331 Section 7 Worksheet

Note: The code here is just one way of implemention, so you don't have to strictly follow this design.

```
public class Movie {
    public String getName() { /* abridged */ }
    public String getInfo() { /* abridged */ }
    public Date getAiringDate() { /* abridged */ }
    public Map<String, String> getStaff() { /* abridged */ }
}

/**
 * Abridged
 */
public class MovieCategoryModel {

    // abridged

    public MovieCategoryModel() { /* abridged */ }
    public void addMovie(Movie movie) { /* abridged */ }
    public void removeMovie(Movie movie) { /* abridged */ }
    public List<Movie> findMovies(String namePrefix) { /* abridged */ }
}

public class MovieCategoryController {
    private final MovieCategoryModel model;
    private final MovieCategoryView view;

    public MovieCategoryController(MovieCategoryModel model, MovieCategoryView view) {
        this.model = model;
        this.view = view;

        view.addAddMovieListener(new AddMovieListener());
        view.addQueryListener(new QueryListener());
        view.addRemoveMovieListener(new RemoveMovieListener());
    }

    private class AddMovieListener implements MovieCategoryView.MovieMVCListener {

        @Override
        public void notifyListener(Movie movie) {
            model.addMovie(movie);
            view.reset();
            view.updateView(movie.getName());
        }
```

```
    }

    private class QueryListener implements MovieCategoryView.MovieMVCListener {

        // only the getName() is used to deliver the information
        @Override
        public void notifyListener(Movie movie) {
            view.reset();
            view.updateView(movie.getName());
        }
    }

    private class RemoveMovieListener implements MovieCategoryView.MovieMVCListener {

        @Override
        public void notifyListener(Movie movie) {
            model.removeMovie(movie);
            view.reset();
            view.updateView("");
        }
    }
}

public class MovieCategoryView {

    public interface MovieMVCListener {
        /**
         * Notify the listener with the given Movie.
         * @param movie The chosen movie to notify, could potentially be null
         */
        void notifyListener(Movie movie);
    }
    private final MovieCategoryModel model;
    private final List<MovieMVCListener> queryListeners;
    private final List<MovieMVCListener> addMovieListeners;
    private final List<MovieMVCListener> removeMovieListeners;

    public void startMainLoop() { /* abridged */ }

    public void addQueryListener(MovieMVCListener listener) {}
    public void addAddMovieListener(MovieMVCListener listener) {}
    public void addRemoveMovieListener(MovieMVCListener listener) {}

    /**
     * Update this view.
     * @param prefix the prefix of all movies that will be displayed
     * @effects This view will show the given movies from now on
     * @modifies this
```

```
     */
    public void updateView(String prefix) {
        List<Movie> movies = model.findMovies(prefix);
        /*
            Dark magic that displays the movies on the screen
         */
    }

    /**
     * Clear the user inputs and reset the view.
     * @effects the view will be as clean as it was initialized.
     */
    public void reset() {
        /*
            Dark magic that clears all user inputs
         */
    }
}

public class MovieCategoryMain {
    public static void main(String[] args) {
        MovieCategoryModel model = new MovieCategoryModel();
        MovieCategoryView view = new MovieCategoryView(model);
        MovieCategoryController controller = new MovieCategoryController(model, view);
        view.startMainLoop();
    }
}
```

1. If we want to record all user input and display them in a seperate frame, where should this be implemented?
    - model
    - view
    - controller
2. Suppose we want to protect our program from code injection in user input, where should we add the checks?
    - model
    - view
    - controller
3. Our client seems to be unhappy with the lack of features in our program. As a result, in the next release we are going to add a "search by date" feature to our Movie Category program. How are we going to implement it?

4. Our client seems to be tired of GUI application, so we are going to release a CLI (command line) version of our program. Which of the following are going to change?

    - model
    - view
    - controller

5. Now our client want to see a brief summary for each filtered movie instead of the current put-all-thing-we-known-on-the-screen version. Where should this be implemented?

    - model
    - view
    - controller

6. Our client want to add a graph to the GUI version of our application that display stats of our database, and parts on the graph should be clickable so that the client can do some special operations on them. Assume the API is provided by the model, where should this be implemented?

    - view
    - controller