Android programming

TX00CK66 Sensor Based Mobile Applications

Lecture Graphs

Jarkko.Vuori@metropolia.fi

Drawing graphs

- It is possible to draw graphs directly to the screen using 2D drawing
  - If you need animation, you create a Canvas object and draw directly to it
  - If slow animation is enough, then drawing directly to the View is possible
    - In this case we extend the View class (i.e. create a custom view), and draw to the canvas given in the onDraw() method, e.g.

```
@Override
protected void onDraw(Canvas canvas) {
  int x = getWidth() / 2;
  int y = getHeight() / 2;
  int radius = y;

  paint.setColor(Color.parseColor("#ffffff");
  canvas.drawCircle(x, y, radius, paint);
}
```

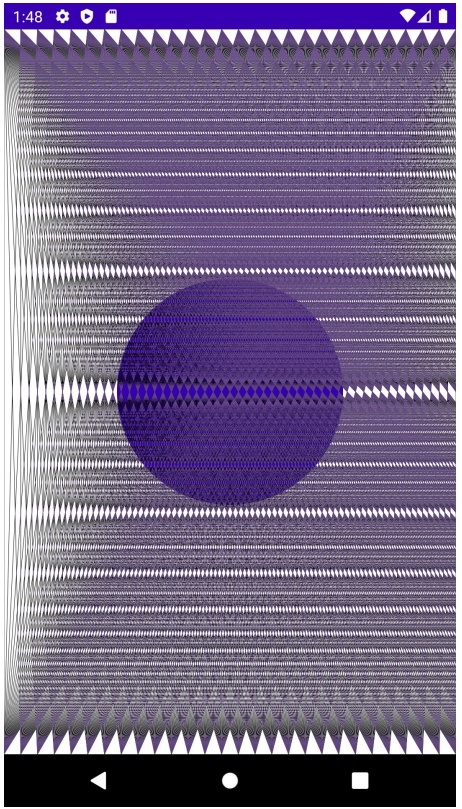
```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(new myview(this));
    }

    class myview extends View {
          public myview(Context context) {
              super(context);
          }

          @Override
          protected void onDraw(Canvas canvas) {
          }
    }
}
```

- 3D graphics support is provided through OpenGL library
  - GLSurfaceView class makes it easier to use OpenGL rendering in your applications

Drawing graphs in Jetpack Canvas

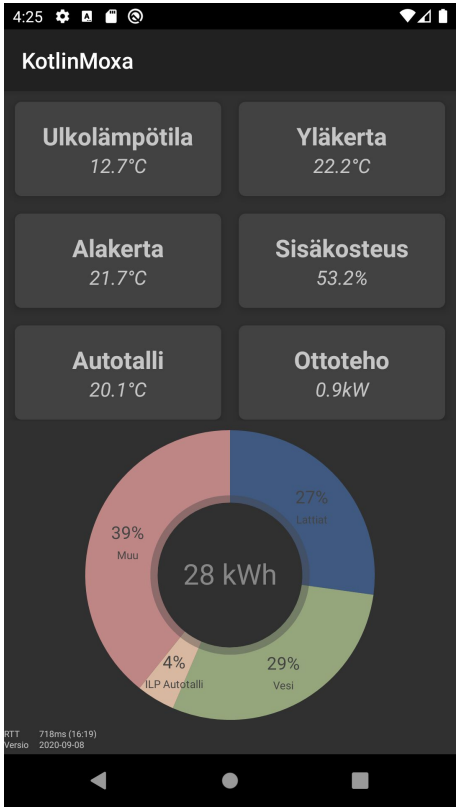
- Not so much different than Layout method

```
@Composable
fun DrawCanvas() {
    Canvas(modifier = Modifier.fillMaxSize()) {
        val gran: Int = 40;
        val canvasWidth = size.width
        val canvasHeight = size.height
        drawCircle(
            color = Purple700,
            center = Offset(x = canvasWidth / 2, y = canvasHeight / 2),
            radius = size.minDimension / 4
        )
        for (x in 0..canvasWidth.toInt()/gran) {
            for (y in 0..canvasHeight.toInt()/gran) {
                drawLine(
                    start = Offset(x = x.toFloat() * gran, y = 0f),
                    end = Offset(x = y.toFloat() * gran, y = canvasHeight),
                    color = Purple200,
                    blendMode = BlendMode.Xor
                )
            }
        }
    }
}
```



