4 Android: Grafische Benutzeroberflächen

Die Programmierung grafischer Benutzeroberflächen ist ein wichtiges und weites Gebiet. „Wichtig“ deshalb, weil die Oberfläche einer Applikation entscheidend dafür ist, wie komfortabel man mit ihr arbeiten kann und ob man sie überhaupt benutzt. „Weit“ deswegen, weil im Laufe der Android-Versionsentwicklung immer neue Techniken hinzugekommen sind, die mittlerweile nicht mehr leicht zu überblicken sind.

Dieses Kapitel steht daher besonders vor dem Dilemma, einerseits nichts Wichtiges wegzulassen, aber anderseits nicht zu stark auszuufern. Es muss sich auf eine grundlegende Einführung in die Android-GUI-Programmierung (GUI = Graphical User Interface) beschränken und kann keinen Anspruch auf Vollständigkeit erheben:

- Das Kapitel zeigt, welche *allgemeinen Bestandteile* eines Android-Studio-Projekts bei der GUI-Programmierung eine Rolle spielen und wie sie interagieren. Wie schon in Kapitel 3 angesprochen, werden grafische Oberflächen durch XML-Ressourcen spezifiziert, auf die der Java-Code der Activities zugreifen kann.
- Die *Java-Klasse* `View` wird eingeführt. `View` ist die Oberklasse für viele konkrete *GUI-Elemente*, wie Textanzeigen, Buttons usw. Sie definiert allgemeine Eigenschaften von Objekten, die an der Oberfläche angezeigt werden können – so z.B. eine Methode zum Zeichnen des Views auf einem Display sowie Methoden und Listener zur Reaktion auf Eingaben in den View. Neben einzelnen GUI-Elementen werden auch *Layouts* durch `View`-Unterklassen spezifiziert.
- Ein Großteil des Kapitels befasst sich mit einer Reihe von *Teilgebieten der GUI-Programmierung*, wie z.B. die Bereitstellung von Auswahlangeboten, die Realisierung bewegter Animationen oder der Umgang mit Gesten. Details dazu folgen in den nächsten Abschnitten.

Gestalterische Fragen werden nicht angesprochen; es wird also nicht diskutiert, wie man eine Oberfläche aus Designer-Sicht gut gestaltet. Hierzu findet man auf der Android-Website einen sehr ausführlichen *Styleguide* mit entsprechenden Empfehlungen (*https://developer.android.com/design/*).

4.1 Realisierung von Standardkonzepten

Zur Gestaltung grafischer Oberflächen gibt es grundlegende Standardkonzepte, die auch von Android unterstützt werden:

- Zur Anzeige und Eingabe von *Texten* sowie zur Anzeige von *Bildern* existieren entsprechende Klassen. Darüber hinaus können *frei gestaltete Grafiken* angezeigt werden, und es können *Multimedia-Clips* (Audio, Video) wiedergegeben werden.
- Durch das Drücken von *Buttons* können Aktionen ausgelöst werden, wobei ein Button mit einem Text beschriftet oder mit einem Bild versehen sein kann.
- Über *Tooglebuttons* und *Switches* können Ein/Aus-Einstellungen vorgenommen werden.
- Auswahlen können über *Radiobuttons*, *Checkboxen*, *Seekbars*, *Menus*, *Dropdown-Listen* (so genannte *Spinner*), *ListViews*, *StackViews*, *GridViews* (= zweidimensionale Anzeigen) oder *Bildergalerien* getroffen werden.
- *Dialoge* und *PopupWindows* erscheinen in eigenen temporären Fenstern und können dabei Eingaben des Benutzers entgegennehmen. *Toasts* zeigen nur Informationen an, bieten aber keine Eingabemöglichkeit.
- Benachrichtigungen, die länger sichtbar bleiben sollen, werden oben im Display in der *Status Bar* angezeigt.

4.2 Berührungen und Gesten

Eine Besonderheit bei Mobilgeräten ist der berührungsempfindliche Bildschirm: Mobilgeräte-Applikation werden dadurch gesteuert, dass der Benutzer das Display antippt oder komplexere Gesten darauf ausführt. Die Applikation muss diese *Berührungen* und *Gesten* erkennen und differenziert darauf reagieren können.

In Android basiert der Umgang mit Berührungen auf Callback-Methoden, die einzelnen Views (also GUI-Elementen) zugeordnet sind und die der Programmierer ausprogrammiert: Berührt der Benutzer einen View, so ruft das Laufzeitsystem eine entsprechende Methode auf. Für verschiedene Arten von Views gibt es spezifische Methoden, wie z.B. `onClick()` für Buttons. Darüber hinaus definiert die Oberklasse `View` die Methode `onTouchEvent()`, die somit in allen `View`-Unterklassen zur Verfügung steht.

