

Please sit with your project group  
(if you have one)

# Class, Responsibility, and Collaboration

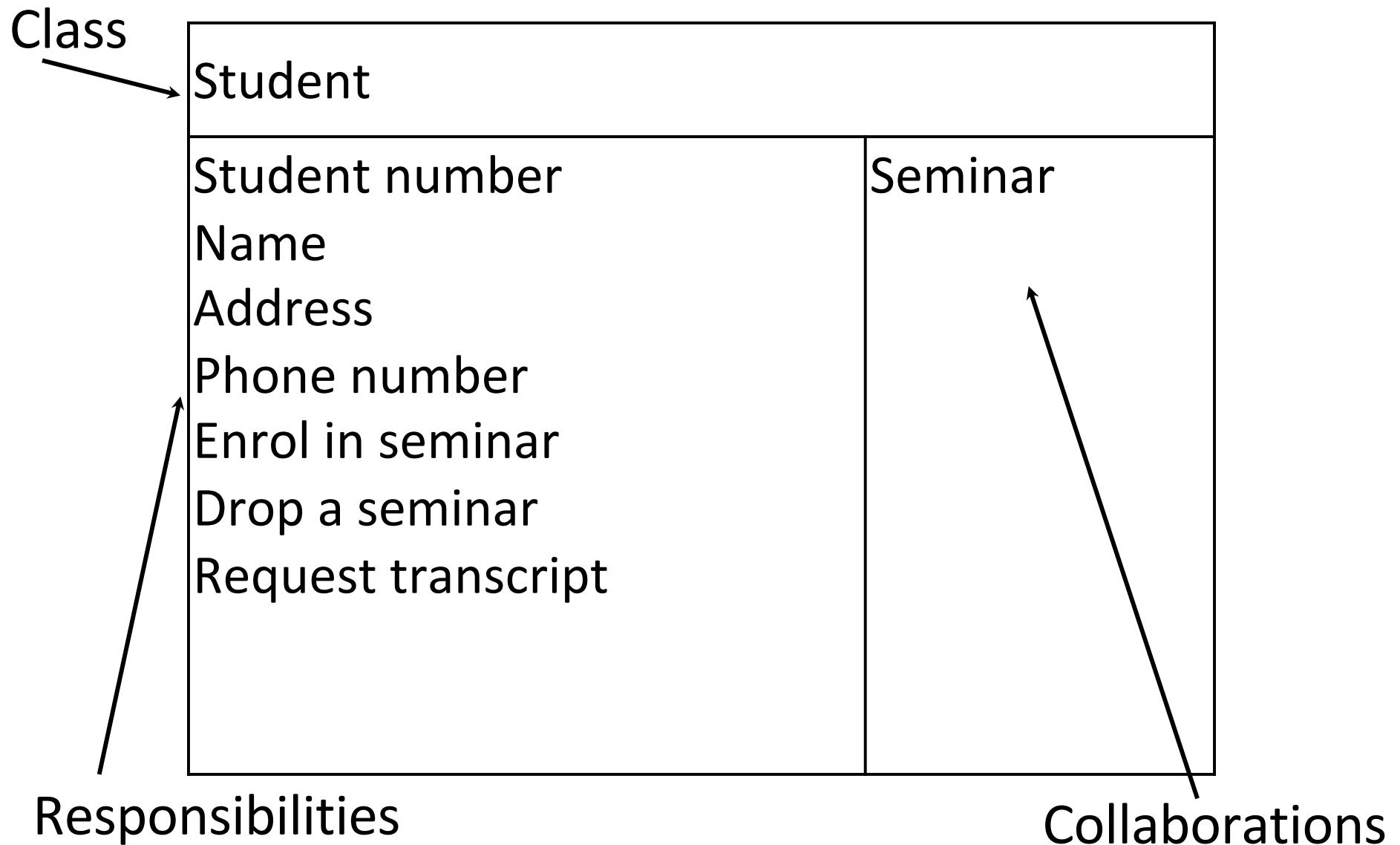
# CRC Cards

- A tool and method for systems analysis and design.
- Part of the Object-Oriented development paradigm.
- Highly interactive and human-intensive.
- Final result: initial set of classes and their relationships.
- *What* rather than *How*.
- Benefits:
  - Cheap and quick: all you need is index cards.
  - Simple, easy methodology.
  - Forces you to be concise and clear.
  - Input from every team member.

# What is a CRC Card?

- **CRC** stands for Class, Responsibility and Collaboration.
  - **Class**
    - An object-oriented class name
    - Include information about super- and sub-classes
  - **Responsibility**
    - What information this class stores
    - What this class does
    - The behaviour for which an object is accountable
  - **Collaboration**
    - Relationship to other classes
    - Which other classes this class uses

# What does a CRC Card Look Like?



# CRC Model

Enrollment	
Mark(s) received Average to date Final grade Student Seminar	Seminar

Transcript	
**See the prototype** Determine average mark	Student Seminar Professor Enrollment

Student Schedule	
**See the prototype**	Seminar Professor Student Enrollment Room

Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

Building	
Building Name Rooms Provide name Provide list of available rooms for a given time period	Room

A **CRC Model** is a collection of CRC cards.

It specifies the Object-Oriented Design (OOD) of the software system.

# How to Create a CRC Model?

- Typically, you are given a description (in English) of the requirements for a software system.
- You work in a team.
- Ideally, you all gather around a table.
- You need a set of index cards and some pens.
- Coffee / other beverages are optional.

