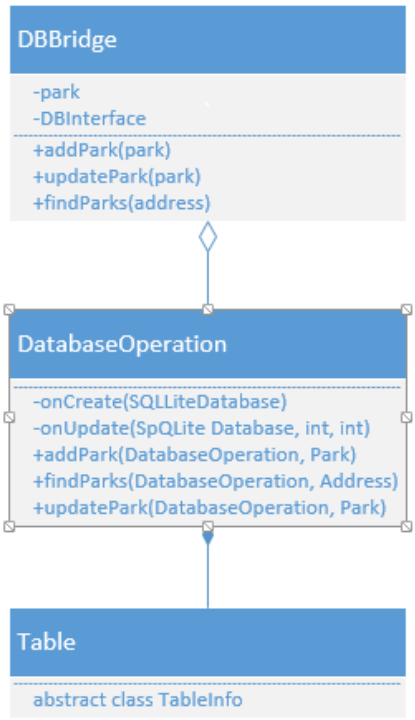
Matt Pennington
Dheeraj Ranga Chinni

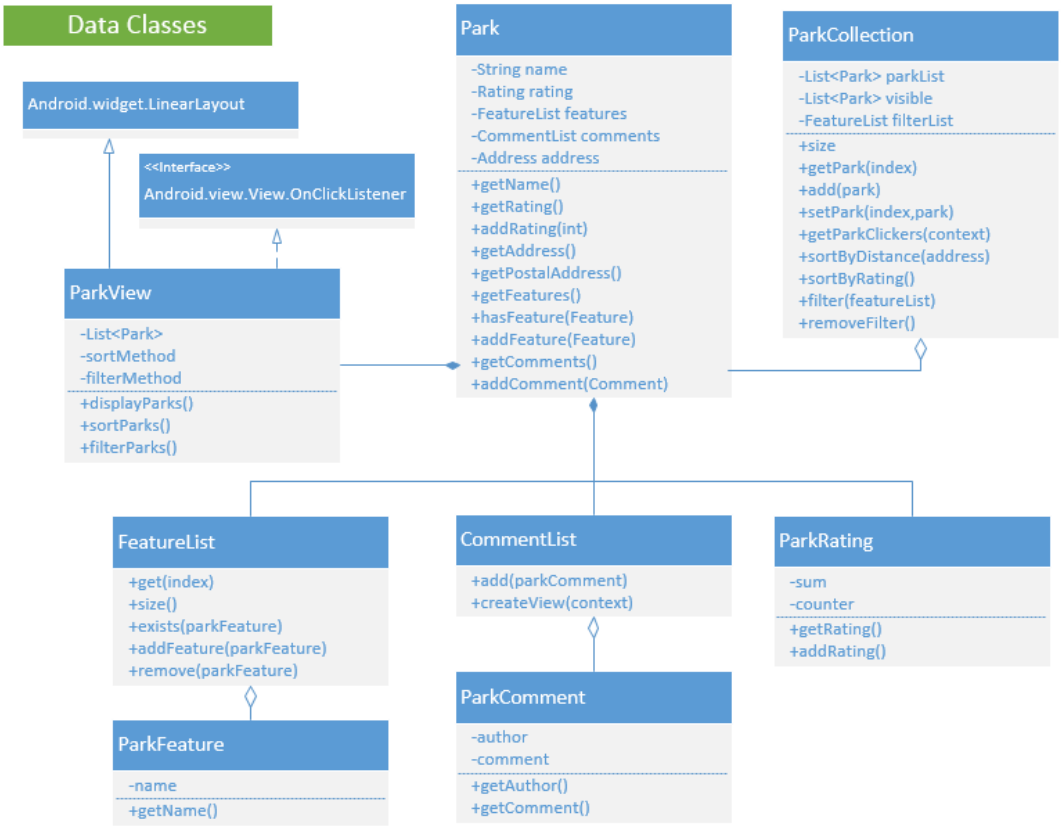
What features were implemented?

