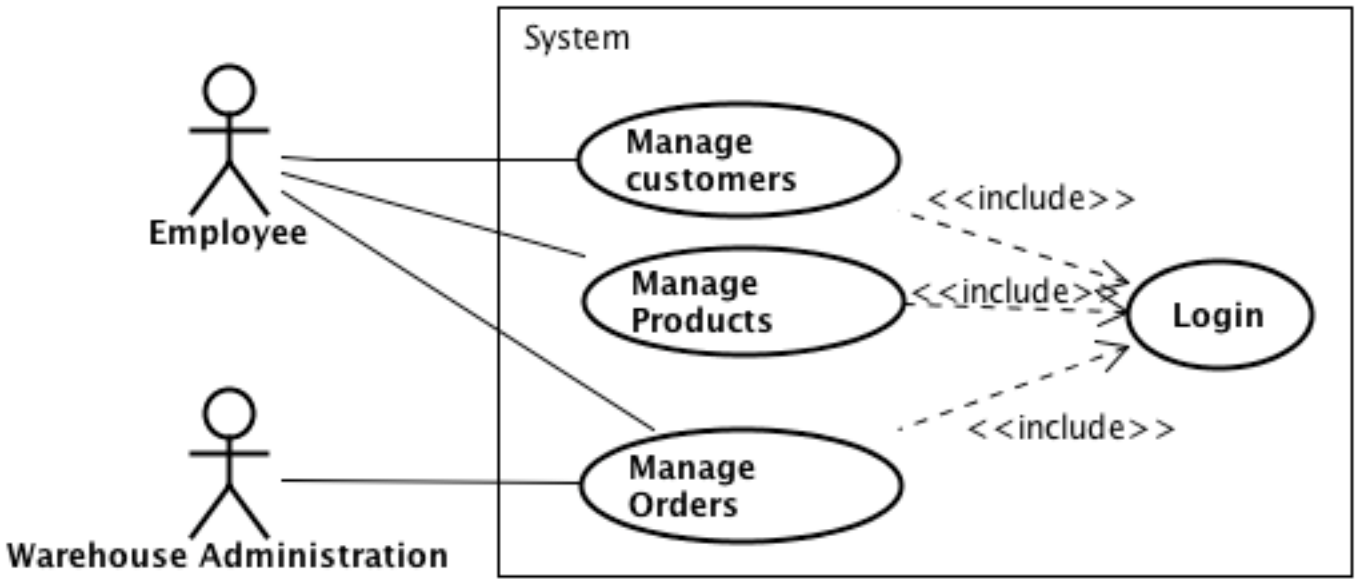
SEQUENCE DIAGRAM

- On a high level of abstraction (only user-system interaction)
- One Sequence Diagram for each Scenario (a Use Case can have many Scenarios)
- Only one arrow should go to the system (if you need more, then it is another scenario ➔ another Sequence Diagram)
- System operation should have necessary information as arguments
- Only the communication form the actors with the system
  - Do not distinguish what part of the system the actor communicates with (this will happen later)
- Do not include object communication for your project
- An actor in the Sequence Diagram has to be an actor in your Use-Case diagram
- If there is communication between an actor and the system in a Scenario there needs to be a relation between that actor and the Use-Case the Scenario belongs to

EXAMPLE


: Employee
: System
: Warehouse Administration
sell_products(e_id, c_id, p_id, quantity)
alt
[Products available]
products_available
[else]
alt
[Maximum orders exceeded]
customer_has_already_five_orders
[else]
products_have_been_ordered
new_order

