Android maintenance section.

Program description:

The WalkingTours application is designed to run on Android based mobile devices targeting the minimum API level 2.3 (also know as Gingerbread). The application is designed to provide the necessary interfaces/utilities which a user can use to record a walk.

The recording functionality of the application refers to the process of tracking the devices' GPS coordinates and displaying their current position on a localised Google Map (based off the GPS coordinates returned). Also in order to record a walk the application enables the ability to add a point of interest to the walk which in turn accesses the devices camera/gallery, this point of interest will be shown on the Google Map as a new Marker with custom details and icons.

The user can then upload or delete this walk depending on their needs. The walk recordings that get uploaded are viewable of the web client thanks to PHP being executed on the server side and a MySQL database.

Program structure:

Flow of control pseudo code for the Android application:

```
application is launched and start screen is displayed
click the create new walk button
the CreateWalk Activity is displayed
enter the information requested and click record walk
        while walk name && walk short description && walk long description are not empty
                while walk name && short description && long walk validate
                        progress to WalkRecording Activity
                else
                        Toast user about incorrect input and why this is
WalkRecording Activity starts
        retrieve devices GPS and pass information to GoogleMap Object
                if GPS is available (is Internet connection ready)
                        display local map
                else
                        use last know location and display map based off that
        display users location on the GoogleMap
user clicks new POI button
CreatNewPOI Activity starts for result
user enters the information requested
click add image button
        prompt user for gallery to camera selection
                if user selects camera
                        launch camera intent and create a new File and store/return captured
                        image in this file
                if user selects gallery
                        launch gallery intent and return the absolute path of the image selected
click create button
        while POI name && POI description are not empty
                while  POI name && POI description pass their validation
                        if imgae is not null
                                create a new PointOfInterest Object with the above information
                                        parce this Object using the Parcelable interface
                                                Pass this parce Object back to the WalkRecording
                                        Activity
                                                        add this new PointOfInterest to the Vector of
                                                        PointsOfInterest
                                                                Add a Marker to GoogleMap with this
                                                                information
user clicks edit walk button
add extras to this intent
        these extras will be the current walk name, walk short description and walk long description
                start EditWalkActivity for result
```

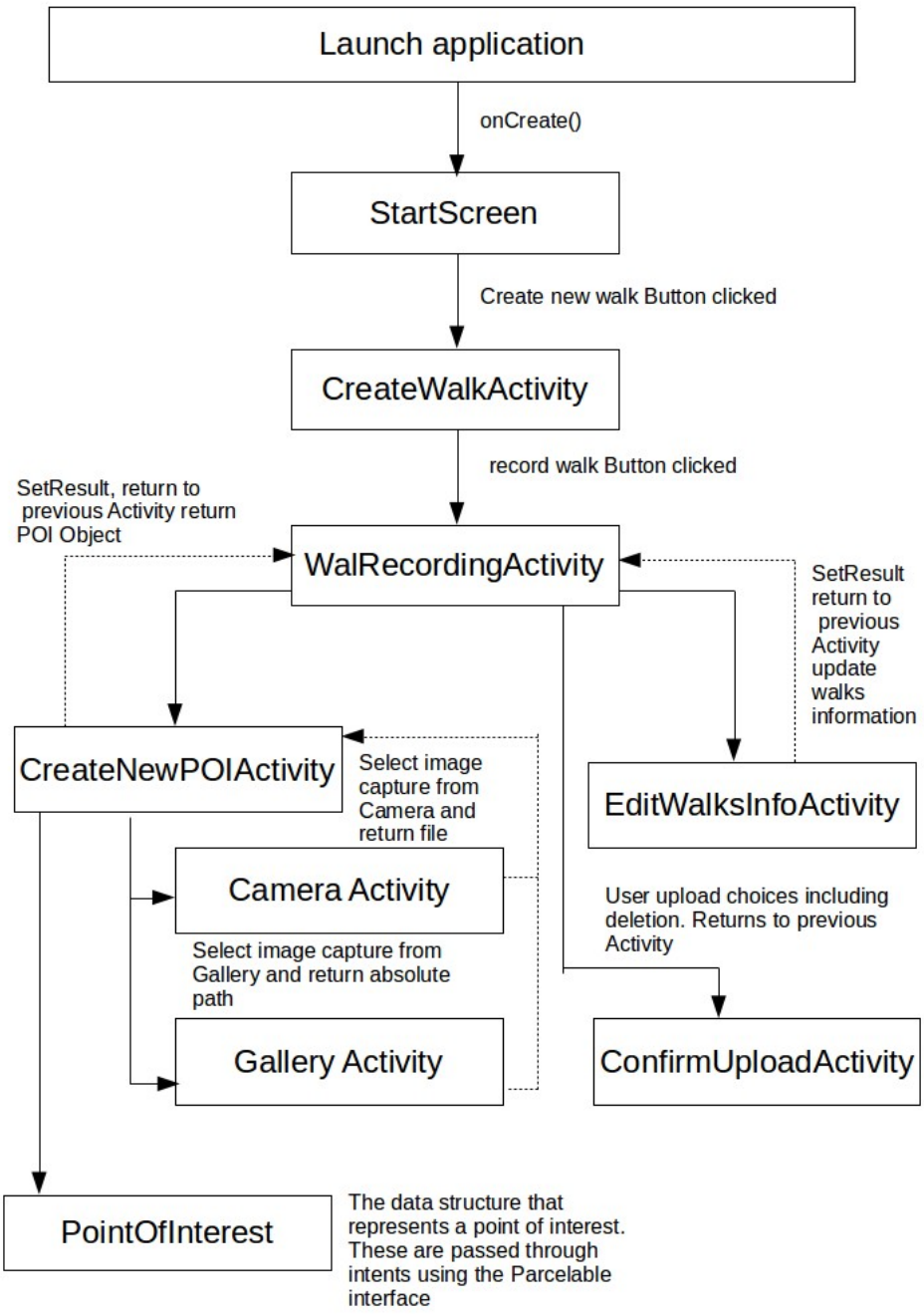
retrieve the passed in intent Extras
set the EditText's on EditWalk Screen to the corresponding passed in values
walk name → passed in walk name extra
walk short description → passed in walk short description extra
walk long description → passed in walk long description
user can now edit the walks information
click update info button when happy with edits
set the result and put the new values as extras
retrieve these extras using onActivityResult()
Update the walks information accordingly

user clicks upload Button
put the walks current information as intent extras
start ConfirmUploadActivity
retrieve the passed in walk details
display these to the user as a quick summary of the current walk

user click confirm button
prompt for upload confirmation
if user accepts upload
open URL connection to upload.php file
set up HTTP POST request
start AsyncTask and pass in POI Vector as argument
loop through the passed in Vector
Create JSON Object to replicate schema specified ind design doc section 5.4.3.4.1
populate JSON with information from Vector
POST this JSON to the sever

else
prompt the user about walk deletion
if user aspects deletion
prompt again reinforcing deletion policy
if user aspects deletion
delete walk and clear Google Map
finish Activity
else
return to ConfirmUploadActitiy
finish
continue recording walk.

