fCSCI 448 – Lab 12A
Wednesday, April 19, 2023
LAB IS DUE BY **Friday, April 28, 2023 11:59 PM**!!

Now that players of Samodelkin can view themselves on a map, the larger plan is to notify players when they are geospatially near another player. This next proof of concept will setup the notification component to alert the user of their current location.

Step 0 – Add the Trigger

Modify `LocationScreen` to add a second button next to the "`Get Current Location`" Button. This button should say "`Notify Me Later`".

This button will only be enabled if the location parameter received by composable is not null.

The `onClick` argument will call through to be created function called `onNotify` and we will pass this function the non-null location.

```
Button(
  enabled = (location != null),
  onClick = { onNotify(location!!) }
) { ... }
```

Note: the not-null assertion operator is safe to call since the button can only be clicked if location isn't null.

Now add the parameter to `LocationScreen` to correspond to `onNotify`.

```
LocationScreen(
  ...,
  onNotify: (Location) -> Unit
)
```

Step 1 – Fire the Alarm

In `MainActivity`, the definition for `onNotify` needs to be supplied. What to do? Start by logging the event has taken place.

We're going to test our notification ability from the background by setting an alarm to perform the notification ten seconds in the future.

Part 1.I – Make the Alarm class

In order to schedule alarms for an exact time, we need permission to do so. Add to the Manifest the following permission

```
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM"/>
```

Now, create a new class called `LocationAlarmReceiver` that extends `BroadcastReceiver` and add a public mutable nullable location member field initialized to null. Also, stub out a member function called `scheduleAlarm` that accepts an `Activity` as a parameter. Lastly, override and stub out `onReceive`.

```
class LocationAlarmReceiver : BroadcastReceiver() {
  var lastLocation: Location? = null

  fun scheduleAlarm(activity: Activity) {
    // Part 1.II
  }

  override fun onReceive(context: Context, intent: Intent) {
    // Part 1.III
  }
}
```

Part 1.II – Schedule the Alarm

Inside the `scheduleAlarm` function, first get a reference to the `AlarmManager`.

```
val alarmManager = activity.getSystemService(Context.ALARM_SERVICE) as AlarmManager
```

Then create an explicit intent for the `LocationAlarmReceiver` class. The following will all go into a companion object on the class. The action will be a custom action specific to our receiver (set it as a constant). Also, add as two extras for the latitude and longitude. Make a createIntent function to bundle it all together.

```
private const val ALARM_ACTION = "448_ALARM_ACTION"
private const val EXTRA_LATITUDE = "latitude"
private const val EXTRA_LONGITUDE = "longitude"
private fun createIntent(context: Context, location: Location?): Intent {
  val intent = Intent(context, LocationAlarmReceiver::class.java).apply {
    action = ALARM_ACTION
    putExtra(EXTRA_LATITUDE, location?.latitude ?: 0.0)
    putExtra(EXTRA_LONGITUDE, location?.longitude ?: 0.0)
  }
  return intent
}
```

Store the result of this newly created function back the `scheduleAlarm` method.

```
val intent = createIntent(activity, lastLocation)
```

Next, package the intent inside a PendingIntent to broadcast.

```
val pendingIntent = PendingIntent.getBroadcast(
    activity,
    0,
    intent,
    PendingIntent.FLAG_IMMUTABLE
)
```

Then, specify the time the alarm should fire. Set it to be ten seconds from now. Log the time it will fire.

```
val alarmDelayInSeconds = 10
val alarmTimeInUTC = System.currentTimeMillis() + alarmDelayInSeconds * 1_000L
Log.d(LOG_TAG, "Setting alarm for ${SimpleDateFormat("MM/dd/yyyy HH:mm:ss",
Locale.US).format(Date(alarmTimeInUTC))}")
```

We are now ready to schedule the alarm. The proper flow and check for this operation is Android version dependent starting with API 31. We are supporting back to API 29 so our code needs to handle all users properly. We'll check at runtime what version of the OS is running. If it's before API 31, then we can schedule the alarm as desired. If it's API 31 or later, then we'll need to ensure the user has enabled the setting to do so.

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
  Log.d(LOG_TAG, "running on Version S or newer, checking if can schedule exact alarms")
  if (alarmManager.canScheduleExactAlarms()) {
    Log.d(LOG_TAG, "can schedule exact alarms")
    alarmManager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP,
                                           alarmTimeInUTC,
                                           pendingIntent)
  } else {
    Log.d(LOG_TAG, "can't schedule exact alarms, launching intent to bring up settings")
    val settingsIntent = Intent(Settings.ACTION_REQUEST_SCHEDULE_EXACT_ALARM)
    startActivity(activity, settingsIntent, null)
  }
} else {
  Log.d(LOG_TAG, "running on Version R or older, can set alarm directly")
  alarmManager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP,
                                         alarmTimeInUTC,
                                         pendingIntent)
}
```

If we were to run our code right now, the alarm would go off in ten seconds. However, there is nobody listening for the alarm. We need to tell our receiver what to do when it is notified that the alarm has gone off.

Part 1.III – Handle the Alarm

Inside of `onReceive`, first log the action of the intent received.

```
Log.d(LOG_TAG, "received alarm for action ${intent.action}")
```

We will want to make sure the `action` received corresponds to our action. If it does, then unpack the latitude and longitude off the intent and log their values.

