Team: (#23)

Avneendra Kanva

Michael Pauly

Title: Media Preference Tracking

Report:

1. What features were implemented?

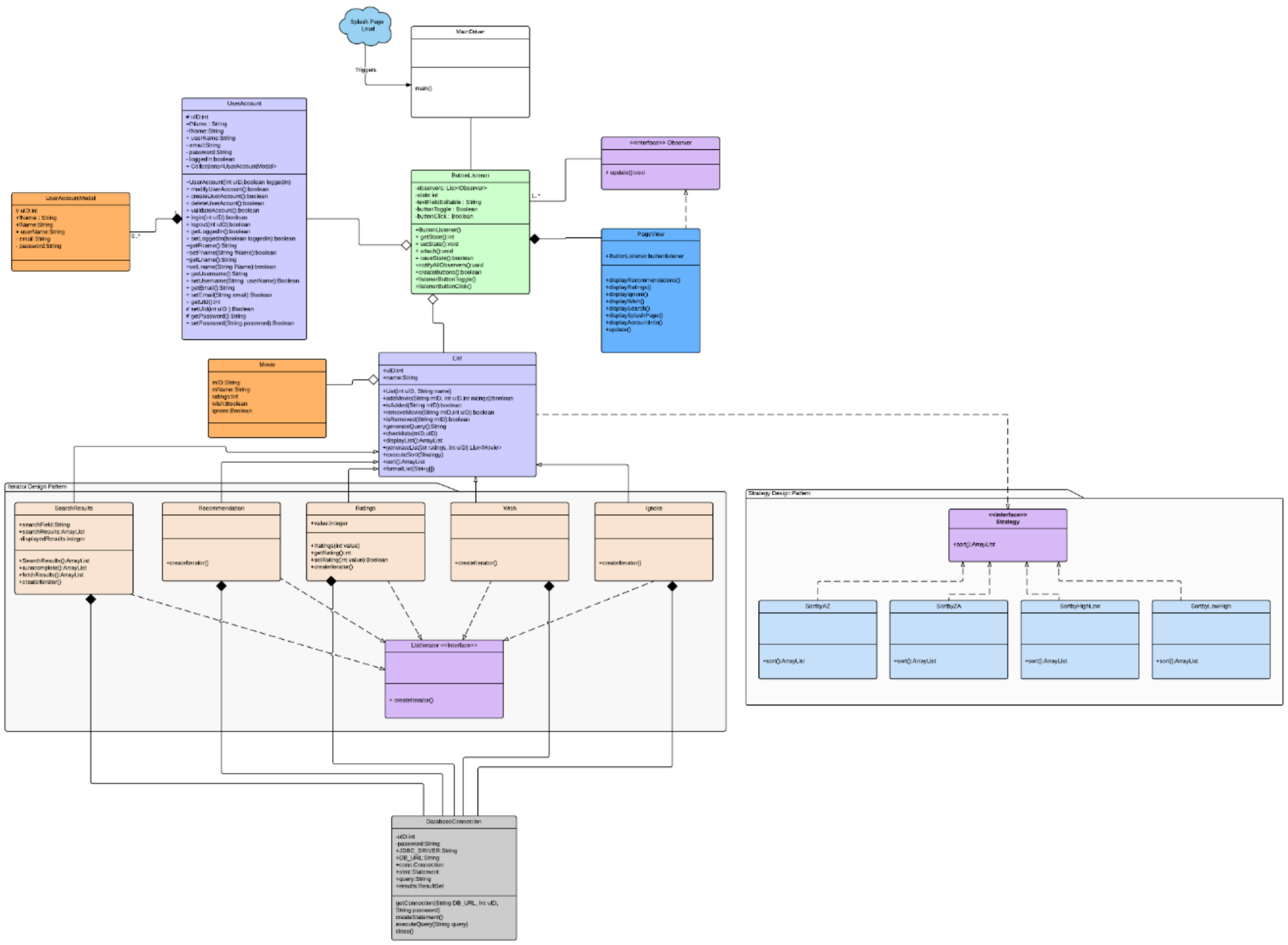
   Features implemented by Avneendra Kanva:
   - UR-01 User can create new account.
   - UR-02 User can log in after creating account.
   - UR-03 User can modify account.
   - UR-05 User can log out.
   - UR-06 User can toggle between recommendations/search/my list modes.
   - UR-07 User can edit preferences of movies.
   - UR-09 User can toggle between ratings/ignore/wishlist.
   - UR-08 User can search movies.
   - FR-01 System verifies duplicate account doesn't exist when user makes account.
   - FR-07 System will add movie to list selected by user and remove it from all others.
   - NFR-01 All users can log in
   - NFR-02 System maintains all user's information.
   - FR-02 System verifies user password is acceptable format
   - FR-03 System will remove a movie from wishlist once it is rated

