Designing applications

Adapted from slide by David Barnes and Michael Kolling

Main concepts to be covered

- Discovering classes
- CRC cards
- Designing interfaces
- Patterns

The verb/noun method

- The nouns in a description refer to ‘things’.
  - A source of classes and objects.
- The verbs refer to actions.
  - A source of interactions between objects.
  - Actions are behavior, and hence methods.

A problem description

The cinema booking system should store seat bookings for multiple theaters.
Each theater has seats arranged in rows.
Customers can reserve seats and are given a row number and seat number.
They may request bookings of several adjoining seats.
Each booking is for a particular show (i.e., the screening of a given movie at a certain time).
Shows are at an assigned date and time, and scheduled in a theater where they are screened.
The system stores the customer’s phone number.

Nouns and verbs

Cinema booking system
Stores (seat bookings)
Stores (phone number)

Theater
Has (seats)

Movie

Customer
Reserves (seats)
Is given (row number, seat number)
Requests (seat booking)

Time

Date

Seat booking

Show
Is scheduled (in theater)

Seat

Seat number

Telephone number

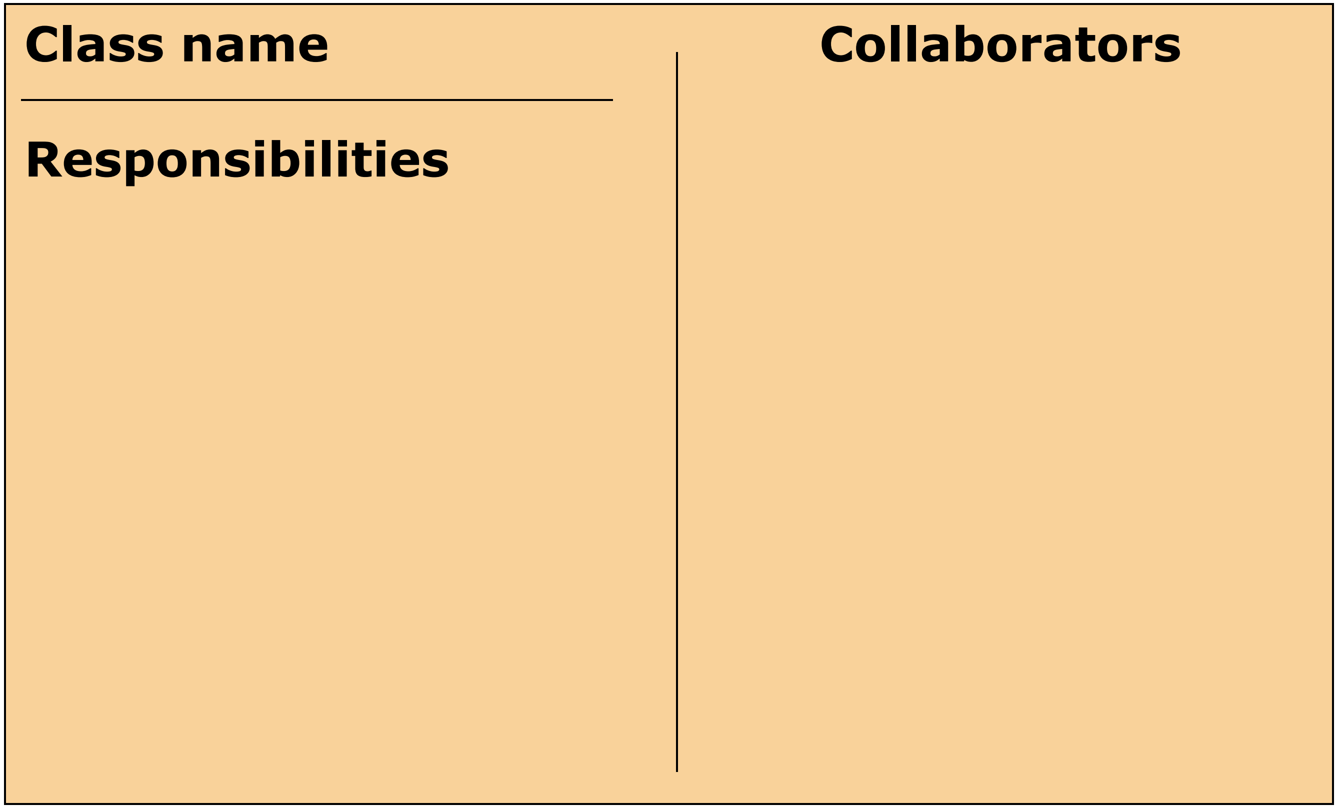
Row

Row number

Using CRC cards

- First described by Kent Beck and Ward Cunningham.
- Each index cards records:
  - A *class* name.
  - The class’ s *responsibilities*.
  - The class’ s *collaborators*.

A CRC card



Class name

Responsibilities

Collaborators

Scenarios

- An activity that the system has to carry out or support.
  - Sometimes known as *use cases*.
- Used to discover and record object interactions (collaborations).
- Can be performed as a group activity.