Flow of Control Diagram:



Program modules:

StartScreen:

This is the basic interface for executing the initial procedures which start the application. This module does not have an awful amount of functionality it is effectively a “welcome” screen so the user knows exactly what they are viewing as well as providing the ability for the user to open the application and then close it again without having to start a recording.

StartScreen methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void addListenerOnButtons()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute a new intent and start the corresponding Activity.

CreateWalkActivity

This is the Activity which provides the user interface for setting up the walk correctly. This Activity holds the user interface elements that provide the opportunity to give the walk a name and both short and long descriptions.

CreateWalkActivity methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void setEditTexts()

This method instantiates the EditText variables which enables the contents of these EditText to be set and or retrieved. This method also applies to correct content restrictions to each of the EditText to sanitise the data entry and prevent possible complications which can arise from malformed data entry.

public void addListenerOnButtons()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute the associated instruction(s).

public String **getWalkTitleText()**

This method simply returns the value of the String value of the EditText Object which houses the walks name.

public String **getWalkShortDescText()**

This method simply returns the value of the String value of the EditText Object which houses the walks short description.

public String **getWalkLongDescText()**

This method simply returns the value of the String value of the EditText Object which houses the walks long description.

public boolean checkInputLength(String**)**

This method is used as part of the input validation for each of the EditText Objects. It returns a boolean value depending on the result of the validation check. If the passed in String has a length of greater than 0 then true is returned, else false is returned. This method prevents “null” Strings being set for the walks information.

WalkRecording:

This Activity provides the main interface and access to the main algorithms associated with the actual recording of the walk. This is where the GPS coordinates are retired as tracked, also the Google Map Object is displayed to the user at this time. From this screen you add start the upload process, edit the walks information and also add new PointsOfInterest to the current walk. This Class extends FragmentActivity so a Google Map Object can be displayed.

WalkRecording methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

Public void AddOnClickListeners()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute the associated instruction(s).

public static Vector<PointOfInterest> **getPois()**

Returns the current Vector<PointOfInterest> which is statically accessible and belongs to the WalkRecording Class. This is so a new Vector<POintOfInterest> isnt acidently created.

protected void onActivityResult(int requestCode**, int** resultCode**,** Intent data**)**

The method is provided as part of the Android language specification. It is used to handle Activities initiate using StartActivityForResult(Intent). This method handles multiple Activity results which are segregated based on requestCode and resultCode. The Intent parameter argument is used to provide the data passed in from the Activity you are expecting the result from.

public static String **getWalkName()**

Returns the current value of the String variable related the the walks name.

public static void setWalkName(String)

Set the current value of the String variable related to the walks name.

public static String **getWalkSDesc()**

Returns the value of the String variable related to the walks short description.

public static void setWalkSDesc(String**)**

Set the current value of the String variable related to the walks short description.

public String **getWalkLdesc()**

This method is used to return the value of the String variable related to the walks long description.

public void setWalkLdesc(String walkLdesc**)**

This method is used to set the value of the String variable related to the walks long description.

public void addPOIToMap(PointOfInterest**)**

This method is used to add a new PointOfInterest to the GoogleMap. The argument PointOfInterest will be used and the value returned from this Object will be represented as a Marker on the Google Map Object using the MarkerOption() parameter of the Map.addMarker method. This method also called the getResizedBitmap method in order to provide a custom image to the Marker.

public Bitmap **getResizedBitmap(**Bitmap **, int , int)**

This method is used to return a new Bitmap image that will be resized based on the value of the passed in Integer arguments. The first int argument is the new Bitmap's height and the later is related to the new Bitmap's width. The first argument is the Bitmap you wish to resize, and it uses a Matrix to resize the Bitmap image.

public void startCountingTimer()

This method is used to display a timer and the current time stamp for each PointOfInterest and the walk itself. To do this it performs a calculation based off of the system time in milliseconds and calculates the time difference between then and the current time. A conversion is then made to convert the millsecond difference between the two times into an hh:mm:ss format.

public String **getTimeStamp()**

This method returns the value of the String variable related to the time stamp.

public void stopTimer()

This method sets the while loop condition within the startCountingTimer() method to true. This stops the while loop thus stopping the timer.

public void deleteWalk()

This method is used to delete the walks information. During this deletion the following actions take place; first the GoogleMap object is cleared using Map.clear() then the Vector<PointOfInterest> is wiped using removeAllElements(). Then all the walk dependent variables are reset to default so all Strings (name and descriptions) are all set to the value “” (empty String). Finally the time stamp String is reset to 00:00:00 and it's text view is updated.

public GoogleMap **getMap()**

This method returns the current GoogleMap Object being used to display the walk to the user.

public void onDestroy()

This method is provided as part of the Android language specification. This is the final method call in the Activity life cycle and it is used to terminate and destroy the current Activity from the devices memory stack.

CreateNewPOIActivity:

This Class is designed to provide the interface required for the user to be able to create new PointOfInterest Objects. This Activity also has the option to launch the Camera or Gallery intent, this is required to progress through the creation process.