# How to Create a CRC Model?

- Read the description. Again. And again.
- Identify core **classes** (simplistic advice: look for nouns).
- Create a card per class (begin with class names only).
- Add **responsibilities** (simplistic advice: look for verbs).
- Which other classes does this class need to talk to to fulfil its responsibilities? Add **collaborators**.
- Add more classes as you discover them.
- Put classes away if they become unnecessary. (But don't tear them up yet!)
- Refine by identifying abstract classes, inheritance, etc.
- Keep adding/refining until everyone on the team is satisfied.

# How Can We Tell It Works?

- A neat technique: a **Scenario Walk-through**.
- Select a scenario and choose a plausible set of inputs for it.
- Manually “execute” each scenario.
  - Start with initial input for scenario and find a class that has responsibility for responding to that input.
  - Trace through the collaborations of each class that participates in satisfying that responsibility.
  - Make adjustments as necessary.
  - Repeat until scenario has “stabilized” (that is, no further adjustments are necessary).

# Developing a CRC Model

# Example

- Consider this description of a software system:
  - *You are developing a software system to facilitate restaurant reviews. Each restaurant corresponds to a certain price range, neighbourhood, and cuisines it serves. Restaurants that serve alcohol must have a license, which they need to renew every year. The system should also report how long, on average, customers wait for take out in restaurants that offer take-out service. When reviewers leave a review for a restaurant, they must specify a recommendation (Thumbs Up or Thumbs Down) and can also leave a comment. An owner of a restaurant can respond to a review with a comment. All users of the system log in with their username. Users can choose to be contacted by email ...*

# Key Steps

A key part of developing the model involves careful analysis of the problem specification. We must:

- **Identify important nouns.**

Underline nouns that may make sensible classes or that describe information a class could be responsible for storing.

- **Choose potential classes.**

From the nouns identified, write down the ones that are potential classes.

- **Identify verbs that describe responsibilities.**

In the problem description, circle verbs that describe tasks that a class may be responsible for doing.

# Identify important nouns

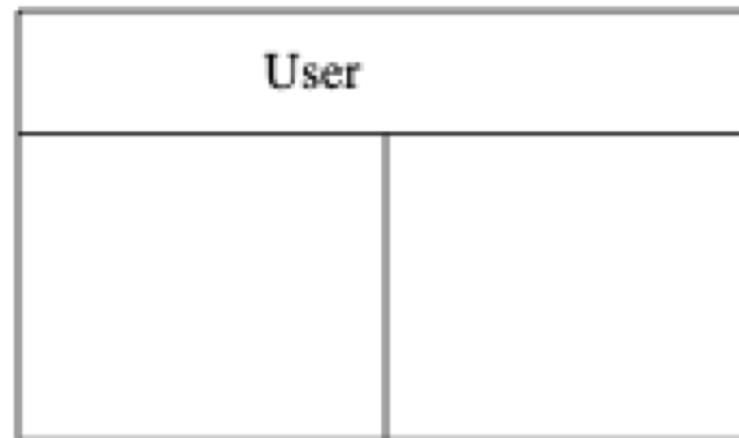
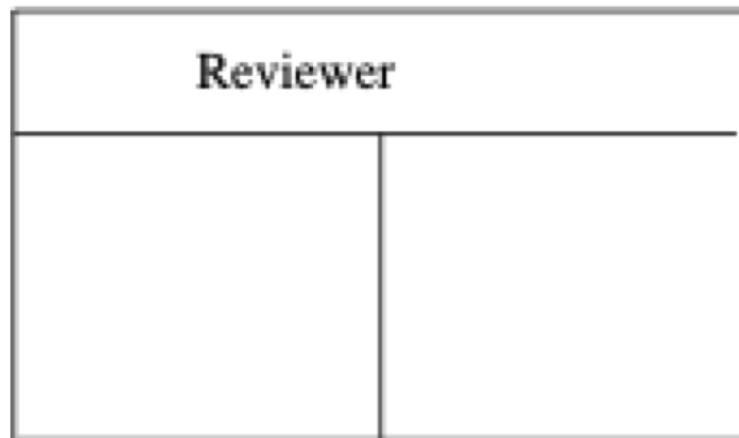
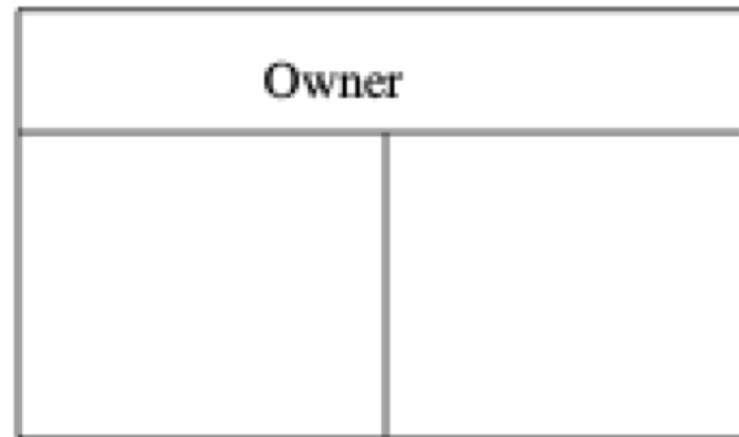
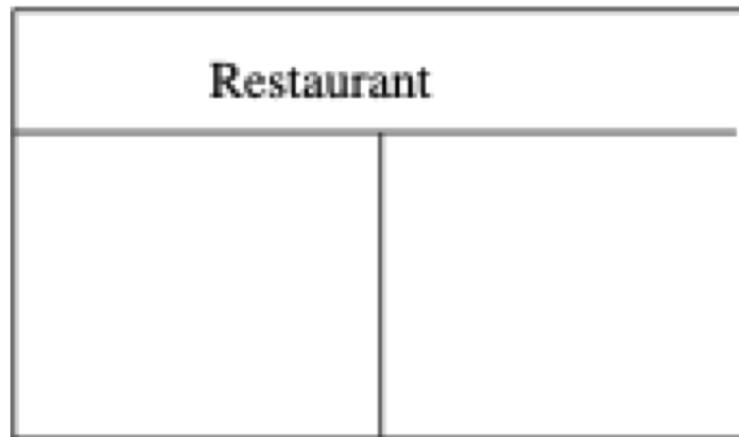
- Let's begin by underlining nouns.
  - *Each restaurant corresponds to a certain price range, neighbourhood, and cuisines it serves. Restaurants that serve alcohol must have a license, which they need to renew every year. The system should also report how long, on average, customers wait for take out in restaurants that offer take-out service. When reviewers leave a review for a restaurant, they must specify a recommendation (Thumbs Up or Thumbs Down) and can also leave a comment. An owner of a restaurant can respond to a review with a comment. All users of the system log in with their username. Users can choose to be contacted by email and ...*

