Klausur zur Veranstaltung
Einführung in die Anwendungsprogrammierung mit Android
im Sommersemester 2019

Martin Kocur 6. August 2019

Nachname .....

Vorname .....

Matrikelnr. ..... **Studiengang, HS (Abk.)** .....

Allgemeine Hinweise

1. Die Bearbeitungszeit beträgt 60 Minuten. Sie können 64 Punkte erreichen.
2. Verwenden Sie für das Lösen der Aufgaben die Programmiersprache Java. Alle gegebenen Code-Fragmente sind ebenfalls in Java verfasst. Halten Sie sich an die bekannten Regeln zur Codequalität.
3. Schreiben Sie Ihren Namen und Ihre Matrikelnummer leserlich unten auf jedes Angabenblatt bevor Sie mit der Bearbeitung beginnen! Blätter ohne diese Angaben werden nicht gewertet. Wenn Sie die Rückseite eines Blattes verwenden, notieren Sie dies bitte auf der Vorderseite. Kennzeichnen Sie eindeutig, zu welcher Aufgabe eine Lösung gehört.
4. Benutzen Sie keine Bleistifte, keine rot schreibenden Stifte und kein TippEx (oder ähnliche Produkte).
5. Die Klausur ist als *Closed Book*-Klausur angelegt. Sie dürfen keine mitgebrachten Quellen oder Notizen zur Bearbeitung der Aufgaben verwenden. Technische Hilfsmittel sind ebenfalls nicht erlaubt.
6. Wenden Sie sich bei Unklarheiten in den Aufgabenstellungen immer an die Klausuraufsicht (Hand heben). Hinweise und Hilfestellungen werden dann, falls erforderlich, offiziell für den gesamten Hörsaal durchgegeben. Aussagen unter vier Augen sind ohne Gewähr.
7. Geben Sie keine mehrdeutigen (oder mehrere) Lösungen an. In solchen Fällen wird stets die Lösung mit der geringeren Punktzahl gewertet. Eine richtige und eine falsche Lösung zu einer Aufgabe ergeben also null Punkte. Das gilt auch für die *Multiple Choice*-Fragen. Kreuzen Sie bei einer Frage zwei richtige und zwei falschen Möglichkeiten an, ergibt das in der Summe null Punkte.

Teil 1: Theorie

Hinweis: Falsche Antworten führen zu **Punktabzug** innerhalb einer Aufgabe. Eine richtige und eine falsche Antwort bei einer Aufgabe geben in der Summe also null Punkte. Bitte kreuzen Sie nur die Antworten an, bei denen Sie sich sicher sind.
Bitte beachten Sie, dass eine korrekte Antwort **zwei Punkte** wert ist.

1) Mehrfachvererbung (2 Punkte)

Wie kann in Java eine Klasse die Eigenschaften und Funktionsweisen mehrerer anderen Klassen, die keinen Verwandschaftsbezug haben, annehmen?

- □ Die Klasse implementiert mittels *implements* ein Interface, um eine gemeinsame Funktionalität zwischen Klassen herzustellen.
- □ Die Klasse erbt mittels *extends* von mehreren Eltern.
- □ Die Klasse greift ohne weiteres auf Eigenschaften und Funktionsweisen von anderen Klassen zu.

2) Intents (2 Punkte)

Wie können Activities miteinander kommunizieren und Daten austauschen?

- □ Mit Intents und Extras in Form von Key-Value Paaren.
- □ Mit Adaptern und ViewGroups.
- □ Mit Permissions im Android Manifest.

3) Kommunikation mit anderen Applikationen (2 Punkte)

Welchen Aussagen im Bezug auf das Aufrufen von anderen Apps aus der eigenen App heraus sind korrekt?

- □ Andere Apps werden mit Hilfe von impliziten Intents aufgerufen.
- □ Andere Apps werden mit Hilfe von expliziten Intents aufgerufen.
- □ Man kann nur andere Apps aufrufen, die man selbst erstellt hat (Überprüfung mittels Zertifikat).