CreateNewPOIActivity methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void setEditText()

This method instantiates the EditText variables which enables the contents of these EditText to be set and or retrieved. This method also applies to correct content restrictions to each of the EditText to sanitise the data entry and prevent possible complications which can arise from malformed data entry.

public void addOnClickListeners()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute the associated instruction(s).

public String **getWPOINameText()**

This methods returns the value of the String variable related to the EditText for the PointOfInterest name data entry.

public String **getPOIDescText()**

This methods returns the value of the String variable related to the EditText for the PointOfInterest description data

entry.

public void addPointOfInterest(double , double , String)

This method I used to actually create new PointOfInterest Objects with the specified values. The first double parameter argument refers to the latitude of the PointOfInterest, the second parameter is the longitude position of the PointOfInterest and the String parameter refers to the time stamp for the PointOfInterest on the current walk. This method then sets the result using setResult so the created PointOfInterest Object can be accessed by the WalkRecording Class.

public void buildImageSelectionPrompt()

This displays a Dialog to the user with prompting them for a choice, this choice refers to the method they want to use to attach an image to a PointOfInterest, Camera or Gallery. Depending on their choice the camera intent or the gallery intent is launched and the return values from these methods are handled elsewhere. This method simply handles the users choice and not the data associated with the choice.

private void dispatchTakePictureIntent()

The purpose of this method is to enable the user to switch the devices camera application in order to capture a fresh image. First of all though, a new File is created using createImageFile() so there is a location to store the image. Once the user is happy with the photo they have taken, the newly related File's path is stored. This path is passed around the application to allow for efficient manipulation of the image file at any time.

private File **createImageFile()**

This method returns a File. This returned File is a new file and its location in memory. This file can be used to store images from the camera and maintain a location which can be accessed at any time to efficiently manipulate the image. The created file will be stored in the devices DIRECTORY_PICTURES.

public void dispatchGalleryIntent()

This method switches the users focus and launches the devices media gallery. Once this intent has been launched the user can select an image which will be attached to the PointOfInterest. The Galery intent is started using startActivityForResult.

protected void onActivityResult(int requestCode**, int** resultCode**,** Intent data**)**

The method is provided as part of the Android language specification. It is used to handle Activities initiate using StartActivityForResult(Intent). This method handles multiple Activity results which are segregated based on requestCode and resultCode. The Intent parameter argument is used to provide the data passed in from the Activity you are expecting the result from.

public String **getRealPathFromURI(**Context **,** Uri **)**

This method is required in order to retrieve the actual file path from an file (image) returned from gallery selection as the gallery will “hide” the real file path. In order to retrieve this file path a Cursor is used in conjunction with a ContentResolver query in order to parse the Uri parameter until the absolute path is located. This absolute path will then be returned as a String. The Context parameter is simply the contect in which this was called, e.g. the Activity you are currently in and the Uri parameter is the Uri of the file you wish to find the true path for.

public boolean checkInputLength(String**)**

This method is used as part of the input validation for each of the EditText Objects. It returns a boolean value depending on the result of the validation check. If the passed in String has a length of greater than 0 then true is returned, else false is returned. This method prevents “null” Strings being set for the walks information.

public void setFilters()

This method is used to apply input sanitation to the EditText's associated with setting th data required for every PointOfInterest. This input sanitation refers to character and size limit restrictions to prevent a breach of the functional requirements. Characters are limited to a-zA-Z0-9 only, with the inclusion of the
' ' (space) character for the description EditText.

ConfirmUploadActivity:

This Activity is used to HTTP POST the walks information to the server in the form of a JSON Object with the scheme a specific in section 5.4.3.4.1 of the design specification. This Activity uses a AsyncTask to perform said task because in android you cannot perform network connections on the main or UI Threads.

ConfirmUploadActivity methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void buildUserPrompt()

This method constructs a Dialog which prompts the user for input in regards to a YES/NO choice. These YES/NO values are represented as upload to server (YES) or delete walk (NO) depending on the users input a different route through the Activity is followed. If the user does in fact want to upload to the server, then the uploadToServer method is executed, else the deletion prompt is shown.

public void buildDeletionPrompt()

This method constructs a Dialog which prompts the user for input in regards to a YES/NO choice. If the user selections YES, then the walk will be deleted and you will be returned to the WalkRecording, else the prompt will be removed from view and the user will remain with this Activity in focus.

private void showUploadMessage()

This method simply used the AlertDialog builder to display a message of upload confirmation to the user.

public void locateUIElements()

This method is used to locate the user interface elements for this Activity. This method enables the ability to add functionality to the Activity such as changing the context of the TextViews.

public void addOnClickListeners()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute the associated instruction(s).

public void uploadToServer(Vector<PointOfInterest> **)**

This method handles the server upload algorithm(s). It takes th current Vector of PointOfInterest Objects (so all the PointOfInterest Objects assoicated with this walk currently) and parses them into a JSON Object with the scheme specified in 5.4.3.4.1 in the design specification. In order to parse this Vector this method makes use of the AsyncTask and it's doInBackground method to prevent performance problems.

private static String **convertInputStreamToString(**InputStream **) throws** IOException

This InputStream converter is used to debug message received from the sever. It will convert the server response (InputSream parameter) into a String which can easily be read/displayed to help with the debugging process. An IOException can be thrown by this method is there is a problem with the associated BufferedReader.

private class UploadASyncTask extends AsyncTask<JSONObject**,** Void**,** Void>

This nested private Class is used to execute the HTTP POST request for the upload of the walks data. You have to use an interface such as the AsyncTask in order to perform these tasks in Android as you cannot do so on the main Thread. This method converts the final JSON Object into a String and that is the data that is sent to the server for parsing.

protected Void **doInBackground(**JSONObject**...)**

This is the “worker” method for the sever upload algorithm. This expected in the background thus, it does not free the application when performing intensive tasks. This method is what actually send the JSON String and handled the server response.

private String **fileToBitmapAndEncode(**File**)**

This method is used to encoded any file into a based 64 String which can then be sent to server for parsing. The File parameter is the file stored in each PointOfInterest Object, this File is then resized so it is easily sent across the server reducing the strain and the base 64 String is encoded and returned.

private static String **getStringFromBitmap(**Bitmap**)**

This method is used to convert a Bitmap image into a base 64 String encoded JPEG. In order to do this it first turns the image into a byte array, which is then encoded into this returned base 64 String representation of the JPEG image. The compression quality is set to 100, which provides a suitable level of image quality.

public Bitmap **getResizedBitmap(**Bitmap **, int , int)**

This method is used to return a new Bitmap image that will be resized based on the value of the passed in Integer arguments. The first int argument is the new Bitmap's height and the later is related to the new Bitmap's width. The first argument is the Bitmap you wish to resize, and it uses a Matrix to resize the Bitmap image.

public int conertTimeStringToSeconds(String timeString**)**

The time stamp is stored as a string and in order to send the time string and store it correctly in the database (see design specification 5.4.2.1) this String must be sent as a numerical representation. To do this efficiently the String is split base on the ':' character providing 3 String arrays which represent [hh][mm][ss] these can them be individually evaluated and turned into seconds which can be added together and sent as a total number of seconds to the server, this total can easily be transformed back into hh:mm:ss.

