Java - Menü und Intent

Menüs

- Auswahl von verschiedenen Möglichkeiten mit Hilfe von Menüeinträgen.
- Menüeinträge können Untermenüs öffnen.
- Definition des Menüs in einer XML-Datei im Ordner *res / menu*.
- Ablage einer ID für das Menü in R.java.
- Verzweigung auf andere Bildschirmseiten.

Kontextmenü

- Menüs, die in einem Kontext zu einem Element stehen.
- Zum Beispiel zu einem Bild wird ein Menü eingeblendet.
- Menüs, welches die gesamte Activity verdecken.
- Sehr seltene Anwendung bei mobilen Geräten.
- Ab Android 3.0 „Contextual action mode“.

Popup-Menüs

- Liste von Menüelementen, die die darunterliegende View teilweise verdeckt.
- Aktivierung durch einen Klick auf eine Schaltfläche.
- Ab Android 3.x.

Definition in der XML-Datei

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />

    <item android:id="@+id/menu_back"
        android:icon="@drawable/ic_ab_back_holo_light"
        android:title="Zurück" />

</menu>
```

Dateiname der xml-Datei

- Der Dateiname beginnt mit einem Kleinbuchstaben.
- Der Dateiname darf nur die Kleinbuchstaben von a...z und die Zahlen von 0...9 widerspiegeln.

… in R.java

```
public static final class menu {
        public static final int popupmenu=0x7f070000;
}
```

- Deklaration in der Datei:

  public static final int dateiname=0x7f070000;
- Jede Datei in dem Ordner *res / menu* symbolisiert eine Konstante in der Klasse menu.
- Der Variablenname beschreibt ein Attribut einer bestimmten Klasse.
- Die Variable ist entsprechend ihres Typs in einer Datei in einem Unterordner des Ordners *res* definiert.

Aufbau der XML-Datei

Prolog

Menü (<menu> </menu>)

Menüelemente (<item />)

Prolog

```
<?xml version="1.0" encoding="utf-8"?>
```

- Der Prolog beginnt mit <?xml und endet mit ?>.
- Die Zeile ist optional.
- Die genutzte Version wird angegeben (version="1.0").
- Die Zeichencodierung (encoding="utf-8") zum Speichern der XML-Datei wird festgelegt. Hier wird der Webstandard UTF-8 genutzt.
- Die Attribut-Werte werden als String übergeben.

Wurzel-Element

```
<menu>
</menu>
```

- Der XML-Tag <menu> … </menu> definiert ein Menü.
- Definitionen von Menüs werden im Ordner *res / menu* abgelegt.

Einbindung eines Namensraumes

```
<menu
        xmlns:android="http://schemas.android.com/apk/res/android"
>
```

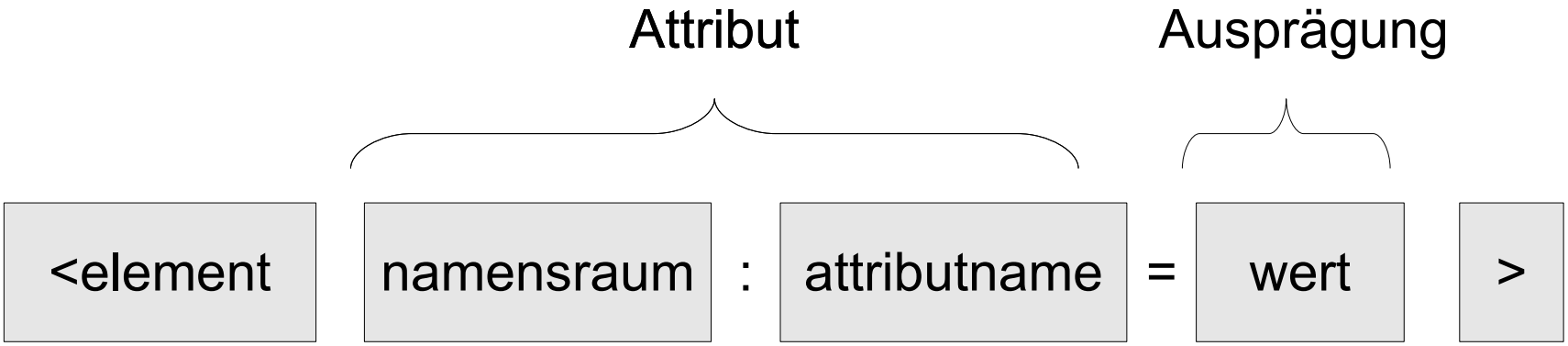
- Einbindung über das Attribut xmlns:android im Wurzelelement.
- Die angegebene URL dient der eindeutigen Identifizierung des Namensraumes.
- Wenn das Menü nicht korrekt geladen werden kann, wird ein Hinweis auf den fehlenden Namensraum eingeblendet.

Menüelemente