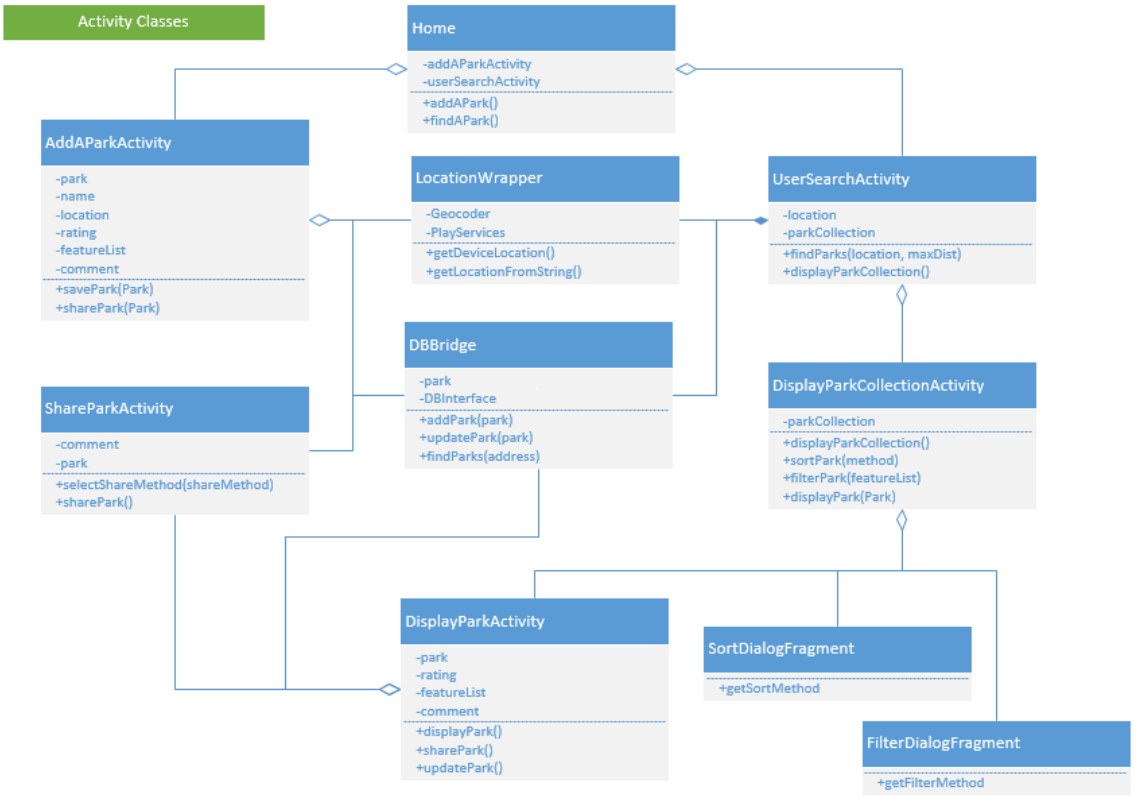
- Activity to gather data from end user
- Learn and record user's location
- Store data in a database
- Retrieve data from database
- Activity to display data to the end user
- Allow user to query, sort, and filter data
- Share information with social media

Class Diagram

Database Classes




Data Classes
Android.widget.LinearLayout
<<Interface>>
Android.view.View.OnClickListener
ParkView
-List<Park>
-sortMethod
-filterMethod
+displayParks()
+sortParks()
+filterParks()
Park
-String name
-Rating rating
-FeatureList features
-CommentList comments
-Address address
+getName()
+getRating()
+addRating(int)
+getAddress()
+getPostalAddress()
+getFeatures()
+hasFeature(Feature)
+addFeature(Feature)
+getComments()
+addComment(Comment)
ParkCollection
-List<Park> parkList
-List<Park> visible
-FeatureList filterList
+size
+getPark(index)
+add(park)
+setPark(index,park)
+getParkClickers(context)
+sortByDistance(address)
+sortByRating()
+filter(featureList)
+removeFilter()
FeatureList
+get(index)
+size()
+exists(parkFeature)
+addFeature(parkFeature)
+remove(parkFeature)
ParkFeature
-name
+getName()
CommentList
+add(parkComment)
+createView(context)
ParkComment
-author
-comment
+getAuthor()
+getComment()
ParkRating
-sum
-counter
+getRating()
+addRating()


