CSCI 4448
Project 7
05-3-2023

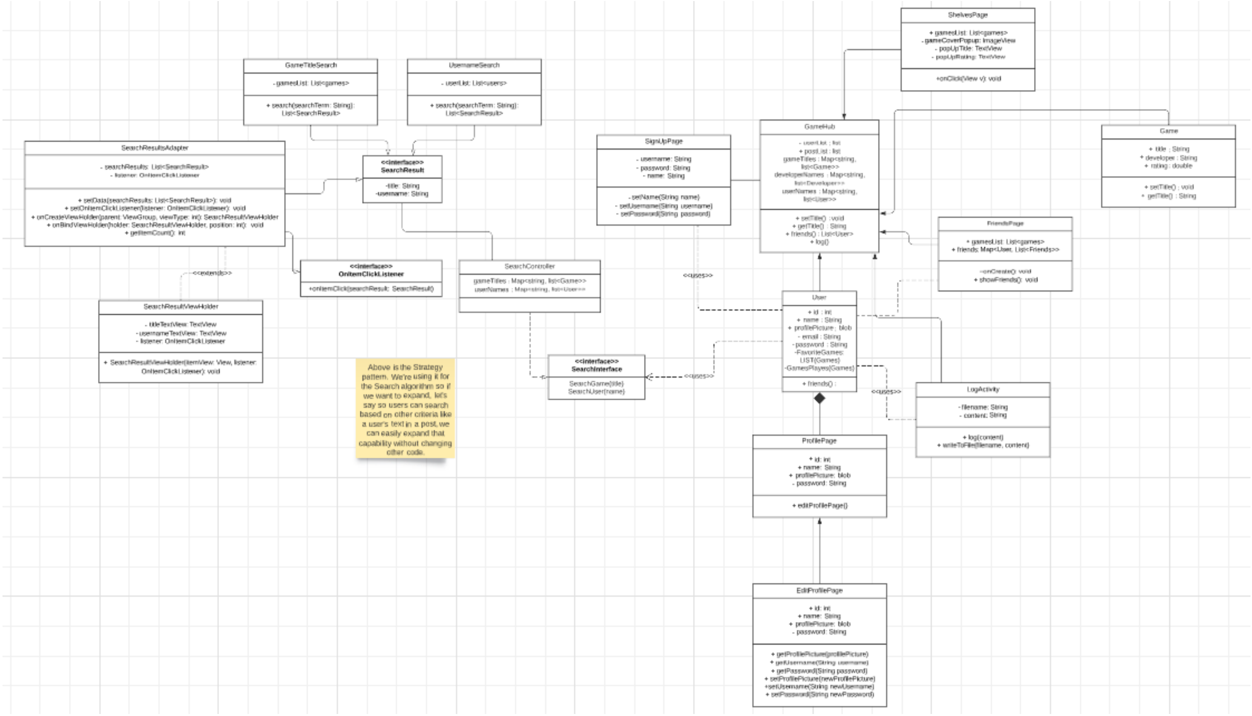
Adrian Ornelas Ruvalcaba
Jonathan Goins
Ethan Kellerhals

GameHub

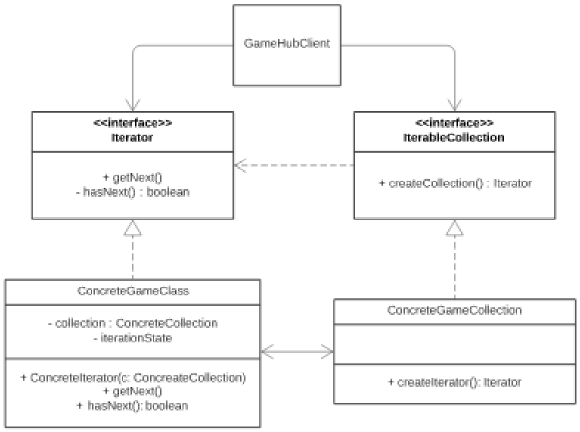
Final State of System Statement

Our group is very proud of the final stage of the project when it comes to building an application from scratch with a concept in mind. While our goals were very ambitious, considering the time to complete the project and use an unfamiliar environment, we implemented many of the features. Still, we needed to integrate them fully as expected. Android Studio was not used amount any of the three team members, and the learning curve was very drastic. The features we implemented are the ability to log/review a video game, a searchable part to find a match, a games library display, friends display, clickable titles, edit profile photos, and login/sign-up features. The features we could not add range from the ability to customize the lists, sound feedback, and the ability to add/remove friends. These features needed to be cut due to the increasing complexity and issues with compatibility with our current code.

