Android-Programmierung

Teil I – Grundlagen

Thomas Mahr

Technische Hochschule Nürnberg Georg Simon Ohm
Fakultät EFI
thomas.mahr@th-nuernberg.de
www.tmahr.de
www.twitter.com/tmohm

6. April 2017

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Inhaltsverzeichnis

Inhaltsverzeichnis

In C++ kann man Objekte statisch auf dem Stapelspeicher (Stack) und dynamisch auf dem Freispeicher (Heap) anlegen.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6    string  s1("s1");
7    string* s2 = new string("s2");
8    cout << s1 << " " << *s2 << endl;
9    delete s2;
10 }
```

CppJava/objekt.cpp

Ein Objekt wird auf dem Stapelspeicher angelegt (Stack).

Ein zweites Objekt wird dynamisch auf dem Freispeicher angelegt (Heap). `s2` zeigt auf das Objekt.

Die beiden Objekte werden verwendet. Den Wert des zweiten Objekts erhält man über `*s2`.

Dynamisch angelegte Objekte werden nicht automatisch gelöscht, sondern müssen mittels `delete` gelöscht werden.

In Java kann man Objekte nur dynamisch auf dem Freispeicher (Heap) anlegen.

```
1  class Objekt
2  {
3    public static void main(String[] args)
4    {
5      String s1 = new String("s1");
6      System.out.println( s1 );
7    }
8  }
```

CppJava/Objekt.java

Ein Objekt wird dynamisch auf dem Freispeicher angelegt (Heap). `s1` ist eine Referenz auf das Objekt. Das mit **new** angelegte Objekt wird nicht mittels `delete` gelöscht, sondern wird *zum geeigneten Zeitpunkt* vom *Garbage Collector* gelöscht.

Auf das Objekt greift man direkt über die Referenz zu.

Inhaltsverzeichnis

In C++ kann man Objekte per Wert, per Referenz oder per Zeiger übergeben.

```
1  #include <iostream>
2  using namespace std;
3
4  void f1(string s) {
5    cout << "f1: " << s << endl;
6  }
7
8  void f2(string* s) {
9    cout << "f2: " << *s << endl;
10 }
11
12 void f3(string& s) {
13   cout << "f3: " << s << endl;
14 }
15
16 int main() {
17   string  s("s");
18   f1(s);
19   f2(&s);
20   f3(s);
21 }
```

Empfängt Objekt per Wert

Empfängt Objekt per Adresse

Empfängt Objekt per Referenz

Übergabe per Wert

Übergabe per Adresse

Übergabe per Referenz

CppJava/uebergabe.cpp

In Java werden Objekte immer per Referenz übergeben.

```
1  class Uebergabe
2  {
3    static void f(String s)
4    {
5      System.out.println(s);
6    }
7
8    public static void main(String[] args)
9    {
10     String s = new String("s");
11     f(s);
12   }
13 }
```

CppJava/Uebergabe.java

Empfängt Objekt per Referenz

Übergabe per Referenz

Inhaltsverzeichnis

In Java gibt es keine globalen Objekte und globale Funktionen. Daher muss die `main`-Funktion ein statisches Element einer Klasse sein.

```
1 class Objekt
2 {
3   public static void main(String[] args)
4   {
5     String s1 = new String("s1");
6     System.out.println( s1 );
7   }
8 }
```

CppJava/Objekt.java

Die `main`-Funktion ist eine statische Elementfunktion.

Inhaltsverzeichnis

```java
class Objekt
{
  public static void main(String[] args)
  {
    String s1 = new String("s1");
    System.out.println( s1 );
  }
}
```

CppJava/Objekt.java

- Kompilieren auf der Kommandozeile:
  - `javac Objekt.java`
  - Erzeugt Java-Byte-Code `Objekt.class`
- Java-Programm ausführen:
  - `java Objekt`

Inhaltsverzeichnis

- Mit dem Schlüsselwort **`const`** wird in C++ ein Objekt als konstant deklariert:

```
const A a;
```

  Die Datenelemente dieses konstanten C++ Objekts dürfen nicht mehr verändert werden.
- In Java gibt es dieses Schlüsselwort nicht. Das Schlüsselwort **`final`** verhindert nur die Änderung einer initialen Zuweisung. Eine mit **`final`** deklarierte Referenz darf dann kein anderes Objekt mehr referenzieren. Dies entspricht dem, was in C++ immer für Referenzen gilt. Für elementare Datentype entspricht **`final`** dem **`const`** in C++.

```
1  class Final
2  {
3    public static void main(String[] args)
4    {
5      String r = new String("s1");
6      r = new String("s2");
7      final String f = new String("s3");
8      // f = new String("s3");
9    }
10 }
```

CppJava/Final.java

Die nicht-finale Referenz `r` referenziert das erste String-Objekt.

`r` referenziert jetzt das zweite String-Objekt.

Die Referenz `f` wird als **`final`** deklariert und mit dem dritten String-Objekt initialisiert.

Dieser Aufruf würde zu einem Kompilerfehler führen. Die Referenz `f` darf jetzt kein anderes Objekt mehr referenzieren.

Inhaltsverzeichnis

In Java gibt es 4 Zugriffsrechte:

```
1 class Zugriffsrechte
2 {
3   private int i;
4   protected int j;
5   int k;
6   public int l;
7 }
```

CppJava/Zugriffsrechte.java

Wie in C++.

Wie in C++.

Paketprivat: Zugriff nur für Klassen des Pakets, in dem die Klasse `Zugriffsrechte` liegt.

Wie in C++.

Inhaltsverzeichnis

```cpp
#include <iostream>
using namespace std;

class Elternklasse {
private:
  int i;
protected:
  void f() { cout << "Elternklasse::f()\n"; }
public:
  Elternklasse(int i=0) : i(i) { cout << "Elternklasse::Elternklasse(
     int)\n"; }
  void g() { cout << "Elternklasse::g()\n"; }
};

class Kindklasse : public Elternklasse {
public:
  Kindklasse(int i) : Elternklasse(i) { cout << "Kindklasse::
     Kindklasse()\n"; }
  void f() {
    cout << "Kindklasse::f()\n";
    Elternklasse::f();
    g();
  }
};

int main() {
  Kindklasse k(1);
  k.f();
}
```

CppJava/vererbung.cpp

Elternklasse

In C++ können im Gegensatz zu Java Funktionswerte vorbelegt werden.

C++ bietet im Gegensatz zu Java Initialisierungslisten.

Kindklasse

Aufruf des Konstruktors der Elternklasse.

Zugriff auf Funktion der Elternklasse.

```
1  class Elternklasse {
2    private int i;
3    protected void f() { System.out.println("
         Elternklasse::f()"); }
4    public Elternklasse(int i) {
5      System.out.println("Elternklasse::Elternklasse(int
           )");
6      this.i = i;
7    }
8    public Elternklasse() {
9      System.out.println("Elternklasse::Elternklasse()")
           ;
10     i = 0;
11   }
12   public void g() { System.out.println("Elternklasse::
         g(int)"); }
13 }
```

CppJava/Elternklasse.java

Elternklasse

In Java gibt es keine Vorbelegung von Funktionswerten,

daher muss der Konstruktor überladen werden.

```java
class Kindklasse extends Elternklasse {
  public Kindklasse() {
    super(0);
    System.out.println("Kindklasse::Kindklasse(0)");
  }
  public Kindklasse(int i) {
    super(i);
    System.out.println("Kindklasse::Kindklasse(int)");
  }
  public void f() {
    System.out.println("Kindklasse::f()");
    super.f();
    g();
  }
}
```

CppJava/Kindklasse.java

```java
class Vererbung {
  public static void main(String[] args) {
    Kindklasse k = new Kindklasse(1);
    k.f();
  }
}
```

CppJava/Vererbung.java

Vererbung mittels des Schlüsselwortes **extends**.

Aufruf des Elternkonstruktors.

Zugriff auf Funktion der Elternklasse.

Inhaltsverzeichnis

In C++ gibt es kein Schlüsselwort für eine Schnittstelle. Man kann allerdings eine Schnittstelle so erstellen:

```
1  #include <iostream>
2  using namespace std;
3
4  struct Schnittstelle {
5    virtual ~Schnittstelle() {}
6    virtual void f() = 0;
7  };
8
9  class Impl : public Schnittstelle {
10   void f() { cout << "f()\n";}
11 };
12
13 int main() {
14   Schnittstelle* s = new Impl();
15   s->f();
16   delete s;
17 }
```

CppJava/schnittstelle.cpp

Definition der Schnittstelle mittels einer Struktur. Bei der Struktur sind die Elemente standardmäßig **public**.

Virtueller Destruktor

Abstrakte Schnittstellenfunktion

Implementierung der Schnittstelle

Implementierung der Schnittstellenfunktion

Auf Schnittstelle programmieren

Funktion über Schnittstelle aufrufen

In Java gibt es das Schlüsselwort `interface` für eine Schnittstelle.

```
1  interface Schnittstelle {
2    void f();
3  }
```

CppJava/Schnittstelle.java

Definition der Schnittstelle mittels einer Struktur. Bei der Struktur sind die Elemente standardmäßig `public`.

```
1  class SchnittstelleImpl implements
       Schnittstelle {
2    @Override
3    public void f() {
4      System.out.println("f()");
5    }
6
7    public static void main(String[] args) {
8      Schnittstelle s = new
           SchnittstelleImpl();
9      s.f();
10   }
11 }
```

CppJava/SchnittstelleImpl.java

Schnittstellenfunktion Implementierung der Schnittstelle.

Diese Annotation (Anmerkung) teilt dem Compiler mit, dass der Programmierer eine Funktion eines Supertyps implementieren oder überschreiben möchte. Falls diese Funktion in keinem Supertyp enthalten ist, meldet der Compiler einen Fehler.

Implementierung der Schnittstellenfunktion.

Auf Schnittstelle programmieren.

Funktion über Schnittstelle aufrufen.

In Java kann man anonyme Klassen erstellen. Das sind Klassen, die keinen Namen besitzen, aber dennoch die Implementierung einer Schnittstelle bieten.

```
1  class SchnittstelleImplAnonym {
2    public static void main(String[] args) {
3      Schnittstelle s = new Schnittstelle() {
4        @Override
5        public void f() {
6          System.out.println("f()");
7        }
8      };
9      s.f();
10   }
11 }
```

CppJava/SchnittstelleImplAnonym.java

Wir erzeugen die Instanz einer anonymen Implementierung der Schnittstelle `Schnittstelle`. Vor dem abschließenden Strichpunkt wird in geschweiften Klammern die Funktion der Schnittstelle implementiert.

Inhaltsverzeichnis

```
1  #include <cmath>
2  #include <iostream>
3  #include <sstream>
4  using namespace std;
5
6  namespace MeinNamensraum
7  {
8    double sinus(double arg)
9    {
10     return sin(arg);
11   }
12 };
13
14 int main()
15 {
16   stringstream s;
17   double arg = M_PI/2;
18   s << "sin(" << arg << ")= ";
19   s << MeinNamensraum::sinus(arg);
20   cout << s.str() << endl;
21 }
```

CppJava/namensraum.cpp

Einbinden der im Standardnamensraum `std` extern deklarierten Daten, Objekte und Funktionen mittels Präprozessor-Anweisung.

Öffnen des Standardnamensraums `std`.

Definition eines eigenen Namensraums `MeinNamensraum`.

Verwendung der im Namensraum `MeinNamensraum` definierten Funktion `sinus`-Funktion.

```java
package de.tmahr.android;
import java.math.*;

class Namensraum {
  static double sinus(double arg) {
    return Math.sin(arg);
  }

  public static void main(String[] args) {
    StringBuilder sb = new StringBuilder();
    double arg = Math.PI/2;
    sb.append("sin(").append(arg).append(")= ");
    sb.append(sinus(arg));
    System.out.println(sb.toString());
  }
}
```

CppJava/de/tmahr/android/Namensraum.java

Die Klasse ist im Paket `de.tmahr.android` enthalten. Die Pakete sollen weltweit eindeutig sein. Daher bestehen Paketnamen häufig aus umgedrehten URLs. Die Paketstruktur spiegelt sich in der Verzeichnisstruktur wider.

Über **`import`** können andere Pakete importiert werden. Dies ähnelt der Präprozessordirektive **`#include`** aus C++.

Aufruf der statischen `sin`-Funktion der Klasse `java.math.Math`

- Übersetzen: `javac -cp de.tmahr.android de/tmahr/android/Namensraum.java`
- Ausführen: `java de.tmahr.android.Namensraum`

Inhaltsverzeichnis

Typische Verwendung einer parametrierbaren Container-Klasse in C++:

```
1  #include <iostream>
2  #include <sstream>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7    vector<string*> v;
8    for(int i=0; i<3; i++)  {
9      stringstream ss;
10     ss << "s" << i;
11     v.push_back(new string(ss.str()));
12   }
13   for(string* s : v) {
14     cout << *s  << endl;
15   }
16   for(string* s : v) {
17     delete s;
18   }
19   v.clear();
20 }
```

Mit `string*` parametrierten `vector` anlegen.

Container befüllen.

Durch Container iterieren und Elemente des Containers ausgeben.

Durch Container iterieren und Elemente des Containers löschen.

CppJava/parametrierbar.cpp

Typische Verwendung einer parametrierbaren Container-Klasse in Java:

```
1  import java.util.ArrayList;
2
3  class Parametrierbar
4  {
5    public static void main(String[] args)
6    {
7      ArrayList<String> v = new ArrayList<
           String>();
8      for(int i=0; i<3; i++)
9      {
10       v.add(new String("s"+i));
11     }
12     for(String s : v)
13     {
14       System.out.println(s.toString());
15     }
16   }
17 }
```

CppJava/Parametrierbar.java

Parametrierbaren Container-Datentyp `ArrayList` importieren.

Mit `String` parametrierte `ArrayList` anlegen.

Container befüllen

Durch Container iterieren und Elemente des Containers ausgeben.

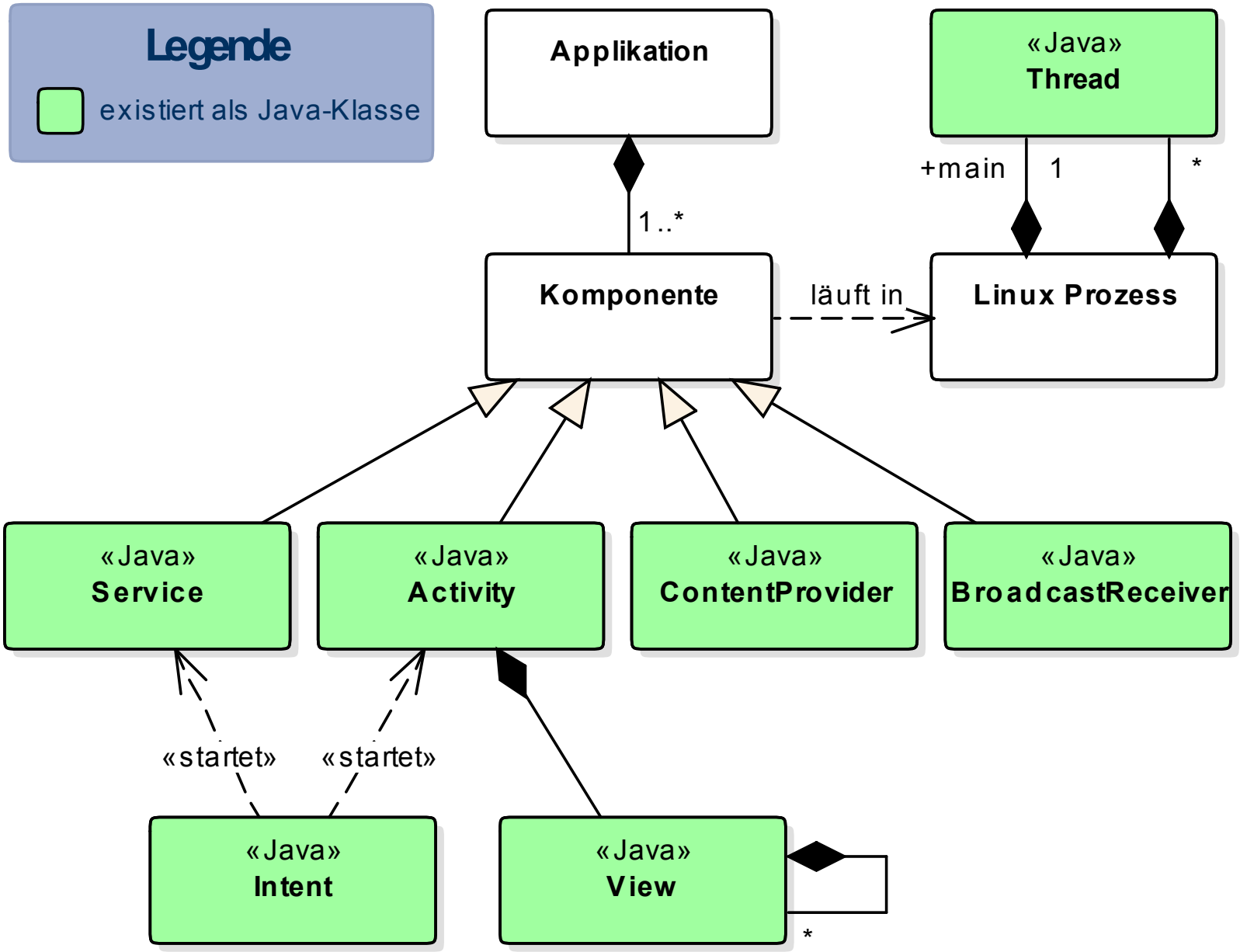
Inhaltsverzeichnis

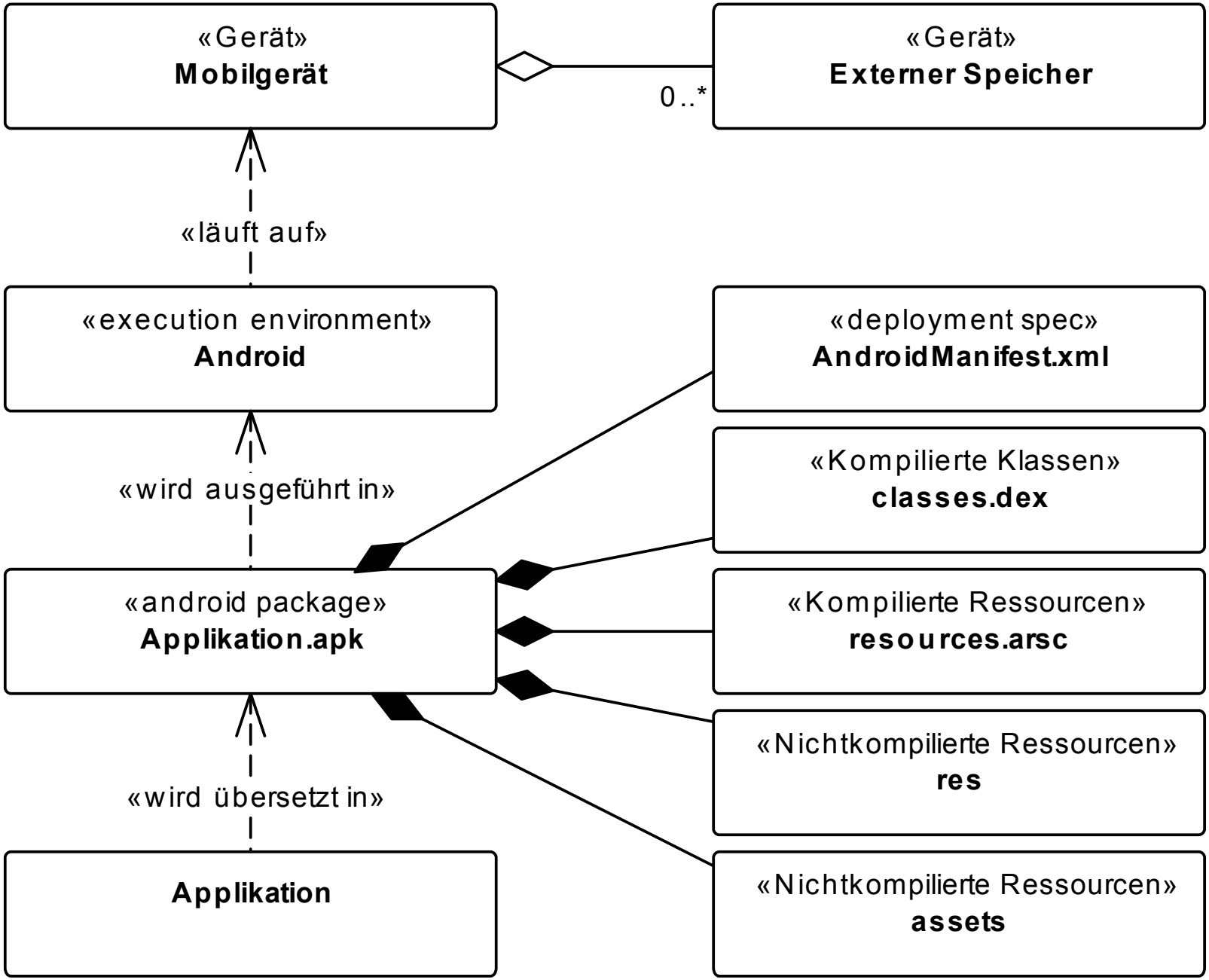
- In Java erben alle Klassen von der Basisklasse `Object` die Funktion `String toString()` (zur `String`-Darstellung des Objekts) und weitere Funktionen.
- In C++ gibt es Header- und Moduldateien (*.h, *.cpp). In Java gibt es diesen Unterschied nicht.
- Java bietet dem Programmierer keine Möglichkeit Operatoren zu überladen. (Daher gibt es auch keine **`friend`**-Mechanismen.)

- In C++ müssen Elementfunktionen mit dem Schlüsselwort **`virtual`** deklariert werden, damit sie virtuell sind. In Java sind alle Elementfunktionen virtuell (falls sie nicht als **`final`** deklariert wurden).
- In Java kann eine als **`final`** deklarierte Elementfunktion nicht überschrieben werden.
- Siehe auch *Comparison of Java and C++* in `http://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B`

Inhaltsverzeichnis

Inhaltsverzeichnis


Legende
existiert als Java-Klasse
Applikation
«Java»
Thread
+main
1
*
1..*
Komponente
läuft in
Linux Prozess
«Java»
Service
«Java»
Activity
«Java»
ContentProvider
«Java»
BroadcastReceiver
«startet»
«startet»
«Java»
Intent
«Java»
View
*


«Gerät»
Mobilgerät
0..*
«Gerät»
Externer Speicher
«läuft auf»
«execution environment»
Android
«wird ausgeführt in»
«android package»
Applikation.apk
«wird übersetzt in»
Applikation
«deployment spec»
AndroidManifest.xml
«Kompilierte Klassen»
classes.dex
«Kompilierte Ressourcen»
resources.arsc
«Nichtkompilierte Ressourcen»
res
«Nichtkompilierte Ressourcen»
assets

Inhaltsverzeichnis

Umfang der Android-API:

- Im Installationsverzeichnis des Android-SDK finden Sie die Java-Quellen der Android-API, z.B. *android-sdk/sources/android-22* für API-Version 22.
- Für die komplette API-Version 22 findet man **10173** Java-Dateien, also 10173 Klassen oder Interfaces.
- Davon fallen **3736** Java-Dateien auf das Paket `android`.

Inhaltsverzeichnis

Dokumentation der Android-API

`http://developer.android.com/index.html`

Inhaltsverzeichnis

Inhaltsverzeichnis

Siehe Live-Demo

Android-Studio im WB.201 und WB.212

- Empfehlung: Verwenden Sie Android-Studio unter Linux.
- Arbeiten Sie unter Linux im Verzeichnis /tmp und sichern Sie Ihr Projekt im Benutzerverzeichnis.

Inhaltsverzeichnis

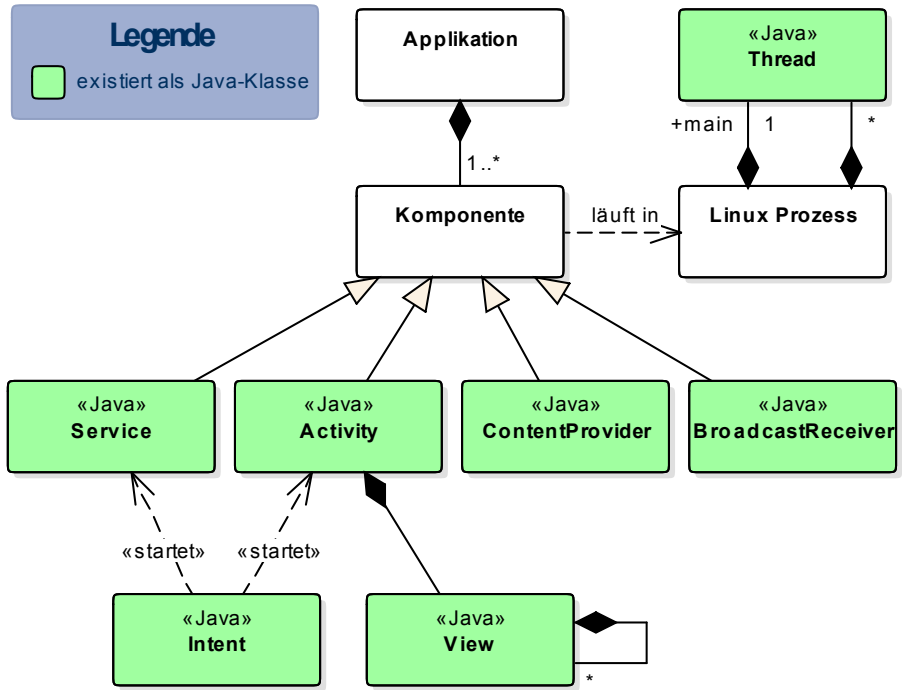
Übung

1. Erstellen Sie ein neues Android-Projekt.
2. Setzen Sie eine `TextView` als `View` der Aktivität.
3. Die Aktivität erstellt eine Instanz der Klasse A.
4. Klasse A besteht aus n Instanzen von B, wobei n dem Konstruktor von A übergeben wird.
5. Klasse B hat einen Namen und einen Wert.
   1. Der Name ist der vollständige Name der Klasse (mit Paketen).
   2. Der Wert wird im Konstruktor der Klasse B zufällig bestimmt (siehe `java.util.Random`).
   3. Die Funktion `String B.toString()` liefert den Text *Name + Wert* zurück.
6. Die Funktion `String A.toString()` liefert für jede Instanz den Wert von `B.toString()` in einer einzelnen Zeile zurück.
7. Die Aktivität zeigt in der `TextView` den Rückgabewert von `A.toString()` an.

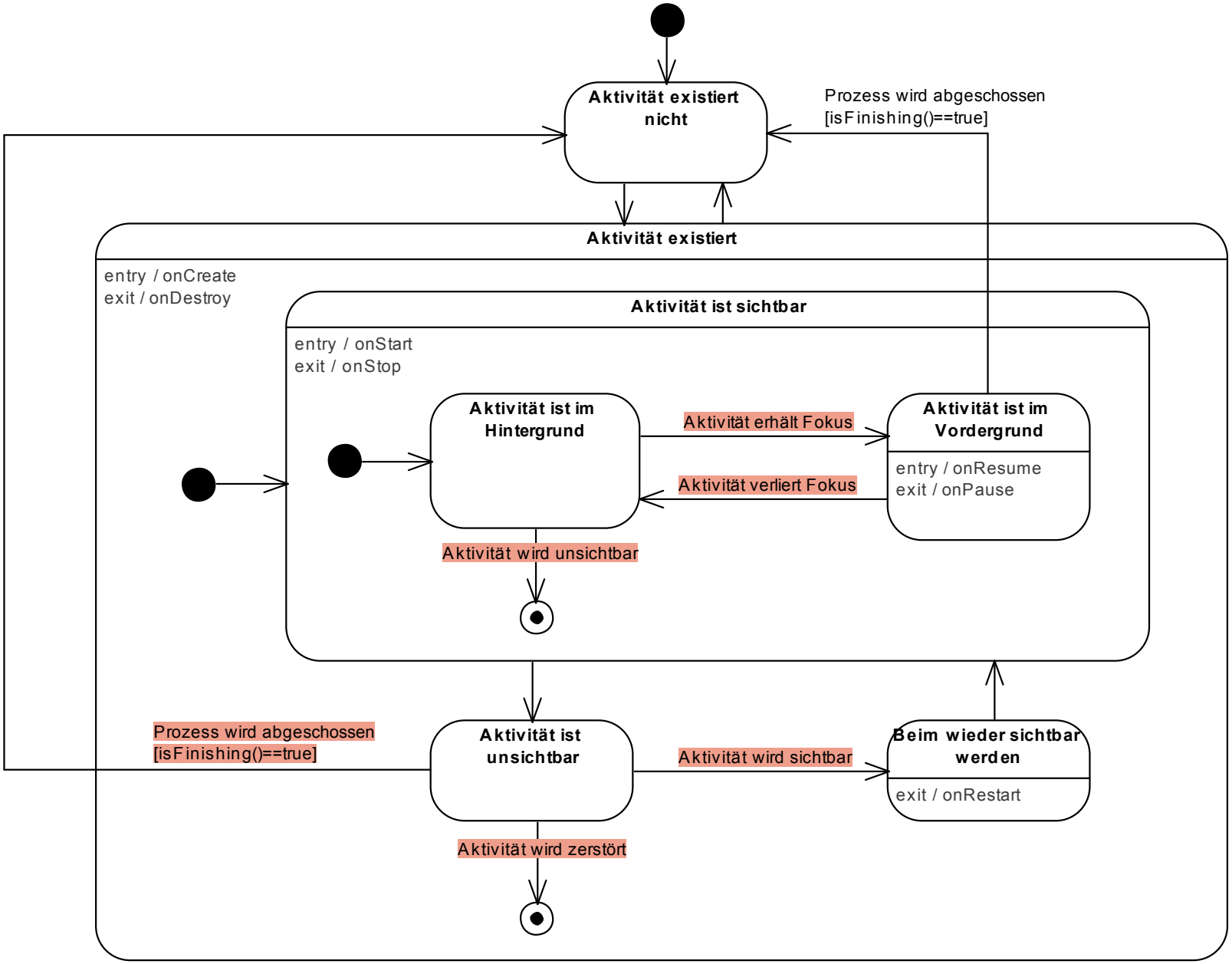
Inhaltsverzeichnis

Aktivität

- Eine Aktivität
  - ist ein Bestandteil einer Applikation und
  - bietet dem Benutzer eine grafische Schnittstelle zur Interaktion.
- Alle Aktivitäten einer Applikation müssen im Android-Manifest eingetragen sein.
- Im Android-Manifest wird die Hauptaktivität vermerkt, die bei Start der Applikation gestartet wird.



Inhaltsverzeichnis



Siehe auch `http://developer.android.com/training/basics/activity-lifecycle/starting.html`