4) Welche Komponenten sind notwendig, um eine Darstellung von UI-Massendaten (z.B. Kontakte) zu realisieren?(2 Punkte)

- □ Datenbank, AsyncTask, DoInBackground-Methode.
- □ ArrayAdapter, Service, Timer.
- □ Adapter, Datenquelle, Layout für einzelne Elemente.

5) Permissions (2 Punkte)

Welche Aussagen sind richtig?

- □ Man kann ab Android 6.0 keine Permissions zur Laufzeit genehmigen.
- □ Permissions müssen im Manifest verankert sein.
- □ Es muss ein <intent-filter> im Manifest angegeben sein, um Permissions zu erteilen.

6) Fragments (4 Punkte)

Welche Aussagen sind richtig?

- □ Fragments leben außerhalb von Activities.
- □ Fragments leben innerhalb von Activites.
- □ Fragments und dazugehörige Activity haben denselben Lifecycle.
- □ Fragments müssen von der Klasse Fragment erben.
- □ Eine Activity kann nur ein Fragment enthalten.

7) Events (4 Punkte)

Wie kann zur Laufzeit auf Events reagiert werden?

- □ Mit (anonymen) inneren Klassen, die als ”Listener” eingesetzt werden.
- □ Gar nicht, jedes UI-Element muss in der XML-Definition auf die dazu gehörende Methode verweisen (z.B. mit dem ”onClick”-Attribut).
- □ Durch die Implementierung eines passenden Interfaces bei der Activity.

8) Datenbanken (4 Punkte)

Warum verwendet man Datenbanken?

- □ Um Daten über den Lebenszyklus einer Applikation hinweg zu speichern.
- □ Um Daten einer Applikation mehreren Usern zur Verfügung zu stellen und diese gleichzeitig damit arbeiten zu lassen.
- □ Um Daten flüchtig zu speichern.

9) Services (2 Punkte)

Welche Aussagen sind richtig?

- □ Services sind an Activities gebunden.
- □ Services sind eine eigene Klasse abgeleitet von Service.
- □ Services blockieren das UI.
- □ Sobald alle Activities ihrer Applikation geschlossen werden, wird der Service beendet.

10) Lifecycle (2 Punkte)

Versehen Sie die unten genannten Methoden mit Ziffern entsprechend ihrer Reihenfolge, in der Sie im Activity Lifecycle aufgerufen werden.
Eine richtig eingeordnete Methode wird mit einem halben Punkt bewertet. Wenn alle korrekt eingeordnet sind, gibt es zwei Punkte.

_____ onStart()

_____ onCreate()

_____ onResume()

_____ onPause()

11) AsnycTask (2 Punkte)

In welcher Reihenfolge werden die Callbacks beim Durchlaufen eines AsyncTask von Anfang bis Ende aufgerufen? Eine richtig eingeordnete Methode wird mit einem halben Punkt bewertet. Wenn alle korrekt eingeordnet sind, gibt es zwei Punkte.

_____ onPostExecute()

_____ doInBackground()

_____ onPreExecute()

_____ onProgressUpdate()

Teil 2: Code implementieren

Hinweis: Schreiben Sie immer zu jeder ihrer Lösungen die Methodensignatur ab, um eindeutig zu kennzeichnen, wie Sie welche Methode implementieren. Geben Sie auch die dazugehörige Klasse an.

12) ToDo - List (Intents und Adapter) (20 Punkte)

In dieser Aufgabe sollen Sie eine App vervollständigen, welche es dem Nutzer ermöglicht To Dos einzugeben, die zunächst in einer ArrayList gespeichert und dann anschließend in einer zweiten Activity in einer Liste angezeigt werden. Die Aufgabe umfasst also zwei Activities, welche bereits alle benötigten View-Elemente beinhalten. Gehen Sie davon aus, dass der gegebene Code korrekt ist und alle nötigen import – Anweisungen vorhanden sind.
Die beigefügten Layout XML-Dateien zeigen die Elemente der jeweiligen Activity.