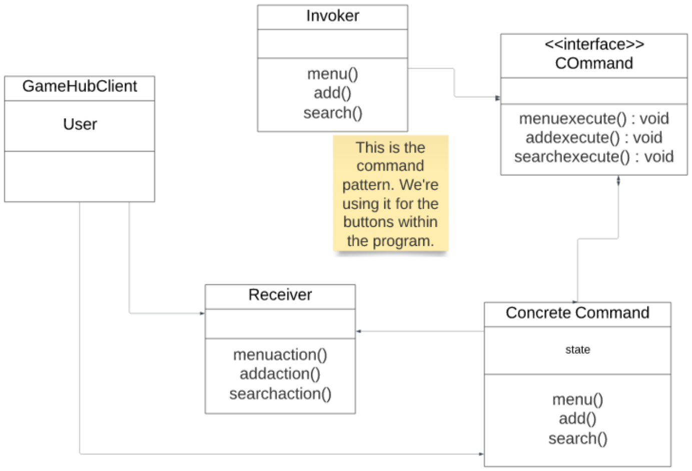
Final Class Diagram and Comparison Statement


GameTitleSearch
- gamesList: List<games>
+ search(searchTerm: String): List<SearchResult>
UsernameSearch
- userList: List<users>
+ search(searchTerm: String): List<SearchResult>
SearchResultsAdapter
- searchResults: List<SearchResult>
- listener: OnItemClickListener
+ setData(searchResults: List<SearchResult>): void
+ setOnItemClickListener(listener: OnItemClickListener): void
+ onCreateViewHolder(parent: ViewGroup, viewType: int): SearchResultViewHolder
+ onBindViewHolder(holder: SearchResultViewHolder, position: int): void
+ getItemCount(): int
<<interface>>
SearchResult
-title: String
-username: String
<<extends>>
<<interface>>
OnItemClickListener
+onItemClick(searchResult: SearchResult)
SearchResultViewHolder
- titleTextView: TextView
- usernameTextView: TextView
- listener: OnItemClickListener
+ SearchResultViewHolder(itemView: View, listener: OnItemClickListener): void
SearchController
gameTitles : Map<string, list<Game>>
userNames : Map<string, list<User>>
<<interface>>
SearchInterface
SearchGame(title)
SearchUser(name)
<<uses>>
SignUpPage
- username: String
- password: String
- name: String
- setName(String name)
- setUsername(String username)
- setPassword(String password)
<<uses>>
ShelvesPage
+ gamesList: List<games>
- gameCoverPopup: ImageView
- popUpTitle: TextView
- popUpRating: TextView
+onClick(View v): void
GameHub
- userList : list
+ postList : list
gameTitles : Map<string, list<Game>>
developerNames : Map<string, list<Developer>>
userNames : Map<string, list<User>>
+ setTitle() : void
+ getTitle() : String
+ friends() : List<User>
+ log()
Game
+ title : String
+ developer : String
+ rating : double
+ setTitle() : void
+ getTitle() : String
FriendsPage
+ gamesList: List<games>
+ friends: Map<User, List<Friends>>
-onCreate(): void
+ showFriends(): void
User
+ id : int
+ name : String
+ profilePicture : blob
- email : String
- password : String
-FavoriteGames: LIST(Games)
-GamesPlayes(Games)
+ friends() :
<<uses>>
LogActivity
- filename: String
- content: String
+ log(content)
+ writeToFile(filename, content)
ProfilePage
+ id: int
+ name: String
+ profilePicture: blob
- password: String
+ editProfilePage()
EditProfilePage
+ id: int
+ name: String
+ profilePicture: blob
- password: String
+ getProfilePicture(profilePicture)
+ getUsername(String username)
+ getPassword(String password)
+ setProfilePicture(newProfilePicture)
+setUsername(String newUsername)
+ setPassword(String newPassword)

Above is the Strategy pattern. We're using it for the Search algorithm so if we want to expand, let's say so users can search based on other criteria like a user's text in a post, we can easily expand that capability without changing other code.