```
if (intent.action == ALARM_ACTION) {
  val latitude = intent.getDoubleExtra(EXTRA_LATITUDE, 0.0)
  val longitude = intent.getDoubleExtra(EXTRA_LONGITUDE, 0.0)
  Log.d(LOG_TAG, "received our intent with $latitude / $longitude")
}
```

In order for the OS to broadcast the alarm intent to our receive, we need to statically register our receiver with the OS. Therefore, this information must go in the AndroidManifest.xml file.

```
<manifest>
  <application>
    <receiver
      android:name=".LocationAlarmReceiver"
      android:exported="false"
      android:enabled="true">
      <intent-filter>
       <action android:name="448_ALARM_ACTION">
     </intent-filter>
```

Part 1.IV – Fire the Alarm

We can now kick off the entire process. Back in `MainActivity`, first create a private member field for the `locationAlarmReceiver`.

```
private val locationAlarmReceiver = LocationAlarmReceiver()
```

Then in the `onNotify` lambda, first set the location then call our `scheduleAlarm` method.

```
onNotify = { lastLocation ->
  locationAlarmReceiver.lastLocation = lastLocation
  locationAlarmReceiver.scheduleAlarm(this@MainActivity)
}
```

Build, deploy, and run the app. Initially, the notify button should be disabled. Get a location and the button will become enabled. Press the button and follow the logs. You should see the message for what time the alarm will go off, then after delay the message that the alarm was received.

Step 2 – Post the Notification

Just like schedule exact alarms is Android version dependent, so is posting notifications. The change has begun with API 33, and again we are supporting back to API 29. Beginning with API 33, the user needs to grant explicit permission for an application to post a notification.

Add to the manifest, the following permission and update the receiver to require the corresponding permission.

```
<manifest>
  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
  <application>
    <receiver
      ...
      android:permission="android.permission.POST_NOTIFICATIONS">
```

Part 2.I – Check for Permission to Post Notification

Make the `scheduleAlarm` function private on `LocationAlarmReceiver`.

Create a second public function called `checkPermissionAndScheduleAlarm` that accepts an `Activity` and an `ActivityResultLauncher` as parameters.

```
fun checkPermissionAndScheduleAlarm(
  activity: Activity,
  permissionLauncher: ActivityResultLauncher<String>
) {
  // TODO
}
```

Here, we will go through our permission check flow – but only if on API 33 or newer. We'll then call through to our `scheduleAlarm` method when appropriate.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
  Log.d(LOG_TAG, "running on Version Tiramisu or newer, need permission")
  if (activity.checkSelfPermission(Manifest.permission.POST_NOTIFICATIONS)
        == PackageManager.PERMISSION_GRANTED) {
    Log.d(LOG_TAG, "have notification permission")
    scheduleAlarm(activity)
  } else {
    if (ActivityCompat
         .shouldShowRequestPermissionRationale(activity,
                                               Manifest.permission.POST_NOTIFICATIONS) {
      Log.d(LOG_TAG, "previously denied notification permission")
      // display toast with reason
    } else {
      Log.d(LOG_TAG, "request notification permission")
      permissionLauncher.launch( Manifest.permission.POST_NOTIFICATIONS )
    }
  }
} else {
  Log.d(LOG_TAG, "running on Version S or older, post away")
  scheduleAlarm(activity)
}
```

Part 2.II – Make The Permission Launcher

Since we may need to request permissions, in `MainActivity` we'll need to setup another permission launcher. It will mirror the `locationPermissionLauncher`. This time the callback will call back through to `locationAlarmReceiver.checkPermissionAndScheduleAlarm`.

The `onNotify` lambda will also need to be updated to use the new `checkPermissionAndScheduleAlarm` method.

At this point, the app can be run to verify the permission flow (use the logs to ensure the correct flow is occurring).

Part 2.III – Post the Notification!

The final step is to actually post the notification when the alarm is received. We need to finish out the `onReceive()` method of our `LocationAlaramReceiver`.

For good practice, first check that we do indeed have permission to post the notification.

```
if (ActivityCompat
      .checkSelfPermission(context, Manifest.permission.POST_NOTIFICATIONS)
         == PackageManager.PERMISSION_GRANTED) {
  Log.d(LOG_TAG, "have permission to post notifications")
  // have permission, TODO post
}
```

First, get a reference to the notification manager.

```
val notificationManager = NotificationManagerCompat.from(context)
```

Then specify the channel ID, name, description.Create the corresponding channel.

```
val channel =
  NotificationChannel(
    channelID,
    channelName,
    NotificationManager.IMPORTANCE_DEFAULT
  ).apply {
    description = channelDesc
  }
notificationManager.createNotificationChannel(channel)
```

Next, create the notification.

```
val notification = NotificationCompat.Builder(context, channelID)
  .setSmallIcon(android.R.drawable.ic_dialog_map)
  .setContentTitle(notificationTitle)
  .setContentText(notificationText)
  .build()
```

Set the title to be "`You are here!`" and the text to be in the format "`You are at` *`lat / long`*" with lat and long replaced with the latitude and longitude received on the intent.

The last step is to post the notification!

```
notificationManager.notify(0, notification)
```

With everything in place – build, deploy, run, and press the notify button. After short delay of following the log messages, you'll see your notification appear in the top notification bar. To further test the receiver, after pressing the notify button you can close your app. The device will wakeup when the alarm fires and post the notification.

Step 3 – Next Steps

The final test just performed has the user elsewhere on their device when the notification arrives. The next step is to bring the user back into the app when they click on the notification.

LAB IS DUE BY **Friday, April 28, 2023 11:59 PM**!!