```
<item android:id="@+id/itemTemperatur"
        android:title="Temperatur" />
```

- Der XML-Tag <item /> definiert ein Menüelement.

Attribute eines XML-Elemente



- Attribute eines XML-Elementes werden im Start-Tag definiert.
- Mit Hilfe von Attributen werden Eigenschaften eines XML-Elements beschrieben.
- Attribute können von dem Eltern-Element geerbt werden.

Leibniz Universität Hannover

Name eines Attributs

<item	...	:	title
<element	...	:	attributname

- Der Name kennzeichnet eindeutig ein Attribut.
- Bitte beachten Sie die Groß- und Kleinschreibung.

Kategorisierung mit Hilfe des Namensraumes

<item	android	:	title
<element	namensraum	:	attributname

- Die Attribute werden mit Hilfe von Kategorien zusammengefasst. Namenskonflikte werden mit Hilfe der Kategorisierung aufgelöst.
- Der Namensraum und der Name des Attributs werden durch den Doppelpunkt miteinander verbunden.
- Die Angabe des Namensraumes ist zwingend erforderlich.
- Der Namensraum wird im Wurzelelement des Menüs definiert (xmlns:android).

Ausprägung eines Attributs

<item	android	:	title	=	"Länge"	>
<element	namensraum	:	attributname	=	"wert"	>

- Die Ausprägung spiegelt den Attributwert wieder.
- Die Ausprägung wird einem Attribut als String übergeben. Der String wird durch Anführungszeichen begrenzt.
- Der Typ des Wertes selber ist abhängig vom Attribut.

Leibniz Universität Hannover

ID eines Menüelements

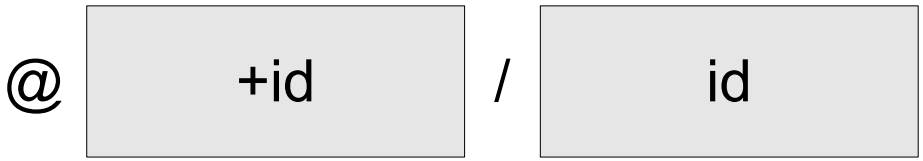
```
android:id="@+id/itemLaenge
```

- Dem Attribut android:id wird eine Ressourcen-ID übergeben.
- Die ID wird in die Datei R.java eingetragen.

Hinweise zum Namen

- Der Name muss eindeutig sein.
- Der Name beginnt immer mit einem Buchstaben.
- Es sollten nur die Buchstaben A ... Z, a .. z und die Ziffern von 0 bis 9 genutzt werden. Der Name darf keine Sonderzeichen enthalten.
- Wörter, in zusammengesetzten Namen, können durch den Unterstrich getrennt werden. Andere Möglichkeit: Kamel-Notation. Das erste Wort beginnt mit einem Kleinbuchstaben. Alle nachfolgenden Wörter beginnen mit einem Großbuchstaben.

Erzeugung einer Ressourcen-ID



- Ressourcen werden durch den Klammeraffen gekennzeichnet.
- Die ID eines Menüelements wird in der xml-Datei mit Hilfe des Attributs android:id definiert.
- Der Kasten „id“ kann durch einen beliebigen benutzerdefinierten Namen ersetzt werden. Dieser Name wird in der Datei *R.java* als Variablennamen für einen Integer-Wert angezeigt.
- +id erzeugt eine neue Ressourcen-ID.

… in R.java

```
public static final class id {
        public static final int buttonMenue=0x7f080000;
        public static final int itemGeschwindigkeit=0x7f080003;
        public static final int itemGewicht=0x7f080004;
        public static final int itemLaenge=0x7f080002;
        public static final int itemTemperatur=0x7f080001;
    }
```

- Deklaration in der Datei:

  `public static final int itemGeschwindigkeit=0x7f080003;`
- Jedes Menüelememt wird in der Klasse `id` als konstante Klassenvariable definiert.

Definition eines Buttons in layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout >
    <Button
        android:id="@+id/buttonMenue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="@dimen/abstandLeft"
        android:layout_marginTop="@dimen/abstandTop"
        android:text="Menue" />
</RelativeLayout>
```

Verweis auf den Button in MainActivity.java

```
import android.widget.Button;
import android.view.View;

public class MainActivity extends Activity{
    private Button btnMenue;

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnMenue = (Button)findViewById(R.id.buttonMenue);
    }
```

Verweis auf das Widget

```
private Button btnMenue;
btnMenue = (Button)findViewById(R.id.buttonMenue);
```

- Die Konstante R.id.buttonMenue in der Datei *R.java* verweist auf ein Widget in der Layout-Datei der Activity.
- Die Methode findViewById() gibt ein Verweis auf eine x beliebige View zurück. Die gewünschte View wird mit Hilfe der ID ermittelt.
- Die Referenz auf eine x beliebige View wird in ein Verweis auf eine Schaltfläche (Button) konvertiert.

… und in der Objektvariablen speichern

```
private Button btnMenue;
btnMenue = (Button)findViewById(R.id.buttonMenue);
```

- Die Methode findViewById() gibt ein Verweis auf eine x beliebige View zurück. Dieser Verweis wird in den gewünschten Typ konvertiert.
- Der konvertierte Verweis wird mit Hilfe des Gleichheitszeichens einer Objektvariablen vom Typ Button zugewiesen.
- Die Objektvariable und das Objekt, auf welches verwiesen wird, sollten vom gleichen Typ sein.

Implementation des Event-Listener

```
import android.widget.Button;
import android.view.View;

public class MainActivity extends Activity
implements OnClickListener{
    private Button btnMenue;

    @Override
    public void onCreate(Bundle savedInstanceState){
        btnMenue = (Button)findViewById(R.id.buttonMenue);
        btnMenue.setOnClickListener(this);
    }
```

Activity-Klasse als Listener-Klasse

```
import android.view.View.OnClickListener;

public class MainActivity extends Activity implements OnClickListener
{
}
```

- Die Klasse MainActivity erbt von der Klasse Activity und implementiert die Schnittstelle OnClickListener.
- Die Klasse MainActivity verpflichtet sich alle Methoden der Klasse OnClickListener zu implementieren (implements).
- Die Klasse OnClickListener hat die Methode onClick(), die in der Klasse implementiert werden muss.

Registrierung des Event-Listener

```
btnMenue.setOnClickListener(this);
```

- Mit Hilfe der Methode setOnClickListener() des Widgets wird das passende Listener-Objekt registiert.
- Das Schlüsselwort this ist ein Platzhalter für das Objekt, welches die Methode aufgerufen hat. this ist in diesem Beispiel ein Platzhalter für Objekte von MainActivity.

