Module: Mobile Application development (Android)

Session 50: Google maps, GPS, Camera Activity and File handling assignment Part 1 (Practice)

### Intro

### For this lab you will be developing a new GPS recording application called WalkAbout. The purpose of the application is to allow users to record their GPS location information as they travel. While the application records the user's GPS data, it displays it back to the user in the form of a path drawn on top of a Google Map. While recording data, the user can launch a Camera activity that will capture and store pictures on an SD-Card. When finished recording, the application gives the user the option of storing the current GPS data as a private application file to be loaded and displayed at a later time.

### Objectives

### At the end of this lab you will be expected to know:

### How to incorporate Google Maps into an application.

### How to register for and receive GPS location information.

### How to draw graphics on the screen using the Canvas class.

### How to create Google Maps Overlays.

### How to use the Camera.

### How to write data to the SD card.

### How to create and delete private application files.

### How to launch and receive results from Activities.

### Activities

### For this lab you will be working with a brand new application, completely independent of the previous labs. Over the course of the lab, you will be iteratively refining and adding functionality to the WalkAbout app. With each iteration you will be improving upon the previous iteration's functionality. You'll start by setting and familiarizing yourself with the Eclipse project. You will then register for a Google Maps API key and begin incrementally developing the main map-viewing Activity. These first few exercises will have you display a map and the user's current position. Next, you will add functionality to record the user's GPS location by registering for and receiving data from what is known as a the GPS Location Provider. After that, you will implement a Camera activity for taking pictures and saving them to the SD-Card. In the final section, you will save a user's GPS path to a file that is private to the application and allow the application to restore itself from the file as well.

### *IMPORTANT:*

### *You will be given a Skeleton Project to work with. This project contains all of the java and resource files you will need to complete the lab. Some method stubs, member variables, and resource values and ids have been added as well. It is important that you not change the names of these methods, variables, and resource values and ids. These are given to you because there are unit tests included in this project as well that depend on these items being declared exactly as they are. These units test will be used to evaluate the correctness of your lab. You have complete access to these test cases during development, which gives you the ability to run these tests yourself. In fact, you are encouraged to run these tests to ensure that your application is functioning properly.*

### 1. Setting Up...

### 1.1 Creating the Project

### To begin, you will need to download and extract the skeleton project for the WalkAbout application.

### [Click Here](https://sites.google.com/site/androidcoursearchive/resources/lab-skeletons/Lab5_v1_stub.zip?attredirects=0) to download the skeleton project.

### Extract the project, making sure to preserve the directory structure.

### Take note of the path to the root folder of the skeleton project.

### Next you will need to setup an Android project for this app.  Use the settings listed below for the remaining fields:

### Project Name:  lab5<userid>

### Build Target:  Goolge API's 1.6 (see [the lecture notes](http://www.google.com/url?q=http%3A%2F%2Fusers.csc.calpoly.edu%2F%257Edjanzen%2Fcourses%2F409W10%2Fpresentations%2FMaps.pdf&sa=D&sntz=1&usg=AFrqEze68NPzcZP7we5K4ZNk2J0iSW-bBQ) for further information)

### Application Name:  WalkAbout

### 1.2 Familiarize Yourself with the Project

### The project contains three java class files and a single XML layout file which you will have to implement. WalkAbout.java will contain the definition for the main WalkAbout Activity class. This is the class that will display the map and the user's recorded path. The WalkAbout class makes use of a very simple XML layout file called map\_layout.xml which you will have to fill in.

### From the WalkAbout Activity you will be able to launch the CameraPreview Activity, which will be defined in the CameraPreview.java file. The CameraPreview Activity class displays a live camera preview on the screen and will allow the user to capture and save a picture to the SD-Card. This file was borrowed from the Google API's Demo application and the logic to display the camera preview is already implemented for you. You will have to fill in the rest.

### The WalkAbout Activity class will rely on the PathOverlay class to draw the user's traveled path on its map. You will implement the logic needed for drawing the path in the PathOverlay.java file.

### A general class diagram for the project is depicted below. Classes you will be implementing are colored yellow, the Google Maps API classes that you will be working with are colored in blue, and the standard Android classes you should be familiar with are colored in white.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1266438315832/labs/lab-5/class-diagram.png?height=303&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-5/class-diagram.png?attredirects=0)

### 2. Using Google Maps

### In this section of the lab you will be working extensively with the Google Maps package. The Google Maps package allow you to include and manipulate Maps in your Android Applications. The general strategy for displaying a map in your application is to have an entire Activity dedicated to viewing a map. This activity must *extend*the [com.google.android.maps.MapActivity](http://code.google.com/android/add-ons/google-apis/reference/com/google/android/maps/MapActivity.html) class, which takes care of all the intricacies involved in setting up and tearing down the services required to support displaying a map.

