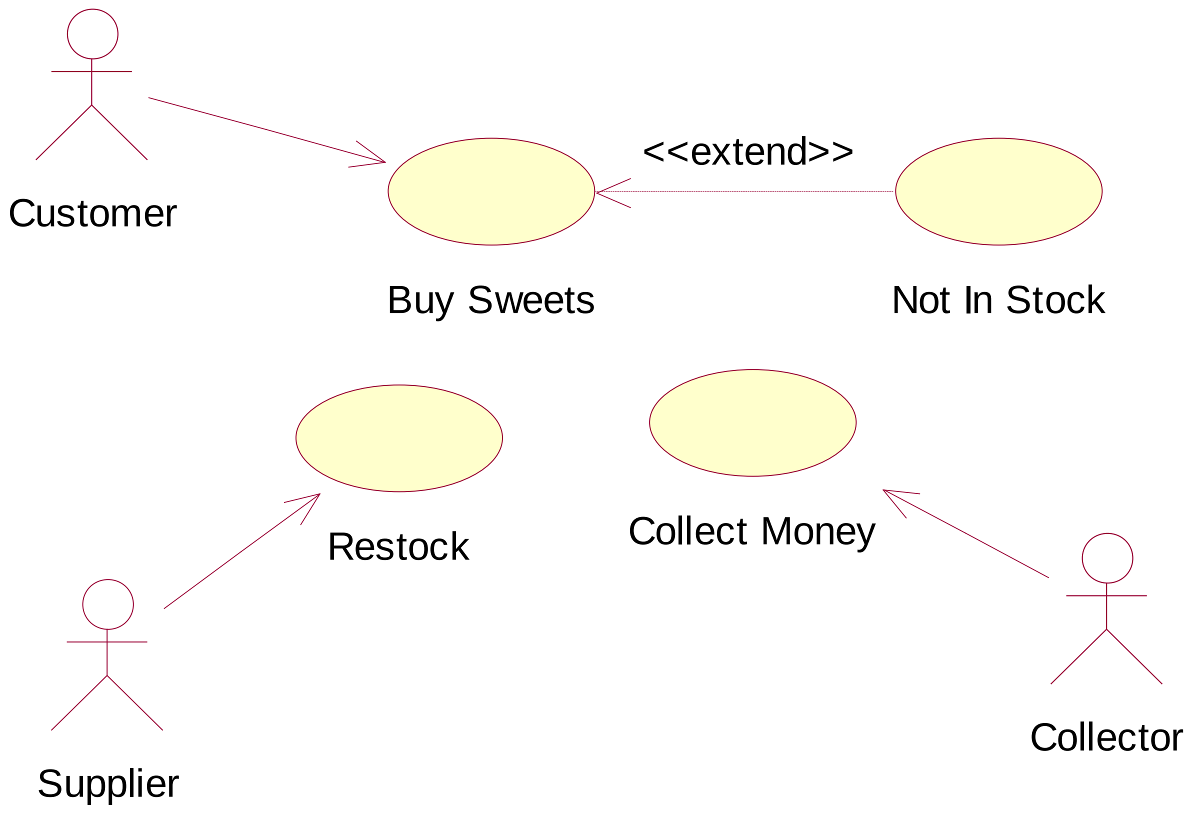
7SENG003W Advanced Software Design

Interaction modelling and Use Case Realization

Interaction modelling

- Use cases are a way of looking at the functional requirements of the system
  - What tasks does the user want to carry out?
  - The externally-visible behaviour of the system
- The set of all use cases describes the complete functionality of the system – from the user's perspective
- But we want to design the system using objects and classes
- Need to turn use case models into objects/classes => sequence diagrams

Use Case Model example : A Self-Service Sweets Machine



A use case scenario : Buy Candy

Use Case Name: Buy Sweets

Principal Actor: Customer

Trigger: Customer wants to buy sweets and inserts money

Main Flow

1. Customer inserts money into coin slot in machine front
2. Money travels from coin slot to cash register
3. Customer selects product code from keypad in machine front
4. Machine front sends product code to cash register
5. Cash register checks sufficient money entered
6. Cash register checks dispenser has product
7. Cash register instructs dispenser to dispense product

Exception

5a. Insufficient Funds

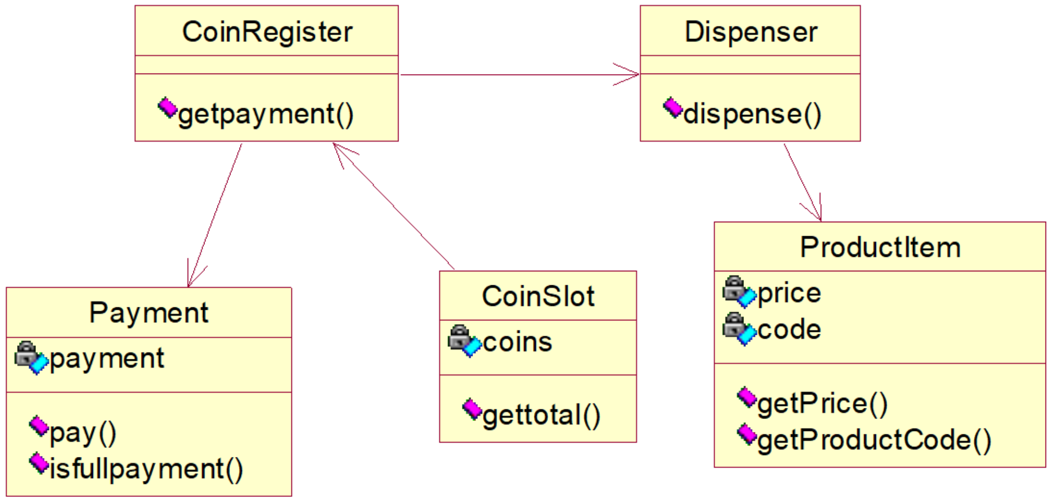
- Cash register instructs machine front to display insufficient funds message
- Customer inserts money into coin slot
- … etc.

6a. Product Not In Stock

1. See Not In Stock Use Case

Where we want to get to…

- The analysis and design of a system, therefore, begin with
  1. A use case model (use case diagram + scenarios)
  2. A domain model (a class diagram showing the relationship between the major entities in the problem domain)
- However, the eventual system design should look something like this example:



- Question: how to we get to a design like this from use cases and scenarios?