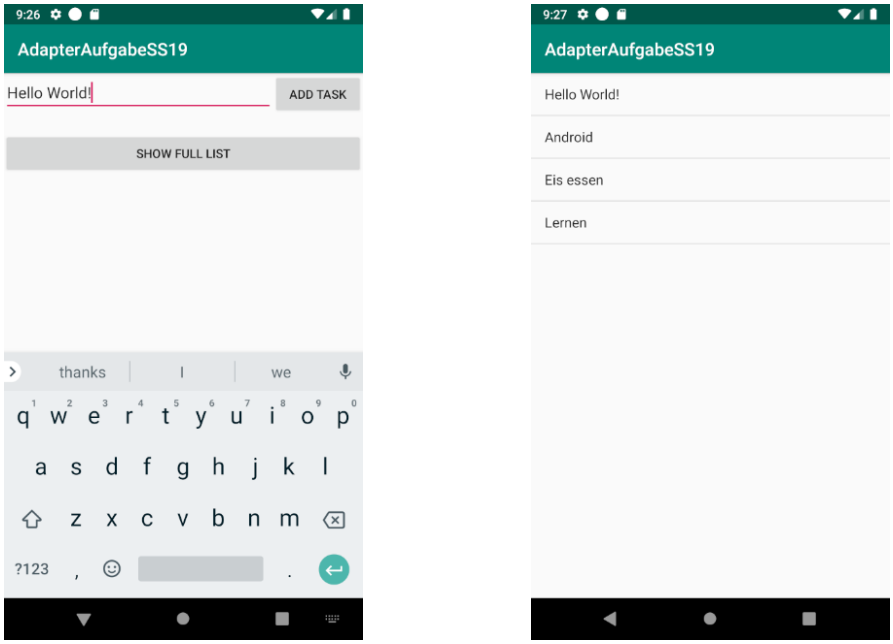

Abbildung 1: Links: Ansicht der MainActivity, Rechts: Ansicht der ResultActivity

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
   /android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <EditText
        android:id="@+id/text_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter new task..." />

    <Button
        android:id="@+id/add_Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add Task" />

    <Button
        android:id="@+id/result_Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show full list" />

</LinearLayout>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
   /android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ResultActivity">
    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/list_view">
    </ListView>
</LinearLayout>
```

```java
public class MainActivity extends AppCompatActivity {


    private Button resultButton, addButton;
    private EditText textInput;

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

    private void initViews() {

    }

    private void setButtonListeners() {

    }
}
```

```java
public class ResultActivity extends AppCompatActivity {

    private ListView listView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_result);
        getExtras();
        initListView();
        initAdapter();
    }