Implementierung der Methode onClick()

```
public void onClick(View v) {
        PopupMenu popup = new PopupMenu(MainActivity.this,
                                                    btnMenue);
        MenuInflater inflater = popup.getMenuInflater();
        inflater.inflate(R.menu.popupmenu, popup.getMenu());

        popup.show();
}
```

Objektvariable vom Typ „PopupMenu“

```
import android.widget.PopupMenu;
PopupMenu popup;
```

- PopupMenu popup deklariert eine Objektvariable vom Typ PopupMenu.
- Die Objektvariable speichert einen Verweis auf ein Popup-Menü.
- Die Bibliothek android.widget.PopupMenu muss für ein Objekt diesen Typs eingebunden werden.

Neues Popup-Menü erstellen

```
import android.widget.PopupMenu;
PopupMenu popup = new PopupMenu(MainActivity.this, btnMenue);
```

- Mit Hilfe von `new PopupMenu()` wird eine Instanz von der Klasse erstellt.
- Der Methode `new()` wird ein Kontext übergeben, in dem das Popup-Menü steht (hier die `MainActivity` selbst).
- Das Menü wird wird an die Objektvariable `btnMenue` verankert. Die Objektvariable verweiset auf ein Widget.
- Mit Hilfe des Gleichheitszeichens wird der Verweis auf das konkrete Popup-Menü der Objektvariablen zugewiesen.

MainActivity.this

```
import android.widget.PopupMenu;
PopupMenu popup = new PopupMenu(MainActivity.this, btnMenue);
```

- Das Schlüsselwort this verweist in Java immer auf das aktuelle Objekt.
- Das aktuelle Objekt beschreibt das Objekt, welches die Methode aufgerufen hat.
- Mit Hilfe von MainActivity.this wird auf das aktuelle Objekt der Klasse MainActivity hingewiesen.

Verweis auf einen „XML-Parser“

```
import android.view.MenuInflater;
MenuInflater inflater = popup.getMenuInflater();
```

- MenuInflater inflater deklariert eine Objektvariable vom Typ MenuInflater.
- MenuInflater lesen eine Ressourcen-Datei mit dem Wurzelelement <menu>.
- Mit Hilfe von getMenuInflater() wird ein Parser zum Lesen eines Popup-Menüs erzeugt.
- Die Bibliothek android.view.MenuInflater muss für ein Objekt diesen Typs eingebunden werden.

Lesen der Menü-Ressource

```
import android.view.MenuInflater;
MenuInflater inflater = popup.getMenuInflater();
inflater.inflate(R.menu.popupmenu, popup.getMenu());
```

- Mit Hilfe der Methode inflate() wird das Popup-Menü befüllt.
- Es wird die Ressource R.menu.popupmenu gelesen. Die Konstante ist in R.java definiert und verweist auf eine XML-Datei in *res / menu*.
- getMenu() liefert ein Verweis auf ein bestimmtes Menü. Dieses Menü wird mit Hilfe der Ressource befüllt. In diesem Beispiel ist es das Standardmenü der Klasse MainActivity.

Menü öffnen

```
popup.show();
```

- Mit Hilfe der Methode show() wird das Popup-Menü angezeigt.

Leibniz Universität Hannover

OnClick() eines Menüelements

```
popup.setOnMenuItemClickListener(
new PopupMenu.OnMenuItemClickListener() {
        public boolean onMenuItemClick(MenuItem item)
        {
            return true;
        }
}
);
```

onMenuItemClick()

```
public boolean onMenuItemClick(MenuItem item)
{
    return true;
}
```

- Jedes Menü-Element hat ein Ereignis „Klick auf das Menü-Element“.
- Dem Event-Handler wird ein Verweis auf das Menü-Element, welches angeklickt wurde, übergeben.
- Der Event-Handler gibt an den Aufrufer zurück, ob das ausgelöste Ereignis behandelt wurde oder nicht.

Registerierung des Event-Listener

```
new PopupMenu.OnMenuItemClickListener() {}
```

- Mit Hilfe von `new Klasse.Listener()` wird ein neues Objekt vom Typ des angegebenen Event-Listener erzeugt.
- Es wird ein Event-Listener im Kontext eines Popup-Menüs erzeugt.
- Die Klasse `PopupMenu` hat einen Event-Listener `OnMenuItemClickListener`. Der Event-Listener und die Klasse werden durch einen Punkt verbunden.

Registerierung des Event-Listener

```
popup.setOnMenuItemClickListener()
```

- Mit Hilfe der Methode setOnMenuItemClickListener() des Menüs wird das passende Listener-Objekt registiert.
- Die Methode wird mit einer Objektvariablen durch den Punkt verbunden.
- Für die Objektvariable links vom Punkt wird ein Event-Listener OnMenuItemClick registiert.

Optionen-Menü

- Anzeige von bis zu sechs Menüeinträge am unteren Rand. Falls mehr Einträge vorhanden sind, wird als sechstes Element *More* angezeigt.
- Öffnen mit Hilfe der Schaltfläche *Menu* bis zur Android-Version 2.x.
- Tablets und Geräte ab der Android-Version 3.x haben häufig kein Menu-Button mehr. Dort kann das Menü durch Einträge in die ActionBar ersetzt werden.