   Features implemented by Michael Pauly:
   - UR-08 User can search movies.
   - UR-12 User can navigate search results.

2. **Which features were not implemented from part 2?**
    - UR-04 User can delete account
    - UR-10 User can add comments to rated movies
    - UR-11 Commented ratings can be viewed when movie is being suggested
    - UR-13 User can sort movies
    - UR-14 User can navigate to IMDB page of movie
    - FR-04 System will suggest results while user is searching
    - FR-06 System will provide URL to IMDB
    - FR-08 System will track displayed search results
    - FR-09 System will not recommend movies from users ignore list
    - FR-10 System will give higher priority to wishlist movies when recommending
    - NFR-03 Database is secure
    - NFR-04 System will restrict number of failed login attempts
    - NFR-05 System will auto generate email to verify user email upon account creation
    - NFR-06 System will auto generate “reset password” email
    - NFR-07 User can only be logged in from one location
    - NFR-08 System will recognize beginning/end of lists and prevent use of non-relevant buttons

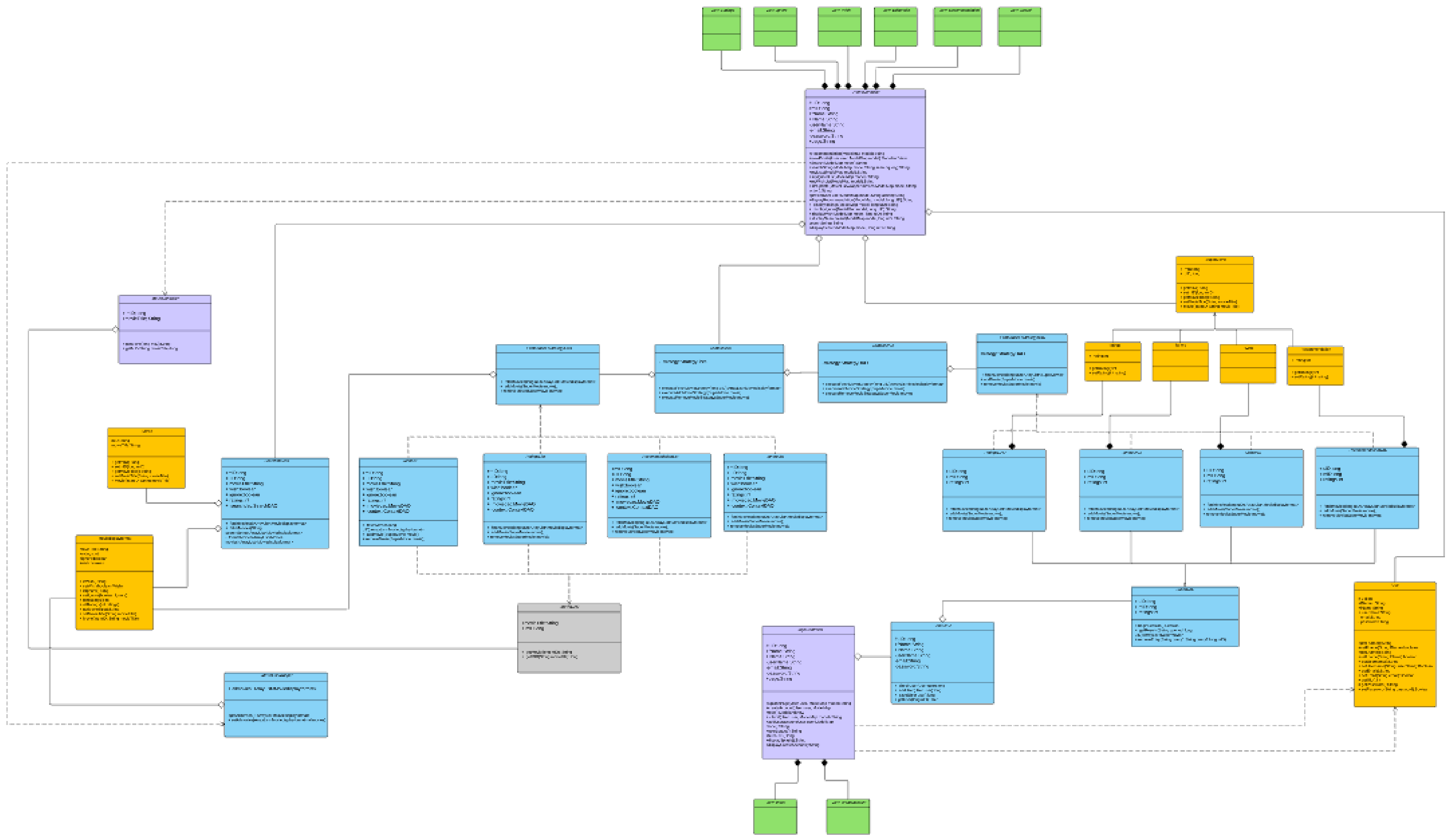
3. Class Diagram:

- Initial Class Diagram:

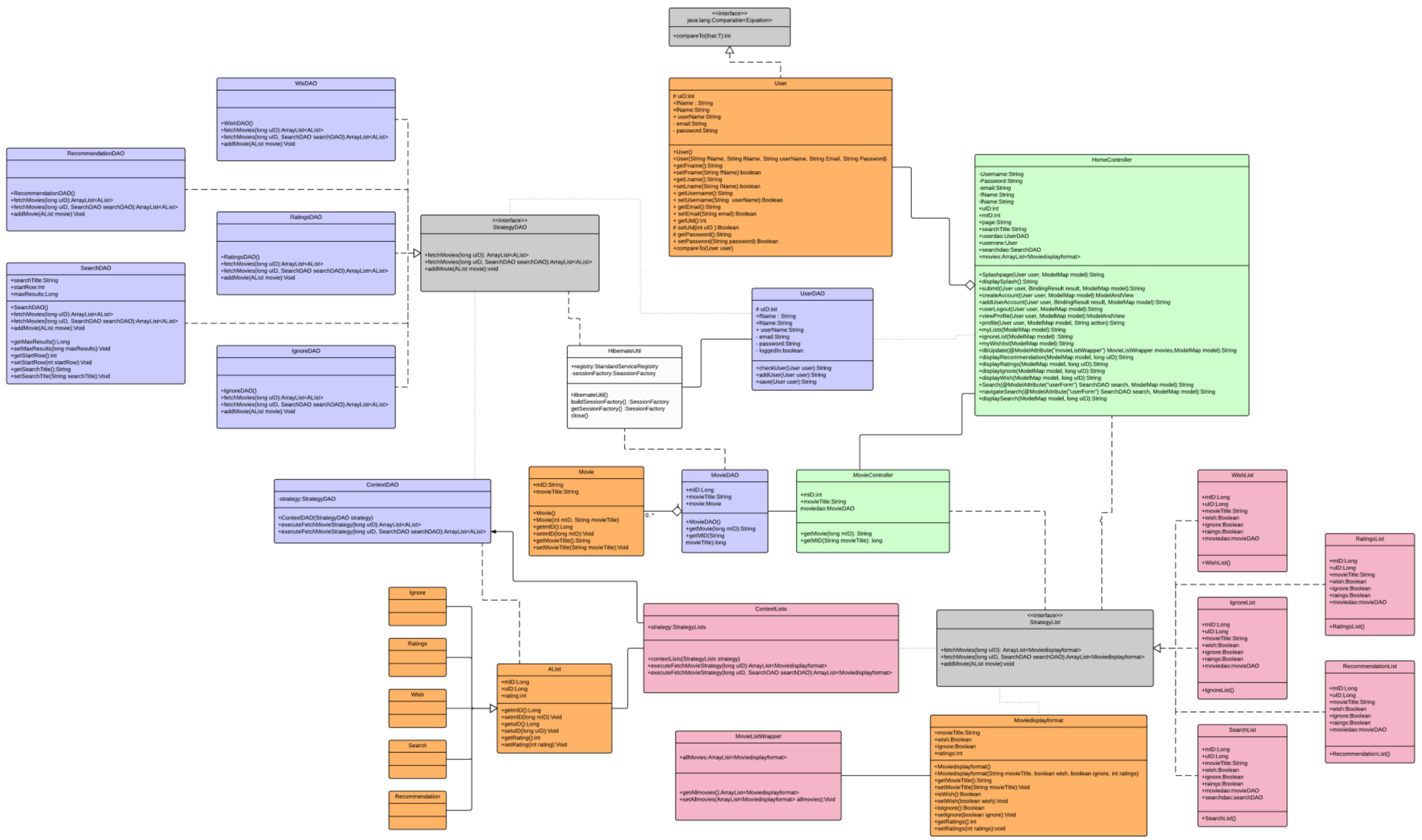


- Final Class Diagram:

  Based on Avneendra's implementation:



Based on Michael’s implementation (using Avneendra’s code):