Drawing graphs

- In most cases it is easier to use a special library to create those graphs from your data
- There are many graph libraries available for the Android system, e.g.:
  - AAChartCore-Kotlin, https://github.com/AAChartModel/AAChartCore-Kotlin
  - AndroidPlot, https://github.com/halfhp/androidplot
  - Graph-View, http://www.android-graphview.org/
  - MP Android Chart, https://github.com/PhilJay/MPAndroidChart
  - AnyChart, https://github.com/AnyChart/AnyChart-Android
- For Jetpack Canvas there are currently only simple libraries available
  - Decent, https://github.com/tehras/charts
  - Very simple, https://github.com/Madrapps/plot
- As an example, we are using the MPAndroidChart library
  - Mainly because it's easy to use and has fairly documentation (problem is that it is not very actively maintained)



Pie graph is done with MP Android Chart version 3.1.0

Android Studio libraries

- It is easy to include any library to your application
  - Add the following line of dependency in **module's** build.gradle file

```
dependencies {
        ...
    implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
        ...
}
```

  - Then graphview library is available to your application
- Android Studio downloads the library from Maven Repository Server we defined in build.gradle
  - Apache Maven is a tool set developed by Apache. It provides e.g. a file server to distribute the libraries
- There are two standard Maven Repository servers; google and Maven Central
  - It is also possible to setup one's own Maven Repository server
- In order to use e.g. JitPack server (https://jitpack.io/), you need to define it in your project build.gradle file as (need to be done in settings.gradle)

```
repositories {
    google()
    mavenCentral()
    maven { url 'https://jitpack.io' }
}
```

MPAndroidChart

- Open source graph plotting library for Android
- Available graph types
  - Line Graphs
  - Bar graphs
  - Point Graphs
  - Custom type
  - Combination of the previous (dual axis graph)
- Scrolling and scaling/zooming is possible
- Responses to taps on the graph are allowed
- Legend and axis titles
- Line color, thickness, label font sizes/color can be changed
- Using MPAndroidChart is easy, documentation is here
  - https://weeklycoding.com/mpandroidchart-documentation/

Layout views in Composable

- If you have an ole widget/xml layout, you can use Androidview to convert it to Composable
  - If you have an xml-layout file, e.g. graph.xml and its widget is

```
<com.github.mikephil.charting.charts.LineChart
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/graph" />
```

- Then we can use AndroidView to wrap layout view to Composable

```
@Composable
fun PlotChart(values: ArrayList<Entry>) {
   AndroidView(
       modifier = Modifier.fillMaxSize(),
       factory = { context: Context ->
           val view = LayoutInflater.from(context)
               .inflate(R.layout.graph, null, false)
           val graph = view.findViewById<com.github.mikephil.charting.charts.LineChart>(R.id.graph)
           // do whatever you want…
           val data = LineData(LineDataSet(values, "Data"))
           graph.data = data
           view // return the view
       },
       update = { view ->
           // Update the view
       }
   )
}
```

MPAndroidChart in Composable

- MPChart graph constructors returns a View. If you need only one graph (the role of layout is minimum), you can use the following code

```
AndroidView(
  modifier = Modifier.fillMaxSize(),
  factory = { context: Context ->
      val view = LineChart(context)
      view.legend.isEnabled = false
      val data = LineData(LineDataSet( values, "BPM"))
      val desc = Description()
      desc.text = "Beats Per Minute"
      view.description = desc;
      view.data = data
    view // return the view
},
update = { view ->
    // Update the view
    view.invalidate()
}
)
```

MPAndroidChart

- MPAndroidChart is easy to use
  - To add data to your chart

```
YourData[] dataObjects = ...;
List<Entry> entries = new ArrayList<Entry>();
for (YourData data : dataObjects) {
    // turn your data into Entry objects
    entries.add(new Entry(data.getValueX(), data.getValueY()));
}
```

  - As a next step, you need to add the List<Entry> you created to a LineDataSet object. DataSet objects hold data which belongs together, and allow individual styling of that data. The below used “Label” has only a descriptive purpose and shows up in the Legend, if enabled.

```
LineDataSet dataSet = new LineDataSet(entries, "Label"); // add entries to dataset
dataSet.setColor(...);
dataSet.setValueTextColor(...); // styling, ..
```

  - Then you can give the data to the graph

```
val graph = view.findViewById<com.github.mikephil.charting.charts.LineChart>(R.id.graph)
graph.data = data
```

  - Entry is MPAndroidChart class who stores x and y values for one point (to be plotted)
    - x coordinate can be also Date value if your horizontal coordinate is a date/time value
    - Notice that Entry class takes two double type parameters and Kotlin is strictly typed language, so if you have integer arguments, you need to convert them to double

Reading list

- Drawing to Canvas
  - https://proandroiddev.com/building-your-first-custom-chart-in-android-with-jetpack-compose-a890fb60878b
- GraphView:
  - http://www.android-graphview.org/
- Android Studio libraries/Maven repository
  - https://inthecheesefactory.com/blog/how-to-upload-library-to-jcenter-maven-central-as-dependency/en