Developing a CRC Model

CSC207 Fall 2015

Example

Consider this description of a software system:

> *You are developing a software system to facilitate restaurant reviews. Each restaurant is in a certain price range and neighbourhood, and has a cuisine that 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 ...

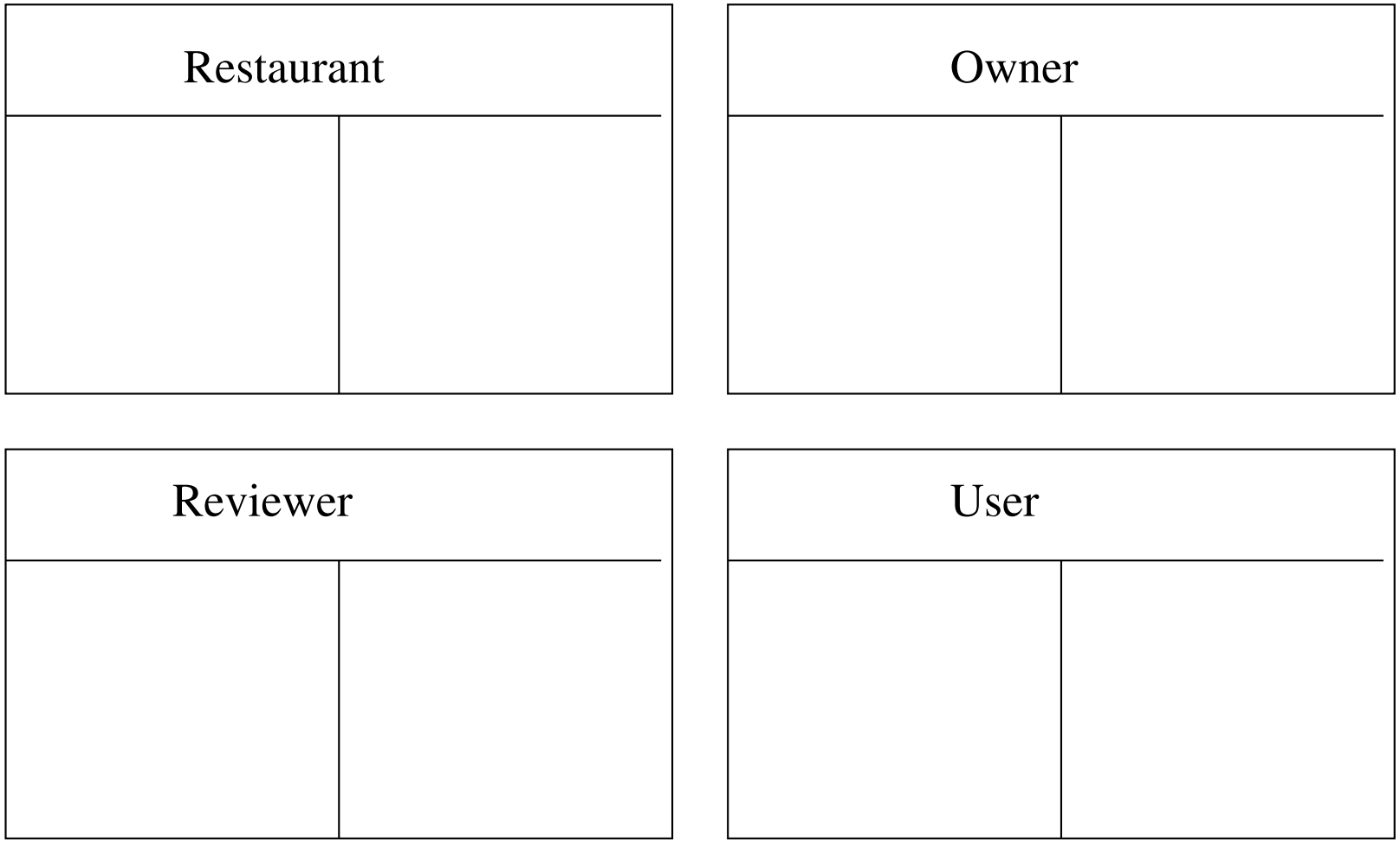
Choose potential classes

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 ...*

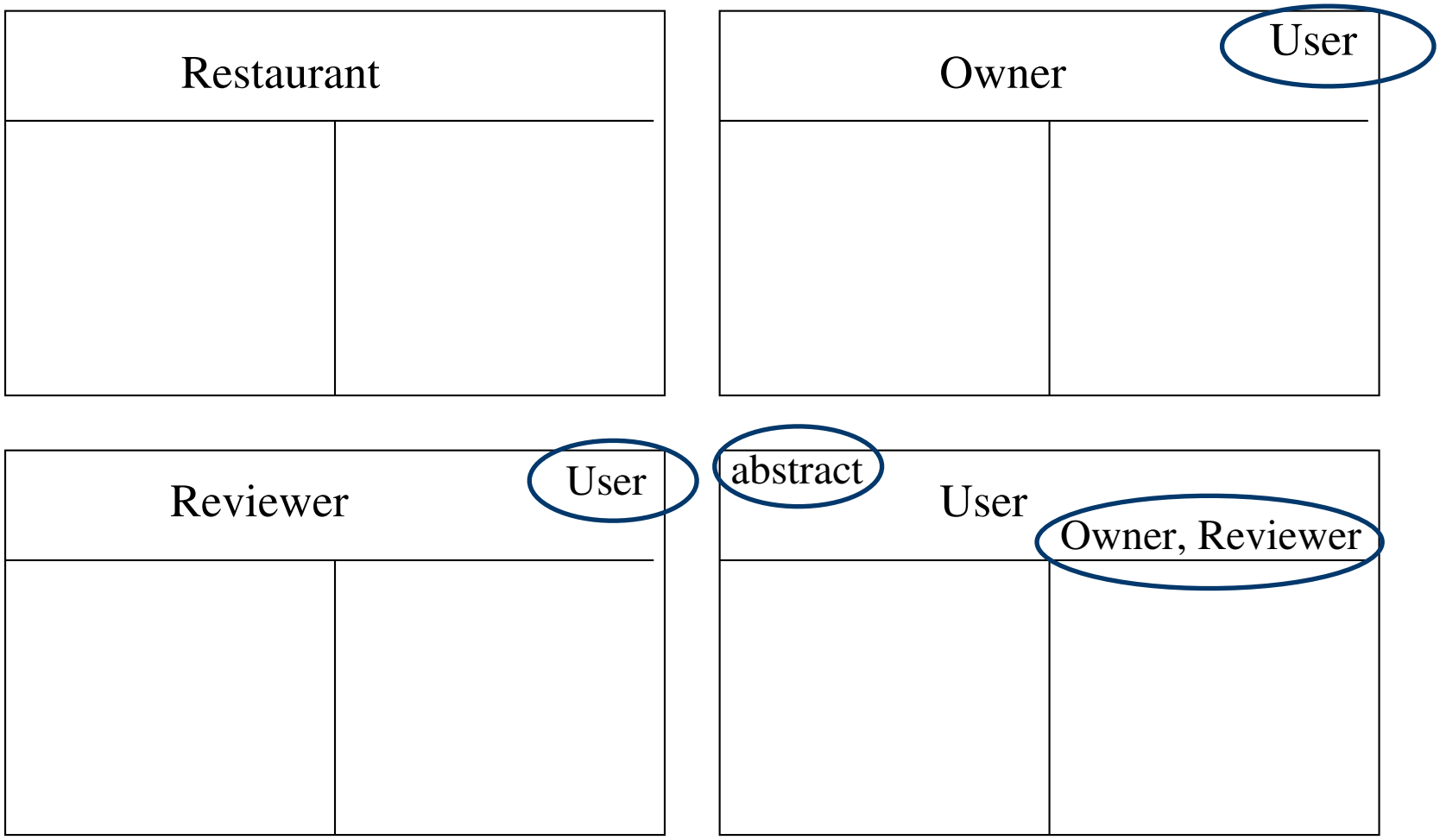
We can start building the model

One class goes on each CRC card:



