* + Model = class that defines the data
  + View = User interface where users cause actions
    - both model and view listen to each other
  + Ex: Playlist and PlaylistView
    - PlaylistView has two fields: Playlist and a JPanel
      * the constructor sets the layout as BorderLayout, sets the layout of the internal JPanel as GridLayout
      * create a for loop that sets each Song into a JLabel with the title being the song's name
    - You may also add a widget where the user can input a new song and it directly adds to the PlaylistView
      * it extends JLabel and implements ActionListener
      * for each field:

song\_field = new JTextField(10);

add(new JLabel("Song: ");

add(song\_field);

* + you can have a slider for the song's rating like so:
    - rating\_slider = newJSlider(0,5,0);
  + and a button to add the song

add\_song\_button = new JButton("+");

add\_song\_button.setActionCommand("add");

add(add\_song\_button);

add\_song\_button.addActionListener(this);

* + then, add a actionPerformed(ActionEvent e){} method
  + Connecting model to view
    - model needs to inform the view whenever it changes so that the view can rebuild to reflect any change
      * approach: make the model observable(make Playlist extend java.util.Observable) and to register PlaylistView as an observer(by implementing Observer)
        + Within the Playlist class, whenever a song is added, removed, or shuffled(or whenever the playlist changes), you must call setChanged() and notifyObservers() to notifyt he observers
        + Within the PlaylistView, make it an observer of Playlist

add a public void update(Observable arg0, Object arg1){} method

first, you remove everything from the list\_panel using list\_panel.removeAll();

then do a for loop that adds a new song label with each song's title to the list\_panel

then call list\_panel.revalidate();

* + Within the view, you can add the list\_view into a JScrollPane, which will then make it possible to add songs beyond the size of the window, introducing a scroll
  + You could also create more interesting song listing
    - SongListingWidget
      * includes a button for removing the song from the playlist and moving them up and down the list
      * for each button, call button.addActionListener(this);
        + causes the SongListingWidget to be notified whenever the button is pressed
    - BorderFactory
      * swing class for creating borders for components
    - Unicode characters
      * lots of glyphs available
      * changed Song to use star glyph for rating
  + Model povides access to an abstraction
    - any manipulation/modifaction done via public methods provided by a model
  + Model-View is good for simple, direct UI elements
    - one model object drivig UI view object
    - UI reflects model state directly
    - user interactions with view have direct mapping to updates/public methods of model
    - complex behavior is harder to express
      * dynamic user interfaces
      * complex models
      * contextual user interactions