10 Analyzing the Problem

Introduction to OOA OOD and UML

2022 Spring

College of Information Science and Engineering

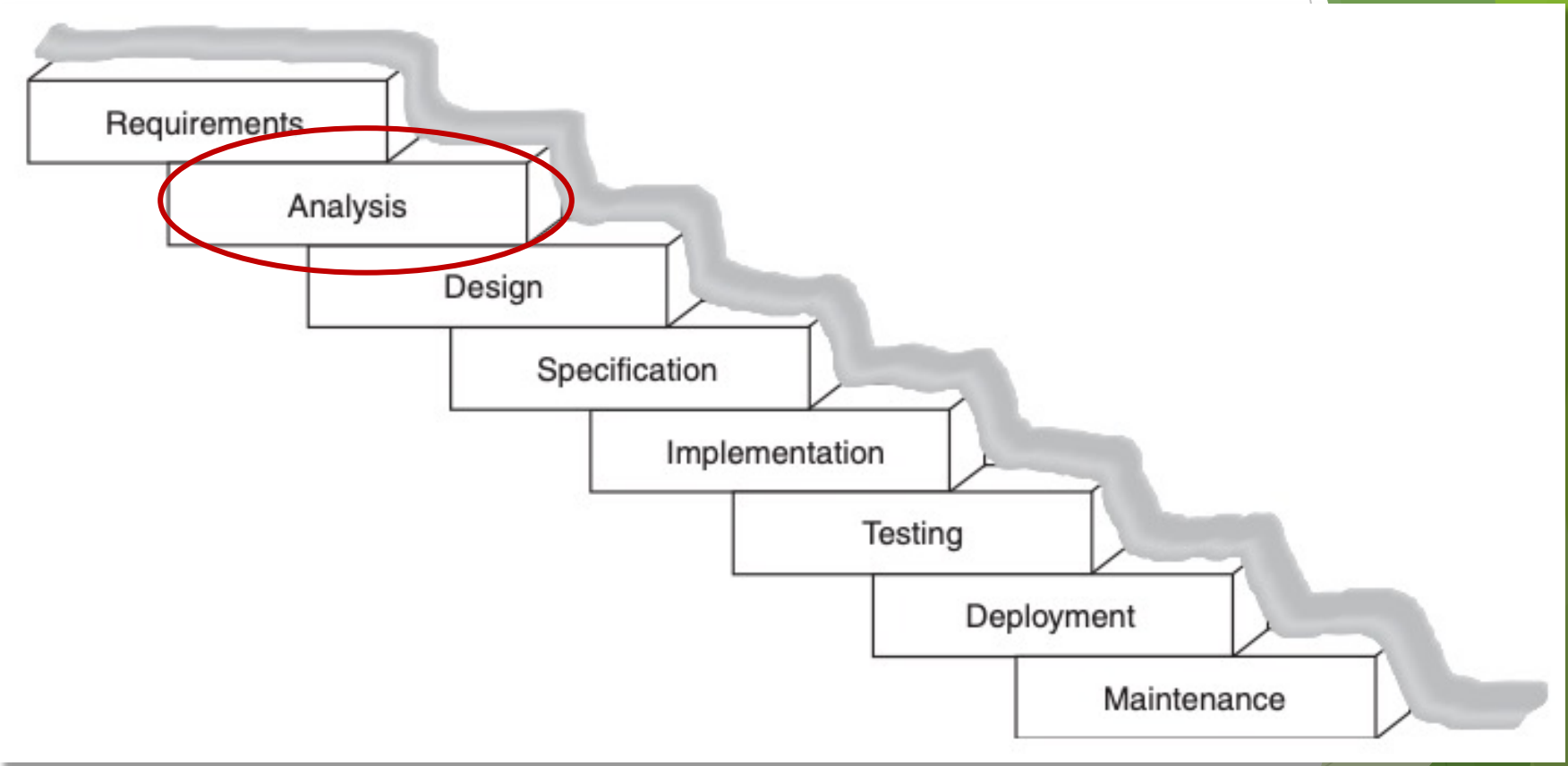
Ritsumeikan University

Yu YAN

Outline

- Introduction
- Overview of the Analysis Process
- Static Analysis
- Dynamic Analysis
- Exercise 10

Where Are We Now?



