Warm-Up Project: Location-Aware Reminders
Due: Monday, January 18th, before class

Project : Android location reminder application

Part 1: Reminder-entry Activity

Part 2: Create storage for the Reminders

Part 3: Location monitoring Service

Part 4: Reminder-display Activity (use of "extras")

Turn in: There is no turn-in to me. You will demo your system working in class - I'll supply the projector, you will need to find a laptop to run your system on.

Prior knowledge: some point early on, you should look at http://developer.android.com/guide/topics/fundamentals.html. I find myself going back to it often as I try to figure out how Android decided to implement their basic framework. You may not understand it all at once, but as you get deeper into the project, come back to it.

Overview:

The reminder-entry Activity will present a user interface (UI) and accept/store four data values (three numeric, one string). The Service will monitor location and trigger (launch) the reminder-display Activity when the phone is near a user specified location. A message will be displayed on the phone by the reminder-display Activity.

Part 1: Reminder-entry activity

The first Activity launched in the application displays the UI. This UI is a form that accepts the four components of the reminders: latitude, longitude, text, and distance.

- Latitude and longitude specify the location where the user would like to be reminded.
- Text contains the message the user would like to receive as a reminder
- Distance is the radius of a circle around the specified location in which to trigger.

When the Add Reminder button is pushed, two things should happen:

1. You should build an instance of a class ReminderEntry that captures the values of all four fields. In part 2, you will store this instance in a Reminders "database".

2. You should print the value of the ReminderEntry instance out to LogCat (see figure 3). To do this, you should have a line of code as follows, where `reminder_entry` is the result of step 1:

```
Log.d( "addButton", reminder_entry.toString() );
```

Of course, you will have to write the toString method to get the print out you see in figure 3.

Reminder: Open the LogCat display through the menu bar (Window > Show view > LogCat). You may have to choose 'Others' from Show view to see the complete list of available views.

Figure 1: Date Entry UI

Figure 2: Completed Data Entry

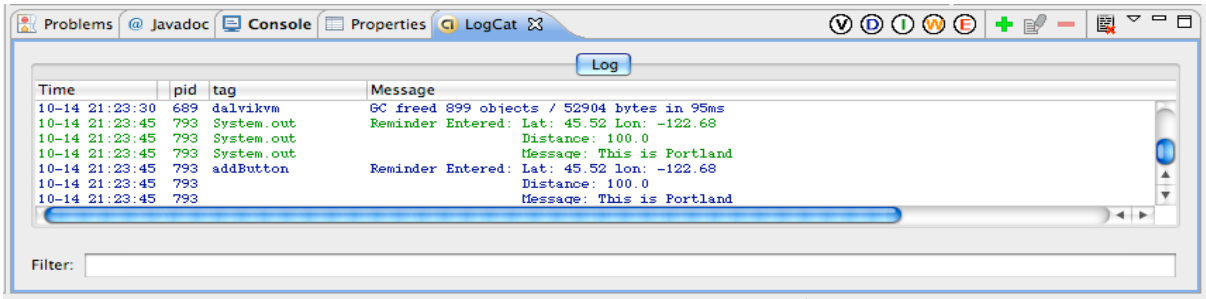

Figure 3: Output in the LogCat message log from both System.out and Log.d

Debugging note: one way to see debugging information on your screen (as opposed to the Eclipse window) is to use the Toast widget. So in addition to placing this debugging line in your code

```
Log.d( "addButton", reminder_entry.toString() );
```

you can also add this

```
Toast.makeText(getBaseContext(),
     "addButton: " + reminder_entry.toString(),
     Toast.LENGTH_LONG
).show();
```

Try it and see if you like it. If it does not stay up long enough, play around with the 3rd arg.

Part 2: Create storage for the Reminders

We will not define a service yet. Instead, we will lay the groundwork by defining a new class that will hold reminders. Your service will need to use this class to iterate through the reminders that have been defined, and check them against the phone's current location.

```
class Reminders {

       ...    //set up instance vars, class vars, constructor

       public static Reminders getInstance(){...}     //supports Singleton

       public void addReminder( ReminderEntry re ){
              if( ... )
                     ...
              else
                     Log.d("addReminder", "attempt to add duplicate");
       }

       public void removeReminder( ReminderEntry re ){
              if( ... )
                     ...
              else
                     Log.d("removeReminder", "attempt to remove non-existent");
       }

       public void clearReminders(){
              ...
       }

       public Iterator<ReminderEntry> getIterator(){
              return ...
       }
}
```

Figure 4: Reminders.java

I am going to give you some specific constraints on your new reminder-storage class (called `Reminders.java`). First, I would like `Reminders` to be a singleton. Here is a good review article on this pattern: http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-designpatterns.html?page=1. I have started you on your way in figure 1 by defining the static method `getInstance` that will return the (single) instance of the `Reminders` class. You need to do the rest of the set-up to complete the Singleton pattern.

