MVC overview

- Model, View, Controller is MVC
  - Model stores and updates state of application
    - Example: calculator, what's the state of a GUI-calculator?
  - When model changes it notifies its views
    - Example: pressing a button on calculator, what happens?
  - The controller interprets commands, forwards them appropriately to model (usually not to view)
    - Example: code for calculator that reacts to button presses
  - Controller isn't always a separate class, often part of GUI-based view in M/VC
- MVC is a fundamental design pattern: solution to a problem at a general level, not specific code per se

MVC in in the j-puzzle suite

- Sliding puzzle game allows users to move pieces near a blank piece to recreate an original image
  - See PuzzleGui, PuzzleController, PuzzleView, PuzzleModel, PuzzleApplet, PuzzleMove
- The model “knows” the location of the pieces
  - Determines if move is legal
  - Makes a move (records it) and updates views
  - Supports move undo
- View shows a board and information, e.g., undo possible
  - See PuzzleView interface, implemented by application and applet

Puzzle MVC: Controller perspective

- In this example, the PuzzleController is a middleman, all communication between views and model via controller
  - Sometimes a middleman class isn’t a good idea, extra layer of code that might not be needed
  - Often in MVC model communicates with multiple views, but communication to model via controller
- In this example one controller holds all the views and executes commands in all views on behalf of model
  - Model only calls showBoard in controller
  - Some of the “intelligence” is in controller, arguably should be in model
- Controller is a candidate for refactoring

Controller and Commands

- Use-case for making a move:
  - Move generated, e.g., by button-click
  - Move forwarded to model (by controller)
    - If move is made, then undo state set to true in views
    - If move is made, then views update board display
- Use-case for undoing a move
  - Undo generated, e.g., by button or menu choice
  - Undo forwarded to model (by controller)
    - If undo changes board, views will be updated
    - If future undo not possible, undo state set to false
- ShowBoard and Undo are both Command classes
  - Command implemented using hook/template method
Hook method and Template Pattern

- Client code (Controller) calls a command’s execute method, all Commands have such a method
  - Execute always has a view parameter
  - Execute has optional other parameter for information
- The execute method is the same in every Command, forwards to the hook method
  - Subclasses of ViewCommand implement hook in application specific way
    - Showing board calls appropriate method in view
    - Undo calls appropriate method in view

Lower level JButton particulars

- PuzzleGui class has panel/grid of buttons for display
  - Pressing button causes action (via controller)
  - Button is displayable, but doesn’t have label
    - If button had label, it would be automatically shown
  - Instead, use setActionCommand to store command
    - Retrieved by getActionCommand, but not a label
    - Button has icon, automatically displayed
- The Icon interface (swing) implemented by ImageIcon
  - See PlainPuzzleIcon and ImagePuzzleIcon
  - An Icon doesn’t typically grow, but in this application we want it to resize when app is resized

Image drawing and construction

- In this application, getIconHeight and getIconWidth are implemented by Icon subclasses, but ignored
  - Not used by puzzle button, drawing resizes as needed
- The paintIcon method passed a Component, information from component used to draw icon appropriately-sized
  - See parameters to drawImage and to fill3DRect passed to graphics object in drawing routines
- Each ImageIcon knows what piece of image it is and draws this scaled piece with complex drawImage method
  - Determining which piece tricky, need static count (1st piece, 2nd piece, ...) on a per-image basis
  - See IconCounterCounter class and use

What about a “real game”

- How to make this a game? What are use cases/scenarios?
  - Shuffle pieces initially, pick random piece near blank and move it, repeat 5, 10, 20 times [degree of difficulty]
    - How does this facilitate auto-solve?
    - What about showing the numbered tiles as a hint when just image puzzle used by client
  - Auto-complete, we can undo string of moves, track all
    - Coalesce moves that are redundant
    - Recognize previous state of board and go back