Definition in der XML-Datei

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />

    <item android:id="@+id/menu_back"
        android:icon="@drawable/ic_ab_back_holo_light"
        android:title="Zurück" />

</menu>
```

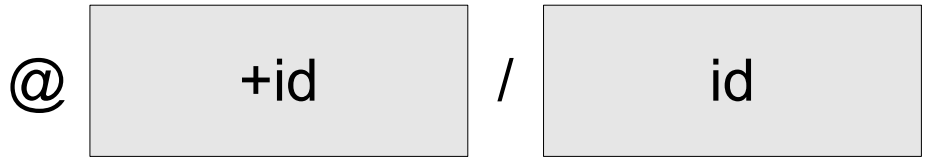
Menüelemente

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />
```

- Mit Hilfe des Tags <item /> werden die verschiedenen Menü-Elemente definiert.
- Falls das Menü-Element kein Untermenü besitzt, kann der Start- und Ende-Tag zusammengefasst werden.

ID eines Menü-Elements

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />
```



- Die ID des Menüelements wird mit Hilfe des Attributs android:id definiert.
- Der Kasten „id“ wird als eine Ressource vom Typ id interpretiert.
- Entsprechend des Typs wird die ID in der Datei R.java neu angelegt.

Icon eines Menü-Elements

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />
```

- Mit Hilfe des Attributs android:icon wird ein Bild für das Menü-Element festgelegt.
- Dem Attribut wird als Wert eine Ressource vom Typ drawable übergeben.

Bilddatei

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
       " />
```

@ drawable / dateiname

- Die Datei ist unter dem angegebenen Namen in dem Ordner *res / drawble* oder deren Alternativen abgelegt.
- Der Dateiname wird ohne die Dateiendung angegeben.
- Bilder können in dem Format „png“, „jpg“ oder „gif“ gespeichert werden. Das Format „png“ sollte bevorzugt werden.

Hinweise zur Gestaltung

- Icons sollten in den Größe 16 x 16 oder 24 x 24 angeboten werden.
- Siehe https://www.google.com/design/spec/style/icons.html#icons-product-icons

Titel eines Menü-Elements

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur" />
```

- Mit Hilfe des Attributs android:title wird die Beschriftung des Menü-Elements festgelegt.

Anzeige in der „Action Bar“

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur"
        android:showAsAction="always"/>
```

- Ab der API 11 kann mit Hilfe des Attributs android:showAsAction das Menü in die Action Bar am unteren Rand eingefügt werden.
- Durch den Attribut-Wert always wird das Menü immer dort, egal wieviel Platz ist, eingefügt.
- Weitere Möglichkeiten siehe http://developer.android.com/guide/topics/resources/menu-resource.html.

Einbindung in die Activity

```
public class MainActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menuoption, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

Event-Handler onCreateOptionsMenu()

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menuoption, menu);
    return super.onCreateOptionsMenu(menu);
}
```

- Der Event-Handler wird beim Erzeugen der Activity automatisiert aufgerufen.
- Dem Event-Handler wird ein Verweis auf ein Menü übergeben.
- Der Event-Handler gibt an den Aufrufer zurück, ob das Menü angezeigt werden soll oder nicht.

Parameter des Event-Handlers

```
public boolean onCreateOptionsMenu(Menu menu) {
}
```

- Initialisierung eines Standard-Optionenmenü mit Hilfe des Parameters.
- Das Standard-Optionenmenü ist in *res / menu* hinterlegt.

Rückgabewert des Event-Handlers

```
return super.onCreateOptionsMenu(menu);
```

- True. Das Menü wird angezeigt.
- False. Das Menü wird nicht angezeigt.
- In diesem Beispiel wird der Event-Handler der Basisklasse aufgerufen. In Abhängigkeit des Rückgabewertes dieses Event-Handlers wird das Menü geöffnet oder nicht.

Untermenü

- Mit Hilfe eines Menüelements wird ein weiteres Menü geöffnet.
- Abbildung von verschiedenen Menüelementen zu einem Oberbegriff.

Leibniz Universität Hannover

Definition in der XML-Datei

```
<item android:id="@+id/menu_temperatur"
        android:icon="@drawable/ic_temperatur"
        android:title="Temperatur"
        android:showAsAction="always">

      <menu
            xmlns:android="http://schemas.android.com/apk/res/android">

        <item android:id="@+id/menu_fahrenheit"
              android:icon="@drawable/ic_temperatur"
              android:title="... in Fahrenheit" />
      </menu>
</item>
```

Verschachtelung von Menüs und Menüelementen

```
<menu>
    <item android:id="@+id/menu_temperatur">

        <menu>
            <item android:id="@+id/menu_Kelvin">

            </item>
        </menu>
    </item>
</menu>
```

Welches Menüelement wurde gewählt?

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_temperatur:
            return true;

        default:
            return super.onOptionsItemSelected(item);
    }

}
```

Event-Handler onOptionsItemSelected()

```
public boolean onOptionsItemSelected(MenuItem item) { }
```

- Parameter: Auf welches Menüelement wurde geklickt? Welches Menüelement wurde ausgewählt.
- Rückgabewert: Wurde das ausgelöste Ereignis behandelt oder nicht?

Fallunterscheidung

```
switch (item.getItemId()) {
        case R.id.menu_temperatur:
            return true;
```

- Dem Schlüsselwort switch folgt in runden Klammern die zu untersuchende Variable.
- Mit Hilfe von item.getItemId() wird die ID des gedrückten Menüelements untersucht.
- case R.id.[menuelement]: Für jedes Menüelement wird mit Hilfe des Schlüsselwortes case ein Fall angelegt.
- Mit Hilfe des Schlüsselwortes return wird die Fallbetrachtung abgeschlossen.

Standardfall

```
switch (item.getItemId()) {
        default:
            return super.onOptionsItemSelected(item);
```

- Mit Hilfe des Schlüsselwortes default wird der Standardfall betrachtet.
- Der Standardfall fängt alle nicht mit case behandelten Fälle ab.
- In diesem Beispiel wird der Event-Handler onOptionsItemSelected der Basisklasse aufgerufen. Der Rückgabewert der Methode wird mit Hilfe von return zurückgegeben.