# Choose potential classes

- But which ones are the main players? These are potential classes.
  - *Each restaurant corresponds to a certain price range, neighbourhood, and cuisines it serves. Restaurants that serve alcohol must have a license, which they need to renew every year. The system should also report how long, on average, customers wait for take out in restaurants that offer take-out service. When reviewers leave a review for a restaurant, they must specify a recommendation (Thumbs Up or Thumbs Down) and can also leave a comment. An owner of a restaurant can respond to a review with a comment. All users of the system log in with their username. Users can choose to be contacted by email and ...*

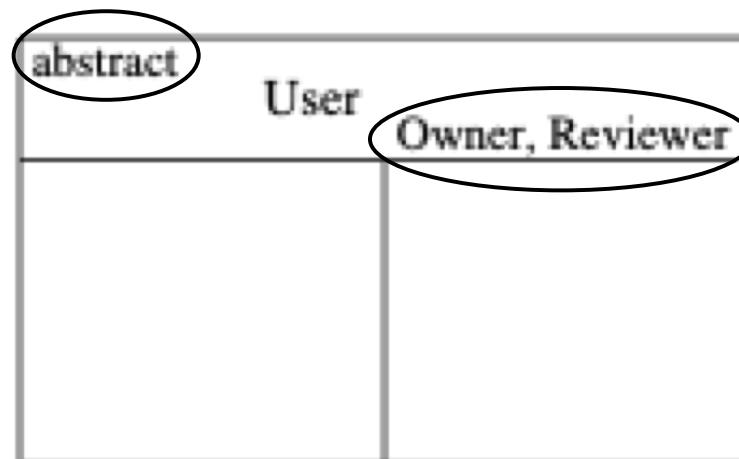
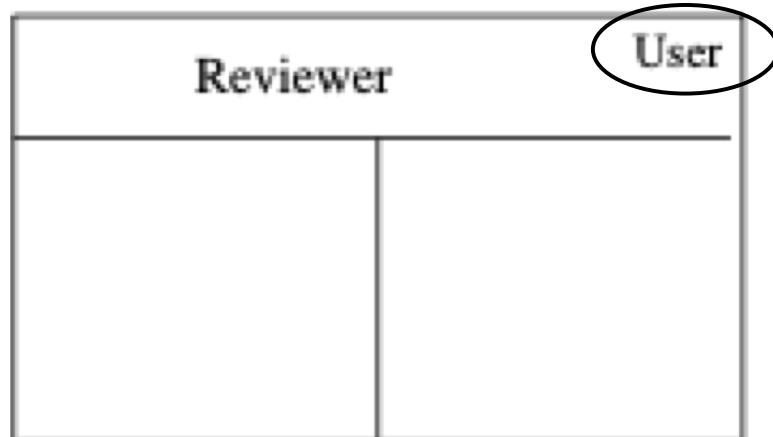
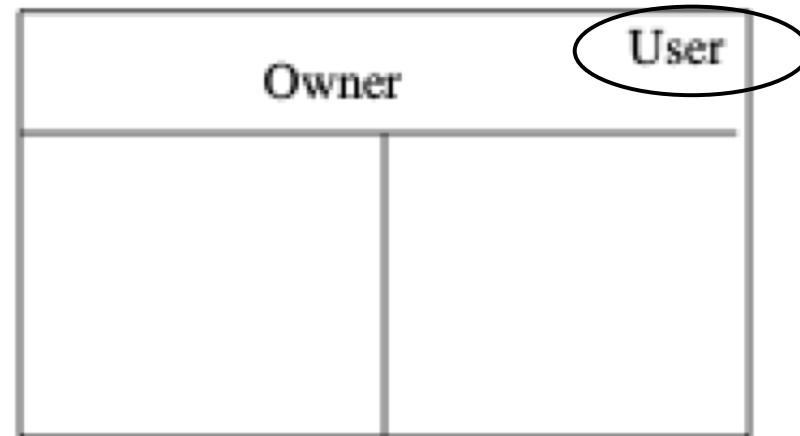
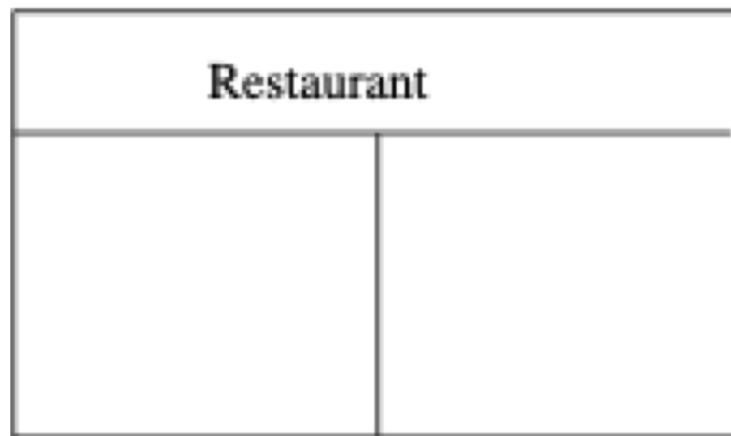
# We can start building the model