### The actual map that gets displayed is an instance of the [com.google.android.maps.MapView](http://code.google.com/android/add-ons/google-apis/reference/com/google/android/maps/MapView.html) class, which extends the standard Android ViewGroup class. This class encapsulates all the gesture logic necessary for handling panning, zooming, and touching objects on the map. The map is just a Google map and can be displayed in three different modes: satellite, street, and traffic. In order to use this class you will need a Google Maps API Key.  We will go over how to do this later.

### By making use of the [com.google.android.maps.Overlay](http://code.google.com/android/add-ons/google-apis/reference/com/google/android/maps/Overlay.html) class you can *overlay*interesting data on top of your map. By either extending this base class or using other overlay classes you can add another interaction layer onto your map. The [com.google.android.maps.MyLocationOverlay](http://code.google.com/android/add-ons/google-apis/reference/com/google/android/maps/MyLocationOverlay.html) class is one such example; it can draw a beacon on the map to display the device's current position and it can draw a compass on the map to display the device's current heading.

### 2.1 Displaying a Map

### You will begin by implementing the functionality necessary to display a full-screen map in the WalkAbout Activity class. The layout for the WalkAbout Activity should be specified in the res/layout/map\_layout.xml file, which you will have to fill in. This will require you to use the com.google.android.maps.MapView class. In order to do this, you will have to register the debug keystore (that Eclipse uses to run your applications) with Google in order to receive a Maps API Key. When finished, your application should appear as depicted in the figure below:

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1266438428039/labs/lab-5/MapView.png?height=400&width=266](https://sites.google.com/site/androidcoursearchive/labs/lab-5/MapView.png?attredirects=0)

### For more information and examples on working with the Google Maps Package see the [documentation site](http://code.google.com/android/add-ons/google-apis/reference/com/google/android/maps/package-summary.html) and/or visit the Android Developer Tutorial on the [Google Maps Package](http://developer.android.com/guide/tutorials/views/hello-mapview.html).

### 2.1.1 Get a Google Maps API Key.

### In order to use the Google Maps API's and classes you will have to register the keystore you use to sign your application with the Google Maps Service. Once your keystore is registered, you will be provided with a Google Maps API key that can be used with any application signed by your keystore.

### Every distribution of the Android Development Toolkit comes with a debug keystore that is used to sign your application when it is launched and run from Eclipse. This is how you are able to run your application on a device or emulator without signing it yourself. For the purposes of this lab, you need only to register your debug keystore. If you end up releasing an application to the market you will need to register an actual keystore.

### Retrieve the MD5 Fingerprint of your debug.keystore file:

### Follow the instructions [here](http://code.google.com/android/add-ons/google-apis/mapkey.html#getdebugfingerprint)

### Register the Fingerprint with Google Maps Service:

### Follow the instructions [here](http://code.google.com/android/add-ons/google-apis/mapkey.html#registering)

### Save the API Key as a resource string in your Application:

### Copy and Paste the API Key into the res/values/strings.xml file in between the mapApiKey string tags:

### <string name="mapApiKey">INSERT\_YOUR\_API\_KEY\_HERE</string>

### *Storing the API Key as a resource allows you to more easily change it in the future.*

### Your application must declare that it uses the Google Maps Library in its manifest. Do so by adding the the following <uses-library ... /> line to your AndroidManifest.xml file. This line should be nested inside of the <application></application> tags:

### <application ...>     <uses-library android:name="com.google.android.maps" /> </application>

### Your application must use an internet connection to retrieve map data so it must also declare that it uses the Internet in its manifest. Do so by adding the following <uses-permission ... /> line to your AndroidManifest.xml file. This line should be nested inside of the <manifest></manifest> tags:

### <manifest ...>

### <uses-permission android:name="android.permission.INTERNET"/>

### </manifest>

### 

### 2.1.2 Fill in the MapView XML Layout File

### The main WalkAbout MapActivity class uses a very simple layout consisting of a root ViewGroup that contains only a single MapView element. Implement this layout by filling in the map\_layout.xml file:

### Use whatever root ViewGroup you like to contain the MapView element.

### When adding the MapView element you need to use the fully qualified class name:

### <com.google.android.maps.MapView ... />

### IMPORTANT: You must give the MapView element an id of "m\_vwMap".

### The MapView should fill the entire contents of the screen.

### When using a MapView element you must give it your API Key. You do this by inserting a special android:apiKey attribute into the MapView element declaration.

