OOTD (Outfit of the Day) Final Report

Team Members:
Allen Wang (aw28928), Qing Wang (qw2328), Chang Park (cwp639)

Repository URL:
https://github.com/allenwang28/ootd

1. Motivation

Everyone has a different ritual to start off his or her day, all of which require figuring out what to wear. To make this task simpler, our application will act as a virtual helper for users to decide what outfit to wear on a given day. There are many people who are indecisive when making these decisions and try on countless outfits every morning. We want to take all the guesswork out of picking out what our user needs to wear for the day, with all the important factors such as the cleanliness of clothes, the weather outside, and color coordination already accounted for. There are hundreds of applications out there that help people their days hour by hour and task by task, but there is a lack of applications that help people organize their clothing.

2. User Benefits

The main benefits that OOTD hopes to add for a user is a combination of organization and convenience. This application keeps track of a person’s whole closet and helps make decisions to save users’ time every morning, or every time they go out and change. Since some people are very disorganized, this could be a necessary application for organizing and keeping track of their clothes to help them dress better and faster.

An active user should be able to save 5 - 10 minutes every morning by using OOTD. Even before getting out of bed, a user can see how weather information and generate a reasonable outfit to wear for the day out of the available and clean clothes. Using our algorithm for clothing matching , he or she won’t have to worry about color coordination, because they can trust that the clothing will look good together. It is one less thing to worry about in the morning. If a user really likes a generated outfit in the application, he or she can even save that outfit and have the option to wear it again in the future.

Another user benefit is simply being able to keep track of all the clothes owned. For instance, if a user tends to purchase high volumes of clothes or likes to trade and sell clothing, this application makes it easy to organize those clothes into categories and also add and remove them as often as they would like. Since OOTD keeps tracks of the color of clothing, a photo, and the many characteristics associated with each article of clothing as well as has a UI for adding/removing, this becomes a very efficient process.

3. Features and Requirements

For the app to be considered successful, the user must have the ability to add, edit, view, and remove clothing that they own. For each article of clothing, it must have information about the type of clothing, the name of the clothing, the color of the clothing, and a picture of the clothing. The app must take color combinations, weather information, and clothing cleanliness into consideration to create suggestions for the user. Finally, from the generated outfits, the user must have the ability to choose the outfit they decided to wear for the day, which informs the app of the level of cleanliness of each article of clothing.

After meeting these goals, more features were added. Clothing objects now contain more information - the number of times the article of clothing was worn, the formality of the clothing, and a default icon if the user decides not to provide a picture. The app is also now capable of adding and storing multiple generated outfits. From this list, the user is able to select which outfit they wore for the day. Furthermore, the user has the ability to have multiple closets based off of a name. Finally, the app now displays the weather information as well on the main screen.

4. Design

OOTD is an Android application, so it is designed with Activity and View classes. Activities have buttons that lead to other activities. Activities can also have zero, one, or more views that are put on top of these activities for customization of the display. Each article of clothing, which has characteristics, a color, and a picture, is made into an object which is put in a closet object. In addition, this closet object will hold any outfits that have been generated in the past.

Most of the functionality of our application runs through our main page (MainActivity). Weather for a certain zip code is displayed, as well as the current outfit and a scrolling preview of outfits and clothes in the closet. There are two horizontal scrolling RecyclerViews to show past outfits, and clothing in the current closet. Buttons on this page will start different activities across the application.