```
1  package de.tmahr.android.unterricht.demoactivity;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.util.Log;
6  import android.view.Menu;
7  import android.view.MenuItem;
8  import android.widget.TextView;
9
10 public class Hauptaktivitaet extends Activity
11 {
12     private TextView textView;
13     private StringBuffer sb = new StringBuffer();
14
15     private void log(String nachricht)
16     {
17         Log.d(this.getClass().getSimpleName(),
               nachricht);
18         sb.append(nachricht).append("\n");
19         textView.setText(sb.toString());
20     }
```

DemoActivity/app/src/main/java/de/tmahr/android/unterricht/demoactivity/Hauptaktivitaet.java (Ausschnitt)

Unsere `Hauptaktivitaet` ist eine `Activity`

Referenz auf ein (noch nicht vorhandenes) Objekt der Klasse `TextView` anlegen

Zum Mitverfolgen des Programmablaufs

Nachricht über *LogCat* ausgeben

Nachricht in der Aktivität anzeigen

```
22      @Override
23      protected void onCreate(Bundle savedInstanceState)
24      {
25          super.onCreate(savedInstanceState);
26          setTitle(this.getClass().getSimpleName());
27          textView = new TextView(this);
28          setContentView(textView);
29          log("onCreate");
30      }
```

DemoActivity/app/src/main/java/de/tmahr/android/unterricht/demoactivity/Hauptaktivitaet.java (Ausschnitt)

Aktivität betritt den Zustand *Aktivität existiert*.

Ruft die Funktion `onCreate` der Elternklasse auf, damit diese auf die Zustandsänderung reagieren kann

Zeigt im Titel der Aktivität den Namen der Klasse an

Legt das `View`-Element zur Anzeige der Log-Nachrichten an und weist der Aktivität das `View`-Element als `View` der Aktivität zu

```
32      @Override
33      public void onPause()
34      {
35          super.onPause();
36          log("onPause");
37          if (isFinishing() == true)
38          {
39              log("isFinishing()==true");
40          }
41          else
42          {
43              log("isFinishing()==false");
44          }
45      }
```

DemoActivity/app/src/main/java/de/tmahr/android/unterricht/demoactivity/Hauptaktivitaet.java
(Ausschnitt)

Aktivität verlässt den Zustand *Aktivität ist im Vordergrund*

liefert `true` zurück, falls die Aktivität beendet wird und liefert `false` zurück, falls die Aktivität pausiert

Beenden einer Aktivität

Beim Beenden einer Aktivität wird die Funktion `onPause` *immer* aufgerufen. Mittels der Funktion `isFinishing` kann man abfragen, ob die Aktivität tatsächlich beendet wird oder nur in den Hintergrund geht.

```
47      @Override
48      public void onResume()
49      {
50          super.onResume();
51          log("onResume");
52      }
53
54      @Override
55      public void onStart()
56      {
57          super.onStart();
58          log("onStart");
59      }
60
61      @Override
62      public void onRestart()
63      {
64          super.onRestart();
65          log("onRestart");
66      }
```

DemoActivity/app/src/main/java/de/tmahr/android/unterricht/demoactivity/Hauptaktivitaet.java
(Ausschnitt)

Aktivität betritt den Zustand *Aktivität ist im Vordergrund*

Aktivität betritt den Zustand *Aktivität ist sichtbar*

Aktivität verlässt den Zustand *beim wieder sichtbar werden*

```
68      @Override
69      public void onStop()
70      {
71          super.onStop();
72          log("onStop");
73      }
74
75      @Override
76      public void onDestroy()
77      {
78          super.onDestroy();
79          log("onDestroy");
80      }
```

DemoActivity/app/src/main/java/de/tmahr/android/unterricht/demoactivity/Hauptaktivitaet.java
(Ausschnitt)

Aktivität verlässt den Zustand *Aktivität ist sichtbar*

Aktivität verlässt den Zustand *Aktivität existiert*

Kippen des Android-Geräts

- Kippt man das Android-Gerät, beendet das Android-Betriebssystem die Aktivität und startet sie neu.
- Auf diese Weise kann das Layout der Aktivität an die neue Lage des Geräts angepasst werden.
- Leider gehen dadurch unsere Logging-Informationen verloren.
- In einem späteren Abschnitt werden wir dieses Problem lösen.

Tastenkürzel des Android Emulators:

Emulated Device Key	Keyboard Key
Power button	F7
Switch to previous layout orientation (for example, portrait, landscape)	KEYPAD_7, Ctrl-F11
Switch to next layout orientation (for example, portrait, landscape)	KEYPAD_9, Ctrl-F12
Toggle fullscreen mode	Alt-Enter
Audio volume up button	KEYPAD_PLUS, Ctrl-F5
Audio volume down button	KEYPAD_MINUS, Ctrl-F6

`http://developer.android.com/tools/help/emulator.html#KeyMapping`

Tipp

1. Nehmen Sie beim Aufruf von `onPause` den „schlimmsten“ Fall an: Die Aktivität geht in den Hintergrund und wird irgendwann vom Betriebssystem wegen Ressourcen-Knappheit beendet.
2. Sichern Sie daher alle Programmeinstellungen, beenden Threads, etc.
3. Beim nächsten Aufruf von `onResume` stellen Sie den ursprünglichen Zustand wieder her.

Inhaltsverzeichnis

Ziele

1. Eine Hauptaktivität soll eine Nebenaktivität starten.
2. Die Hauptaktivität soll der Nebenaktivität Parameter übergeben.
3. Die Nebenaktivität soll der Hauptaktivität Parameter zurückgeben.
4. Die Hauptaktivität soll eine unbekannte Aktivität starten, um eine bestimmte Aufgabe zu erfüllen (z.B. eine HTML-Seite anzeigen).

Die 4 Teilaufgaben setzen wir in 4 Schritten um. Erster Schritt:

1. Wir entwickeln eine Hauptaktivität und eine Nebenaktivität.
2. Die Hauptaktivität erhält einen Schalter.
3. Beim Auslösen des Schalters wird die Nebenaktivität gestartet.
4. Nach Beenden der Nebenaktivität kehrt die Applikation zur Hauptaktivität zurück.

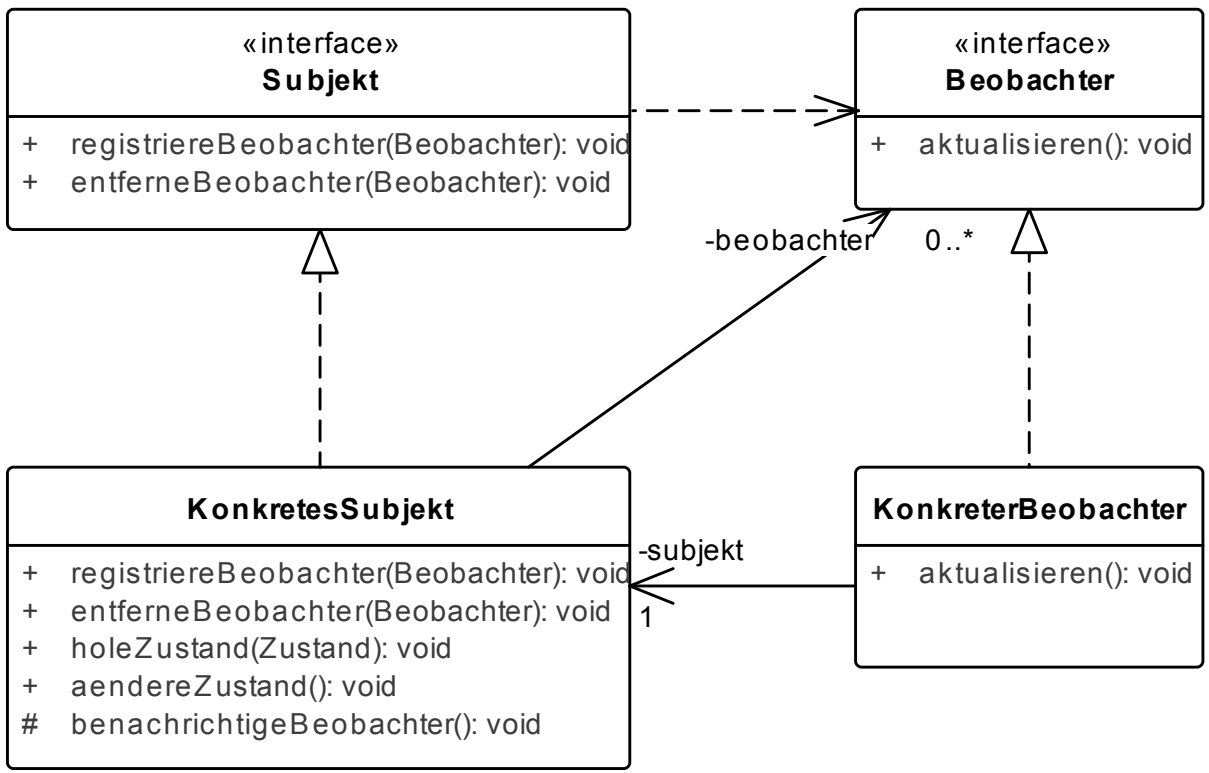
Schalter auslösen:

- Die Hauptaktivität muss darüber informiert werden, dass der Schalter gedrückt wurde.
- Hierfür nutzt Android das *Beobachter-Entwurfsmuster (Observer Design Pattern)*.

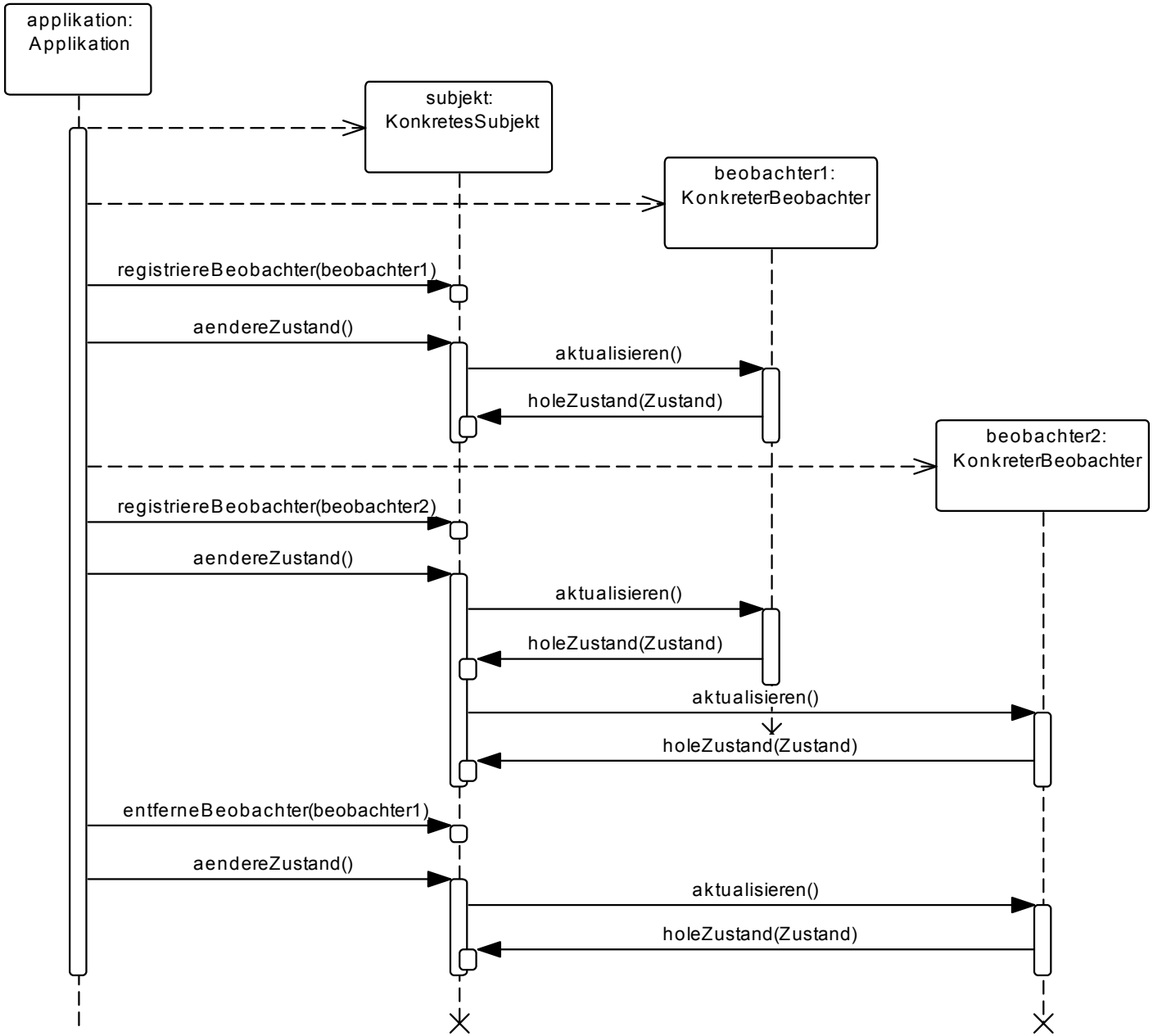
Inhaltsverzeichnis

Beobachtermuster

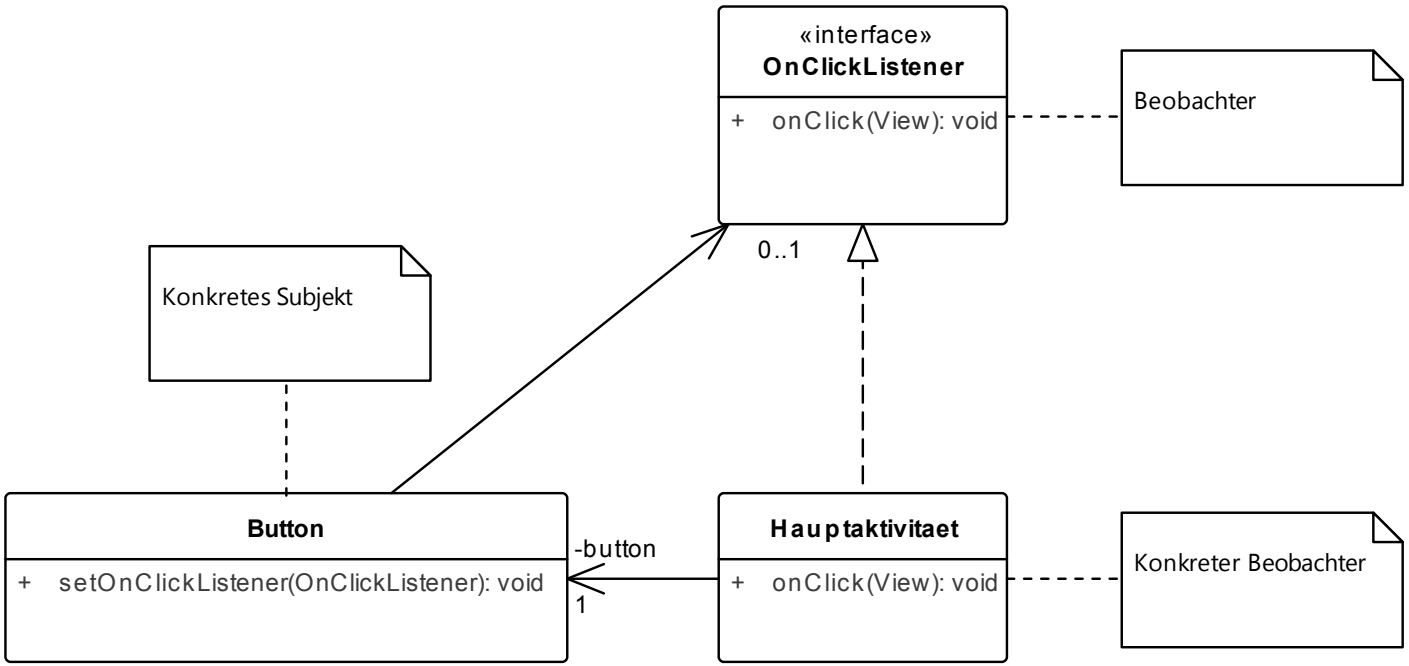
Das Muster definiert eine Beziehung zwischen einem Subjekt und mehreren Beobachtern, wobei das Subjekt alle Beobachter über Zustandsänderungen des Subjekts benachrichtigt.



- Interface Subjekt: Ein konkreter Beobachter muss nur diese Schnittstelle eines Subjekts kennen, um sich bei dem Subjekt zu registrieren.
- Interface Beobachter: Ein Subjekt muss nur diese Schnittstelle eines Beobachters kennen, um den Beobachter über eine Zustandsänderung zu benachrichtigen.
- KonkretesSubjekt:
  - Ein Subjekt kann beliebig viele Beobachter haben.
  - holeZustand(): Über diese Funktion kann sich ein Beobachter den Zustand des Subjekts holen, nachdem er darüber benachrichtigt wurde, dass sich der Zustand des Subjekts geändert hat.
- KonkreterBeobachter: Ein konkreter Beobachter implementiert die Schnittstelle und enthält weitere Funktionen, die allerdings das Subjekt nicht kennen muss.


applikation:
Applikation
subjekt:
KonkretesSubjekt
beobachter1:
KonkreterBeobachter
registriereBeobachter(beobachter1)
aendereZustand()
aktualisieren()
holeZustand(Zustand)
beobachter2:
KonkreterBeobachter
registriereBeobachter(beobachter2)
aendereZustand()
aktualisieren()
holeZustand(Zustand)
aktualisieren()
holeZustand(Zustand)
entferneBeobachter(beobachter1)
aendereZustand()
aktualisieren()
holeZustand(Zustand)

Anwendung des Beobachtermusters auf unsere Aufgabe:



Im Gegensatz zur allgemeineren Form des Beobachtermusters hat das Subjekt hier nur 0 oder 1 Beobachter.

Beobachtermuster auf Code-Ebene::