ListViews als Menü nutzen

- Liste von scrollbaren Elementen.
- Definition in *android.widget.ListView*.

ListView in einer Layout-Datei

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
>
   <ListView
      android:id="@+id/massangaben"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:entries="@array/listKategorie"
   />

</RelativeLayout>
```

Definition einer ListView

```
<ListView
    android:id="@+id/massangaben" />
```

- Der XML-Tag <ListView /> definiert eine Listenansicht in der Layout-Datei.
- Mit Hilfe des Attributs android:id wird der Listenansicht eine eindeutige ID gegeben.

Elemente einer ListView

```
<ListView
    android:id="@+id/massangaben"
     android:entries="@array/listKategorie"
/>
```

- Mit Hilfe des Attributs android:entries werden die Listenelemente definiert.
- In diesem Beispiel wird ein Array von Strings genutzt.

Ressource „Array von Strings“

@ array / name

- Mit Hilfe des Schlüsselwortes array wird eine Ressource vom Typ „Array von Strings“ beschrieben.
- Rechts vom Schrägstrich wird der Name des Arrays angegeben.
- Das Array von Strings wird in der Datei strings.xml im Ordner *res / values* definiert.

Leibniz Universität Hannover

Array von String in strings.xml

```
<resources>
    <string-array name="listTemperatur">
        <item>Celsius</item>
        <item>Fahrenheit</item>
        <item>Kelvin</item>
    </string-array>
</resources>
```

Start- und Ende-Tag

```
<string-array name="listTemperatur">


 </string-array>
```

- Ein String-Array beginnt mit dem Tag <string-array> und endet mit </string-array>.
- Jedes Array hat das Attribut name. Dem Atttribut wird als Wert ein eindeutiger Name übergeben. In diesem Beispiel wird der Name listTemperatur zur Identifizierung des Arrays genutzt.

Elemente des Arrays

```
<string-array name="listTemperatur">
    <item>Celsius</item>
    <item>Fahrenheit</item>
</string-array>
```

- Jedes Element in dem Array beginnt mit <item> und endet mit </item>.
- Zwischen dem Start- und Ende-Tag wird der Titel des Elements gesetzt.

Klick auf ein Listenelement

```
AdapterView.OnItemClickListener itemClickListener =
    new AdapterView.OnItemClickListener()
{
     public void onItemClick(AdapterView<?> listView, View v,
                                  int position, long id)
     {
        String selectedFromList =(String)
                                      (listView.getItemAtPosition(position));
        Toast.makeText(v.getContext(),"Klick auf : " +
                        selectedFromList ,Toast.LENGTH_SHORT).show();
     }
};
```

Event Handle onItemClick

```
public void onItemClick(AdapterView<?> listView, View v,
                                        int position, long id)
{
}
```

- Der Benutzer klickt auf ein Element in der ListView.
- Der Event-Handle gibt keinen Wert an den Aufrufer zurück.

1. Parameter des Event-Handlers

```
public void onItemClick(AdapterView<?> listView, ...)
{
}
```

- Als erster Parameter wird die AdapterView übergeben, in der das Ereignis ausgelöst wurde.
- Die AdapterView ist eine Schnittstelle zwischen einer ViewGroup und deren Elemente. Als ViewGroup kann eine ListView, GridView, Spinner, etc. genutzt werden.
- <?> kennzeichnet die generische Erzeugung der Schnittstelle in Abhängigkeit des auslösenden Elements.

2. Parameter des Event-Handlers

```
public void onItemClick(AdapterView<?> listView, View v, ...)
{
}
```

- Die View (das Element), welches ausgewählt wurde.

3. Parameter des Event-Handlers

```
public void onItemClick(AdapterView<?> listView, View v,
                        int position, ...)
{
}
```

- Die Elemente der ListView werden von 0 bis n durchnummeriert.
- Die Position gibt die Position des ausgewählten Listenelements zurück.

4. Parameter des Event-Handlers

```
public void onItemClick(AdapterView<?> listView, View v,
                        int position,  long id)
{
}
```

- Der Parameter id gibt die ID des gedrückten Listenelements als Ganzzahl zurück.

Erzeugung des Event-Listener

```
new AdapterView.OnItemClickListener() {}
```

- Mit Hilfe von `new Klasse.Listener()` wird ein neues Objekt vom Typ des angegebenen Event-Listener erzeugt.
- Es wird ein Event-Listener im Kontext einer `AdapterView` erzeugt.
- Die Klasse `AdapterView` hat einen Event-Listener `OnItemClickListener`. Der Event-Listener und die Klasse werden durch einen Punkt verbunden.

Anonyme Event-Listener

```
import android.widget.AdapterView;

AdapterView.OnItemClickListener itemClickListener =
        new AdapterView.OnItemClickListener()
```

- Die Objektvariable verweist auf ein Event-Listener vom Typ AdapterView.OnItemClickListener.
- Der Objektvariablen wird ein Verweis auf den neu erzeugten Event-Listener übergeben.

Verweis auf eine ListView