`onTouchEvent()` erhält einen Parameter, der das Berührungsereignis detailliert beschreibt (Ort auf dem Display, Zeitpunkt, Druckstärke, …). Der View kann diese Informationen auswerten und geeignet reagieren. Android kann dabei zwischen mehreren Fingern unterscheiden, die das Display berühren, sodass man eine Applikation nicht nur mit einem Finger steuern kann.

Eine *Geste* ist eine Folge von Berührungsereignissen, die ein bestimmtes Muster aufweist. Beispielsweise setzt sich eine Wischbewegung aus Berührungen zusammen, die rasch in einer bestimmten Richtung erfolgen. *Gesture-Detectors* untersuchen Berührungsfolgen darauf, ob sie einem solchen Muster entsprechen und lösen dann, durch Aufruf von Callback-Methoden in Listenern, Aktionen aus, die der Programmierer definiert hat. Android stellt solche Detektoren bereit, sodass man sie nicht selbst programmieren muss.

4.3 Frei gestaltete Grafiken und Animationen

Jedes `View`-Objekt besitzt eine Methode `onDraw()`, die festlegt, wie der View auf dem Display dargestellt wird. Dazu werden im Körper von `onDraw()` Methoden wie `drawLine()` oder `drawText()` aufgerufen, die Linien, Texte usw. auf das Display zeichnen. `onDraw()` ist eine Callback-Methode: Das Laufzeitsystem ruft sie auf, wenn es den View auf das Display bringen möchte – beispielsweise, wenn die zugehörige Activity gestartet wird.

Für die vorgegebenen Klassen für Textanzeigen, Buttons usw. sind die `onDraw()`-Methoden bereits definiert. Ein Programmierer kann aber weitere Unterklassen von View schreiben und in deren `onDraw()`-Methoden frei festlegen, wie diese Views gezeichnet werden sollen. Auf diese Weise können *selbstgestaltete Grafiken* auf dem Display angezeigt werden.

Im Normalfall kontrolliert das Laufzeitsystem, wann ein View auf dem Display gezeichnet und wann seine Darstellung aktualisiert wird, denn das Laufzeitsystem ruft `onDraw()` auf. Bei der `View`-Unterklasse ***`SurfaceView`*** ist das anders: Hier kann ein Thread in der Applikation steuern, wann die Anzeige (neu) gezeichnet wird, und hat damit die Kontrolle über die Darstellung. Auf diese Weise können *Animationen* realisiert werden, bei denen der Thread die Eigenschaften der angezeigten Objekte (Position, Farbe, …) fortlaufend ändert.

Alternativ können Animationen durch XML-Beschreibungen dynamischer Abläufe spezifiziert werden.

4.4 Fragments

Eine Android-Applikation soll auf einer Vielzahl von Geräten laufen können, von Smartphones über Tablets bis hin zu Android-fähigen TV-Geräten. Dabei kann ihre Anzeige nicht auf allen Geräten gleich aussehen, sondern sie muss sich an die unterschiedlichen Displaygrößen anpassen.

Zur Lösung dieses Problems wurden in Android 3.0 die so genannten *Fragments* eingeführt. Ein Fragment hat sein eigenes Layout, fasst also mehrere GUI-Elemente zu einer Einheit zusammen. Je nach Größe des Displays wird jeweils nur ein Fragment oder es werden mehrere Fragments gleichzeitig angezeigt. Der Benutzer muss also entweder zwischen Fragments umschalten, oder er bekommt mehrere mit einem Blick zu sehen. Die dahinterliegende Funktionalität der Applikation ist aber jeweils dieselbe.

Mit Fragments wird also eine zusätzliche Schicht zwischen der Activity mit ihrem Layout und den darin enthaltenen Einzelelementen eingeführt. Dies erhöht die Flexibilität, verkompliziert aber die Programmierung. Die Open Handset Alliance möchte Programmierer dazu bringen, nur noch mit Fragments zu arbeiten und deklariert einige Klassen aus der „Vor-Fragment-Zeit“ als „deprecated“, also als nicht mehr unterstützt. Allerdings funktionieren diese Klassen nach wie vor, und da die Programmierung mit ihnen einfacher ist als mit den neueren fragmentbasierten Klassen, werden sie auch in diesem Modul eingesetzt.

4.5 Navigation

Eine Applikation besteht typischerweise aus mehreren Activities mit jeweils eigenen Oberflächen, zwischen denen man durch Drücken von Oberflächenelementen navigieren kann. Seit kurzem bietet Android Unterstützung bei der systematischen Programmierung solcher Applikationen.

