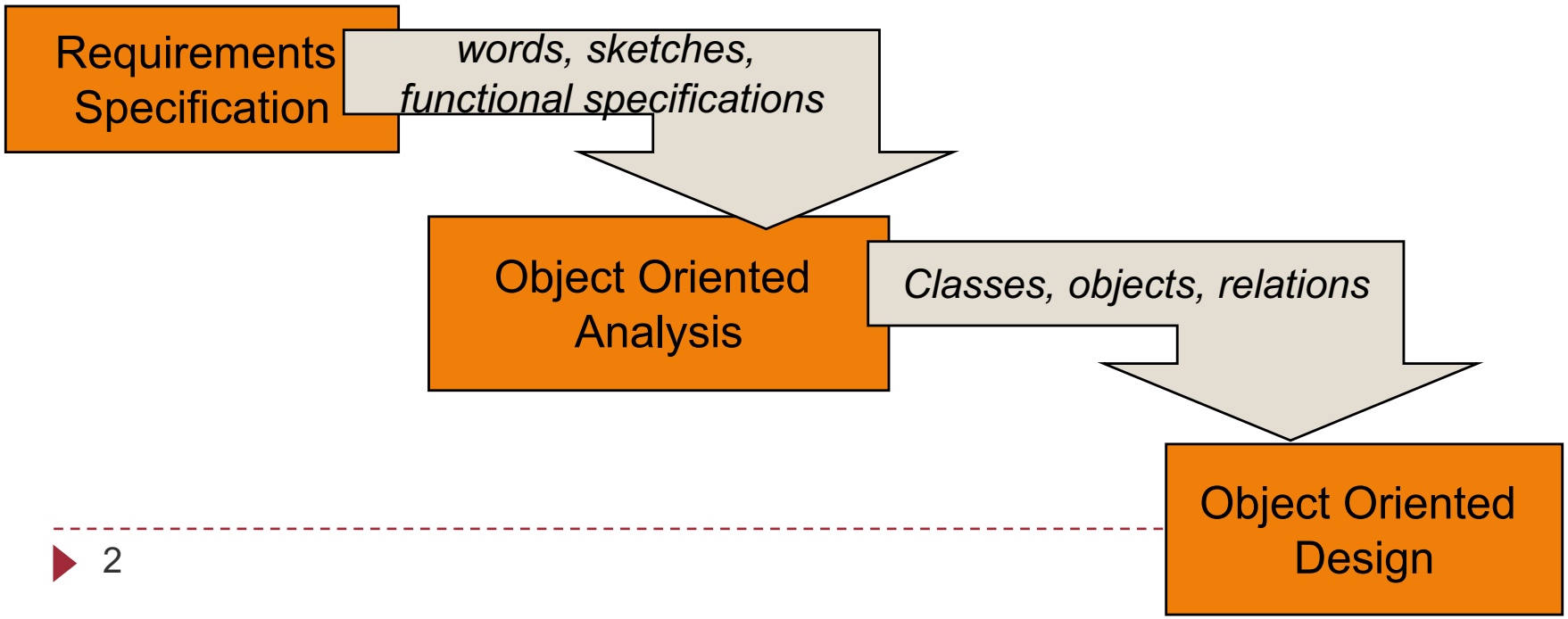
Object-Oriented Analysis techniques

- OO Analysis and Design - moving into design
- Class Diagrams (again)
- Royal Service Station case study
- CRC Diagrams
- Template Classes
- Exercises

Object-oriented analysis (OOA)

- Goal: to fully specify the problem and the application domain **without introducing a bias to any particular implementation**. [Rumbaugh et. al 91]
- OOA is a transition from specifications to design



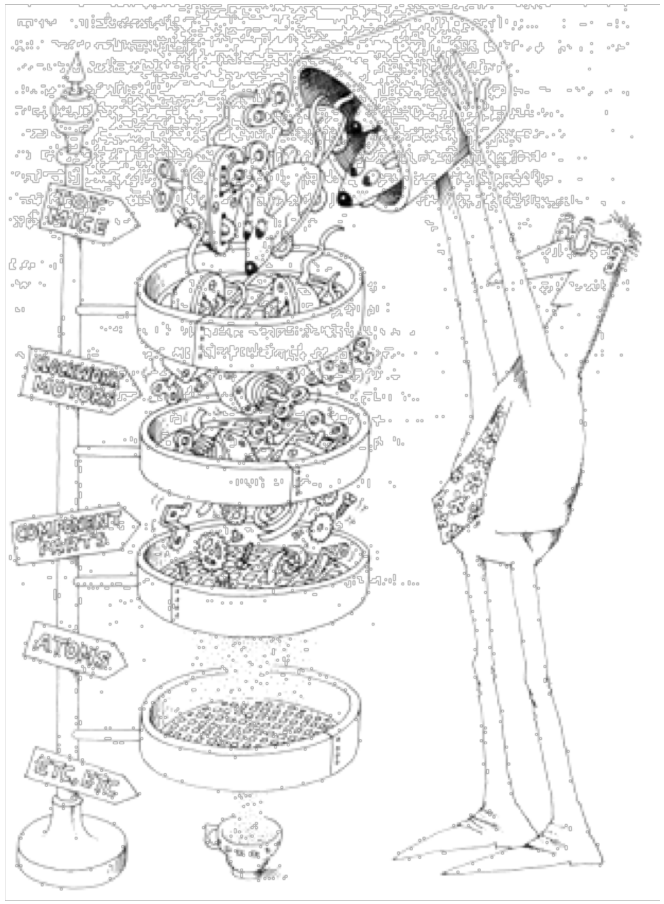
Vocabulary and notations

- Vocabulary of OOA: Similar to object-oriented design:
  - Classes and objects
  - Generalization/specialization (‘inheritance’)
- Notations:
  - Class diagrams
  - Sequence diagrams
  - Interaction diagrams
  - Type diagrams

Analysis vs. design

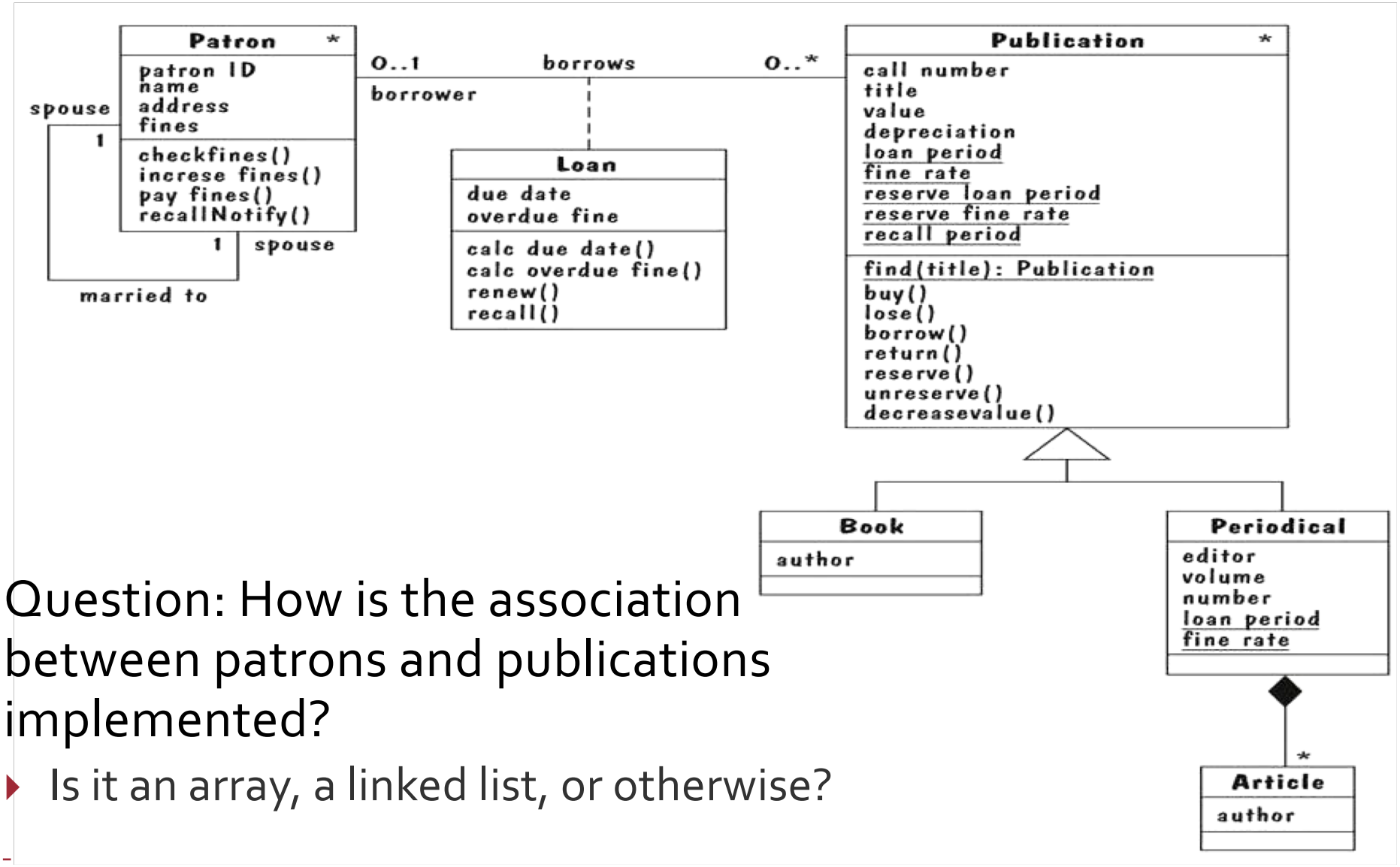
- Software analysis is more **abstract** than software design in the following senses:

	Analysis	*Design*
purpose	what	how
activity	analysis	synthesis
vocabulary	problem	solution
subject	requirements	implementation



More on Class Diagrams

Class diagrams example: library



- Question: How is the association between patrons and publications implemented?
  - Is it an array, a linked list, or otherwise?