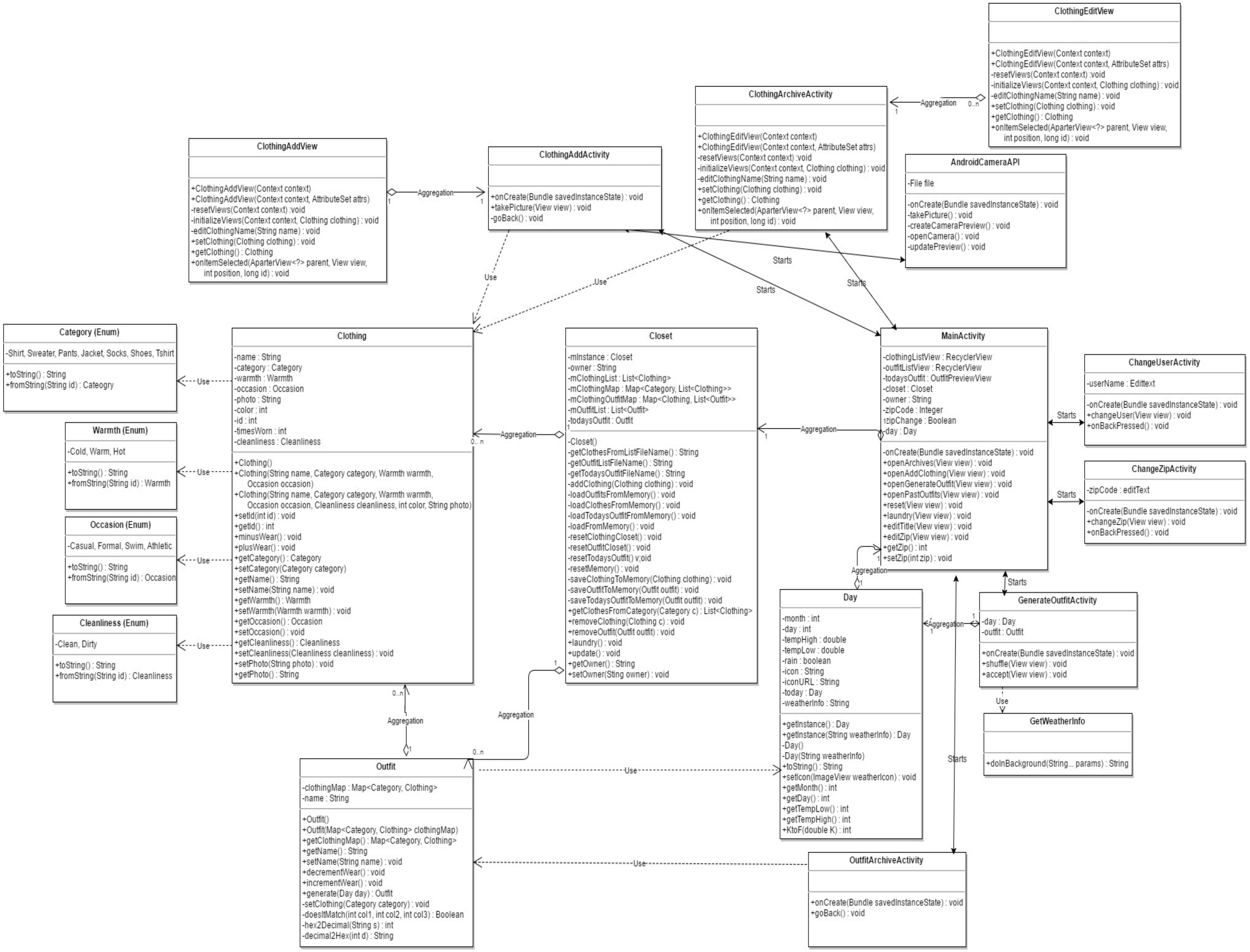
Each clothing object has unique characteristics. Of these, some are represented as Enums so that there are a limited number of choices, including: category (shirt, pants, etc.), warmth, occasion (formal, casual, etc.), and cleanliness. Clothing objects are created in the ClothingAddActivity, and through the ClothingAddView all characteristics of the clothing are chosen, and there is a button to the AndroidCameraAPI activity to take a picture. We used a revised version of sample code written by Google to take pictures using Android's camera2 library. The pictures taken are stored in external storage on the phone outside the application so that they can be used in other applications. The color of the clothing is chosen on a color wheel view from an imported library called ColorPickerView. Unlike the camera, we import this color wheel in the gradle manually and simply use its functions to include it in our application. Clothing will be added to the closet upon completion, or discarded.

