Labsheet 9 – Saturday 20th November, 2021

Android Wear Development and Working with Device Sensors

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/pvo-osiz-cuh

Introduction

Wear OS (formerly Android Wear) is a version of Google's Android operating system designed for smartwatches and other wearables. From a coding perspective, working with a wearable project doesn't have to be significantly different from any other Android project. However, the user interface is different because you work with a significantly smaller device.

Smart watches differ from smart phones in terms of design and interactivity due to distinct usage circumstances and smaller screens. The layout is simpler and more reliant on swiping actions to operate. In real life, an Android Wear device seldom runs on its own. Instead, the wearable runs in concert with a phone.

Task 1: Create new Project and Wear AVD

1. New Project: In Android Studio, start a new project and select the Wear OS option. Choose the Blank Wear Activity template and click Next. The Pair with Empty Phone App option on this page lets you pair your wearable app with a phone. However, Android recommends against this as the wear app should run independently of a phone app. If you do check the Pair with Empty Phone App option, you will have a dual project containing both a mobile and a wear folder.

   We can test the app on an actual watch by uploading the app using Wi-Fi or Bluetooth; however we will use the emulator for testing inplace of an actual smartwatch (or other wearable device).

2. New AVD: Open the Android Virtual Device Manager and click the Create Virtual Device button. In the Category list, select Wear OS. Select Android Wear Square and click Next.

   Unlike the Phone category, the Wear OS category focuses on device types rather than specific device models. All these devices have the same density and slightly varying resolutions so the only real difference is the shape.

3. Select a system image: The system image gives you an installed version of the Wear OS. You can choose the version on the top. Click on the download link and wait for the installation to finish.

4. Verify the AVD configuration: On the next screen, verify the AVD configuration. This screen summarizes the options you choose over the last few screens, and gives you the option of changing them. Click on Finish when done. Run the AVD.

5. Testing: For the full Android Wear experience, you may want two devices — a wearable and a phone. For this lab exercise, we will ignore the phone and use an emulated Android Wear device. But the other ways to set up a testing environment are.

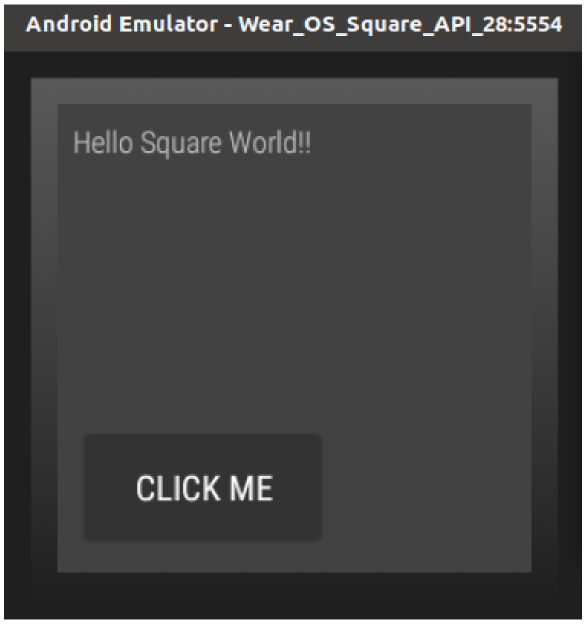
Connect a real phone to your development computer and use an emulated Android Wear device » Install the Android Wear app from the Google Play Store on your phone. » Use the Developer options in the phone's Settings screen to enable USB debugging on the phone. » **(With an Android Wear emulator)** Type the following command in your development computer's command prompt window or Terminal app: `adb -d forward tcp:5601 tcp:5601` This command forwards your emulator's communications to the connected phone. » **(On a real wearable device)** Use the Developer options in the device's Settings screen to enable ADB debugging on the Wear device. » In the Android Wear app on the phone, pair the phone with the emulator (or with the real wearable device).

For connecting a real wearable device
 	
Connect the device to your development computer using a USB cable or Create a Bluetooth connection between the phone and the wearable. To do that
 	
Enable the Debugging over Bluetooth option on the wearable. You also have to set up Debugging over Bluetooth in the Android Wear app on the phone and type the commands
`adb forward tcp:4444 localabstract:/adb-hub`
`adb connect localhost:4444`
on your development computer.