EditWalksInfoActivity:

The Activity is used to edit the walks information during the recording process. This Class provides the necessary methods and user interface elements required in order for the user to make run time adjustments to the walks name, short description and its long description. These updates are reflected mediately.

EditWalksInfoActivity methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void locateUIElements()

This method is used to locate the user interface elements for this Activity. This method enables the ability to add functionality to the Activity such as changing the context of the TextViews. This method also applies to correct content restrictions to each of the EditText to sanitise the data entry and prevent possible complications which can arise from malformed data entry these restrictions are identical to those in place in the CreateWalkActivity.

public void addListenerOnButtons()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute a new intent and start the corresponding Activity.

public String **getWalkTitleText()**

Returns the contents of the EditText responsible for the data input for the walks title in String form.

public String **getWalkShortDescText()**

Returns the contents of the EditText responsible for the data input for the walks short description in String form.

public String **getWalkLongDescText()**

Returns the contents of the EditText responsible for the data input for the walks long description in String form.

public void setEditTextValues()

This method uses the information passed to this Activity by the WalkRecording Activity. The information passed to this Activity represents the walks current name and its short and long descriptions. If any of these passed in values are NULL then this method does nothing, if none of them are NULL then there EditText's on this screen will be updated to reflect the walks current information.

public boolean checkInputLength(String**)**

This method is used as part of the input validation for each of the EditText Objects. It returns a boolean value depending on the result of the validation check. If the passed in String has a length of greater than 0 then true is returned, else false is returned. This method prevents “null” Strings being set for the walks information.

HelpScreen:

This Activity is used to provide a new user with basic, but informative information to help them navigate and get the maximum amount of functionality from this application. It uses a very simple method of navigation to switch the help screen information, and can be accessed from every page.

HelpScreen methods:

protected void onCreate(Bundle**)**

This is part of the Android programming language/specification. It is called upon an Activities creation. This method take a Bundle as a parameter argument which can be used to “save” the state of your Activity when switching between applications and or Activities.

public void setFirstScreen()

This method located the first element in the help screen resources Vector<String> and sents the main content panel (TextView) to reflect this value.

Public void AddOnClickListeners()

This method enables the user interface to allow user interaction events. In this instance it enables the button clicks to execute the associated instruction(s).

public void switchToNextHelpText()

This is used to update the main TextView on screen to reflect a new paragraph of help text. In order to do this the current help screen information's location in the Vector<String> of resources is tracked, then this is incremented and the next index resource is selected and displayed to the user. This only happens if there is a next resource in the Vector<String> (current index +1 < Vector.size()).

public void switchToPrevHelpText()

This is used to update the main TextView on screen to reflect a new paragraph of help text. In order to do this the current help screen information's location in the Vector<String> of resources is tracked, then this is decremented and the previous index resource is selected and displayed to the user. This only happens if there is a previous resource in the Vector<String> (so if current index != 0).

private void populateHelpList()

This method is used to fill the Vector<String> with the necessary String resources which represent the help screen content for each page. Add any new help resources to this method body.

PointOfInterest:

This Class is used to represent exactly what every PointOfInterst should look like and what data is can contain. This Class also implements the Parcelable interface so these Objects can easily be passed between Activities efficiently at run time. This is one of the main data structures for this application.

PointOfInterest methods:

public PointOfInterest(String **,** String **,** **double** l**,** **double**, String)

Constructor for the PointOfInterest Objects. This creates a new PointOfInterest Object in memory with the specified values which refer to; name, description, longitude, latitude and the time stamp. This is called by the CreateNewPOIActivity.

public PointOfInterest(Parcel**)**

The Parcelable constructor. This is to decode the encoded Parcelable Objects back into their pre-encoded form. This is to again allow the application to easily manipulate these Objects between Classes/Activities.

public String **getImagePath()**

Returns the file path for the image associated with this PointOfInterest Object, this will be in String form.

public void addImage(File)

This methods provided the ability to attach a file which will point to the image you want associated with this PointOfInterest Object. The File parameter will be the the File associated with this PointOfInterest Object.

public PointOfInterest **makeAndReturnPOI (**String **,** String **,** **double** **,** **double** **,** String)

Returns a creates a new PointOfInterest Objects. This creates a new PointOfInterest Object in memory with the specified values which refer to; name, description, longitude, latitude and the time stamp. This is called by the CreateNewPOIActivity.

public void writeToParcel(Parcel **,** **int** **)**

This encoded the PointOfInterest Objects into a Parcel form which can then be decoded back into Object form using **PointOfInterest(**Parcel**)** method. This is essential for this application do not remove this method. The Parcel parameter is the Parcel destination for the encoded Object and the int parameter is used to specify any flags/options you wish to use during the encoding.

public String **getName()**

Returns the value of the variable associated with the PointOfInterests name in String form.

public String **getDescription()**

Returns the value of the variable associated with the PointOfInterests description in String form.

public double getLongitude()

Returns the PointOfnterests longitude which is represented as a double.

public double getLattitude()

Returns the PointOfnterests latitude which is represented as a double.

public File **getImage()**

Returns the File (image) associated with this PointOfInterest.

public String **getTimeStamp()**

Returns the value of the variable associated with the time stamp for this PointOfInterest in String form.

public void setLat(double)

Specify the value for this PointOfInterest latitude, this must be presented as a double.

public void setLng(double)

Specify the value for this PointOfInterest longitude, this must be presented as a double.

public void setName(String)

Specify the value for this PointOfInterest name variable, this must be represented as a String.

public void setDescription(String)

Specify the value for this PointOfInterest description variable, this must be represented as a String.

public void setTimeStamp(String)

Specify the value for this PointOfInterest time stamp variable, this must be represented as a String.