- One class goes on each CRC card:



# We can start to identify inheritance

- Adding parent class (upper RHS) and child classes (lower RHS):

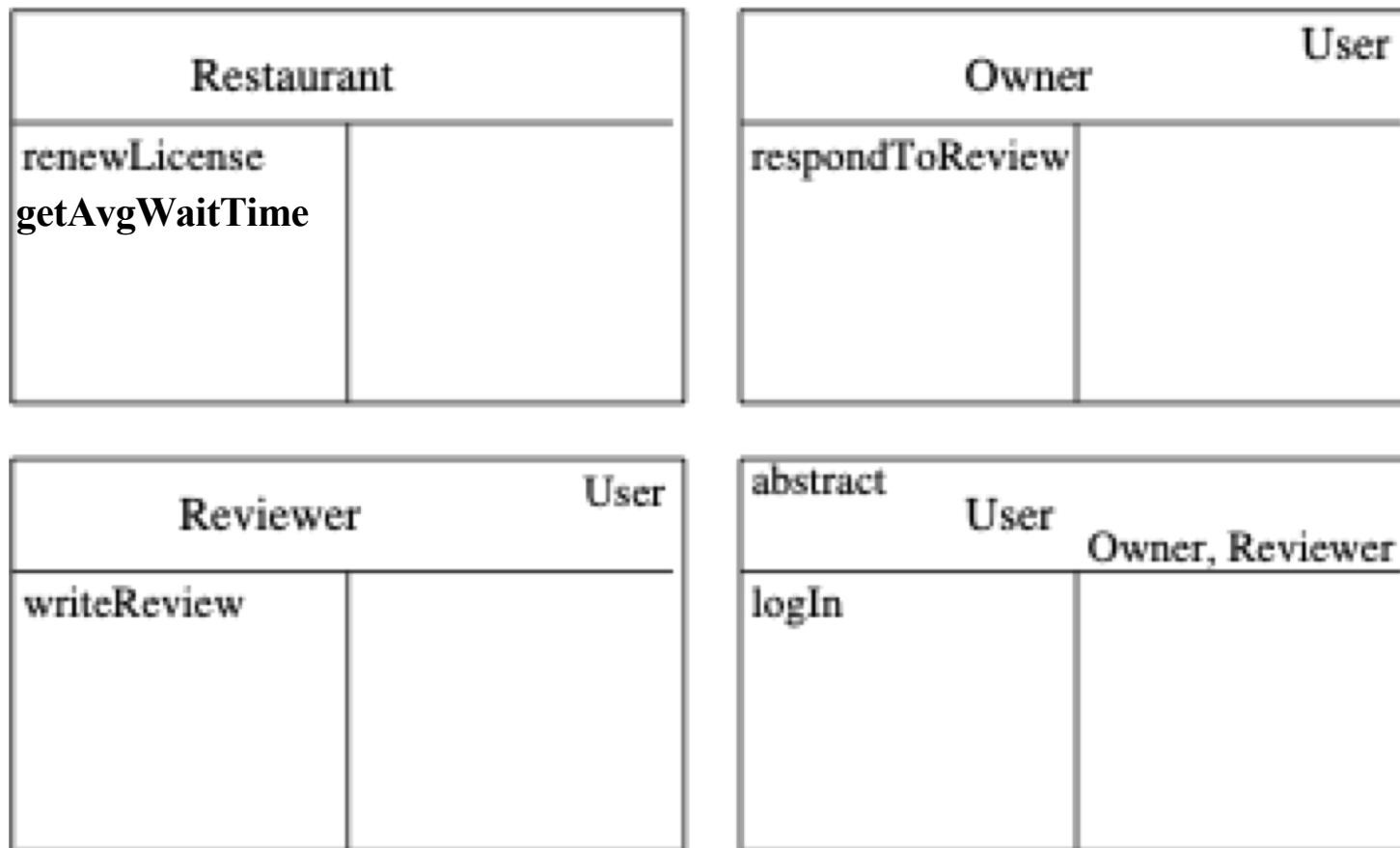


# Identify verb phrases that describe responsibilities.

- And what are the classes responsible for *doing*?
  - *Each restaurant corresponds to a certain price range, neighbourhood, and cuisines it serves. Restaurants that serve alcohol must have a license, which they need to renew every year. The system should also report how long, on average, customers wait for take out in restaurants that offer take-out service. When reviewers leave a review for a restaurant, they must specify a recommendation (Thumbs Up or Thumbs Down) and can also leave a comment. An owner of a restaurant can respond to a review with a comment. All users of the system log in with their username. Users can choose to be contacted by email and ...*