```
12 public class Hauptaktivitaet1 extends Activity implements
       View.OnClickListener
13 {
14     private Button button;
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java (Ausschnitt)

Die Aktivität implementiert die Beobachter-Schnittstelle `OnClickListener`. Das Interface `OnClickListener` ist in der Klasse `View` enthalten. Daher wird es über `View.OnClickListener` angesprochen.

```
21     @Override
22     protected void onCreate(Bundle savedInstanceState)
23     {
24         super.onCreate(savedInstanceState);
25         setTitle(this.getClass().getSimpleName());
26         button = new Button(this);
27         button.setText("Rufe Nebenaktivität auf");
28         button.setOnClickListener(this);
29         setContentView(button);
30         log("onCreate");
31     }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java (Ausschnitt)

Hier wird der Beobachter (Aktivität) beim Subjekt (Button) registriert.

```
33      @Override
34      public void onClick(View v)
35      {
36          if(v==button)
37          {
38              log("onClick");
39              Intent intent = new Intent(this,
                    Nebenaktivitaet1.class);
40              startActivity(intent);
41          }
42      }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java
(Ausschnitt)

Das Subjekt (Button) ruft diese Funktion des Beobachters (Aktivität) auf, sobald sich der Zustand geändert hat (Schalter gedrückt).

Inhaltsverzeichnis

Intent

- Die Kommunikation zwischen Aktivitäten geschieht über sogenannte *Intents*.
- Intent lässt sich mit *Vorhaben* übersetzen, und drückt aus, dass eine geplante Interaktion zwischen Aktivitäten auch fehlschlagen kann.
- Dies ist insbesondere dann der Fall, wenn eine Hauptaktivität eine Nebenaktivität starten möchte, die namentlich nicht bekannt ist, sondern von der nur sichergestellt werden muss, dass sie eine bestimmte Funktionalität übernehmen kann. Beispiel:
  - Es soll irgendeine Aktivität gestartet werden, die eine HTML-Seite darstellen kann. Das Android-Betriebssystem wählt eine geeignete Aktivität aus, oder bietet dem Nutzer die Auswahl an.

Android-API: `android.content.Intent`

- “An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service.”
- “An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.”

`http://developer.android.com/reference/android/content/Intent.html`

Die zu startende Nebenaktivität ähnelt unserer Aktivität aus dem vorangegangenen Abschnitt:

```
1  package de.tmahr.android.unterricht.demointent;
2
3  import android.app.Activity;
4  import android.content.DialogInterface;
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.view.Menu;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.widget.Button;
11 import android.widget.TextView;
12
13 public class Nebenaktivitaet1 extends Activity
14 {
15     private TextView textView;
16     private StringBuffer sb = new StringBuffer();
17
18     private void log(String nachricht)
19     {
20         Log.d(this.getClass().getSimpleName(), nachricht);
21         sb.append(nachricht).append("\n");
22         textView.setText(sb.toString());
23     }
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState)
27     {
28         super.onCreate(savedInstanceState);
29         setTitle(this.getClass().getSimpleName());
30         textView = new TextView(this);
31         setContentView(textView);
32         log("onCreate");
33     }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Nebenaktivitaet1.java
(Ausschnitt)

Hauptaktivität:

```
1  package de.tmahr.android.unterricht.
       demointent;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.view.Menu;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.widget.Button;
11
12 public class Hauptaktivitaet1 extends
       Activity implements View.OnClickListener
13 {
14     private Button button;
15
16     private void log(String nachricht)
17     {
18         Log.d(this.getClass().getSimpleName()
               , nachricht);
19     }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java (Ausschnitt)

Die Aktivität implementiert die Beobachter-Schnittstelle `OnClickListener`. Das Interface `OnClickListener` ist in der Klasse `View` enthalten. Daher wird es über `View.OnClickListener` angesprochen.

Referenziert den Schalter.

```
21      @Override
22      protected void onCreate(Bundle
            savedInstanceState)
23      {
24          super.onCreate(savedInstanceState);
25          setTitle(this.getClass().
                getSimpleName());
26          button = new Button(this);
27          button.setText("Rufe Nebenaktivität
                auf");
28          button.setOnClickListener(this);
29          setContentView(button);
30          log("onCreate");
31      }
```

`Button` instanziieren.

Hier wird der Beobachter (Aktivität) beim Subjekt (Button) registriert.

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java (Ausschnitt)

```
33      @Override
34      public void onClick(View v)
35      {
36          if(v==button)
37          {
38              log("onClick");
39              Intent intent = new Intent(this,
                    Nebenaktivitaet1.class);
40              startActivity(intent);
41          }
42      }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet1.java
(Ausschnitt)

Das Subjekt (Button) ruft diese Funktion des Beobachters (Aktivität) auf, sobald sich der Zustand geändert hat (Schalter gedrückt). Als Argument übergibt der Schalter eine Referenz auf sich selbst.

Anhand des Funktionsarguments kann die Aktivität das `View`-Objekt ermitteln, das diese Funktion aufruft. Dies ist wichtig, falls die Aktivität mehrere `View`-Subjekte beobachtet.

`Intent` instanziieren. Das erste Argument ist der `Context`, auf den sich das `Intent` bezieht. Da eine `Activity` ein `Context` ist, wird hier **`this`** übergeben. Das zweite Argument ist ein `Class`-Objekt, das zur Laufzeit für eine bestimmte Klasse steht. Wir übergeben das `Class`-Objekt, das für unsere Nebenaktivität steht.

Aktivität mittels expliziten Intent starten

Inhaltsverzeichnis

Zweiter Schritt:

- Die Hauptaktivität soll der Nebenaktivität Parameter übergeben.
- Hierzu kann man dem `Intent`, mit dem die Nebenaktivität gestartet wird, ein Schlüssel-Wert-Paar mitgeben. In der Nebenaktivität kann aus dem `Intent` über den Schlüssel der Wert ausgelesen werden.
- Für die Übergabe eines **`int`**-Werts dient die Elementfunktion `Intent.putExtra(String schluessel, int wert)`.

Um nicht ein weiteres Projekt anlegen zu müssen gehen wir so vor:

1. Wir erstellen die Aktivitäten `Hauptaktivitaet2` und `Nebenaktivitaet2`.
2. Wir registrieren die beiden Aktivitäten anstelle der ursprünglichen Aktivitäten in `AndroidManifest.xml`.

Geändertes AndroidManifest:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/
       res/android"
3      package="de.tmahr.android.unterricht.demointent" >
4      <application
5          android:allowBackup="true"
6          android:icon="@mipmap/ic_launcher"
7          android:label="@string/app_name"
8          android:theme="@style/AppTheme" >
9          <activity android:name="de.tmahr.android.
               unterricht.demointent.Nebenaktivitaet3" >
10         </activity>
11         <activity
12             android:name="de.tmahr.android.unterricht.
                   demointent.Hauptaktivitaet3"
13             android:label="@string/app_name" >
14             <intent-filter>
15                 <action android:name="android.intent.
                       action.MAIN" />
16                 <category android:name="android.intent.
                       category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20 </manifest>
```

DemoIntent/app/src/main/AndroidManifest.xml

Jede Aktivität, die in der Applikation verwendet wird, muss im Manifest eingetragen sein.

Die Aktivität, die bei Start der Applikation ausgeführt werden soll, muss so markiert sein.

```
14 public class Nebenaktivitaet2 extends Activity
15 {
16     private TextView textView;
17     private StringBuffer sb = new StringBuffer();
18     final static String SCHLUESSEL1 = "SCHLUESSEL1";
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Nebenaktivitaet2.java (Ausschnitt)

Dieser Schlüssel wird von der Hauptaktivität beim Parametrieren des Intents benutzt. Daher ist das statische Datenelement Paket-privat.

```
27     @Override
28     protected void onCreate(Bundle savedInstanceState)
29     {
30         super.onCreate(savedInstanceState);
31         setTitle(this.getClass().getSimpleName());
32         textView = new TextView(this);
33         setContentView(textView);
34         log("onCreate");
35         Intent intent = getIntent();
36         int wert = intent.getIntExtra(SCHLUESSEL1, 0);
37         log("Empfangener Wert: " + wert);
38     }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Nebenaktivitaet2.java (Ausschnitt)

Intent ermitteln, mit dem die Aktivität gestartet wurde.

`int`-Wert über Schlüssel auslesen. Das zweite Funktionsargument 0 dient als Vorgabewert, falls kein Wert ausgelesen werden kann.

```
13 public class Hauptaktivitaet2 extends Activity implements
       OnClickListener
14 {
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet2.java
(Ausschnitt)

```
34   @Override
35   public void onClick(View v)
36   {
37     if(v==button)
38     {
39       log("onClick");
40       Intent intent = new Intent(this,Nebenaktivitaet2.
           class);
41       intent.putExtra(Nebenaktivitaet2.SCHLUESSEL1, 42);
42       startActivity(intent);
43     }
44   }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet2.java
(Ausschnitt)

`Intent` instanziieren. Das erste Argument ist der `Context`, auf den sich das `Intent` bezieht. Da eine `Activity` ein `Context` ist, wird hier **`this`** übergeben. Das zweite Argument ist ein `Class`-Objekt, das zur Laufzeit für eine bestimmte Klasse steht. Wir übergeben das `Class`-Objekt, das für unsere Nebenaktivität steht.

`int`-Wert anhand Schlüssel übergeben.

Aktivität mittels expliziten Intent starten

Inhaltsverzeichnis

Dritter Schritt:

- Die Nebenaktivität soll der Hauptaktivität Parameter zurückgeben.
- Hierzu erstellt die Nebenaktivität beim Beenden einen zweiten `Intent`. Diesem wird mit der Funktion `Intent.putExtra(String schluessel, String wert)` mittels eines zweiten Schlüssel ein Wert übergeben. Dieser Wert soll jetzt anstelle eines **`int`**- ein `String`-Wert sein.
- Für die Übergabe eines **`int`**-Werts dient die Elementfunktion `Intent.putExtra(String schluessel, int wert)`.
- Die Hauptaktivität startet jetzt die Nebenaktivität nicht mehr mit `startActivity(Intent intent)` sondern mit `startActivityForResult(Intent intent, int requestCode)`.

- Auf diese Weise wird die Hauptaktivität über das Beenden der Nebenaktivität mittels der Funktion `Activity.onActivityResult(...)` informiert.
- Anhand des `requestCode` kann die Hauptaktivität zwischen verschiedenen Nebenaktivitäten unterscheiden.
- Die Hauptaktivität überschreibt die Funktion `void onActivityResult(int requestCode, int resultCode, Intent intent)`, um auf das Beenden der Nebenaktivität zu reagieren:
  - Das erste Argument `requestCode` entspricht dem `requestCode` des Aufrufs von `startActivity`.
  - Das zweite Argument `resultCode` gibt an, ob die Nebenaktivität ordnungsgemäß abgearbeitet wurde.
  - Das dritte Argument `intent` ist der von der Nebenaktivität erzeugte Intent, der der Parameterübergabe dient.

```
14 public class Nebenaktivitaet3 extends Activity
15 {
16     private TextView textView;
17     private StringBuffer sb = new StringBuffer();
18     final static String SCHLUESSEL1 = "SCHLUESSEL1";
19     final static String SCHLUESSEL2 = "SCHLUESSEL2";
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Nebenaktivitaet3.java (Ausschnitt)

Schlüssel für die Parameterübergabe von der Hauptaktivität an die Nebenaktivität

Schlüssel für die Parameterübergabe von der Nebenaktivität an die Hauptaktivität

```
41     @Override
42     public void onBackPressed()
43     {
44         Intent intent = new Intent();
45         intent.putExtra(SCHLUESSEL2, "onBackPressed");
46         setResult(RESULT_OK, intent);
47         finish();
48     }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Nebenaktivitaet3.java (Ausschnitt)

Wird aufgerufen, wenn die Android-Zurück-Taste gedrückt wurde

`Intent` für die Parameterrückgabe erstellen

Schlüssel-Wert-Paar dem `Intent` hinzufügen. Der Wert ist jetzt ein `String`.

Ordnungsgemäße Abarbeiten der Nebenaktivität im `Intent` vermerken.

Die Nebenaktivität muss jetzt explizit beendet werden, da die Funktion `onBackPressed` überschrieben ist.

```
15    private Button button;
16    private final static int REQUEST_CODE = 747;
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet3.java (Ausschnitt)

Dient zur Kennung der gestarteten Nebenaktivität (wichtig falls Hauptaktivität mehrere Nebenaktivitäten startet)

```
35    @Override
36    public void onClick(View v)
37    {
38      if(v==button)
39      {
40        log("onClick");
41        Intent intent = new Intent(this,Nebenaktivitaet3.class)
              ;
42        intent.putExtra(Nebenaktivitaet3.SCHLUESSEL1, 42);
43        startActivityForResult(intent, REQUEST_CODE);
44      }
45    }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet3.java (Ausschnitt)

Startet die Nebenaktivität mittels expliziten Intent. Die Hauptaktivität wird über das Beenden der Nebenaktivität anhand des `requestCode` informiert.

```
47    @Override
48    protected void onActivityResult(int requestCode, int
          resultCode, Intent intent)
49    {
50      super.onActivityResult(requestCode, resultCode, intent);
51      if(requestCode==REQUEST_CODE)
52      {
53        if(resultCode==RESULT_OK)
54        {
55          String wert = intent.getStringExtra(Nebenaktivitaet3.
              SCHLUESSEL2);
56          button.setText("Rufe Nebenaktivität auf (Rückgabewert
                = " + wert + ")");
57          log("onActivityResult OK");
58        }
59      }
60    }
```

Wird nach Beenden der Nebenaktivität aufgerufen.

Prüfen, ob es sich um die erwartete Nebenaktivität handelt.

Prüfen, ob Nebenaktivität ordnungsgemäß abgearbeitet wurde.

Rückgabewert aus Intent auslesen.

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet3.java (Ausschnitt)

Inhaltsverzeichnis

Vierter Schritt:

- Die Hauptaktivität soll eine unbekannte Aktivität mittels impliziten Intents starten.
- Hierzu übergibt die Hauptaktivität dem Intent nicht mehr die zu startende Klasse der Nebenaktivität sondern einen Parameter, aufgrund dessen das Android-Betriebssystem eine geeignete Aktivität ermittelt.

```
29    @Override
30    public void onClick(View v)
31    {
32      if(v==button)
33      {
34              Random rand = new Random();
35              int r = rand.nextInt(3);
36              switch (r)
37              {
38                  case 0:
39                      starteAktivitaetHttp();
40                      break;
41                  case 1:
42                      starteAktivitaetMail();
43                      break;
44                  default:
45                      starteAktivitaetImage();
46                      break;
47              }
48      }
49    }
```

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet4.java (Ausschnitt)

Pseudozufallszahlengenerator instanziieren.

Zufälligen `int`-Wert im Bereich [0,1,2] bestimmen und je nach Zufallswert eine Nebenaktivität mittels impliziten Intents starten.

```
51      private void starteAktivitaetHttp()
52      {
53          Intent intent = new Intent(Intent.ACTION_VIEW,Uri
                .parse("http://www.th-nuernberg.de"));
54          startActivity(intent);
55      }
56
57      private void starteAktivitaetMail()
58      {
59          Intent intent = new Intent(Intent.ACTION_VIEW,Uri
                .parse("mailto:thomas.mahr@th-nuernberg.de"));
60          intent.putExtra(Intent.EXTRA_SUBJECT, "Hallo");
61          startActivity(intent);
62      }
63
64      private void starteAktivitaetImage()
65      {
66          Intent intent = new Intent(Intent.ACTION_PICK);
67          intent.setType("image/*");
68          startActivityForResult(intent, REQUEST_CODE);
69      }
```

HTTP-Seite anzeigen

E-Mail versenden

Bild auswählen

DemoIntent/app/src/main/java/de/tmahr/android/unterricht/demointent/Hauptaktivitaet4.java
(Ausschnitt)

Inhaltsverzeichnis

Übung

1. Erstellen Sie ein neues Android-Projekt.
2. Die Aktivität A hat einen Schalter.
3. Beim Betätigen des Schalters wird eine neue Instanz von A gestartet.
4. Der Schalter gibt die Tiefe der Aufrufhierarchie an. Beim Applikationsstart ist der Schalter mit 0 beschriftet.
5. Über die Android-Zurück-Taste wird eine Aktivität beendet.

Inhaltsverzeichnis

Weitere Vorgehensweise:

- Wir wollen verschiedene Android-Techniken kennenlernen: Persistenz, Layout, Grafik, Audio, Sensorik, ...
- Um nicht für alle Techniken einzelne Projekte anlegen zu müssen oder die Hauptaktivitäten im Android-Manifest immer wieder ändern zu müssen, erstellen wir eine Hauptaktivität mit einer Liste der Techniken.
- Hinter den Techniken stehen Nebenaktivitäten, die von der Listenaktivität gestartet werden.
- Für diesen Zweck bietet Android eine spezielle Aktivität: `ListActivity`
- Im einfachsten Fall müssen wir uns bei der Benutzung der `ListActivity` nicht um das Layout der Liste kümmern.

```
13 public class Demos extends ListActivity
14 {
15     private String techniken[] = new String[]
16         {
17             DemoLifecycle1.class.getSimpleName(),
18             DemoSharedPreference.class.getSimpleName(),
19             DemoLifecycle2.class.getSimpleName(),
20             DemoSaveInstance.class.getSimpleName(),
21             DemoAssets.class.getSimpleName(),
22             DemoExternerSpeicher.class.getSimpleName(),
23             DemoInternerSpeicher.class.getSimpleName(),
24             DemoDatenbank.class.getSimpleName(),
25             DemoLayout.class.getSimpleName(),
26             DemoLinearLayout.class.getSimpleName(),
27             DemoListView.class.getSimpleName(),
28             DemoLayoutQualifizierer.class.getSimpleName(),
29             DemoFragmentStatisch.class.getSimpleName(),
30             DemoFragmentDynamisch.class.getSimpleName(),
31             DemoThread.class.getSimpleName(),
32             DemoService1.class.getSimpleName(),
33             DemoService2.class.getSimpleName(),
34             DemoIntentService.class.getSimpleName(),
35             DemoBoundService.class.getSimpleName(),
36             DemoBoundIntentService.class.getSimpleName(),
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/Demos.java (Ausschnitt)

Die Klasse `ListActivity` erweitert `Activity` um Listeneigenschaften

Array mit Namen der Nebenaktivitäten initialisieren

```
42      }
43
44      @Override
45      protected void onCreate(Bundle savedInstanceState)
46      {
47          super.onCreate(savedInstanceState);
48          setTitle(this.getClass().getSimpleName());
49          ArrayAdapter adapter = new ArrayAdapter<>(this
                , android.R.layout.simple_list_item_1,
                techniken);
50          setListAdapter(adapter);
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/Demos.java (Ausschnitt)

Der `ArrayAdapter` verbindet das Namens-Array `techniken` mit den Elementen der Liste. Das Layout eines Listenelements ist in der vordefinierten Layout-Ressource `android.R.layout.simple_list_item_1` festgelegt.

Der Adapter verbindet jetzt das Namens-Array mit der in `ListActivity` enthaltenen Liste.

```
51      }
52
53      @Override
54      protected void onListItemClick(ListView l, View v,
            int position, long id)
55      {
56          super.onListItemClick(l, v, position, id);
57          String name = techniken[position];
58          log(name);
59
60          try
61          {
62              Class<?> c = Class.forName("de.tmahr.
                   android.unterricht.demos." + name);
63              Intent intent = new Intent(this, c);
64              startActivity(intent);
65          }
66          catch (ClassNotFoundException e)
67          {
68              e.printStackTrace();
69          }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/Demos.java (Ausschnitt)

`ListActivity` erweitert `Activity` um die Funktion `onListItemClick`. Diese wird aufgerufen, wenn der Benutzer ein Element der Liste ausgewählt hat.

Auch die Elternklasse muss über das Ereignis informiert werden.

Name der zu startenden Nebenaktivität ermitteln.

`Class<T>` ist eine parametrierbare Klasse. `T=?` bedeutet, dass `T` ein beliebiger Typ sein kann. Instanzen von `Class` beziehen sich auf Klassen und Interfaces in einer laufenden Java-Applikation.

Die Suche nach der Klasse kann fehlschlagen.

Inhaltsverzeichnis

- Die erste Nebenaktivität, die aus der Auswahlliste gestartet werden kann, ist ein Demo zum Lebenszyklus einer Aktivität.
- Da wir mit der Technik aus einem vorangegangenen Kapitel vertraut sind, können wir den vorhandenen Code weitgehend unverändert übernehmen.