```
ListView listView = (ListView) findViewById(R.id.massangaben);
listView.setOnItemClickListener(itemClickListener);
```

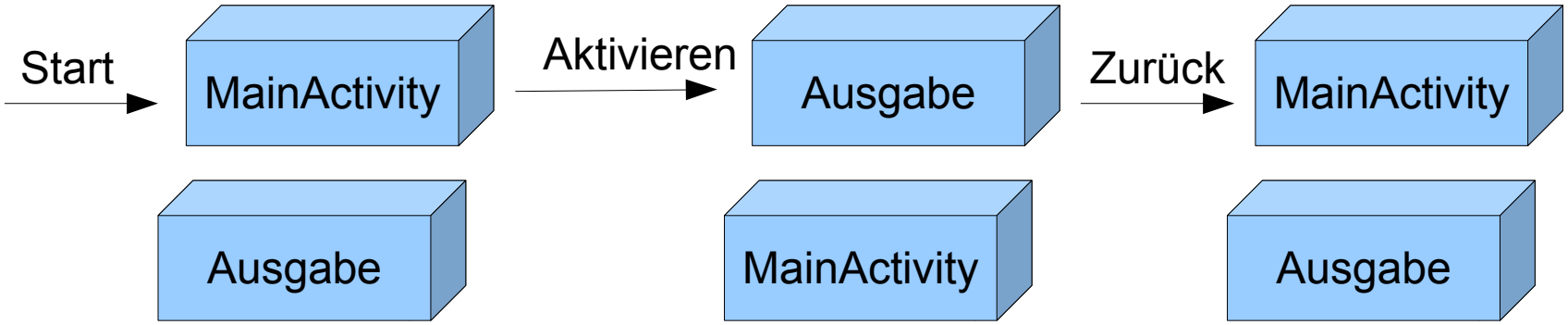
- Mit Hilfe von der Methode findViewById() wird ein Verweis auf eine x beliebige View zurückgegeben. Die View wird durch den Parameter festgelegt.
- Der Verweis wird in die gewünschte View konvertiert.
- Der konvertierte Verweis wird in einer Objektvariablen vom gleichen Typ gespeichert.

Verweis auf eine ListView

```
ListView listView = (ListView) findViewById(R.id.massangaben);
listView.setOnItemClickListener(itemClickListener);
```

- Mit Hilfe der Methode `setOnItemClickListener()` der View wird ein zu der Methode passender Listener registriert.
- In diesem Beispiel wird ein Listener vom Typ `OnItemClickListener` für die ListView registriert.

Mehrere Bildschirmseiten in einer App



MainActivty

```
package de.AndroidExample14_Intents;
```

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
…
```

```
public class MainActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState){ }
    public boolean onCreateOptionsMenu(Menu menu) {}
    public boolean onOptionsItemSelected(MenuItem item) {}
    private void temperaturEingabe(char einheit) {}
}
```

Activity „Ausgabe“

```
package de.AndroidExample14_Intents;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
```

```
public class ActivityAnzeige extends Activity{

    @Override
    public void onCreate(Bundle savedInstanceState){ }

}
```

Hinweise

- Für jede Activity wird eine datei.java in dem Ordner src angelegt.
- Jede Activity wird als Klasse, die von der Basisklasse `Activity` erbt, definiert.
- Jede Activity überschreibt die Methode `onCreate` der Basisklasse. In dieser Methode wird das Layout der Activity festgelegt.
- Jede Activity muss im AndroidManifest.xml deklariert werden.

Leibniz Universität Hannover

OnCreate() einer Activity

```
@Override
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.anzeige);
}
```

- Die Methode initialisiert die Activity.
- Startpunkt der Activity.

Methodenkopf

```
@Override
   public void onCreate(Bundle savedInstanceState)
   {
   }
```

- Die Methode ist öffentlich (public). Auf die Methode kann von außen zugegriffen werden.
- Die Methode gibt an den Aufrufer keinen Wert zurück (void).
- Die Methode bekommt ein Bundle übergeben. Bundles speichern den Status einer Activity. Zum Beispiel ein Nutzer ändert die Orientierung von Hoch auf Quer. Die Daten in einem Formular bleiben erhalten, obwohl die Activity zerstört wurde.

Überschreiben der Methode in der Basisklasse

```
@Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);


   }
```

- super.onCreate() ruft die Methode onCreate() der Basisklasse auf. Der Platzhalter super verweist auf die Basisklasse Activity.
- In jeder überschriebenen Methode aus dem Lebenszyklus einer Activity muss die passende Methode aus der Basisklasse aufgerufen werden.

Layout laden

```
@Override
   public void onCreate(Bundle savedInstanceState{
            setContentView(R.layout.anzeige);
   }
```

- Mit Hilfe der Methode setContentView() wird das Layout für die Klasse geladen.
- Der Methode wird die Ressourcen-ID der Layout-Datei übergeben.
- Ressourcen-IDs werden automatisch generiert und in der Datei *R.java* gespeichert.
- Für Layout-Dateien ist die ID in der Klasse R.layout gespeichert.

Deklaration im AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.AndroidExample14_Intents"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
    <activity android:name="MainActivity"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name="ActivityAnzeige"
          android:label="@string/app_anzeige">
    </activity>
  </application>
</manifest>
```

Leibniz Universität Hannover

AndroidManifest.xml

Prolog: <?xml version="1.0" encoding="utf-8"?>

Package : <manifest xmlns:android="http://schemas.android.com/apk/res/android" >

App: <application android:label="@string/app_name">

Activity:
<activity android:name="ActivityName" > </activity>
<activity android:name="ActivityName" > </activity>
<activity android:name="ActivityName" > </activity>

</application>

</manifest>

Prolog

```
<?xml version="1.0" encoding="utf-8"?>
```

- Der Prolog beginnt mit <?xml und endet mit ?>.
- Die Zeile ist optional.
- Die genutzte Version wird angegeben (version="1.0").
- Die Zeichencodierung (encoding="utf-8") zum Speichern der XML-Datei wird festgelegt. Hier wird der Webstandard UTF-8 genutzt.

Package

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.AndroidExample14_Intents"
    android:versionCode="1"
    android:versionName="1.0">

</manifest>
```

- Die Definition des Manifest beginnt mit <manifest> und endet mit </manifest>.
- Bezeichnung des Packages für eine App.
- In einem Packge werden alle Activities einer App gesammelt.