Possible ways the association between Patrons and Publications could be implemented

- Using either a **Set** or a **List**.
- If the Loan class contains information about the way a Patron borrows a publication (for example, the due date), you could implement it like this:

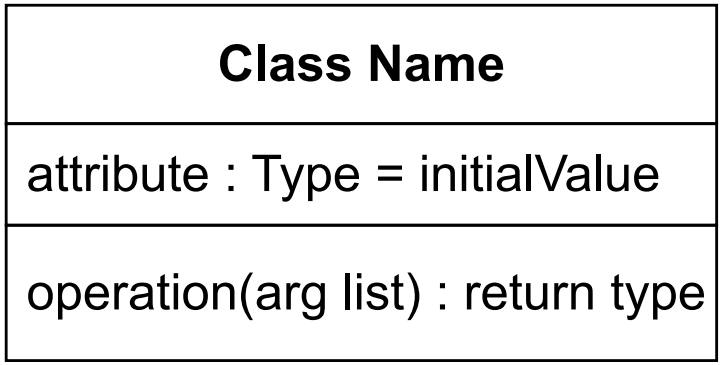
```
class Patron
{
      Set<Loan> loans;
 }

class Loan
{
      Patron patron;
      Publication publication;
      Date dueDate;
}

class Publication
{
      Set<Loan> loans;
}
```

- Alternatively: Using a **Map**

```
public class Patron
{
private Map<Publication, Loan> loansByPublication;
}

public class Loan
 {
 //Loan's properties
Date dueDate;
}

public class Publication
{
private Map<Patron, Loan> loansByPatron;
}
```

Requirement modelling technique: class diagrams

- 3 parts to a class
  - name
  - attributes
  - operations



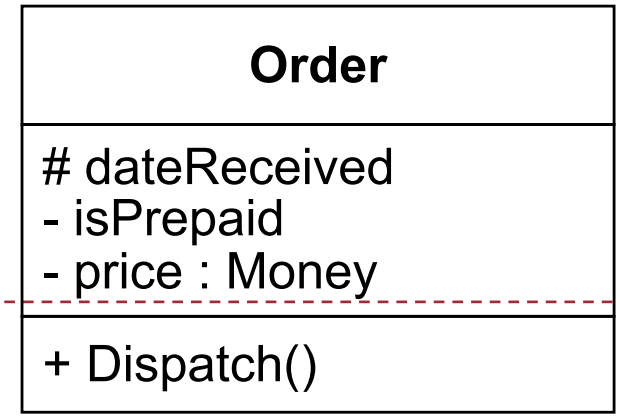
- Access Specifications

+ Public (Any class that can see the container can also see and use the classes)

Protected (Only classes in the same container or a descendent of the container can see and use the classes.

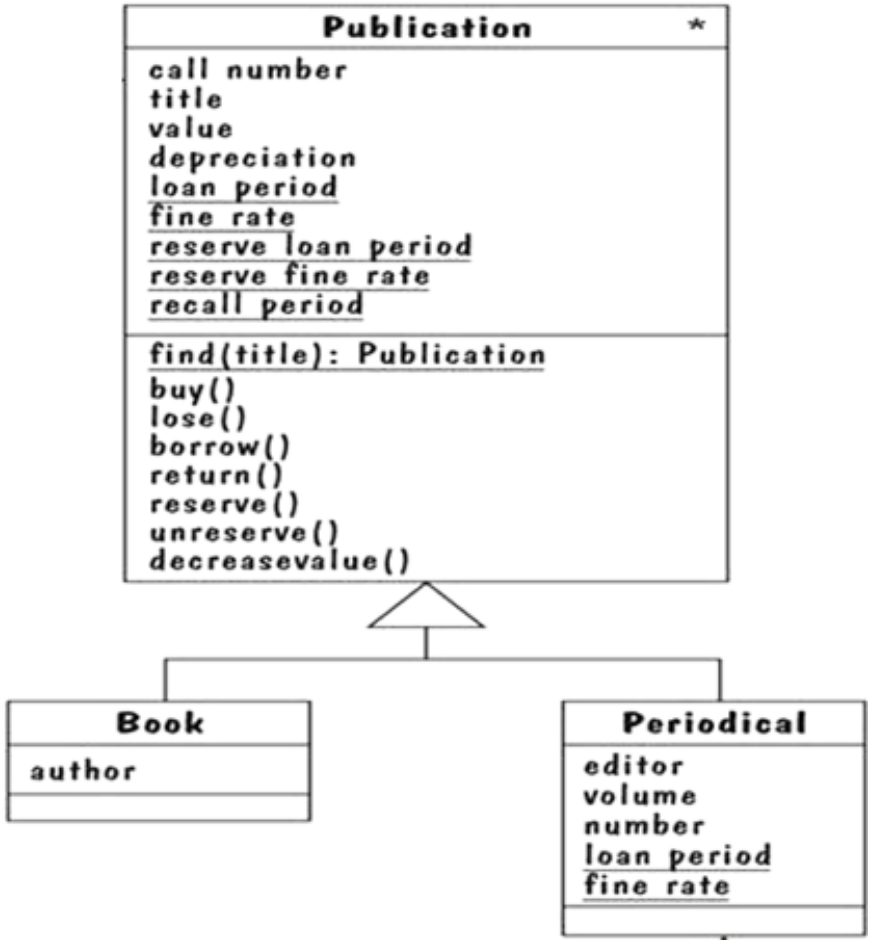
- Private (only classes in the same container can see and use the classes)

~ Package (Only classes within the same package as the container can see and use the classes)



Class diagrams: inheritance

- Inheritance captures the "is-kind-of" relation
  - Think of subclasses as subcategories
- Use it to define classes incrementally



Adding Generalization Structure

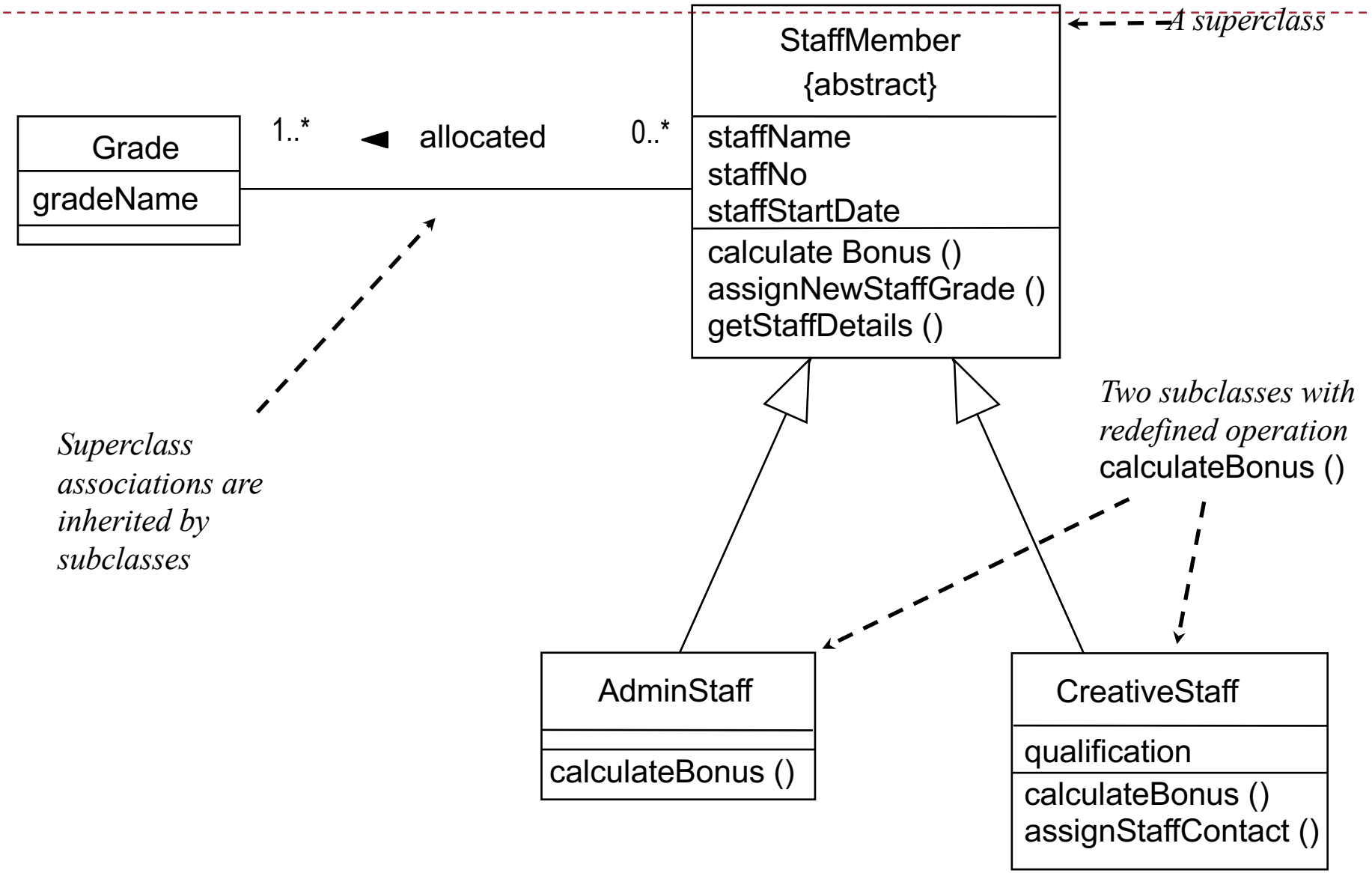
- Add generalization structures when:
  - Two classes are similar in most details, but differ in some respects
  - May differ:
    - In behaviour (operations or methods)
    - In data (attributes)
    - In associations with other classes