<<interface>>
java.lang.Comparable<Equation>
+compareTo(that:T):int
User
# uID:int
+fName : String
+lName:String
+ userName:String
- email:String
- password:String
+User()
+User(String fName, String lName, String userName, String Email, String Password)
+getFname():String
+setFname(String fName):boolean
+getLname():String
+setLname(String lName):boolean
+ getUsername():String
+ setUsername(String userName):Boolean
+ getEmail():String
+ setEmail(String email):Boolean
+ getUId():int
# setUId(int uID):Boolean
# getPassword():String
+ setPassword(String password):Boolean
+compareTo(User user)
WisDAO
+WishDAO()
+fetchMovies(long uID):ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):Void
RecommendationDAO
+RecommendationDAO()
+fetchMovies(long uID):ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):Void
RatingsDAO
+RatingsDAO()
+fetchMovies(long uID):ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):Void
SearchDAO
+searchTitle:String
+startRow:Int
+maxResults:Long
+SearchDAO()
+fetchMovies(long uID):ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):Void
+getMaxResults():Long
+setMaxResults(long maxResults):Void
+getStartRow():int
+setStartRow(int startRow):Void
+getSearchTitle():String
+setSearchTtle(String searchTitle):Void
IgnoreDAO
+IgnoreDAO()
+fetchMovies(long uID):ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):Void
<<interface>>
StrategyDAO
+fetchMovies(long uID): ArrayList<AList>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<AList>
+addMovie(AList movie):void
HibernateUtil
+registry:StandardServiceRegistry
-sessionFactory:SeassionFactory
HibernateUtil()
buildSessionFactory() :SessionFactory
getSessionFactory() :SessionFactory
close()
UserDAO
# uID:int
+fName : String
+lName:String
+ userName:String
- email:String
- password:String
- loggedIn:boolean
+checkUser(User user):String
+addUser(User user):String
+save(User user):String
HomeController
-Username:String
-Password:String
-email:String
-fName:String
-lName:String
+uID:int
+mID:int
+page:String
+searchTitle:String
+userdao:UserDAO
+usernew:User
+searchdao:SearchDAO
+movies:ArrayList<Moviedisplayformat>
+Splashpage(User user, ModelMap model):String
+displaySplash():String
+submit(User user, BindingResult result, ModelMap model):String
+createAccount(User user, ModelMap model):ModelAndView
+addUserAccount(User user, BindingResult result, ModelMap model):String
+userLogout(User user, ModelMap model):String
+viewProfile(User user, ModelMap model):ModelAndView
+profile(User user, ModelMap model, String action):String
+myLists(ModelMap model):String
+ignoreList(ModelMap model) :String
+myWishlist(ModelMap model):String
+dbUpdate(@ModelAttribute("movieListWrapper") MovieListWrapper movies,ModelMap model):String
+displayRecommendation(ModelMap model, long uID):String
+displayRatings(ModelMap model, long uID):String
+displayIgnore(ModelMap model, long uID):String
+displayWish(ModelMap model, long uID):String
+Search(@ModelAttribute("userForm") SearchDAO search, ModelMap model):String
+navigateSearch(@ModelAttribute("userForm") SearchDAO search, ModelMap model):String
+displaySearch(ModelMap model, long uID):String
ContextDAO
-strategy:StrategyDAO
+ContextDAO(StrategyDAO strategy)
+executeFetchMovieStrategy(long uID):ArrayList<AList>
+executeFetchMovieStrategy(long uID, SearchDAO searchDAO):ArrayList<AList>