Algorithms:

Recording the walk/Adding points of interest:

1. First of all the user starts the Walking Tours application.
2. Then user will now be represented with the Start Screen where the logo is show (a globe) as well as a simple welcome message, a brief message related to the help section and a create new walk button.
3. The user now click this “create new walk” button.
4. The createWalkActivity now launches as the user is show the screen titled “Create a new Walk”
5. On this screen there is 3 EditText Widgets all labeled to describe their purpose, they represent the walks name, and both the short and long descriptions. These Edit Text's have sanitation restrictions put in place. A walk name cannot contain ' ' (the space character) nor can it exceed 15 characters in length, the short description cannot exceed 100 characters and the long description cannot exceed 1000 characters. Also, none of the edit texts will except any characters except for a-zA-Z0-9 and ' ' (excluding the name EditText).
6. To start Recording click the “Record Walk” button
    1. Once this is clicked the application will validate the data inputs to check their length (check they aren't null or length 0) as the other validation takes place as the user types. This size validation is prevent the application crashing due to null entries.
    2. If this validation passed the RecordWalk Activity will start.
7. The user will be greeted with a new screen with the walk name they entered in step 6.1 at the top of the screen above the displayed Google Map Object. When this Activity is created the devices GPS location is retrieved using NETWORK communications through the devices Internet connection (WI-fi) and the map Object's location reflects this by displaying a locaised map, with a zoom level of 14 of the local area. The users current location will be displayed on the map using the Google Map's API map.setMyLocationEnabled**(true)** the user will appear as a blue dot at their current location. This updates real time so they have a visual representation of where they currently are/heading. At this stage a static Vector<PointOfInterest> in created.
8. The user has 3 options at this screen:
    1. create a new PointOfInterest
        1. Once the “new POI” button is clicked, the devices current GPS coordinates are again retrieved using LocationManager and these coordinate values for the latitude and longitude and added to an intent as extras with suitable String key values. The time stamp is also retrieved as that stage. The time stamp is calculated using the time the application was started based on the devices current time in millisecnds, you then subtract the current time in milliseconds away from this start time and get the absolute value. You now have the run time in miliseconds, to convert this to seconds divide by 1000. from this time in seconds you can convert to hh:mm:ss using hours = seconds / 3600, mins = seconds / 60, seconds = seconds % 60.
        2. The CreateNewPOIActivity is then started using startAcitivityForResult with the intent with the added extra data used as the starting intent. In the onCreate method for CreateNewPOIActivity these passed in extras as retreived using the specifies String keys in step 8.1.1 and locally stored.

3. The user is now required to enter the information in the EditText' on screen, these relate to the PointOfInterests name and description. These EditText's also have data input sanitation where the only values allowed are a-zA-Z0-9 and ' ' (' ' is not allowed in the name EditText). In addition the name cannot exceed 15 characters and the description cannot exceed 100.
4. Once this information has been correctly entered the user must add an image, click the “add image” button.
    1. A prompt is now displayed asking the user if they want to use an existing image (from device gallery) or capture a new image (using device's camera).
        1. If the user opts to use the device gallery then the gallery intent is launch using the content filter of image/* limiting the selection to images so videos cannot break the application.
            1. One you have selected the image, you are returned to the Walking Tours application in the same position as you were before. You must now get the absolute path from the returned camera image so you can pass this through the application as a String and encode the image later on.
            2. In order to do this see the details on the getRealPathFromURI method in the previous section.
    2. If the user opts to use the devices camera, you first need to create a File in which this new image will be stored, this file will be in the external media directory on the device. For more information on this see the method detail for createImageFile() in the above section.
        1. The camera intent will now launch. The user will take a photo as normal as when they are happy with the photograph they will be returned to the same location with no information loss.
            1. This newly taken photo is now stored in the external media storage on the device you retrieve the absolute file path for this and you can now access this image at any time.
5. Once the image selection has taken place, the user clicks the “create” button.
    1. Validation I performed to check that:
        1. none of the data fields are null or of length 0 (name, descirption)
        2. check that there is a file associated with the PointOfInterest
    2. if 1.1 and 1.2 pass then a new PointOfInterest is created with the name and description specified in stage 3, the coordinate from stage 2 and the image file from stage 4. This new PointOfInterest Object is then encoded into it's parceable form and set as the result of this Activity.
6. On return to the WalkRecording Class the result set in section 5 is evaluated and checked for its origin and if the result was OK.
    1. If these tests turn out to be in fact from stage 5.2 then the PointOfInterest is then decoded back into it's Object form and added to the Vector<PointOfInterest>.
    2. Once it has been added to the Vector<PointOfInterest> it also needs to be added and displayed to the user on the GoogleMap. To do this, you must create a custom Marker using the MarkerOptions utility provided by the GoogleMaps API.
        1. To make add a custom icon to a Marker you need to convert the image to the Bitmap file format. To do this you retrieve the file path stored for the PointOfnterest you created in step 5.2. With this retrieve path you can pass it as a parameter to the BitmapFactory.decodeFile(file path) method which return a Bitmap image.
            1. This image will be very large, so the image is now resized using getResizedBitMap method which resizes the image to 100x100 pixels.
                1. To do this it divides the full size Bitmap width/height by the new width/hieght and uses these values in a Matrix Object In order to achieve a scaled version of the Bitmap image.
        2. Once you have resized the image step 2.1.1.1 you can now add the rest of the data to the Marker
            1. To do this use the attributes provides by the Marker API and simply getters/setters for the encapsulated PointOfInterest Object(s)
                1. example:
                    1. .title(PointOfInterest.getName())
                    2. .snipper(PointOfInterest.getDescription() + “ “ + PointOfInterest.getTimeStamp())
                2. steps 1.1 and 1.2 will create a marker with a title and a description which is visiable when clicked. You then want to set the icon for this Marker to the image created in step 2.1.1.1.
                3. Once you have completed steps 2.1.1 the Marker will be shown on the Google Map.
7. Repeats steps 1 through for an PointOfInterest you want to add.

2. Edit the walks information
    1. in order to edits the walks information: the walk's name, and both its short and long descriptions th user clicks the “Edit walk” button

1. Before the EditWalksInfo Activity is started the current walk information must be retrieved from the WalkRecording Activity (our current location) these variables will then be put into a new Intent using as intent data extras with corresponding sensible String keys so they can easily retrieved.
    1. Once step 1.1 has been completed the EditWalksActivity will be started using startActivityForResult with the new Intent with the added extras as the parameter.
2. Once the EditWalksActivity has started the extras put into the intent will be retrieved and the values will be displayed in the corresponding EditText locations.
3. The user can now edit these values:
    1. the edited values must meet the restrictions specified early in stage 5.
        1. if these validations pass the user can click the update info button
            1. a check is performed to check the updated values length (check it is large than 0 to prevent “”) and also check against NULL entities.
                1. If stage 1.1.1 passed all OK, then create a new Intent and put the new updated values as extras and set this Intent as the result.
    2. On return to the WalkRecording Activity check the result set in stage 1.1.1.1 and check the result == RESULT_OK
        1. if so then updates the walk's information.
    3. Repeat step 2 as many times as you need to.