    private void getExtras() {

    }

    private void initListView(){

    }

    private void initAdapter(){

```

```
24      }
25  }
```

12a) Implementierung der Klasse MainActivity

Vervollständigen Sie als Erstes die Klasse `MainActivity`. Beginnen Sie damit, in der `initViews`-Methode die Member-Variablen mit den korrekten UI-Elementen zu belegen. Entnehmen Sie die ID's der View Elemente aus der entsprechenden Layout XML.

Anschließend müssen Sie für die Buttons `OnClickListener` setzen. Über den `addButton` kann der User ein neues ToDo Item erzeugen. Mit Hilfe des `resultButton` kann die `ResultActivity` aufgerufen werden. Für die jeweiligen Listener können Sie z.B. eine Instanz einer anonymen Klasse verwenden, die Sie mit `new View.OnClickListener() {...}` erzeugen. Diese Klasse muss dann eine Methode `public void onClick(View v)` implementieren.

Erzeugen Sie im nächsten Schritt eine ArrayList die Elemente vom Typ String enthalten kann, um die einzelnen ToDo Items zu speichern. Dafür soll beim Klick auf den `addButton` der eingegebene Text aus dem Inputfeld der ArrayList übergeben werden. Anschließend kann das Inputfeld geleert werden.

Zuletzt gilt es noch die `ResultActivity` über `resultButton` aufzurufen. An diese soll über einen `Intent` die ArrayList mit den jeweiligen ToDos gesendet werden. Instantiieren Sie einen `Intent` mit den korrekten Parametern (Anwendungs-`Context` und Klassentyp der Ziel-Activity).

Lösung zu a)

Fortsetzung: Lösung zu a)

12b) Implementierung der Klasse ResultActivity

Nun muss noch die `ResultActivity` implementiert werden. Auch hier gilt als Erstes: belegen Sie die Member-Variable `listView` in der Methode `initListView` mit dem richtigen UI-Element. Erzeugen Sie anschließend in der `getExtras` Methode eine Variable vom Typ `Bundle` mit dem Informationspaket aus dem Intent. Um die Liste mit den Informationen zu füllen, müssen diese aus dem `Bundle` geholt werden. Initialisieren Sie zusätzlich dazu in der Methode `initAdapter` einen Adapter vom Typ `ArrayAdpater<String>` und vervollständigen Sie die Adapterfunktionalität, um der `listView` die Daten aus der ArrayList zur Verfügung zu stellen. Das Layout der einzelnen Listeneinträge ist das Standardlayout `android.R.layout.simple_list_item_1`. Über die Methode `notifyDataSetChanged` wird der Adapter über Veränderungen der Daten informiert.

Lösung zu b)

Fortsetzung: Lösung zu b)

13) Kochrezepte aus dem Web (Interfaces) (16 Punkte)

Gegenstand dieser Aufgabe ist eine Applikation, die es dem User ermöglicht Tags einzugeben, welche dann gebündelt in einer ArrayList gespeichert und in einer TextView dargestellt werden. Anschließend werden alle Rezepte, die einen dieser Tags beinhalten, von einer "Kochwebseite"geladen und in derselben Activity (`MainActivity`) in einer Liste dargestellt. Die Aufgabe umfasst also nur eine Activity, welche alle benötigten View-Elemente beinhaltet. Eine zweite Klasse namens `BackgroundTask` erbt von `AsyncTask` und ist für das Laden der Daten verantwortlich. Sie brauchen sich nicht um die Funktionalität der Serveranfrage des AsyncTasks kümmern. Für diese Aufgabe brauchen sie ein eigens erstelltes Interface, das als Callback für das Herunterladen fungiert und die Progressbar bei jedem Update aktualisiert.
In der gegebenen Klasse `MainActivity` wird bereits eine `ProgressBar` realisiert. Bei Klick auf den `addFoodTagButton` werden die vom Nutzer eingegebenen Tags als String in einer ArrayList gespeichert und in der `TextView foodTags` nacheinander kommasepariert angezeigt. Bei Klick auf den `getRecipesButton` sollen nun mit Hilfe der `BackgroundTask` Klasse alle Rezepte aus dem Web geladen werden.
Hinweis: Gehen Sie davon aus, dass das UI und die Click-Handler bereits implementiert sind, der gegebene Code korrekt ist und alle notwendigen `import`-Anweisungen bereits vorhanden sind.

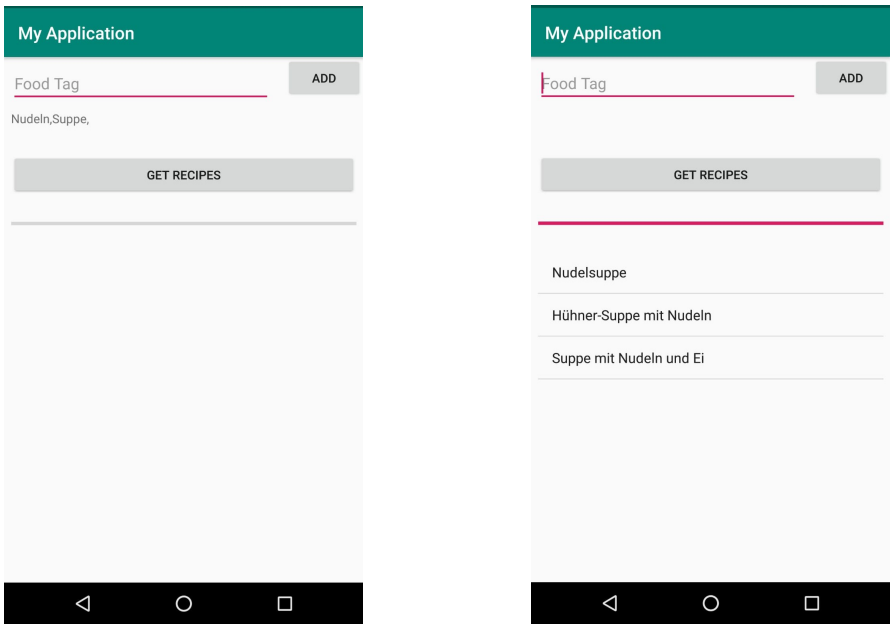

Abbildung 2: Ansicht der MainActivity