```java
public class DemoLifecycle1 extends Activity
{
    private TextView textView;
    private StringBuffer sb = new StringBuffer();

    private void log(String nachricht)
    {
        Log.d(this.getClass().getSimpleName(),
            nachricht);
        sb.append(nachricht).append("\n");
        textView.setText(sb.toString());
    }

    @Override
    protected void onCreate(Bundle
        savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setTitle(this.getClass().getSimpleName())
            ;
        textView = new TextView(this);
        setContentView(textView);
        log("onCreate");
    }

    @Override
    public void onPause()
    {
        super.onPause();
        log("onPause");
        if (isFinishing() == true)
        {
            log("isFinishing()==true");
        } else
        {
            log("isFinishing()==false");
        }
    }

    @Override
    public void onResume()
    {
        super.onResume();
        log("onResume");
    }

    @Override
    public void onStart()
    {
        super.onStart();
        log("onStart");
    }

    @Override
    public void onRestart()
    {
        super.onRestart();
        log("onRestart");
    }

    public void onStop()
    {
        super.onStop();
        log("onStop");
    }

    @Override
    public void onDestroy()
    {
        super.onDestroy();
        log("onDestroy");
    }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLifecycle1.java
(Ausschnitt)

Inhaltsverzeichnis

1. Wir wollen nun das offene Probleme lösen, dass beim Kippen des Geräts die bisherige Logging-Information verloren geht, weil die Aktivität neu gestartet wird.
2. Dazu müssen wir die bisherige Logging-Information zwischenspeichern. Hierzu dient die Klasse `SharedPreference`.
3. Außerdem wollen wir die Logging-Funktionen nicht mehr von einer Aktivität in die andere kopieren. Daher kapseln wir diese in einer Klasse `Logger`.

- Mittels des `PreferenceManager` und der `SharedPreferences` lassen sich Daten einer Aktivität speichern.
- Die statische Funktion `PreferenceManager.getDefaultSharedPreferences(context)` liefert die `SharedPreferences` für eine Applikation zurück.
- In unserem Fall werden die Daten im XML-Format an dieser Stelle im Dateisystem des Android-Geräts gespeichert: `/data/data/de.tmahr.android.unterricht.demos/shared_prefs/de.tmahr.android.unterricht.demos_preferences.xml`
- Mittels des *File Explorers* des *Dalvik Debug Monitor Server (DDMS)* (bei Eclipse) oder *Android Device Monitors* (bei Android-Studio) kann man auf die XML-Datei zugreifen.

```
12 public class DemoSharedPreference extends Activity
13 {
14     private TextView textView;
15     private StringBuffer sb = new StringBuffer();
16     private String text;
17     private int wert;
18     private final static String KEY_TEXT = "
           KEY_TEXT_" + DemoSharedPreference.class.
           getSimpleName();
19     private final static String KEY_WERT = "
           KEY_WERT_" + DemoSharedPreference.class.
           getSimpleName();
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSharedPreference.java
(Ausschnitt)

Exemplarischer Text, der mittels `SharedPreferences` gespeichert werden soll.

Exemplarischer Wert, der mittels `SharedPreferences` gespeichert werden soll.

Schlüssel zum Speichern des Texts.

Schlüssel zum Speichern des Werts.

```
39      @Override
40      protected void onPause()
41      {
42          super.onPause();
43          log("onPause");
44          SharedPreferences p = PreferenceManager.
               getDefaultSharedPreferences(this);
45          SharedPreferences.Editor editor = p.edit();
46          editor.putString(KEY_TEXT, text);
47          editor.putInt(KEY_WERT, wert);
48          editor.commit();
49      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSharedPreference.java (Ausschnitt)

Wenn die Aktivität in den Hintergrund geht, werden die Daten gespeichert.

Liefert die `SharedPreferences` anhand des `Context` (**`this`**) für diese Applikation zurück.

Zum Schreiben der Daten wird ein Editor benötigt.

`String` unter entsprechendem Schlüssel speichern.

`int`-Wert unter entsprechendem Schlüssel speichern.

Schreibt die editierten Daten auf das Dateisystem.

```
51      @Override
52      protected void onResume()
53      {
54          super.onResume();
55          log("onResume");
56          SharedPreferences p = PreferenceManager.
               getDefaultSharedPreferences(this);
57          text = p.getString(KEY_TEXT, "");
58          wert = p.getInt(KEY_WERT, 0);
59          log("text=" + text + ", wert=" + wert);
60          text += "+";
61          wert++;
62      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSharedPreference.java
(Ausschnitt)

Wenn die Aktivität in den Vordergrund geht, werden die Daten geladen.

`String` anhand des Schlüssels ermitteln. Falls kein Wert gefunden wird, wird der im zweiten Argument übergebene Vorgabewert zurückgeliefert.

`int` anhand des Schlüssels ermitteln. Falls kein Wert gefunden wird, wird der im zweiten Argument übergebene Vorgabewert zurückgeliefert.

Die Daten würden normalerweise nicht in der onResume-Methode verändert werden, sondern an anderer Stelle.

Inhaltsverzeichnis

```
6  import java.io.PrintWriter;
7  import java.io.StringWriter;
8
9  public class Logger
10 {
11     private TextView textView;
12     private StringBuffer sb = new StringBuffer();
13     private String tag;
14
15     public Logger(String tag, TextView textView, String
           logInitText)
16     {
17         this.tag = tag;
18         this.textView = textView;
19         sb.append(logInitText);
20     }
21
22     public void log(String s)
23     {
24         Log.d(tag, s);
25         sb.append(s).append("\n");
26         if (textView != null)
27         {
28             textView.setText(sb.toString());
29         }
30     }
```

In dieser `TextView` werden die Nachrichten ausgegeben.

Dient der Verkettung aller Nachrichten.

LogCat-Nachrichten benötigen diesen `tag`.

Der Konstruktor erhält neben `tag` und `TextView` einen initialen Nachrichtenpuffer. Dies ist für den Fall wichtig, wenn eine Aktivität aufgrund eines Kipp-Ereignisses neu gestartet werden muss. Dabei wird ein neuer Logger instanziert, der auf diese Weise den bisherigen, wiederhergestellten Nachrichtentext zugewiesen bekommt.

Nachricht ausgeben.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/logger/Logger.java (Ausschnitt)

```
32      public void log(Exception e)
33      {
34          StringWriter sw = new StringWriter();
35          e.printStackTrace(new PrintWriter(sw));
36          log(sw.toString());
37      }
38
39      public void clearLog()
40      {
41          sb.setLength(0);
42          if (textView != null)
43          {
44              textView.setText("");
45          }
46      }
47
48      public String getLoggedText()
49      {
50          return sb.toString();
51      }
52  }
```

Beschreibung einer Ausnahme ausgeben.

Nachrichten-Historie löschen.

Nachrichten-Historie zurückliefern (um sie speichern zu können).

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/logger/Logger.java (Ausschnitt)

Integrieren wir `Logger` in das Demo für den Lebenszyklus einer Aktivität erhalten wir:

```
11 import de.tmahr.android.unterricht.demos.
       logger.Logger;
12
13 public class DemoLifecycle2 extends Activity
14 {
15     private Logger logger;
16     private final static String KEY_LOGGER = "
           KEY_LOGGER_" + DemoLifecycle2.class.
           getSimpleName();
```

Referenz auf einen `Logger` anlegen.

Schlüssel definieren.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLifecycle2.java
(Ausschnitt)

```
18      @Override
19      protected void onCreate(Bundle
            savedInstanceState)
20      {
21          super.onCreate(savedInstanceState);
22          setTitle(this.getClass().getSimpleName
                ());
23          TextView textView = new TextView(this)
                ;
24          setContentView(textView);
25          SharedPreferences p =
                PreferenceManager.
                getDefaultSharedPreferences(this);
26          String logHistory = p.getString(
                KEY_LOGGER, "");
27          logger = new Logger(this.getClass().
                getSimpleName(), textView,
                logHistory);
28          logger.log("onCreate");
29      }
```

`SharedPreferences` öffnen.

Historie der geloggten Nachrichten laden.

Logger instanziieren und Nachrichtenhistorie übergeben.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLifecycle2.java
(Ausschnitt)

```
31      @Override
32      public void onPause()
33      {
34          super.onPause();
35          logger.log("onPause");
36
37          SharedPreferences p =
                PreferenceManager.
                getDefaultSharedPreferences(this);
38          SharedPreferences.Editor editor = p.
                edit();
39
40          if (isFinishing() == true)
41          {
42              logger.log("isFinishing()==true");
43              editor.putString(KEY_LOGGER, "");
44          } else
45          {
46              logger.log("isFinishing()==false")
                    ;
47              editor.putString(KEY_LOGGER,
                    logger.getLoggedText());
48          }
49          editor.commit();
50      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLifecycle2.java (Ausschnitt)

Zum Schreiben der Daten wird ein Editor benötigt.

Falls die Aktivität beendet wird, weil der Benutzer die Android-Zurück-Taste betätigt hat, wird die Nachrichtenhistorie gelöscht, so dass beim nächsten Start der Aktivität der Logger mit einer leeren Nachrichtenhistorie initialisiert wird.

Falls die Aktivität beendet wird, weil das Android-Gerät gekippt wurde, wird die Nachrichtenhistorie gespeichert, so dass beim anschließenden Start der Aktivität der Logger mit einer bisherigen Nachrichtenhistorie initialisiert wird.

Schreibt die editierten Daten auf das Dateisystem.

Inhaltsverzeichnis

Falls Sie diesen Tipp nicht beachten:

Tipp

1. Nehmen Sie beim Aufruf von `onPause` den „schlimmsten“ Fall an: Die Aktivität geht in den Hintergrund und wird irgendwann vom Betriebssystem wegen Ressourcen-Knappheit beendet.
2. Sichern Sie daher alle Programmeinstellungen, beenden Threads, etc.
3. Beim nächsten Aufruf von `onResume` stellen Sie den ursprünglichen Zustand wieder her.

Dann könnten Sie mit dem auf der nächsten Seite beschriebenen Szenario konfrontiert sein:

1. Sie haben Aktivität A gestartet.
2. Aktivität A geht in den Hintergrund.
3. In der `onPause`-Funktion sichern Sie keine Daten.
4. Das Betriebssystem benötigt Ressourcen und beendet Aktivität A.

Dann ist dies die letzte Möglichkeit zur Sicherung der Daten der Aktivität A:

1. Das Betriebssystem ruft im oben genannten Fall die Funktion `onSaveInstanceState(Bundle b)` auf.
2. In dieser Funktion sichern Sie die Daten in dem `Bundle`.
3. Beim nächsten Aufruf der Funktion `onCreate(Bundle b)` wird das `Bundle` übergeben.
4. Aus diesem `Bundle` können Sie die gespeicherten Daten wiederherstellen.

```
11 public class DemoSaveInstance extends Activity
12 {
13     private Logger logger;
14     private String text = "";
15     private int wert;
16     private final static String KEY_TEXT = "
           KEY_text_" + DemoSaveInstance.class.
           getSimpleName();
17     private final static String KEY_WERT = "
           KEY_wert_" + DemoSaveInstance.class.
           getSimpleName();
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSaveInstance.java (Ausschnitt)

Exemplarischer Text, der mittels `Bundle` gespeichert werden soll.

Exemplarischer Wert, der mittels `Bundle` gespeichert werden soll.

Schlüssel zum Speichern des Texts.

Schlüssel zum Speichern des Werts.

```
19      @Override
20      protected void onCreate(Bundle b)
21      {
22          super.onCreate(b);
23          setTitle(this.getClass().getSimpleName());
24          TextView logView = new TextView(this);
25          setContentView(logView);
26          logger = new Logger(this.getClass().
                getSimpleName(), logView, "");
27          logger.log("onCreate");
28
29          if (b != null)
30          {
31              logger.log("savedInstanceState!=null");
32              text = b.getString(KEY_TEXT);
33              wert = b.getInt(KEY_WERT);
34          }
35          else
36          {
37              logger.log("savedInstanceState==null");
38          }
39
40          logger.log("wert=" + wert + ", text=" + text)
                ;
41          text += "+";
42          wert++;
43      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSaveInstance.java
(Ausschnitt)

Diese Referenz zeigt auf eine `Bundle`-Instanz, falls die Aktivität zuvor wegen Ressourcen-Knappheit beendet wurde.

Prüfen, ob das `Bundle` existiert.

Daten auslesen und wiederherstellen.

Daten auslesen und wiederherstellen.

Die Daten würden normalerweise nicht hier verändert werden, sondern an anderer Stelle.

```
45      @Override
46      protected void onSaveInstanceState(Bundle b)
47      {
48          super.onSaveInstanceState(b);
49          logger.log("onSaveInstanceState");
50          b.putString(KEY_TEXT, text);
51          b.putInt(KEY_WERT, wert);
52      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoSaveInstance.java (Ausschnitt)

Diese Funktion wird aufgerufen, falls die Aktivität zuvor wegen Ressourcen-Knappheit beendet wurde.

In diesem `Bundle` können die Daten gesichert werden.

Auch die Elternklasse muss die Gelegenheit haben Daten zu sichern.

Wert unter Schlüssel speichern.

Wert unter Schlüssel speichern.

Inhaltsverzeichnis

Übung: Stimmungsbarometer

1. Erstellen Sie ein neues Android-Projekt.
2. In einer Liste sollen verschiedene Stimmungen angezeigt werden.
3. Ein Benutzer kann immer nur eine Stimmung auswählen.
4. Wählt der Benutzer eine Stimmung aus, so wird diese Stimmung mittels eines Toasts[a] angezeigt.
5. Wird die Applikation beendet und neu gestartet, zeigt die Applikation die letzte ausgewählte Stimmung mittels eines Toasts an.

[a] `http://developer.android.com/reference/android/widget/Toast.html`

Inhaltsverzeichnis

- Das Android-Manifest muss die von der Applikation benötigten Rechte deklarieren.
- Beispiel: Zugriff auf den externen Speicher:
  - `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>`
- Die angeforderten Rechte werden in Abhängigkeit der auf dem Gerät installierten Android Version und der Version des Ziel-SDK unterschiedlich gewährt:

	SDK < 23	SDK ≥ 23
Android < 6.0	alte Rechteverwaltung	alte Rechteverwaltung
Android ≥ 6.0	alte Rechteverwaltung	neue Rechteverwaltung

Android Version $<$ 6.0 oder SDK Version $<$ 23

1. Das Android-Manifest muss die von der Applikation benötigten Rechte deklarieren.
2. Android informiert den Benutzer vor der Installation der Applikation über die angeforderten Rechte.
3. Falls der Benutzer die angeforderten Rechte akzeptiert, wird die Applikation installiert.
4. Sind diese Rechte nicht im Manifest angefordert und versucht die Applikation dennoch auf den externen Speicher zuzugreifen, wirft das Betriebssystem eine Ausnahme.

Android Version $\geq$ 6.0 und SDK Version $\geq$ 23

1. Das Android-Manifest muss die von der Applikation benötigten Rechte deklarieren.
2. Android informiert den Benutzer vor der Installation der Applikation *nicht* über die angeforderten Rechte.
3. Unkritische Rechte (normal permissions) werden automatisch gewährt; alle anderen Rechte (dangerous permissions) muss die Applikation während ihrer Ausführung anfordern.
4. Gewährt der Benutzer das Recht nicht, kann die Applikation darauf reagieren.

Rechte zur Laufzeit anfordern: `https://developer.android.com/training/permissions/requesting.html`

Dangerous permissions (API 23):

Permission Group	Permissions
CALENDAR	READ_CALENDAR, WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG, ADD_VOICEMAIL, USE_SIP, PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous

Normal permissions (API 23):

- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_NETWORK_STATE
- ACCESS_NOTIFICATION_POLICY
- ACCESS_WIFI_STATE
- BLUETOOTH
- BLUETOOTH_ADMIN
- BROADCAST_STICKY
- CHANGE_NETWORK_STATE
- CHANGE_WIFI_MULTICAST_STATE
- CHANGE_WIFI_STATE
- DISABLE_KEYGUARD
- EXPAND_STATUS_BAR
- GET_PACKAGE_SIZE
- INSTALL_SHORTCUT
- INTERNET
- KILL_BACKGROUND_PROCESSES
- MODIFY_AUDIO_SETTINGS
- NFC
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- RECEIVE_BOOT_COMPLETED
- REORDER_TASKS
- REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- REQUEST_INSTALL_PACKAGES
- SET_ALARM
- SET_TIME_ZONE
- SET_WALLPAPER
- SET_WALLPAPER_HINTS
- TRANSMIT_IR
- UNINSTALL_SHORTCUT
- USE_FINGERPRINT
- VIBRATE
- WAKE_LOCK
- WRITE_SYNC_SETTINGS

Inhaltsverzeichnis

- Die im vorangegangene Abschnitt vorgestellten `SharedPreferences` zeigten bereits eine Möglichkeit der Speicherung und Wiederherstellung der Daten einer Aktivität.
- In diesem Kapitel lernen wir drei weitere Möglichkeiten kennen:
  1. Assets: nur lesen
  2. Interner Speicher: lesen und schreiben
  3. Externer Speicher: lesen und schreiben

Inhaltsverzeichnis

- Assets sind Behälter für Daten, die beim Compilieren der Applikation (*.apk) hinzugefügt werden.
- Um Assets in die Applikation einzubinden geht man bei Android-Studio so vor:
  1. Im Project-Browser mit rechter Maustaste auf *app*
  2. New > Folder > Assets Folder
  3. Gegebenenfalls Unterverzeichnisse im Assets-Verzeichnis erstellen
  4. Dateien in Assets speichern.
- Auf Assets kann nur lesend zugegriffen werden.
- Assets bieten sich beispielsweise für die Speicherung von kürzeren Audiodateien an.

```
19 public class DemoAssets extends Activity
20 {
21     private Logger logger;
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState)
25     {
26         super.onCreate(savedInstanceState);
27         setTitle(this.getClass().getSimpleName());
28         TextView textView = new TextView(this);
29         setContentView(textView);
30         logger = new Logger(this.getClass().getSimpleName(),
               textView, "");
31         logger.log("onCreate");
32         auslesen("texte/meintext.txt");
33     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoAssets.java (Ausschnitt)

Inhalt der spezifizierten Quelle auslesen.

```
35      private void auslesen(String assetName)
36      {
37          logger.log("auslesen");
38          AssetManager manager = getAssets();
39          try
40          {
41              InputStream is = manager.open(assetName);
42              String text = textEinlesen3(is);
43              logger.log(text);
44          }
45          catch (IOException e)
46          {
47              logger.log("Kann " + assetName + " nicht öffnen!")
                    ;
48          }
49      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoAssets.java (Ausschnitt)

Inhalt der spezifizierten Quelle auslesen.

Auf den `AssetManager` der Aktivität zugreifen.

Daten über den `AssetManager` öffnen.

Text mit einer der drei Funktionen `textEinlesen1`, `textEinlesen2` oder `textEinlesen3` einlesen.

```
51      private static String textEinlesen1(InputStream is)
52      {
53          Scanner s = new Scanner(is,"UTF8").useDelimiter("\\x04
                ");
54          return s.hasNext() ? s.next() : "";
55      }
56
57      private static String textEinlesen2(InputStream is) throws
            IOException
58      {
59          ByteArrayOutputStream bs = new ByteArrayOutputStream()
                ;
60          byte[] bytes = new byte[4096];
61          int len;
62          while((len=is.read(bytes))>0)
63          {
64              bs.write(bytes,0,len);
65          }
66          return new String(bs.toByteArray(),"UTF8");
67      }
```

Kompakte Text-Einlesefunktion mittels Java-Scanner-Klasse

Steuerzeichen für EOF (end of file)

Diese Einlesevariante liest einen Byte-Strom aus und interpretiert die Bytes als UTF8-Text.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoAssets.java (Ausschnitt)

```
69      private static String textEinlesen3(InputStream is) throws
            IOException
70      {
71          BufferedReader br = new BufferedReader(new
               InputStreamReader(is));
72          StringBuilder sb = new StringBuilder();
73          String line;
74          while((line=br.readLine()) != null)
75          {
76              sb.append(line).append("\n");
77          }
78          return sb.toString();
79      }
```

Diese Einlesefunktion verwendet die Standard-Texteinlesefunktion aus Java.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoAssets.java (Ausschnitt)

Inhaltsverzeichnis

- Auf den externen Speicher kann man lesend und schreibend zugreifen.
- Diese Zugriffsrechte muss man im Android-Manifest geltend machen: siehe Abschnitt Rechteverwaltung

```
<uses-permission android:name="android.
permission.WRITE_EXTERNAL_STORAGE"/>
```

- Mittels des *File Explorers* des *Dalvik Debug Monitor Server (DDMS)* (bei Eclipse) oder *Android Device Monitors* (bei Android-Studio) kann man auf die Daten des externen Speichers zugreifen.

Das folgende Beispiel verwendet die alte Rechteverwaltung: Android Version $<$ 6.0 oder SDK Version $<$ 23

Der Zugriff auf den externen Speicher muss im Manifest eingetragen werden:

```
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE"/>
```

```
20 public class DemoExternerSpeicher extends Activity
21 {
22     private Logger logger;
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState)
26     {
27         super.onCreate(savedInstanceState);
28         setTitle(this.getClass().getSimpleName());
29         TextView textView = new TextView(this);
30         setContentView(textView);
31         logger = new Logger(this.getClass().getSimpleName
               (), textView, "");
32         logger.log("onCreate");
33         aufSpeicherZugreifen();
34     }
```

Diese Funktion schreibt, liest und löscht Daten vom externen Speicher.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoExternerSpeicher.java
(Ausschnitt)

```
36      private void aufSpeicherZugreifen()
37      {
38          String state = Environment.
                getExternalStorageState();
39          if (!Environment.MEDIA_MOUNTED.equals(state))
40          {
41              logger.log("Das Gerät hat leider keinen
                    externen Speicher!");
42          }
43          else
44          {
45              File dir = Environment.
                    getExternalStorageDirectory();
46              logger.log("getExternalStorageDirectory: " +
                    dir.toString());
47              String pfad = dir.getAbsolutePath() + File.
                    separator + "DemoExternerSpeicher.txt";
48              logger.log("getAbsolutePath: " + pfad);
49              schreiben(pfad);
50              lesen(pfad);
51              loeschen(pfad);
52          }
53      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoExternerSpeicher.java (Ausschnitt)

Ermittelt den Zustand des externen Speichers. Der Zustand wird als statischer `String` zurückgeliefert.