The closet that holds outfits and clothes is stored in internal storage using input and output streams into text files that can only be accessed within the app. Each different closet will store into different files. There are many functions to remove or add outfits and clothes either individually or all at once, and those will similarly change those files in internal memory.

Weather is determined by an external library called OpenWeatherMap. We had to make an account to use it for up to 7200 basic calls per day up to a certain limit. Our application keeps this information in a Singleton Day object that holds a date, a low and high temperature, and a weather status. This is displayed for the user on the main activity but is also taken into account for generating outfits for the day along with clothing characteristics. This library is pulled with JSON objects and arrays, and uses a class called GetWeatherInfo to put into this day object. The day object is only updated once at application startup as well as when the location is changed, regardless of how many times we return to the main activity.

The ClothingArchiveActivity holds a ViewPager holding a carousel view of ClothingEditViews that can all be changed or removed. OutfitArchiveActivity does a similar thing with outfits. There are also activities for changing zip code and closet name, which will then refresh and go back to an updated main activity. Finally, we decided to include a button for doing laundry that simply refreshes the main activity and changes the timesWorn for every clothing in the closet to 0.

Below we have included the full class diagram with all the class variables and specific methods, as well as their connections. The dotted lines show a class using another, and solid (usually double-sided) lines with “Starts” shows activities that can reach another. There are also many aggregation relationships between classes shown with a solid line and a diamond shape. Every class with “Activity” in the name (plus the AndroidCameraAPI) extend the AppCompatActivity class and every class with the name View extends the LinearLayout class, but these are not explicitly shown in the class diagram for simplicity.



```
ClothingEditView
---
+ClothingEditView(Context context)
+ClothingEditView(Context context, AttributeSet attrs)
-resetViews(Context context) :void
-initializeViews(Context context, Clothing clothing) : void
-editClothingName(String name) : void
+setClothing(Clothing clothing) : void
+getClothing() : Clothing
+onItemSelected(AparterView<?> parent, View view,
    int position, long id) : void
```

```
ClothingArchiveActivity
---
+ClothingEditView(Context context)
+ClothingEditView(Context context, AttributeSet attrs)
-resetViews(Context context) :void
-initializeViews(Context context, Clothing clothing) : void
-editClothingName(String name) : void
+setClothing(Clothing clothing) : void
+getClothing() : Clothing
+onItemSelected(AparterView<?> parent, View view,
    int position, long id) : void
```

```
ClothingAddView
---
+ClothingAddView(Context context)
+ClothingAddView(Context context, AttributeSet attrs)
-resetViews(Context context) :void
-initializeViews(Context context, Clothing clothing) : void
-editClothingName(String name) : void
+setClothing(Clothing clothing) : void
+getClothing() : Clothing
+onItemSelected(AparterView<?> parent, View view,
    int position, long id) : void
```

```
ClothingAddActivity
---
+onCreate(Bundle savedInstanceState) : void
+takePicture(View view) : void
-goBack() : void
```

```
AndroidCameraAPI
-File file
---
-onCreate(Bundle savedInstanceState) : void
-takePicture() : void
-createCameraPreview() : void
-openCamera() : void
-updatePreview() : void
```