EXAMPLE



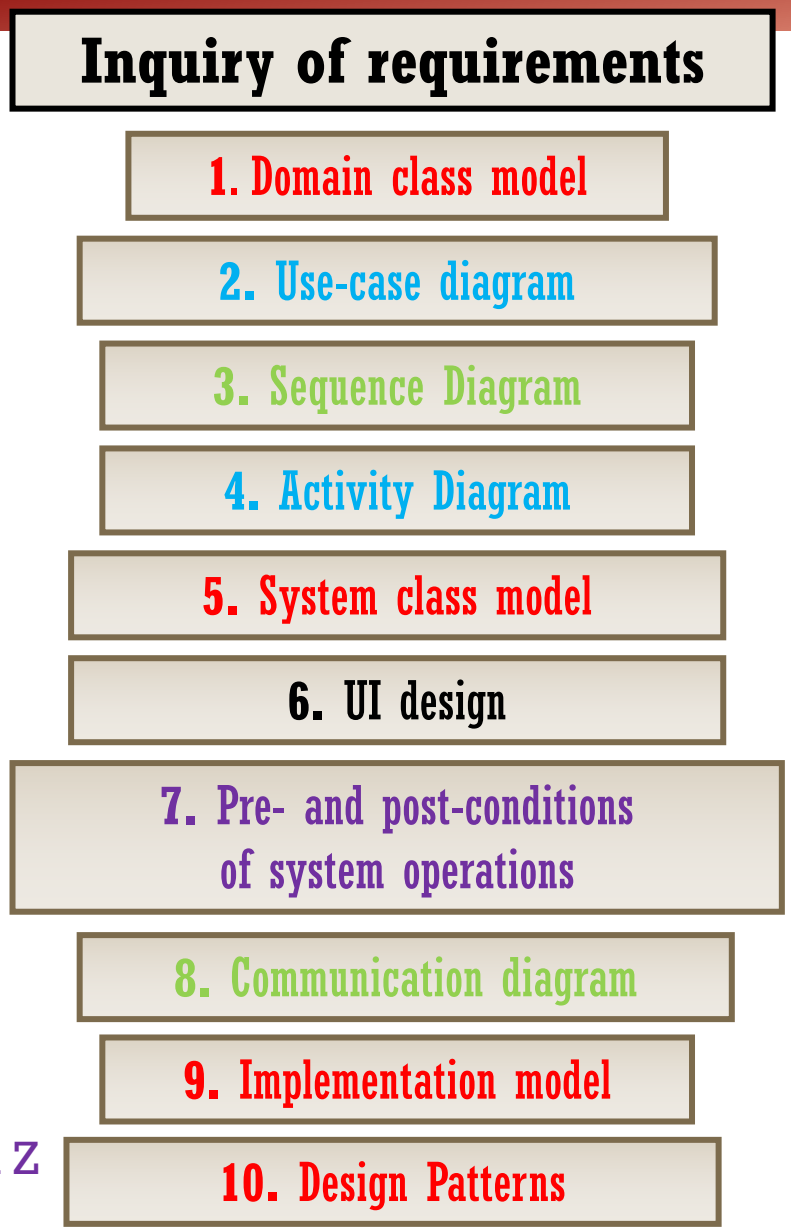
Manage customers:
- Add customer
- Delete customer