Purposes of Problem Analysis

- Analysis is about discovering what the system is going to handle, rather than deciding how to do the handling
- The purpose of Analysis is to decompose a complex set of requirements into the essential elements and relationships on which we will base our solution
- Analysis is modeling the real-world objects as system engineering objects

Static and Dynamic Analysis

- An analysis model has both static and dynamic parts:
  - A static analysis model can be depicted by using an analysis class diagram.
  - An analysis class diagram shows the objects that the system will handle and how those objects are related to each other
  - A dynamic analysis model can be depicted by using a communication diagrams.
  - A communication diagrams demonstrates that the static model is feasible

Inputs to Analysis

The business requirements model

The system requirements model

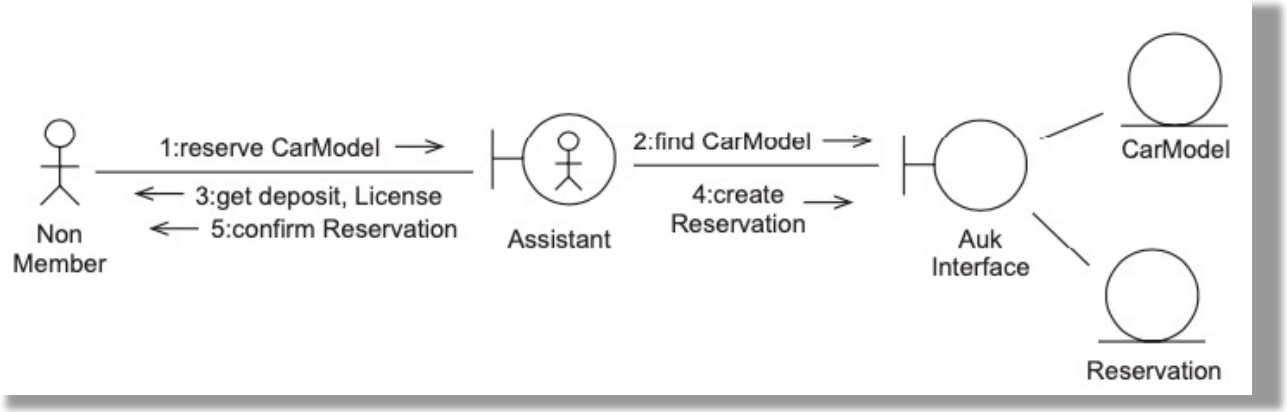
- Those inputs must be transformed into a model of objects that will be processed by the proposed system, along with their attributes and relationships.

Outline

- Introduction
- Overview of the Analysis Process
- Static Analysis
- Dynamic Analysis
- Exercise 10

- Analysis has the following steps which you repeat until you and your sponsors are happy:

1. Use the system requirements model to find **candidate classes** that describe the objects that might be relevant to the system and record them on a class diagram
2. Find **relationships** (association , aggregation, composition and inheritance) between the classes
3. Find **attributes** (simple, named properties of the objects) for the classes
4. Walk through the system use cases, checking that they're supported by the objects that we have, fine-tuning the classes, attributes and relationships as we go – this **use case realization** will produce operations to complement the attributes
5. Update the glossary and the nonfunctional requirements as necessary – the use cases themselves should not need updating, although perhaps they will need some correcting



- **iCoot** system use case list will be:
- (U1) Browse Index: A Customer browses the index of CarModels
- (U2) View Results: A Customer is shown the subset of CarModels that was retrieved
- (U3) View CarModel Details: A Customer is shown the details of a retrieved CarModel, such as description and advert
- (U4) Search: A Customer searches for CarModels by specifying Categories, Makes and engine sizes

Outline

- Introduction
- Overview of the Analysis Process
- Static Analysis
- Dynamic Analysis
- Exercise 10