4.6 Applikationsbeispiele

Die folgenden Applikationsbeispiele illustrieren den Lernstoff:

- *LayoutAndroid*, *ActivitiesUeberlagertAndroid* und *StylesThemesAndroid*: Die Applikationen zeigen verschiedene Möglichkeiten, das Layout der grafischen Oberfläche einer Applikation festzulegen.

- *SelectionsAndroid*: Die Applikation demonstriert den Umgang mit einfachen Buttons, Radiobuttons, Checkboxen, Togglebuttons/Switches, Seekbars, Bildergalerien, Spinnern (= Dropdown-Listen), GridViews (= zweidimensionalen Anzeigen) und ListViews/ListActivities sowie StackViews (= Auflistungen von Auswahlmöglichkeiten).
- *MenusAndroid*: Die Applikation stellt drei Arten von Menus vor. Optionsmenus erscheinen bei Drücken des Menubuttons oder werden permanent in der Actionbar oben im Display angezeigt. Kontextmenus erscheinen, wenn der Benutzer Views auf der aktuell angezeigten GUI drückt. Popupmenus werden durch Anweisungen im Programmcode angezeigt.
- *NotifDialogsAndroid*: Die Applikation zeigt Möglichkeiten, außerhalb der eigentlichen GUI der Activity mit dem Benutzer zu interagieren. Es gibt hier Toasts zur Ausgabe von Informationen in einem eigenen Fenster, Dialoge und PopupWindows mit Möglichkeiten für Benutzereingaben sowie Notifications in der Statusbar oben im Display.
- *TouchGesturesAndroid*: Die Applikation illustriert den Umgang mit einfachen Berührungsereignissen und mit Gesten – sowohl mit einem als auch mit mehreren Fingern.
- *GrafAnimMMAndroid*: Die Applikation fasst mehrere Teilapplikationen zusammen, die grafische und multimediale Daten ausgeben. Sie zeigt, wie man `View`-Unterklassen mit einer selbst definierten Grafikausgabe schreibt, wie man animierte, also bewegte Grafikausgaben definiert und steuert und wie man Audio- und Videoclips ausgibt.
- *SelfDefinedView*: Die Applikation zeigt, wie der Programmierer selbst neue Oberflächenelemente definieren kann.
- *FragmentsAndroid*: Die Applikation demonstriert den Einsatz von Fragments. Man sieht insbesondere, wie das Laufzeitsystem die Anzeige automatisch an die Displaygröße des ausführenden Geräts anpasst.
- *AppNavigationAndroid*: Die Applikation illustriert, wie man mit Hilfe des Navigation Editors von Android Studio festlegt, auf welchen Wegen sich der Benutzer durch die Oberflächen einer Applikation bewegen kann.

4.7 Was ist zu tun?

Arbeiten Sie nun die Folien zu Kapitel 4 durch. Folgen Sie wieder den Arbeitsanweisungen, insbesondere bezüglich der Beispielprogramme. Dabei werden Sie

- nochmals sehen, wie in einem Android-Projekt grafische Oberflächen definiert werden und wie Activities darauf zugreifen,
- verschiedenartige GUI-Konzepte und -Techniken kennenlernen und
- Hinweise für die Vertiefung der Thematik, insbesondere durch Internet-Quellen, bekommen.

Ergänzende Exkurse führen zu Lehrvideos mit zugehörigen Android-Projekten (*http://www.nt.th-koeln.de/vogt/vma/videos.html*), in denen weitere GUI-Techniken vorgestellt werden – beispielsweise zusätzliche Techniken zur Animation von Bildschirmausgaben. Sie können diesen Exkursen unmittelbar oder auch zu einem späteren Zeitpunkt folgen. Wenn Sie Zeit und Lust haben, so können Sie sich zudem näher mit den Empfehlungen zur GUI-Gestaltung beschäftigen (*https://developer.android.com/design/*).