Second, I would like the `Reminders` class to return an `Iterator` so that other classes can iterate through the list of reminders. You can choose whatever data structure you wish as the private data that stores all the reminders. You won't let outsiders see this data structure. Instead, you will return an instance of `Iterator<ReminderEntry>` as seen in the `getIterator` method in figure 4. Outsiders can use the methods on the `Iterator` interface to iterate through the reminders. How do you produce an `Iterator` from your private data structure? Look at the methods on your data structure. It likely has a method just for returning in `Iterator`! You might find the example on this page useful: http://java.sun.com/developer/technicalArticles/J2SE/generics/. It does show how to define a data structure (in this case, an `ArrayList`) using generics. And then how to get the iterator for the list and use it. The Ex1 and Ex2 examples are all you should need.

The other change we need to make is to `ReminderEntry`. I will give you my version of the class in figure 5.

```java
public class ReminderEntry {

	private final int radius;
	private final double latitude;
	private final double longitude;
	private final String text;

	public ReminderEntry(double lat, double lon, String text, int k) {
		this.latitude = lat;
		this.longitude = lon;
		this.radius = k;
		this.text = text;
	}

	//getters go here for radius, lat, lon and text

	@Override
	public boolean equals( Object re2 ){
		...
	}

	@Override
	public String toString(){
		return "GeoAlarm:\n" +
		"\t" + "lat=" + latitude + "\n" +
		"\t" + "lon=" + longitude + "\n" +
		"\t" + "rad=" + radius + "\n" +
		"\t" + "txt=" + text;
	}
}
```

Figure 5: ReminderEntry.java

As you can see, there is a new method that we will need, `equals`. The equals method is defined in the Object class. Its behavior is to do an == on the two objects being compared. As you know, this will only be true if both objects are the same instance. That is not what we want. We want two reminders to be equal if all of their numeric fields are == and the string fields are .equals. In other words, I would like the following code to print "true":

```java
ReminderEntry re1 = new ReminderEntry(45, 45, "test", 45);
ReminderEntry re2 = new ReminderEntry(45, 45, "test", 45);
System.out.println( re1.equals( re2 ) );
```

Unfortunately, if we use the default `equals`, we will have false printed: even though re1 and re2 have the same internal values, they are not the same instance, hence false. I would like you to add a method `equals` as I have started for you. Make sure your new `equals` will print true in the above case. Also note you will have to cast from the `Object` to a `ReminderEntry` in the body of your method.

```
...
addButton.setOnClickListener(
       new OnClickListener() {
              public void onClick(View v) {
                     double lat = ...
                     double lon = ...
                     String displayText = ...
                     int distance = ...

                     Reminders rems = Reminders.getInstance();
                     rems.addReminder(
                            new ReminderEntry(lat, lon, displayText, distance));

                     //write out reminders to log to make sure
                     Iterator<ReminderEntry> iter = null;
                     iter = rems.getIterator();
                     ReminderEntry temp = null;
                     while (iter.hasNext()) {
                            temp = iter.next();
                            Log.d("onClick", temp.toString());
                     }
              }
        });
```

Figure 6: Use With Add Button

Finally, make the change to your data entry activity so that when someone presses the Add button, you add the reminder to `Reminders` (as I have in figure 6). I would also like you to include the debugging loop I have in figure 6 so that the grader (and you) can assure that your reminder was stored correctly. Once you have `Reminders` and `ReminderEntry` set up correctly, you should be able to copy and paste the code from figure 6 into you Add-button handler as shown.

Strategy for attacking this part

1. Get the `equals` method working in `ReminderEntry`. You will need this for the `Reminders` class.
2. Set up the Singleton portion of `Reminders`. Test it by adding this code to `onClick`:

   ```
   Reminders rems = Reminders.getInstance();
   ```

   You can put a breakpoint in `getInstance` to make sure it is working.
3. Set up `addReminder`. Now test it in `onClick` by adding the code:

   ```
   rems.addReminder(
          new ReminderEntry(lat, lon, displayText, distance));
   ```

   You can put a breakpoint in `addReminder` to make sure it is working.

4. Set up `getIterator` in `Reminders`. Add following code to test it in `onClick`:

```
//write out reminders to log to make sure
Iterator<ReminderEntry> iter = null;
iter = rems.getIterator();
ReminderEntry temp = null;
while (iter.hasNext()) {
        temp = iter.next();
        Log.d("onClick", temp.toString());
}
```

5. Define `removeReminders` and `clearReminders`. There is no code for testing these two methods this week. Both will be used in following weeks.

