# Practical 10.1 & 10.2 - Using Location

In this practical you will build a "teleporter" app. The interface contains a single button. When the user presses the button, they are "teleported" to a **random** latitude and longitude, and shown the name of the closest city and a picture (fetched dynamically off the web) associated with that location. You will then create a **second** version of the app that provides the same functionality, but uses **the phone's GPS location**, rather than a randomly generated one.

This is a complicated app and requires you to use several Android classes and multiple web services. You will not be given detailed instructions about all of those services -- you will need to explore them on your own. You will be returning complex data from your web service calls, so you will need an appropriate class architecture. There is a lot of functionality required, so you will need to be very careful about code modularity. Plan your time carefully to make sure you can get everything done by the due date.



**Please perform this development in the following stages**. Commit each stage. This will ensure that you get each computational component working correctly. It will also help you see how well your architecture responds to changing requirements.

1. Implement an interface and the logic needed to generate and display random (but legal) latitude and longitude values when a button is clicked. Latitude ranges from -90 to +90; longitude ranges from -180 to +180. Display your generated values with three significant digits, as shown above. Ignore the "closest city" functionality at this stage.
2. Use HttpUrlConnection and AsyncTask to call a web service that accepts the latitude and longitude values and returns the nearest city. Display the city name in an appropriate TextView. The site geoplugin.net provides a simple web service for this. An example query is:

http://www.geoplugin.net/extras/location.gp?lat=-45.8787605&long=170.5027976&format=json

Which returns:

{"geoplugin\_place":"Dunedin","geoplugin\_countryCode":"NZ","geoplugin\_region":"Otago", "geoplugin\_regionAbbreviated":"F7","geoplugin\_latitude":"-45.8741600", "geoplugin\_longitude":"170.5036100","geoplugin\_distanceMiles":0.32, "geoplugin\_distanceKilometers":0.52}

If the service finds no city within range, it returns [[]]. Make sure your code deals with this gracefully. See www.geoplugin.com/webservices for full API documentation.

1. Modify your application so that it ***always*** teleports the user to a latitude and longitude where ***there is*** a closest city (i.e. geoplugin returns a non-empty value). You will need to change your logic so that the app repeatedly generates a random latitude and longitude and makes a fetch from geoplugin. It continues to do this until it gets a meaningful result. Do not make this change by copy-pasting your existing AsyncTask from stage 1 and trying to modify it in situ. This will lead to many bugs and take a lot of time (it requires more changes than it would at first appear). Instead, **plan a new AsyncTask designed correctly for this new functionality.** If you haven't already done so, you should also consider making a nice data class to send everything you need into PostExecute as a tidy class instance.
2. In theory, the logic you added in step 3 could produce an infinite loop. In practice, it is extremely unlikely to do so. However, it can take several seconds to find a good location, and during that time, your user may think the app has frozen. To prevent this, provide appropriate feedback via the Android ProgressDialog class. Look at the documentation to see how to use ProgressDialog, and where the code should go. (Hint: Remember all those methods in AsyncTask we didn't use...?)
3. Extend the functionality of your app so that, in addition to the closest city, your app displays an image related to that city. Fetch this image dynamically from the web. To do this, you will use Flickr -- an image service that supports search by tag. You are responsible for learning how to use Flickr's image search API. If you do not wish to register for your own API key, you can use the IN721 key: eda41a123d459be0f85276d37290651e. Please note that this key is limited to 100 searches per day. So please develop carefully and test parsimoniously. Some things to note about Flickr search:
   1. Not all cities returned by geoplugin will have associated images in the Flickr database. Make sure you can detect this outcome and respond to it gracefully, giving the user appropriate feedback. Do not keep teleporting around until you find a place with an image -- just work out how to cope sensibly with both cases.
   2. The Flickr API is quite standard, but its JSON feed is not. Specifically, it does not return an image URL. It returns the pieces that you use to ***construct*** an image URL. Read the docs carefully.
   3. The default string returned by Flickr is also not a syntactically correct JSON object. It has some extra characters at the front and end. **Do not** try to fix this by chopping characters manually off the string. There is a query parameter that will sort the problem -- use it.
4. Create a **second version** of your app that provides the same "image fetch" functionality, but uses the phone's GPS values to determine its location. The "closest city" information and image should automatically update as the user moves in the world (i.e. there should be no button). Use a GPS spoofer (when running on a phone) or the Emulator controls (when running in an AVD) to test. Pay close attention to your permission management.

The architecture for this version is substantially different from the architecture for the version you implemented in Task 5 (although you will be able to reuse much of the logic), so it will be quicker to start a new Android Studio project with the appropriate classes and methods, rather than to try to modify your existing one in situ.