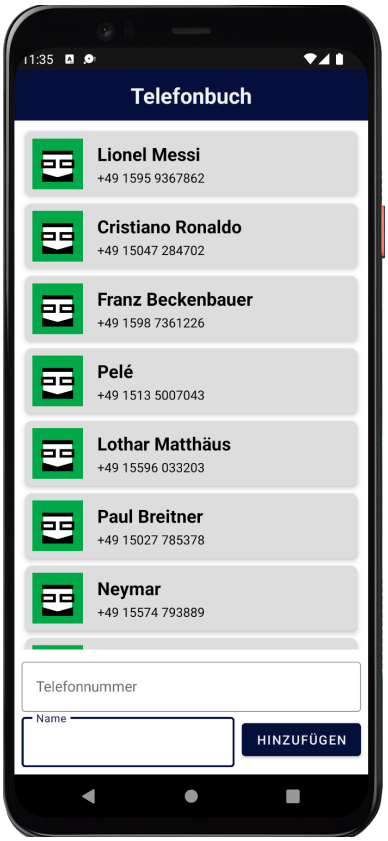
Kotlin 2.9 - Wiederholung

Aufgaben

Hinweis: Für diese Aufgabe gibt es keine Vorlage.

1. TELEFONBUCH - Wiederholung

In dieser Aufgabe programmierst du eine App zu 100 % selbst. Am Ende steht ein voll funktionales Telefonbuch.



- Öffne Android Studio und starte ein neues Projekt mit der Vorlage “Empty Activity” namens “Telefonbuch”
- Implementiere das Data Binding
    - Öffne zuerst die Datei `build.gradle(Module)`, um hier das Data Binding zu aktivieren

Hinweis: ⇨ siehe Folien 2.3 Seite 5

- Öffne nun das `activity_main.xml` Layout und konvertiere das darin enthaltene `ConstraintLayout` zu einem data binding Layout

  Hinweis: ⇨ siehe Folien 2.3 Seite 6

- Wechsle nun in die `MainActivity` und vervollständige das Databinding, in dem du die `binding` Variable anlegst und initialisierst

  Hinweis: ⇨ siehe Folien 2.3 Seite 7

- Lege deine App Farben fest
  - Überlege dir, welche Farben deine App prägen sollen und lege diese in der `colors.xml` Datei an

    Hinweis: Für die Wahl der Farben kannst du das Color Tool verwenden

  - Öffne die `themes.xml` Datei für das Light-Theme, sowie die `themes.xml (night)` Datei für das Dark-Theme und ersetze die Standard-Farben durch deine Eigenen.

    Achtung: ersetze in beiden Themes `parent="Theme.MaterialComponents.DayNight.DarkActionBar"` durch `parent="Theme.MaterialComponents.DayNight.NoActionBar"`

  - Um später auf deine Theme-Farben zuzugreifen, kannst du diese mit `?attr/` ansprechen. Beispiel: `android:textColor="?attr/colorPrimary"`
    Natürlich kannst du deine Farben im Nachhinein jederzeit ändern, um den Look deiner App anzupassen
- Wechsle wieder zur `activity_main.xml` Datei und designe das Layout für dein Telefonbuch. Denk daran, passende Constraints zu setzen. Orientiere dich gerne an obigem Screenshot.
  - Füge eine `Toolbar` oben in das Layout ein
  - Füge innerhalb mittig der `Toolbar` eine `TextView` ein und setze den Text auf "Telefonbuch"
  - Füge ganz unten einen `Button` ein. Er soll den Text "Hinzufügen" anzeigen. Vergib dem `Button` eine eindeutige ID
  - Füge ganz unten zudem zwei `TextInputLayout` ein, in denen jeweils ein `TextInputEditText` enthalten ist.
    - Ändere den `style` von beiden `TextInputLayout` zu `Widget.MaterialComponents.TextInputLayout.OutlinedBox`
    - In einem `TextInputEditText` soll als `hint` "Telefonnummer" angezeigt werden und im Anderen "Name"