Finding Classes

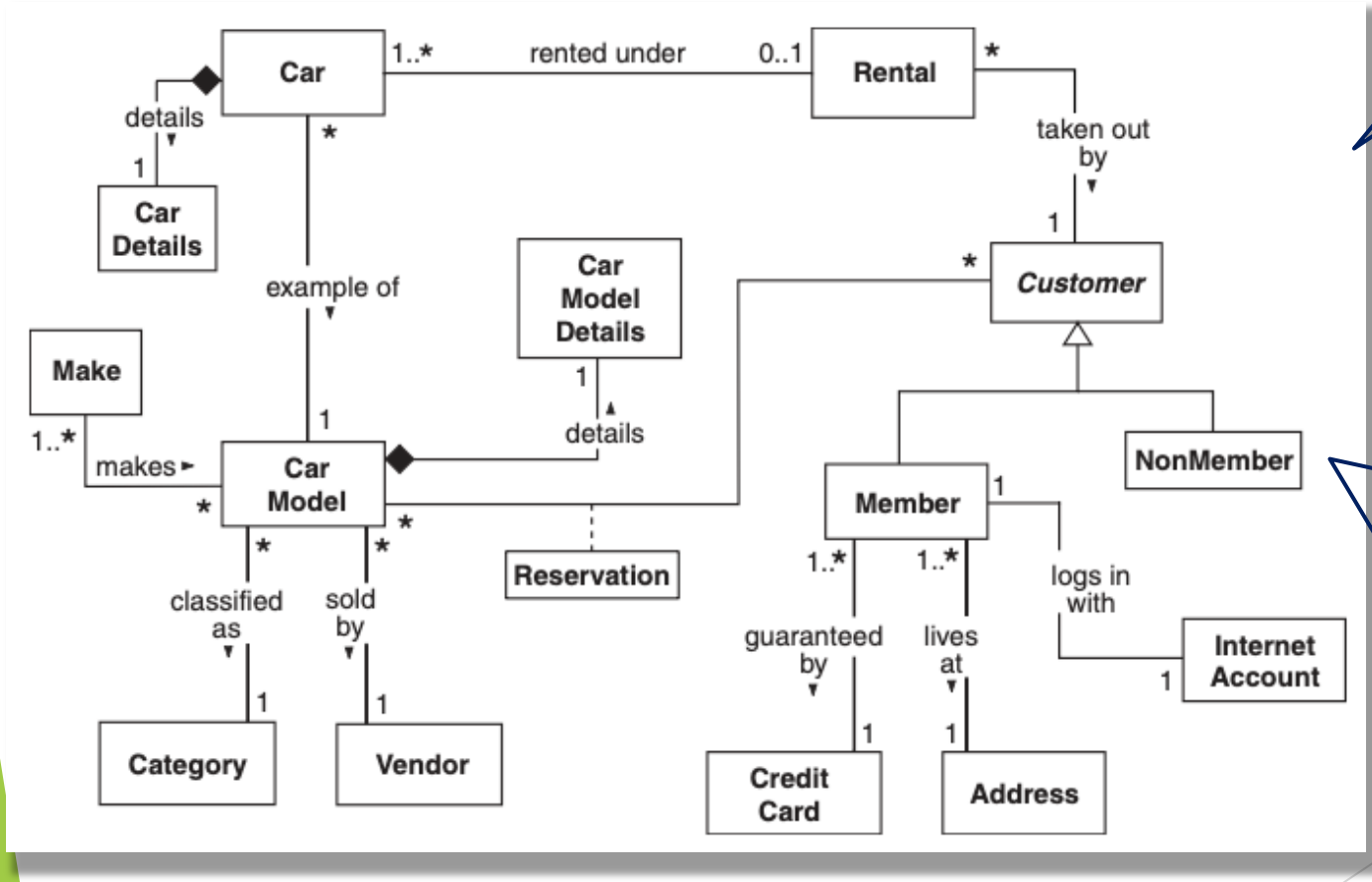
- From business/system requirements modeling, we have a good source of candidate classes from use cases
- Candidate classes are often indicated by **nouns** in the use cases, For example:
  - The system itself, such as, 'system' or 'iCoot'
  - Actors, such as, 'Assistant' or 'Head Office'
  - Boundaries, such as, 'customer applet' or 'head office link'
  - Trivial types, such as, 'strings' and 'numbers'
- Short descriptions for the candidate classes that are left after this filter process should be added to the glossary

Identifying Class Relationships

- Four possible types of relationship that could be used to draw between candidate classes:
  - Association: Objects of one class are associated with objects of another class
  - Aggregation: Strong association – an instance of one class is made up of instances of another class
  - Composition: Strong aggregation – the composed object can't be shared by other objects and dies with its composer
  - Inheritance: A subclass inherits all of the attributes and behavior of its superclass(es)

Drawing Class Diagrams

- A class diagram shows us what classes exist and how they are related



An analysis class diagram for iCoot

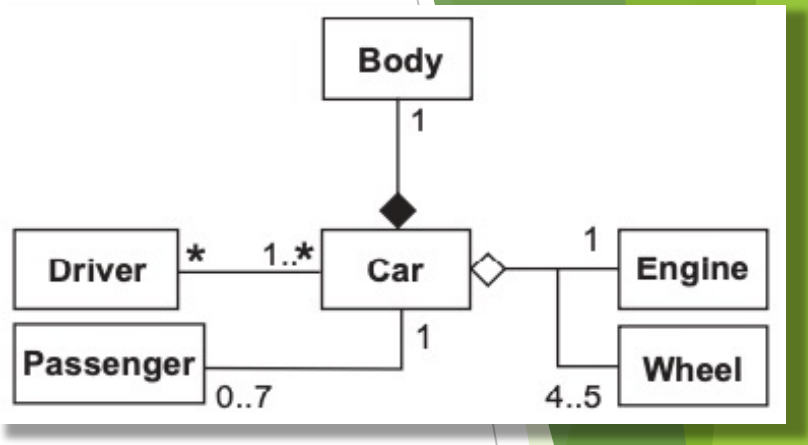
Every class is represented as a box with the class name inside(in **bold**). If the class is abstract, the class name is *italicized*