Prüft, ob das Gerät einen externen Speicher verfügt.

Ermittelt das Verzeichnis des externen Speichers.

Dateipfad erstellen.

Datei auf externen Speicher schreiben.

Datei von externem Speicher lesen und ausgeben.

Datei löschen.

```
55      private void schreiben(String pfad)
56      {
57          try
58          {
59              BufferedWriter writer = new BufferedWriter(new FileWriter(new
                    File(pfad)));
60              writer.write("Dieser Text soll extern gespeichert werden!");
61              writer.close();
62          }
63          catch (IOException e)
64          {
65              logger.log(e);
66          }
67      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoExternerSpeicher.java (Ausschnitt)

```
69      private void lesen(String pfad)
70      {
71          try
72          {
73              FileInputStream fis = new FileInputStream(pfad);
74              Scanner scanner = new Scanner(fis, "UTF8").useDelimiter("\\
                    x04");
75              logger.log("Gelesener Text: " + (scanner.hasNext() ? scanner.
                    next() : ""));
76              try
77              {
78                  fis.close();
79              }
80              catch (IOException e)
81              {
82                  logger.log(e);
83              }
84          }
85          catch (FileNotFoundException e)
86          {
87              logger.log(e);
88          }
89      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoExternerSpeicher.java (Ausschnitt)

```
91      private void loeschen(String pfad)
92      {
93          File file = new File(pfad);
94          if (!file.delete())
95          {
96              logger.log("Die Datein " + pfad + " kann nicht gelöscht
                    werden.");
97          }
98      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoExternerSpeicher.java (Ausschnitt)

Inhaltsverzeichnis

- Auf den internen Speicher kann man lesend und schreibend zugreifen.
- Für den Zugriff bietet die Klasse `Activity` die Funktionen `openFileOutput` und `openFileInput`.

```
18 public class DemoInternerSpeicher extends Activity
19 {
20   private Logger logger;
21
22     @Override
23     protected void onCreate(Bundle
           savedInstanceState)
24     {
25         super.onCreate(savedInstanceState);
26         setTitle(this.getClass().getSimpleName());
27         TextView textView = new TextView(this);
28         setContentView(textView);
29         logger = new Logger(this.getClass().
             getSimpleName(), textView, "");
30         logger.log("onCreate");
31         aufSpeicherZugreifen();
32     }
```

Diese Funktion schreibt und liest Daten vom externen Speicher.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoInternerSpeicher.java
(Ausschnitt)

```
34      private void aufSpeicherZugreifen()
35      {
36          final String dateiname = "
               DemoInternerSpeicher.txt";
37          final String text = "Dieser Text soll im
               internen Speicher gespeichert werden";
38          logger.log("Zu schreibender Text: " + text
               );
39          schreiben(dateiname,text);
40          lesen(dateiname);
41      }
```

Datei auf internen Speicher schreiben.

Datei von internem Speicher lesen und ausgeben.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoInternerSpeicher.java
(Ausschnitt)

```
43      private void schreiben(String dateiname,
            String text)
44      {
45          try
46          {
47              FileOutputStream fos = openFileOutput(
                    dateiname, Context.MODE_PRIVATE);
48              fos.write(text.getBytes());
49              fos.close();
50          }
51          catch (FileNotFoundException e)
52          {
53              logger.log(e);
54          }
55          catch (IOException e)
56          {
57              logger.log(e);
58          }
59      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoInternerSpeicher.java
(Ausschnitt)

Die Funktion `openFileOutput` erbt unsere Klasse von `ContextWrapper`, welche eine Basisklasse von `Activity` ist.

```
61      private void lesen(String dateiname)
62      {
63          try
64          {
65              FileInputStream fis = openFileInput(
                    dateiname);
66              Scanner scanner = new Scanner(fis,"
                    UTF8").useDelimiter("\\x04");
67              String gelesenerText = scanner.hasNext
                    () ? scanner.next() : "";
68              logger.log("Eingelesener Text: " +
                    gelesenerText);
69              fis.close();
70          }
71          catch (FileNotFoundException e)
72          {
73              logger.log(e);
74          }
75          catch (IOException e)
76          {
77              logger.log(e);
78          }
79      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoInternerSpeicher.java
(Ausschnitt)

Die Funktion `openFileInput` erbt unsere Klasse von `ContextWrapper`, welche eine Basisklasse von `Activity` ist.

Inhaltsverzeichnis

Übung

1. Erstellen Sie ein neues Android-Projekt.
2. Fügen Sie den Assets eine Textdatei hinzu.
3. Beim Starten der Applikation soll ein zufälliger Ausschnitt der Textdatei aus den Assets ausgelesen werden.
4. Der ausgelesene Text soll über den Logger ausgegeben werden.
5. Außerdem soll der Text auf den externen Speicher geschrieben werden.
6. Überprüfen Sie im Android-Device-Monitor, ob der Text auf dem Dateisystem des Geräts gespeichert wurde.

Inhaltsverzeichnis

Inhaltsverzeichnis

Punktdichte (Pixeldichte)

- Definition der eindimensionalen Punktdichte d:

$$d := \frac{N}{l}$$

  mit Anzahl der Punkte N und Länge l.
- Einheit: $[d] = \frac{1}{\mathrm{m}}$
  - Einheit dpi (dots per inch[a])
    - $\mathrm{dpi} := \frac{1}{\mathrm{in}} = \frac{1}{0{,}0254\ \mathrm{m}} = 39{,}37 \frac{1}{\mathrm{m}}$
  - Einheit dpc (dots per centimeter[b])
    - $\mathrm{dpc} := \frac{1}{\mathrm{cm}} = 100 \frac{1}{\mathrm{m}} = 2{,}54\ \frac{1}{\mathrm{in}} = 2{,}54\ \mathrm{dpi}$

[a] Anzahl Punkte pro Zoll
[b] Anzahl Punkte pro Zentimeter

Die Android-Geräte werden in verschiedene Pixeldichteklassen eingeteilt:

Name	Abkürzung	Pixeldichte	Skalierung
Low-DPI	LDPI	≈ 120	0,75
Medium-DPI	MDPI	≈ 160	1
High-DPI	HDPI	≈ 240	1,5
Extra-high-DPI	XHDPI	≈ 320	2
Extra-extra-high-DPI	XXHDPI	≈ 480	3
Extra-extra-extra-high-DPI	XXXHDPI	≈ 640	4

- Die physikalische Größe eines Pixels l_1 ist von der Pixeldichte d des Gerätes abhängig, auf dem das Pixel dargestellt wird: $l_1 = \frac{1}{d}$
- Um für die Entwickler die Unterstützung unterschiedlich großer Geräte zu erleichtern, wird eine künstliche Pixelgröße l' eingeführt, die von der Pixeldichte d des Gerätes unabhängig ist.
- Hierzu wird die Pixeldichte d auf die Pixeldichte $d^\star = 160\mathrm{dpi}$ des ersten Android Gerätes[1] normiert.

[1] T-Mobile G1

Dichte-unabhängige Pixelgröße (density-independent pixels)

- Definition der Dichte-unabhängigen Pixelgröße l':

$$l' := \frac{d^\star}{d} = \frac{160\ \mathrm{dpi}}{d} = \frac{160 \cdot l}{N} \cdot \frac{1}{\mathrm{in}} = 160 \cdot l_1 \cdot \frac{1}{\mathrm{in}}$$

- Einheit: $[l'] = 1$
  - Obwohl l' dimensionslos ist, verwendet man für die Dichte-unabhängige Pixelgröße die künstlichen Einheiten
    - $\mathrm{dp} := 1$ oder
    - $\mathrm{dip} := 1$
  - dp und dip stehen für density-independent pixels und werden gesprochen als „dip“.

Beispiel 1:

Ein Gerät hat die Bildschirmbreite l und die Pixeldichte d. Daraus lassen sich die Anzahl der Pixel $N = l \cdot d$ und die Bildschirmbreite $l' = l \cdot 160\ \mathrm{dpi}$ in dp berechnen:

Gerät Nr.	l	d	N	l'
1	$1{,}5''$	$120\ \mathrm{dpi}$	180	$240\ \mathrm{dp}$
2	$1{,}5''$	$160\ \mathrm{dpi}$	240	$240\ \mathrm{dp}$
3	$1{,}5''$	$240\ \mathrm{dpi}$	360	$240\ \mathrm{dp}$

Beispiel 2:

- Ein Icon besteht aus 32×32 Pixeln.
- Die Punktdichte auf dem Gerät beträgt $d = 320\ \mathrm{dpi}$.

Daraus folgt:

- Dichte-unabhängige Breite des Icons: $l' = 32 \cdot \frac{160\ \mathrm{dpi}}{320\ \mathrm{dpi}}\ \mathrm{dp} = 16\ \mathrm{dp}$
- Das Icon hat die Größe $16\ \mathrm{dp} \times 16\ \mathrm{dp}$.

- Häufig verwendete Dichte-unabhängige Pixelgrößen:

dp	mm	in
4	0,635	0,025
16	2,540	0,100
48	7,620	0,300
72	11,430	0,450

- Online-Umrechnung zwischen verschiedenen Größen: Android Pixel Calculator
  `http://angrytools.com/android/pixelcalc/`.

Android-Geräte werden in verschiedene Bildschirmgrößenklassen eingeteilt:

Name	Mindestgröße
Small	$426\ \mathrm{dp} \times 320\ \mathrm{dp}$
Normal	$470\ \mathrm{dp} \times 320\ \mathrm{dp}$
Large	$640\ \mathrm{dp} \times 480\ \mathrm{dp}$
Extra-large	$960\ \mathrm{dp} \times 720\ \mathrm{dp}$

Skalenunabhängige Pixelgröße (scale-independent pixels)

- Die skalenunabhängige Pixelgröße s' ist die Dichte-unabhängige Pixelgröße l' multipliziert mit der vom Benutzer eingestellten Skalierung f der **Schriftgröße**:

$$s' := l' \cdot f$$

- Einheit: $[s'] = 1$
- Obwohl s' dimensionslos ist, verwendet man für die skalenunabhängige Pixelgröße die künstliche Einheit sp.

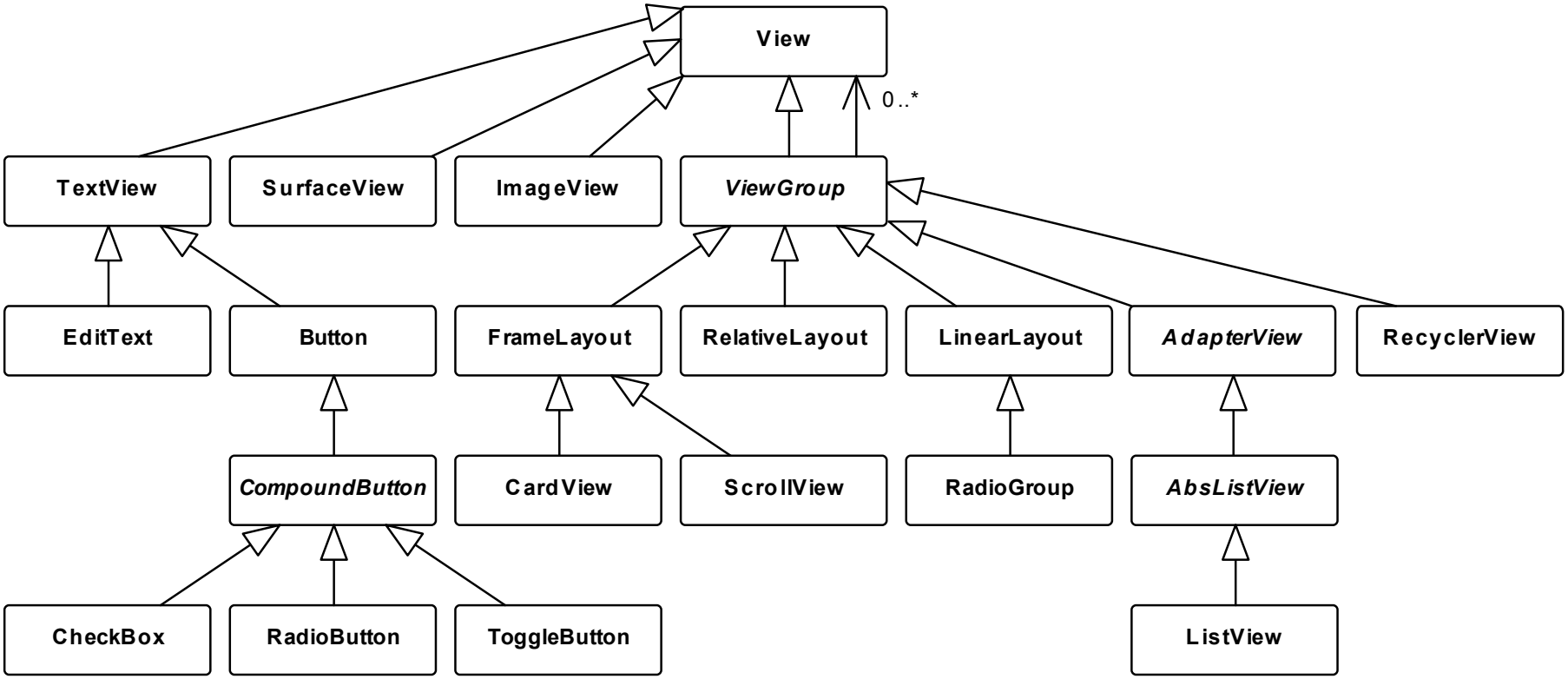
Beispiel:

- Schriftgröße: $s' = 14\ \mathrm{sp}$
- Pixeldichte XHDPI (Skalierung $\sigma = 2$)
- Vom Benutzer eingestellte Schriftskalierung: $f = 125\%$
- Schriftgröße in Pixeln: $s = s' \cdot \sigma \cdot f = 14 \cdot 2 \cdot 1{,}25 = 35$

Inhaltsverzeichnis

- Eine `View` verwaltet die Darstellung eines rechteckigen Gebiets auf dem Bildschirm.
- Eine `View` nimmt Benutzerinteraktionen entgegen.
- `http://developer.android.com/reference/android/view/View.html`

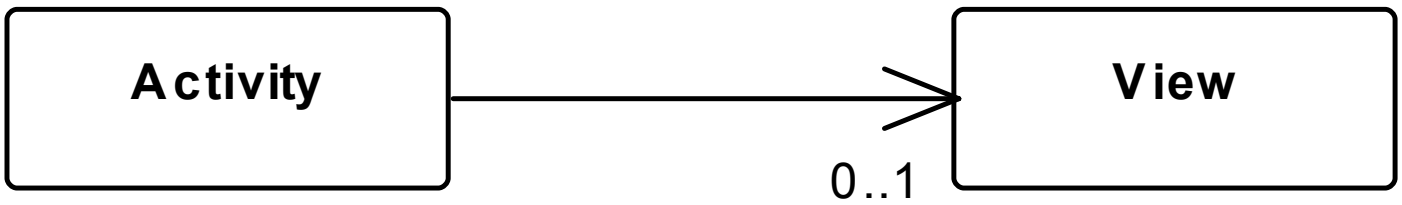
Ausschnitt aus der `View`-Klassen-Hierarchie



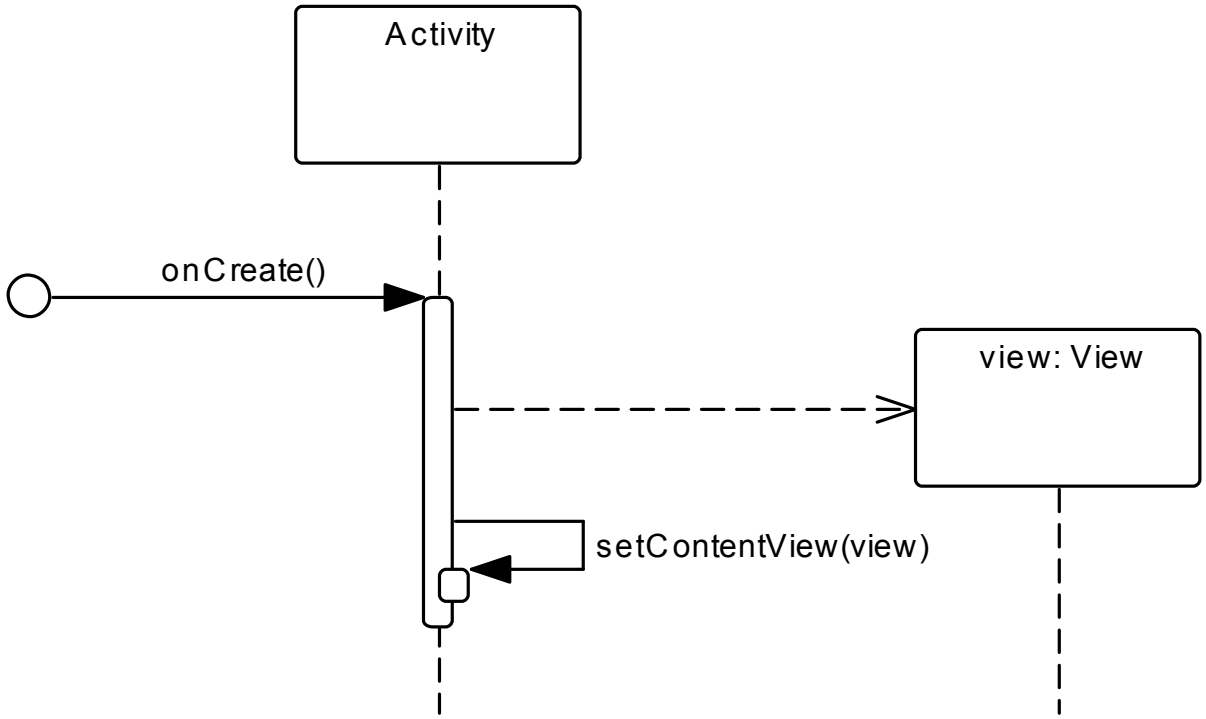
Abstrakte Klassen:

- `AbsAdapterView` stellt Listen dar.
- `AdapterView` stellt (viele) Daten dar, die über einen Adapter zur Verfügung gestellt werden.
- `CompoundButton` ist ein Schalter mit 2 Zuständen.
- `ViewGroup` enthält weitere View-Elemente.

Einer Aktivität (`Activity`) wird genau eine Ansicht (`View`) zugeordnet:



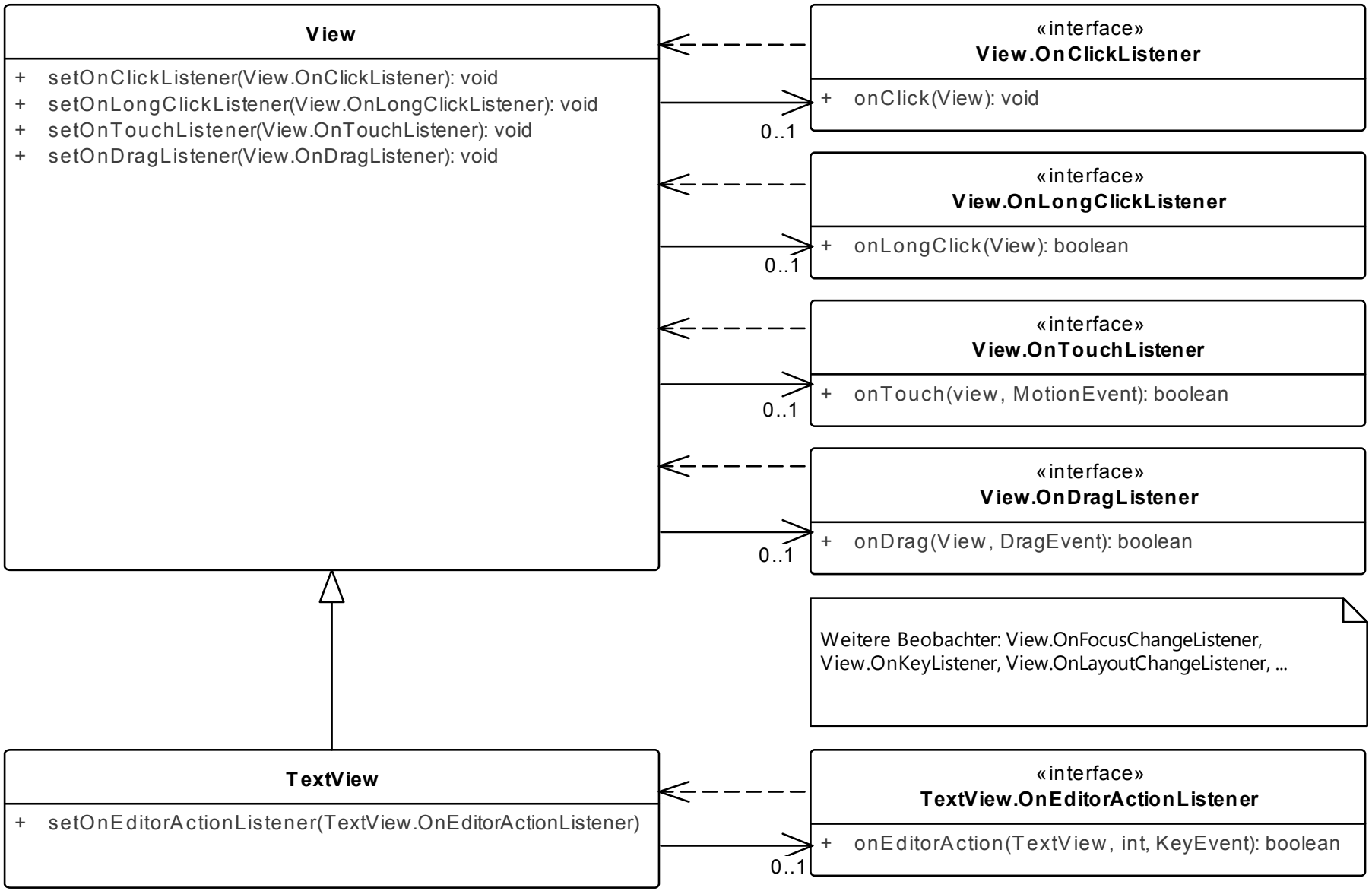
Die Verbindung zwischen Ansicht und Aktivität wird in der `onCreate`-Funktion der Aktivität hergestellt:



Anstelle der Funktion `setContentView(View view)` kann man auch die überladene Funktion `setContentView(int layoutResId)` verwenden, bei der eine in einer XML-Layout-Datei definierte Ansicht instanziiert wird.

Inhaltsverzeichnis

- Ein Anwender interagiert i.d.R. mit der Applikation über die `View`-Objekte.
- Ein `View`-Objekt meldet ein Interaktionsereignis über das Beobachtermuster ▸ siehe Abschnitt Beobachtermuster an seine Beobachter.
- Die Beobachter sind Interfaces, die in den jeweiligen `View`-Klassen eingebettet sind.
- Beispiele siehe nächste Seite:


View
+ setOnClickListener(View.OnClickListener): void
+ setOnLongClickListener(View.OnLongClickListener): void
+ setOnTouchListener(View.OnTouchListener): void
+ setOnDragListener(View.OnDragListener): void
«interface»
View.OnClickListener
+ onClick(View): void
0..1
«interface»
View.OnLongClickListener
+ onLongClick(View): boolean
0..1
«interface»
View.OnTouchListener
+ onTouch(view, MotionEvent): boolean
0..1
«interface»
View.OnDragListener
+ onDrag(View, DragEvent): boolean
0..1
Weitere Beobachter: View.OnFocusChangeListener,
View.OnKeyListener, View.OnLayoutChangeListener, ...
TextView
+ setOnEditorActionListener(TextView.OnEditorActionListener)
«interface»
TextView.OnEditorActionListener
+ onEditorAction(TextView, int, KeyEvent): boolean
0..1

So kann eine Aktivität auf ein Benutzerereignis reagieren:

- Entweder implementiert die Aktivität das dem Ereignis entsprechende Interface und registriert sich als Beobachter bei der `View`.
- Oder man implementiert das Interface mittels einer anonymen Klasse und registriert die Instanz als Beobachter bei der `View`.

Im Falle eines `View.OnClickListener` kann man auch so vorgehen:

- Die Aktivität stellt eine öffentliche Funktion `public void aufKlickReagiern(View v)` zur Verfügung.
- Im XML-Layout weist man dem `View`-Element diese Funktion als Reaktion auf ein Klick-Ereignis mittels `android:onClick="aufKlickReagiern"` zu.

Inhaltsverzeichnis

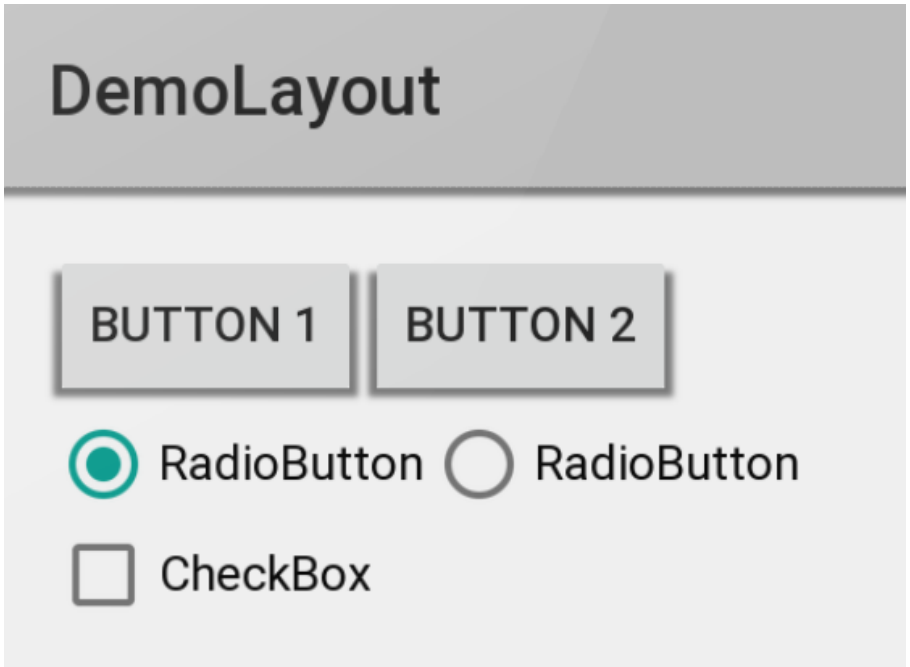
- Bisher haben wir die `View`-Objekte direkt im Java-Code erstellt.
- Für komplexere Layouts ist dies jedoch unpraktisch, vor allem wenn das Layout von der Orientierung des Gerätes, der Bildschirmgröße, der Bildschirmauflösung oder weiteren Faktoren abhängig ist.
- Alternativ kann das Layout in einer XML-Datei festgelegt werden.
- Die in XML definierten `View`-Elemente werden als Java-Objekte angelegt und können im Java-Programm referenziert werden.

- Das in XML definierte Layout besteht in der Regel aus einer `ViewGroup`, die weitere `View`-Elemente enthält.
- Diese `ViewGroup` wird (einschließlich ihrer enthaltenen `Views`) instanziiert und der Aktivität mittels der bekannten Funktion `Activity.setContentView(View view)` zugewiesen.
- Dieser direkte Funktionsaufruf bleibt jetzt jedoch dem Programmierer verborgen.
- Denn er verwendet jetzt die überladene Funktion `Activity.setContentView(int layoutResId)`.
- Der `int`-Wert `layoutResId` wird automatisch generiert und identifiziert eine XML-Layout-Datei.
- Beim Ausführen der Funktion `Activity.setContentView(int layoutResId)` werden alle `View`-Objekte der in XML definierten Elemente instanziiert und schließlich die einbettende `ViewGroup` der Aktivität zugewiesen.

- Die `View`-Elemente erhalten in XML eine eindeutige Kennung.
- Dadurch kann man
  - innerhalb der XML-Datei auf die einzelnen `View`-Elemente Bezug nehmen und
  - innerhalb des Java-Codes die `View`-Objekte referenzieren.
- Definition einer Kennung für ein `View`-Element in XML:
  `android:id="@+id/beispiel"`
  - `@` referenziert eine Ressource.
  - `+` erstellt eine neue Ressource. (Das Symbol stört aber nicht, wenn man sich nur auf eine bestehende Ressource bezieht.)
  - `id` markiert den Typ der Ressource.
  - `beispiel` ist der Name der Ressource.
- In Java kann man über `findViewById(beispiel)` auf das `View`-Objekt zugreifen.

- Das Layout für das nächste Demo ist in der Datei *activity_demo_layout.xml* definiert.
- Die zugehörige Kennung wird automatisch generiert: `public static final int R.layout.activity_demo_layout`.
- Die Klasse `layout` ist in die Klasse `R` eingebettet.

Benutzeroberfläche der Aktivität `DemoLayout`:



Komposition des Layouts:

- `RelativeLayout`
  1. `Button`
  2. `Button`
  3. `RadioGroup`
     1. `RadioButton`
     2. `RadioButton`
  4. `CheckBox`

Das Layout ist in der auf den nächsten Seiten gezeigten XML-Datei definiert:

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
2                  xmlns:tools="http://schemas.android.com/tools"
3                  android:layout_width="match_parent"
4                  android:layout_height="match_parent"
5                  android:paddingBottom="@dimen/
                       activity_vertical_margin"
6                  android:paddingLeft="@dimen/
                       activity_horizontal_margin"
7                  android:paddingRight="@dimen/
                       activity_horizontal_margin"
8                  android:paddingTop="@dimen/
                       activity_vertical_margin"
9                  tools:context="de.tmahr.android.unterricht.demos
                       .DemoLayout">
10
11     <Button
12         android:id="@+id/buttonZeigeFragment1"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:layout_alignParentLeft="true"
16         android:layout_alignParentTop="true"
17         android:text="Button 1"/>
18
19     <Button
20         android:id="@+id/buttonZeigeFragment2"
21         android:layout_width="wrap_content"
22         android:layout_height="wrap_content"
23         android:layout_alignBottom="@+id/buttonZeigeFragment1"
24         android:layout_toRightOf="@+id/buttonZeigeFragment1"
25         android:text="Button 2"/>
```