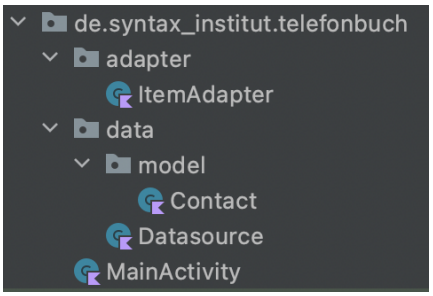
        - Vergib beiden `TextInputEditText` eine eindeutige ID

    Hinweis: ⇨ siehe Folien 2.2 Seite 10

    - Füge zwischen dem oberen `TextInputLayout` und der `Toolbar` eine `RecyclerView` ein. Gib der `RecyclerView` hier einen passenden `LayoutManager` und `scrollbars` auf `vertical`. Vergib der `RecyclerView` eine eindeutige ID

      Hinweis: ⇨ siehe Folien 2.6 Seite 11

- Designe das Layout eines Listeneintrags. Erstelle dafür eine neue "Layout Resource File" im Ordner `res/layout`, nenne sie `list_item` und ändere das Root element zu einer `MaterialCardView`. Hinweis: ⇨ siehe Folien 2.6 Seite 12
    - Füge innerhalb der `MaterialCardView` ein `ConstraintLayout` ein
    - Ziehe eine `ImageView` in das `ConstraintLayout` und füge mit einem Klick auf das `+` im "Pick a Resource" Fenster ein Bild hinzu, welches für alle Kontakte angezeigt werden soll.
      Nimm hierfür eine Vektorressource oder füge ein eigenes Bild hinzu
    - Füge zudem zwei `TextView` im `ConstraintLayout` ein und passe die Optik und Constraints an. Gib jeder `TextView` eine eindeutige ID
    - Passe die Optik und insbesondere die Größe und die Margins der `MaterialCardView` an. Du kannst dir den Listeneintrag probeweise in der `RecyclerView` anzeigen lassen, mit folgendem Attribut:
      `tools:listitem="@layout/list_item"`
- Nun müssen wir die `RecyclerView` in den Code einbinden und dort befüllen lassen.
    - Erstelle eine Package-Struktur, in der wir die einzelnen Klassen der `RecyclerView` organisieren. Erstelle zudem die Klassen `Contact`, `Datasource` und `ItemAdapter` innerhalb der Struktur. Am Ende sollte es so aussehen:

    - Zuerst definieren wir in `Contact` den Inhalt eines Listeneintrags (Telefonbucheintrag)
        - Ändere den Typ der Klasse zu einer `data class`

        - Ergänze den Konstruktor um zwei Attribute, ein `String` Attribut für den Namen und ein `String` Attribut für die Telefonnummer (Bonusfrage: Warum nehmen wir hier kein `Int` als Typ?)
    - Jetzt füllen wir `Datasource` mit unseren Informationen. Erstelle dafür innerhalb der Klasse eine Funktion `loadContacts()`, mit dem Rückgabetyp `MutableList<Contact>`.
        - Die Funktion liefert eine veränderliche Liste zurück, die aus `Contact` Objekten besteht (mindestens 10)
        - Jedes `Contact` Objekt wird mit zwei Parametern erstellt:
          ein String für den Namen und ein String für die Telefonnummer.
          Tipp: Hierfür bietet sich ein “phone number generator” an, der zufällige Telefonnummern generiert und der schnell mithilfe einer Googlesuche zu finden ist.