```
Category (Enum)
-Shirt, Sweater, Pants, Jacket, Socks, Shoes, Tshirt
---
+toString() : String
+fromString(String id) : Cateogry
```

```
Warmth (Enum)
-Cold, Warm, Hot
---
+toString() : String
+fromString(String id) : Warmth
```

```
Occasion (Enum)
-Casual, Formal, Swim, Athletic
---
+toString() : String
+fromString(String id) : Occasion
```

```
Cleanliness (Enum)
-Clean, Dirty
---
+toString() : String
+fromString(String id) : Cleanliness
```

```
Clothing
-name : String
-category : Category
-warmth : Warmth
-occasion : Occasion
-photo : String
-color : int
-id : int
-timesWorn : int
-cleanliness : Cleanliness
---
+Clothing()
+Clothing(String name, Category category, Warmth warmth,
    Occasion occasion)
+Clothing(String name, Category category, Warmth warmth,
    Occasion occasion, Cleanliness cleanliness, int color, String photo)
+setId(int id) : void
+getId() : int
+minusWear() : void
+plusWear() : void
+getCategory() : Category
+setCategory(Category category)
+getName() : String
+setName(String name) : void
+getWarmth() : Warmth
+setWarmth(Warmth warmth) : void
+getOccasion() : Occasion
+setOccasion() : void
+getCleanliness() : Cleanliness
+setCleanliness(Cleanliness cleanliness) : void
+setPhoto(String photo) : void
+getPhoto() : String
```

```
Closet
-mInstance : Closet
-owner : String
-mClothingList : List<Clothing>
-mClothingMap : Map<Category, List<Clothing>>
-mClothingOutfitMap : Map<Clothing, List<Outfit>>
-mOutfitList : List<Outfit>
-todaysOutfit : Outfit
---
-Closet()
-getClothesFromListFileName() : String
-getOutfitListFileName() : String
-getTodaysOutfitFileName() : String
-addClothing(Clothing clothing) : void
-loadOutfitsFromMemory() : void
-loadClothesFromMemory() : void
-loadTodaysOutfitFromMemory() : void
-loadFromMemory() : void
-resetClothingCloset() : void
-resetOutfitCloset() : void
-resetTodaysOutfit() v;oid
-resetMemory() : void
-saveClothingToMemory(Clothing clothing) : void
-saveOutfitToMemory(Outfit outfit) : void
-saveTodaysOutfitToMemory(Outfit outfit) : void
+getClothesFromCategory(Category c) : List<Clothing>
+removeClothing(Clothing c) : void
+removeOutfit(Outfit outfit) : void
+laundry() : void
+update() : void
+getOwner() : String
+setOwner(Sting owner) : void
```

```
MainActivity
-clothingListView : RecyclerView
-outfitListView : RecyclerView
-todaysOutfit : OutfitPreviewView
-closet : Closet
-owner : String
-zipCode : Integer
+zipChange : Boolean
-day : Day
---
-onCreate(Bundle savedInstanceState) : void
+openArchives(View view) : void
+openAddClothing(View view) : void
+openGenerateOutfit(View view) : void
+openPastOutfits(View view) : void
+reset(View view) : void
+laundry(View view) : void
+editTitle(View view) : void
+editZip(View view) : void
+getZip() : int
+setZip(int zip) : void
```

```
ChangeUserActivity
-userName : Edittext
---
-onCreate(Bundle savedInstanceState) : void
+changeUser(View view) : void
+onBackPressed() : void
```

```
ChangeZipActivity
-zipCode : editText
---
-onCreate(Bundle savedInstanceState) : void
+changeZip(View view) : void
+onBackPressed() : void
```

```
Day
-month : int
-day : int
-tempHigh : double
-tempLow : double
-rain : boolean
-icon : String
-iconURL : String
-today : Day
-weatherInfo : String
---
+getInstance() : Day
+getInstance(String weatherInfo) : Day
-Day()
-Day(String weatherInfo)
+toString() : String
+setIcon(ImageView weatherIcon) : void
+getMonth() : int
+getDay() : int
+getTempLow() : int
+getTempHigh() : int
+KtoF(double K) : int
```

```
GenerateOutfitActivity
-day : Day
-outfit : Outfit
---
+onCreate(Bundle savedInstanceState) : void
+shuffle(View view) : void
+accept(View view) : void
```

```
GetWeatherInfo
---
+doInBackground(String... params) : String
```