Manage orders:
- sell products
- create purchase order

Manage products:
- list products
- add/delete products

AFTER WEEK 3

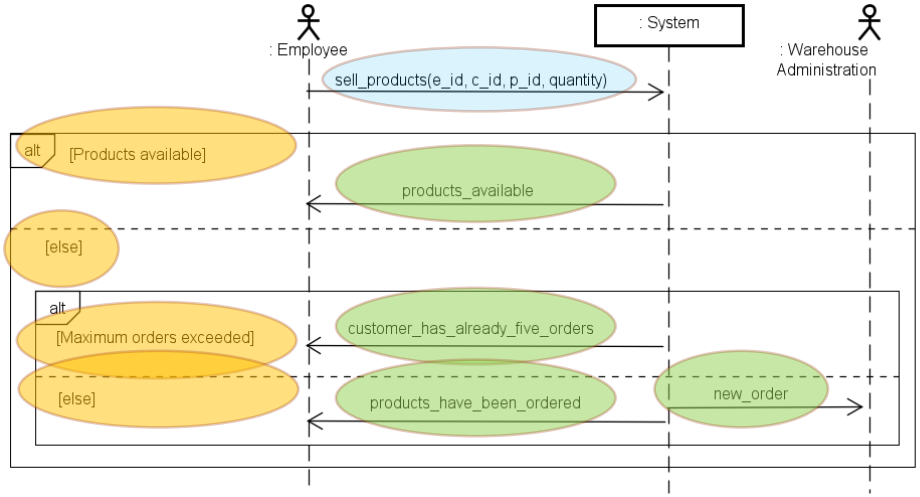
Overview of Process

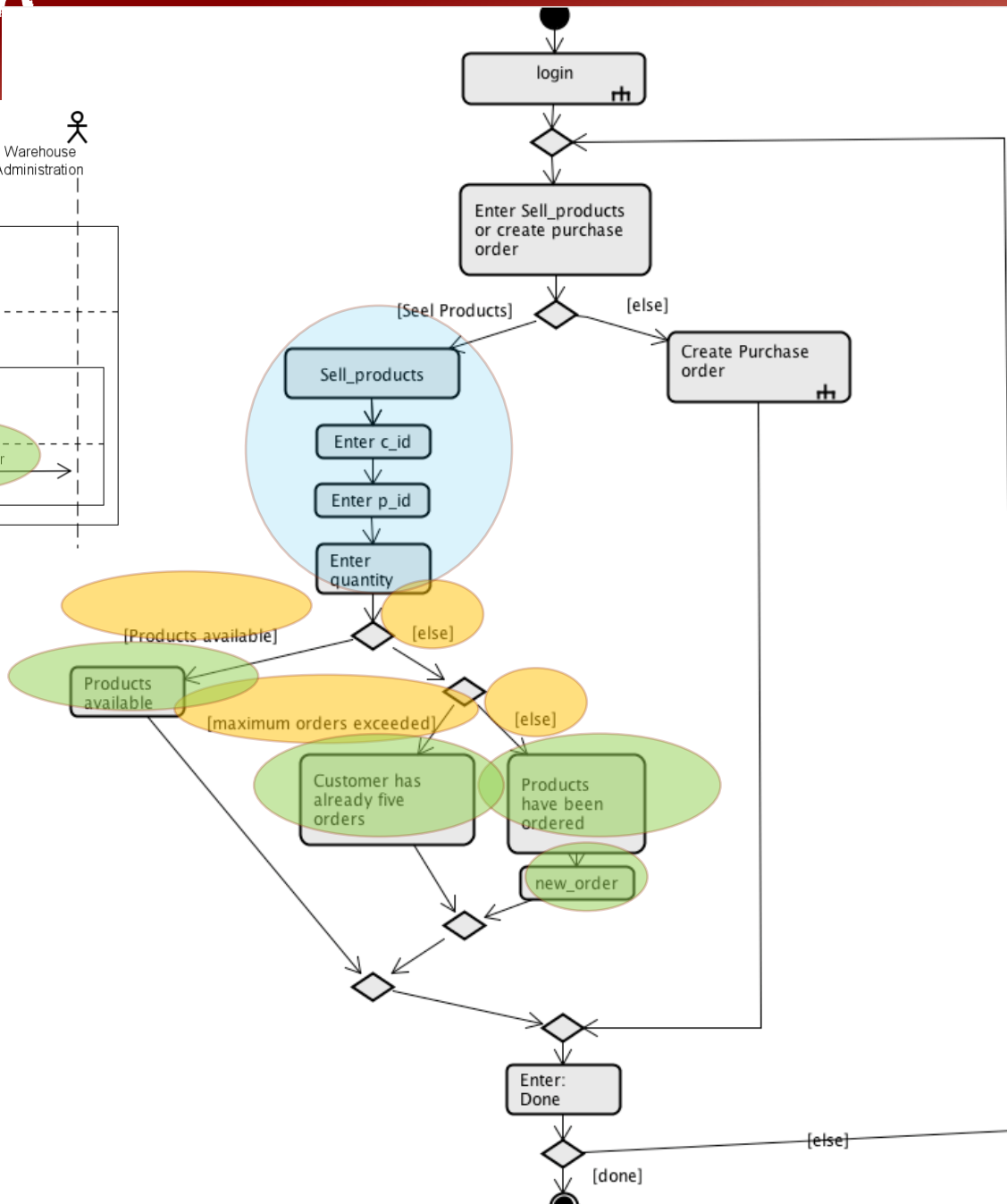


ACTIVITY DIAGRAM

- Create one Activity diagram for each Use Case – Actor relation
- An Activity diagram set the Sequence Diagrams into relation
- The events and system operations need to be consistent to the SD
- If you do not have an SD for a scenario which needs to be in the AD then you can abstract it
- Make sure the AD is consistent to the SDs and the Use Case model