### Set the attribute equal to the string resource that contains the API Key.

### android:apiKey="@string/mapApiKey"

### *This is how you reference resources from within a resource file.*

### Set the MapView to clickable by adding an android:clickable attribute with a value of true.

### 

### 2.1.3 Display the Map

### The WalkAbout MapActivity will display the MapView. All initialization relating to the layout should be done in WalkAbout.initLayout(), which gets called by onCreate():

### Inflate your map\_layout.xml file.

### Initialize the MapView m\_vwMap member variable by retrieving a reference to the MapView element in the XML layout file.

### Force the Zoom Controls to be displayed in by making a call to MapView.setBuiltInZoomControls(...) (hint: you don't actually type MapView).

### You should be able to run your application and see a Google Map. When you touch the map, the Zoom Controls should be displayed centered along the bottom of the screen.

### 

### 2.2 Adding Map Overlays

### As stated before, Overlays are used when you want to draw objects on top of a MapView. Such objects could represent pin points for restaurant locations, a route for displaying directions, or a simple logo. Each MapView instance is responsible for drawing the overlays that get displayed on top of it. As such, each MapView instance has a list of all Overlays that it should draw. When you want to add an Overlay to a MapView, you simply retrieve the MapView's list of Overlays and add to it.

### The Overlay objects are different from regular View objects in that they are not requested to be drawn by the system. Instead they are requested to be drawn by the MapView when the system requests that the MapView draw itself. As such, when you make changes to an Overlay, you cannot explicitly request that the overlay be re-drawn by calling a method like View.invalidate() or View.postInvalidate(). In order to get the Overlay objects to be redrawn you must call View.invalidate() or View.postInvalidate() on the MapView that owns the Overlay objects.

### You will now add an Overlay to your current MapView which will display the device's current location and heading on top of the MapView. When finished, your application should appear as depicted in the figure below:

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1266438624028/labs/lab-5/MyLocationOverlay.png?height=400&width=266](https://sites.google.com/site/androidcoursearchive/labs/lab-5/MyLocationOverlay.png?attredirects=0)

### 2.2.1 Display Your Location

### In order for your application to display your location, it must access Positioning data from a Location Provider. In order to do this, it must declare that it uses the ACCESS\_FINE\_LOCATION permission in its manifest.

### Do so by adding the following <uses-permission ... /> line to your AndroidManifest.xml file. This line should be nested inside of the <manifest></manifest> tags:

### <manifest ...>      <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/> </manifest>

### Enable the map to display your current location and heading by initializing and adding a com.google.android.maps.MyLocationOverlay to your MapView. This should be done in the WalkAbout.initLayout() method:

### Initialize m\_locationOverlay with a new MyLocationOverlay object.

### Enable the MyLocationOverlay object's Compass and MyLocation beacon by using the MyLocationOverlay.enableCompass() and MyLocationOVerlay.enableMyLocation() methods.

### It is important to note that the MyLocationOverlay depends on the GPS or Network Location settings being enabled. These methods will do nothing if you do not have either the Network or GPS Location setting enabled.

### Retrieve m\_vwMap's list of attached Overlay objects by calling its getOverlays() method.

### Add the m\_locationOverlay to the list of Overlay objects.

### You should be able to run your application and see a Google Map containing a beacon pointing out your current location and a compass displaying your current heading.

### 

### 2.3 Initialize the WalkAbout Options Menu

### The WalkAbout Activity has an Options Menu that will display five different MenuItems. Create the Options Menu as follows: Do not worry about setting OnMenuItemClickListeners yet, you will do that later.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1266438483846/labs/lab-5/WalkAboutOptionsMenu.png?height=400&width=266](https://sites.google.com/site/androidcoursearchive/labs/lab-5/WalkAboutOptionsMenu.png?attredirects=0)

### "Start/Stop": This MenuItem acts as a toggle for either starting or stopping GPS Location recording. This recording is a record of the path that you have traveled. While recording, the MenuItem should display "Stop."  While stopped, the MenuItem should display "Start."