Scenario 1

- A customer calls the cinema and wants to make a reservation for two seats tonight to watch the classic move *Star Wars: Episode IV A New Hope*. The cinema employee starts using the booking system to find and reserve a seat.
- This is the set up, now we play out the scenario playing the rolls of the different classes.

Scenario cont.

- The cinema employee wants to find all showings of *Star Wars IV* that are on tonight.
  - So we can note on the `CinemaBookingSystem` CRC card, as a responsibility: *Can find shows by title and day*.
  - We can also record class Show as a collaborator.
- We have to ask ourselves: How does the system find the show? Who does it ask?
  - One solution might be that the `CinemaBookingSystem` stores a collection of shows.
  - This gives us an additional class: the collection (might be an ArrayList or some other data structure).

Scenario cont.

- Assume that three shows come up: one at 5:30 p.m., one at 9 p.m., and one at 11:30 p.m. The employee informs the customer of the times, and then chooses the one at 9 p.m.
  - The employee checks the details of that show (if it is sold out, which theater it runs in, etc.).
  - `CinemaBookingSystem`: *Retrieves and displays show details*, and on the `Show` card you write: Provides details about theater and number of free seats.
- Assume there are plenty of free seats. The customer chooses seats 13 and 14 in row 12.
  - The employee makes that reservation.
  - Note on the `CinemaBookingSystem` card: *Accepts seat reservations from user*.

Scenario cont.

- Now we need to play through exactly how the seat reservation works.
  - A seat reservation is clearly attached to a particular show, so the `CinemaBookingSystem` should probably tell the show about the reservation;
  - it delegates the actual task of making the reservation to the `Show` object. `Show` class: *Can reserve seats*.
- What exactly does the `Show` class do with the request to reserve a seat.
  - Assume it has a link to a `Theater` object and the `Theater` should probably know about the exact number and arrangement of seats it has.
  - Note that each `Show` should have its own instance of the `Theater` object because several shows can be scheduled for the same `Theater.`

Scenario cont.

- Now the theater has accepted a request to make a reservation (Note this on the card: *Accepts reservations request)*. How does it deal with it?
  - The theater could have a collection of seats in it or it could have a collections of rows (each row being a separate object) and rows hold seats.
  - Which alternative is better?
  - A collection of rows might make it easier to find adjacent seats
  - `Theater` card: *Stores Rows*. Rows is a collaborator.
- Note on the `Row` class: *Stores collection of seats.*

Scenario cont.

- Getting back to the `Theater` class, we have not yet worked out exactly how it should react to the seat reservation request.
  - Let us assume it finds the requested row and then makes a reservation request with the seat number to the `Row` object.
- Next, we note on the `Row` card: *Accepts reservations request for seat*. It must then find the right `Seat` oject (note this as a responsibility).
  - Tells the `Seat` object that it is now reserved.
- `Seat` card: *Accepts reservations.* The seat itself can remember whether it has been reserved.
  - `Seat` card: *Stores reservation status* (free/reserved)
  - What about information of the person making the reservation?

A partial example

CinemaBookingSystem	**Collaborators**
Can find shows by title and day.	**`Show`**
Stores collection of shows.	**`Collection`**
Retrieves and displays show details.	
...	

Find One or Two partners

- In your group pick one of the following scenarios and play through in your group. Be sure to mark responsibilities and collaborators.
  - A customer requests five seats together. Work out exactly how five adjoining seats are found.
  - A customer calls and says he forgot the seat numbers he was given for the reservation he made yesterday. Could you please look up the seat numbers again?
  - A customer calls to cancel a reservation. He can give his name and show but has forgotten the seat numbers.
  - A customer calls who already has a reservation. She wants to know whether she can reserve another seat next to the ones she already has.

Scenarios as analysis

- Scenarios serve to check the problem description is clear and complete.
- Sufficient time should be taken over the analysis.
- The analysis will lead into design.
  - Spotting errors or omissions here will save considerable wasted effort later.

Class design

- Scenario analysis helps to clarify application structure.
  - Each card maps to a class.
  - Collaborations reveal class cooperation/object interaction.
- Responsibilities reveal public methods.
  - And sometimes fields; e.g. “Stores collection ...”

Designing class interfaces

- Replay the scenarios in terms of method calls, parameters and return values.
- Note down the resulting signatures.
- Create outline classes with public-method stubs.
- Careful design is a key to successful implementation.

Documentation

- Write class comments.
- Write method comments.
- Describe the overall purpose of each.
- Documenting now ensures that:
  - The focus is on *what* rather than *how*.
  - That it doesn’t get forgotten!

Cooperation

- Team-working is likely to be the norm not the exception.
- Documentation is essential for team working.
- Clean O-O design, with loosely-coupled components, also supports cooperation.

Prototyping

- Supports early investigation of a system.
  - Early problem identification.
- Incomplete components can be simulated.
  - E.g. always returning a fixed result.
  - Avoid random behavior which is difficult to reproduce.