660046669 – Mobile and Ubiquitous Computing
Coursework 1 – Weather App

Basic Structure & UI

Starting from what the user sees, we have three main Activities:

1. The “Current” weather activity
2. The “Forecast” activity
3. The “Settings” activity

The Current weather activity has two main blocks: the current, detailed weather; a summary of forecast for 3 (portrait) to 5 (landscape) days. The entire background colour of that activity changes based on what the current weather is (for example: yellow when sunny, bright yellow when overcast, grey for light rain, blue for rain, deep blue for night, …) The current weather is held within a “Constraint Layout” which changes based on the user’s device’s orientation to use the space better. However, the structure stays similar: the location of the weather displayed and the temperature on the first row, at opposite edges of the screen, the weather icon splitting them in the middle or coming down right under in the middle of the screen and right under a description of the weather. Finally, under comes additional details: the humidity, wind and pressure.

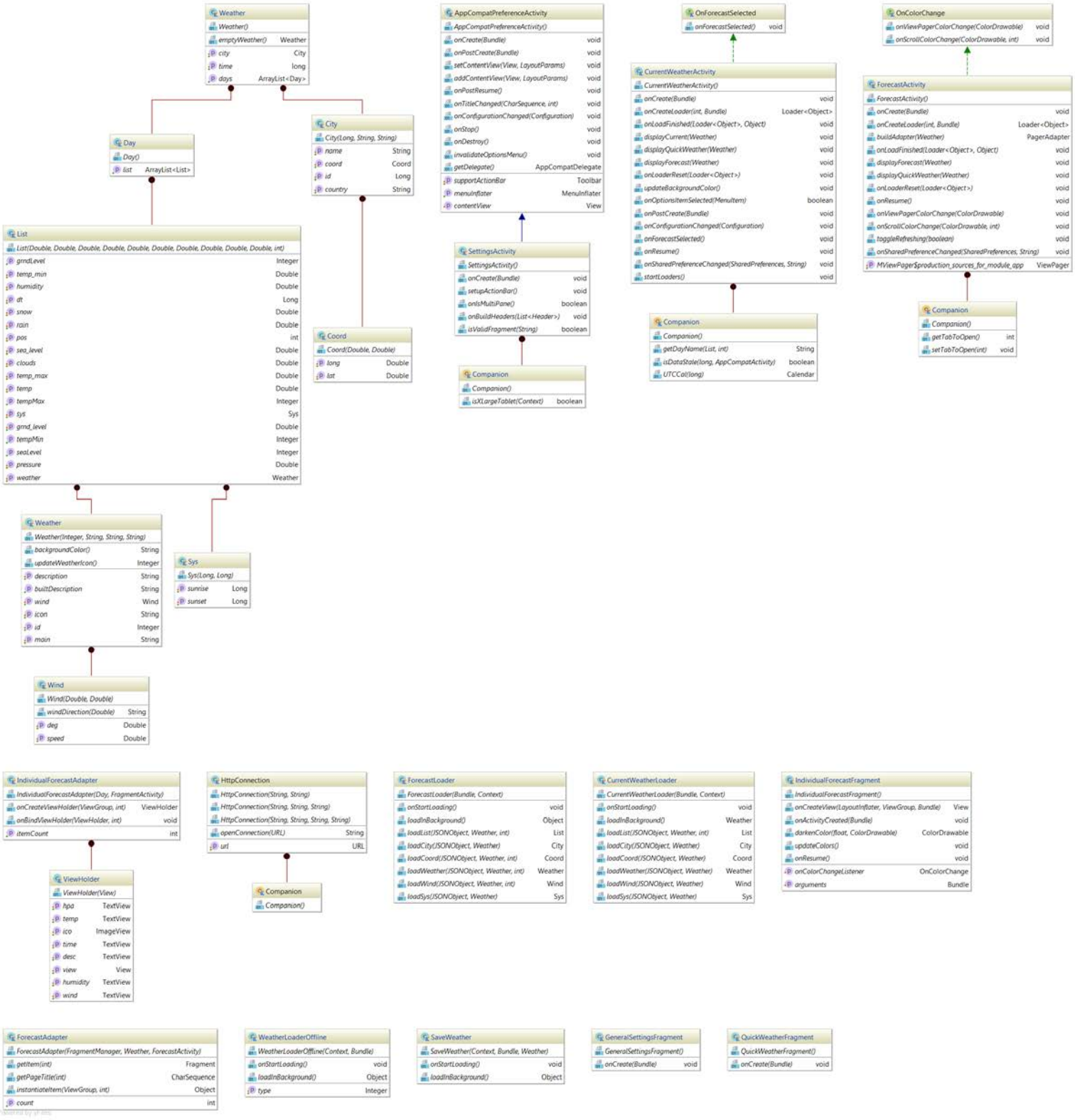
The second block is a summary of forecast. This second block is a horizontal Linear Layout, the “Forecast” block which itself contains vertical Linear Layouts, the “Day” blocks. Each “Day” has the day at the top and the weather from now (for the first day)/midnight (for the rest) until 21h (the last data available for that day). Since this is only a summary of the forecast, it only contains the time of the data forecasted, the icon representing the weather as well as temperature for that 3-hour block. Touching any of those blocks brings us to the second activity: Forecast.