How to start modelling from use cases

- Use cases can be used to drive design and development of a system
- In order to achieve this, we need to start designing on the basis of the information in the use cases themselves
  - This process is called *Use case realization*
- A use case realization is a description of a set of objects and how they interact with each other to support/implement the functionality described by the use case
  - Bring together the functional behaviour described in the use case (dynamic model) and the classes described in the domain model (static model)
- Jacobsen:
  - *"It is only when you have interaction diagrams for all courses of events in all use cases that you can be certain that you have found all of the roles that the system requires each object to play and, thus, the responsibilities of each object".*
- How do we produce a realization?
  - By looking at the details in the scenarios, with the help of the domain model

Goals of use case realization

- According to Rosenberg ("Use-Case Driven Object Modelling with UML"), there are three goals for use case realization
  1. Allocate behaviour among boundary, entity and control objects
     - Robustness analysis involved identifying control objects etc. – now need to refine that model and allocate specific bits of behaviour to specific objects
  2. Show the detailed interactions that occur over time among objects associated with each of your use cases
     - I.e, show which messages are invoked on/sent to each object over time that participates in a particular task
  3. Finalize the distribution of responsibility among classes
     - We work on the static model using knowledge from our dynamic model

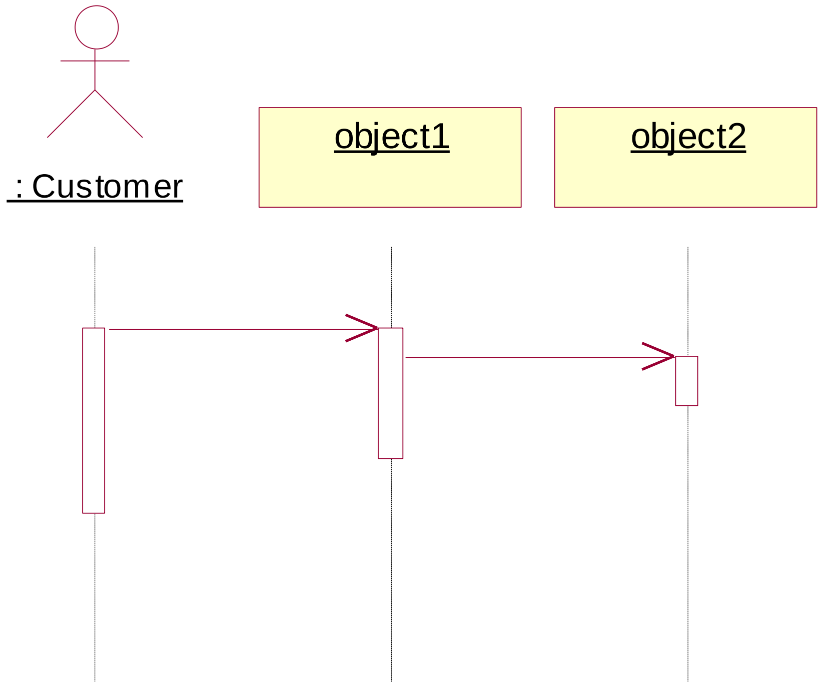
Interaction diagrams

- Use case realizations in UML use *interaction diagrams* to show how objects interact with each other. There are two types of interaction diagram:
  - **Collaboration diagrams** : basically object diagrams with messages shown over the links between the objects
  - **Sequence diagrams** : a diagram that makes explicit the temporal sequence of messages passed between objects

- Sequence diagrams more commonly used for use case realizations because
  - they model the message sequencing in a clear, visual way
  - Their linear structure matches linear narrative of scenarios

Realization

- Realizing a use case is the process of deciding what objects are needed and how they should interact with each other
  - The objects and their interactions should implement the use case
  - It is the first step in moving away from simply describing what a system should do to how it should do it
- The interaction between objects that realize a use case is often shown in a sequence diagram
- The sequence diagram shows the user as an actor, objects and their lifelines (the period of time of their existence), and the messages that pass between them



The Buy Candy use case scenario again

Use Case Name: Buy Sweets

Principal Actor: Customer

Trigger: Customer wants to buy sweets and inserts money

Main Flow

1. Customer **inserts money** into coin slot in **machine front**
2. Money **travels** from coin slot **to cash register**
3. Customer **selects product code** from keypad in machine front
4. Machine front **sends product code** to cash register
5. Cash register **checks sufficient money** entered
6. Cash register checks **dispenser has product**
7. Cash register instructs dispenser to **dispense product**