Demos/app/src/main/res/layout/activity_demo_layout.xml (Ausschnitt)

Ein `RelativeLayout` ist eine `View`, die weitere `View`-Objekte enthält und diese relativ zueinander ausrichtet.

Der erste Schalter wird am linken oberen Rand des einbettenden `View`-Elements ausgerichtet (`alignParent...`) und die Höhe und Breite so gewählt, dass der Text des Schalters angezeigt werden kann (`wrap_content`).

Der zweite Schalter wird am ersten Schalter ausgerichtet (`alignBottom`, `toRightOf`).

```
27      <RadioGroup
28          android:id="@+id/radioGroup1"
29          android:layout_width="wrap_content"
30          android:layout_height="wrap_content"
31          android:layout_alignParentLeft="true"
32          android:layout_below="@+id/buttonZeigeFragment1"
33          android:orientation="horizontal">
34
35          <RadioButton
36              android:id="@+id/radio1"
37              android:layout_width="wrap_content"
38              android:layout_height="wrap_content"
39              android:checked="true"
40              android:text="RadioButton"/>
41
42          <RadioButton
43              android:id="@+id/radio2"
44              android:layout_width="wrap_content"
45              android:layout_height="wrap_content"
46              android:text="RadioButton"/>
47
48      </RadioGroup>
```

Demos/app/src/main/res/layout/activity_demo_layout.xml (Ausschnitt)

```
50      <CheckBox
51          android:id="@+id/checkBox1"
52          android:layout_width="wrap_content"
53          android:layout_height="wrap_content"
54          android:layout_alignParentLeft="true"
55          android:layout_below="@+id/radioGroup1"
56          android:text="CheckBox"/>
57
58      <ScrollView
59          android:id="@+id/scrollView1"
60          android:layout_width="wrap_content"
61          android:layout_height="wrap_content"
62          android:layout_alignParentBottom="true"
63          android:layout_below="@id/checkBox1">
64
65          <LinearLayout
66              android:layout_width="match_parent"
67              android:layout_height="match_parent"
68              android:orientation="vertical">
69
70              <TextView
71                  android:id="@+id/textViewLogger"
72                  android:layout_width="wrap_content"
73                  android:layout_height="wrap_content"/>
74          </LinearLayout>
75      </ScrollView>
76
77  </RelativeLayout>
```

Demos/app/src/main/res/layout/activity_demo_layout.xml (Ausschnitt)

Für die XML-Tags `android:layout_width` und `android:layout_height` der `View`-Elemente können diese Werte verwendet werden:

- `wrap_content`: Die Breite bzw. Höhe orientiert sich am darzustellenden Inhalt des `View`-Elements.
- `match_parent`: Die Breite bzw. Höhe orientiert sich am übergeordnetem, einbettendem `View`-Element.
- `...dp`: Feste Größe in der Einheit dp (density-independent pixels, siehe `http://developer.android.com/guide/practices/screens_support.html`). Nachteil: Auf unterschiedlichen Geräten und bei unterschiedlichen Orientierungen des Geräts wird möglicherweise nicht das gesamte Layout dargestellt. Eine Lösung könnten hier das weiter unten gezeigte `LinearLayout` mit Wichtung sein.

```
14 public class DemoLayout extends Activity
15 {
16     private Logger logger;
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState)
20     {
21         super.onCreate(savedInstanceState);
22         setTitle(this.getClass().getSimpleName());
23         setContentView(R.layout.activity_demo_layout);
24         TextView textViewLogger = (TextView)findViewById(
               R.id.textViewLogger);
25         logger = new Logger(this.getClass().getSimpleName
               (), textViewLogger, "");
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLayout.java (Ausschnitt)

Der Aktivität wird über die Kennung `R.layout.activity_demo_layout` das in der Datei *activity_demo_layout.xml* definierte Layout zugeordnet. Dabei werden alle enthaltenen `View`-Elemente instanziiert.

Auf die jetzt instanziierten `View`-Elemente kann man mit dieser Funktion und über die in XML definierte Kennung `textViewLogger` für diese `TextView`-Instanz zugreifen.

```
26          final Button button1 = (Button)findViewById(R.id.
               buttonZeigeFragment1);
27          button1.setOnClickListener(new View.
               OnClickListener()
28          {
29              @Override
30              public void onClick(View v)
31              {
32                  logger.log("View.OnClickListener().
                       onClick: Schalter \"" + button1.
                       getText() + "\" gewählt");
33              }
34          });
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLayout.java (Ausschnitt)

Auf die jetzt instanziierten `View`-Elemente kann man mit dieser Funktion und über die in XML definierte Kennung `button1` für diese `Button`-Instanz zugreifen.

Im Gegensatz zu früheren Beispielen implementiert die Aktivität jetzt nicht mehr die `OnClickListener`-Schnittstelle. Um dennoch auf den Tastendruck reagieren zu können, wird eine anonyme Implementierung der Schnittstelle instanziiert und dem Schalter als Beobachter hinzugefügt.

Da `onClick` asynchron abläuft und dabei die Referenz `button1` verwendet, muss sichergestellt sein, dass die Referenz gültig und unverändert ist. Daher ist die Referenz als **`final`** deklariert.

```
35          final Button button2 = (Button)findViewById(R.id.
               buttonZeigeFragment2);
36          button2.setOnClickListener(new View.
               OnClickListener()
37          {
38              @Override
39              public void onClick(View v)
40              {
41                  logger.log("View.OnClickListener().
                       onClick: Schalter \"" + button2.
                       getText() + "\" gewählt");
42              }
43          });
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLayout.java (Ausschnitt)

Auf die jetzt instanziierten `View`-Elemente kann man mit dieser Funktion und über die in XML definierte Kennung `button2` für diese `Button`-Instanz zugreifen.

Im Gegensatz zu früheren Beispielen implementiert die Aktivität jetzt nicht mehr die `OnClickListener`-Schnittstelle. Um dennoch auf den Tastendruck reagieren zu können, wird eine anonyme Implementierung der Schnittstelle instanziiert und dem Schalter als Beobachter hinzugefügt.

```
44          final CheckBox checkBox = (CheckBox)findViewById(
               R.id.checkBox1);
45          checkBox.setOnClickListener(new View.
               OnClickListener()
46          {
47              @Override
48              public void onClick(View v)
49              {
50                  StringBuilder sb = new StringBuilder();
51                  sb.append("View.OnClickListener().onClick
                       : CheckBox \"").append(checkBox.
                       getText()).append("\" ");
52                  if(checkBox.isChecked())
53                  {
54                      sb.append("gewählt");
55                  }
56                  else
57                  {
58                      sb.append("nicht gewählt");
59                  }
60                  logger.log(sb.toString());
61              }
62          });
63          logger.log("onCreate");
64      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLayout.java (Ausschnitt)

Auf die jetzt instanziierten `View`-Elemente kann man mit dieser Funktion und über die in XML definierte Kennung `checkBox1` für diese `CheckBox`-Instanz zugreifen.

Im Gegensatz zu früheren Beispielen implementiert die Aktivität jetzt nicht mehr die `OnClickListener`-Schnittstelle. Um dennoch auf den Tastendruck reagieren zu können, wird eine anonyme Implementierung der Schnittstelle instanziiert und dem Schalter als Beobachter hinzugefügt.

Inhaltsverzeichnis

- In einem LinearLayout werden die `View`-Elemente linear in horizontaler oder vertikaler Richtung angezeigt.
- Die Breiten bzw. Höhen der Elemente können zueinander gewichtet mittels des XML-Tags `android:layout_weight` festgelegt werden. Bei vertikaler Ausrichtung setzt man dann häufig den Wert `android:layout_height = "0dp"`, bei horizontaler Ausrichtung entsprechend `android:layout_width = "0dp"`.
- Der freie Platz wird unter den mit `android:layout_weight` markierten `View`-Elementen entsprechend deren Wichtungen verteilt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/
       apk/res/android"
3                android:layout_width="match_parent"
4                android:layout_height="match_parent"
5                android:orientation="vertical">
6
7      <Button
8          android:layout_width="wrap_content"
9          android:layout_height="wrap_content"
10         android:text="w: wrap, h: wrap"
11         android:id="@+id/buttonZeigeFragment1"/>
12
13     <Button
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="w: wrap, h: wrap, gravity: right"
17         android:id="@+id/buttonZeigeFragment2"
18         android:layout_gravity="right"/>
19
20     <Button
21         android:layout_width="wrap_content"
22         android:layout_height="wrap_content"
23         android:text="w: wrap, h: wrap, gravity: center"
24         android:id="@+id/button3"
25         android:layout_gravity="center_horizontal"/>
```

Dieses Ein `LinearLayout` ordnet die `View`-Elemente vertikal an.

Breite uns Höhe sind so groß, dass der Inhalt (Text) dargestellt werden kann.

Wie der vorangegange Schalter, nur mir Ausrichtung auf der rechten Seite.

Mittige Ausrichtung.

Demos/app/src/main/res/layout/activity_demo_linear_layout.xml (Ausschnitt)

```
27      <Button
28          android:layout_width="match_parent"
29          android:layout_height="0dp"
30          android:layout_weight="1"
31          android:text="w: match, h: 0dp, weight=1"
32          android:id="@+id/button4" />
33
34      <Button
35          android:layout_width="match_parent"
36          android:layout_height="wrap_content"
37          android:text="w: match, h: wrap, gravity: center"
38          android:id="@+id/button5"
39          android:layout_gravity="center_horizontal" />
40
41      <Button
42          android:layout_width="match_parent"
43          android:layout_height="0dp"
44          android:layout_weight="2"
45          android:text="w: match, h: 0dp, weight=2"
46          android:id="@+id/button6"
47          android:layout_gravity="center_horizontal" />
48
49  </LinearLayout>
```

Demos/app/src/main/res/layout/activity_demo_linear_layout.xml (Ausschnitt)

Die Höhe des 4. Schalters wird mit dem Faktor 1 gewichtet.

Die Breite des Schalters orientiert sich an der Breite des einbettenden Elements.

Die Höhe des 6. Schalters wird mit dem Faktor 2 gewichtet. Die verfügbare Höhe des einbettenden Elements wird nach Abzug der Höhen der Schalter 1, 2, 3 und 5 auf die Schalter 4 und 6 im Verhältnis 1:2 aufgeteilt.

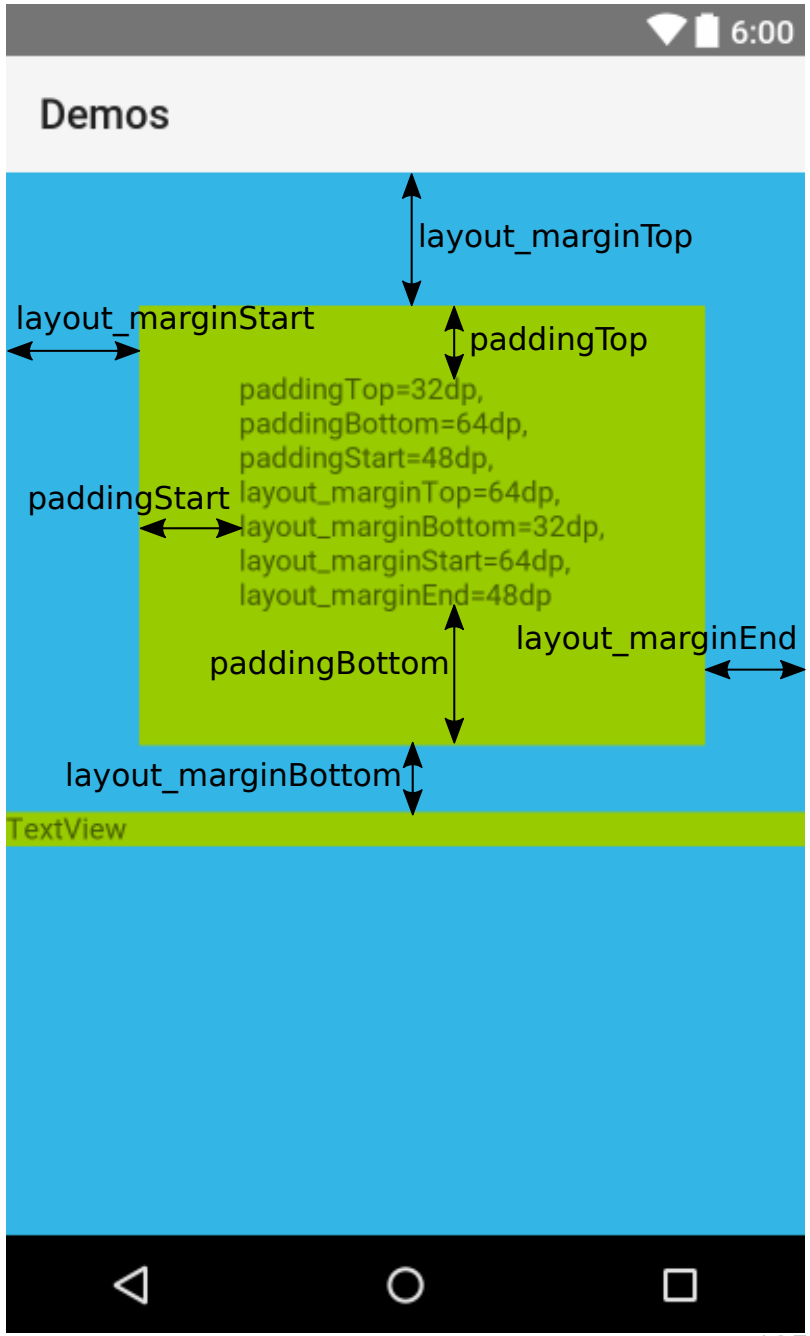
Inhaltsverzeichnis

Für eine `View` können zwei verschiedene Abstandsmaße definiert werden:

- Padding ist ein Abstand, der innerhalb des `View`-Rechtecks liegt.
- Margin ist ein Abstand, der um das `View`-Rechteck liegt.

In der Abbildung ist dargestellt:

- blauer Hintergrund eines `LinearLayouts`
- zwei grüne `TextViews`



XML-Layout:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                android:layout_width="match_parent"
4                android:layout_height="match_parent"
5                android:background="@android:color/holo_blue_light"
6                android:orientation="vertical">
7      <TextView
8          android:id="@+id/textView1"
9          android:text="paddingTop=32dp, paddingBottom=64dp, paddingStart=48dp,
               layout_marginTop=64dp, layout_marginBottom=32dp, layout_marginStart=64dp,
               layout_marginEnd=48dp "
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:background="@android:color/holo_green_light"
13         android:paddingBottom="64dp"
14         android:paddingTop="32dp"
15         android:paddingStart="48dp"
16         android:layout_marginTop="64dp"
17         android:layout_marginStart="64dp"
18         android:layout_marginEnd="48dp"
19         android:layout_marginBottom="32dp"/>
20
21     <TextView
22         android:id="@+id/textView2"
23         android:text="TextView"
24         android:layout_width="match_parent"
25         android:layout_height="wrap_content"
26         android:background="@android:color/holo_green_light"/>
27 </LinearLayout>
```

Demos/app/src/main/res/layout/activity_demo_paddings_margins.xml

Inhaltsverzeichnis

Übung: Speed-Reader 1

Erstellen Sie eine Applikation mit den folgenden Eigenschaften:

1. Am oberen Ende des Layouts befindet sich ein Schalter S1.
2. Beim Drücken des Schalters S1 wird ein Text aus den Assets geladen.
3. Unterhalb des Schalters S1 befindet sich ein mehrzeiliges, vertikal rollbares Textfenster T1, in dem der geladene Text angezeigt wird.
4. Unterhalb des Textfensters T1 befindet sich ein Schalter S2.
5. Beim Drücken des Schalters S2 wird der Text des Textfensters T1 gelöscht.
6. Unterhalb des Schalters S2 befindet sich ein einzeiliges Textfenster T2 zum Anzeigen einzelner Worte des geladenen Textes.