Adding Structure

- Two types of staff:

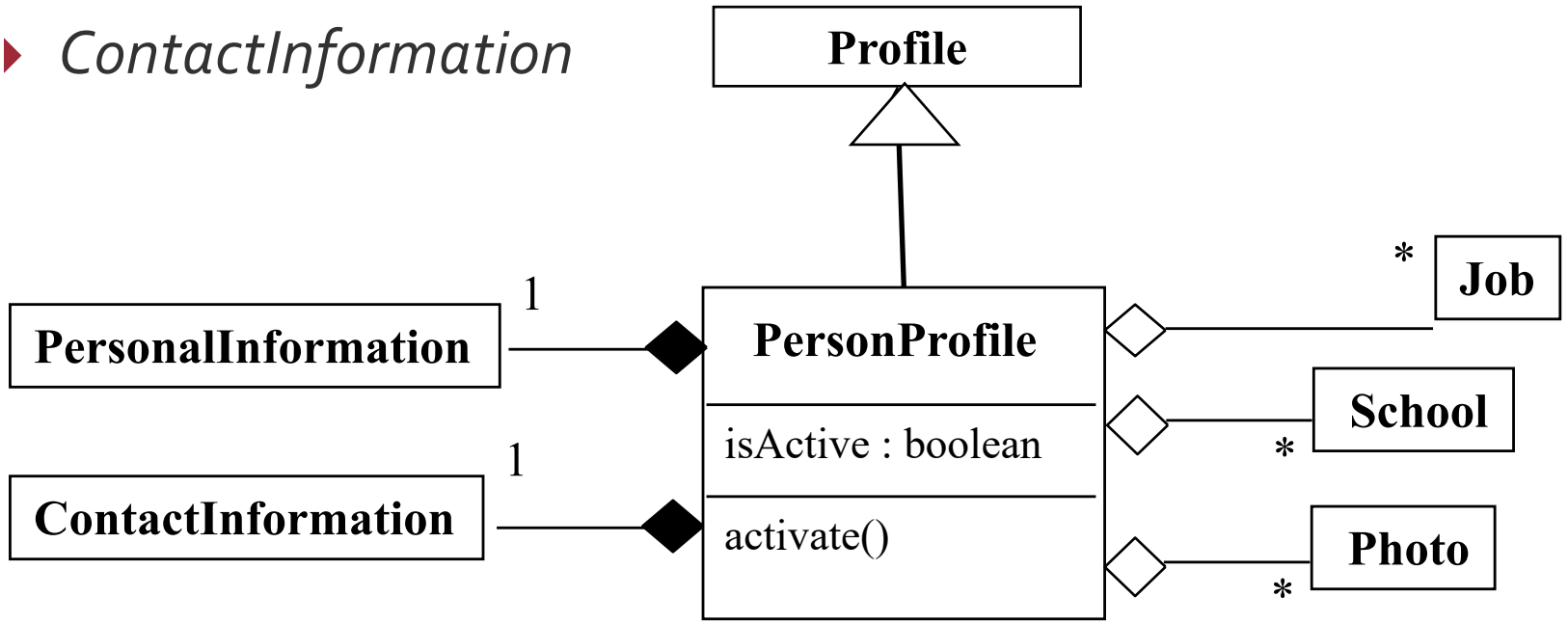
Creative	Have qualifications recorded Can be client contact for campaign Bonus based on campaigns they have worked on
Admin	Qualifications are not recorded Not associated with campaigns Bonus not based on campaign profits

Adding Structure:



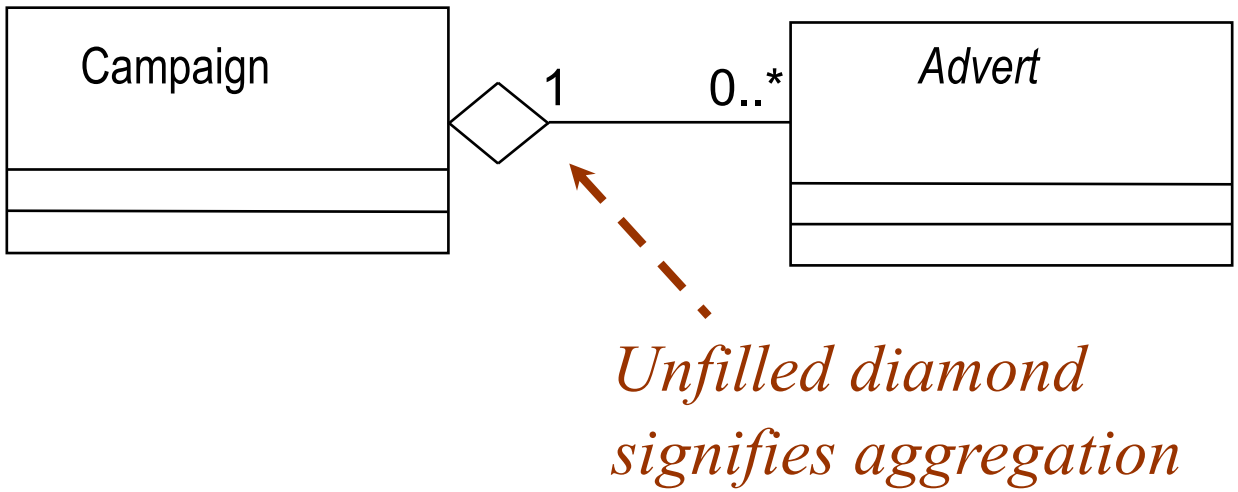
Class diagrams: Aggregation Vs. Composition

- Aggregation: Collection of elements
  - Photos, jobs, schools
- Composition: Whole-part relation
  - *PersonalInformation*
  - *ContactInformation*



Aggregation and Composition

- Special types of association, both sometimes called whole-part
- A campaign is made up of adverts:

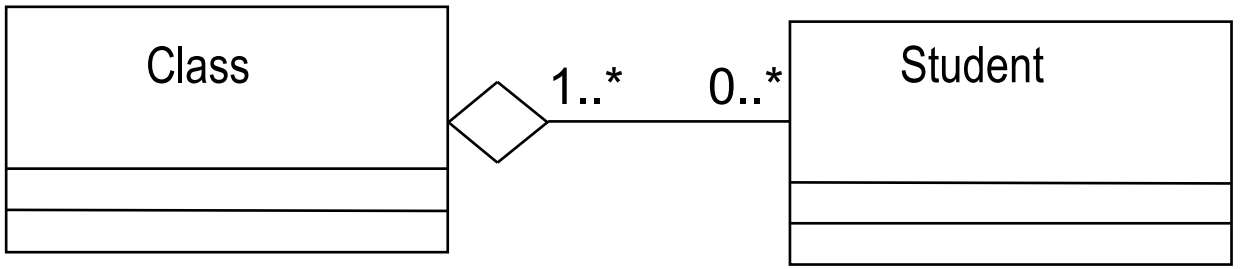


Aggregation and Composition

- Aggregation is essentially any whole-part relationship
- Semantics can be very imprecise
- Composition is ‘stronger’:
  - Each part may belong to only one whole at a time
  - When the whole is destroyed, so are all its parts

Aggregation and Composition

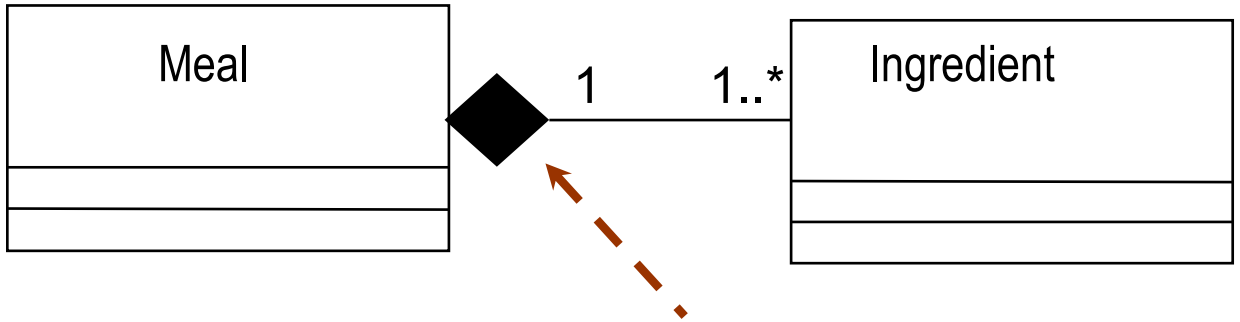
- An everyday example:



- Clearly not composition:
  - Students could be in several classes
  - If class is cancelled, students are not destroyed!