Multiplicity (1)

- Multiplicity is defined only for association, aggregation, composition
  - How many objects are involved in a relationship (the multiplicity of relationship)?
    - ✓ If it is composition, the answer is always one
    - ✓ If it is the other case (association or aggregation) and no multiplicity is shown, the answer is not simply known
    - ✓ The followings are some instructions:
      - n: Exactly n (exactly n objects are involved)
      - m..n: Any number in the range m to n (inclusive)
      - p..*: Any number in the range p to infinity
      - *: Shorthand for 0..*
      - 0..1: Optional

Multiplicity (2)

- An example of the multiplicity of the relationship:
  - A car has one Engine
  - An Engine is part of one Car
  - A Car has four or five Wheels
  - Each Wheel is part of one Car
  - A Car is always composed of on Body
  - A Body is always part of one Car and it dies with that Car
  - A Car can have any number of Drivers
  - A Driver can drive at least one Car
  - A Car has up to seven Passengers at a time
  - A Passenger is only in one Car at a time

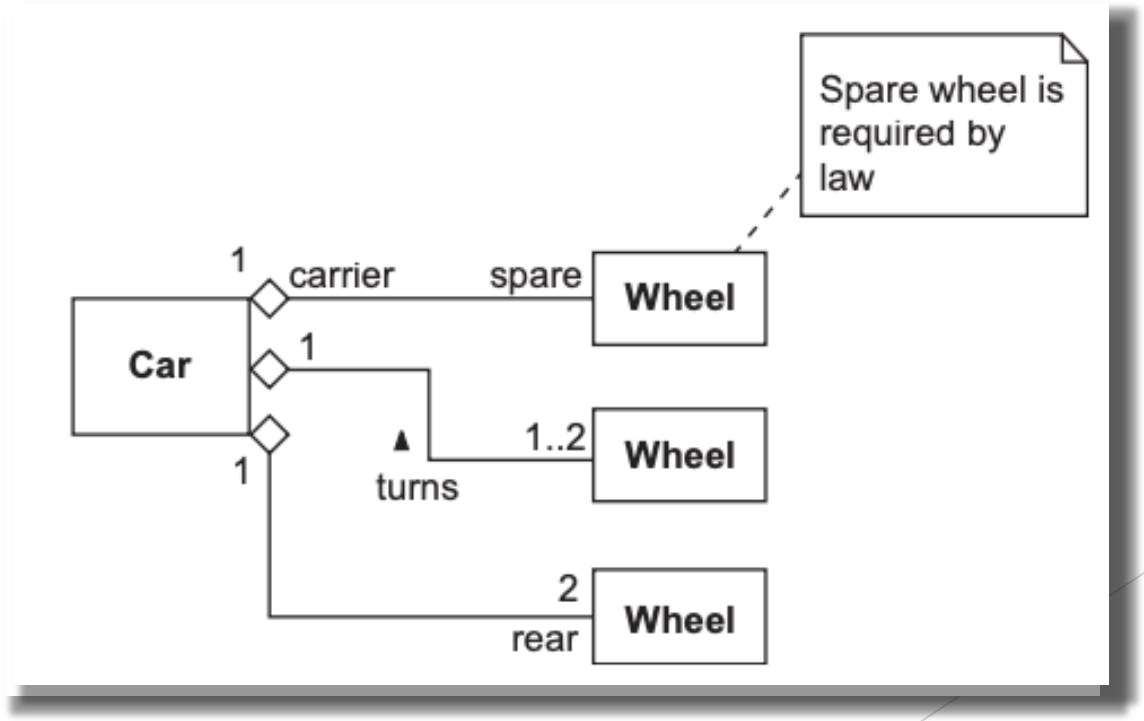


Association Labels, Roles and Comments (1)

- An association label can be given to association, composition and aggregation, for indicating the nature of the association
  - If it's not obvious which way the association name should be read, a black arrowhead can be used
- As well as association names, we can show roles for indicating the part played by an object in the association
  - The role is shown as a label near the object that plays the role
- In addition, we can also show a comment in a relationship
  - An arbitrary piece of text enclosed in an icon that looks like a piece of paper, connected to the relevant part of the diagram by means of a dashed line