Fortsetzung

Speed-Reader 1 *Fortsetzung*

7. Die Worte werden mittig in T2 angezeigt.
8. Unterhalb des Textfensters T2 befindet sich Schalter S3.
9. Beim ersten Drücken des Schalters S3 wird das erste Wort des geladenen Textes im Textfenster T2 angezeigt. Bei nächsten Drücken wird das nächste Wort des Textes angzeigt, usw.
10. Wird der geladene Text mit Schalter S2 gelöscht, muss auch der Inhalt in T2 gelöscht werden.
11. Die Schalter S1, S2, S3 müssen treffend beschriftet werden.

Inhaltsverzeichnis

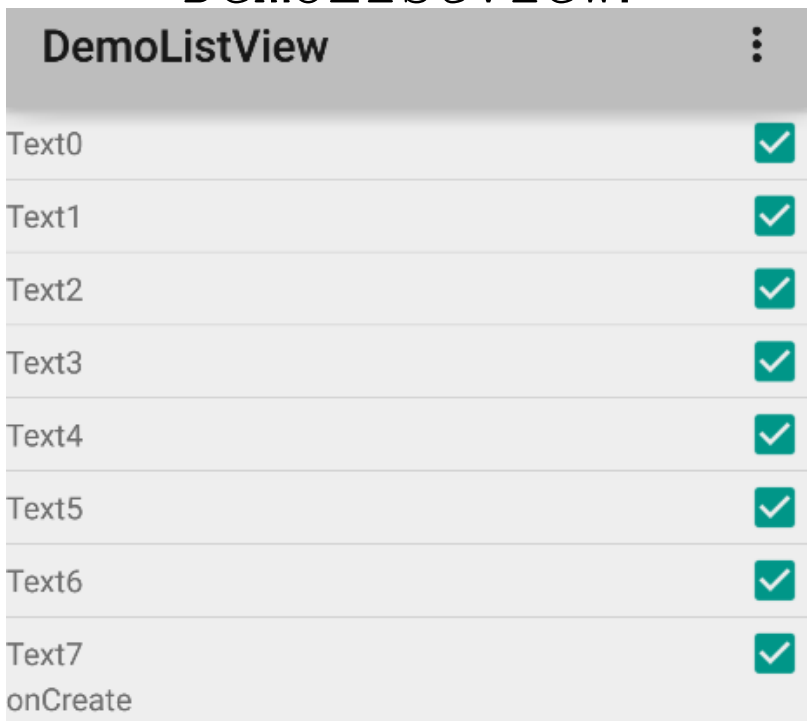
- In einem vorangegangene Kapitel haben wir bereits eine einfache `ListView` verwendet, um Zeichenketten darzustellen.
- Jetzt wollen wir in der Liste komplexere Daten darstellen.
- Dafür müssen wir
  - einen eigenen Adapter definieren
  - das Layout einer Listenzeile definieren

In der Liste darzustellende Daten

```
108     class Daten
109     {
110         String text;
111         boolean wert;
112         Daten(String text, boolean wert
                )
113         {
114             this.text = text;
115             this.wert = wert;
116         }
117     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java
(Ausschnitt)

Benutzeroberfläche der Aktivität `DemoListView`:



XML-Layout der Aktivität:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                xmlns:tools="http://schemas.android.com/tools"
4                android:layout_width="match_parent"
5                android:layout_height="match_parent"
6                android:orientation="vertical"
7                tools:context="de.tmahr.android.unterricht.demos.DemoListView">
8
9      <ListView
10         android:id="@+id/listView1"
11         android:layout_width="match_parent"
12         android:layout_height="0dip"
13         android:layout_weight="1">
14     </ListView>
```

Demos/app/src/main/res/layout/activity_demo_list_view.xml (Ausschnitt)

```
16      <ScrollView
17          android:id="@+id/scrollView1"
18          android:layout_width="match_parent"
19          android:layout_height="0dip"
20          android:layout_weight="1">
21
22          <LinearLayout
23              android:layout_width="match_parent"
24              android:layout_height="wrap_content"
25              android:orientation="vertical">
26
27              <TextView
28                  android:id="@+id/textViewDemoListViewLogger"
29                  android:layout_width="match_parent"
30                  android:layout_height="match_parent"
31                  android:text="TextView"/>
32
33          </LinearLayout>
34      </ScrollView>
35
36  </LinearLayout>
```

Demos/app/src/main/res/layout/activity_demo_list_view.xml (Ausschnitt)

XML-Layout einer Listenzeile:

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

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"/>

    <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"/>

</RelativeLayout>
```

Demos/app/src/main/res/layout/activity_demo_list_view_item.xml

Zugehöriger Adapter:

```
51      class MeinAdapter extends BaseAdapter
52      {
53          private Context context;
54          private ArrayList<Daten> daten;
55
56          MeinAdapter(Context context, ArrayList<Daten> daten)
57          {
58              this.context = context;
59              this.daten = daten;
60          }
61
62          @Override
63          public int getCount()
64          {
65              return daten.size();
66          }
67
68          @Override
69          public Object getItem(int position)
70          {
71              return daten.get(position);
72          }
73
74          @Override
75          public long getItemId(int position)
76          {
77              return position;
78          }
```

Der Adapter kennt die darzustellenden Daten.

Liefert die Anzahl der Daten (Zeilen der Liste) zurück.

Liefert die Daten an einer bestimmten Position zurück.

Als eindeutige Kennung einer bestimmten Zeile kann man die Zeilennummer zurückliefern.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java (Ausschnitt)

```
80          @Override
81          public View getView(int position, View
               convertView, ViewGroup parent)
82          {
83              final Daten d = daten.get(position);
84
85              LayoutInflater inflater = (LayoutInflater)
                   context.getSystemService(Context.
                   LAYOUT_INFLATER_SERVICE);
86              View view = inflater.inflate(R.layout.
                   activity_demo_list_view_item, null);
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java (Ausschnitt)

Liefert das `View`-Objekt zurück, das eine bestimmte Zeile der Liste darstellt.

Zur Umwandlung einer Layout-Kennung in ein Layout mit instanziierten `View`-Objekten benötigt man einen `LayoutInflater`. (Beim Erzeugen der `View`-Objekte einer Aktivität in der `onCreate`-Methode, kümmert sich die Methode `setContentView` darum.)

Der `LayoutInflater` instanziiert gemäß des in der Layout-Datei *activity_demo_list_view_item.xml* definierten Layouts die `View`-Objekte, die zur Darstellung einer Zeile der Liste benötigt werden und liefert das gesamte Layout, hier ein `RelativeLayout`-Objekt, bestehend aus weiteren `View`-Objekten zurück (`RelativeLayout` erweitert `View`).

```
87              TextView textView = (TextView) view.
                   findViewById(R.id.textView1);
88              textView.setText(d.text);
89
90              final CheckBox checkBox = (CheckBox) view.
                   findViewById(R.id.checkBox1);
91              checkBox.setChecked(d.wert);
92              checkBox.setOnClickListener(new View.
                   OnClickListener()
93              {
94                  @Override
95                  public void onClick(View v)
96                  {
97                      d.wert = checkBox.isChecked();
98                      StringBuilder sb = new StringBuilder
                           ();
99                      sb.append(d.text).append(d.wert ? "
                           aktiviert" : " deaktiviert");
100                     logger.log(sb.toString());
101                 }
102             });
103
104             return view;
105         }
106     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java (Ausschnitt)

`view` referenziert hier das `RelativeLayout`-Objekt, das weitere `View`-Objekte enthält. Mittels der Funktion `findViewById` kann man über die im XML-Layout festgelegten Kennungen auf die einzelnen `View`-Objekte zugreifen.

Aktivität:

```
22 public class DemoListView extends Activity
23 {
24     private Logger logger;
25     private ListView listView;
26     private static final String KEY_LISTVIEW_SCROLL_POSITION =
           DemoListView.class.getSimpleName() + "_SCROLL_POSITION
          ";
27
28     @Override
29     protected void onCreate(Bundle savedInstanceState)
30     {
31         super.onCreate(savedInstanceState);
32         setTitle(this.getClass().getSimpleName());
33         setContentView(R.layout.activity_demo_list_view);
34         logger = new Logger(this.getClass().getSimpleName(), (
              TextView)findViewById(R.id.
              textViewDemoListViewLogger), "");
35         logger.log("onCreate");
36         listViewInitialisieren();
37     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java (Ausschnitt)

Unter diesem Schlüssel wird die Position der `SrollView` in den `SharedPreferences` gespeichert, so dass beim nächsten Start der Aktivität die letzte Position angezeigt wird.

Der Aktivität wird über die Kennung `R.layout.activity_demo_list_view` das in der Datei *activity_demo_list_view.xml* definierte Layout zugeordnet. Dabei werden alle enthaltenen `View`-Elemente instanziiert.

Liste mit Daten befüllen und Darstellung konfigurieren.

```
39      private void listViewInitialisieren()
40      {
41          listView = (ListView) findViewById(R.id.listView1
                );
42          ArrayList<Daten> daten = new ArrayList<>();
43          for (int i = 0; i < 20; i++)
44          {
45              daten.add(new Daten("Text" + i, true));
46          }
47          MeinAdapter adapter = new MeinAdapter(this, daten
                );
48          listView.setAdapter(adapter);
49      }
```

Darzustellende Daten erzeugen.

Adapter instanziieren.

Liste mittels Adapter konfigurieren.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoListView.java (Ausschnitt)

Scroll-Position der Liste speichern:

```
119     @Override
120     public void onPause()
121     {
122         super.onPause();
123         logger.log("onPause");
124         SharedPreferences p =
               PreferenceManager.
               getDefaultSharedPreferences(
               this);
125         SharedPreferences.Editor editor
                = p.edit();
126         int scroll = listView.
               getFirstVisiblePosition();
127         editor.putInt(
               KEY_LISTVIEW_SCROLL_POSITION
               , scroll);
128         editor.commit();
129     }
130
131     @Override
132     public void onResume()
133     {
134         super.onResume();
135         logger.log("onResume");
136         SharedPreferences p =
               PreferenceManager.
               getDefaultSharedPreferences(
               this);
137         int scroll = p.getInt(
               KEY_LISTVIEW_SCROLL_POSITION
               , 0);
138         if(listView!=null)
139         {
140             if(listView.getCount()>
                   scroll)
141             {
142                 listView.
                       setSelectionFromTop(
                       scroll, 0);
143             }
144             else
145             {
146                 listView.
                       setSelectionFromTop
                       (0, 0);
147             }
148         }
149     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/-DemoListView.java
(Ausschnitt)

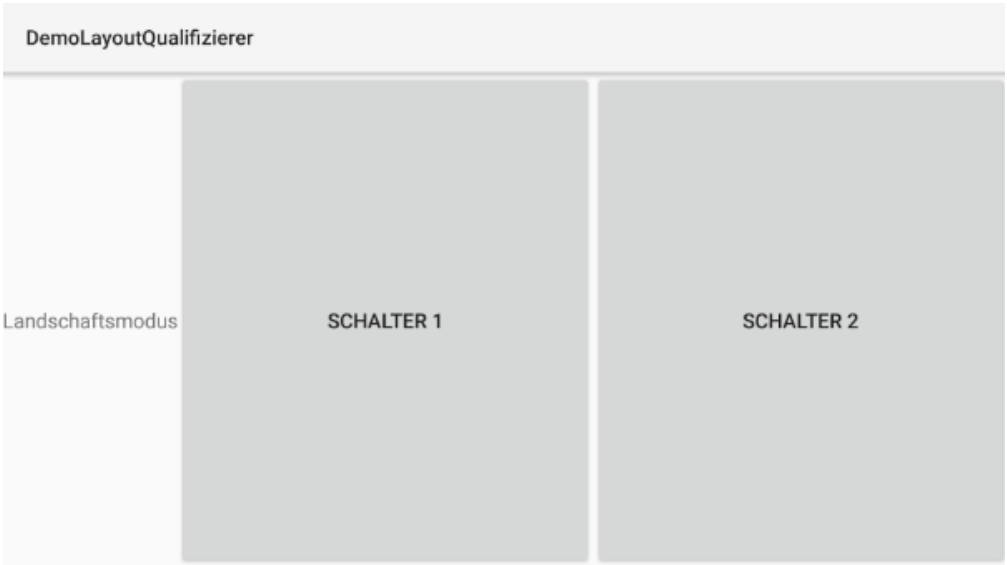
Inhaltsverzeichnis

- In der Android-Entwicklung kann man die Ressourcen außerhalb des Java-Codes in XML-Dateien im Verzeichnis *res* speichern.
- Das Verzeichnis *res* kann man weiter unterteilen in *animator*, *anim*, *color*, *drawable*, *layout*, *menu*, *mipmap*, *raw*, *values*, *xml*.
- Diese Verzeichnisnamen können um Qualifizierer ergänzt werden, die den Inhalt spezifizieren.
- Das Android-Betriebssystem kann dann je nach Situation die geeigneten Ressourcen auswählen.
- Siehe `https://developer.android.com/guide/topics/resources/providing-resources.html`

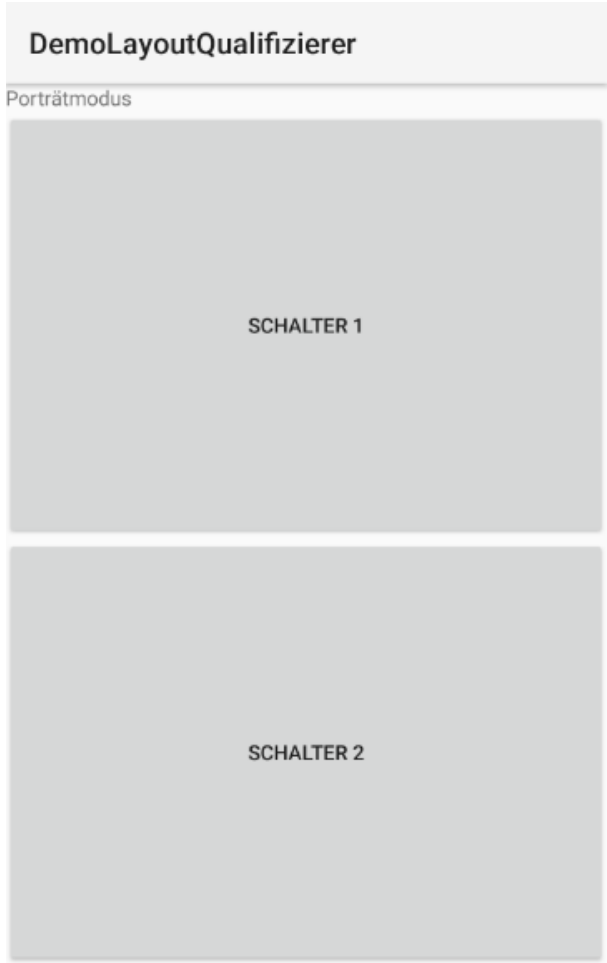
Beispiel Layout:

- Neben dem Layout-Verzeichnis *layout* kann man das Verzeichnis *layout-port* erstellen, und in diesem die Layouts definieren, die geladen werden sollen, falls das Android-Gerät aufrecht steht (Porträt).
- Für unterschiedliche Pixeldichten kann man unterschiedliche Layouts definieren: *layout-ldpi*, *layout-hdpi*, *layout-xxxhdpi*, etc.
- Diese Qualifizier können auch kombiniert werden, z.B. *layout-port-ldpi*

layout/activity_demo_layout_qualifizierer.xml



layout-port/activity_demo_layout_qualifizierer.xml



Landschaftsmodus:

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

     <TextView
         android:id="@+id/textView3"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Landschaftsmodus"/>

     <Button
         android:id="@+id/button2"
         android:layout_width="0dp"
         android:layout_height="match_parent"
         android:layout_weight="1"
         android:text="Schalter 1"/>

     <Button
         android:id="@+id/button7"
         android:layout_width="0dp"
         android:layout_height="match_parent"
         android:layout_weight="1"
         android:text="Schalter 2"/>
</LinearLayout >
```

Demos/app/src/main/res/layout/activity_demo_layout_qualifizierer.xml

Porträtmodus:

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

     <TextView
         android:id="@+id/textView3"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Porträtmodus"/>

     <Button
         android:id="@+id/button2"
         android:layout_width="match_parent"
         android:layout_height="0dp"
         android:layout_weight="1"
         android:text="Schalter 1"/>

     <Button
         android:id="@+id/button7"
         android:layout_width="match_parent"
         android:layout_height="0dp"
         android:layout_weight="1"
         android:text="Schalter 2"/>
</LinearLayout >
```

Demos/app/src/main/res/layout-port/activity_demo_layout_qualifizierer.xml

Beide Layout-Dateien haben denselben Namen, liegen aber in verschiedenen Verzeichnissen.

```
6  public class DemoLayoutQualifizierer extends Activity
7  {
8      @Override
9      protected void onCreate(Bundle savedInstanceState)
10     {
11         super.onCreate(savedInstanceState);
12         setTitle(getClass().getSimpleName());
13         setContentView(R.layout.
               activity_demo_layout_qualifizierer);
14     }
15 }
```

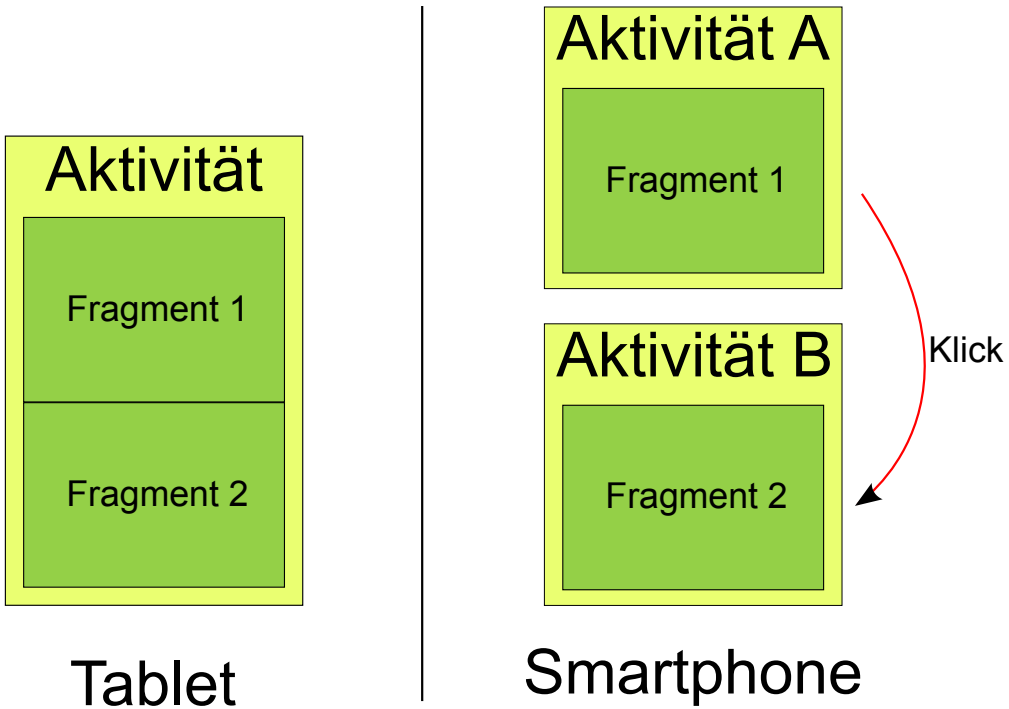
Im Java-Code wird nicht zwischen den beiden Layout-Dateien unterschieden.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoLayoutQualifizierer.java
(Ausschnitt)

Inhaltsverzeichnis

Fragment

- Die Anzeige einer Aktivität lässt sich in Fragmente aufteilen.
- Diese Fragmente können in Abhängigkeit der Bildschirmgröße zu unterschiedlichen Anzeigen kombiniert werden.



Inhaltsverzeichnis

- Das nächste Beispiel zeigt eine Aktivität, die ein statisches Fragment verwendet.
- Ein statisches Fragment wird im Gegensatz zu einem dynamischen Fragment bei der Erstellung der Applikation festgelegt.
- Die Aktivität lädt ein Layout, welches das Fragment enthält, und setzt den Text des Schalters des Fragments.

Layout der Aktivität:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/
       res/android"
3                xmlns:tools="http://schemas.android.com/tools"
4                android:layout_width="match_parent"
5                android:layout_height="match_parent"
6                android:orientation="vertical"
7                tools:context="de.tmahr.android.unterricht.
                     demos.DemoFragmentStatisch">
8      <TextView
9          android:id="@+id/textView"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Dieser Text wird in einer Activity
               definiert." />
13     <FrameLayout
14         android:id="@+id/framelayout_demo_fragment_statisch"
15         android:layout_width="match_parent"
16         android:layout_height="match_parent">
17         <fragment
18             android:id="@+id/
                   fragment_demo_fragment_statisch_fragment"
19             android:layout_width="match_parent"
20             android:layout_height="match_parent"
21             class="de.tmahr.android.unterricht.demos.
                   DemoFragmentStatisch$MeinFragment" />
22     </FrameLayout>
23 </LinearLayout>
```

Demos/app/src/main/res/layout/activity_demo_fragment_statisch.xml

Das Layout der Aktivität besteht aus einer `TextView` und diesem `FrameLayout`. Das `FrameLayout` enthält ein Fragment.

Hier wird das Fragment über den vollständigen Klassennamen festgelegt.

```java
package de.tmahr.android.unterricht.demos;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

public class DemoFragmentStatisch extends Activity
{
    private static void log(String s)
    {
        Log.d(DemoFragmentStatisch.class.getSimpleName(), s);
    }
    private MeinFragment meinFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setTitle(this.getClass().getSimpleName());
        setContentView(R.layout.activity_demo_fragment_statisch);
        meinFragment = (MeinFragment) getFragmentManager().findFragmentById(R.id.
            fragment_demo_fragment_statisch_fragment);
        meinFragment.button.setText("Fragment-Button beschriftet von Aktivität");
        log("onCreate");
    }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentStatisch.java (Ausschnitt)

Lädt das zuvor gezeigte Layout mit dem im Layout festgelegten Fragment.

Sucht das angegebene Fragment in der gesetzten View der Aktivität.

Setzt den Text des im Fragment enthaltenen Schalters.

```
32      public static class MeinFragment extends Fragment
33      {
34          Button button;
35
36          @Override
37          public void onCreate(Bundle savedInstanceState)
38          {
39              super.onCreate(savedInstanceState);
40              log(getClass().getSimpleName()+".onCreate");
41          }
42
43          @Override
44          public View onCreateView(LayoutInflater inflater, ViewGroup
                container, Bundle savedInstanceState)
45          {
46              log(getClass().getSimpleName()+".onCreateView");
47              View view = inflater.inflate(R.layout.
                    fragment_statisch_fragment, container, false);
48              TextView textView = (TextView) view.findViewById(R.id.
                    textView1);
49              button = (Button) view.findViewById(R.id.button);
50              return view;
51          }
52      }
53  }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentStatisch.java (Ausschnitt)

Das Fragment erweitert die Basisklasse `Fragment`.