Questions:

1. Why is there an error in FrameLayout of MainActivity Layout when the project is first created?

Try it yourself: Add a Button on MainActivity Screen which will change the Hello World text on clicking.



Special wearable classes

» BoxInsetLayout: Defines a layout that works well with round screens by creating a box, within which you place the display elements.

» SwipeDismissFrameLayout: Enables a user to dismiss any view by swiping the screen from left to right. You need this ability when an app, such as a map view, needs some method for dismissing the activity as a whole. The article Exit full-screen activities on Wear tells more about this layout and its use.

» WearableRecyclerView: Provides a curved layout, such as the layout used for the main Wear application launcher. This view maximizes the use of screen real estate so that you don't have to create a scrolling list of options but can instead place the options around the outside circumference of the display. Short options appear at the top and bottom of the list because the top and bottom have the least space to offer. You can see more about this layout at Create lists on Wear OS.

» AmbientModeSupport: Provides ambient mode support when used with the AmbientModeSupport. AmbientCallbackProvider interface. Ambient mode means that your app remains displayed at all times, such as in a particular kind of watch face. You aren't actually expecting the user to do much other than look. Interactive mode is what most developers think about because the user is doing something with the app. You can read more about this app at Keep your app visible on Wear.

Task 2: Watch face project

Wear OS is a wearable platform designed for small, powerful devices, worn on the body. It is designed to deliver useful information when you need it most, intelligent answers to spoken questions, and tools to help reach fitness goals.

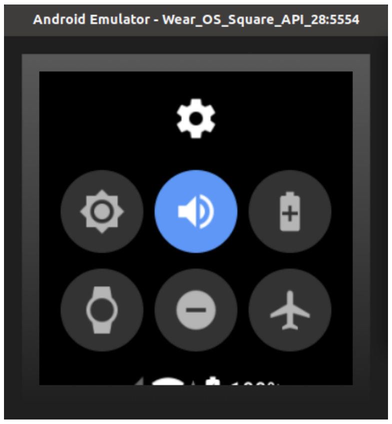
Being such a personal device, style is a big part of it. Aside from offering a choice of manufacturers, customisable watch faces give users even more ways to express their personal style.

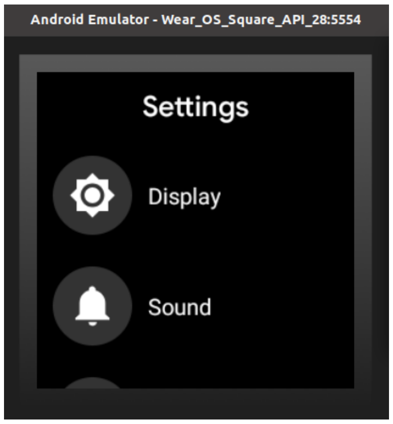
A watch face is essentially a native service that runs in the background on an Wear OS device. Within this service, there is an engine that renders each screen. You can think of this as an animator flipping through a book of moving cartoon drawings.

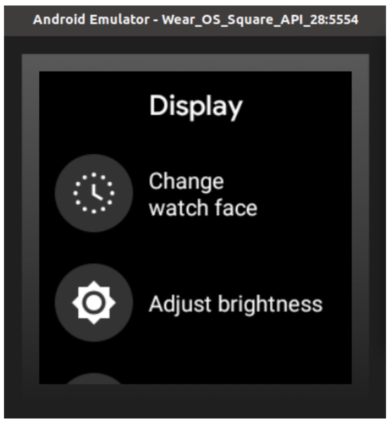
1. New Project: Create a new project. This time instead of Blank Activity choose Watch Face as a Template. (Note: A Watch Face isn't an activity)

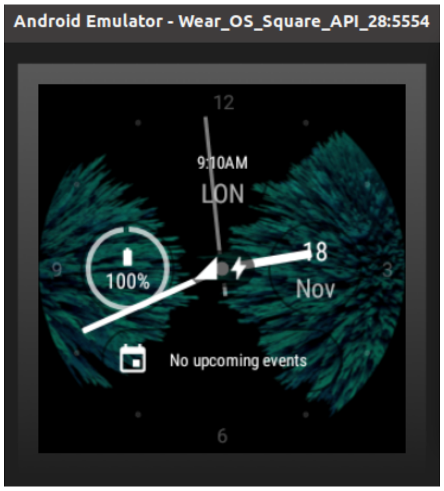
2. Run the project: Since we are launching a service and not an activity you can see that in Run Configuration (Run -> Edit Configurations) the Launch Option is set as Nothing instead of Default Activity. This project will be affecting the emulator wallpaper.