The Forecast activity is made of a set of tabs with each tab holding a different day, starting from the current day until the last day of available data. In each tab is a vertically arranged set of cards in a Card Layout with each card containing detailed data for a specific 3-hour block, like the current data. The location is not displayed since the main screen is the “Current” activity which does contain the location and it is assumed users will see that page first. However, the rest of the data such as the weather icon, time, temperature, humidity, wind and pressure are all available. Again, the layout changes in a minor way to make better use of space in landscape orientation. The background of each tab is a gradient of the “weather colour” of the first card all the way to that of that last card, changing for each one. The Action Bar and the status bar colours change to reflect the colour of the first fully visible card.

The navigation drawer also has 2 sections: the header which has what I call “Quick Weather” and the different navigation options. “Quick Weather” is simply a secondary location that the user chooses which shows quick, “at-a-glance” summary information about the current weather in that location. The different navigation options are the “Current”, “5-day forecast” and “Settings” which take the user to the right activity and highlight based on the current selection

Finally, we have the Setting activity which takes the user to a screen containing two options: either “General” or “Quick Weather” which take the user to either the General settings for the app which means changing the location of the current/forecast weather, or the Quick Weather which means changing the location of the “Quick Weather” weather in the navigation drawer.

From those 3 different activities, we get quite a busy UML diagram:


Weather
Weather()
emptyWeather() Weather
city City
time long
days ArrayList<Day>
Day
Day()
list ArrayList<List>
City
City(Long, String, String)
name String
coord Coord
id Long
country String
List
List(Double, Double, Double, Double, Double, Double, Double, Double, Double, Double, int)
grndLevel Integer
temp_min Double
humidity Double
dt Long
snow Double
rain Double
pos int
sea_level Double
clouds Double
temp_max Double
temp Double
tempMax Integer
sys Sys
grnd_level Double
tempMin Integer
seaLevel Integer
pressure Double
weather Weather
Coord
Coord(Double, Double)
long Double
lat Double
Weather
Weather(Integer, String, String, String)
backgroundColor() String
updateWeatherIcon() Integer
description String
builtDescription String
wind Wind
icon String
id Integer
main String
Sys
Sys(Long, Long)
sunrise Long
sunset Long
Wind
Wind(Double, Double)
windDirection(Double) String
deg Double
speed Double
AppCompatPreferenceActivity
AppCompatPreferenceActivity()
onCreate(Bundle) void
onPostCreate(Bundle) void
setContentView(View, LayoutParams) void
addContentView(View, LayoutParams) void
onPostResume() void
onTitleChanged(CharSequence, int) void
onConfigurationChanged(Configuration) void
onStop() void
onDestroy() void
invalidateOptionsMenu() void
getDelegate() AppCompatDelegate
supportActionBar Toolbar
menuInflater MenuInflater
contentView View
SettingsActivity
SettingsActivity()
onCreate(Bundle) void
setupActionBar() void
onIsMultiPane() boolean
onBuildHeaders(List<Header>) void
isValidFragment(String) boolean
Companion
Companion()
isXLargeTablet(Context) boolean
OnForecastSelected
onForecastSelected() void
CurrentWeatherActivity
CurrentWeatherActivity()
onCreate(Bundle) void
onCreateLoader(int, Bundle) Loader<Object>
onLoadFinished(Loader<Object>, Object) void
displayCurrent(Weather) void
displayQuickWeather(Weather) void
displayForecast(Weather) void
onLoaderReset(Loader<Object>) void
updateBackgroundColor() void
onOptionsItemSelected(MenuItem) boolean
onPostCreate(Bundle) void
onConfigurationChanged(Configuration) void
onForecastSelected() void
onResume() void
onSharedPreferenceChanged(SharedPreferences, String) void
startLoaders() void
Companion
Companion()
getDayName(List, int) String
isDataStale(long, AppCompatActivity) boolean
UTCCal(long) Calendar
OnColorChange
onViewPagerColorChange(ColorDrawable) void
onScroll(ColorChange(ColorDrawable, int) void
ForecastActivity
ForecastActivity()
onCreate(Bundle) void
onCreateLoader(int, Bundle) Loader<Object>
buildAdapter(Weather) PagerAdapter
onLoadFinished(Loader<Object>, Object) void
displayForecast(Weather) void
displayQuickWeather(Weather) void
onLoaderReset(Loader<Object>) void
onResume() void
onViewPagerColorChange(ColorDrawable) void
onScroll(ColorChange(ColorDrawable, int) void
toggleRefreshing(boolean) void
onSharedPreferenceChanged(SharedPreferences, String) void
MViewPager$production_sources_for_module_app ViewPager
Companion
Companion()
getTabToOpen() int
setTabToOpen(int) void
IndividualForecastAdapter
IndividualForecastAdapter(Day, FragmentActivity)
onCreateViewHolder(ViewGroup, int) ViewHolder
onBindViewHolder(ViewHolder, int) void
itemCount int
ViewHolder
ViewHolder(View)
hpa TextView
temp TextView
ico ImageView
time TextView
desc TextView
view View
humidity TextView
wind TextView
HttpConnection
HttpConnection(String, String)
HttpConnection(String, String, String)
HttpConnection(String, String, String, String)
openConnection(URL) String
url URL
Companion
Companion()
ForecastLoader
ForecastLoader(Bundle, Context)
onStartLoading() void
loadInBackground() Object
loadList(JSONObject, Weather, int) List
loadCity(JSONObject, Weather) City
loadCoord(JSONObject, Weather, int) Coord
loadWeather(JSONObject, Weather, int) Weather
loadWind(JSONObject, Weather, int) Wind
loadSys(JSONObject, Weather) Sys
CurrentWeatherLoader
CurrentWeatherLoader(Bundle, Context)
onStartLoading() void
loadInBackground() Weather
loadList(JSONObject, Weather, int) List
loadCity(JSONObject, Weather) City
loadCoord(JSONObject, Weather) Coord
loadWeather(JSONObject, Weather) Weather
loadWind(JSONObject, Weather) Wind
loadSys(JSONObject, Weather) Sys
IndividualForecastFragment
IndividualForecastFragment()
onCreateView(LayoutInflater, ViewGroup, Bundle) View
onActivityCreated(Bundle) void
darkenColor(float, ColorDrawable) ColorDrawable
updateColors() void
onResume() void
onColorChangeListener OnColorChange
arguments Bundle
ForecastAdapter
ForecastAdapter(FragmentManager, Weather, ForecastActivity)
getItem(int) Fragment
getPageTitle(int) CharSequence
instantiateItem(ViewGroup, int) Object
count int
WeatherLoaderOffline
WeatherLoaderOffline(Context, Bundle)
onStartLoading() void
loadInBackground() Object
type Integer
SaveWeather
SaveWeather(Context, Bundle, Weather)
onStartLoading() void
loadInBackground() Object
GeneralSettingsFragment
GeneralSettingsFragment()
onCreate(Bundle) void
QuickWeatherFragment
QuickWeatherFragment()
onCreate(Bundle) void

Storing and Retrieving Weather Data

Starting from the top left of the UML diagram, we have the "Weather" class. This class helps us store downloaded current weather/forecast data in a way that makes it easy to:

1. Store it in the database,
2. Retrieve it from the database
3. Display it in an efficient manner.

This class also allows to us to store methods to process the data as needed. Its structure however might seem complicated. At the base of the class, we have 3 fields: "time", a Long which stores the time at which this data was retrieved, "city" a City class which stores information about the city where the data comes from, and "days", an ArrayList of type Day class which stores the different days of data available. This class also contains a method "emptyWeather()" which we will encounter later.

The City class contains an instance of Coord class, an "id" of type Long as well as the name and country of the city, both type String. The Coord class simply contains two Longs for longitude and latitude.

Back in the Weather class, we have an ArrayList of Days. The class Day represents one Day of data, meaning each of the blocks of data in each instance of Days are from the same Calendar day. The Day class only contains a ArrayList of type List class.