We can start to identify inheritance

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



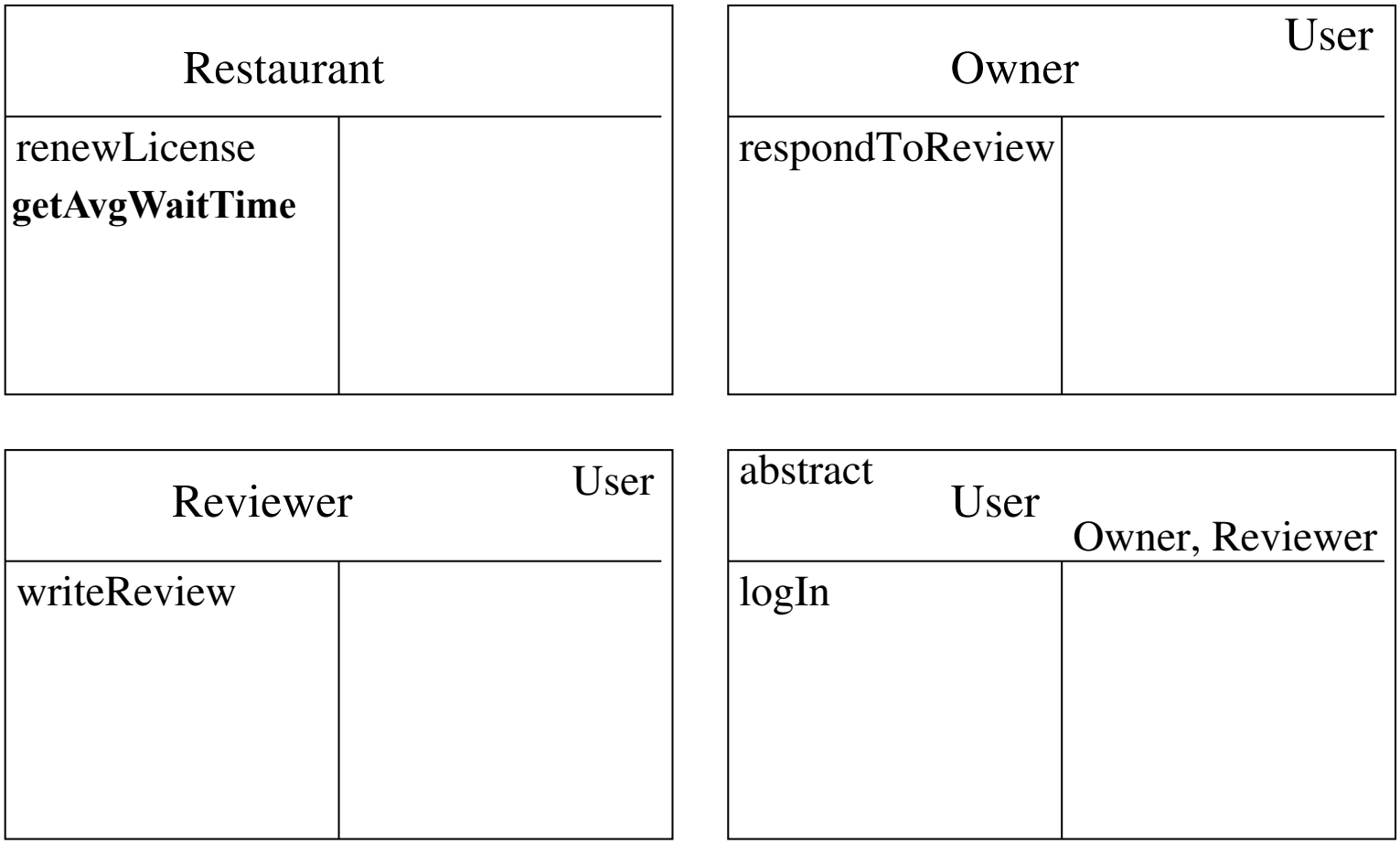
Identify verbs that describe responsibilities.