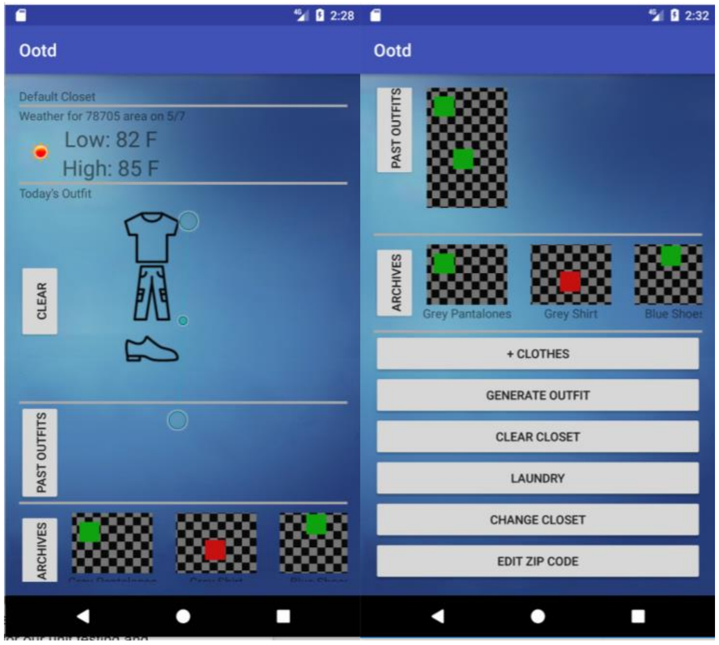
```
Outfit
-clothingMap : Map<Category, Clothing>
-name : String
---
+Outfit()
+Outfit(Map<Category, Clothing> clothingMap)
+getClothingMap() : Map<Category, Clothing>
+getName() : String
+setName(String name) : void
+decrementWear() : void
+incrementWear() : void
+generate(Day day) : Outfit
-setClothing(Category category) : void
-doesItMatch(int col1, int col2, int col3) : Boolean
-hex2Decimal(String s) : int
-decimal2Hex(int d) : String
```

```
OutfitArchiveActivity
---
+onCreate(Bundle savedInstanceState) : void
+goBack() : void
```