Name und Namensraum

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.AndroidExample14_Intents"
    android:versionCode="1"
    android:versionName="1.0">
```

- Das Attribut xmlns:android legt den Namensraum fest. In dem Namensraum ist der Paketname eindeutig.
- Das Attribut package definiert den Paketnamen.Dieser Name sollte weltweit eindeutig sein.

Version

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.AndroidExample14_Intents"
    android:versionCode="1"
    android:versionName="1.0">
```

- Die Attribute android:versionCode und android:versionName repräsentieren die Version der Applikation.
- Der Versionscode ist ein ganzzahliger Wert.
- Der Versionname definiert die verschiedenen Versionen innerhalb eines Versionscodes.
- In diesem Beispiel liegt die App in der ersten Version vor. An der Version wurde keine Veränderungen vorgenommen.

Applikation

```
<application android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
</application>
```

- Die Application beginnt mit <application> und endet mit <application>.
- Das Attribut android:label legt eine Bezeichnung für die Application fest. Die Angabe wird als Header angezeigt.
- Das Attribut android:icon definiert ein Icon für die Acitivity.

Attribute

```
<activity android:name="ActivityAnzeige"
                android:label="@string/app_anzeige">
</activity>
```

- Die Activity beginnt mit <activity> und endet mit <activity>.
- Die Acitivity beschreibt eine Bildschirmseite.

Attribute einer Activity

```
<activity android:name="ActivityAnzeige"
            android:label="@string/app_anzeige">
</activity>
```

- Das Attribut android:name definiert eine eindeutige Bezeichnung für die Activity. Der Name der Activity entspricht dem Klassennamen der Acitivity. Der Klassennamen stimmt mit dem Namen der Java-Datei überein.
- Das Attribut android:label definiert eine „Überschrift“ für die Activity.

Intent

- Passives Objekt.
- Beschreibung einer Aktion für das Betriebssystem.
- Impliziter Aufruf mit Hilfe von Intent-Filtern im AndroidManifest.
- Explizierter Aufruf aus einer anderen Activity heraus in der gleichen App durch Angabe des Names.

Implizites Intent

```
<activity android:name="MainActivity"
            android:label="@string/app_name">

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
                    android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>
```

Leibniz Universität Hannover

Intent-Filter

- Definition in der AndroidManifest.xml.
- Die Filter beginnen mit <intent-filter> und enden mit <intent-filter>.
- Der Intent-Filter wird ein Activity zugeordnet.

Aktion

```
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

- Was ist zu tun?
- Konstanten der Klasse android.intent.
- Mit Hilfe des Wertes android.intent.action.MAIN wird definiert, dass am Startpunkt einer Activity begonnen werden soll und kein Content weitergereicht wird.
- Diese Aktion ist nur einmal in der Datei AndroidManifest.xml vorhanden.

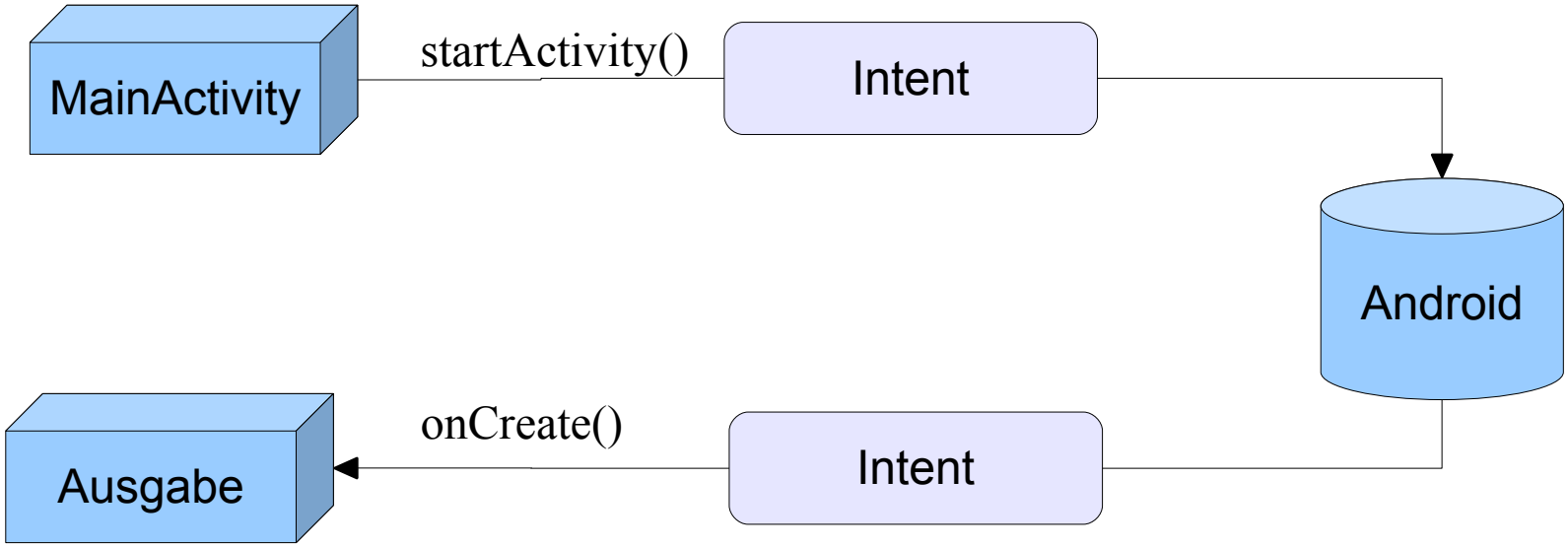
Kategorie