3. Upload the walk to the server
    1. see Uploading to server below

Uploading data to the sever:

1. The user clicks to the “upload” button
    1. Before the ConfirmUploadActivity is started a new Intent is created. To this intent the walks current information is passed as separate intent extras with corresponding String keys, that is the walks name and both its short and long descriptions.
        1. The ConfirmUploadActivity is then started with the Intent created in stage 1.1 as the parameter.
2. Inside the onCreate method for the ConfirmUploadActivity the extras put in the Intent in stage 1.1 are retrieved and locally stored in variables. The walk name and walk short description are then displayed to the user as a brief summary of the walk.
3. Once to user is ready to upload, click the “confirm” button
    1. A prompt will be displayed asking the user if they want to upload the walk or delete the walk
        1. if at stage 3.1 the user selects the want to upload the walk:
            1. the static Vector<PointOfInterest> is retrieved using the getter provided In the WalkRecording Class, this is then passed to the uploadToServer method.
            2. The upload to server method will generate the security hash and salt to pass server authentication the user does not know about these, nor do they matter to the user.
            3. A JSON Object is created using Several nested JSON Objects/Arrays matching the schema in section 5.3.4.3.1. This JSON Object is then populated with the correct values by looping through the retrieve Vector<PointOfInterest> .
                1. for every element in Vector<PointOfInterest>
                    1. Convert the current PointOfInterest's time stamp to seconds using convertTimeStringToSeconds
                    2. encode the current PointOfInterest image into base 64 String using fileToBitmapAndEncode
                    3. create a new JSON Array for the images.
                    4. create a new JSON Object
                        1. put the current PointOfInterest's name into the stage 3.1.4 JSON Object with the key “name”
                        2. put the current PointOfInterest's latitude into the stage 3.1.4 JSON Object with the key “latitude”
                        3. put the current PointOfInterest's longitude into the stage 3.1.4 JSON Object with the key “longitude”
                        4. put the current PointOfInterest's time stamp (result of stage 1.1) into the stage 3.1.4 JSON Object with the key “timestamp”
                    5. create a new JSON Array
                        1. add the current PointOfInterest's name into the JSON Array created in stage 3.1.5
                        2. add the current PointOfInterest's description into the JSON Array created in stage 3.1.5
                        3. add the JSON Array created in stage 3.1.5 to the JSON Object created in stage 3

6. add the images to the JSON Array created in stage 3.1.3
    1. add the file name for the current PointOfInterest'simage with the .”jpeg” extension to the JSON Array referenced in stage 6
    2. add the 64 base encoded image String to the JSON Array referenced in stage 6
7. add the JSON Array referenced in stage 6 to the JSON Object created in stage 3.
8. add the remaining information to th JSON Object created in stage 3:
    1. add the walk name with the key “title”
    2. add the walks short description with the key “shortDesc”
    3. add the walks long description with the key “longDesc”
    4. add the sever authorisation SHA1 from stage 1.1.2 with the key “auth”
9. Execute the AsyncTask

2. Pass the finalised JSON Object created after stage 1 is complete to the AsyncTask which is a private nested Class.
    1. Execute the doInBackground method using the passed in JSON Object from stage 2 as the parameter
        1. Create a new HttpParams Object
        2. Create a new HttpClient Obectwith the stage 1 HttpParams as the parameter
        3. Create a new HttpPost Object with the URL String for the upload.php code as the parameter
        4. Create a new List<NameValuePair>
            1. add the passed in JSON Object's (from stage 2.1) toString to the List<NameValuePair> from stage 2.4 with “data” as the String key.
        5. Create a new UrlEncodedFormEntity with the List<NameValuePair> from stage 4.1 as the parameter
        6. set the UrlEncodedFormEntity (from stage 2.1.5) content endoing to HTTP.UTF_8 using setContentEncoding(HTTP.TUF_8)
        7. set the HttpPost's (from stage 2.1.3) entity as the entity created in stage 2.1.6
        8. set the HttpPost's (from stage 2.1.3) header to be something meaningful for the server log.
        9. Execute the HttpPost request:
            1. create a HttpResponse Object with the HttpPost object referenced in stage 2.1.9 as the parameter
4. Server upload is now complete.

The main data areas:

There are two main data areas which are used and manipulated throughout the Walking Tours application. These two main data areas refer to the design a structure of a point of interest on the walk this is represented as the PointOfInterest Class. The second data structure is contained in WalkRecoring Class. The data structure referenced here is a Vector which contains all the current points of interest on the current walk.

PointOfInterest Obejct(s):

A PointOfInteret Object is used to model and represent a location that user wants to add to the current walk. These Objects holds encapsulated data related to a PointOfInterest's name, description its GPS coordinated represented as separate longitude and latitude double variables as well as the File associated with the image attached to this PointOfInterest.

The way in which these Objects are created is using the interface provided in the CreateNewPOIActivity. These Objects implement the Parcelable interface so these Objects can be encoded into Parcel form, which can then easily be passed around Activities in order to maintain references to each PointOfInterest in memory thus allowing them to be added to the Vector<PointOfInterest>. Once these Objects have been decoded from their Parcel form back into their Object form, and added to the Vector<PointOfInterest> the encapsulated data can easily be retrieved by accessing x index in the Vector and using the getter/setters provided as part of the PointOfInterest Class interface.

Vector<PointOfInterest> in the WalkRecording Class:

The WalkRecording Class houses a statically created Vector<PointOfInterest> which contains every “location” that has been added to the current walk. These locations are represented using the PointOfInterest Objects as discussed above.

The reason a Vector is used use as the data structure is because Vectors are very effectively Thread safe ArrayLists. The provides us with the ability to specify that a List is required but as opposed to an Array implementation the size does not have to specified. This allows the user to not be limited by a size restriction to the PointOfInterst List.

This Vector is declares as static because it belongs to the Class itself. Also the fact that it is static allows it to be

manipulated and accessed by other Activities without having to create a new Vector in each Class that requires access, or indeed needing to pass the whole contents of the Vector through Intent Extras.

The elements in this Vector as accessed by the ConfirmUploadActivity and the uploadToServer method in order to populate the JSON Object specified earlier. Every PointOfIntest element in this Vector is parsed at this stage, and it's encapsulated data is retrieved. This is required in order to complete the upload process and pass server validation.

Files:

.java Files:

Any and all Android source code must be contained in files suffixed with the .java notation and should be complied to produce .class files. These .java files should be placed in the relevant package location relevant to the applications structure. These .class files should be placed in the bin folder, all though this will normally be handled by the Java complier either through an IDE or using the javac command line tool.