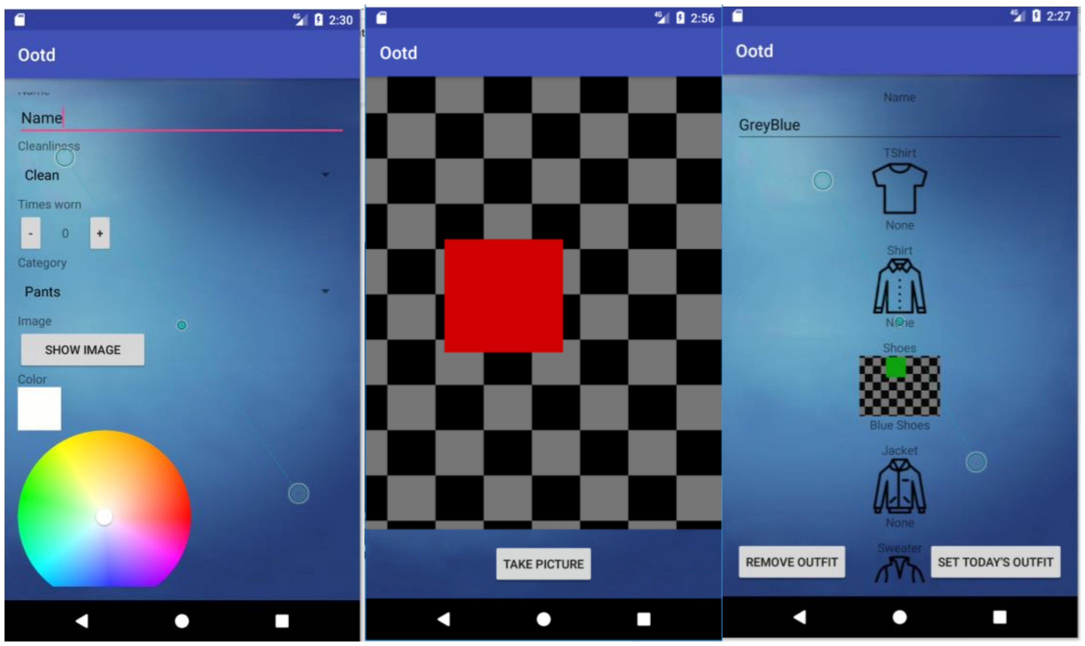
5. User Interface

This application was designed to be as easy to use as possible. Because we centered the design around one main screen, no function will be more than 2 clicks away for the user after starting up the application. We used a neutral, blue background across the app that will not distract the user or strain their eyes. Buttons were enlarged so that the user will have no problem reading the text on them and so that the chances of accidentally clicking the wrong button will be minimized.

On the main activity, at the very top you can see which closet you are viewing. Under that, you can see the weather for a certain zip code on the current day, which features a changing weather icon and a high/low temperature. Below that, we have three sections which are today's outfit, a scrolling view of past outfits, and a scrolling view of all clothes in this closet. There are a number of buttons below that which lead to the various functions. We wanted to have this information all in front of the user for maximum convenience. The two screenshots below show the main activity page.



The other relevant UI portions of the application would be the clothing add/edit view and the outfit generation view. The clothing add view has an EditText for the name, multiple Spinners (drop-down menus) for characteristics, a color wheel, and buttons for taking a picture/adding the clothing. For the outfit generation, we have all the default clothing icons listed vertically relative to their position on a person's body. By clicking a shuffle button, the user will have an auto-generated outfit with the pieces replacing the icons as necessary. There is a text box to set the name for an outfit, and buttons to remove/add an outfit or set it as the one that will be worn for the day. The screenshots for the clothing add/edit, taking pictures, and outfit generation respectively are shown below.


Name
Cleanliness: Clean
Times worn: 0
Category: Pants
Image
Color

Name: GreyBlue
TShirt: None
Shirt: None
Shoes: Blue Shoes
Jacket: None
Sweater

Imagine John, a busy young professional who has a hard time figuring out what to wear in the morning. He can open OOTD and check out the weather for the day and look through all this clothes at a glance. John will scroll through his past outfits to see if there's any he wants to wear today, but ultimately decides to generate a new outfit instead. This application can let John do all of that in less than a minute. Now, imagine that he is staying at home with his parents. He can quickly change the zip code to update the weather and change the closet to show what clothes he has stored at their place. He may purchase some items while staying at home, and can easily add those to his closet database from the app as well.

6. Testing

Since our project is an Android application, we will be using some of the Android-specific automated testing suites. Our project will use Android JUnit test classes including both unit tests and instrumented tests to confirm the methods of each class function for our unit testing and integration testing. For UI and Activity functionality, we will use the Espresso plugin which more thoroughly covers integration and system testing across the entire app. Finally, we will use GenyMotion and test our application across multiple types of Android devices and Android APK's as a means of system testing.

6.1 Junit Tests

For our Java-specific classes, we will use JUnit unit tests to automate our testing. For each of our classes, we will test the logic using normal cases as well as boundary conditions. This will cover the unit testing portion of our project. For instance, for a Clothing class that holds data about the specific clothing, we will test the getter/setter methods. The only logic that we have would be incrementing or decrementing the number of times the clothing was worn, so we will need to test that it does not go into a negative value.

