Lecture 39: Object-Oriented Design

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki

What are objects?

- Objects have
  - State/Properties — represented by instance variables
  - Behavior — represented by methods
    - accessor and mutator methods

Calculator

- Calculator class: User interface
  - including buttons and display
  - No real methods — construct & associate listeners
- State class: Current state of computation
  - Methods invoked by listeners
  - Communicate results to user interface
- Listener classes: Communicate from interface to state

Model-View-Controller

State

- Instance variables:
  - partialNumber, numberInProgress?, numStack, calcDisplay
- Methods:
  - addDigit(int Value)
  - doOp(char op)
  - enter, clear, pop
**Model-View-Controller**

- Dissociate user interface with the “model”
  - “model” represents actual computation
  - May have multiple alternate user interfaces
    - Mobile vs laptop versions of UI
- Model should be unaffected by change in UI.
- In Java UI generally served by “event thread”
  - If tie up event-thread with computation then user interface stops being responsive.

**Designing Programs**

- Identify the objects to be modeled
- List properties and behaviors of each object
  - Model properties with instance variables
  - Model behavior with methods (write spec)
- Refine by filling in the details
  - Hold off committing to details of representation as long as possible.

**Implementation**

- Write in small pieces. Test thoroughly before moving on.
- Solve simpler problem first — use “stubs” if necessary.
- Refactor as code becomes more complex.

**Principles of OO Design**

- Class should have a single responsibility
- Methods should have a single responsibility
- Program to an interface, not an implementation
- Prefer composition to inheritance
Let’s Make an OO Design

- Write a system to help new business: Ryde!
  - Dispatch autonomous “taxis” and “shuttles” to give passengers rides.
- Handle all interactions:
  - Take request, dispatch vehicle, pick up passenger, deposit passenger at destination

Objects/Classes

- Company
  - operates taxis/shuttles
  - receives calls
  - schedules vehicles
- Taxi
  - Transports one passenger
- Shuttle
  - Transports one or more passengers
- Vehicle
  - Picks up passenger
  - Arrives at pickup location
  - notifies company of arrival
  - notifies company of drop-off
- Passenger
  - Requests ride
  - Enters vehicle
  - Exits vehicle
- Location

Vehicle Class

- Properties
  - Company
  - CurrentLocation
  - TargetLocation
- Constructor needs
  - company, location

Methods

- notify company at arrival
- notify company at destination
- set pickup location
- pickup passenger
- unload passenger
- isFree
- getCurrentLocation
- setCurrentLocation
- getTargetLocation
- setTargetLocation
- clearTargetLocation

Company Class

- Properties
  - Collection of taxis & shuttles
  - Trips to be scheduled
- Constructor needs
  - fleet of vehicles

Methods

- Receive trip request
- Dispatch taxi
- Dispatch shuttle
- getCurrentLocation(vehiclen)
More to be specified ...

- Company
  - operates taxis/shuttles
  - receives calls
  - schedules vehicles

- Taxi
  - Transports one passenger

- Shuttle
  - Transports one or more passengers

- Vehicle
  - Picks up passenger
  - Arrives at pickup location
  - notifies company of arrival
  - notifies company of drop-off

- Passenger
  - Requests ride
  - Enters vehicle
  - Exits vehicle
  - Location

After Specifying

- Write in small steps
- Test each method thoroughly using JUnit or other testing mechanisms
- Don't be afraid to refactor as new issues arise.
  - Often better to start over then continue with flawed design.

Readings on Object-Oriented Design

- Practical Object-Oriented Design in Ruby: An Agile Primer by Sandi Metz, 2013
- Design Patterns: Elements of Reusable Object-Oriented Software by “Gang of Four”, 1994