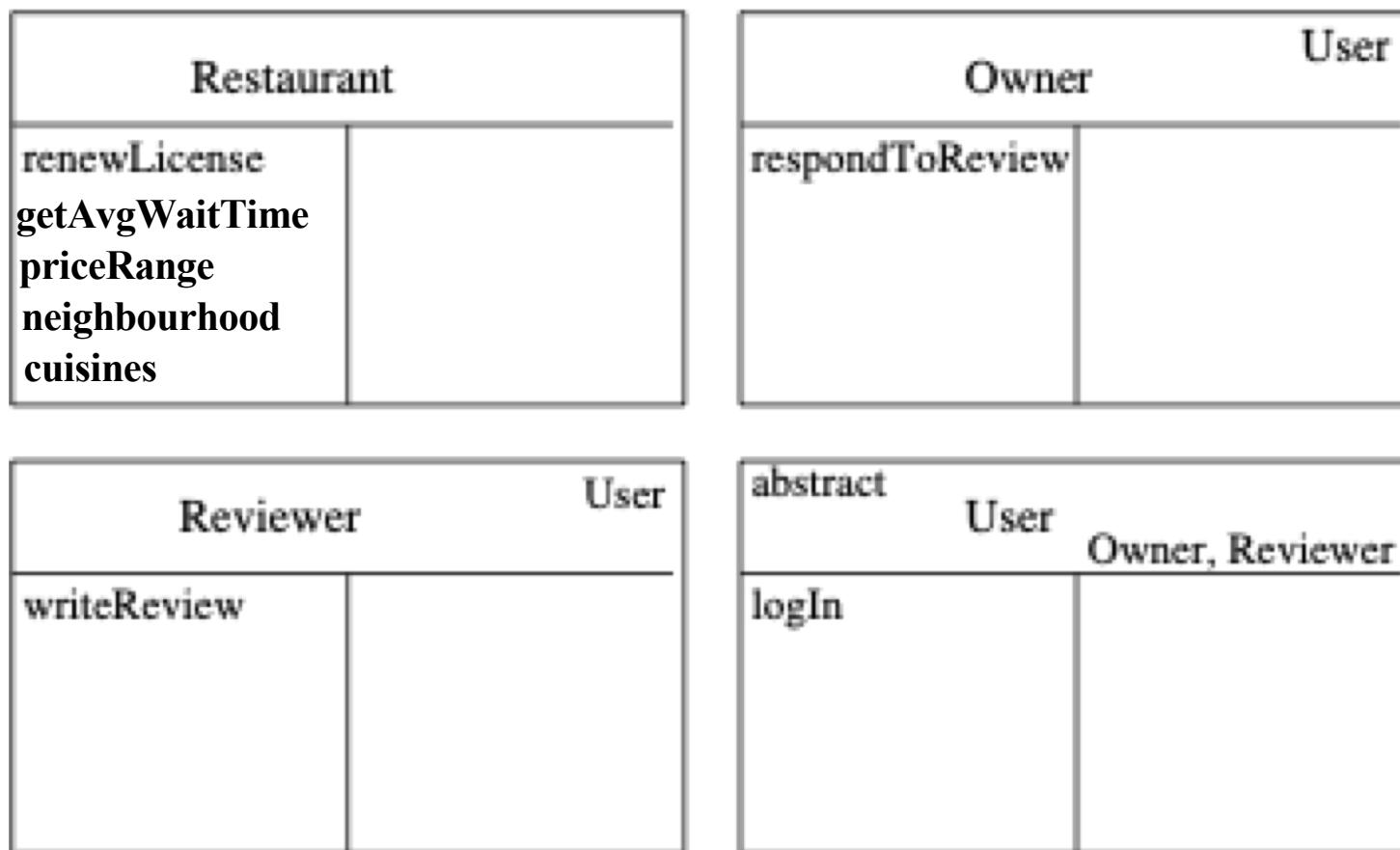
# Example

- Adding some “what they do” responsibilities:



# Example

- Adding some “what they store” responsibilities for Restaurant :

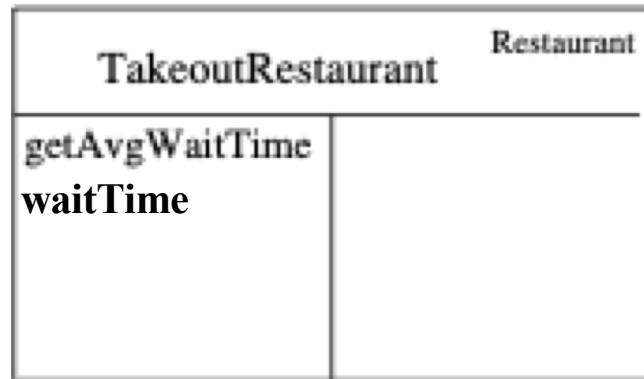
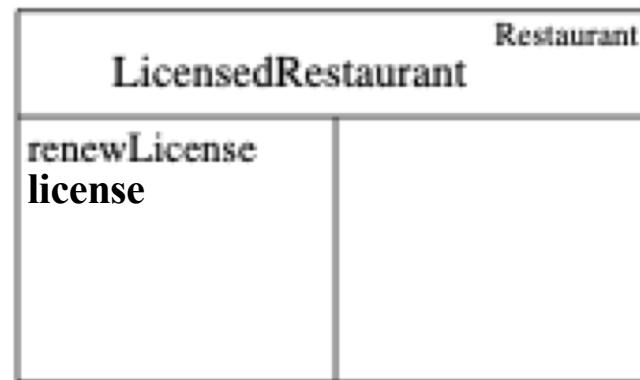
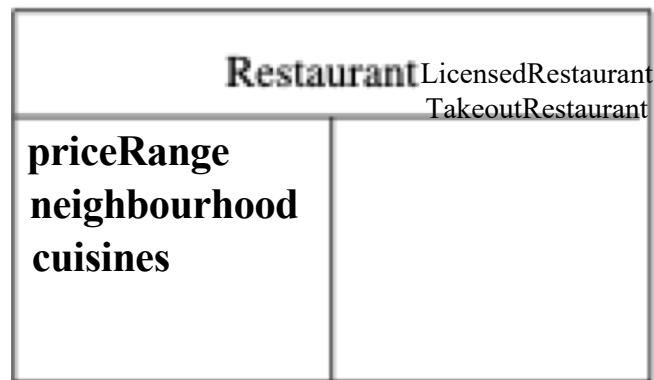