Exception

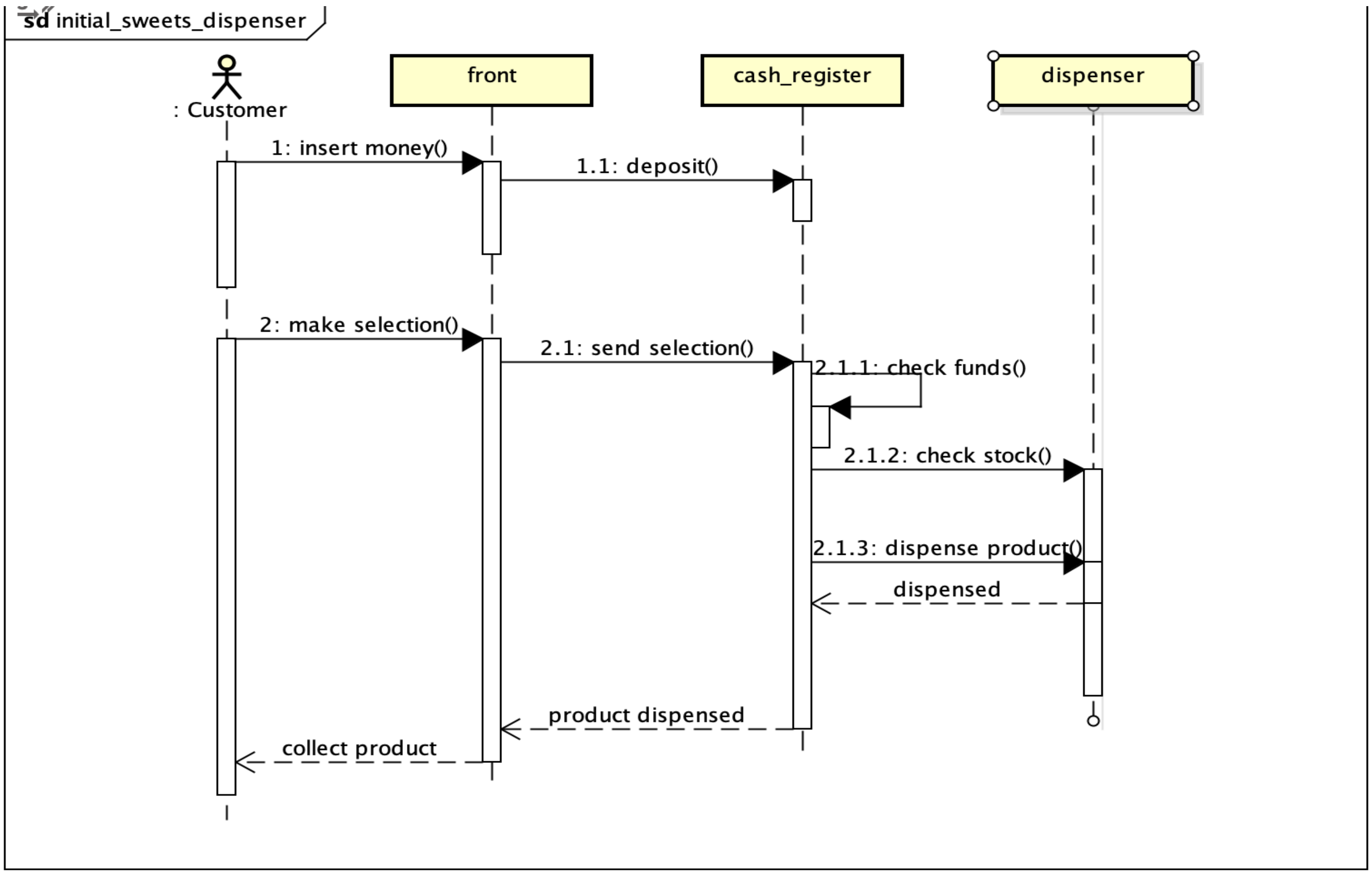
5a. Insufficient Funds

- Cash register instructs machine front to display insufficient funds message
- Customer inserts money into coin slot
- … etc.

6a. Product Not In Stock

1. See Not In Stock Use Case

Sequence diagram for sweets dispenser



Scenario -> Sequence Diagram

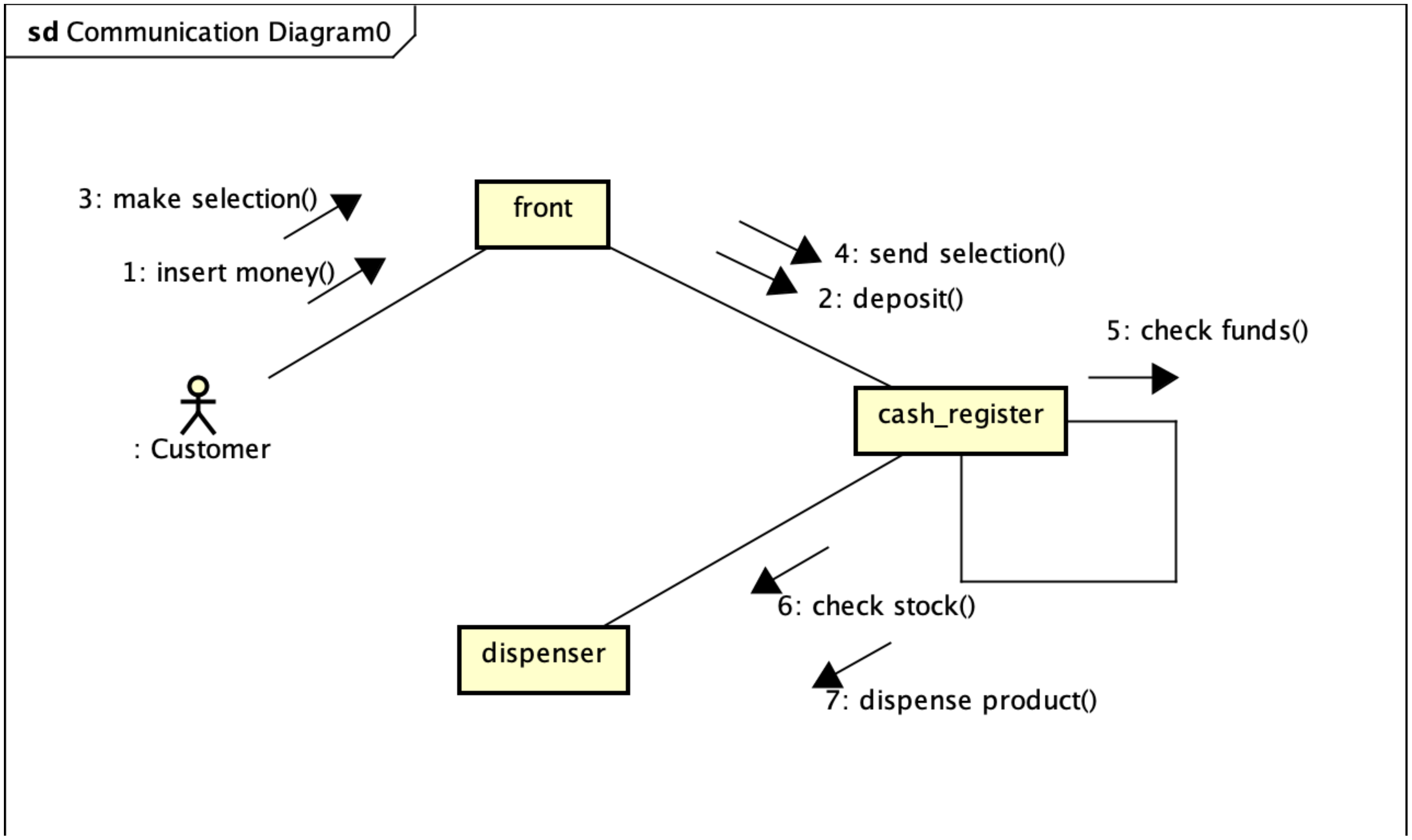
- How does the sequence diagram realize the use case?
  - It takes one scenario of the use case – e.g., the main flow
  - It turns the detail of the scenario into a set of interactions between the actors participating in the scenario and a set of objects
  - The sequence diagram shows these interactions over a period of time (from top to bottom in the diagram)

- How do we know who the actors will be?
  - They are described in the use case diagram and listed in the scenario
- How do we know what the objects will be?
  - We look at the domain model (for guidance) and read the scenario (hint: it often helps to write the scenario with the domain model to hand)
- How do we know what the messages between the actor(s)/objects will be?
  - The messages represent the things that happen in the scenario