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 ...*

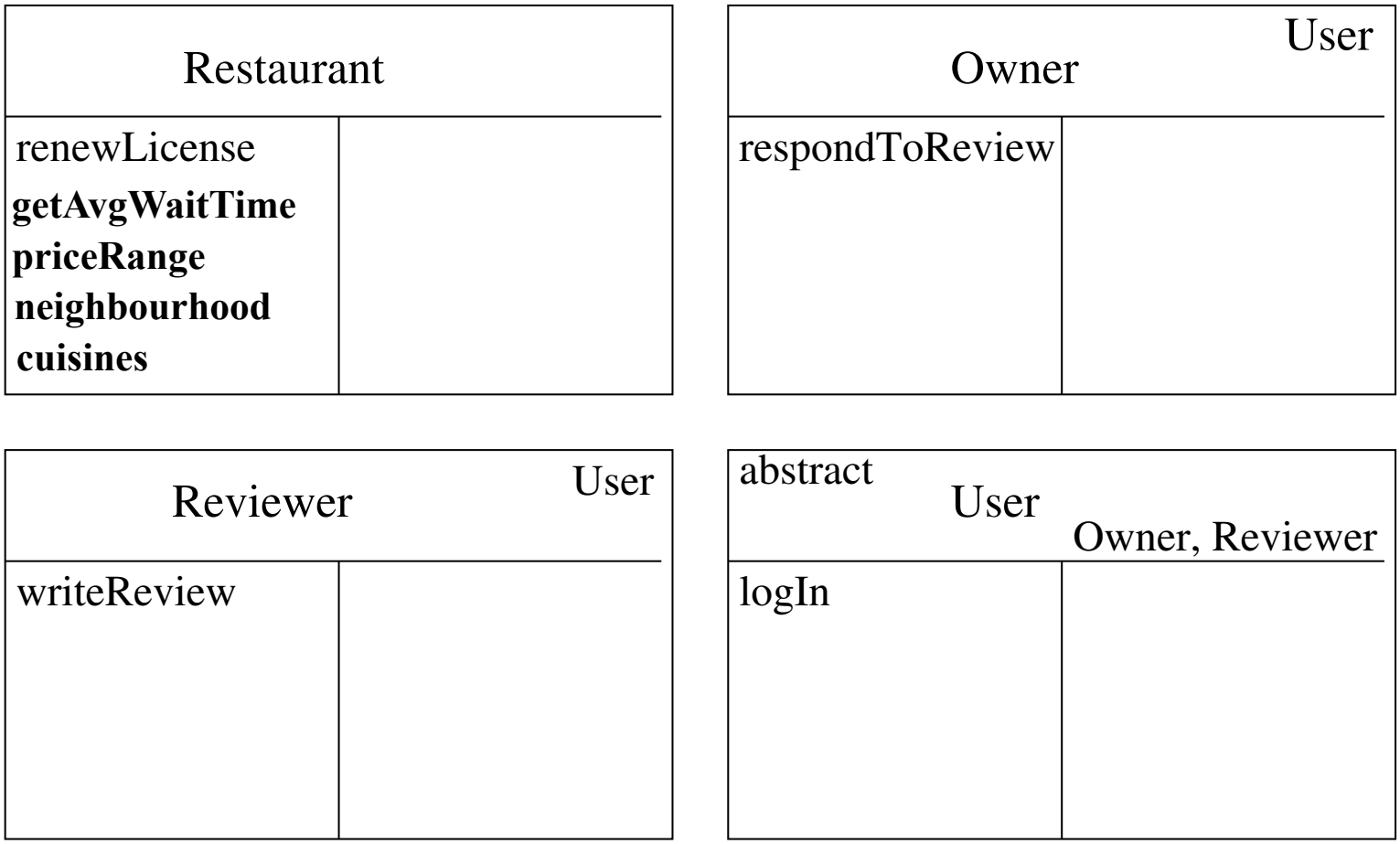
Example

Adding some “what they do” responsibilities:



Example

Adding some "what they store" responsibilities for `Restaurant`:



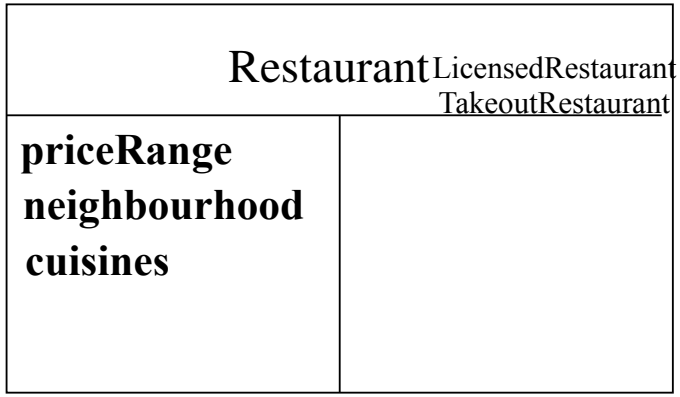
A problem with class `Restaurant`

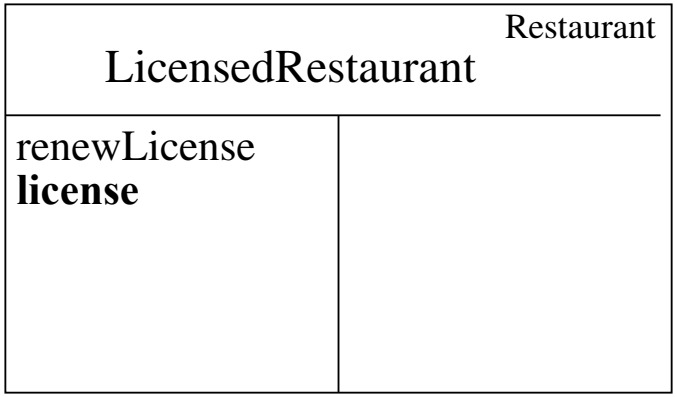
What about the responsibility of storing licenses? But not all restaurants have licenses!

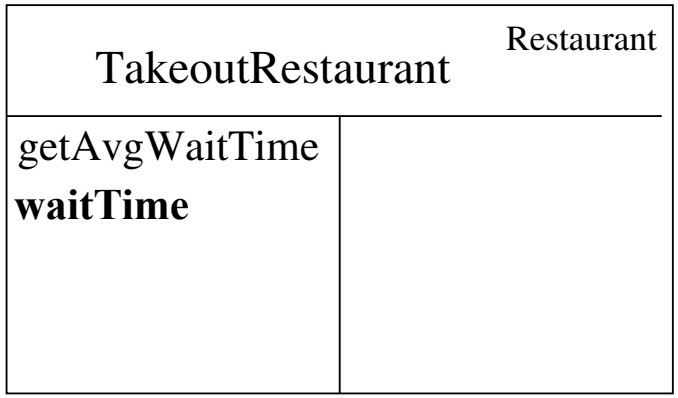
Solution: We need a new type of `Restaurant`. Also, move `renewLicense` responsibility.

What about the responsibility of storing wait times? Not all Restaurants offer takeout!

Solution: We need a hierarchy.