# A problem with class Restaurant

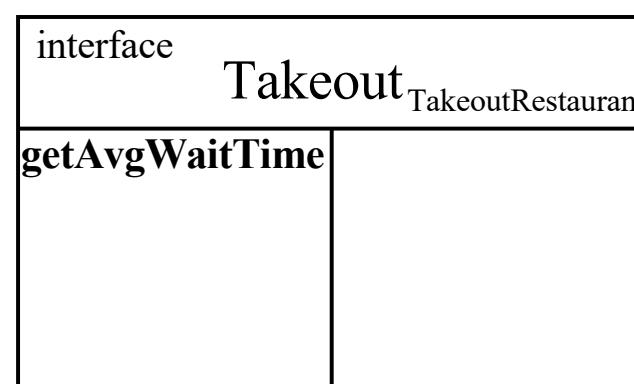
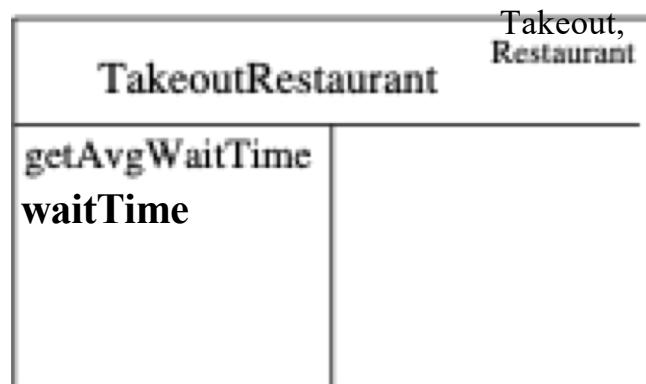
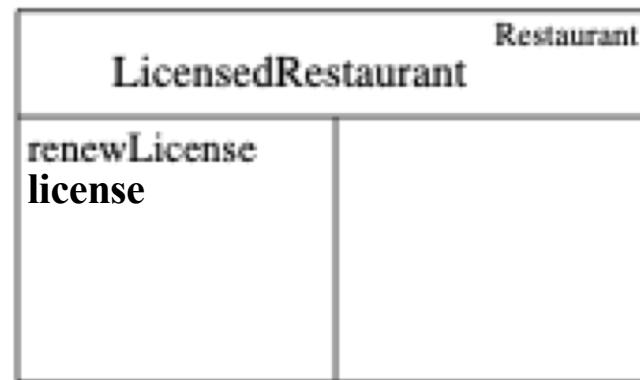
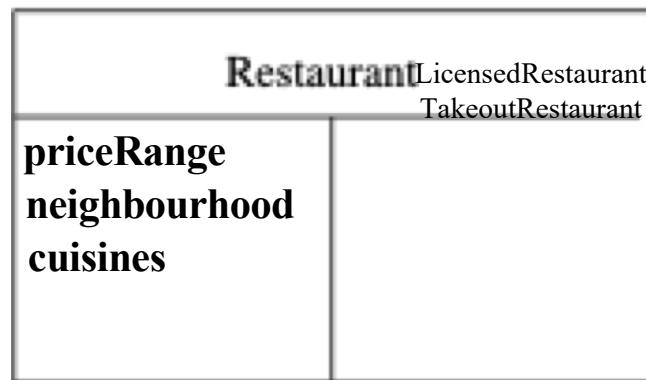
- What about the responsibility of storing licenses? But not all restaurants have licenses!
- Solution: We need a new type of Restaurant. Also, move `renewLicense` responsibility.



- What about the responsibility of storing wait times? Not all Restaurants offer takeout!
- Solution: We need a hierarchy.



- What if a restaurant is both a `LicensedRestaurant` and a `TakeOutRestaurant`?
- Solution: Use an interface.