Collaboration diagrams

- The collaboration diagram is a similar diagram to the sequence diagram – it shows equivalent information, but from a different perspective
- A sequence diagram simply shows messages being sent between objects over time
  - It doesn't show the links between objects – you assume they exist if objects are able to send messages to each other
- A collaboration diagram makes explicit the links between objects
  - The links can be decorated with information about the messages that pass between objects, but this information is not the primary purpose of the diagram
  - To show the sequence in which messages are sent, they are numbered
  - Moreover, the object links are not shown over time, but from a static perspective
    - So if an object is created during an interaction, you must annotate to show this happening (although you are unable to say exactly when during the interaction…)

Collaboration diagram example



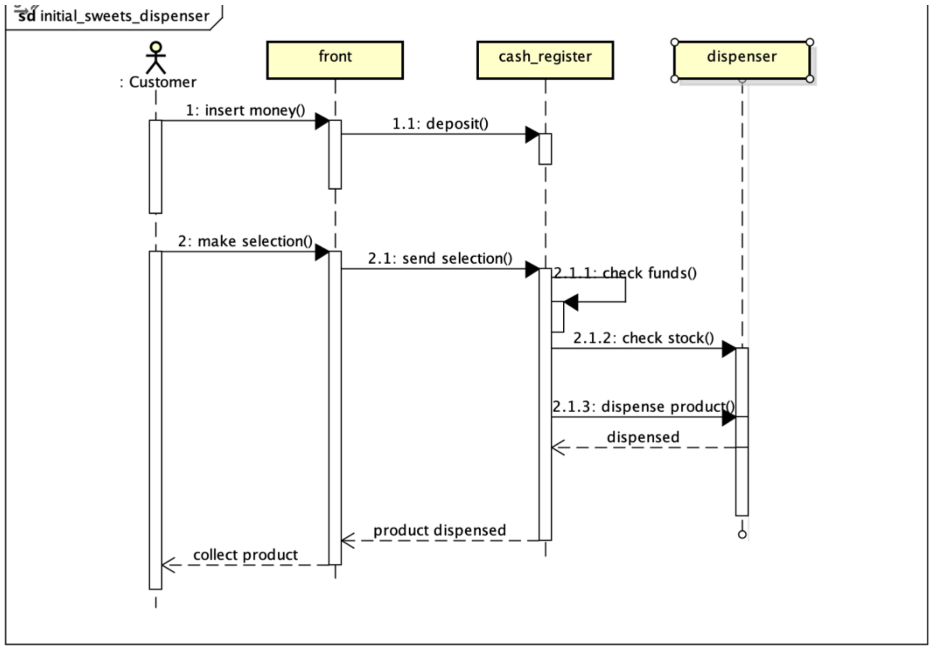
What information is included here that is absent from the sequence diagram?

Static models

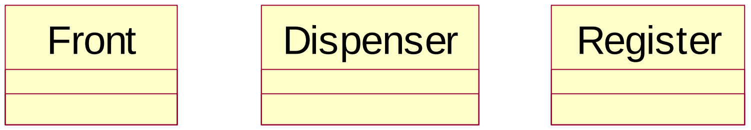
- The sequence diagram and the collaboration diagram are very useful in working out how objects interact with each other to implement the functionality required by a use case
- The interaction diagrams that realize a use case, therefore, provide part of the dynamic model of the system
- Once we have realized all the use cases we have a complete dynamic model of the system
- But we also need a static model – to model the classes in the system
  - Model their structure (attributes)
  - Model their interface (operations…)
  - Model their relationships (associations, generalization etc)
- This information can be gleaned from the dynamic model by careful reading …

Class Diagrams of realizations

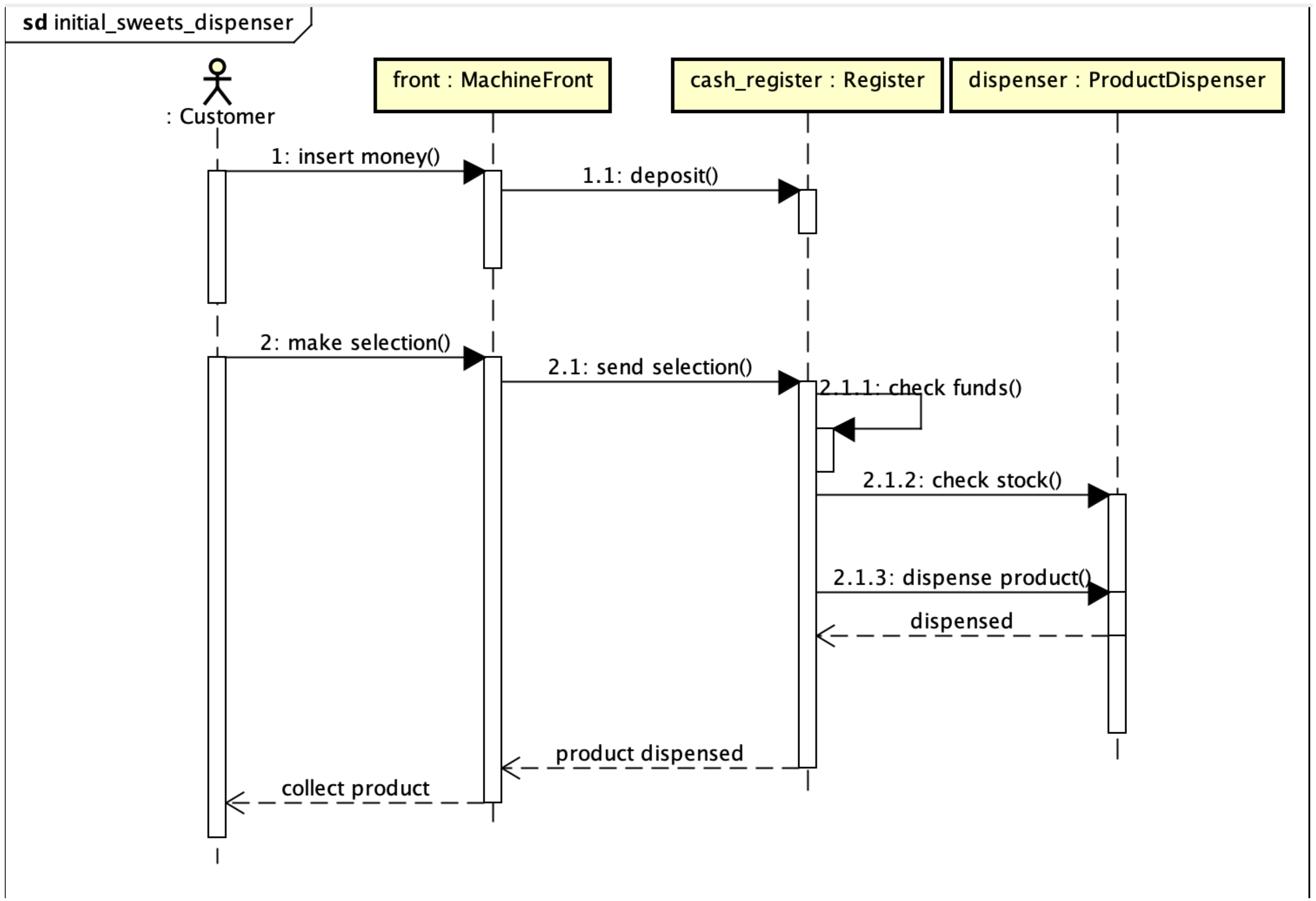
- Take a use case realization:
- What are the classes of the objects



