

Android

Übungsbeispiele



Dr. Heinz Schiffermüller
HTBLA-Kaindorf
Abteilung EDVO

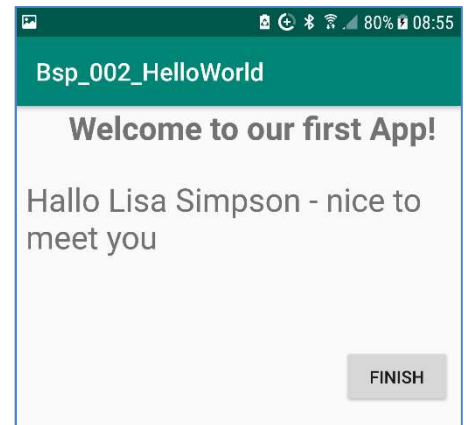
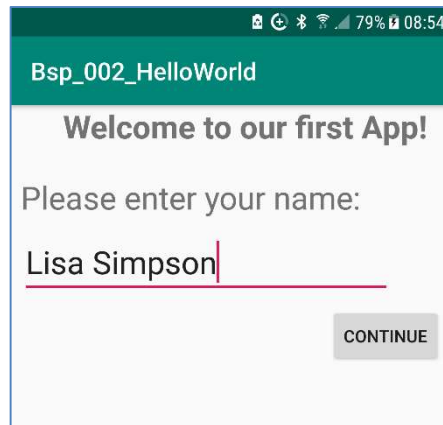
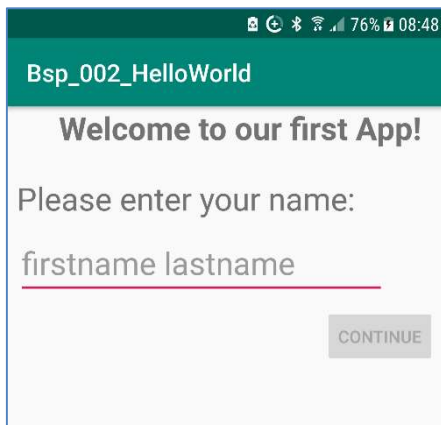
Erstellung: August 2019
Letzte Überarbeitung: August 2019

Bsp_101_HelloAndroid

Erstelle eine einfache Standard App in Android Studio mit dem Begrüßungstext "Hello Android". Starte die App auf deinem Smartphone und im Emulator.

Bsp_102_HelloWorld

HelloWorld-App bestehend aus einem Begrüßungstext, einem Eingabe-Textfeld und einem Button. Durch Klicken auf den Button wird das Textfeld ausgelesen, der Begrüßungstext wird geändert, das Eingabetextfeld verschwindet und die Beschriftung des Buttons wird von Continue auf Finish geändert:




Bsp_103_CurrencyConverter

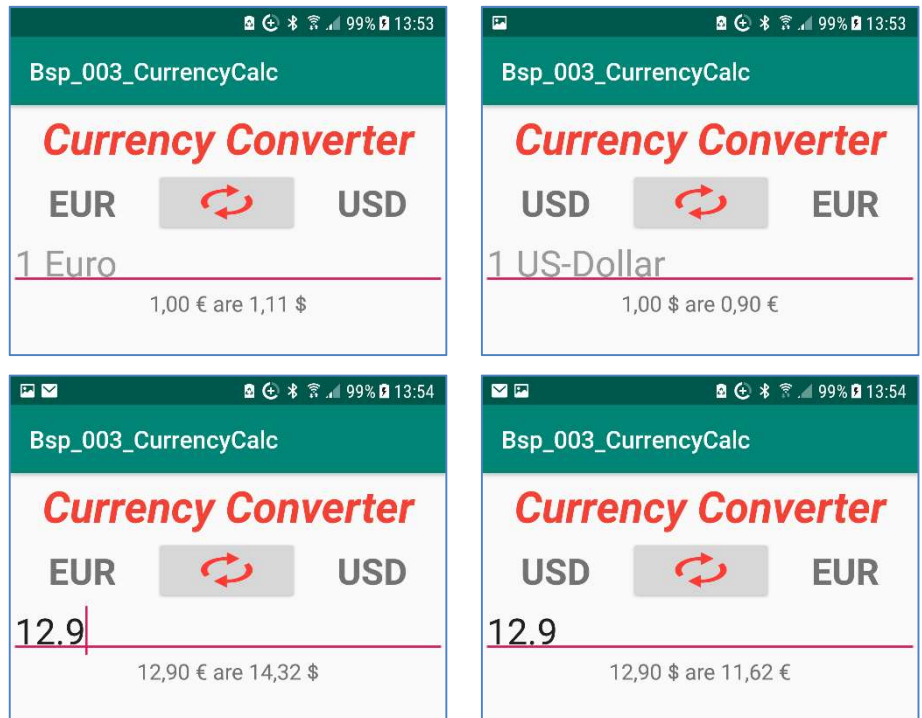


Erstelle eine App zur Umwandlung von Euro-Beträgen in Dollar und umgekehrt. Verwende dazu folgendes Layout:

Programmablauf:

Der Währungs-Wechsel-Button  ändert die Umrechnung von Euro in Dollar und umgekehrt. Es werden die Beschriftungen der TextViews (EUR und USD) ausgetauscht, und die aktuell angezeigte Umrechnung wird neu berechnet.

Im EditText können Beträge ganzzahlig oder als Kommazahlen eingegeben werden. Nach jeder Eingabeänderung wird die Umrechnung in der unteren TextView sofort aktualisiert. Wird der Inhalt im EditText vollständig gelöscht, so wird als Hint entweder 1 Euro oder 1 US-Dollar angezeigt.





Programmbeschreibung:

Datei `activity_main.xml`:


Verwende ein geschachteltes Linear Layout um die GUI entsprechend der Abbildung zu erstellen.

Klasse `MainActivity.java` sind folgende Methoden zu implementieren:



`onChangeCurrency()`

-  `onClick`-Event für den ImageButton: Jeder Button-Klick bewirkt das Austauschen der Währungen. Zu ändern sind die Beschriftungen aller betroffenen TextView-Elemente.
-  Registriere das Event über die XML-Datei.

`afterTextChanged()`

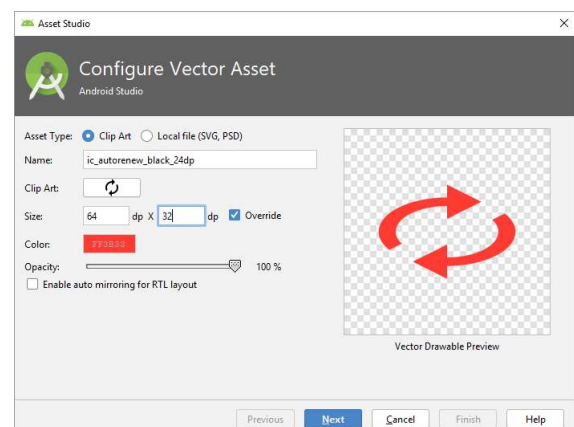
-  Implementiere das `TextWatcher`-Interface als innere, anonyme Klasse und registriere sie als `TextChangedListener` für das EditText-Element.

`convertAndUpdate()`

-  Implementiere in dieser Methode die Programm-Logik.
-  Verwende die Methode in den beiden Events.

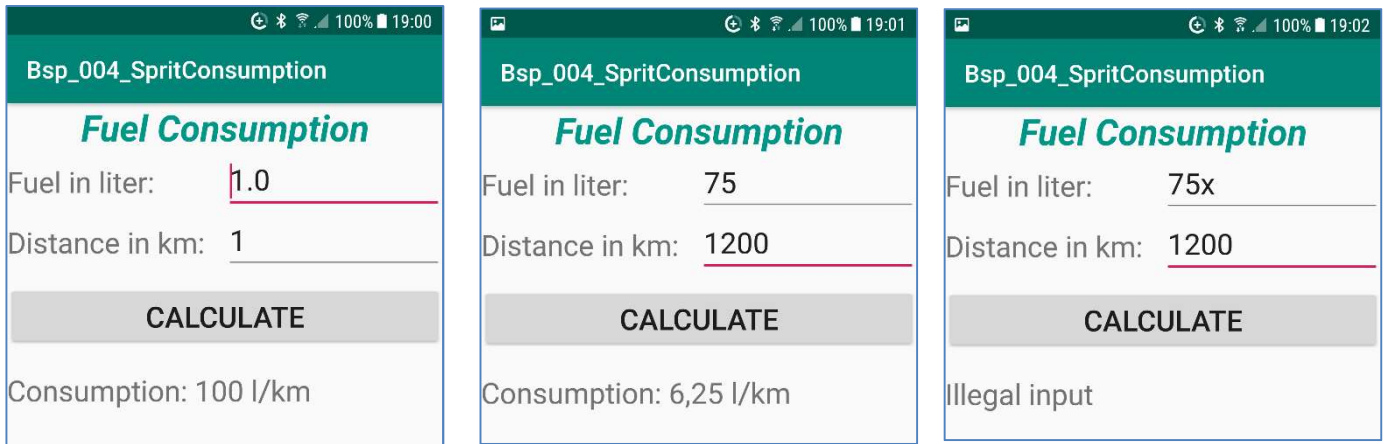
Hinweis:

Um das Image für den Button einzufügen: im Project-View von Android Studio das `res/drawable` Verzeichnis öffnen und anklicken. Über das Kontext-Menü **New** → **Vector Asset** das Asset Studio öffnen. Auf das Icon neben *Clipart* klicken und als Suchstring *autorenew* eingeben. Das Clipart auswählen und konfigurieren. Verwende anstelle eines *Button* einen *ImageButton*.



Bsp_104_FuelConsumption

Erstelle eine App zur Berechnung des durchschnittlichen Benzinverbrauchs auf Basis der verbrauchten Menge an Benzin in Liter und der gefahrenen Entfernung in Kilometer. Verwende dazu folgendes Layout:

**Programmablauf:**

Die Anzeige bei Programmstart ist der ersten Abbildung zu entnehmen.

In den EditText Eingabefeldern werden der Wert für den Benzinverbrauch als Kommazahl und der Wert für die gefahrenen Kilometer als ganze Zahl eingegeben. Nach Klicken auf den Calculate-Button wird der durchschnittliche Verbrauch pro 100 km berechnet und in der unteren TextView ausgegeben.

Programmbeschreibung:

Datei `activity_main.xml`:

Verwende ein RelativeLayout um die GUI entsprechend der Abbildung zu erstellen.

Klasse `MainActivity.java` sind folgende Methoden zu implementieren:

`onCalculateConsumption()`

- onClick-Event für den Button: implementiere das `OnClickListener`-Interface als innere, anonyme Klasse und registriere sie für den Button.

`onCalcConsumption()`

- Methode in der die Programm-Logik implementiert wird.
- Verwende Exception-Handling bei fehlerhafter Eingabe.

Erweiterung:

Verwende eine grüne Schrift für die Ausgabe wenn der Verbrauch unter 10 Liter liegt, sonst eine rote Schrift.

Bsp_105_PocketCalculator

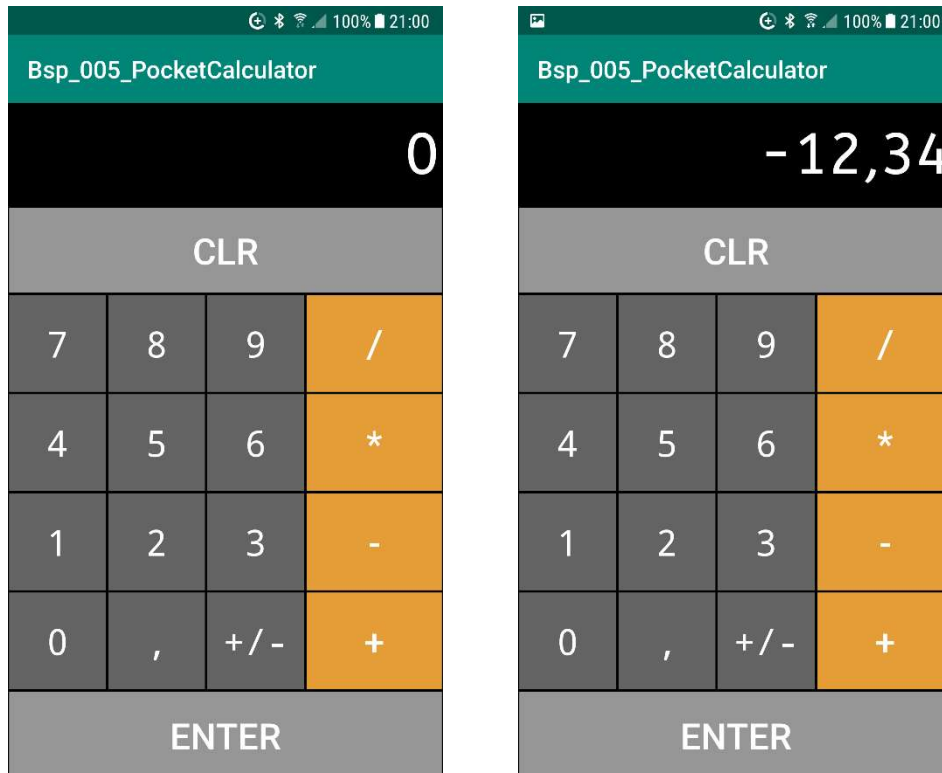
Erstelle eine App für einen einfachen Taschenrechner mit Postfix Notation. Bei der Postfix-Notation werden zuerst die beiden Werte, getrennt durch die ENTER-Taste, eingegeben und anschließend erst der Operator:

Tastenfolge Infix-Notation: $1 + 1 =$

Tastenfolge Postfix-Notation: $1 \text{ ENTER } 1 +$

Der Vorteil der Postfix Notation ist, dass keine Klammerungen notwendig sind.