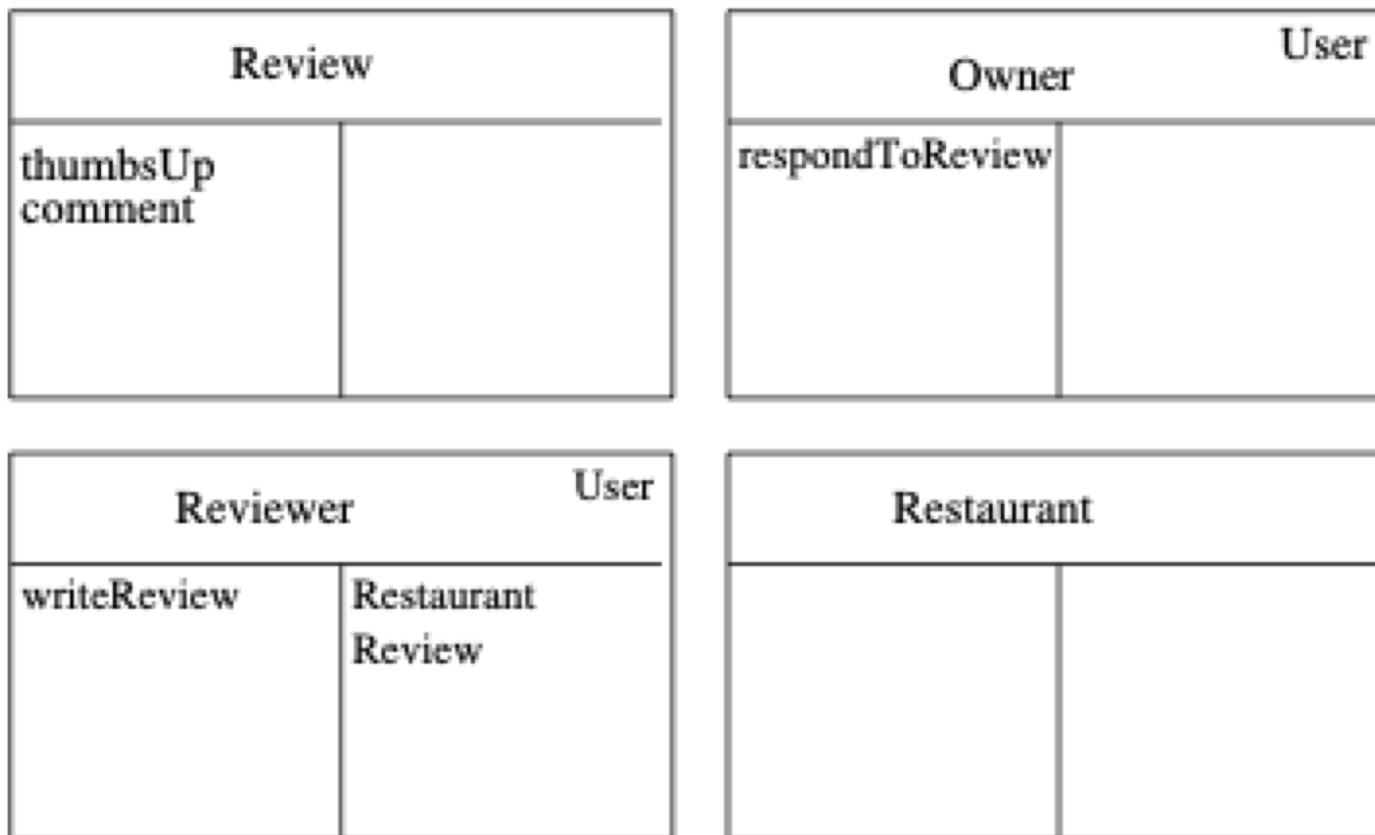


# Expanding the Model

- Let's look more closely at reviews.
  - *Each restaurant corresponds to a certain price range, neighbourhood, and cuisines it serves. Restaurants that serve alcohol must have a license, which they need to renew every year. The system should also **report how long, on average, customers wait for take out** in restaurants that offer take-out service. When reviewers leave a review for a restaurant, they must specify a recommendation (Thumbs Up or Thumbs Down) and can also leave a comment. An owner of a restaurant can **respond to a review** with a comment. All users of the system **log in** with their username. Users can choose to be contacted by email and ...*

# Example

- To write a review... it looks like we need a Review class.