Aggregation and Composition

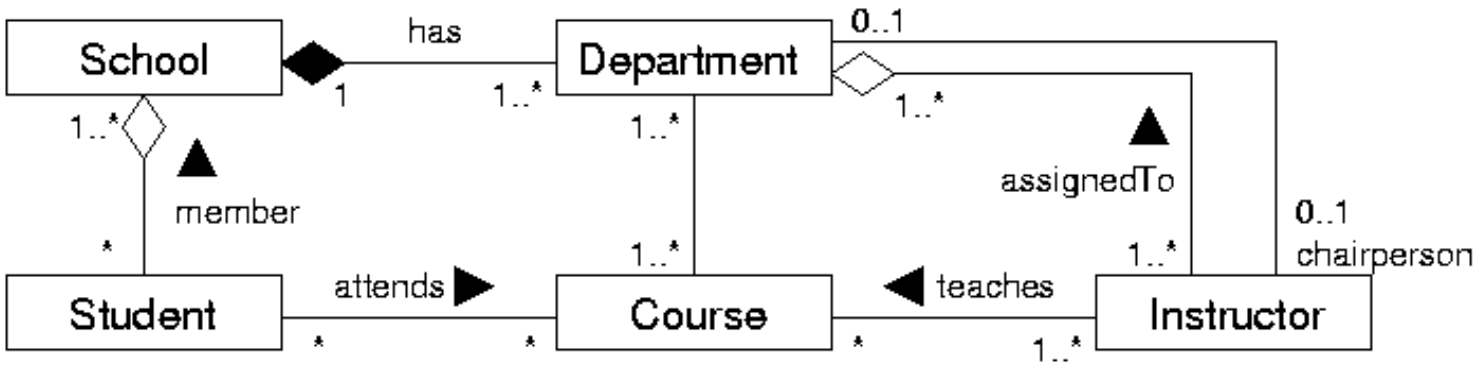
- Another everyday example:



Filled diamond signifies composition

- This is (probably) composition:
  - Ingredient is in only one meal at a time
  - If you drop your dinner on the floor, you probably lose the ingredients too

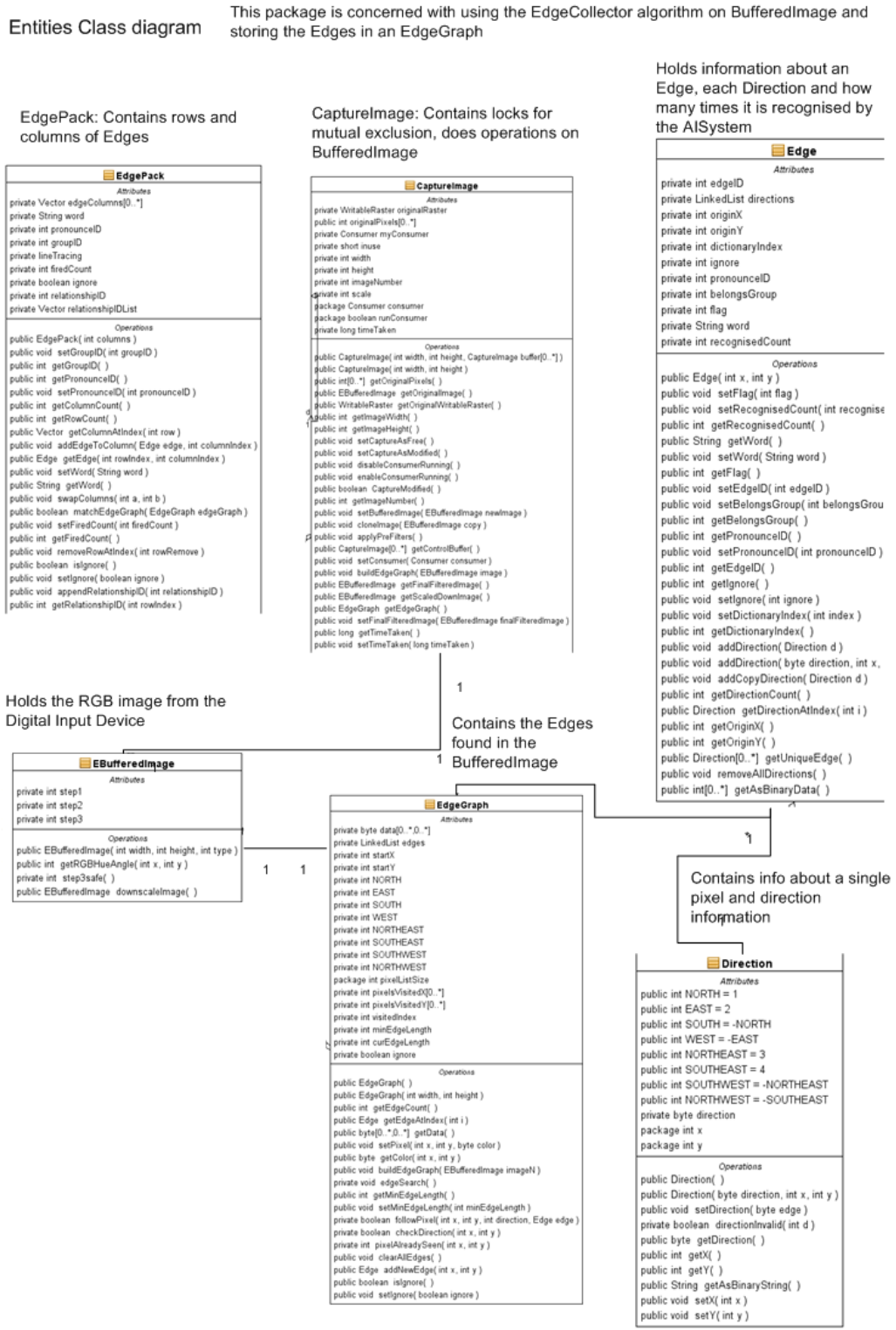
Class diagrams: cardinality (‘multiplicity’)



- Note that associations can—
  - be directed
  - be labelled
  - have a cardinality on each end

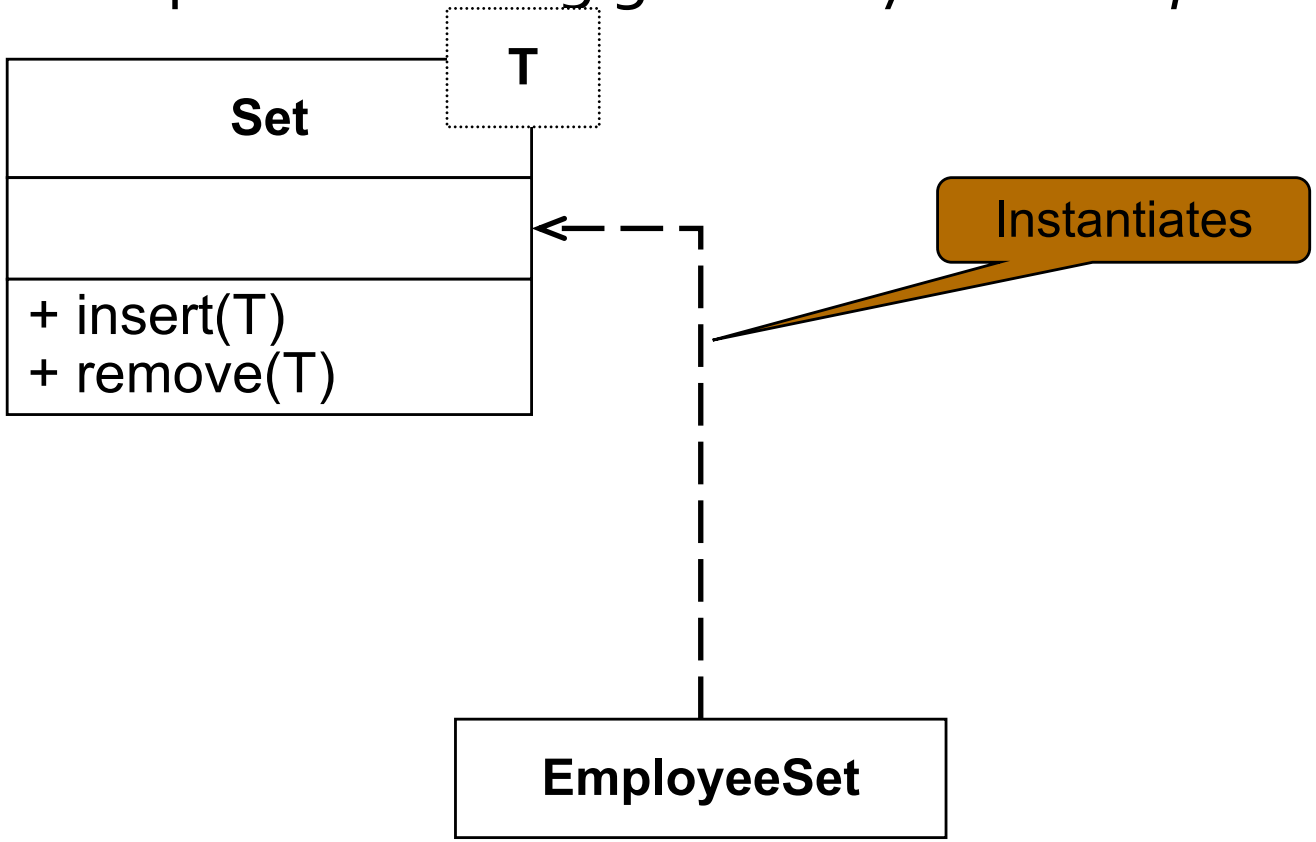
Bad class diagrams

- What's wrong with this diagram?