Verwende für den Taschenrechner folgendes Layout:

**Programmablauf:**

Bei Programmstart entspricht die Anzeige der linken Abbildung. Die CLR Taste löscht alle Eingaben – auf der Anzeige erscheint wieder die Null. Die ENTER-Taste dient zum Trennen zweier Zahlen. Die Komma-Taste (',') zur Eingabe eines Dezimaltrennzeichens – natürlich ist nur eines pro Zahl möglich. Die Plus-Minus-Taste ('+/-') fügt am Beginn der Zahl ein negatives Vorzeichen ein oder löscht es wieder. Die Operator-Tasten dienen zum Berechnen der Werte. Zu Beginn einer Zahl, darf keine Null ('0') stehen. Die Anzeige am Display ist auf maximal 12 Ziffern zu begrenzen.




Programmbeschreibung:

Datei `activity_main.xml`:

Verwende ein `RelativeLayout` um die GUI entsprechend der Abbildung zu erstellen.

Klasse `MainActivity.java` sind folgende Methoden zu implementieren:

- `onDigitClick()`: onClick-Event für alle Ziffern-Buttons (0-9). Registriere die Methode für alle Ziffern über die XML-Datei. Dient zur Anzeige aller Ziffern (0-9) auf dem Display.
- `onClear()`: Löschen der aktuellen Anzeige (das Display zeigt 0 an) und Löschen des Stacks. Registriere die Methode über eine innere anonyme Klasse.
- `onKomma()`: Einfügen des Dezimaltrennzeichens (wenn noch nicht vorhanden!). Registriere die Methode über eine innere anonyme Klasse.

-  **onSign ()**: Toggle-Button für negatives Vorzeichen (-). Registriere die Methode über eine innere anonyme Klasse.
-  **onOperator ()**: Handler-Methode für alle Operatoren (+,-,*,/). Berechnung vom Ergebnis und Anzeige am Display. Die beiden Operanden werden vom Stack gelesen, das Ergebnis wird wieder auf dem Stack gespeichert. Registriere die Methode für alle vier Buttons über eine innere Klasse.
-  **onEnter ()**: Einlesen der Eingabe vom Display. Der eingegebene Wert bleibt am Display sichtbar und wird auf einen Stack gelegt. Anschliessend kann eine neue Zahl eingegeben werden. Registriere die Methode über eine innere anonyme Klasse.

Die Anzeige der Zahl am Display erfolgt mit einem String. Die Handlermethoden **onDigit()**, **onKomma()**, **onClear()** und **onSign()** ändern den Anzeige-String. In den Methoden **onOperator()** und **onEnter()** erfolgt die Umwandlung vom Anzeige-String in einen numerischen double-Wert.

Es sind alle Exceptions abzufangen und mit Fehlermeldungen auszugeben. Mögliche Exceptions treten auf bei Division durch Null, oder durch illegale Stack Zugriffe, z.B. wenn bei leerem Stack eine Operator Taste gedrückt wird. Verwende für die Ausgabe der Fehlermeldung einen Toast.

Die Zwischenspeicherung von eingegebenen Zahlen oder von Zwischenergebnissen erfolgt auf einem Stack. Bei Programmstart wird 0 im Display angezeigt.

Die Klasse **Stack** entsprechend dem Klassendiagramm:

push (): legt einen neuen Wert auf den Stack.

pop (): Löscht einen Wert vom Stack und gibt ihn zurück.

isFull (): **true** wenn der Stack voll ist.

isEmpty (): **true** wenn der Stack leer ist.

clear (): löscht den Stack.

Exceptions sind zu werfen wenn auf einen vollen Stack geschrieben wird (Stack overflow), oder von einem leeren Stack gelesen wird.

Stack
- werte : double[10] - top : int
+ push(wert : double) : void + pop() : double + isFull() : boolean + isEmpty() : boolean + clear() : void

Bsp_106_MatheTrainer



Erstelle eine App