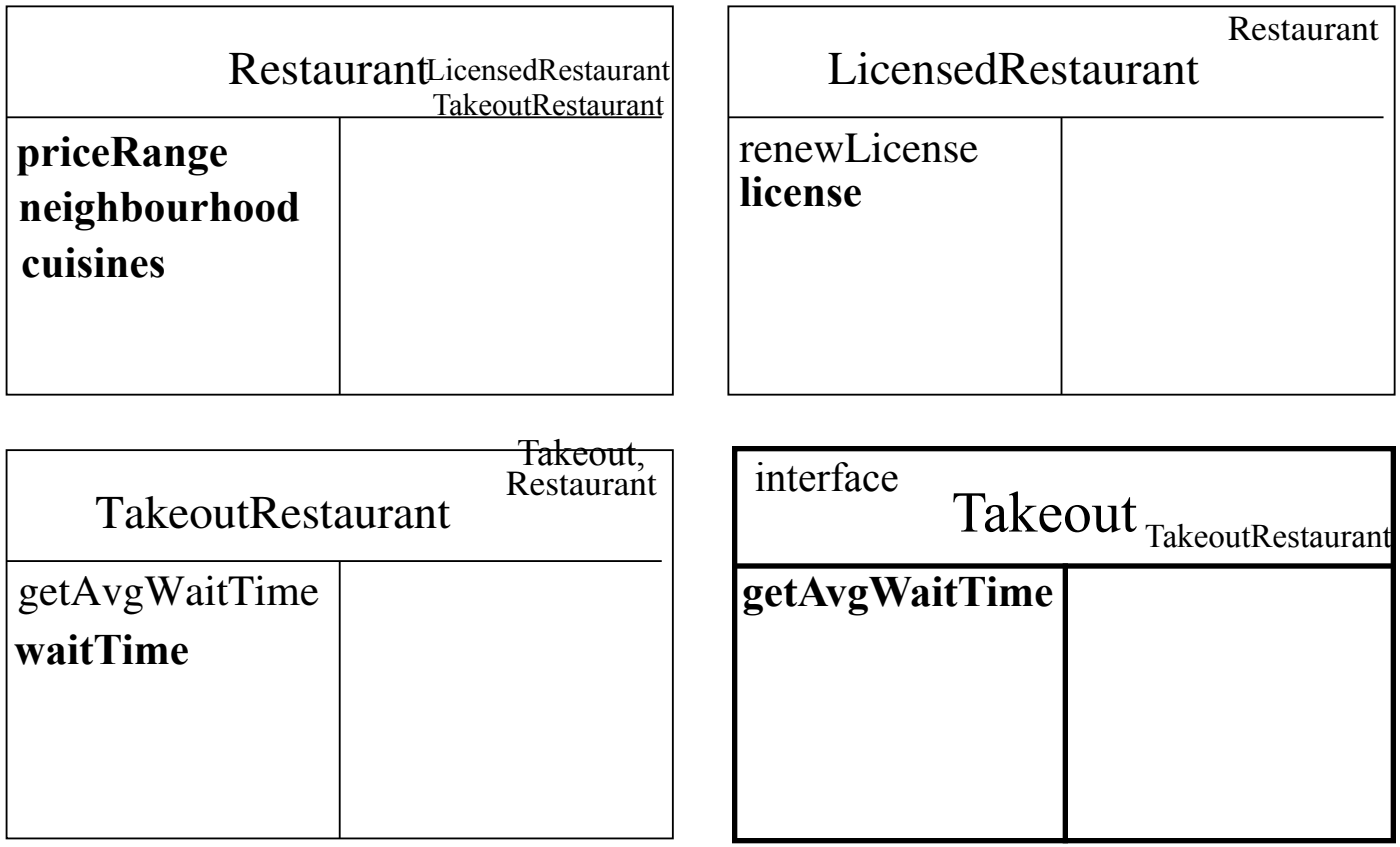






What if a restaurant is both a `LicensedRestaurant` and a `TakeOutRestaurant`?

Solution: Use an interface.