Class diagrams: modelling genericity

- Class diagrams are commonly used to specify detailed design
- For example: modelling genericity and *templates*



Template Classes

- The Template Parameter specifies the formal parameters which will be substituted by the actual parameters in a binding. A template class defines a family of classes. You may create class with template parameter to form the template class.
- Benefits:
  - A template class allows for its functionality to be reused by any bound class.
  - Templates are an efficient way of allowing one piece of code to operate on many different types. Easy to handle requirement changes.
  - By creating templates for classes, you benefit from having clear separation of implementation. If any change request has to be done, it can be easily handled by modifying the template, and the changes will be reflected in the implementation for all instances.

Example

- Suppose that you have an application where you need a list of objects of class Customer.
- In order to use a template, we must first create a class based on the template, a process commonly referred to as instantiation, and then use the newly created list class for creating list objects.

We might declare a List template as follows:

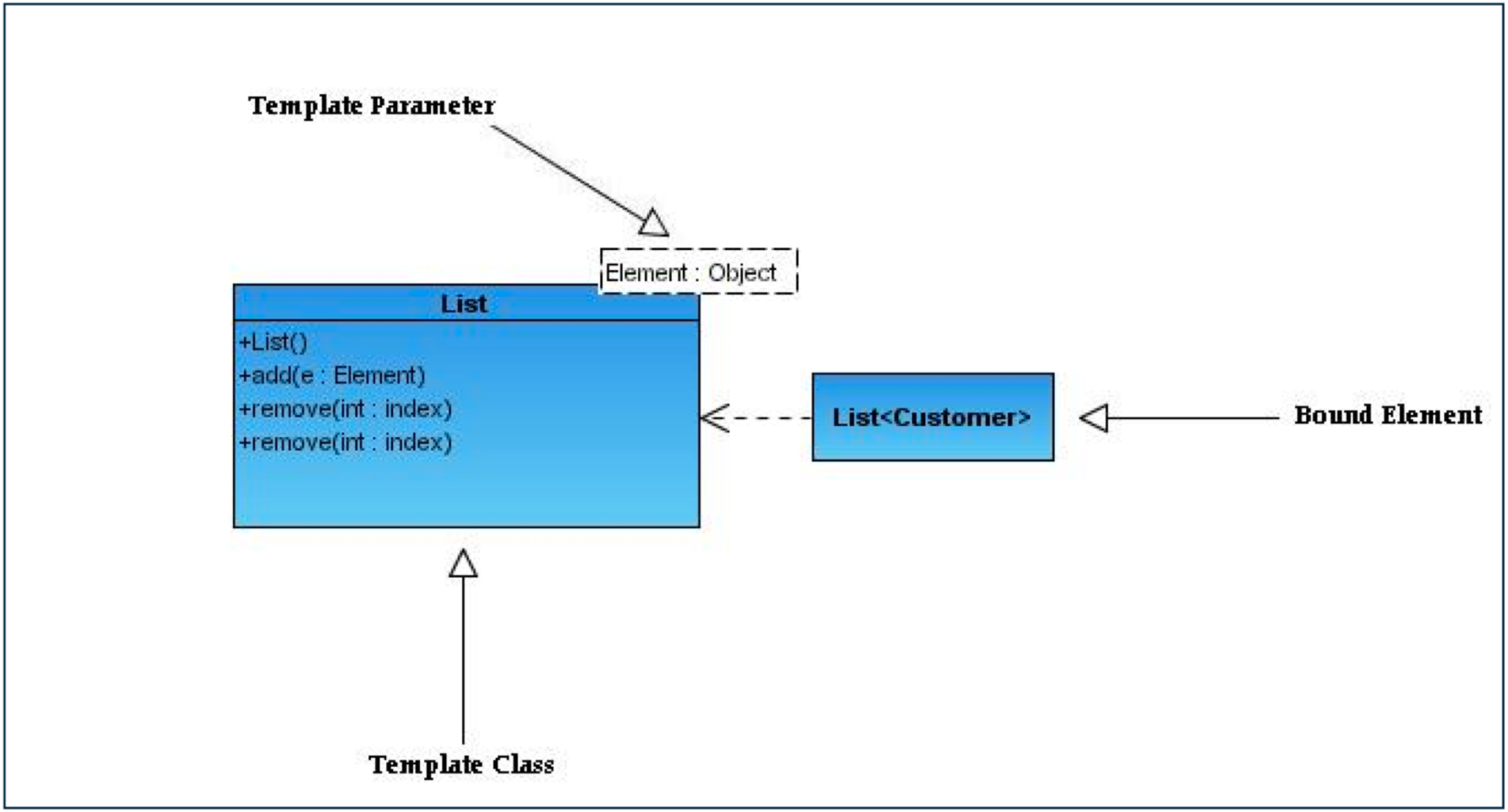
```
public class List
{
  public List() {}
  public void add (Element e) {}
  public void remove(Element e){}
  public void remove (int index) {}
}
```

```
// instantiate
class CustomerList = new List<Customer>;

// create new object for CustomerList
CustomerList custList = new CustomerList();

// create a new customer object
Customer c = new Customer();

// put the customer on the list
custList.add(c)
```

The above demonstrates how to use the template.

Example: Template in UML

Below diagram shows the binding between the model class **Customer** and the **List** template.



OOA Case Study: Royal Service Station

From text to class diagrams

Royal Service Station

Requirements

- Royal Service station provides three types of services
- The system must track bills, the product and services
- System to control inventory
- The system to track credit history, and payments overdue
- The system applies only to regular repeat customer
- The system must handle the data requirements for interfacing with other system
- The system must record tax and related information
- The station must be able to review tax record upon demand
- The system will send periodic message to customers
- Customer can rent parking space in the station parking lot
- The system maintain a repository of account information
- The station manager must be able to review accounting information upon demand
- The system can report an analysis of prices and discounts
- The system will automatically notify the owners of dormant accounts
- The system can not be unavailable for more than 24 hours
- The system must protect customer information from unauthorized access

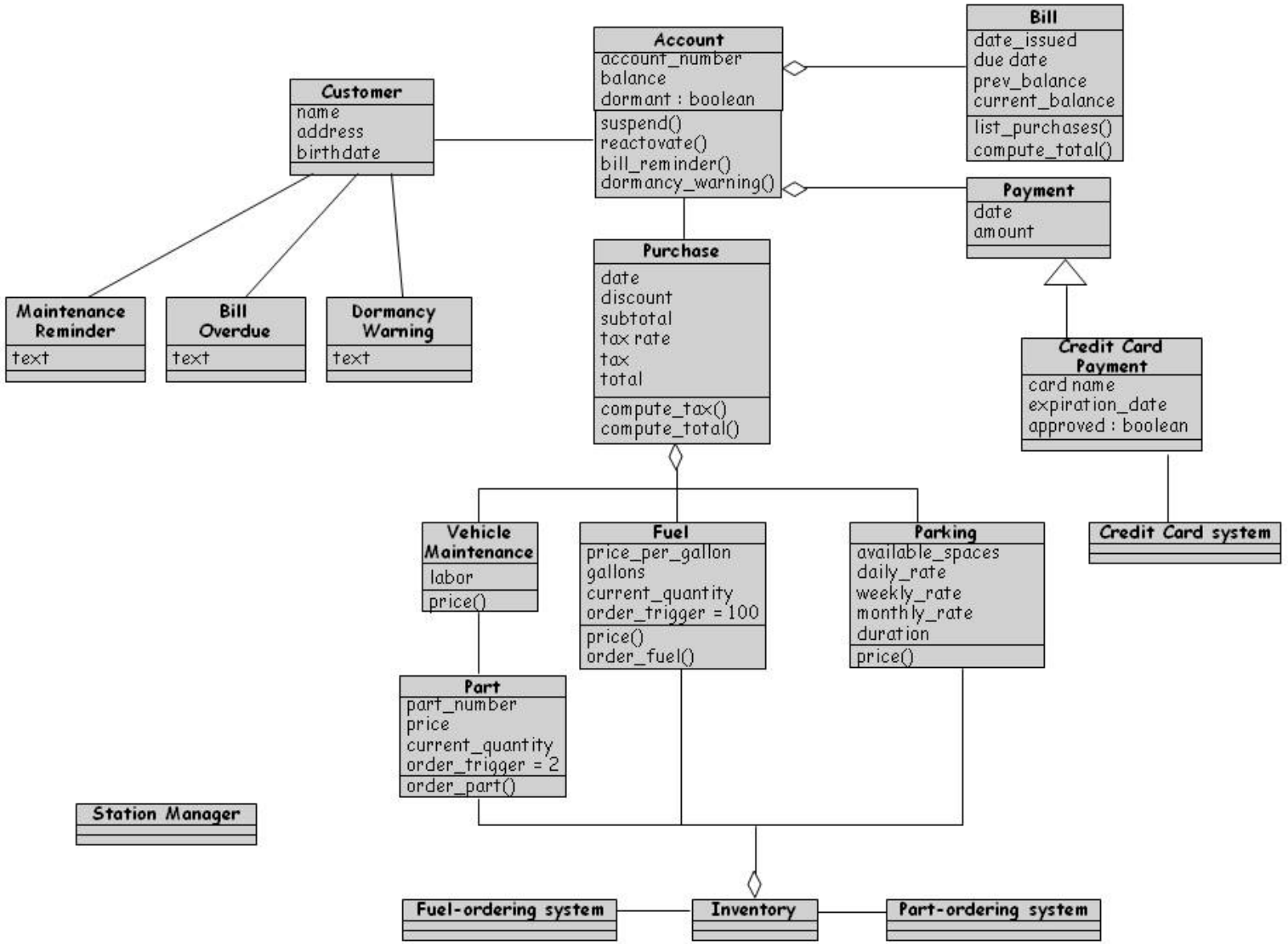
Identifying Behaviors

- Imperative verbs – at the beginning of sentences
- Passive verbs
- Actions
- Membership in
- Management or ownership
- Responsible for
- Services provided by an organization eg. use-cases