### Initialize the text for the MenuItem with R.string.startRecording resource string.

### Later on, you will dynamically update the text to reflect the current recording state.

### Use WalkAbout.STARTSTOP\_MENU\_ITEM constant as the Id for the MenuItem.

### "Save": This MenuItem will save the current recorded path.

### Initialize the text for the MenuItem with R.string.save.

### Use the WalkAbout.SAVE\_MENU\_ITEM constant as the Id for the MenuItem.

### "Load": This MenuItem loads the last saved path.

### Initialize the text for the MenuItem with R.string.load.

### Use the WalkAbout.LOAD\_MENU\_ITEM constant as the Id for the MenuItem.

### "Take Picture": This MenuItem launches a Camera Activity that allows you to take pictures.

### Initialize the text for the MenuItem with R.string.takePicture.

### Use the WalkAbout.PICTURE\_MENU\_ITEM constant as the Id for the MenuItem.

### "Enable GPS": This MenuItem launches the Device Settings Activity that allows you to enable the GPS Provider.

### Initialize the text for the MenuItem with R.string.enableGPS.

### Use the WalkAbout.ENABLEGPS\_MENU\_ITEM constant as the Id for the MenuItem.

### 3. Using Location Services

### The Android System provides services for determining the current location of the device. The framework for working with these location based services lives under the [android.location](http://developer.android.com/reference/android/location/package-summary.html) package. A number of useful classes live inside this package:

### LocationManager: Provides an interface for the location based services. You will be interacting with this class most of the time when trying to obtain location information.

### LocationProvider: LocationProviders are classes that provide updates on the current location of the device. There exists a LocationProvider for each different technology that determines location. There exists a GPS LocationProvider and a Network LocationProvider. Each LocationProvider specifies criteria that must be satisfied in order for it to be used. For example, the GPS LocationProvider requires that the device have GPS hardware and that it be enabled.

### Criteria: This class allows you to programmatically outline criteria you would like from a LocationProvider. Such criteria may include accuracy, power consumption, altitude reporting, speed reporting, monetary cost, etc. You can then setup and give an instance of this class to the LocationManager and allow it to choose the LocationProvider that best matches your criteria.

### LocationListener: This is an interface that defines call backs for different events that are generated by LocationProviders. These can be registered with a particular LocationProvider to receive updates when the location changes or the state of the LocationProvider changes.

### Location:A data object containing location based information. Generally contains latitude and longitude, as well as a date and timestamp containing the time at which the location was determined. Might also contain elevation, speed, and heading information.

### Location based utility classes: A utility class for Geo-Coding which transforms addresses to GPS Coordinates and back, a data object class for addresses, and some classes for monitoring the status of the GPS satellites the device is locked onto.

### In general, you usually query the LocationManager for the information you are seeking. You can check the current status of a LocationProvider, you can check the last known location reported by a LocationProvider, and you can register to receive updates from a LocationProvider all through the LocationManager*(just to name a few)*. To receive updates on location information directly from a LocationProvider, you need to implement the LocationListener interface and register yourself with the LocationProvider.

### In the subsections that follow, you add functionality to monitor and enable the GPS LocationProvider in the WalkAbout Activity class. You will record changes in location as the user's path. You will then implement an Overlay class that will draw this path onto the WalkAbout Activity's MapView object.

### 3.1 Testing & Enabling the GPS LocationProvider

### It is entirely possible that the user has disabled the GPS hardware. Before you can monitor and record data from the GPS LocationProvider, the GPS hardware must be enabled. In this particular subsection, you will begin by querying the LocationManager for whether the GPS hardware is enabled on the device. If GPS is not enabled, you will allow the user to launch the Location Settings Activity to enable it from the "Enable GPS" Options MenuItem. After the user is done editing the Location Settings, they can return to the WalkAbout Activity by hitting the back button. When the GPS hardware is enabled the "Enable GPS" MenuItem will no longer be visible in the Options Menu.

### The following diagram depicts two use cases *(Click on the image to view it at full size)*. Both use cases have the application starting up without GPS hardware enabled and without the Network LocationProvider enabled. The first use case follows the dark green arrows along the top of the figure. The User clicks the menu button, then clicks the "Enable GPS" MenuItem. The Location Settings Activity launches, the user then decides not to enable the GPS. The user clicks the back button and is returned to the Walkabout Activity. The user clicks the Menu button, and the "Enable GPS" MenuItem is still visible. Notice that the Network and GPS Location Providers are disabled, that the MyLocationOverlay's Current Location Beacon is not present, and Compass heading location is inaccurate. This is because the MyLocation Overlay relies on the LocationProvider for displaying this information.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1266438518757/labs/lab-5/EnableGPS2.png?height=250&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-5/EnableGPS2.png?attredirects=0)

### The second use case follows the golden arrows along the bottom of the figure. The User clicks the menu button, then clicks the "Enable GPS" MenuItem. The Location Settings Activity launches, the user then decides to enable the GPS. The user checks the "Enable GPS satellites" check box *(Note that this option may be labeled different in different version of Android).* The user clicks the back button and is returned to the Walkabout Activity. The user clicks the Menu button, and the "Enable GPS" MenuItem is no longer visible. Notice that when the user returns to the WalkAbout Activity after enabling the GPS hardware, that the MyLocationOverlay's current Location Beacon is now present, and Compass heading location is now accurate. It is also important to note that if the GPS Hardware is enabled when the application starts, then the "Enable GPS" MenuItem should not be visible.