  - Take classes in the domain model as starting point



Adding classes to realization



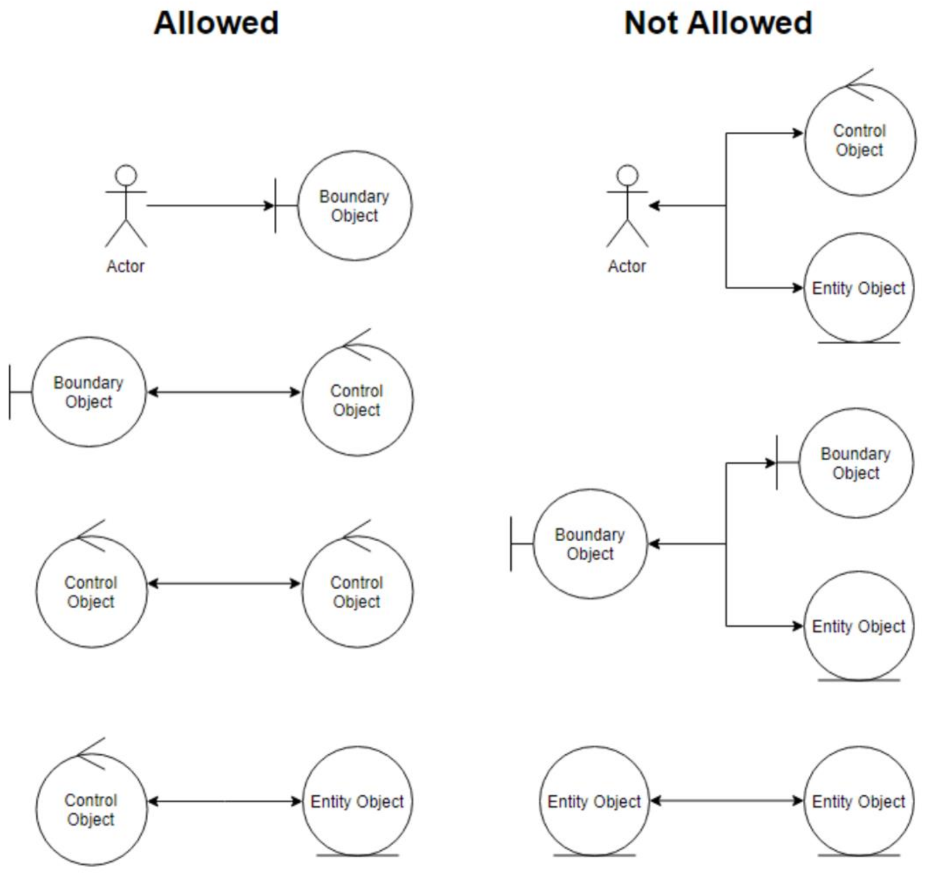
Boundary/Entity/Control

- We can classify objects by their type of responsibility
  - Each object should have only one type of responsibility

- **Boundary**: objects used as interfaces by actors to the rest of the system
- **Entity**: objects that are usually objects from the domain model – encapsulate some data identified from our analysis of the problem domain
- **Control**: objects which serve as the “glue” between boundary and entity objects, and which end up controlling processes and delegating work