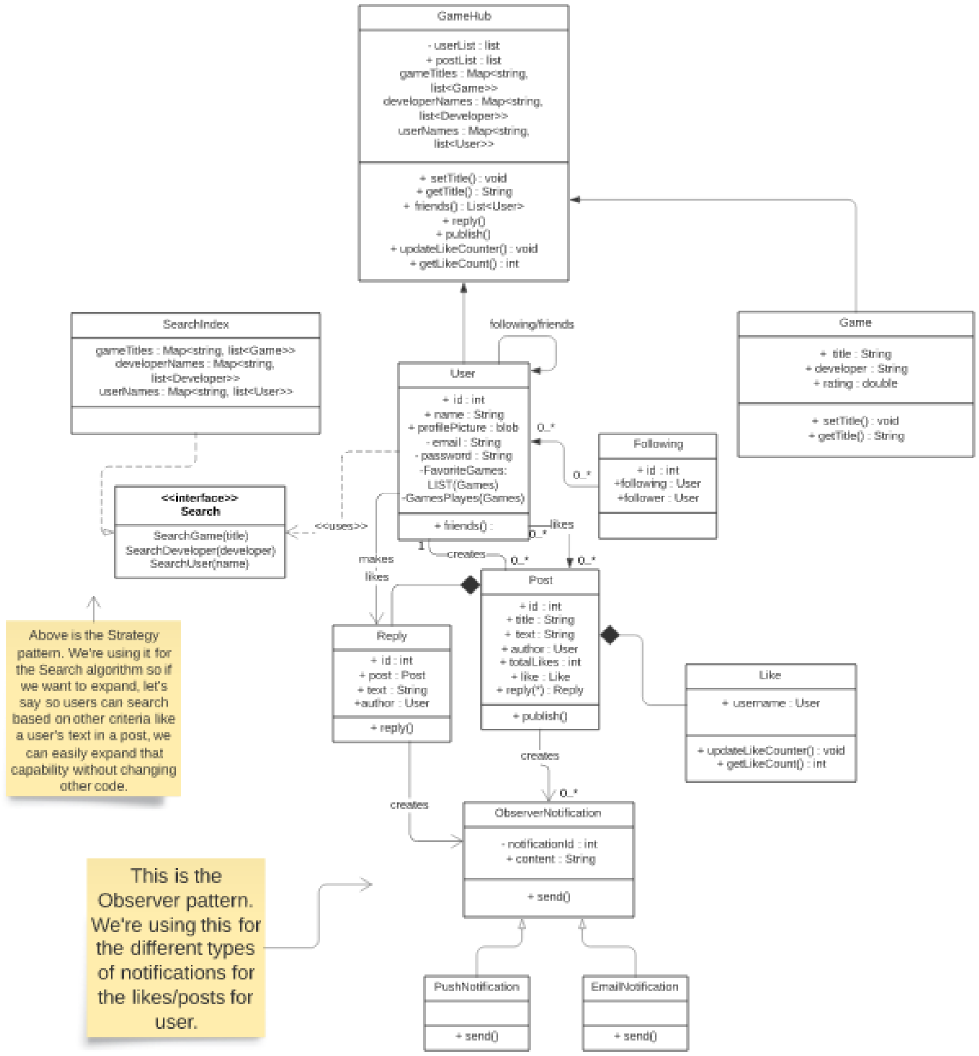
This is the Iterator pattern. We're using this for the list/page of games that the user can browse through.



This is the command pattern. We're using it for the buttons within the program.

GameHub

Ethan, Jon, and Adrian | April6, 2023



Above is the Strategy pattern. We're using it for the Search algorithm so if we want to expand, let's say so users can search based on other criteria like a user's text in a post, we can easily expand that capability without changing other code.

This is the Observer pattern. We're using this for the different types of notifications for the likes/posts for user.

- The final UML class diagram is at the top of all the others. The Invoker UML class diagram is still the same. However, we didn’t implement the Command Pattern or Observer Pattern. The most significant difference between both UML diagrams is the number of extra classes we had to create. The search function needed a lot of helpers due to the RecyclerView and having to bind data to the results that were popping up. Another difference is the subtraction of the notification, like, and reply classes. We could not make this app live, so those classes could not have worked—the lessening of the command pattern. We were unsure if the buttons between pages counted as the command pattern, thus the removal of it. Another change was the addition of the signup and sign-in classes.

Third-Party Code vs. Original Code Statement

Many of the original code references how to use android studio and its built-in features. Due to not using third-party sources initially and needing to familiarize me with android studio, the code switches into complexity. Most methods were added due to learning the GUI. The group created the art for the application to build an intuitive design centered around the app's use case.

Third-Party: Many of the code used by the third party was references from stack overflow or Android studio documentation to know how the watch function and feature are used and implemented. The code was meant to be redesigned to build around the strategies we used in the class. Designing the patterns and structure of the classes used, for example, logger and command, we used the methods presented in the previous assignments to go based on that. Also, using outside sources for Android studio restrictions.
Sources:
https://developer.android.com/reference/android/os/AsyncTask
https://developer.android.com/reference/android/widget/ScrollView
https://developer.android.com/reference/android/widget/Switch
https://stackoverflow.com/questions/1974489/how-to-get-text-from-edittext
https://developer.android.com/guide/topics/ui/notifiers/toasts
https://stackoverflow.com/questions/13927269/use-of-ondestroy-in-android
https://developer.android.com/guide/components/activities/activity-lifecycle
Some sources are in the files

Statement on the OOAD process for your overall Semester Project

A positive and negative design process was the UI of the application. Our original design was meant to feel familiar to users. There were issues or memory allocation, constraints with application widgets and photos used, and implementing the OO patterns. When implementing these features, we hit a significant barrier in trying to figure out what is possible to achieve quickly.

Another issue we ran into was choosing our four design patterns and planning where they should be used. Determining which OO patterns should go before coding was essential to avoid future errors. This proved challenging and required some changes in implementation to meet the assignment's deadline and be as close to our goal as possible when thinking about code reusability.

Finally, team communication and collaboration resulted from this project when designing and developing this application with OO patterns in mind. We are all students with challenging and different classes, so it was a positive experience collaborating with a team, consistently reaching out for help, assisting one another, and assigning roles based on strengths. While everything could be better, there was some difficulty with a set design pattern; it was well communicated.