### 3.1.1 Initialize LocationManager

### In various parts of the WalkAbout class, you will request location information from the LocationManager attached to this application context. The WalkAbout class will store a reference to this LocationManager object in the m\_locationManager member variable for convenience. Initialize this member variable in the WalkAbout.initLocationData() method:

### Retrieve the LocationManager by calling the Activity.getSystemService(...) method, passing in the Context.LOCATION\_SERVICE constant.

### *Remember that this instance of the WalkAbout class is an Activity and therefore all public and protected Activity methods are members of this object.*

### set m\_locManager to the object returned to you. You will have to cast the return value, because getSystemService(...) returns an object of type Object.

### 

### 3.1.2 Dynamically Update Options Menu

### The Options menu should display the "Start/Stop" MenuItem as disabled if the GPS Location Provider *is disabled*. Additionally, the Options menu should NOT display the "Enable GPS" MenuItem at all if the GPS Location Provider *is enabled*.

### Each time the Options menu is displayed, you should check to see if the GPS Location Provider is enabled and update the "Start/Stop" and "Enable GPS" MenuItems accordingly.

### You must do this from the Activity.onPrepareOptionsMenu(...). If this is done from the Activity.onCreateOptionsMenu(...) it will only be done the first time the menu is shown. See the [Android Documentation on this method](http://developer.android.com/reference/android/app/Activity.html#onPrepareOptionsMenu%28android.view.Menu%29) for more details.

### You can determine whether a Location Provider is enabled by calling the LocationManager.isProviderEnabled(...) method and passing in the name of the Provider. See the [Android Documentation on this method](http://developer.android.com/reference/android/location/LocationManager.html#isProviderEnabled%28java.lang.String%29) for more details.

### The names of the different Providers are stored as string constants in the LocationManager class. Check the [LocationManager Constants Documentation](http://developer.android.com/reference/android/location/LocationManager.html" \l "isProviderEnabled%28java.lang.String%29) to determine which string to pass in.

### You should be able to run your application and test that the "Start/Stop" and "Enable GPS" MenuItems are enabled and invisible respectively when the GPS Provider *is enabled*. They should also be disabled and visible respectively when the GPS Provider *is disabled*.

### 

### 3.1.3 Implement "Enable  GPS" MenuItem

### When the GPS Location Provider is disabled, the user will be able to launch the settings activity to enable the GPS Location Provider from the "Enable GPS" Options MenuItem. Set and fill in the OnMenuItemClickListener for the "Enable GPS" MenuItem:

### Instantiate a new Intent object, passing in the Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS *Action* String constant.

### Call the Activity.startActivityForResult() method passing in the intent you just created and the WalkAbout.ENABLE\_GPS\_REQUEST\_CODE static constant.

### We pass in the Request Code so that when we handle the result, we know which request generated the result. The result of any Activity that we launch gets processed by the same method and the Request Code mechanism lets us know which result is coming from which Activity.

### You should be able to run your application and test that you can launch the settings activity to enable the GPS provider from the "Enable GPS" MenuItem. When you hit the Back button you should return to the WalkAbout Activity. However, you should note that the MyLocationOverlay still won't be working.

**3.1.4 Handle the Location Settings Activity Result**

After the user is done with the Location Settings Activity and returns to the WalkAbout Activity, you need to try re-enabling the MyLocationOverlay. Since you called the Activity.startActivityForResult(...) method to launch the Location Settings Activity, the WalkAbout Activity.onActivityResult(...) method will be called. The requestCode parameter in this method will contain the value that you passed into the startActivityForResult(...) method. You should test this requestCode parameter to see if it matches the WalkAbout.EBABLE\_GPS\_REQUEST\_CODE constant. You test the requestCode because this same method will be called when any Activity that you launch returns a result *(You must test this value because you will be launching another activity later and you want to be able to identify which Activity is returning a result)*.

Do this by overriding and filling in the Activity.onActivityResult(...) method:

* Make sure to call super.onActivityResult(...).
* Test the requestCode argument to ensure that it matches the request code that you used when you launched the CameraPreview activity.
  + If the request codes don't match, then don't do anything and just return. If they do match then continue on.
* Attempt to re-enable the Compass and MyLocation beacon as you did in section 2.2.1.

You should now run your application with all Location Providers disabled and test that the MyLocation Overlay is displayed properly once you enable the GPS LocationProvider.

### 