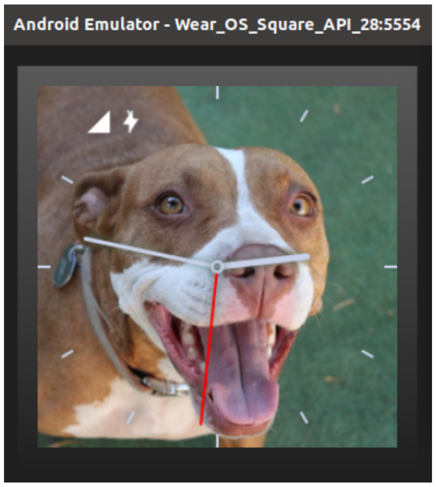
3. Change Watch Face: When you see the default screen of the watch, drag down and go to Settings. In Settings choose Display. In the Display choose Change watch face. Select **Elements Analog** or **My Analog** from See more watch faces. You can explore more by changing the style or background Image.











Let us add the date to the watch face (and you can use this approach for your own additions):

MyWatchFace.java

```
// import java.time.LocalDateTime;
// import java.time.format.DateTimeFormatter;


@Override
public void onDraw(Canvas canvas, Rect bounds) {
   long now = System.currentTimeMillis();
   mCalendar.setTimeInMillis(now);

   drawBackground(canvas);
   drawDate(canvas);
   drawWatchFace(canvas);
}

private void drawDate(Canvas canvas) {
   Paint paint = new Paint();
   paint.setColor(Color.WHITE);
   paint.setStyle(Paint.Style.FILL);
   canvas.drawRect(90f, 160f, 200f, 190f, paint);
   paint.setColor(Color.BLACK);
   paint.setTextSize(20f);
   LocalDateTime localDate = LocalDateTime.now();
   canvas.drawText(localDate.format(DateTimeFormatter.ofPattern("M/d/y")), 95f,
180f, paint);
}
```

References

[android/wear-os-samples: Multiple samples showing best practices in app and watch face development on Wear OS.]

[Create a watchface for Wear OS]

https://www.youtube.com/watch?v=VkvHKtmsvYA

https://www.youtube.com/watch?v=AK38PJZmIW8

Working with Device Sensors

Introduction to Sensors

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

The Android platform supports three broad categories of sensors:

- Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

- Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

- Position sensors

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

You can use most Android devices in either portrait or landscape mode. So as a developer, you must design your app’s interface with both modes in mind.

If you hold the device so that the screen’s height is greater than the screen’s width, the screen’s orientation is **portrait**. If you hold the device so that the screen’s width is greater than the screen’s height, the screen’s orientation is **landscape**.

If you lay the device flat on the ground so that the top of the device points to the Earth’s magnetic North Pole, the device’s yaw, pitch, and roll values are all 0. If you keep the device flat on the ground while you turn the device, you’re changing the **yaw**. If you lift the top of the device (while the bottom of the device still touches the ground) you’re changing the **pitch**. If you lift one side of the device (while the other side still touches the ground) you’re changing the **roll**. Android uses radians to measure yaw, pitch, and roll.

Android Sensor API (Sensor)

Android sensor API provides many classes and interfaces. The important classes and interfaces of sensor API are as follows:

1. SensorManager class

The **android.hardware.SensorManager** class provides methods :

- to get sensor instance,
- to access and list sensors,
- to register and unregister sensor listeners etc.

Instances of the Manager classes connect your code to the device's hardware sensors. Calling getSystemService provides access to sensor managers. You can get the instance of SensorManager by calling the method getSystemService() and passing the SENSOR_SERVICE constant in it.

SensorManager sm = (SensorManager)getSystemService(SENSOR_SERVICE);

2. Sensor class

The **android.hardware.Sensor** class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc. Instances of android.hardware.Sensor represent the sensors themselves.

3. SensorEvent class

Its instance is created by the system. It provides information about the sensor.