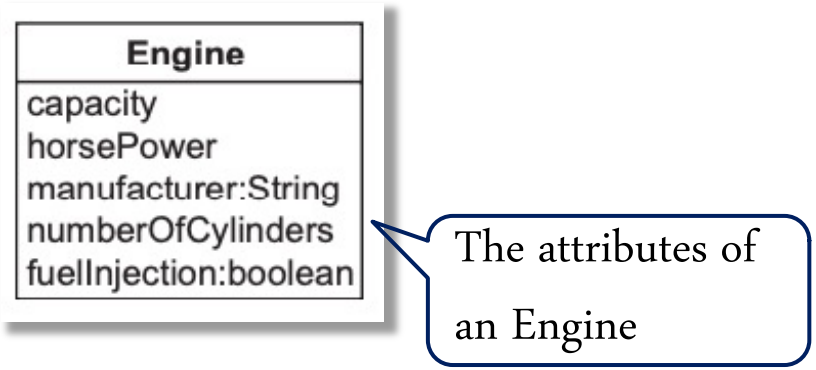
Association Labels, Roles and Comments (2)

- An example of the association labels, Roles and comments of the relationship
  - A car has one Wheel acting as a spare
  - The spare Wheel has one Car acting as its carrier
  - A Car has two rear wheels



Attributes (1)

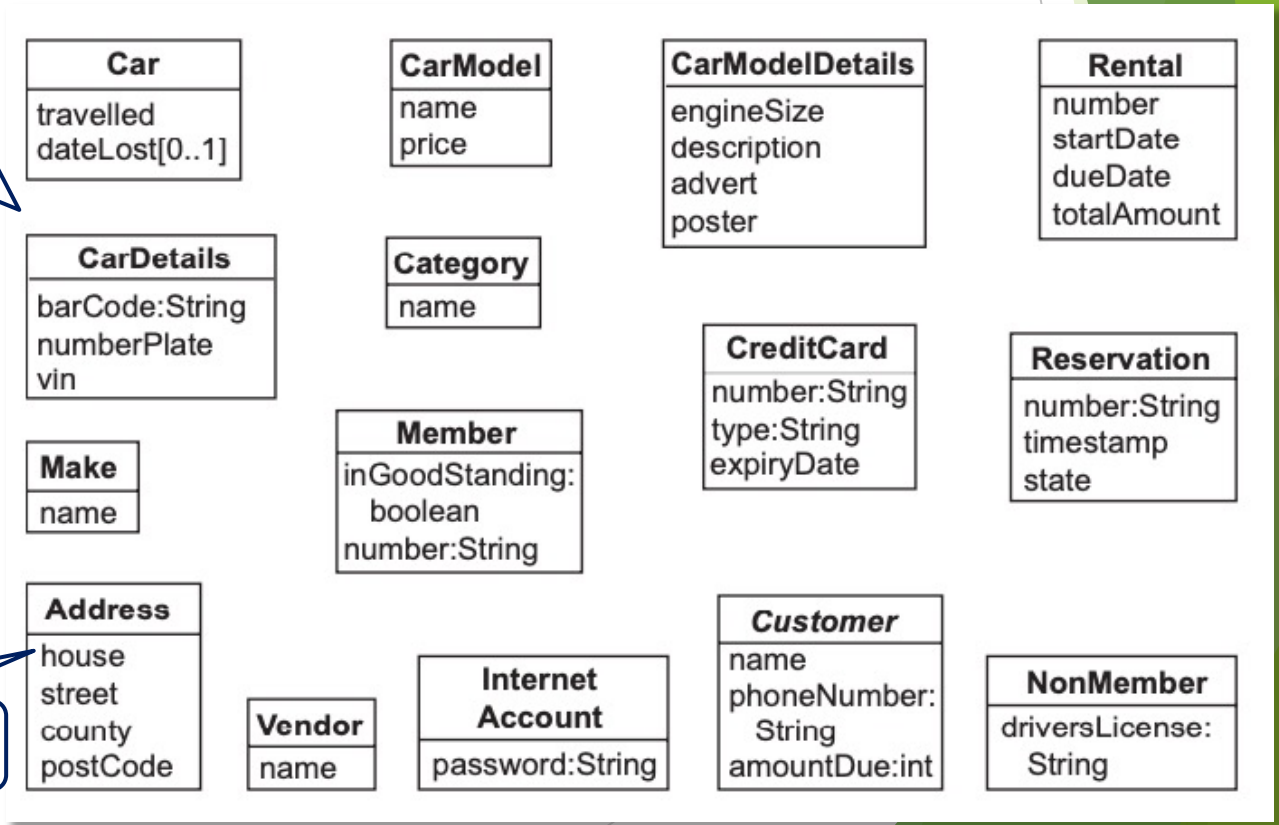
- Review:
  - An attribute is a property of an object, such as its name
  - Each attribute can be given a type, which is either a class or a primitive.
  - The attribute types can be omitted during analysis



Attributes (2)

- Attributes can be shown on a class diagram by adding a compartment under the class name
- To save space, we document attributes separately instead as an **attribute list**, complete with descriptions.