Movie
+mID:String
+movieTitle:String
+Movie()
+Movie(int mID, String movieTitle)
+getmID():Long
+setmID(long mID):Void
+getMovieTitle():String
+setMovieTitle(String movieTitle):Void
0..*
1
MovieDAO
+mID:Long
+movieTitle:String
+movie:Movie
+MovieDAO()
+getMovie(long mID):String
+getMID(String movieTitle):long
MovieController
+mID:int
+movieTitle:String
moviedao:MovieDAO
+getMovie(long mID): String
+getMID(String movieTitle): long
WishList
+mID:Long
+uID:Long
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+raings:Boolean
+moviedao:movieDAO
+WishList()
RatingsList
+mID:Long
+uID:Long
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+raings:Boolean
+moviedao:movieDAO
+RatingsList()
IgnoreList
+mID:Long
+uID:Long
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+raings:Boolean
+moviedao:movieDAO
+IgnoreList()
RecommendationList
+mID:Long
+uID:Long
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+raings:Boolean
+moviedao:movieDAO
+RecommendationList()
SearchList
+mID:Long
+uID:Long
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+raings:Boolean
+moviedao:movieDAO
+searchdao:searchDAO
+SearchList()
Ignore
Ratings
Wish
Search
Recommendation
AList
+mID:Long
+uID:Long
+rating:int
+getmID():Long
+setmID(long mID):Void
+getuID():Long
+setuID(long uID):Void
+getRating():int
+setRating(int rating):Void
ContextLists
+strategy:StrategyLists
+contextLists(StrategyLists strategy)
+executeFetchMovieStrategy(long uID):ArrayList<Moviedisplayformat>
+executeFetchMovieStrategy(long uID, SearchDAO searchDAO):ArrayList<Moviedisplayformat>
<<interface>>
StrategyList
+fetchMovies(long uID): ArrayList<Moviedisplayformat>
+fetchMovies(long uID, SearchDAO searchDAO):ArrayList<Moviedisplayformat>
+addMovie(AList movie):void
MovieListWrapper
+allMovies:ArrayList<Moviedisplayformat>
+getAllmovies():ArrayList<Moviedisplayformat>
+setAllmovies(ArrayList<Moviedisplayformat> allmovies):Void
Moviedisplayformat
+movieTitle:String
+wish:Boolean
+ignore:Boolean
+ratings:int
+Moviedisplayformat()
+Moviedisplayformat(String movieTitle, boolean wish, boolean ignore, int ratings)
+getMovieTitle():String
+setMovieTitle(String movieTitle):Void
+isWish():Boolean
+setWish(boolean wish):Void
+isIgnore():Boolean
+setIgnore(boolean ignore):Void
+getRatings():int
+setRatings(int ratings):void

3. (Continued)

- <u>What Changed and Why?</u>

  During Part 2, we were not entirely sure on how to render HTML pages from the Java code or how to persist data into the database. The entire concept of Model View Controller as well as Hibernate were new to both individuals involved with this project. However, while Avneendra was working towards implementation of the project, functionalities of Spring MVC and Hibernate became clearer to him. The initial intent of using JDBC and Michael's learning experience of stored procedures were abandoned due to the significance and relevance of Hibernate.

  There was quite a bit that changed about the structure of the project. Major portions of the class diagram had to be redesigned after having a better understanding of Spring MVC as well as database interaction via Hibernate. The original approach to the class diagram was based around a more focused solution, meaning utilizing less of the premade functionality of Java, Spring, and other packages.

