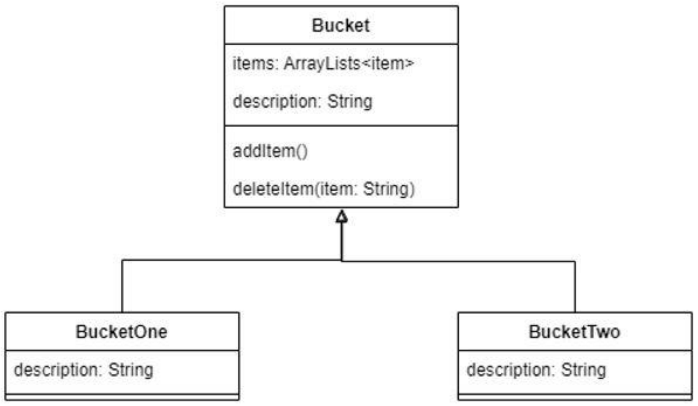
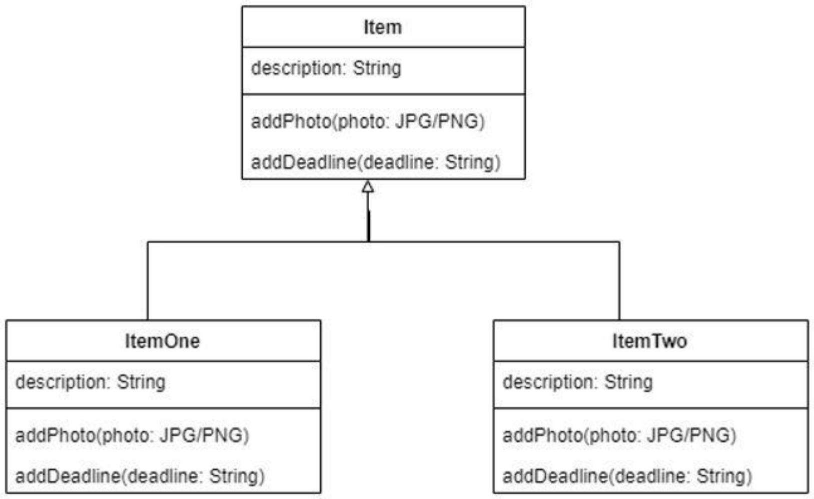
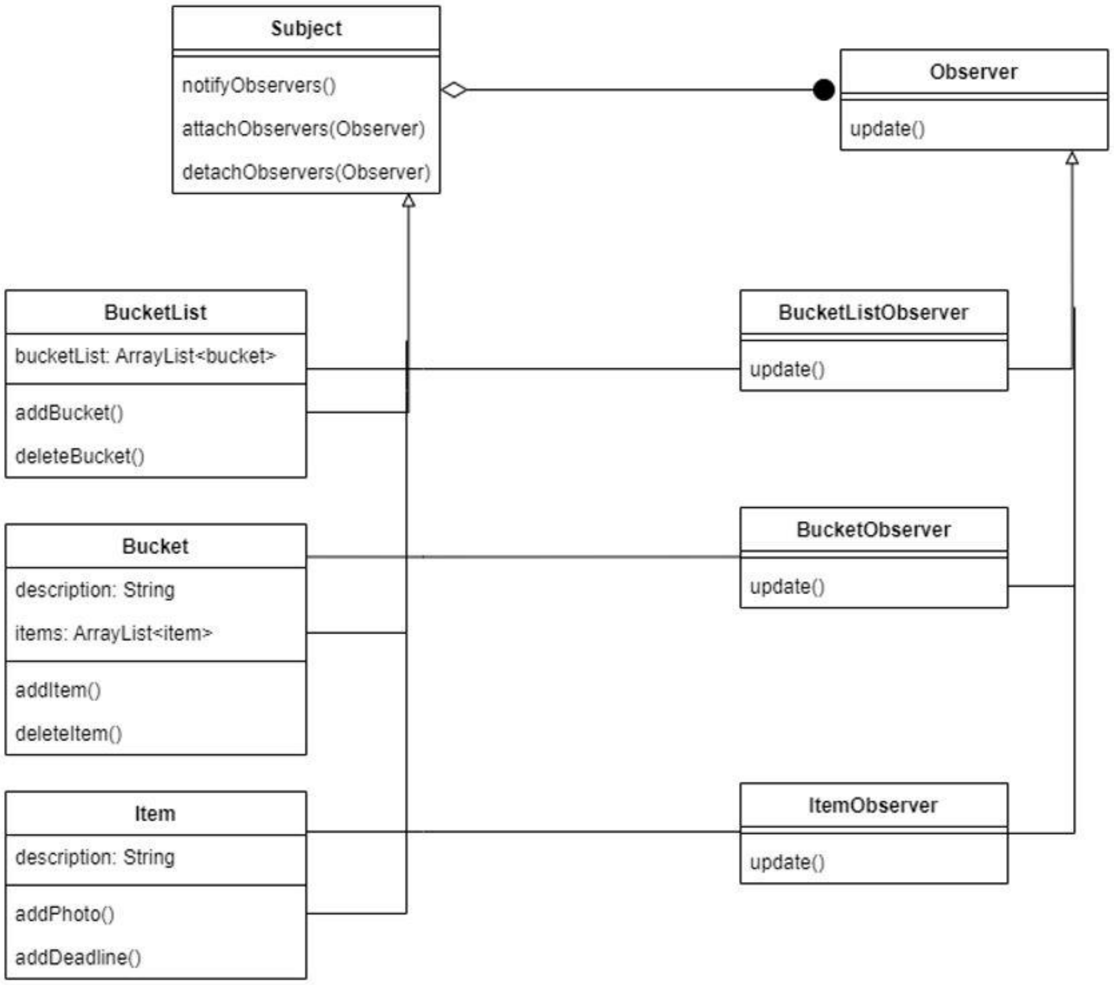
Design Pattern Diagrams:

A design pattern that we chose for our application was the Factory Design Pattern. We believed that this pattern would be the best fit for our bucket and item creation. Our thoughts for why this is the right pattern to use is that the buckets and items that are created within our application follow the same design concept but differ with each implementation. The Factory pattern is perfect for implementing this concept because it allows for a base object to be used in which other objects implement, these objects are then able to redefine some of the methods needed to make the object unique. Our buckets and lists follow this structure, all buckets contain an array list of items and they have a description as well as methods to add and delete items. Buckets created by the user will follow this structure but the user will be able to create the description so this will be overridden. This also works the same way for items which have descriptions and then the ability to create deadlines and add photos. We do not want all of the descriptions to be the same though so that is why we use the Factory pattern, this will be overridden so that the user can create descriptions that identify the specific item. In the class diagrams below there are two buckets and two items for each respective diagram, we used two in each diagram for simplicity but in reality there can be as many buckets and items that implement the class that contains the base content.



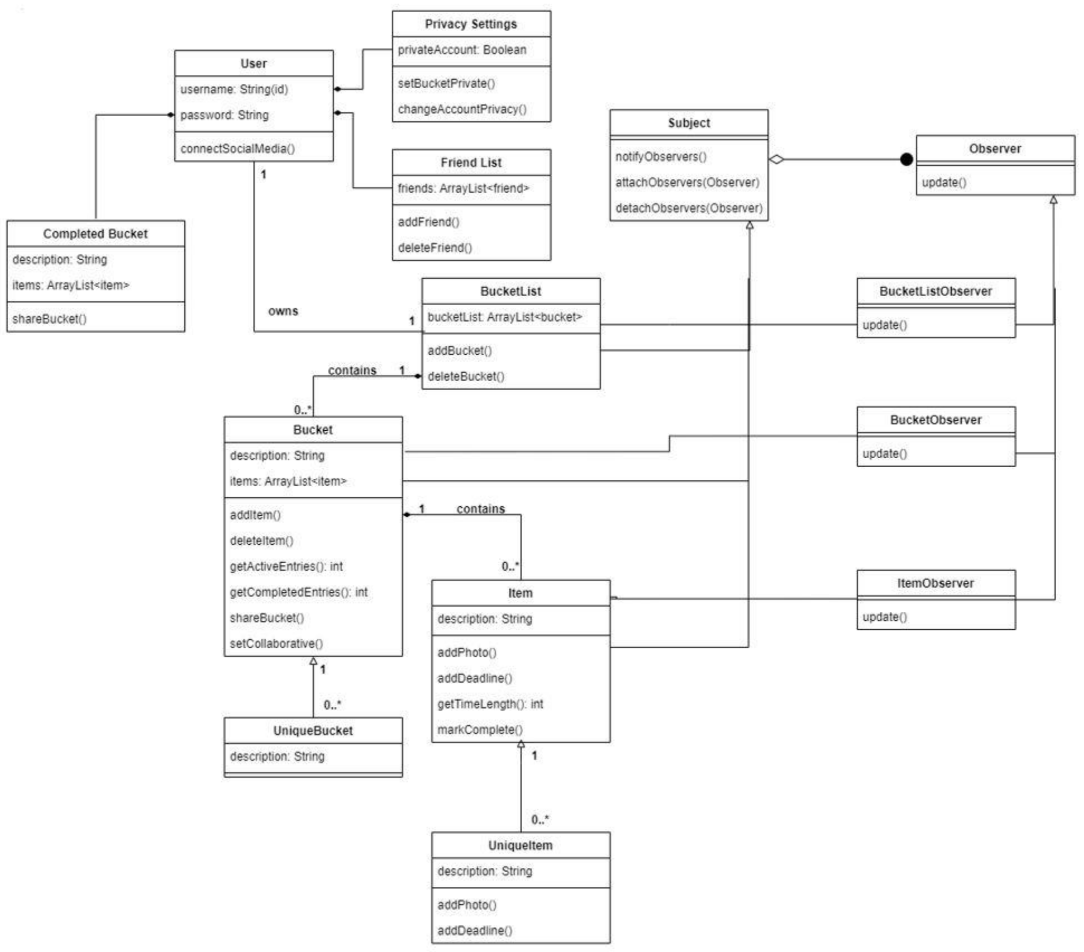


We then decided that another design pattern that would be very effective for our application would be the Observer pattern. This pattern would handle the updating of all of the lists and objects that are contained in the application. These lists and objects include the bucket list which holds all of the users buckets, the bucket which is a list of all of the items/tasks within that bucket, and then finally the item which contains a description and then optional photo and deadline. Our application uses the MVC architectural pattern and the observer design pattern is a crucial component of these architectures so that is why we decided to use it. The diagram below shows how all of these components would be updated using this pattern.


Subject
notifyObservers()