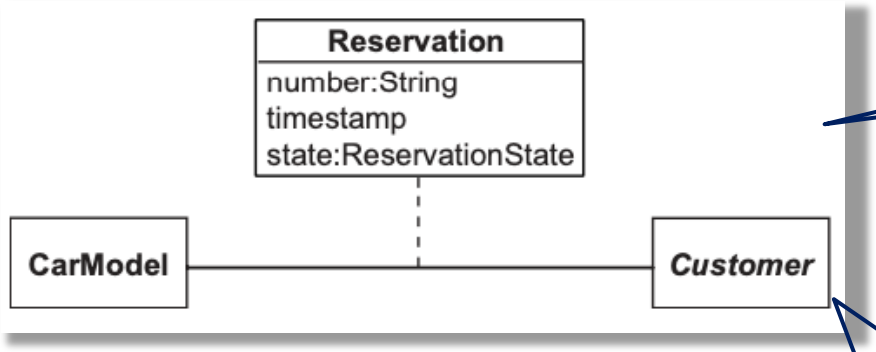
If the Car is lost, we record the date it was lost, otherwise we record nothing.



An attribute list for iCoot

Association Classes

- An association class can be introduced alongside an association.
- An association class represents attributes and operations that exist only because the association exists.



An association class from iCoot

A CarModel can be associated with any number of Customer objects and a Custormer can be associated with any number of CarModel objects. For each link, there is a corresponding Reservation object that has a number, time-stamp and state.

Outline

- Introduction
- Overview of the Analysis Process
- Static Analysis
- Dynamic Analysis
- Exercise 10

Introduction (1)

- We perform dynamic analysis for the following reasons:
  - To confirm that our class diagram is complete and accurate, so that we can fix it sooner rather than later (this may involve adding, deleting or modifying classes, relationships, attributes and operations)
  - To gain confidence that our modeling up to this point can be implemented in software (we're not the only ones that should be confident before we proceed, our sponsors are just as important)
  - To verify the functionality of the user interfaces that will appear in the final system (it's a good idea to partition access to the system into separate interfaces, along use case lines, before we dive into detailed design)

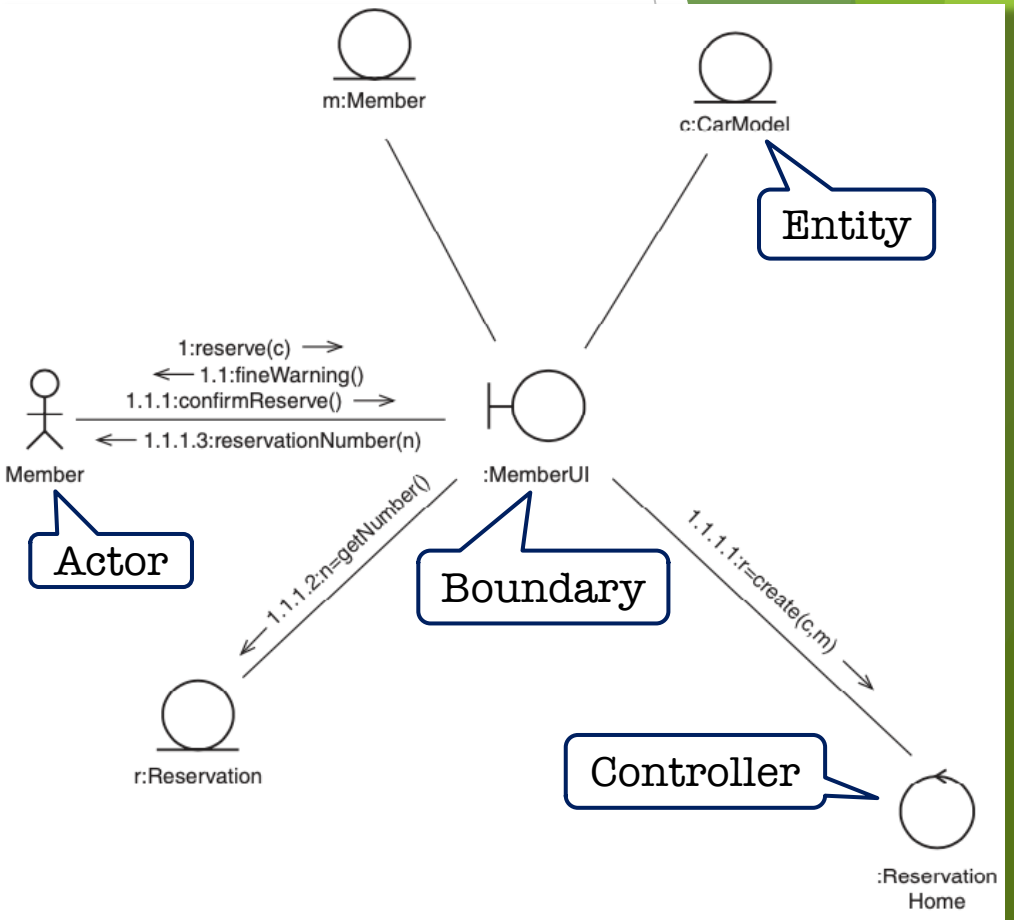
Introduction (2)