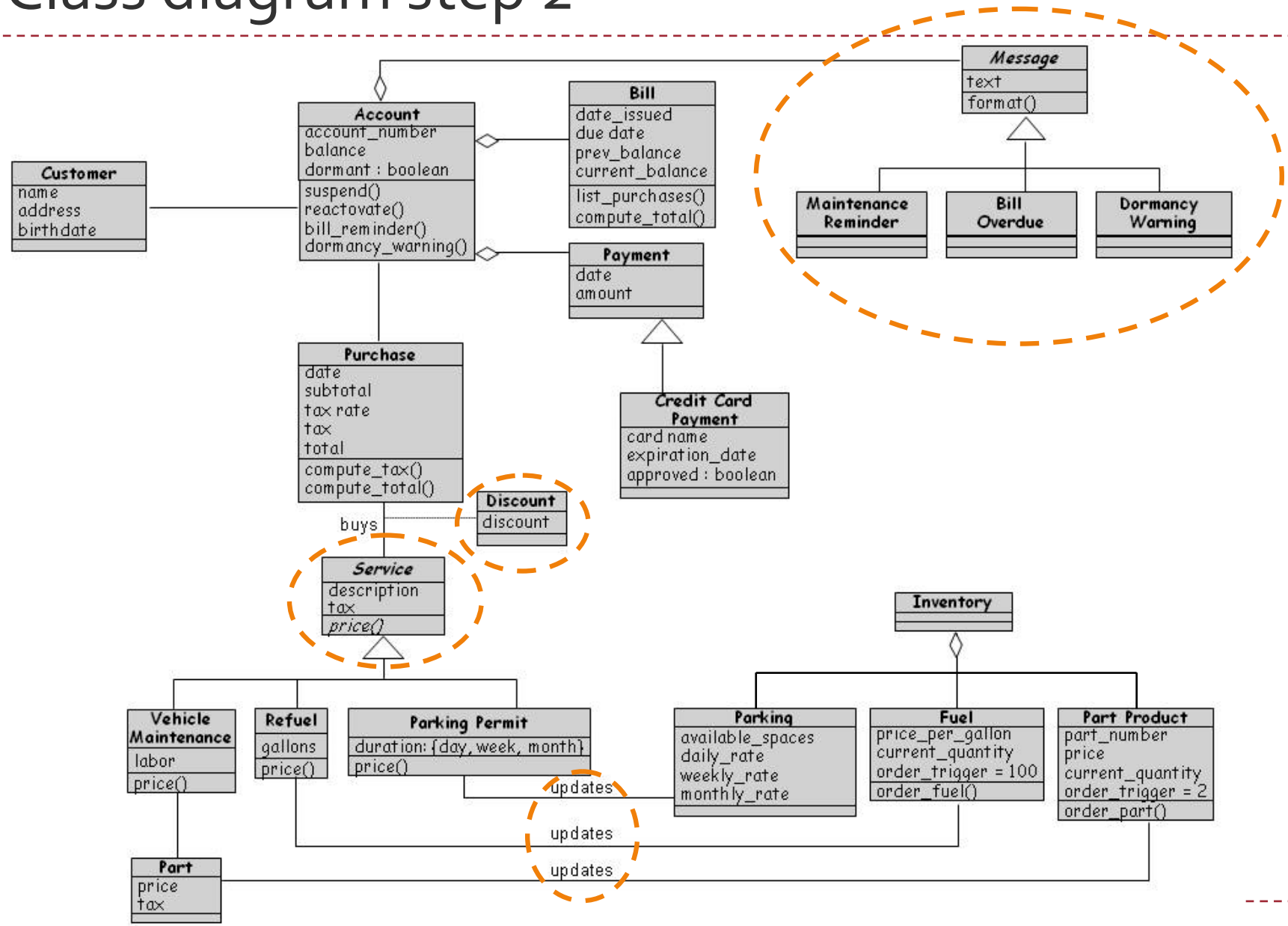
Grouping of Attributes and Classes

Attributes	Classes
Personal check	Customer
Tax	Maintenance
Price	Services
Cash	Parking
Credit card	Fuel
Discounts	Bill
Name	Purchase
Address	Maintenance reminder
Birthdate	Station manager
	Overdue bill letter
	Dormant account warning
	Parts
	Accounts
	Inventory
	Credit card system
	Part ordering system
	Fuel ordering system

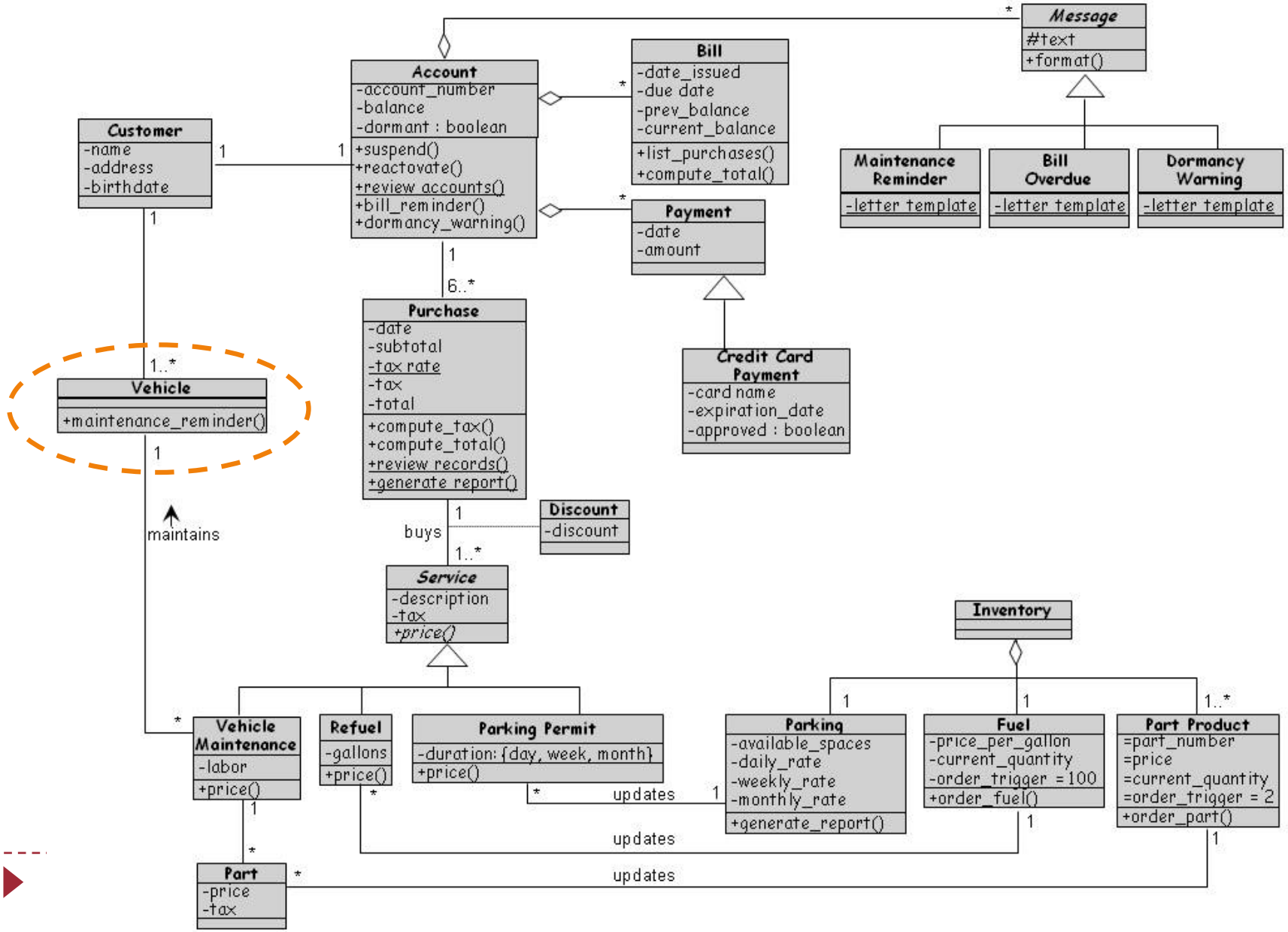
Class diagram step 1 (based on our knowledge of the domain)



Class diagram step 2



Class diagram step 3



Class / Responsibility / Collaborations

CRC Cards

- Class–Responsibility–Collaboration cards help to model interaction between objects
- Used as a way of:
  - Identifying classes that participate in a scenario
  - Allocating responsibilities - both operations and attributes (*what can I do?* and *what do I know?*)
- For a given scenario (or use case):
  - Brainstorm the objects
  - Allocate to team members
  - Role play the interaction

OOA technique: CRC cards

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*

CRC Cards: Example

Class Name *Vehicle Traffic Light*	
Responsibilities	Collaborations
Receive Open or Close signal	*Intersection (send signals)* *Road (ingoing, outgoing)*

From use case to CRC

Example use case description

The campaign manager selects the required campaign for the client concerned and adds a new advert to the existing list of adverts for that campaign. The details of the advert are completed by the campaign manager.