          Hinweis: ⇨ siehe Folien 2.6 Seite 10
    - Die Hauptarbeit der `RecyclerView` findet jetzt in der Klasse `ItemAdapter` statt.
      Hinweis: ⇨ siehe Folien 2.6 Seite 13
        - Erstelle eine innere Klasse `ItemViewHolder`, sie ist dafür verantwortlich, die items innerhalb der `RecyclerView` zu halten. Sie akzeptiert einen Parameter `itemView` vom Typ `View` und hat den Rückgabetyp `RecyclerView.ViewHolder(itemView)`. Innerhalb der Klasse werden zwei Klassenvariablen vom Typ `TextView` definiert, in denen die `TextView` für den Namen & die Telefonnummer aus der `itemView` gespeichert werden.
        - Passe die Klasse `ItemAdapter` nun an, indem du im Konstruktor eine Variable `dataset` vom Typ `List<Contact>` anlegst, in der später die Liste aus `Datasource` übergeben wird. Lass die Klasse zusätzlich von `RecyclerView.Adapter<ItemAdapter.ItemViewHolder>()` erben.
        - Die Klasse ist nun rot unterlegt, da die Implementation einiger Funktionen fehlen. Drücke `^+i` und wähle alle 3 Funktionen aus, um dies zu beheben und die Funktionen zu implementieren
        - Entferne die `TODO` Zeile in `getItemCount()`, in dieser Funktion soll einfach nur die Größe des `dataset` zurückgeliefert werden
        - Entferne die `TODO` Zeile in `onCreateViewHolder()`, in dieser Funktion soll das `list_item` Layout gebaut werden und in einem `ItemViewHolder` zurückgegeben werden
        - Entferne die `TODO` Zeile in `onBindViewHolder()`, in dieser Funktion sollen die im `ItemViewHolder` enthaltenen `TextView` die Informationen (Name & Telefonnummer) aus dem `Contact` Objekt zugewiesen bekommen

    - In der `MainActivity` werden alle Einzelkomponenten nun miteinander verbunden
        - Führe die Funktion `loadContacts()` auf ein Objekt der Klasse `Datasource` aus und speichere die Rückgabe in einer Variablen `contacts`
        - Erstelle ein neues Objekt der Klasse `ItemAdapter`, übergebe als Parameter `contacts` und speichere es in einer Variable `itemAdapter`
        - sprich die `RecyclerView` mithilfe der `binding` Variable an und weise dem `adapter` der `RecyclerView` den erstellten Adapter `itemAdapter` zu
- Als Nächstes programmieren wir die Funktion des Hinzufügens neuer Kontakte
    - Gib dem "Hinzufügen" Button in der `MainActivity` einen onClickListener
    - Folgendes soll bei einem Klick passieren:
        - Eine Variable `name` soll angelegt werden und mit dem Inhalt des entsprechenden `TextInputEditText` gefüllt werden
        - Eine Variable `number` soll angelegt werden und mit dem Inhalt des entsprechenden `TextInputEditText` gefüllt werden

          Hinweis: sprich den `TextInputEditText` mit `binding` und deiner vergebenen ID an und darin das `text` Attribut. Anschließend muss du den Text noch zu einem String konvertieren, bevor du ihn zuweist.
        - Falls weder der Inhalt von `name`, noch der Inhalt von `number` ein leerer String ist (also: **""**), soll ein neues `Contact` Objekt, mit `name` und `number` als Parameter, erstellt werden und zu der Liste `contacts` hinzugefügt werden. Anschließend muss nur noch der Adapter informiert werden, dass sich die Liste geändert hat.

          Hinweis: `itemAdapter.notifyItemInserted(contacts.size - 1)`
- Die Funktionalität des Telefonbuchs ist fertig! Verpasse der App noch ein schickes Icon

  Hinweis: ⇨ siehe Folien 2.4 Seite 6, im Resource Manager ein neues "Image Asset" hinzufügen

  Hinweis: evtl musst du in den Ordnern `res/mipmap/ic_launcher` und `res/mipmap/ic_launcher_round` die Dateien mit der **.webp** Endung löschen
- Passe deine App gerne an und erweitere sie mit weiteren Funktionen und Features!

Viel Erfolg!