Fakultät für Informatik und Mathematik

OTH REGENSBURG

Übungen zum Fach ZAPP

2. RecyclerView

2.1 Basics

> Eine der wichtigsten Komponenten einer jeder App sind Listen, welche eine dynamische Anzahl von Einträgen darstellen können. Unter Android verwenden wir dazu im Rahmen dieser Übung die sogenannte `RecyclerView`.
>
> Ziele der Übung:
> - Anlegen eines RecyclerView Adapters zur Anzeige von beliebigen Listeneinträgen.
> - Recycling der einzelnen Zeilen der RecyclerView inklusive Erhalt deren Status.

Aufgaben

1. Legen Sie ein neues Projekt an. Als Template können Sie `Empty Activity` auswählen, dadurch wird automatisch die Einstiegs-Activity samt Layout generiert.
2. Fügen Sie folgende Zeile in das `build.gradle` file des app modules hinzu, diese ermöglich uns die Verwendung der RecyclerView

   > implementation ‘com.android.support:recyclerview-v7:28.0.0’’

3. Legen Sie ein neues Layout an oder verwenden Sie ein vorhandenes und fügen Sie das Layout Element für die `RecyclerView` hinzu

```
<android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

1. Erstellen Sie nun zusätzlich noch ein Layout für die Zeile der RecyclerView welche zur Anzeige der unterschiedlichen Einträge genutzt wird, bestehend aus einer `CheckBox` (`res -> layout -> Rechtsklick -> New -> Layout resource file`). Achten Sie darauf, dass das Root-Layout Element als Höhe `wrap_content` hat
2. Als nächstes legen wir den Adapter für die RecyclerView an. Verwenden Sie dazu folgende Boilerplate Klasse und lösen Sie die TODOs

```
public class ListAdapter extends
RecyclerView.Adapter<ListAdapter.ViewHolder> {
    private final LinkedList<String> mList;
    private LayoutInflater mInflater;

    ListAdapter(Context context, LinkedList<String> list) {
        mInflater = LayoutInflater.from(context);
        // TODO: Initalize list given by the constructor parameter
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        // TODO: Inflate item view
        View mItemView = mInflater.inflate(..., ..., false);
        return new CustomViewHolder(mItemView, this);
    }

    public void onBindViewHolder(ViewHolder viewHolder, int position)
{
        // TODO: Set text depending on the current position
        String mCurrent = ...
        viewHolder.mCheckBox.setText(...);
    }

    @Override
    public int getItemCount() {
        // TODO: Return size of data set
        return 0;
    }

    public class ViewHolder extends RecyclerView.ViewHolder {
        CheckBox mCheckBox;

        ViewHolder(@NonNull final View itemView) {
            super(itemView);

            // TODO: Get reference to checkbox via
itemView.findById(...)
        }
    }
}
```

Nun müssen wir nur noch unseren Adapter mit der RecyclerView verbinden. Erzeugen Sie eine Referenz auf die RecyclerView und fügen Sie folgende Zeilen in Ihrer MainActivity hinzu:

```
mAdapter = new ListAdapter(this, mList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

Erklärung: Die `RecyclerView` benötigt einen `LayoutManager` zum darstellen der Items. Wir wollen unsere Einträge untereinander darstellen, daher legen wir uns einen `LinearLayoutManager` an und setzen diesen für die RecyclerView.

Damit die RecyclerView etwas anzeigt, sollten wir selbstverständlich einen entsprechenden Datensatz übergeben (zweiter Übergabeparameter der Adapterklasse). Legen Sie also eine Liste, bestehend aus mehreren Strings, an und übergeben Sie diese der RecyclerView. Unkreative dürfen auch gerne nachfolgendes Code-Fragment verwenden:

```
ArrayList<String> items = new ArrayList<>();
Random rnd = new Random();
for (int i = 0; i < 50; i++)
    items.add(((char) ('A' + rnd.nextInt('Z' - 'A'))) + " " +
Integer.toString(i));
Collections.sort(items);
```

Führen Sie nun das Projekt aus: Ihre RecyclerView sollte nun alle Einträge anzeigen.

2.1 Interaktion und Optimierung

Nun möchten wir bei einem Klick auf eine Zeile der RecyclerView auch deren Position anzeigen lassen. Geben Sie dazu Ihrer `CheckBox` im Konstruktor des `CustomViewHolder` einen `OnClickListener` und geben Sie die Position mittels eines Toasts aus - zum Beispiel:

```
Toast.makeText(view.getContext(),
    String.format(Locale.GERMAN, "Position: %d is checked %s",
        getLayoutPosition(), mCheckBox.isChecked()),
    Toast.LENGTH_SHORT)
    .show();
```

Info zum Toast:

- Der erste Parameter ist der Kontext auf dem die Nachricht ausgegeben werden soll.
- Der zweite Parameter ist der auszugebende Text.
- Der dritte Parameter ist die Dauer der Anzeige des Textes. (`Toast.LENGTH_LONG` bzw. `Toast.LENGTH_SHORT`)

Wenn Sie nun die erste Zeile anklicken und weiter herunterscrollen, werden Sie feststellen, dass auf einmal auch eine weitere Zeile angewählt ist. Dies ist darauf zurückzuführen, dass die RecyclerView, wie der Name schon impliziert, die einzelnen rows recycled. Folglich müssen wir uns die angewählten Positionen merken und jedes Mal wenn `onBindViewHolder` aufgerufen wird, die CheckBox entsprechend an- oder abwählen.
Überlegen Sie sich eine Lösung und implementieren Sie diese zur Bewältigung des Problems.

Tipp: Ein SparseBooleanArray könnte hilfreich sein; den checked Zustand können Sie mittels CheckBox#setChecked(boolean val) festlegen