Part 3: Define the Service

We now have the ability to add reminders and store them in a data structure. The next task is to give our application some functionality. We will do that by defining a service that will watch where the phone moves and check to see if a reminder triggers.

On a real phone, there is a GPS chip (e.g., http://en.wikipedia.org/wiki/SiRFstar_III) that supplies location information to the android operating system. Our java code, in turn, can ask the operating system to tell us when the phone moves. That is what we will do this week: we will define the code, in our service, that gets movement information from the operating system. But since we don't have a real phone nor real GPS chip, nothing will move, right? The cool thing is that we can simulate movement using the DDMS package of Eclipse. It will allow us to type in lat and lon values to pretend we have moved to those coordinates. The android operating system will believe we have actually traveled to that spot!

Defining the service class in the source path

Right-click on the enclosing folder of your existing files, and choose new Java class. Fill in the name `ReminderService.java`. Copy the code from figure 1 into the new file.

What does the service class look like

Study the code in figure 7. There is only one place you need to worry about, the handler code for `onLocationChanged`. You will be passed a `Location` object that has the lat-lon pair for where the phone has moved to. You need to figure out if that movement should trigger a reminder. There are two ways of doing this. One, you can google on algorithms for computing the distance between two lat-lon pairs, and then translate the algorithm into java. Or two, you can look for Java library methods that already do this for you. I was going to be mean and let you find these methods yourself, but I relented; look here for just what you want:
http://developer.android.com/reference/android/location/Location.html#distanceBetween%28double,%20double,%20double,%20double,%20float[]%29. This method has the added bonus of telling you the bearing of the user.

What about the activity that will print the reminder

What if you discover, using the `distanceBetween` method, that you should trigger a reminder? That is what part 4 focuses on. For now, you can just use Log.d to show a triggered reminder. Or a Toast object.

It's running but how do I move the phone

See figure 8. This is a DDMS view. You will see the Emulator Control on the left. Scroll down and you will see the ability to send a lat-lon pair. This will fool the OS into thinking the phone moved to the spot you choose. If you move within range of a reminder, you should see your Log.d method print a message (if you have coded it right).

```java
import java.util.Iterator;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;

public class ReminderService extends Service implements LocationListener {

        private LocationManager lm;

        @Override
        public void onCreate() {
                lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        }

        // called when the Service is started
        @Override
        public void onStart(Intent i, int id) {
                lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5, 5, this);
        }

        // called when the Service is destroyed
        @Override
        public void onDestroy() {
                lm.removeUpdates(this);
        }

        // Methods to implement the LocationListener interface

        // called when OS gets new location info
        @Override
        public void onLocationChanged(Location location) {
                // code goes here for what to do when the location updates
        }

        // Don't worry about the rest

        // Ignore this, we aren't using this functionality but need to override it.
        @Override
        public IBinder onBind(Intent intent) { return null; }

        // need these to implement the rest of the LocationListener interface
        @Override
        public void onProviderDisabled(String provider) { }

        @Override
        public void onProviderEnabled(String provider) { }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) { }

}
```

Figure 7: Service

Miscellaneous stuff

Need to add the service in the manifest file. Need to start the service.