The List class is where most of the data is contained. This class contains another (confusingly named) Weather class, a Sys class, a Long which holds what time the data represents and series of Doubles for the temperature, min and max, pressure, at ground level and sea level, humidity, clouds, rain and snow. The Sys class holds two Longs with the data for sunrise and sunset. The inner Weather class is made up of one more class, Wind, as well as an Integer id, and String main, description and icon. It also has two methods for determining the background colour associated with a 3-hour block as well as the weather icon to use. Finally, the Wind class has two Doubles to store speed and direction of the wind, as well as methods to convert the direction from degrees to a bearing and convert the speed from m/s to km/h. While this structure might seem very convoluted to an outsider, it makes parsing the data, storing it, retrieving it and displaying it a much easier job since we are able to iterate over days and hours and retrieve only the data we need. It also allows us to bind methods to data and easily fetch "pre-formatted" data.

Weather Loaders

The project has a 3 main loader class: the CurrentWeatherLoader, the ForecastLoader and the WeatherOfflineLoader. The first two are quite similar in their function: retrieve the data from the API as JSON strings, parse through the strings and build a Weather object. The only difference between the two is that the first uses the "current" API and creates only one day with one data block in the Weather object, whereas ForecastLoader uses the "forecast" API and creates as many days and as many data blocks in each day as needed to fit all the data. WeatherOfflineLoader is multi-purpose and allows to fetch either Current Weather data or Forecast data from the database.

Activity

1. The Current and Forecast Activities start out in similar ways, in order:
2. Set the content view (the layout for the Activity)
3. Add the custom toolbar and the navigation drawer
4. Add the animated Menu button to the toolbar and synchronise it with the drawer

5. Set the behaviour of each navigation option in the drawer
6. Initialise the behaviour for pull-to-refresh (which we will talk about later)
7. Getting our views
8. Get our city/country in the settings, or default to Exeter, UK
9. Start the loaders

CurrentWeatherActivity

The first loader to fire in CurrentWeather is the QUICK_WEATHER_LOADER. Very simply, it loads the weather for Quick Weather in the navigation drawer and displays it, if no data is available (no internet connection), it will call display an "emptyWeather()". The second to fire is the CURRENT_WEATHER_OFFLINE whose behaviour is more elaborate: first the loader will try to fetch data from the database. This data contains a timestamp (the "time" attribute of Weather object). If there is no data available (first use, or the data has been removed), or if the data is too old (based on user settings), then a second loader will be started which goes fetch the latest data from the online API, stores it in the database and displays it. Should there be an error while fetching online data, such as no Internet connection or a failure of the API, the "emptyWeather()" method will be called. This method creates a Weather object with dummy data of no value (no location, 0 degrees, cloudy). We also display a "Snackbar" to tell the user no data is available. The behaviour is identical for the "FORECAST_LOADER_OFFLINE" which is fired after CURRENT_WEATHER_OFFLINE. This behaviour (first checking the database, then if data is stale or null, fetching from online, then displaying no data as failsafe) allows us to save on API key calls as well as preserving battery and user mobile data usage. The "Pull-to-Refresh" behaviour is inverse: it will first try to fetch latest data from online, then as backup go to the database and finally empty weather data if no data is available at all. This behaviour allows us to never have to show the user a blank screen, or crash if there is no available data.

The displayCurrent loads the data into the appropriate fields

The displayForecast iterates through the days and inflates a vertical Linear Layout, and then for each day iterates through the hours and inflates a Constraint Layout in which the appropriate fields are filled in.

ForecastActivity

The behaviour of the loaders in ForecastActivity is identical to the behaviour of the ones in CurrentWeatherActivity.

The process to display the data works like so: there is a ForecastAdapter which loads the appropriate tabs and their names based on the data in the Days list. Each tab is then a "IndividualForecastFragment" to which the data for the day is passed. Each IndividualForecastFragment then has its own IndividualForecastAdapter which takes in the days data and inflates each "hour card" for each block of data in the List array of Day. This design allows me to only create as many tabs and as many cards as is needed for the data that is available instead of having "empty cards" or "empty tabs" when not all the data is available.

SettingsActivity

The SettingsActivity which is-a PreferenceActivity. I therefore use the "onBuildHeaders" to load the different menus, which are linked to their respective "PreferenceFragment" (QuickWeatherFragment and GeneralSettingsFragment). Changing the preferences in either of these changes the corresponding setting in the SharedPreferences which are accessed by the classes/loaders to load the appropriate weather data.