attachObservers(Observer)
detachObservers(Observer)
Observer
update()
BucketList
bucketList: ArrayList<bucket>
addBucket()
deleteBucket()
BucketListObserver
update()
Bucket
description: String
items: ArrayList<item>
addItem()
deleteItem()
BucketObserver
update()
Item
description: String
addPhoto()
addDeadline()
ItemObserver
update()

DCD

In our DCD we combined the two design patterns above along with some additional information to show a complete representation of how our application will function. In this diagram there are classes named ‘UniqueBucket’ and ‘UniqueItem’, these represent the buckets and items that will be created by users and that is why they are titled unique. We received points off on the DCD for deliverable 5 because ‘UniqueBucket’ and ‘UniqueItem’ don’t appear in our code but they are a part of this application and are very important. The title that they have in this diagram is just used to identify these components but when they are actually implemented they are implemented as buckets and items with the unique name that the user decides to use. ‘Completed Bucket’ is also something that we received points off for since it wasn’t in our application but this component is a part of our application and has its own user story for it. This bucket is the bucket that will contain all of the items that have been completed.



Privacy Settings
privateAccount: Boolean
setBucketPrivate()
changeAccountPrivacy()
User
username: String(id)
password: String
connectSocialMedia()
Friend List
friends: ArrayList<friend>
addFriend()
deleteFriend()
Subject
notifyObservers()
attachObservers(Observer)
detachObservers(Observer)
Observer
update()
Completed Bucket
description: String
items: ArrayList<item>
shareBucket()
1
owns
1
BucketList
bucketList: ArrayList<bucket>
addBucket()
deleteBucket()
BucketListObserver
update()
contains
1
0..*
Bucket
description: String
items: ArrayList<item>
addItem()
deleteItem()
getActiveEntries(): int
getCompletedEntries(): int
shareBucket()
setCollaborative()
BucketObserver
update()
1
contains
0..*
Item
description: String
addPhoto()
addDeadline()
getTimeLength(): int
markComplete()
ItemObserver
update()
1
0..*
UniqueBucket
description: String
1
0..*
UniqueItem
description: String
addPhoto()
addDeadline()