.xml files:

XML files in Android are used to specify the system resources required. These resources range from the layout of each screen (Activity) to the permissions/hardware the application with need to interact with on the device as well as specifying all the Sting resources you will need to display in your TextView's, Buttons and EditText's.

These files should be place in the res/layout directory of the project (excluding manifest.xml and Strings.xml) and should be suffixed with the .xml extension.

Android manifest XML file:

This xml file is extremely important. This manifest file contains all the information required to execute your application as intended. This XML file contains all the permissions your application needs to run as well as the hardware and or the Android Operating System interfaces it will require access to in order to complete it's objectives.

Also in this file your activities are specified and declared to be Activities. This tells the Android Operating System to treat them differently to standard Classes allowing them to be brought into focus. If you create an new Activity or to need access device hardware you must specify the permissions in this file or you cannot run/access what you require.

Strings XML file:

This file is created by Android for you. This being said you need to define any colours or static String resources in this file so you can access them at or before run time. It is bad practice and not advised in Android to manually define Strings for you Widgets you should always define them in this Strings.xml file.

This file is located in res/values do not move this file location or delete.

image file(s):

Every PointOfInterest Object has an File associated with it. This File represents the image attached to the PointOfnterest. These files will be stored in the Android device's external storage, this will be handled by the Android Operating system. It must be said though, the file created when taking a new image using the devices camera are encoded as a JPEG. Take this into account if you make any modifications to the application.

Interfaces:

The application interacts with many externally provided interfaces. These interfaces enable the apllication to perform several tasks directly related to the functional requirements as specified in the requirements documentation. These taks include accessing the devices media storage e.g. camera/gallery intents, accessing the devices network state to enable GPS retrieval and accessing the Google API's to display a Google Map Object to the user.

camera/gallery

The Android application specifies in it's manifest that it will requires access to the camera via android.hardware.camera inside a users-feature tag. This is a requirement and without a camera this application cannot be started. Also the manifest specifies that this application will attempt to write to the devices external storage. This is required when you try to create a new image file just before the camera intent in started. Without this permission you have no way of

saving the newly captured image, so you could no longer access it again once you have finished with the camera application.

accessing devices network connection:

The manifest specifies that the application will try to access several sections of the Android language specification. These sections refer to the ability to first see if the device is currently connected to Internet and secondly retrieve the GPS coordinates based on the devices current location, the later is done using the LocationManager Objects and interface provided as part of the Android libraries.

The permissions specified in the manifest are; ACCESS_NETWORK_STATE, ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, ACCESS_MOCK_LOCATION and ACCESSS_WIFI_STATE. These all relate and directly effect the ability to access/evaluate the devices' current network connection state. These permissions are required and should not be removed or changed.

using Google Maps for Android:

In order to display a Google Map Object to the user there is several steps you need to follow:

in the manifest you need to specify the following permissions packageName.permission.MAPS_RECIEVE and com.google.android.providers.gsf.permission.READ_GSERVICES. These allow the application to access the Google Maps for android v2 API.

On top of the above, also in the manifest you need to make sure the device that want to run this application has open version 2 (minimum) this is a requirement and without this you cannot run the application, because if you did the Google Map Object cannot be displayed with this version.

The Google API Console is also used in order to register the application for the Google Maps for Android v2 API access. This key is based on your SHA1 key provided upon your Android install. If you do ot register this SHA1 key and your package name in the Google API Console under the Google Maps for Android v2 section, you will receive and authentication error upon attempting to render the Google Map Object.

Suggestions For improvement:

Walk will be deleted when the back button is pressed:

The Android Operating system provides a back button on every screen that is displayed to the user. This button I located near another two buttons which are home and a button which displays all running applications. If the user is on the ConfirmUploadActivity screen and they press this back button either purposely or accidentally then the user will be returned the WalkRecording Activity but the currents walks information will be deleted.

This is obviously not an intended feature, and it happens because the Google Map is created in the onCreate method in the WalkRecording Activity. In order to prevent this the Google Map Object should only be created once and a check should be performed to see if the map has any Markers or if any PointsOfInterest are currently being stored. If either of these turns out to be true, then do no delete the walks information.

some minor display changes, some elements are incorrectly positioned:

This is not related to the functionality of the application but it is related to the aesthetics. On the StartScreen there is a help text Text Label which tells the user if they click the blue help icon at any time they will be taken to the help screen(s). This Text View should be centered when this applications design is revised as currently it looks out of place as it is shifted to the far left hand side of the screen.

This is very trivial is all honesty, it ranks lowest in order of importance in these discussed maintenance tasks.

Add the ability to add multiple photos to a PointOfInterest:

It maybe the case that user wants to add multiple photos to a particular PointOfInterest Object. The sever is capable of handling this currently but there is no way to add multiple images to a PointOfInterest through the applications current interface. If the user has already attached an image, and they go through the image adding process again then the most recent image will be associated with the PointOfIterest Object and the “old” image will be removed.

This will provide greater functionality to the application as it would be useful to both the application user a the web site

user to view multiple images.

This should be fairly simple to implement and will involve a new data structure in the PointOfInterest model, this data structure should hold a list of all Files associated with this PointOfInterest.

prompt the user if and why the upload has failed:

Currently the user is prompted with the same upload message. This message tells the user the that the upload has been completed. Sometimes the upload may fail, an the user will have no knowledge of this so when they upload and receive the upload complete message and go to view their walk on the via the web application they will be confused as to why it is not displaying.

The Android application already has measures in place to retrieve and parse the server's response but currently it does not an awful lot with the response. The server will return a keyword of true if the upload is success, and false if failed. You can then parse this response String and look for the keywords true or false and display an appropriate message to the user.

This should be fairly simple to implement, and improves the users performance significantly.

re factor the code:

Code refactoring is one of the most important maintenance tasks. It is important to re factor the code in order to make other maintenance tasks easier. Also with “clean” re factored code, de bugging/making the existing code more efficiently become much, much easier as you can see what is happening without having “clouded” vision.

Potentially you could refactor this application's source code and extract the Google Map Object into it's own Class and set of data structures as well as potentially handling the GPS tracking in a separate Class as well. There are several other ways you can refactor, these are just two, potential the more useful examples.

update the minimum API level at more versions are released:

Currently the minimum SDK/API level targeted in Android 2.3 also know as Gingerbread. This is because there are many devices running version 2.3 still so these users cannot be avoided. This being said the general application targets is 4.4 (KitKat) this is the current latest release for the Android Operating System.