- This kind of classification is very similar to MVC – Model-View-Controller

Boundary/Entity/Control rules

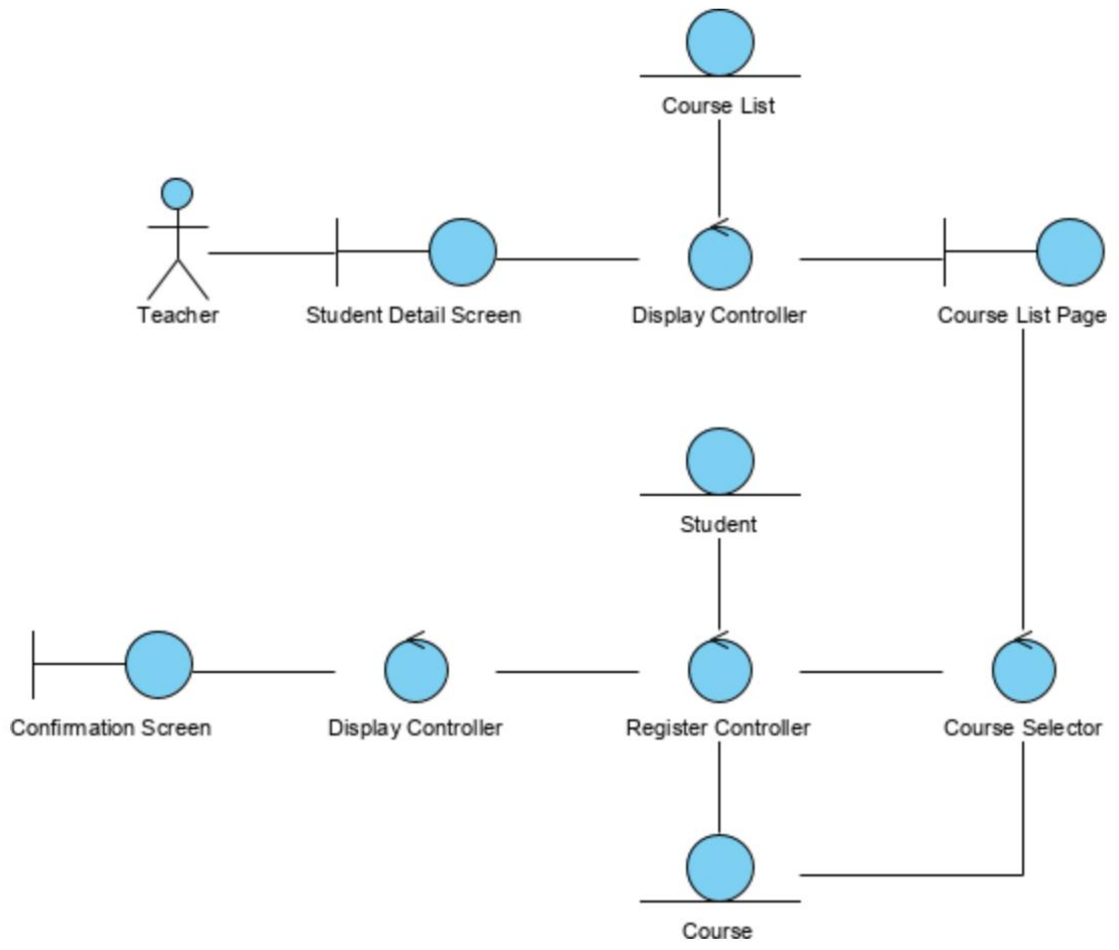


Example

- 1. The teacher views a student's details
- 2. The teacher selects the option to add the student to a course
- 3. The Course manager provides a list of courses to be displayed to the teacher.
- 4. The teacher selects the name of a course
- 5. The teacher informs the registrar to register the student on the course
- 6. The registrar registers the student on the course
- 7. The registrar confirms the registration to the teacher

(taken and adapted from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/)

Example (contd.)

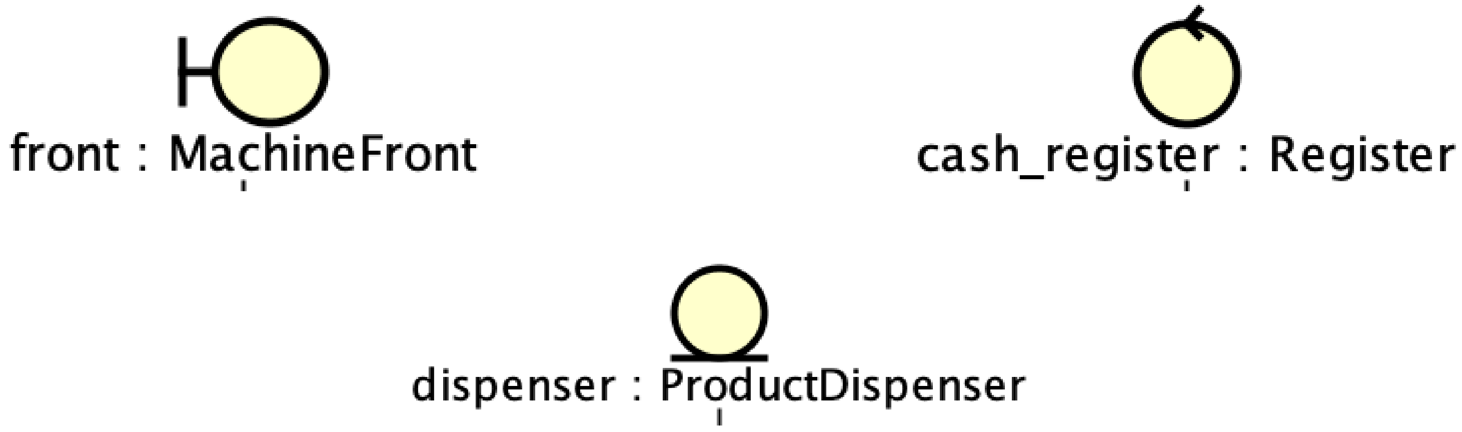


"Robustness diagram", typically a class diagram using stereotypes and Icon display

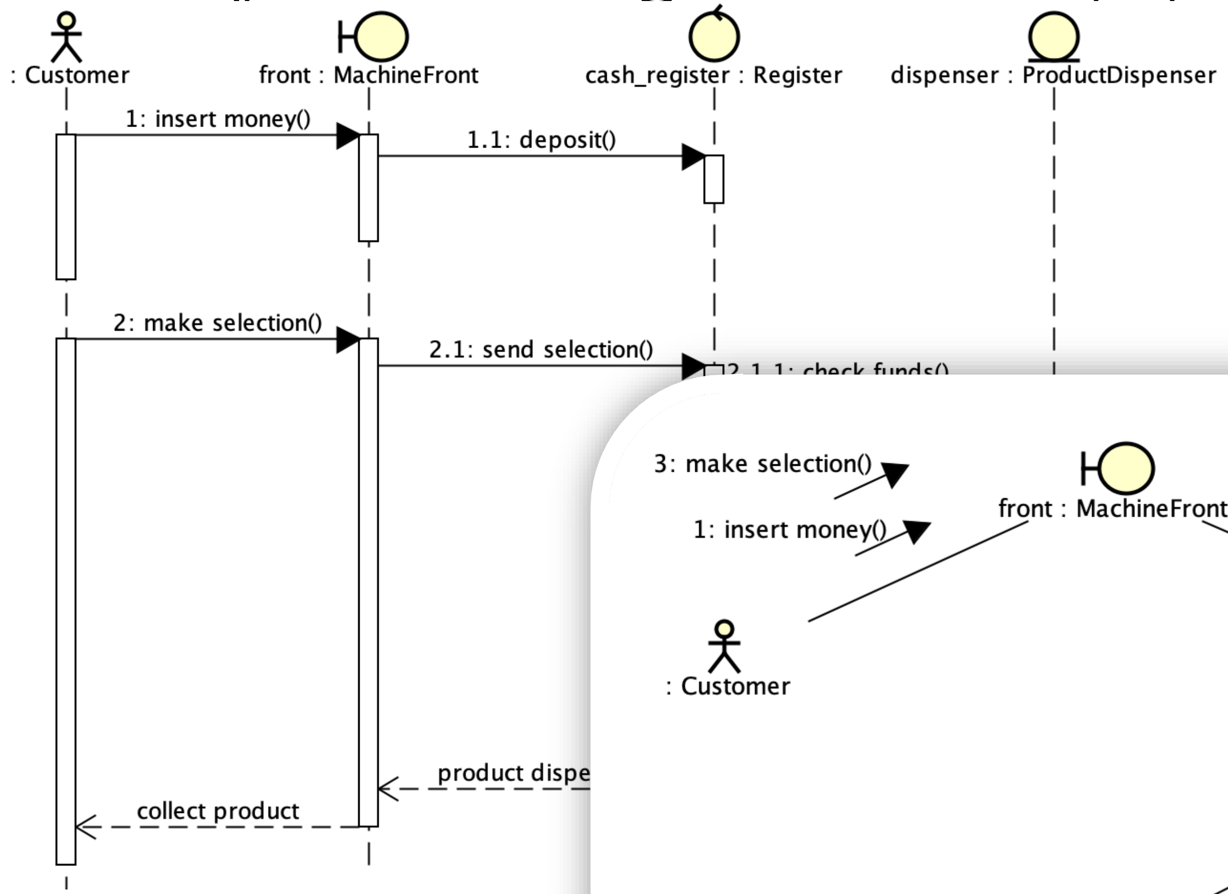
From https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/