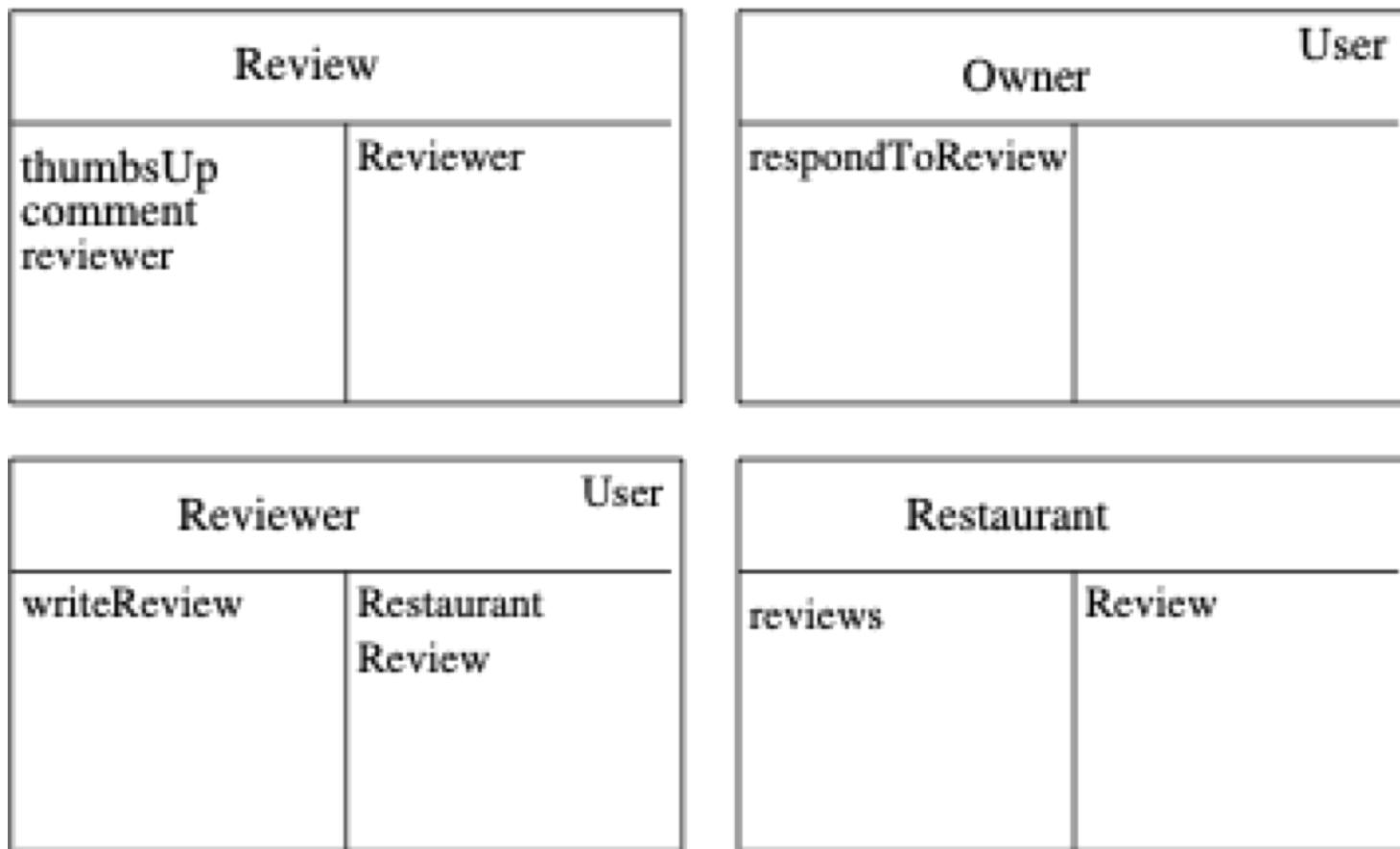


# Example

- We have some design decisions to make:
  - Does a Review know which Restaurant it is for?
  - Does a Review know who wrote it?
  - Where do Reviews live? With a Restaurant? With a Reviewer?

# Example

- Make your decisions. Here is one possibility:

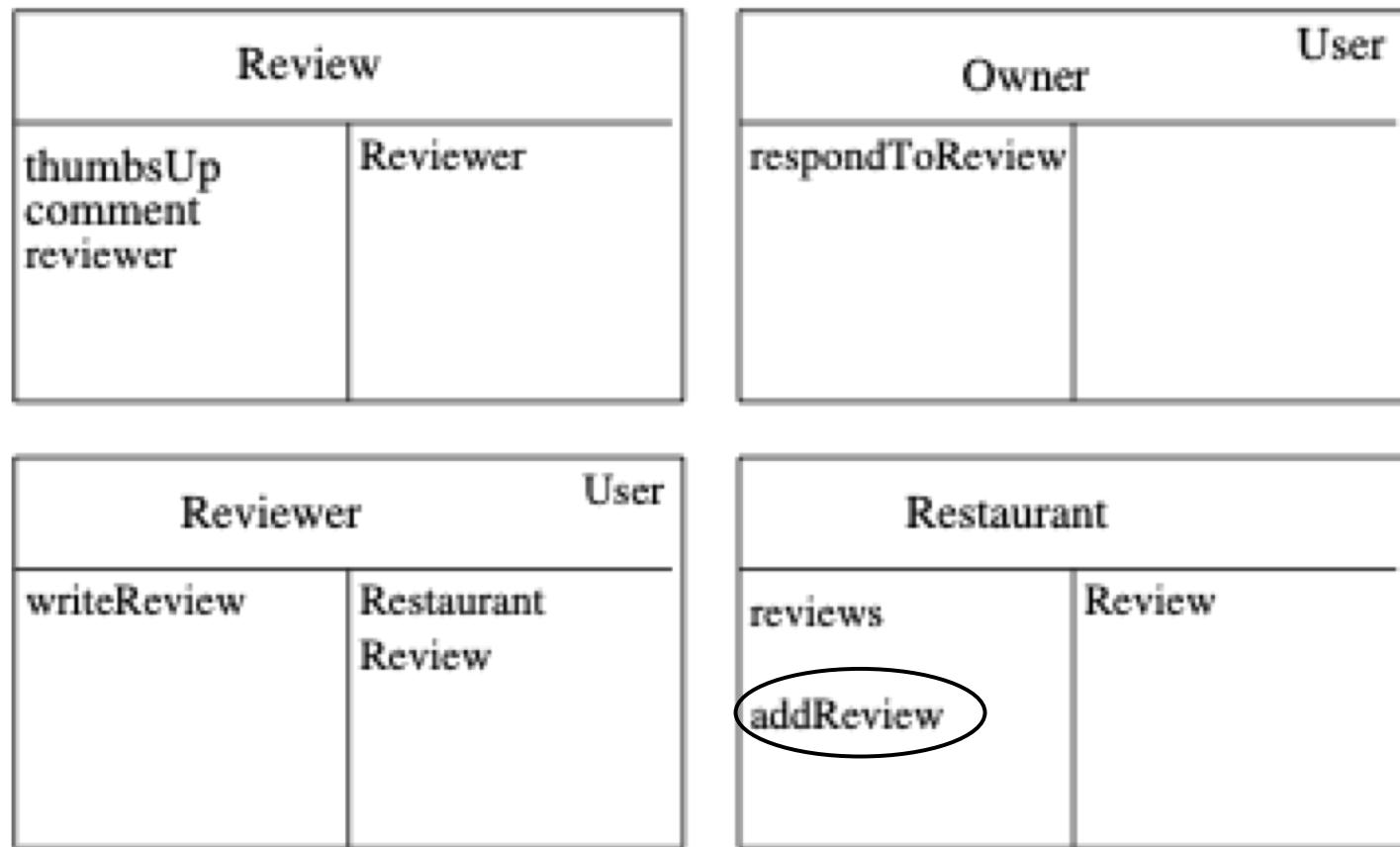


# Example

- Let's see if this works...
- Scenario: *write a review*.
- Scenario walk-through:
  - To write a review, a **Reviewer** needs to:
    - create a **Review**
    - provide it to the **Restaurant**
    - the **Restaurant** needs to add it
  - We are missing the last responsibility...

# Example

- Adding the new responsibility.



# Exercise -- for practice at home

- Continue building the CRC Model by completing a scenario walkthrough for the `respondToReview` scenario.
- Now execute a scenario walk-throughs to convince yourself that your design works.
- Complete the model by adding all functionality in the description.