- Identify classes involved
- Work through the scenario to identify the CRC responsibilities and collaborations

CRC cards: Examples (cont.)

Class Name	*Client*
Responsibilities	Collaborations
Provide client information. *Provide list of campaigns.*	*Campaign provides campaign details.*

Class Name	*Campaign*
Responsibilities	Collaborations
Provide campaign information. *Provide list of adverts.* *Add a new advert.*	*Advert provides advert details.* *Advert constructs new object.*

Class Name	*Advert*
Responsibilities	Collaborations
Provide advert details. *Construct adverts.*	

CRC Cards

- Effective role play depends on an explicit strategy for distributing responsibility among classes
- For example:
  - Each role player tries to be lazy
  - Persuades other players *their* class should accept responsibility for a given task
- May use ‘Paper CASE’ to document the associations and links

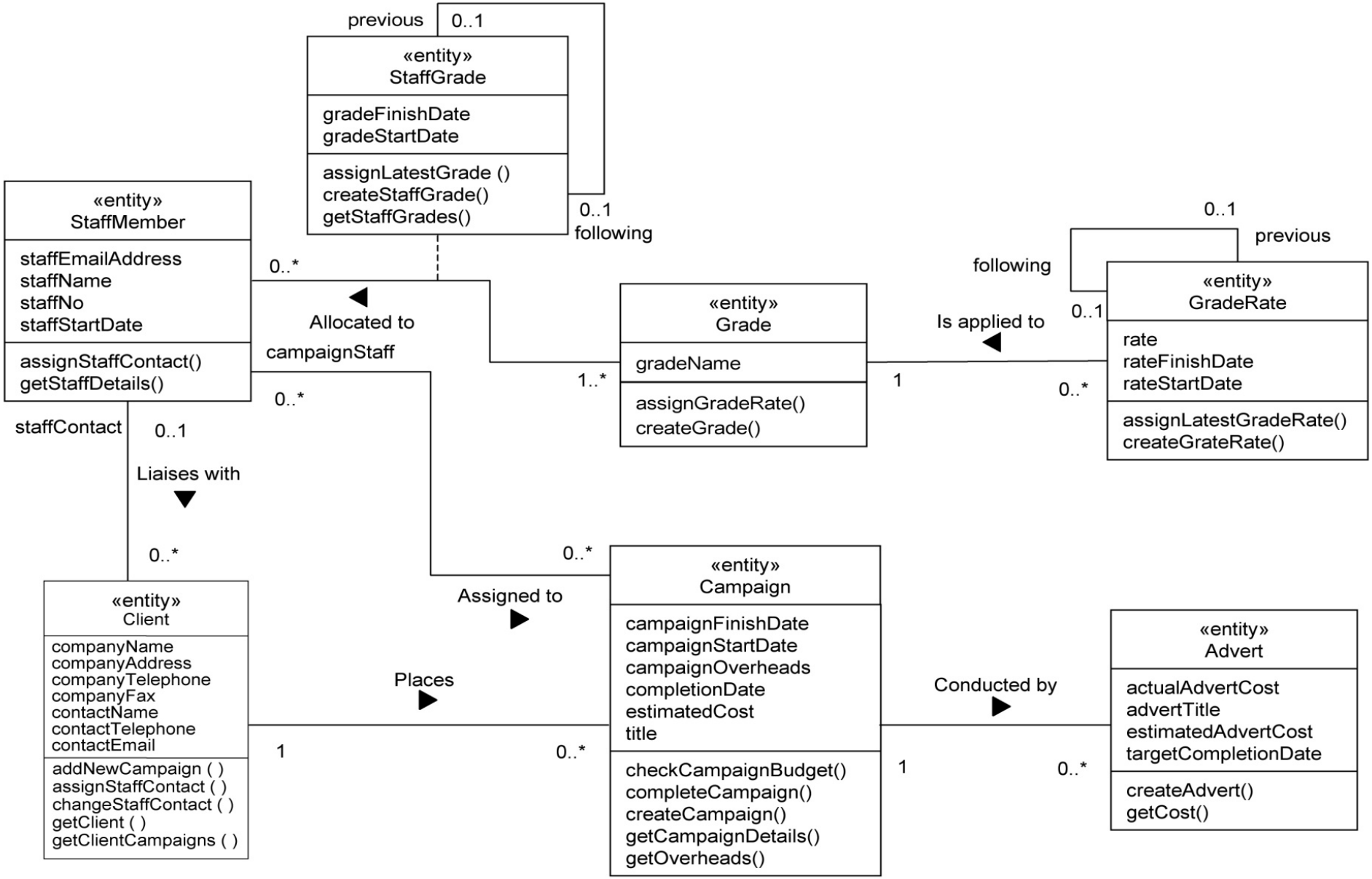
Summary

- OO Analysis and Design - moving into design
- Class Diagrams
  - Access specifications
  - Inheritance
  - Associations
  - Composition/Aggregation
  - Cardinality
- Template Classes
- Royal Service Station case study
- CRC Diagrams

Further reading

- James Rumbaugh (1991). Object-oriented modeling and design. Prentice-Hall.
- Chapter 8 Bennett – refining the Analysis model
- Chapter 7 Bennett – CRC diagrams
- Pfleeger – Section 6.4, Representing OO Designs in the UML (Royal Service Station example)

Exercise: translate relations to English

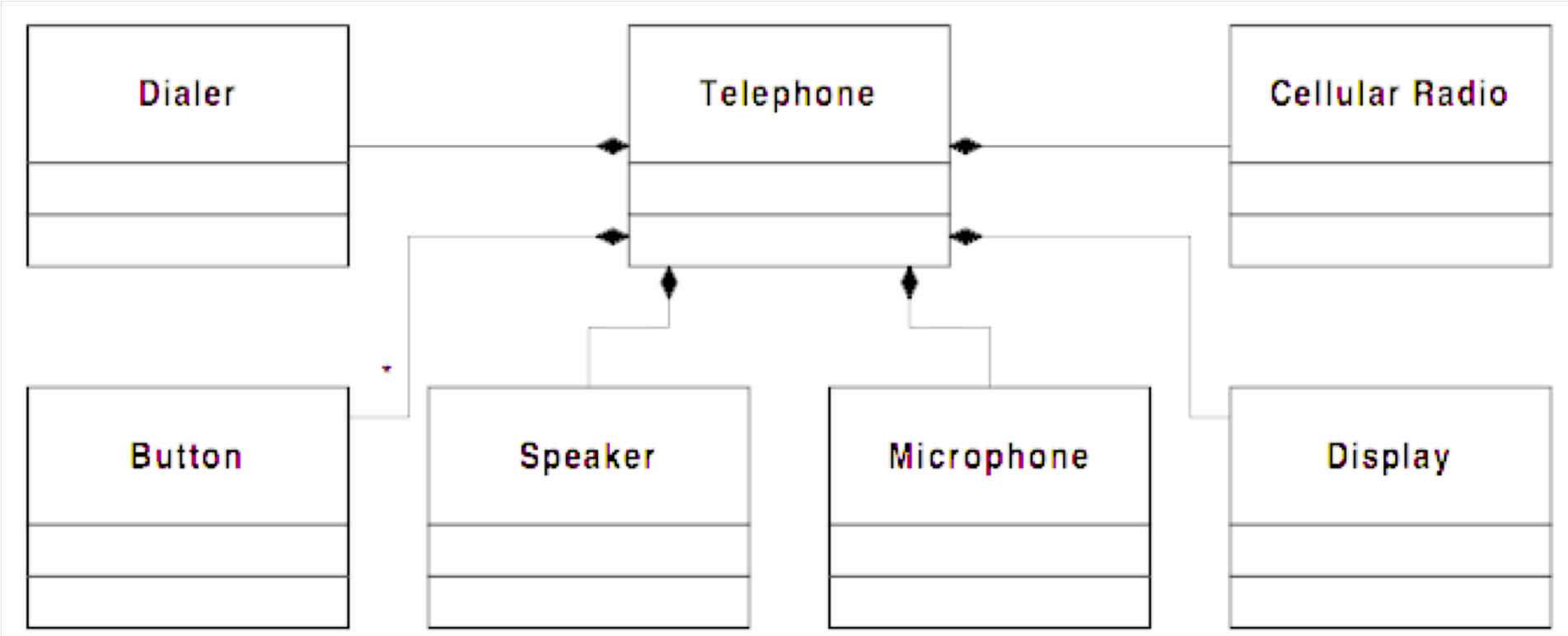


Exercise: reconciling class and collaboration diagrams (part 1)

- Consider the software that controls a very simple cellular telephone. Such a phone has buttons for dialing digits, and a "send" button for initiating a call. It has "dialer" hardware and software that gathers the digits to be dialed and emits the appropriate tones. It has a cellular radio that deals with the connection to the cellular network. It has a microphone, a speaker, and a display.
- Draw a simple class diagram for this based around a simple Telephone class.