```java
public class MainActivity extends AppCompatActivity
{

    private EditText foodTagInput;
    private Button getRecipesButton,addFoodTagButton;
    private ProgressBar progressBar;
    private ArrayList<String> foodTagsList;
    private textView foodTags;

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

    private void init(){
         /*(...) Neben der Initialisierung des UIs und aller
            relevanten Elemente, wird hier die Logik für den
            addFoodTagButton und die Darstellung der Rezepte in
             einer Liste implementiert*/
        getRecipes();
    }

    private void getRecipes(){
        getRecipesButton.setOnClickListener(new View.
           OnClickListener() {
            @Override
            public void onClick(View v) {
                progressBar.setMax(foodTagsList.size());
        // übergibt alle Werte von foodTagsList an die
           doInBackground Methode der BackgroundTask Klasse
                new BackgroundTask().execute(foodTagsList.
                   toArray(new String[0]));
                foodTags.setText("");
            }
        });
    }
}
```

```java
public class BackgroundTask extends AsyncTask<String,Integer,
   ArrayList<String>> {

    public BackgroundTask(){

    }
```

```
6
7       @Override
8       protected ArrayList<String> doInBackground(String...
            strings) {
9           ArrayList<String> result = new ArrayList<>();
10          for (int i = 0; i < strings.length;i++ ){
11              //(...) Logik für Serveranfrage und Verarbeiten der
                    Response.
12
13
14          }
15          return result;
16      }
17
18      @Override
19      protected void onProgressUpdate(Integer... values) {
20
21      }
22
23      @Override
24      protected void onPostExecute(ArrayList<String> arrayList) {
25
26      }
27  }
```

13a) Implementierung eines Interfaces

Erstellen Sie ein Interface über das die Kommunikation zwischen MainActivity und BackgroundTask in Teilaufgabe 13b) realisiert wird. Dieses Interface soll Methoden bereitstellen, die die MainActivity darüber benachrichtigen, wenn

- die ProgressBar aktualisiert werden soll,
- der Download der Rezeptdaten abgeschlossen ist.

Ordnen Sie den Methoden die korrekten Parameter- und Rückgabetypen zu.

Lösung zu a)

13b) Implementierung der Kommunikation zwischen MainActivity und BackgroundTask

Stellen Sie nun die Verbindung zwischen MainActivity und BackgroundTask her, um beide Klassen über Ihr Interface miteinander kommunizieren zu lassen. Übergeben Sie dafür das Interface an den BackgroundTask bei der Objekterzeugung. Denken Sie daran den Listener an geeigneter Stelle in `BackgroundTask` zu registrieren, um die MainActivity über den aktuellen Stand und die Ergebnisse des Downloads der Rezepte zu informieren. Sie müssen sich nicht um das tatsächliche Updaten der ProgressBar sowie das Darstellen der Ergebnisse in der `MainActivity` kümmern. Dafür ist es ausreichend, wenn sie die Methodensignatur (inkl. Übergabeparameter) der einzelnen Methoden des Interfaces in der `MainActivity` angeben.

Hinweis: Schreiben sie in ihrer Lösung die Klassen- und Methodensignatur ab. Geben sie im Zweifel auch Zeilennummern an, um eindeutig zu kennzeichnen an welcher Stelle im Code sie Veränderungen durchführen.

Lösung zu b)