In the future it maybe the case that 2.3 becomes obsolete so there is no point targeting this version anymore. The code should be updated to reflect the new “minimum” being widely used, and the old methods should be updated to reflect new principle in this newer release.

Randomly general the authentication salt:

The sever expects an encrypted String to be used as a method of authentication to check the validation of the HttpPost source entity. Currently this value is hard coded so it is a security flaw as anyone with access to this salt String can easily replicate the authentication.

In order to improve the overall security this String should be randomly generated. This can either be a random String made up on “nonsense” characters etc jlnsabpnakhk or it could be a word retrieved from a stored dictionary, all through the salt wont be random itself the word selected will be chosen at random making it very difficult to replicate for every upload.

Things to watch when making changes:

The static Vector<PointOfInterest> can be accessed from any Class:

The fact that the Vector<PointOfInterest> is static and can get retrieved as any point means that if you are not careful you may accidentally overwrite the current Vector<PointOfInterest>. You must take care when adding to the Vector, you should minimalise the location in which you access the Vector<PointOfInterest> to avoid potential complications that may arise if you miss treat this static property.

The current minimum targeted API:

It must be noted when making changes to the application that the minimum target API (2.3 in this current version) may not have or support the features you want to implement. If this is the case, then don't simply change the target minimum

version you have to either postpone the update or find a work around that will allow you to achieve the same results.

Adding unsupported features will render the application unusable for users with the minimum API level specified installed on their device this is obviously not want you want. Before adding functionality or updating th existing code take the time to read up about this current minimum API and see if what to want to implement is supported.

Permissions must be specified in the Android manifest XML file:

If you want to need to add access to a new external interface/API to the application or even a new Activity then you must specify these in the manifest XML file. If you do not specify these new additions your application will simply not function as you expected, thus rendering the whole process a bit redundant until you add the information to the manifest.

It is a good idea before you start writing code for these new addition to add them to the manifest straight away to avoid the possible complication discussed in the above paragraph.

Any changes to the PointOfInterest Objects must be reflected in the JSON Object:

Since the JSON Object houses the information that is sent to the server via a HTTP POST request, any updates to the PointOfInterest data structure must also be added to this JSON Object, and the server team should also be informed. For example if you want to add a new field to a PointOfInterest e.g. Date, then the JSON Object's scheme should have the ability to represent this new information otherwise it will not accurately represent the full extent of the walks information.

Physical limitations of the program:

Screen resolution:

Because the layouts for every screen /every form widget (buttons, TextView's etc.) sizes are specified in terms of display pixels if a screen has an incredibly small or large screen resolution th application will look very different. If the device has a very small dpi (display pixels per pinch) then every form widget will appear very large and some elements may get hidden by others, effectively rendering the application unusable. Opposed to this is if a device has a very large dpi value then each form element will appear very small and maybe very difficult to interactive with, causing the same problem of the application being near to unsealable.

Processing power:

All though this application isn't very computationally intensive it is always a requirement for any system that it has a suitable amount of processing power. There isn't really a specified minimum for this application but the more power the better, As long as the device can run smooth when operating with th Android Operating system installed it should be able to run this Walking Tours application without any problems.

No camera on the device:

If the device running the Walking Tours application do not have a built in or externally attached camera then this application will not run on this device. This is because in the android manifest the camera hardware is specified as a requirement, so this restriction is unavoidable.

Lack of network connection ability:

If the device running the walking tour application do not have the ability or loses Internet connectivity then the application will not function correctly. IF the device has no Internet connection then a localised Google Map will not be displayed instead the default location will be displayed.

If the device looses connection at run time, then the GPS coordinates will be retrieved using the devices late know location. This will obviously cause problems due to inaccuracies, how inaccurate these coordinates are will depends on how long the device loosed connectivity for.

Rebuilding and Testing:

Rebuilding:

In order to rebuild the application you will the the files contained in the src folder and the res folder, you will also need the manifest XML file. These directories (src and res) are located in the Walking Tours root directory, so is the manifest.

First create a new Android project in your IDE (e.g. Eclipse) and replace the created src/res directories with the corresponding directories specified above, do the same for the manifest file. Alternatively if you are using Eclpise IDE you can simply import the project saving both time and hassle.

Regardless of which method you used to import your application you need to import the google play services library into you workspace otherwise you will not be able to access the API's. To do this it will depend on your IDE, but using Eclpise go to file → import → existing android code into workspace. From this menu you will need to locate your google place services library, this is found in your Android SDK install directory → extras → google → google_play_services and click import. You now need to "point" your project to this library by right clicking on the project and selecting add " external library", when prompted select the google play services library you added to you workspace. The two projects should now be linked.

Also on every new SDK install you perform the above steps with you will need to add the add the SHA1 key for the install to the Google API Console: Google Maps for Android v2 list of accepted keys, if you do not do this you cannot access Google Maps correctly.

Testing:

There are several Test Classes already set up inside the tests package. In order to run these test simply right click on any test Class or the test package and click run as → Android JUnit Test.

These tests should now run, and upon result your IDE or terminal window will now display the results of these tests. If these tests pass, then your IDE will display a green symbol next to the test(s) that pass and a grey symbol if there is a failure. A stack trace is available based on the message you provided to the test to help see you see the reasons of this failure.

Adding new tests:

If you want to add new tests to existing Classes simply import the necessary libraries and write your JUnit tests as normal, abiding by JUnit version 3 standards. You can no re run your tests and see if they pass or fail using the steps specified above.

Make sure if you edit the one of the provided test Classes then make you you're test conforms with the current test content, make sure you are testing the write Class and make sure you fully document what the test does and why you are performing the test(s).

adding a new test Class:

If you want to add a new Test Class you need to create a new Class as normal in the test package, and give it a meaningful name. Before you starting writing tests you need to make this Class inherit from the correct super Class, either ActivityUnitTestCase<ActivityName> or ActivityInstrumentationTestCase2<ActivityName> depending on your test intentions.

You also need to set up a Constructor for this test Class. This Constructor is very situation and its contents can vary depending on what you are testing. Refer to the existing test Classes and their constructor to check the syntax and what you expected to include inside this method, This is very important and cannot be missed.

Finally when writing a test you can set up a "setUp" method. This method I executed before the tests so in this method you declare and instantiate any variables or instances of Objects/Activities that you need during your tests. It is important to note that when you are writing tests you need to add the correct annotation and use the correct naming convention or the tests will not run. The annotations you will need are "@SmallTest" or "@BigTest" depending on your test case's method body. The naming convention is all you test case method name have to begin with the word test (JUnit version 3 requirement).