Possible answer:

- What is wrong with this?



Exercise: part 2

- Draw a collaboration diagram for the following use case:

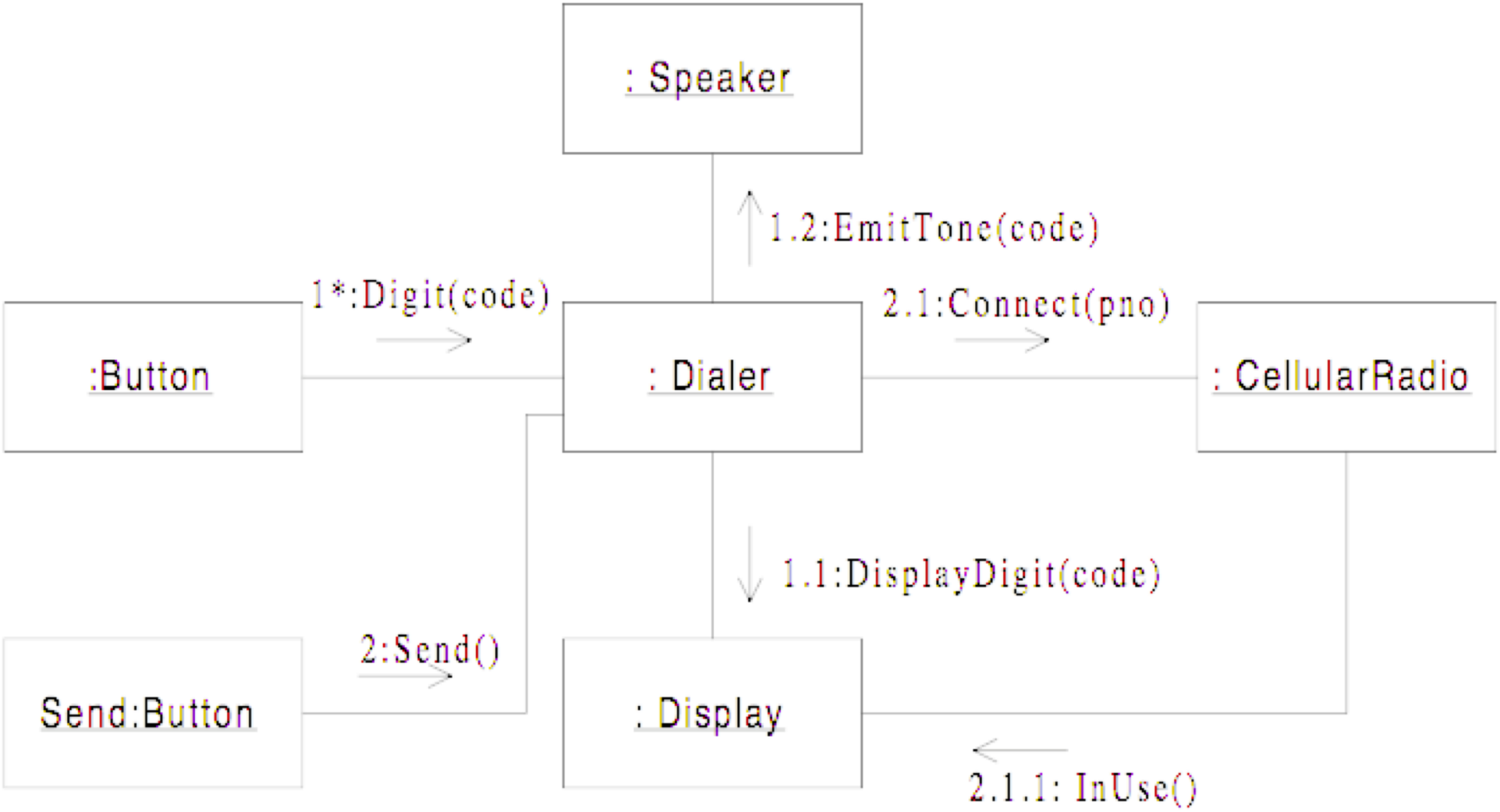
Use case: Make Phone Call

- 1. User presses the digit buttons to enter the phone number.
- 2. For each digit, the display is updated to add the digit to the phone number.
- 3. For each digit, the dialer generates the corresponding tone and emits it from the speaker.
- 4. User presses "Send"
- 5. The "in use" indicator is illuminated on the display
- 6. The cellular radio establishes a connection to the network. 7. The accumulated digits are sent to the network.
- 8. The connection is made to the called party.

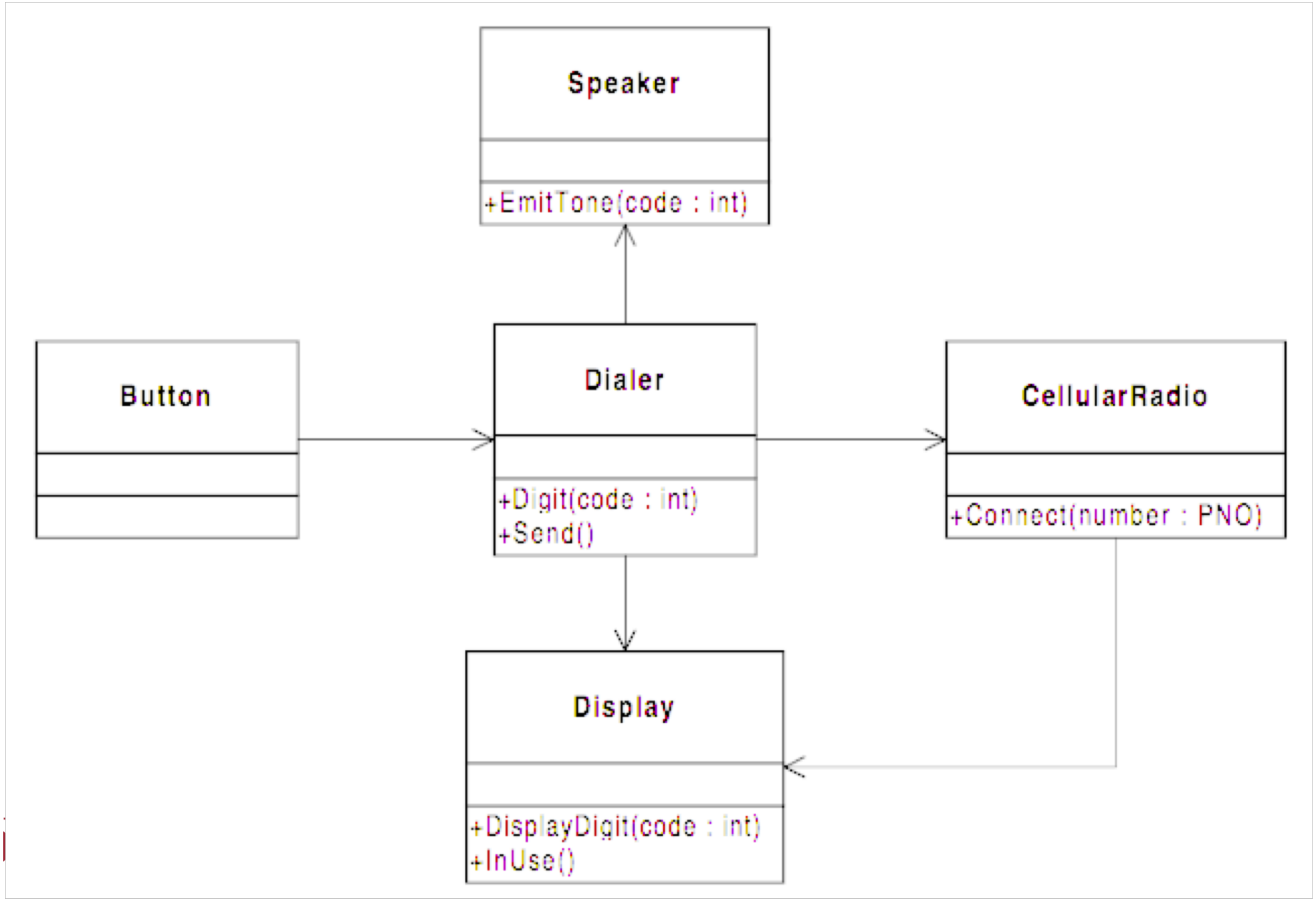
- Does your dynamic model match to your static model?
- If not, how can you reconcile this?
- Reconcile your static model

Possible answer

- How do we reconcile this with the class diagram?



Outline of a solution



Exercise: Relationships, roles and classes

- Draw a class diagram representing the relationship between parents and children. Take into account that a person can have both a parent and a child. Annotate associations with roles and multiplicities.