- The most important part of dynamic analysis is use case realization. Use case realization has the following steps:
  - Walk through the system use cases, simulating the messages sent between objects and recording the results on communication diagrams
  - Introduce operations on the objects that receive the messages
  - Add classes to represent boundaries (system interfaces) and controllers (placeholders for complex business process or for the creation and retrieval of objects), as necessary
- UML tool: communication diagram, for recording the message sending results between analysis objects

Drawing Use Case Realizations (1)

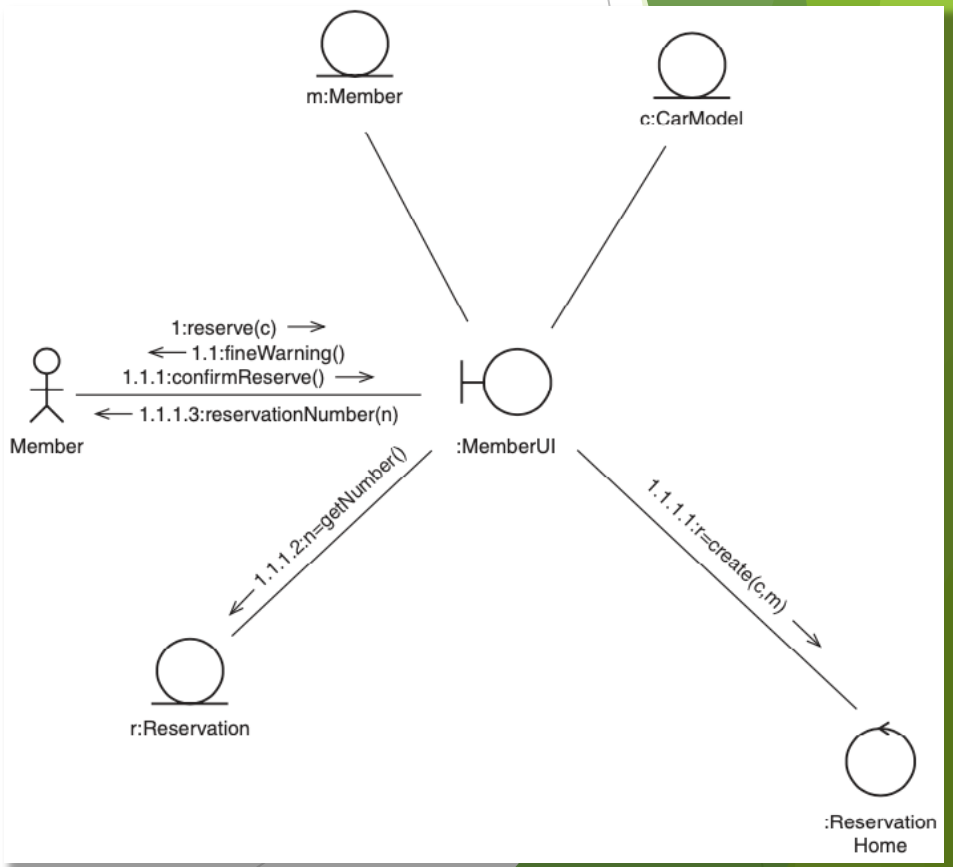
- An informal description:

"A Member actor asks the MemberUI to reserve a particular CarModel; the Member is warned that there is a fine if the corresponding car is not collected when it arrives; once the Member confirms that they do wish to make a reservation, the MemberUI asks the ReservationHome to create a new Reservation, passing in the CarModel and the Member (which the user interface already has, as a result of logging on); finally, the MemberUI gets the number from the new Reservation and passes it to the Member. "