4. SensorEventListener interface

It provides two callback methods to get information when sensor values (x,y and z) change **onSensorChanged()** or sensor accuracy changes **onAccuracyChanged()**.

Android Location API (Location)

Using the Location Manager | Android Developers

The LocationManager isn't in the android.hardware package because sensing location is abstracted for various sensing techniques. The LocationManager class represents GPS readings, cell tower usage, and other things. You can estimate a device's location based on Wi-Fi hotspot usage, the device's IP address, user queries, last known location, and readings borrowed from nearby mobile devices. The LocationManager deals generically with places on Earth, not specifically with GPS hardware.

Try it yourself: Display Yaw,Roll and Pitch values using android.hardware.Sensor and the latitude and longitude values using android.location.Location.

MainActivity.java

```
package com.example.mysensorapplication;

import static android.hardware.Sensor.TYPE_ACCELEROMETER;
import static android.hardware.Sensor.TYPE_MAGNETIC_FIELD;

import android.Manifest;
```

```java
import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.pm.PackageManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Build;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;

import static android.Manifest.permission.ACCESS_COARSE_LOCATION;
import static android.Manifest.permission.ACCESS_FINE_LOCATION;

import androidx.appcompat.app.AlertDialog;
import androidx.core.app.ActivityCompat;

public class MainActivity extends Activity {

   private ArrayList permissionsToRequest;
   private ArrayList permissionsRejected = new ArrayList();
   private ArrayList permissions = new ArrayList();

   SensorManager sensorManager;
   Sensor magFieldSensor, accelerometer;
   SensorEventListener sensorListener;
   LocationListener locationListener;
   LocationManager locationManager;
   TextView orientationView, locationView;

   private float[] gravityValues = new float[3];
   private float[] geoMagnetValues = new float[3];
   private float[] orientation = new float[3];
   private float[] rotationMatrix = new float[9];

   private final static int ALL_PERMISSIONS_RESULT = 101;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       sensorManager = (SensorManager)
               getSystemService(Context.SENSOR_SERVICE);
       magFieldSensor = sensorManager
               .getDefaultSensor(TYPE_MAGNETIC_FIELD);
       accelerometer = sensorManager
               .getDefaultSensor(TYPE_ACCELEROMETER);
       sensorListener = new MySensorEventListener();
       locationListener = new MyLocationListener();
       locationManager = (LocationManager)
               getSystemService(Context.LOCATION_SERVICE);
       orientationView =
               (TextView) findViewById(R.id.orientationView);
```

```
        locationView =
                (TextView) findViewById(R.id.locationView);
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(sensorListener,
                magFieldSensor, SensorManager.SENSOR_DELAY_UI);
        sensorManager.registerListener(sensorListener,
                accelerometer, SensorManager.SENSOR_DELAY_UI);
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {
            permissions.add(ACCESS_FINE_LOCATION);
            permissions.add(ACCESS_COARSE_LOCATION);

            permissionsToRequest = findUnAskedPermissions(permissions);
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {


                if (permissionsToRequest.size() > 0)
                    requestPermissions((String[]) permissionsToRequest.toArray(new
String[permissionsToRequest.size()]), ALL_PERMISSIONS_RESULT);
            }

            return;
        }
        locationManager.requestLocationUpdates
                (LocationManager.NETWORK_PROVIDER,
                        10, 1, locationListener);
    }


    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(sensorListener);
        locationManager.removeUpdates(locationListener);
    }


    class MySensorEventListener implements SensorEventListener {
        @Override
        public void onSensorChanged(SensorEvent event) {
            int sensorEventType = event.sensor.getType();
            if (sensorEventType == Sensor.TYPE_ACCELEROMETER) {
                System.arraycopy
                        (event.values, 0, gravityValues, 0, 3);
            } else if (sensorEventType ==
                    Sensor.TYPE_MAGNETIC_FIELD) {
                System.arraycopy
                        (event.values, 0, geoMagnetValues, 0, 3);
            } else {
                return;
            }
            if (SensorManager.getRotationMatrix(rotationMatrix,
                    null, gravityValues, geoMagnetValues)) {
                SensorManager.getOrientation(rotationMatrix,
                        orientation);
                orientationView.setText
```

```java
                        ("Yaw: " + orientation[0] + "\n"
                                + "Pitch: " + orientation[1] + "\n"
                                + "Roll: " + orientation[2]);
            }
        }
        @Override
        public void onAccuracyChanged(Sensor sensor,
                                      int accuracy) {
            if (accuracy <= 1) {
                Toast.makeText(MainActivity.this, "Please shake the " +
                        "device in a figure eight pattern to " +
                        "improve sensor accuracy!", Toast.LENGTH_LONG)
                        .show();
            }
        }
    }
    class MyLocationListener implements LocationListener {
        @Override
        public void onLocationChanged(Location location) {
            locationView.setText
                    ("Latitude: " + location.getLatitude() + "\n"
                            + "Longitude: " + location.getLongitude());
        }
        @Override
        public void onProviderDisabled(String provider) {
        }
        @Override
        public void onProviderEnabled(String provider) {
        }
        @Override
        public void onStatusChanged(String provider,
                                    int status, Bundle extras) {
        }
    }
    private ArrayList findUnAskedPermissions(ArrayList wanted) {
        ArrayList result = new ArrayList();

        for (Object perm : wanted) {
            if (!hasPermission((String) perm)) {
                result.add(perm);
            }
        }
        return result;
    }

    private boolean hasPermission(String permission) {
        if (canMakeSmores()) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                return (checkSelfPermission(permission) ==
PackageManager.PERMISSION_GRANTED);
            }
        }
        return true;
    }

    private boolean canMakeSmores() {
        return (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP_MR1);
    }


    @TargetApi(Build.VERSION_CODES.M)
    @Override
```

```
   public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults) {

       switch (requestCode) {

           case ALL_PERMISSIONS_RESULT:
               for (Object perms : permissionsToRequest) {
                   if (!hasPermission((String) perms)) {
                       permissionsRejected.add(perms);
                   }
               }

               if (permissionsRejected.size() > 0) {


                   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                       if (shouldShowRequestPermissionRationale((String)
permissionsRejected.get(0))) {
                           showMessageOKCancel("These permissions are mandatory for
the application. Please allow access.",
                                   new DialogInterface.OnClickListener() {
                                       @Override
                                       public void onClick(DialogInterface dialog,
int which) {
                                           if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.M) {
                                               requestPermissions((String[])
permissionsRejected.toArray(new String[permissionsRejected.size()]),
ALL_PERMISSIONS_RESULT);
                                           }
                                       }
                                   });
                           return;
                       }
                   }

               }

               break;
       }

   }

   private void showMessageOKCancel(String message, DialogInterface.OnClickListener
okListener) {
       new AlertDialog.Builder(MainActivity.this)
               .setMessage(message)
               .setPositiveButton("OK", okListener)
               .setNegativeButton("Cancel", null)
               .create()
               .show();
   }
}
```