- <u>How did upfront design help development?</u>

  There were a number of requirements and design patterns that were discussed and discovered in the process of making the preliminary class diagram. This provided a solid foundation and road map towards implementation. It was an excellent tool to facilitate and guide discussion, as well as discover team member's understandings of system requirements and interaction. It was also a valuable tool to reflect upon and recognize some of the challenges we faced could have been resolved with various design patterns.

4. **Did you make use of any design patterns in implementation of final prototype? How so? If not, where could you make use of design patterns in your system?**

<u>Design patterns implemented by Avneendra:</u>

I used Spring MVC for this project. It by default as defined in the framework, uses Observer Design pattern for the Model and View. View acts as a subscriber which gets notified whenever Model has changed. Strategy Design pattern is used by the controller. Composite design pattern is used by the View.

I used Strategy design pattern to generate the list of movies to display based on the view that the user has selected or clicked. I further implemented Strategy design pattern to decide upon which table in the database among the four (Recommendation, Ratings, Ignore, and Wish) upon which to execute the query. This was similar to the nested Strategy design pattern.

I used very similar to proxy design pattern. The updates made by user on any page were not persisted to the database until the user clicked on Save on that page.

Adapter Design pattern could have been used to integrate Search list class within the Strategy List class interface. It was returning a different type of collection of Array List than all the other 4 lists.

Iterator design pattern could have been used to iterate through list of movies that were returned to the controller by the database.

4. (Continued)

Design Patterns by Michael Pauly:

The proxy design pattern lends itself well to this project, keeping user profile updates on client side until the user saves all edits.

The command design pattern would have allowed a user to change ratings/toggle wish list/toggle ignore list options multiple times, in effect undoing the last action performed, without requiring an update to the data set with each action. This could also tie back into using Proxy for more deliberate commits to the database.

Prototype combined with factory could be heavily relied upon to create the interactive result sets, but a form of strategy was used that was effective as well.

The intent of building the web pages and result sets could be done with a combination of design patterns. Template could be used in determining what information to add or remove from any return results, as there was discussion of the system prioritizing or removing information based on each users profile information (e.g. check ignore list before recommending movies, or recognize that ignore list is empty and return results). Decorator would have also been an interesting choice, allowing pages to create the lists and navigation buttons, lists to create the movie object and sort options, and the movies themselves to create the rate/ignore/wish buttons.

5. **What have you learned about the process of analysis and design now that you have stepped through the process to create, design, and implement a system?**

When approaching a new challenge, there can never be enough preparation. A complete understanding of the problem to be solved as well as knowledge regarding all tools available are invaluable. Due to the dynamic nature of programming, it is pertinent to discuss every aspect of a project. More upfront communication can help catch unforeseen obstacles or solutions and thus provide more direction during implementation. Design patterns are an excellent summation of succinct tools to resolve both common and difficult requirements.

The design and analysis required by completing both the activity diagrams and use case diagrams really helped in laying the foundation for the requirements. This was an excellent point of reference during the implementation part of the project. The discussion and thought that went into making class diagrams and sequence diagrams helped immensely throughout the project. This pre work facilitated learning Spring MVC, database interaction, as well as the significance of dependency injection.

Thinking about design patterns and coding to abstraction were new concepts to both party members. While implementing the project, Avneendra would recognize inefficiencies and redundancy in the code and begin to investigate including design patterns. After having implemented this project, Avneendra has come away with a better understanding of both design and structure of this project and how to apply the open-closed principle not only to this implementation, but also its importance for interchangeability in future projects.

The depth of functionality provided by both Spring and Hibernate, and their strong intractability with XML makes these an obvious choice for this type of project. Utilizing GIT as not only a data backup, but a means to collaborate quickly was also a very valuable skill acquired. This project as well as the course material tied really well into each other. With a better understanding of UML diagrams, MVC, Design patterns (and anti-patterns) a more intelligent discussion and concrete direction can be established prior to beginning work. Further efforts to utilize and understand dependency injection will also be necessary in addition to solidifying all of the skills acquired through this process.