Class, Responsibility, Collaboration (CRC) Cards – Ticket Vendor Example

1. Specification:

Create a program that keeps track of ticket sales for an auditorium that has 32 rows of seats with a varied number of seats in each row. Each ticket is associated with a seat (row letter and seat number such as A12), a price (low ~ \$10, medium ~ \$30, or high ~ \$50), whether or not the seat is for sale or complementary (for example, sold to the public or given to friends of a performer), name of occupant (who bought the ticket), their e-mail address, and the date, time, and name of the performance so that tickets for each seat can be sold to different people at different times.

It should be possible to get a list of names of all people who bought tickets for a particular date. For a given performance, it should also be possible to print a grid of x's and o's where an x represents a seat that is occupied (that is, it's associated ticket has been sold) and an o represents seats for which no ticket has yet been sold.

Anyone who buys a high-priced ticket automatically joins the Gold Members Club which is an e-mail list. It should be possible to generate an e-mail list of all people in the Gold Members Club, and to delete someone from that e-mail list upon request.

A ticket holder should be able to create and access an account that stores the seats, dates, and names of performances attached to all of the tickets which they have purchased.

2. Classes:

This is where major design decisions are made. Can we store all of the information in the `UserAccount`s?

No, because we do not want to accidentally sell the same seat to the same show to two different people. So information about each show should be stored separately from the user accounts.

Could we have a `ShowManager` which keeps instances of class `Show` in which we store information associated with each seat and ticket?

Alternatively, could we let seats be entries in a collection or multi-dimensional array that exists within the `Show` class, generating tickets for each by accessing a `TicketManager`?

There are many good designs that can fulfill this specification. When creating CRC cards, you need to choose one design, work with it, and adjust it as problems arise.

3. Responsibilities

More than one class needs access to the seat number and row number of a given ticket. Where should we store that information?

For the purposes of this handout, we decide to have a `Ticket` class. But where do we instantiate `Ticket`? In the `TicketManager` and pass it to the `UserAccount`, or should we instantiate the ticket class in the `UserAccount` and pass that information to the `Show` and its associated `TicketManager`?

Likewise, should we store the e-mail list in the `TicketVendorSystem`, or create a method that generates that list whenever we need it? If the latter, how do we store the information about whether or not a user is on that list?

4. Collaborations

Collaborators can be classes that are adjacent on the inheritance hierarchy (that is, one is the parent of the other), or instantiate one inside the other, or one gets passed as an argument into a method in the other.

Collaboration is symmetric. So, if class `A` collaborates with class `B`, then class `B` collaborates with class `A`. For example, the `TicketManager` will collaborate with `Ticket` objects (through instantiation), so the `TicketManager` collaborates with class `Ticket` (through creating and storing instances of `Ticket`).

On the next page is one possible set of CRC cards that describes one piece of software that fulfills the given specification. Class names are at the top of each card. The responsibilities (variables and methods) of that class are written on the left, while collaborating classes are listed on the right.

TicketVendorSystem	
- store collection of user accounts - store ShowManager - generate lists of show names, times, with unsold seats - generate email list for club members only - generate and sell ticket	- UserAccount - ShowManager

UserAccount	
- store, getter, and setter for name, address, email - store, getter, and setter IsMemberOfClub - getter for collection of tickets bought - create list of shows, list of days/times/seats - request deletion from email list - view and buy tickets	- TicketVendorSystem - ShowManager

ShowManager	
- store collection of shows - getter for unsold tickets from a given show - request ticket creation - getters for ticket (by show, by seat + show, by user + date) - getter for ticket collection by user	- TicketVendorSystem - Show - Ticket - UserAccount

Show	
- store TicketManager - pass ticket creation request to TicketManager - create collection of emails for all tickets of a given price - toString for map of sold and unsold seats - getter for tickets in TM - getter for ticket by owner	- Ticket - ShowManager - TicketManager

TicketManager	
- store and getter for a representation of all seats and associated tickets - create ticket for seat - setter to store ticket in representation of seats - generate ticket for chosen seat - store collection of tickets - getters for all emails, names, etc in collection of tickets	- Ticket - Show

Ticket	
- store name, date, time of show - store seat number/letter price - store name, address, email of buyer - getters for everything - constructor sets show name of show (setters for everything else) - getter/setter/store “IsComp”	- Show - TicketManager - ShowManager

Warm-Up Questions:

1. According to the CRC cards, where is the user's name stored? User Account.

2. How does the system generate a list of users names for a given show? Ticket Manager.

3. How does the system generate the map of seats with x's and o's showing which seats are available? Can you think of another way to accomplish this?

Show. Ticket Manager.

Questions About The Current Design:

4. In which class does execution start? User Account request to view tickets

5. Where does class `Show` get instantiated? by Show Manager

6. Where are instances of class `Ticket` stored? Ticket Manager

7. How does a `UserAccount` get information about available seats for a particular show?

Ticket Vendor System.

8. How does a user buy a ticket (represented by a `Ticket`)?

User Account → Ticket Vendor → list of unsold Ticket → Ticket store Buyer info

9. Once a `Ticket` is generated, where does it go? How many classes have direct or indirect access to the information contained in that `Ticket`?

Ticket Manager. Direct: bottom 3; Indirect: Top 2

10. Should `UserAccount` collaborate with class `Show`? Why or why not?

No. should not be able to modify show directly. can just check Ticket.

Questions About Changing the Design:

11. Do we need a separate class for `Show` or can we reorganize the `ShowManager` so we no longer need instances of `Show`? If so, should we? Yes. Yes. ∵ multiple show possible

12. How can we change the design so that the same `TicketVendorSystem` sells tickets to multiple venues (different rooms with different numbers of rows and columns of seats)?

Venue Manager + Venue class.

↑

attach Show Manager.