Objects that implement Listener interfaces receive notice of changes to sensor values. Instances of MySensorEventListener and MyLocationListener fill these roles. Register the listeners in the activity's onResume method and unregister the listeners in the activity's onPause method. Your app should stop listening when the activity pauses. If you forget to unregister, the user's battery might die of exhaustion.

Getting location information means simply calling location.getLatitude() and location.getLongitude(). Feed values from the device's level gravity sensor or the device's magnetometer into the SensorManager.getRotationMatrix method.

Few Points:

- To sense the device's location, your app must have "android.permission.ACCESS_FINE_LOCATION". Sensing orientation requires no particular permission.
- Calls to registerListener have delay parameters which tells the device how often to check the sensor's value. The choices are SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME, SENSOR_DELAY_NORMAL, and SENSOR_DELAY_UI. The SENSOR_DELAY_GAME value is appropriate for game playing, and the SENSOR_DELAY_UI value is best for displaying the information.
- When you implement the SensorEventListener interface, you must create an onAccuracyChanged method. The predefined accuracy values are SENSOR_STATUS_UNRELIABLE with int value 0, SENSOR_STATUS_ACCURACY_LOW with int value 1, SENSOR_STATUS_ACCURACY_MEDIUM with int value 2, and SENSOR_STATUS_ACCURACY_HIGH with int value 3. For some reason, shaking the device in a figure-eight pattern tends to improve orientation sensitivity.

Homework

Build a Step counter / Heart rate monitor on the wearable device.
google/wear-sensors
Android Wear activity that reads and displays sensor data from the device