Activity Classes
Home
-addAParkActivity
-userSearchActivity
+addAPark()
+findAPark()
AddAParkActivity
-park
-name
-location
-rating
-featureList
-comment
+savePark(Park)
+sharePark(Park)
LocationWrapper
-Geocoder
-PlayServices
+getDeviceLocation()
+getLocationFromString()
UserSearchActivity
-location
-parkCollection
+findParks(location, maxDist)
+displayParkCollection()
DBBridge
-park
-DBInterface
+addPark(park)
+updatePark(park)
+findParks(address)
DisplayParkCollectionActivity
-parkCollection
+displayParkCollection()
+sortPark(method)
+filterPark(featureList)
+displayPark(Park)
ShareParkActivity
-comment
-park
+selectShareMethod(shareMethod)
+sharePark()
DisplayParkActivity
-park
-rating
-featureList
-comment
+displayPark()
+sharePark()
+updatePark()
SortDialogFragment
+getSortMethod
FilterDialogFragment
+getFilterMethod

What design patterns were used?

Facade - During development, it became quickly apparent that using Google's libraries for location systems was not going to be straightforward. There are multiple libraries that must be used. We were only able to gain functionality with the location libraries on one device that we tested. When testing on the BlueStacks android emulator, this system does not work. To increase the number of supported devices in the future, even more libraries will need to be used, particularly HTTP API calls with regards to phones with no cell network, GPS, or Play Services support. Due to the high demand for libraries and the need to for flexibility between devices, we decoupled all interactions with the Android GeoCoder and GoogleAPIClient and encapsulated them within the LocationWrapper class. We limit the calls made to this class to getDeviceLocation() and getLocationFromString(String). This improves the cohesion of several classes and obfuscates the complexity of the location operations.

Adapter: I have written the code for the database which has implement some specific methods which are pre-defined. I had to create a new class called databaseopertions which implements all the methods for the database. I have used these methods in the dbbridge class which is accessed by the addApark activity for inserting data into the database, Usersearch activity to get the parks in the database and displaypark activity to update the database. Addapark activity will access the insert method in the databridge which will calls the method in the database operations to insert data in db.

Bridge: There is a dbbridge class which acts as bridge between the SQLite and the system, this dbbridge can be used with some other databases by just implementing the methods according to that specific database.

How did the system change from HW 4?

We have created a dbbridge which is working as a bridge between the database and the methods application. This lead to implement the Bridge pattern in the project.

Shifts in Responsibilities. When we moved from the design phase into the development phase, some of the classes exchanged responsibilities in order to improve cohesion. For example, we had originally included the method for sorting and filtering the ParksCollection in the DisplayParkCollectionActivity class. However, when it came time to implement this class, it became readily apparent that this would create tight coupling between the ParksCollection and the DisplayParksCollection classes. Thus we delegated the sorting and filtering responsibilities from the Activity class to the Collection class, which had direct access to the data. Another example of shifting responsibilities was to move the creation of display elements from the Activity classes to the Collection classes. As the Collection Classes had access to the data elements and the Activities did not, it was simpler for the CommentList activity to have a method for creating a View object (createView) and call it from whichever Activity happened to need it at runtime. As it turns out, this was probably not the best design decision, which is explained in detail in the next section.

What have you learned about design and analysis?

We have learnt about how to design the system using ooad concepts and applying design patterns.

Variability Analysis is critical. One of the most time consuming tasks for this project was developing code to display and interact with the design elements (View objects). During the design phase, I failed to plan for this task and simply delegated the creation of View objects to each individual Activity class. After completing several Activities, I realized in hindsight that a lot of code I had written was duplicated. If I had had a greater understanding of user interface design before beginning the project I would try to reduce the scope of this task by encapsulating it in a design pattern. If I had first designed a ViewFactory, I could have developed the code for creating UI elements once and reused the constructors as needed throughout the application. In the current state, if I need to extend the application to support different screen sizes and orientations using the current design structure, it will be a monumental task. With a ViewFactory, it would be much simpler. What I've learned from this mistake is that spending extra time and effort during the variability analysis phase can lead to exponential improvements later in terms of reducing time in the development and maintenance phases.