4.8 Übungsaufgaben

1. Grundlegende GUI-Techniken:
   - In welcher Art von Datei wird der Name (die „Id“) eines Buttons festgelegt?
   - An welche Methode kann man den Namen eines Buttons übergeben und erhält dann eine (Java-)Referenz auf diesen Button?
   - Mit welcher Methode registriert man einen Listener für einen Button? Welches Interface und welche Methode dieses Interfaces muss die Listener-Klasse implementieren?
   - Mit welcher Methode zeigt man auf dem Display das an, was in einer Layout-Datei definiert wurde? In welcher Klasse ist diese Methode wohl definiert?
   - Schreiben Sie ein Programm mit folgenden GUI-Komponenten: Ein Eingabefeld, ein Ausgabefeld, ein Button „Quadrat“ (berechnet das Quadrat der Zahl aus dem Eingabefeld und schreibt das Ergebnis in das Ausgabefeld) und ein zweiter Button „Wurzel“ (berechnet entsprechend die Wurzel).
2. Menus / Action Bars
   - Menus/Action Bars werden definiert im Unterverzeichnis _____.
   - Ein _____menu (bzw. eine Action Bar) bezieht sich auf die gesamte Activity, ein _____menu nur auf einen View in seiner Oberfläche.

- In welchem Zusammenhang stehen die Methoden `onCreateOptionsMenu()` und `onOptionsItemSelected()`?
- In welchem Zusammenhang stehen die Methoden `registerForContextMenu()`, `onCreateContextMenu()` und `onContextItemSelected()`?
- Schreiben Sie eine Activity mit einer Action Bar (oder alternativ einem Optionsmenu). Es sollen drei Auswahlmöglichkeiten „eins“, „zwei“ und „drei“ angeboten werden. Wird eine dieser Möglichkeiten ausgewählt, so soll ein Toast erscheinen, der den entsprechenden Text anzeigt.

3. AdapterViews
   - Ein AdapterView dient zur

     O Anpassung des Layouts an die Fähigkeiten eines Geräts.

     O Darstellung von Daten aus einer Datenquelle.

     O Übergabe von Daten in einen Listener.
   - Ein Menu, das nach unten aufklappt (im allgemeinen Sprachgebrauch also ein „Dropdown Menu“), kann man in Android programmieren durch einen / eine

     O Gallery O ListView O Spinner O Checkbox O Toast
   - Wie wird ein „Dropdown Menu“ (siehe vorheriger Punkt) mit Auswahlmöglichkeiten gefüllt? Wie wird die Reaktion festgelegt, die erfolgen soll, wenn der Benutzer eine dieser Möglichkeiten auswählt?
4. Toasts, Notifications und Dialoge
   - Was ist der Unterschied zwischen einem Toast, einer Notification und einem Dialog?
   - Ein Benutzer kann einen Text eingeben über einen

     O EditText O TextView O Toast

     O PopupWindow O DatePickerDialog
   - In welchem Zusammenhang stehen die Methoden `showDialog()` und `onCreateDialog()`? Wie werden dabei verschiedene Dialoge unterschieden?
   - Schreiben Sie eine Klasse für ein Popup-Fenster, das ein Textfeld und einen Button enthält. Das Textfeld soll zunächst den Wert 5 anzeigen. Mit jedem Click auf den Button soll der Wert im Textfeld um 1 herun-

tergezählt werden. Das Fenster soll geschlossen werden, wenn die Anzeige den Wert 0 erreicht.

5. Berührungsereignisse und Gesten
   - Bei einem Berührungsereignis wird die Methode _____ des betreffenden _____ ausgeführt. Der Parameter dieser Methode ist von der Klasse _____. Komplexere Gesten werden erkannt in der Methode _____ eines _____; die Reaktion darauf erfolgt in einer Methode eines zugehörigen _____.
   - Welche Daten stehen in einem `MotionEvent`-Objekt? Mindestens drei Nennungen!
   - Das Action-Attribut eines `MotionEvent`-Objekts kann folgende Werte annehmen:

     O `ACTION_DOWN` O `ACTION_SCALE` O `ACTION_FLING`

     O `ACTION_MOVE` O `ACTION_UP`
   - In welchem Zusammenhang stehen erstens Gesten und zweitens Objekte der Klasse `MotionEvent`?
   - Was sind die Schritte, mit denen eine komplexe Geste erkannt und auf sie reagiert wird?
   - Vervollständigen Sie das folgende Codestück:

```
public boolean on_______Event(__________ ev) {
  myGestureDetector.___________(___);
  final int action = ev.get_________();
  switch (action & MotionEvent.ACTION_MASK) {
    case MotionEvent._________:
```

6. Animationen
   - Was in einen View gezeichnet wird, wird durch seine Methode _____ (*) definiert. Der Parameter dieser Methode ist von der Klasse _____ (**). Die Methode wird bei den meisten Views vom _____ aufgerufen.
   - Wer ruft aber bei einem `SurfaceView`-Objekt die Methode (*) auf?
   - Nennen Sie zwei Methoden der Klasse (**)!
   - Wozu dient ein „Surface Holder“?
   - Muss man für eine Animation unbedingt einen Thread programmieren?