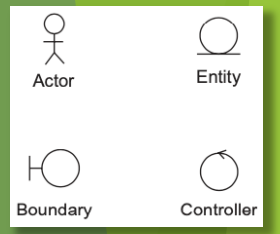


Drawing Use Case Realizations (2)

- Analysis-level communication diagrams can show:
  - Actors interacting with boundaries (for example, the Member interacts with a MemberUI)
  - Boundaries interacting with objects *inside* the system (for example, the MemberUI interacts with a ReservationHome, a Member, a CarModel and a Reservation)
  - Objects inside the system interacting with boundaries to external systems



Boundaries, Controllers and Entities

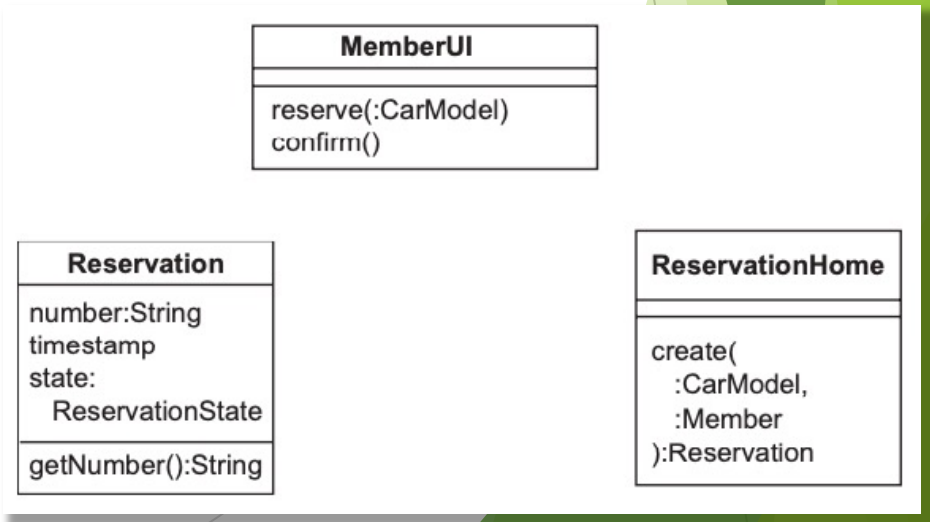


- Elements of a communication diagram
  - Actor: A person (usually) or system (occasionally) existing outside the system
  - Boundary: An object at the edge of the system, between the system and the actors. For system actors, boundaries provide a communication path. For human actors, a boundary means a user interface, capturing commands and queries and displaying feedback and results.
  - Entity: An object inside the system, representing a business concept such as a customer, a car or a car model and containing useful information. Typically, entities are manipulated by boundary and controller objects, rather than having much behavior of their own. Entity classes are the ones that appear on our analysis class diagram.
  - Controller: An object inside the system that encapsulates a complex or untidy process. A controller is a service object that provides the following kinds of service: control of all or part of a system process; creation of new entities; retrieval of existing entities.

Adding Operations to Classes

- Every message on a communication diagram corresponds to an operation on a class
- Record the operations in order to have a complete set of use case realizations
- Operations can be shown on a class diagram in a separate compartment below the attribute compartment.
- Operations can also be documented as a separate **operation list**, to save space

Some operations from iCoot



Class Vocabularies

Static (analysis), Dynamic (analysis), Multiplicity (in association links), Controller (in communication diagrams)

Summary

- The followings were considered:
  - ✓ How to perform the analysis phase of software development
  - ✓ How to build a static analysis model showing the business-oriented objects, along with their relationships, on a class diagram.
  - ✓ How dynamic analysis can improve and verify the static model, using communication diagrams

Outline

- Introduction
- Overview of the Analysis Process
- Static Analysis
- Dynamic Analysis
- Exercise 10

Exercise 10

- Deadline: **2022/06/23 (Thur.) 9:00**
- Please submit your answer file "UML_Ex10_Your group member names.pdf" to "Exercise 10" under "Assignments" tab in Manaba +R
- The maximum points for "Exercise 10" will be **10p**
- If you put a wrong file name or wrong file format, your assignment will not be evaluated. Please be careful!

Ex10 (Group work)

- In the last week, each group selected their own business and did the requirement analysis for the business.
- This week, each group will use the requirement analysis results from last week as input to do the problem analysis for their business.
- This week, please design:
  1. An analysis class diagram as shown in the slide #13.
  2. A class attribute list as shown in the slide #19.
  3. A class operation list as shown in the slide # 27.
- For the analysis class diagram, there are following requirements
  1. Each relationship should include the multiplicity and association label/role.
  2. At least one comment should be given to one of the classes.
  3. Design at least one association class