```
<intent-filter>
        <category
            android:name="android.intent.category.LAUNCHER" />
```

- Wo. wann oder warum wird versucht die Activity zu nutzen?
- Die Angabe ist optional.
- Der Wert android.intent.category.LAUNCHER sagt aus, dass der Einstiegspunkt auf der ersten Ebene der Application liegt.

Explizite Intents

- Aufruf von Activities aus der MainActivity heraus.
- Welche Activity soll durch ein Menüelement aufgerufen werden? Welche Daten werden der aufzurufenden Activity als Startwert mitgegeben?



Text senden

```
Intent sendErgebnis = new Intent(Intent.ACTION_SEND);

sendErgebnis.setType("text/plain");

sendErgebnis.putExtra(Intent.EXTRA_TEXT, strText);
sendErgebnis.putExtra(Intent.EXTRA_SUBJECT,
                              "Umrechnung von Celsisu in");

startActivity(sendErgebnis);
```

Objektvariable vom Typ Intent

```
import android.content.Intent;


Intent sendErgebnis;
```

- Für die Deklaration der Objektvariablen muss die Bibliothek android.content.Intent eingebunden werden.
- Die Objektvariable kann einen Verweis auf einen impliziten Intent speichern.

Verweis auf einen implizitenten Intent

```
import android.content.Intent;

Intent sendErgebnis = new Intent(Intent.ACTION_SEND);
```

- Mit Hilfe von new Intent() wird eine Instanz von der Klasse erstellt.
- Der Methode new() wird die Aktion übergeben, die ausgeführt werden soll. Mit Hilfe der Konstanten ACTION_SEND wird unformatierter Text versandt.

Für welche Daten ist die Aktion?

```
Intent sendErgebnis = new Intent(Intent.ACTION_SEND);
sendErgebnis.setType("text/plain");
```

- Mit Hilfe von setType() wird der Format des Contents festgelegt.
- Der Methode werden MIME-Type übergeben. In diesem Beispiel wird unformatierter Text als Content genutzt.

Weitere Informationen

```
Intent sendErgebnis = new Intent(Intent.ACTION_SEND);
sendErgebnis.setType("text/plain");
sendErgebnis.putExtra(Intent.EXTRA_TEXT, strText);
sendErgebnis.putExtra(Intent.EXTRA_SUBJECT,
                                    "Umrechnung von Celsisu in")
```

- Mit Hilfe von putExtra() werden zusätzliche Informationen definiert.
- In diesem Beispiel wird der Methode als erster Parameter eine Konstante der Klasse Intent übergeben, die die Informationen klassifizieren.
- Als zweiter Parameter wird der Text für die Informationen übergeben.

Starten der Activity

```
Intent sendErgebnis = new Intent(Intent.ACTION_SEND);
startActivity(sendErgebnis);
```

- Die Beschreibung der zu startenden Activity wird der Methode startActivity übergeben.
- In diesem Beispiel wird eine E-Mail angezeigt.

Eigene Activity starten

```
Intent showErgebnis = new Intent(MainActivity.this,
                                 ActivityAnzeige.class );
showErgebnis.putExtra("AUSGABE", strText);
startActivity(showErgebnis );
```

Verweis auf das implizite Intent

```
Intent showErgebnis = new Intent(MainActivity.this,
                                 ActivityAnzeige.class );
showErgebnis.putExtra("AUSGABE", strText);
startActivity(showErgebnis );
```

- In welcher Umgeben soll die Aktion starten? Mit Hilfe von MainActivity.this wird die neue Activity im Kontext der MainActivity gestartet.
- Welche Activity soll gestartet werden? Es wird eine Klasse ActivityAnzeige gestartet. Die Deklaration der Activity ist in der Datei AndroidManifest.xml vorhanden.

Weitergabe von Informationen

```
Intent showErgebnis = new Intent(MainActivity.this,
                                 ActivityAnzeige.class );
showErgebnis.putExtra("AUSGABE", strText);
```

- Mit Hilfe von putExtra() werden zusätzliche Informationen definiert.
- In diesem Beispiel wird der Methode als erster Parameter eine eine selbstdefinierte ID für die zusätzliche Information übergeben.
- Als zweiter Parameter wird die Information für die zu startende Activity der Methode übergeben.

Starten der Activity

```
Intent showErgebnis = new Intent(MainActivity.this,
                                 ActivityAnzeige.class );
showErgebnis.putExtra("AUSGABE", strText);
startActivity(showErgebnis );
```

- Die Beschreibung der zu startenden Activity wird der Methode startActivity übergeben.
- In diesem Beispiel wird eine Klasse ActivityAnzeige gestartet.

Weitere Informationen verarbeiten

```
import android.content.Intent;
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.anzeige);
        Intent intent = getIntent();
        Bundle daten = intent.getExtras();

        TextView ausgabeFeld = (TextView)
                                        findViewById(R.id.HinweisAusgabe);
        ausgabeFeld.setText(daten.getString("AUSGABE"));
    }
```

Wer hat die Activity gestartet?

```
Intent intent = getIntent();
```

- In der Objektvariablen wird ein Verweis auf das Intent gespeichert, welches die Activity gestartet hat.

Zusätzliche Informationen laden

```
Intent intent = getIntent();
Bundle daten = intent.getExtras();
```

- Die Methode getExtras() gibt die zusätzlichen Informationen als Bundle zurück.
- Die Methode gibt eine Feld von String-Variablen zurück.

Zusätzliche Informationen lesen

```
Intent intent = getIntent();
Bundle daten = intent.getExtras();
daten.getString("AUSGABE")
```

- Der Methode getString wird eine ID für eine zusätzliche Information übergeben.
- Mit Hilfe der ID gibt die Methode den passenden Wert zu der ID zurück.