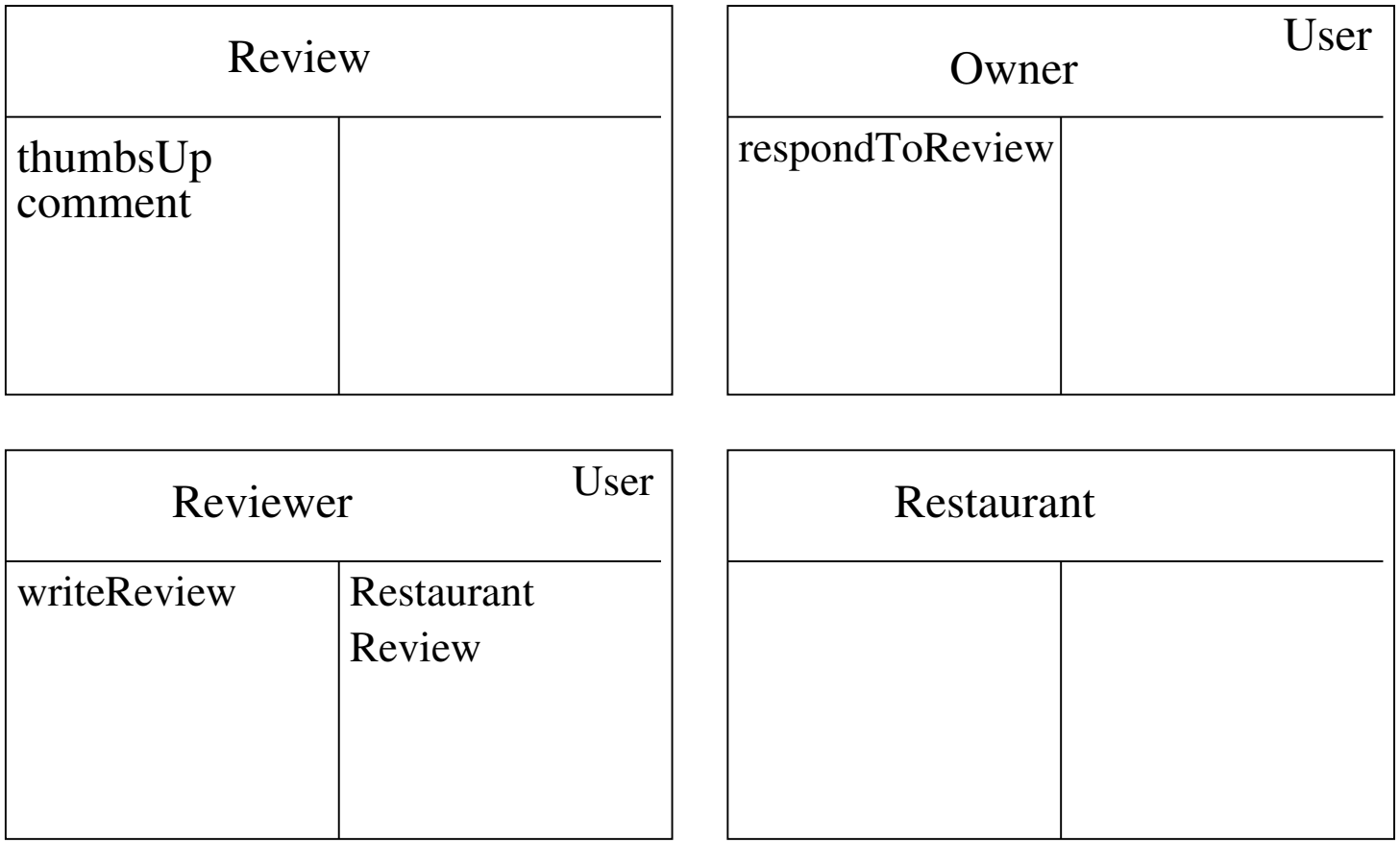
Expanding the Model

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:

Review	
thumbsUp comment reviewer	Reviewer

Owner	User
respondToReview	

Reviewer	User
writeReview	Restaurant Review

Restaurant	
reviews	Review

Example

Let’s see if this works...

Scenario: *write a review*.