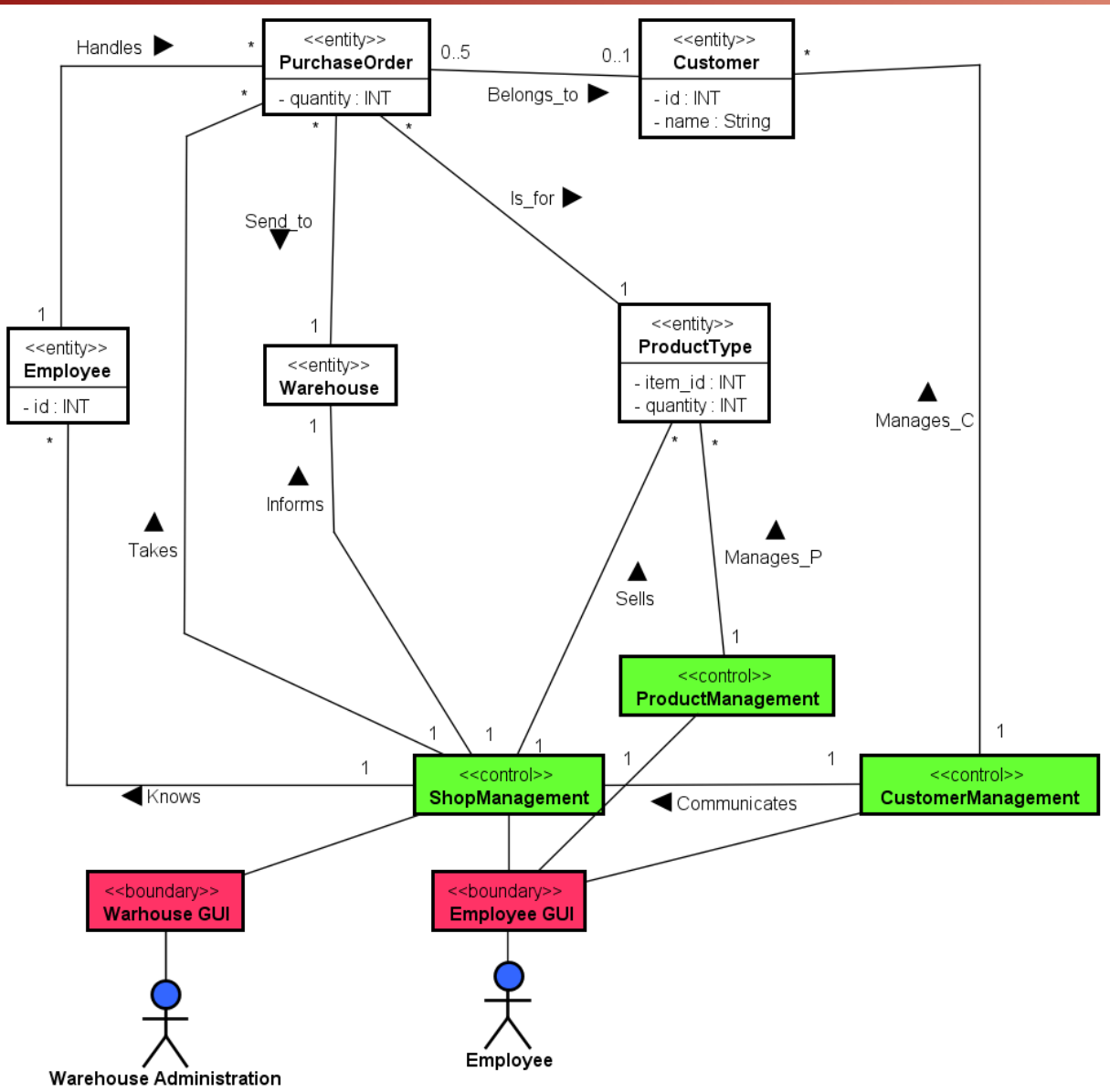
ALTERNATIVE





SYSTEM CLASS MODEL

- Is derived from Domain Class model
- Also uses information form Use-Case diagram
- Uses Stereotypes
  - Actor from Use-Case diagram needs to be actor in System Class model
  - For each actor one Boundary class
  - For each Use-Case one Control Class
  - Rest are usually Entities


Handles
<<entity>>
PurchaseOrder
- quantity : INT
0..5
0..1
Belongs_to
<<entity>>
Customer
- id : INT
- name : String
Send_to
Is_for
<<entity>>
Employee
- id : INT
<<entity>>
Warehouse
<<entity>>
ProductType
- item_id : INT
- quantity : INT
Manages_C
Informs
Takes
Sells
Manages_P
<<control>>
ProductManagement
Knows
<<control>>
ShopManagement
Communicates
<<control>>
CustomerManagement
<<boundary>>
Warhouse GUI
<<boundary>>
Employee GUI
Warehouse Administration
Employee