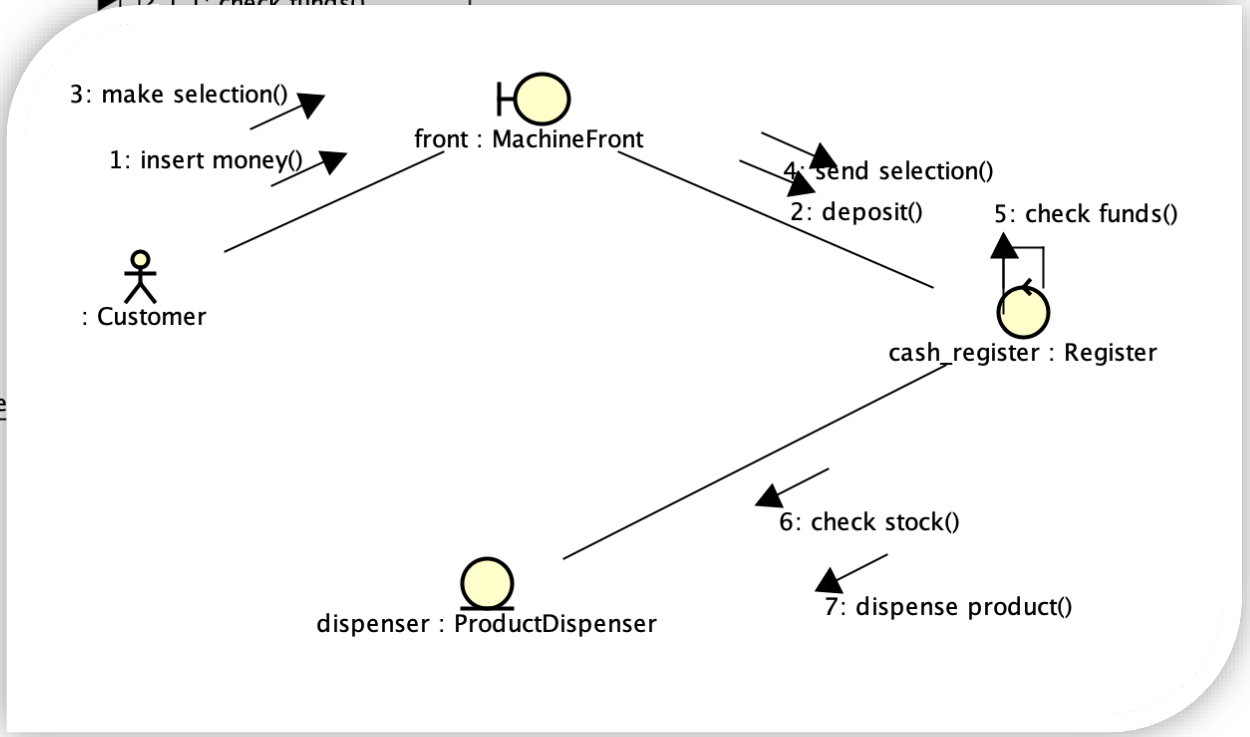
Adding Boundary, Entity, and Control

- Which objects are boundaries, entities, or control objects?
- front: MachineFront – actor interfaces with this object = Boundary
- Dispenser : ProductDispenser – deals with product(data), so entity
  - Can only talk to control objects
- cash_register : Register – makes decisions (has enough money, tells dispenser to dispense product), so control



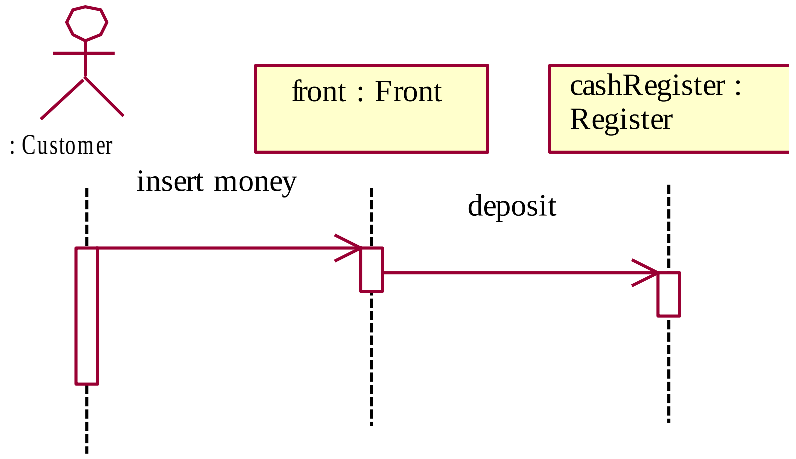
Seq/Comm diagrams with B/E/C





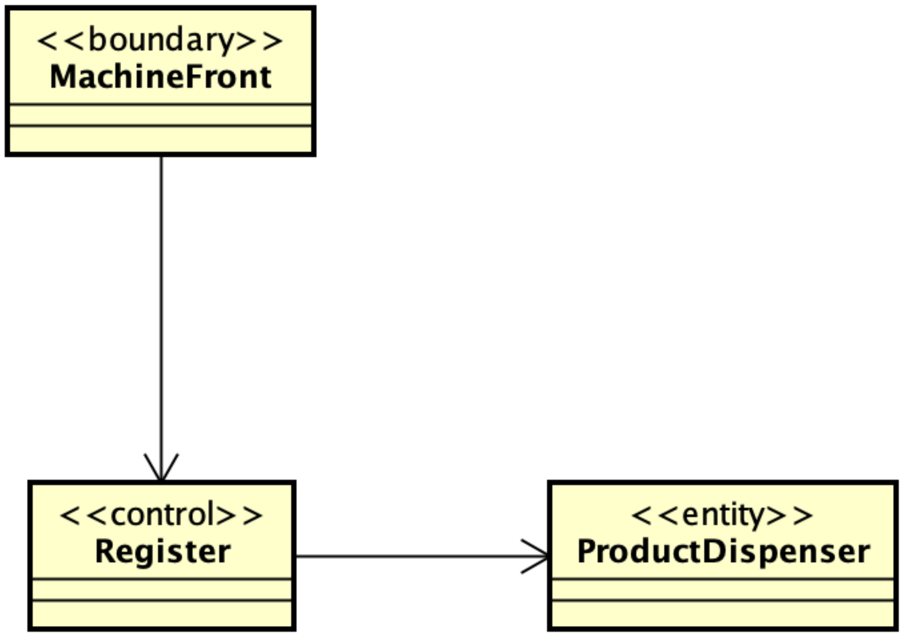
Creating a class diagram

- Given the sequence diagram with classes assigned to the objects, how do we create a class diagram showing how these classes are related?
- If one object is able to **send a message** to another object, then it must **have a link** to that second object