Scenario walk-through:

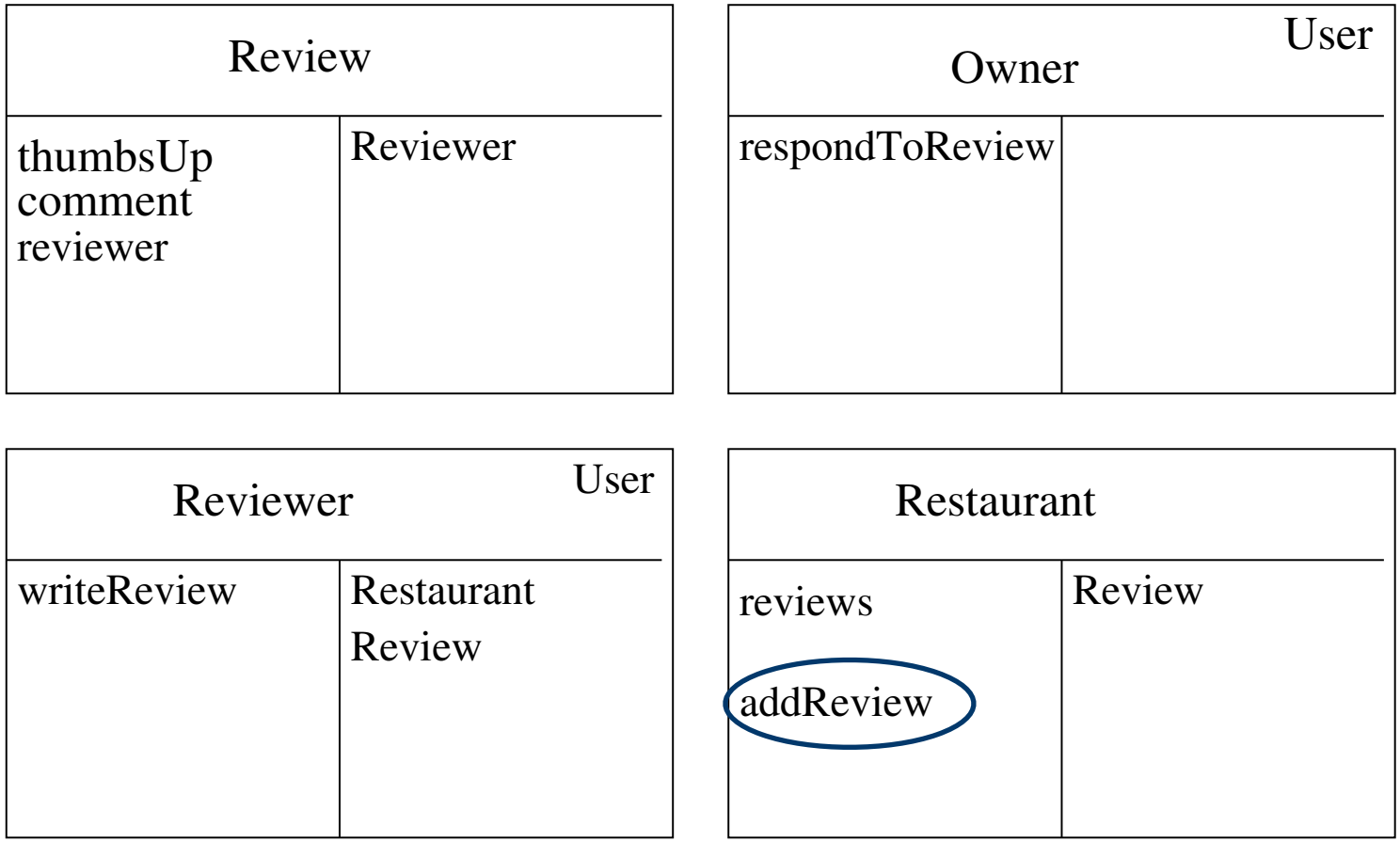
To write a review, a `Reviewer` needs to:

- 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.