In terms of JUnit integration testing, we will use the JUnit instrumented tests to run parts of our application and see how some of the buttons and features work together. These tests, which focus on outfit generation and closet databasing for multiple users, will cover multiple classes and make sure that they are functional when they interact. We will write JUnit tests to ensure path coverage of all possible permutations of weather and clothing characteristics in order to ensure that the outfit generation will always create an outfit that is suitable for the weather, and that every item in the outfit is clean. For instance, if it is cold and rainy on a specific day, we would want the outfit generated for that day to include both cold weather items such as a sweater and pants, as well as items for rain such as a rain jacket. In addition, we will test every branch in each method of our Closet class, which holds multiple Clothing objects and provides methods for altering and viewing clothes, and works with the app's internal storage when saving clothes. Below is a excerpt of a Closet class test:

```
@Before
public void setUp() throws Exception {
   closet = Closet.getInstance();
   closet.reset();
}

@Test
public void addSingleClothing() {
   Clothing clothing = new Clothing("pantaloons", Category.PANTS, Warmth.HOT,
Occasion.CASUAL, Cleanliness.DIRTY1, Color.RED,  "pantaloons.jpg");

   closet.saveClothing(clothing);
   List<Clothing> list = closet.getClothingList();
   assertTrue(list.size() == 1);
   Clothing _clothing = list.get(0);
   assertEquals("pantaloons", _clothing.getName());
   assertEquals(Category.PANTS, _clothing.getCategory());
   assertEquals(Warmth.HOT, _clothing.getWarmth());
   assertEquals(Occasion.CASUAL, _clothing.getOccasion());
   ...
}
```

6.2 Espresso Tests

Espresso is a native testing framework within Android Studio that has a library that sets up front end and user interface tests through simulated user interactions within a single target application. We will use Espresso to supplement our project's integration testing. We will be able to use these tests in order to verify that the UI elements that we want to create exist and are indeed at the correct positions. We will also be able to assert that pressing specific buttons affect the correct objects by checking that the values match in our views. Finally, we will be able to test our imported libraries that we use within our application since we do not have the source code for each method in our application. All of these testing functions contrast to the JUnit tests because they are black-box testing methods, which only affect the physical inputs that Espresso will provide and the outputs that arise.

We will confirm that the methods we tested with JUnit have access to each other and are called correctly with a variety of execution path tests through our application. We can run through the buttons throughout our application for adding/removing/editing clothing, going through our closet archives, creating outfits, etc. The most important part of this will be making sure that information is passed and updated when we travel from activity to activity and also that each link is functional and performs the correct action without crashing the application or going to an incorrect state. Ideally, we will go cover every prime path between activities, and branch coverage regarding buttons that remain on the same activity or view.

Three main libraries were either imported or copied into our application for which we will test primarily through Espresso, and very thoroughly (since we were not responsible for creating it). These are the Android camera2 for taking pictures, the ColorPicker color wheel, and the OpenWeatherMap weather library. We will make sure these are well integrated into our application and that we parse the correct information from them. This will be done by seeing that certain execution paths will not crash our application and by manually confirming the pictures attached to our clothes, the color setting that is selected, and the weather icons, respectively. Below is an except from a generated Espresso test for our MainActivity page:

```
@Test
public void mainActivityTestExists() {
       ViewInteraction button = onView(allOf(childAtPosition(childAtPosition(
IsInstanceOf.<View>instanceOf(android.widget.LinearLayout.class),5),0),isDisplayed()));
       button.check(matches(isDisplayed()));
       ViewInteraction button2 = onView(allOf(childAtPosition(childAtPosition(
IsInstanceOf.<View>instanceOf(android.widget.LinearLayout.class),5),1),isDisplayed()));
       button2.check(matches(isDisplayed()));
       …
}
```

6.3 GenyMotion Tests

GenyMotion is another application that exists primarily as a device emulator for Android. While Android Studio does provide its own set of devices that can be emulated, GenyMotion has a wider selection of Android devices to emulate across multiple Android application package (APK) versions. We used GenyMotion to test a subset of our execution paths we tested on Espresso (that goes through all basic functions) on all possible Android APK's and even for different types of devices. This way, we will confirm that we are compatible with the maximum possible number of users.

The maximum APK version that any of our Android imports or libraries require is APK 21, so we will stick with devices that exceed that APK version. Therefore, we have devices across APK versions 21, 22, 23, 24, and 25.

For our Android Studio tutorial during class, we used a Google Nexus 5 for our primary device throughout development. This was ideal because Google releases the most updated versions of the Nexus phones (5 - 9) for these emulations. However, we can also test on newer Samsung Galaxy phones (S6 - S8), as well as GenyMotion's custom Android phone and tablet devices with each APK version and without any company-specific bloatware.