  - In this case, object `front` must have a link to object `cashRegister` in order to be able to send the `deposit` message to it

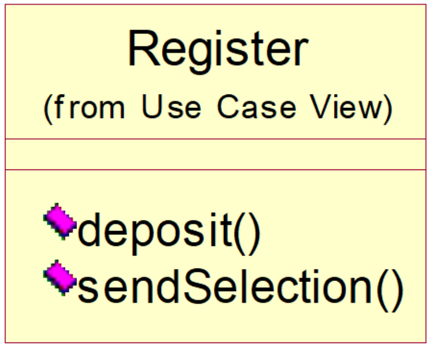
Example class diagram



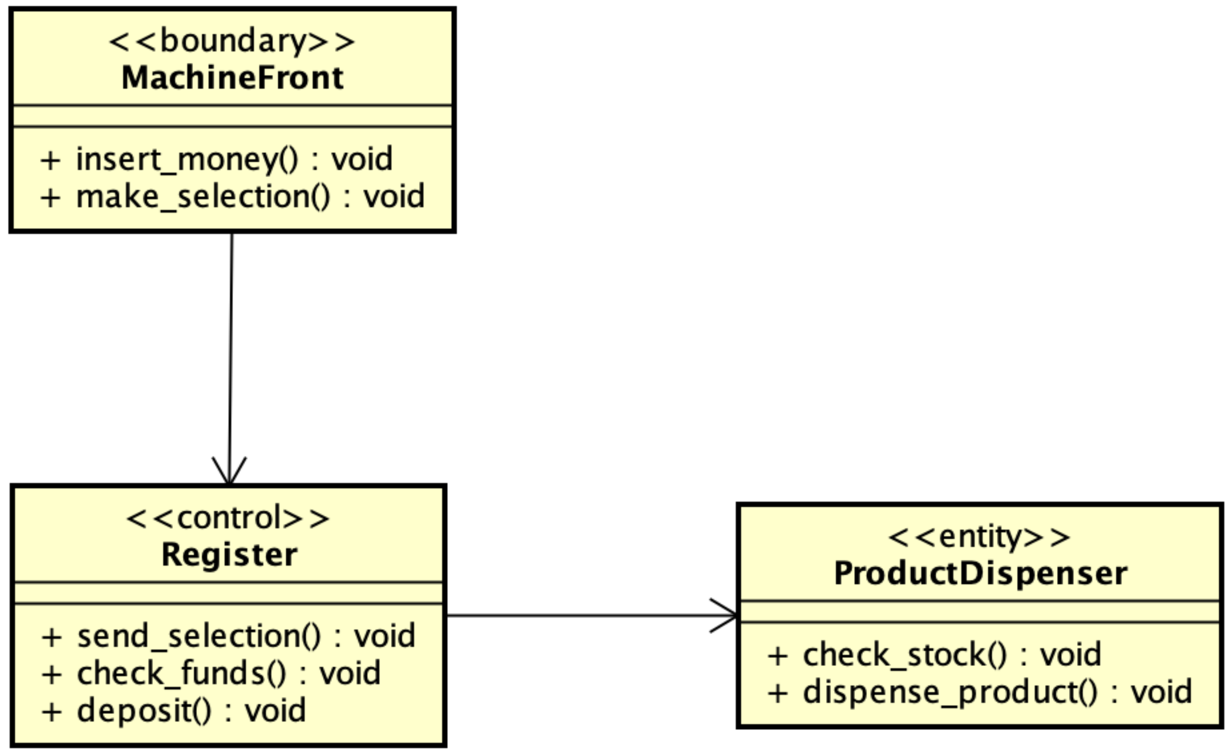
Question: why are the relationships between the classes shown as uni-directional associations?

Adding operations to classes

- If an object is able to send a message to another object, then we assume that the second object is able to understand the message (in languages such as C++ and Java)
- To find out what operations to provide in the class, look at what messages objects of that class are receiving in the interaction diagrams
- For example, the Register class
  - The register object in the Buy Candy realization receives the following messages:
    - Deposit
    - Send selection
  - Hence the class could be:



Class diagram



Recipe

- Write scenarios – make sure you use ”interesting” and informative nouns and verbs
  - Nouns = candidates for objects, Verbs = candidates for messages
- Translate scenarios into sequence diagrams
  - Preserve the narrative
- Turn sequence diagrams into communication diagrams
  - If two objects communicate, they must be linked
  - Communication diagrams help you see the links between objects
- Create classes for your objects
  - If two objects are linked, then their classes must have an association
  - Make this a plain association at first, then decide if aggregration or composition
  - Note: no generalization/inheritance at this point – decide this later
- If one object sends a message to a 2nd object, that message must be in the interface of the 2nd object’s class
  - Build up your class interfaces, message by message
- Everything driven off the scenarios agreed with client – so model ends up matching what client signed off on

The Analysis Model

- We need to realize each scenario in order to fully understand
  - What objects need to exist in the system at run-time
  - When they are created and destroyed
  - How they interact, I.e, what messages they send to each other
  - What are their classes
  - How are those classes structured, what operations do they provide, how are they related

- Once we have all this information, we have a full analysis model
  - a high-level view of the system
  - Essentially a first-draft design

Further reading

- Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example
  - Chapters 5 (Robustness Analysis) and 7 (Sequence diagrams)
  - Available online via Safari
- Software Requirements Using the Unified Process: A Practical Approach
  - Available from library online (see Chapter 13)
- Robustness analysis
  - http://www.popkin.com/customers/customer_service_center/tip_of_the_week/03_24_03/tipofweek.htm
  - http://pyre.third-bit.com/helium/extern/rosenberg04.pdf