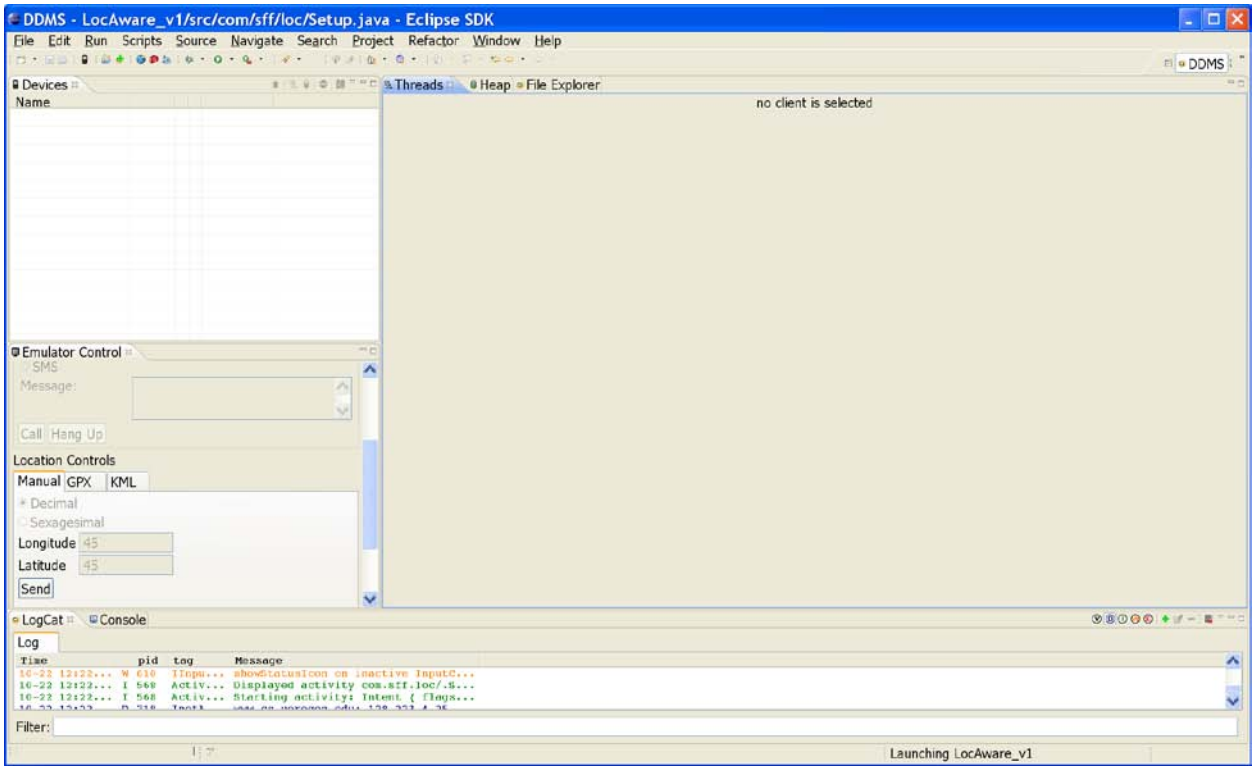

Figure 8: simulating movement

Extra Credit

You may notice that I am using onStart and onDestroy in figure 7. I am doing so because in my version of this application, I decided to allow the user to stop and start the service. Why? Because on a real phone, GPS applications seem to suck battery life. So I wanted to allow the user to shut the service off when not using it. I'll give you extra credit if you give this capability to the user. You will need to add a checkbox to the activity: when checked, start the service; when unchecked, stop the service (i.e., `stopService(intent)`). Might also want to add a static flag to the service that notes whether it is running or not. Outsiders can look at the flag to see service state.

Part 4: Deal with a triggered reminder

We will use another activity to display a triggered reminder. It is fairly straightforward. The main concept I would like you to learn is passing data as extras.

In the new activity, the first component you will need is a text area to show the message.

The second component is a button labeled "Dismiss". If the user clicks this button, you should not delete the reminder: allow it to trigger again in the future.

The third component is a button labeled "Delete". If the user clicks this button, you should delete the reminder from Reminders so that it is gone from your application: it will not trigger again.

You line up the 3 components on the screen in any way that is pleasing to you.

What new things will you need

You will need a new activity, which handles the display screen. Your service should launch this activity when it notices that the phone has moved in range of a reminder. When you start an activity from a service, you will need to include the following in the service:

```
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

My current understanding of the need for this flag is that the service is not running as part of an application. Instead, it is running in the background no matter what activity/application is in the foreground. Hence, starting an activity is viewed as a new task (not in the service's local scope). On the other hand, starting one activity from another in the same application uses the same task/application framework. So you do **not** need to include this flag for activity-to-activity activation within the same application. I might be wrong about this so welcome others comments.

Don't forget to update the manifest to include your new activity.

How does the display activity know what to display?

I think the simplest way to solve this problem is to use "extras", primitives and objects you can add to an intent from the service. When the display is created, it can pull these extras out and use them. See the sample code below.

```
//In the service
Intent intent = new Intent();
intent.setClassName(this, "edu.uoregon.test.Activity2");
intent.putExtra("myExtra", new Integer(1));   //this adds an extra
startActivity(intent);

//In Activity2's onCreate() method
Intent callingIntent = getIntent();
Integer i = (Integer)callingIntent.getSerializableExtra("myExtra");
```

Note that this only works because the `Integer` class implements `Serializable`. In general, if we do this:

```
intent.putExtra("whatever", obj);
```

then obj must be a `Serializable` object. The bad news is that I bet your `ReminderEntry` class is not `Serializable` at the moment; this is the object you want to pass. The good news is that it is easy to make it serializable. Just edit your class def as follows:

```
class ReminderEntry implements Serializable {
```

Then you should be good to go: you can pass an instance of `ReminderEntry` to the display activity using `putExtra` method. The display activity can pull out the pieces and display them on the screen. Cool.

How do I get back to the reminder-entry screen

If you just **`finish()`** from the display activity, the reminder-entry activity should automatically pop back into view. No need to try to start it.

What's Missing

It might be useful to read reminders, in xml format, from a file. This is handy - allows a web-app to define reminders and store them on server. Phone loads them when it starts up. We will need to get back to this to set up our study.

I also played around with storing reminders on the phone. I have code that will dump the Reminders class to a local file and then reload that file on demand. Not as fancy as sql database, but works ok.