Auf diesen Schalter greift die Aktivität zu.

Wird beim Anlegen des Fragments aufgerufen.

Liefert die Anzeige des Fragments.

Weist dem Fragment ein Layout zu.

Sucht Elemente im geladenen Layout.

Layout des Fragments:

```
1  <FrameLayout xmlns:android="http://schemas.android.com/
       apk/res/android"
2              android:layout_width="match_parent"
3              android:layout_height="match_parent">
4      <TextView
5          android:id="@+id/textView1"
6          android:layout_width="wrap_content"
7          android:layout_height="wrap_content"
8          android:text="Dieser Text wird in einem Fragment
               definiert."
9          android:textAppearance="?android:attr/
               textAppearanceLarge"/>
10     <Button
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Button des Fragments"
14         android:id="@+id/button"
15         android:layout_gravity="center"/>
16 </FrameLayout>
```

Demos/app/src/main/res/layout/fragment_mein.xml

Das Layout des Fragments besteht aus einer `TextView`

und einem Button.

Inhaltsverzeichnis

- Bei einem dynamischen Fragment wird im Gegensatz zu einem statischen Fragment das anzuzeigende Fragment zur Laufzeit der Applikation bestimmt.
- Siehe Beispiel: *DemoFragmentDynamisch*

- Die orange Fläche ist im Layout der Aktivität definiert.
- Die grüne Fläche ist im Layout von Fragment 1 definiert.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                xmlns:tools="http://schemas.android.com/tools"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                tools:context="de.tmahr.android.unterricht.demos.DemoFragmentDynamisch"
                android:background="@android:color/holo_orange_light">

    <Button
        android:id="@+id/buttonZeigeFragment1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Zeige Fragment 1"
        android:onClick="onClickZeigeFragment1" />

    <Button
        android:id="@+id/buttonZeigeFragment2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/buttonZeigeFragment1"
        android:text="Zeige Fragment 2"
        android:onClick="onClickZeigeFragment2" />
```

Layout der Aktivität.

Demos/app/src/main/res/layout/activity_demo_fragment_dynamisch.xml

```
27      <Button
28          android:id="@+id/buttonEntferneFragment"
29          android:layout_width="match_parent"
30          android:layout_height="wrap_content"
31          android:layout_alignParentLeft="true"
32          android:layout_below="@+id/buttonZeigeFragment2"
33          android:text="Entferne Fragment"
34          android:onClick="onClickEntferneFragment"/>
35
36      <FrameLayout
37          android:id="@+id/frame"
38          android:layout_width="match_parent"
39          android:layout_height="match_parent"
40          android:layout_alignParentLeft="true"
41          android:layout_below="@+id/buttonEntferneFragment">
42      </FrameLayout>
43
44  </RelativeLayout>
```

Demos/app/src/main/res/layout/activity_demo_fragment_dynamisch.xml

Im Gegensatz zum statischen Fragment enthält das `FrameLayout` noch kein Fragment. Das einzubettende Fragment wird erst zur Laufzeit festgelegt.

```
16 public class DemoFragmentDynamisch extends Activity
17 {
18     private Fragment aktuellesFragment;
19     private void log(String nachricht)
20     {
21         Log.d(this.getClass().getSimpleName(), nachricht);
22     }
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState)
26     {
27         super.onCreate(savedInstanceState);
28         setTitle(this.getClass().getSimpleName());
29         setContentView(R.layout.activity_demo_fragment_dynamisch);
30     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentDynamisch.java (Ausschnitt)

```
32      public void onClickZeigeFragment1(View view)
33      {
34          Button button = (Button)view;
35          log(button.getText() + " ausgewählt");
36          zeigeFragment(Fragment1.erstellen("Fragment 1 wurde
               angeklickt"));
37      }
38
39      public void onClickZeigeFragment2(View view)
40      {
41          Button button = (Button)view;
42          log(button.getText() + " ausgewählt");
43          zeigeFragment(Fragment2.erstellen("Fragment 2 wurde
               angeklickt"));
44      }
45
46      public void onClickEntferneFragment(View view)
47      {
48          entferneFragment();
49      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentDynamisch.java (Ausschnitt)

Der erste Schalter lädt `Fragment1`.

`Fragment1` erstellen und darstellen.

Der zweite Schalter lädt `Fragment2`.

`Fragment2` erstellen und darstellen.

Der dritte Schalter entfernt das Fragment.

```
51      private void zeigeFragment(Fragment fragment)
52      {
53          FragmentManager fm = getFragmentManager();
54          FragmentTransaction ft = fm.beginTransaction();
55          ft.replace(R.id.frame, fragment);
56          ft.commit();
57          aktuellesFragment = fragment;
58      }
59
60      private void entferneFragment()
61      {
62          if(aktuellesFragment!=null)
63          {
64              FragmentManager fm = getFragmentManager();
65              FragmentTransaction ft = fm.beginTransaction();
66              ft.remove(aktuellesFragment);
67              ft.commit();
68              aktuellesFragment = null;
69          }
70      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentDynamisch.java
(Ausschnitt)

Der `FragmentManager` ermöglicht den Zugriff auf Fragmente innerhalb der Aktivität.

Öffnet eine Transaktion zur Bearbeitung der Fragmente.

Weist dem `FrameLayout` der Aktivität ein Fragment zu.

Beginnt den Abschluss der Transaktion.

Entfernt das Fragment.

```
72      public static class Fragment1 extends Fragment
73      {
74          private String text;
75          private final static String KEY_TEXT = "KEY_TEXT";
76          private void log(String nachricht)
77          {
78              Log.d(this.getClass().getSimpleName(), nachricht);
79          }
80
81          @Override
82          public View onCreateView(LayoutInflater inflater,
                ViewGroup container, Bundle bundle)
83          {
84              log("onCreateView");
85              View view = inflater.inflate(R.layout.
                    fragment_dynamisch_fragment1, container, false)
                    ;
86              TextView textView = (TextView) view.findViewById(R
                    .id.textView);
87              textView.setText(text);
88              return view;
89          }
```

Das erste Fragment.

Dieser Schlüssel wird für die Initialisierung des Attributs `text` des Fragments verwendet.

Erstellt die Ansicht des Fragments aus einem XML-Layout

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentDynamisch.java
(Ausschnitt)

```
91      public static Fragment1 erstellen(String text)
92      {
93          Fragment1 fragment = new Fragment1();
94          Bundle b = new Bundle();
95          b.putString(KEY_TEXT, text);
96          fragment.setArguments(b);
97          return fragment;
98      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/-DemoFragmentDynamisch.java
(Ausschnitt)

Die dynamischen Fragmente sollten über diese statische Methode erstellt werden, denn auf diese Weise können Konstruktion (mittels Konstruktor) und die Initialisierung der Attribute mittels `Bundle` voneinander getrennt werden. Dies ist notwendig, da es möglich ist, dass der `FragmentManager` das Fragment ohne Aufruf von `erstellen` neu anlegt und das Fragment mittels `Bundle` initialisiert.

Fragment erstellen.

Die für die Initialisierung des Fragment verwendeten Argumente werden in das `Bundle` eingefügt und das `Bundle` dem Fragment übergeben.

```
100         @Override
101         public void onCreate(Bundle bundle)
102         {
103             super.onCreate(bundle);
104             Bundle args = getArguments();
105             if (args != null)
106             {
107                 text = args.getString(KEY_TEXT);
108                 log("onCreate: text="+text);
109             }
110             else
111             {
112                 log("onCreate");
113             }
114         }
115     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/-DemoFragmentDynamisch.java (Ausschnitt)

Hier wird das zuvor erstellte `Bundle` übergeben.

Der Wert aus dem `Bundle` wird dem Attribut des Arguments übergeben.

Layout von `Fragment1`:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                android:layout_width="match_parent"
4                android:layout_height="match_parent"
5                android:gravity="center_horizontal"
6                android:orientation="vertical"
7                android:background="@android:color/holo_green_light">
8
9      <TextView
10         android:id="@+id/textView"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:textAppearance="?android:attr/textAppearanceLarge" />
14
15 </LinearLayout>
```

Demos/app/src/main/res/layout/fragment_dynamisch_fragment1.xml

```
117     public static class Fragment2 extends Fragment
118     {
119         private String text;
120         private final static String KEY_TEXT = "KEY_TEXT";
121         private void log(String nachricht)
122         {
123             Log.d(this.getClass().getSimpleName(), nachricht);
124         }
125
126         @Override
127         public View onCreateView(LayoutInflater inflater,
                ViewGroup container, Bundle bundle)
128         {
129             log("onCreateView");
130             View view = inflater.inflate(R.layout.
                    fragment_dynamisch_fragment2, container, false)
                    ;
131             EditText editText = (EditText) view.findViewById(R
                    .id.editText1);
132             editText.setText(text);
133             return view;
134         }
```

Das erste Fragment.

Dieser Schlüssel wird für die Initialisierung des Attributs `text` des Fragments verwendet.

Erstellt die Ansicht des Fragments aus einem XML-Layout

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoFragmentDynamisch.java (Ausschnitt)

```
136     public static Fragment2 erstellen(String text)
137     {
138         Fragment2 fragment = new Fragment2();
139         Bundle b = new Bundle();
140         b.putString(KEY_TEXT, text);
141         fragment.setArguments(b);
142         return fragment;
143     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/-DemoFragmentDynamisch.java (Ausschnitt)

Die dynamischen Fragmente sollten über diese statische Methode erstellt werden, denn auf diese Weise können Konstruktion (mittels Konstruktor) und die Initialisierung der Attribute mittels `Bundle` voneinander getrennt werden. Dies ist notwendig, da es möglich ist, dass der `FragmentManager` das Fragment ohne Aufruf von `erstellen` neu anlegt und das Fragment mittels `Bundle` initialisiert.

Fragment erstellen.

Die für die Initialisierung des Fragment verwendeten Argumente werden in das `Bundle` eingefügt und das `Bundle` dem Fragment übergeben.

```
145         @Override
146         public void onCreate(Bundle bundle)
147         {
148             super.onCreate(bundle);
149             Bundle args = getArguments();
150             if (args != null)
151             {
152                 text = args.getString(KEY_TEXT);
153                 log("onCreate: text="+text);
154             }
155             else
156             {
157                 log("onCreate");
158             }
159         }
160     }
161 }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/-DemoFragmentDynamisch.java (Ausschnitt)

Hier wird das zuvor erstellte `Bundle` übergeben.

Der Wert aus dem `Bundle` wird dem Attribut des Arguments übergeben.

Layout von `Fragment2`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:gravity="center_horizontal"
              android:orientation="vertical"
              android:background="@android:color/holo_blue_light">

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:ems="10"
        android:textStyle="italic"
        android:inputType="textMultiLine"  >
        <requestFocus />
    </EditText>

</LinearLayout>
```

Demos/app/src/main/res/layout/fragment_dynamisch_fragment2.xml

Inhaltsverzeichnis

Übung: Statische Fragmente

Erstellen Sie eine Applikation mit den folgenden Eigenschaften:

1. Die Applikation zeigt zwei statische Fragmente F1 und F2.
2. Im Landschaftsmodus werden die Fragmente nebeneinander angezeigt.
3. Im Porträtmodus werden die Fragmente untereinander angezeigt.
4. F1 enthält einen ToggleButton TB.
5. F2 enthält eine CheckBox CB.
6. Aktiviert ein Benutzer TB, so wird automatisch CB aktiviert.
7. Deaktiviert ein Benutzer TB, so wird automatisch CB deaktiviert.

Inhaltsverzeichnis

Inhaltsverzeichnis

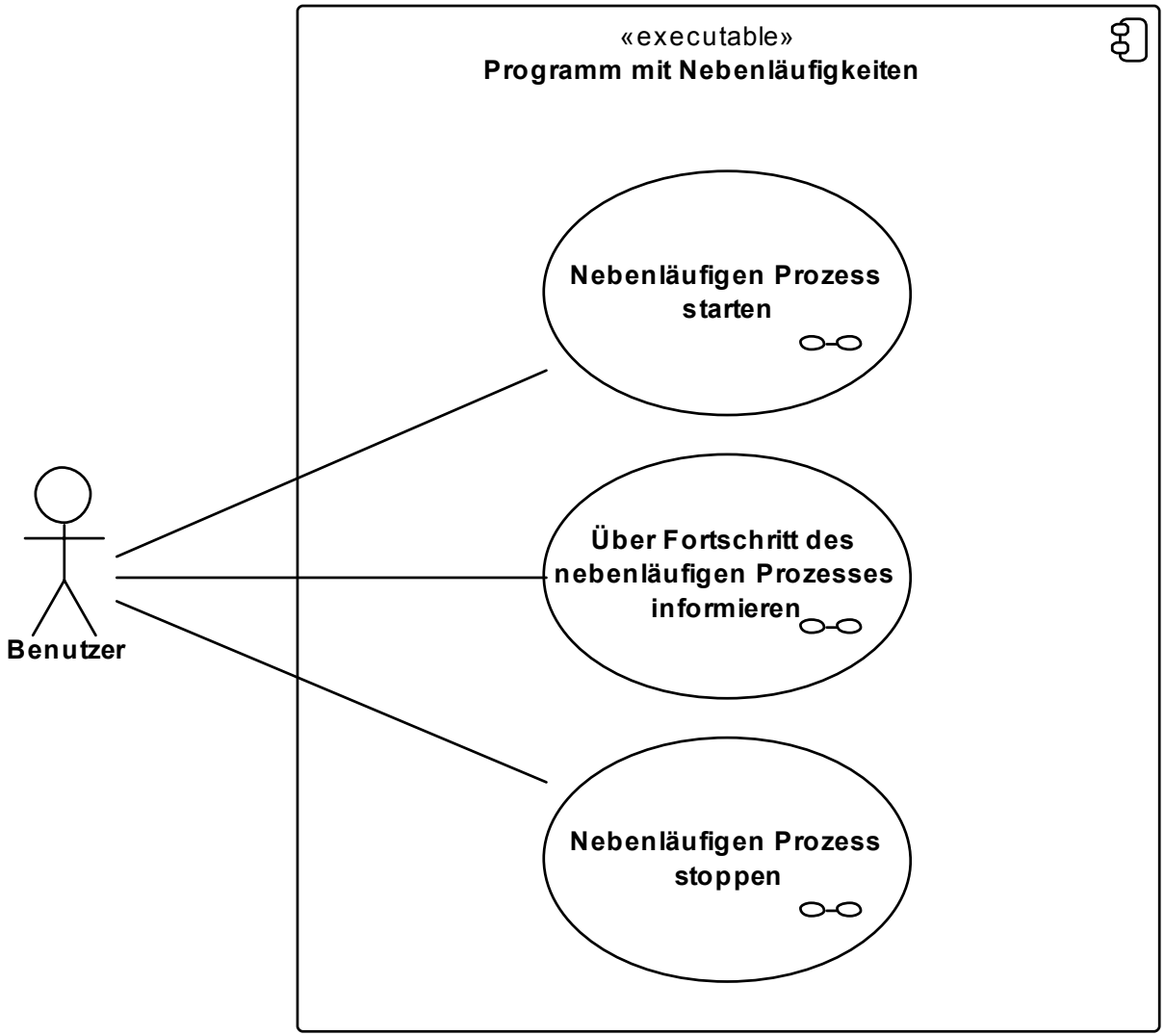


Abbildung: Anwendungsfälle

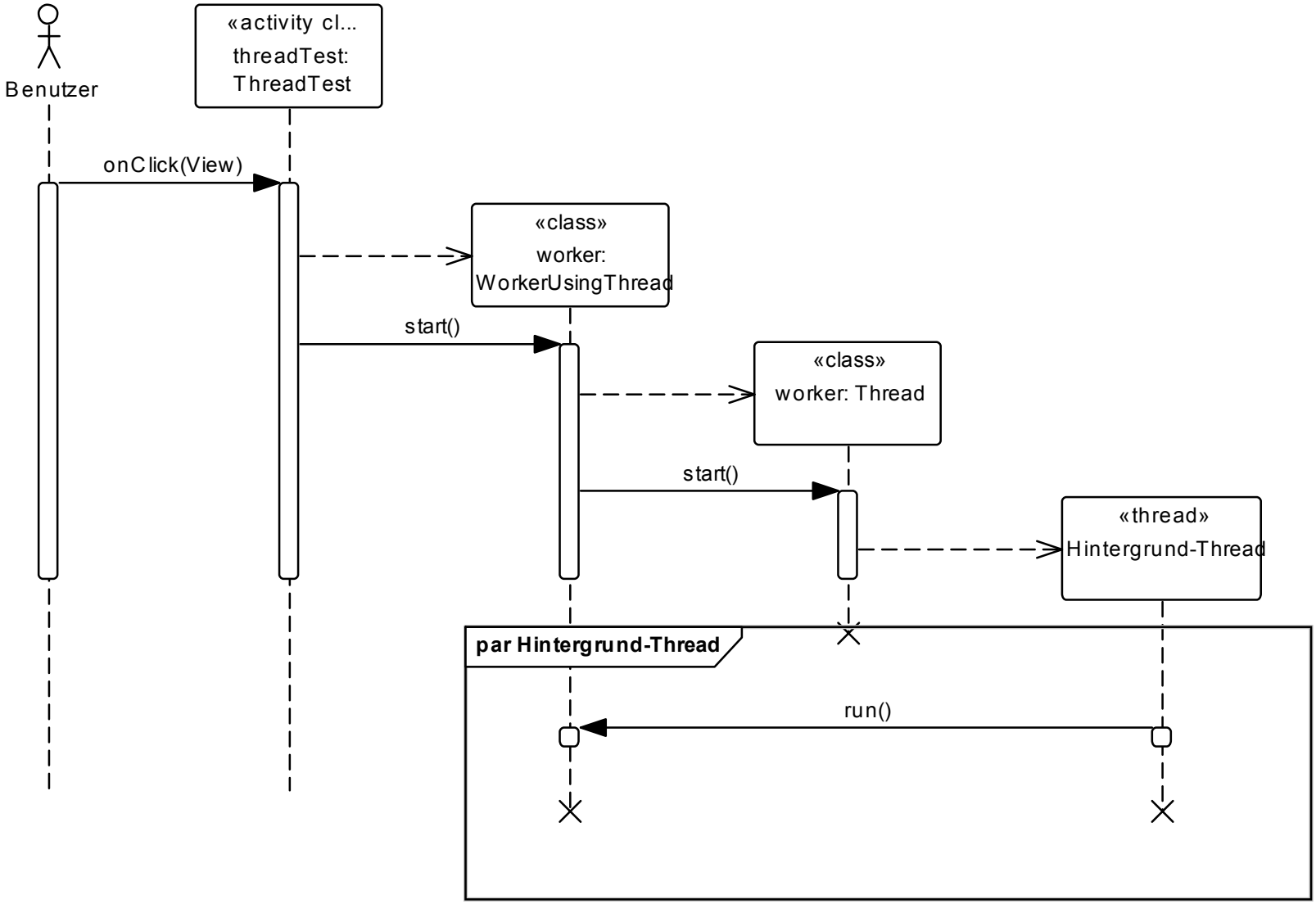


Abbildung: Thread starten

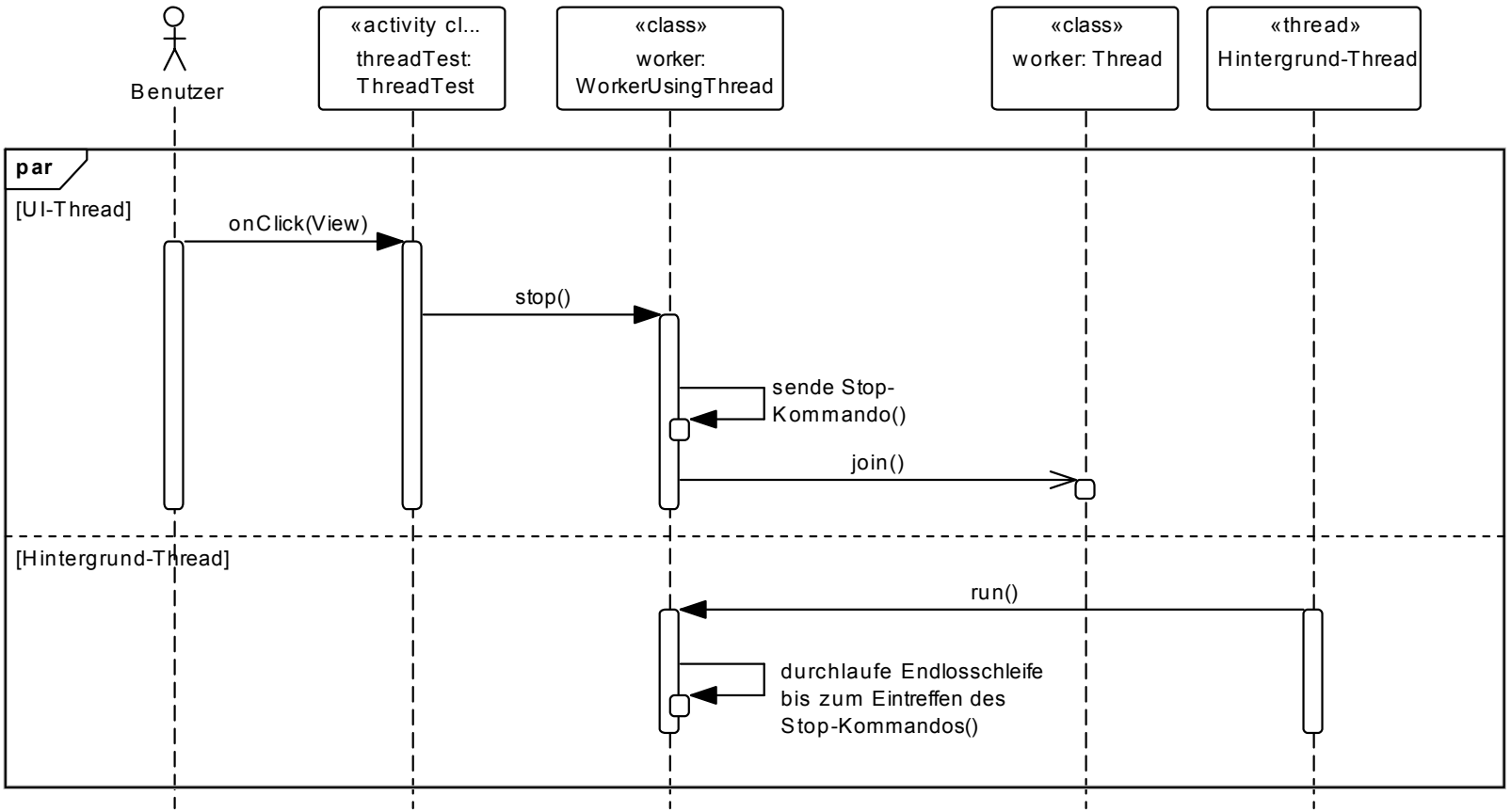


Abbildung: Thread stoppen

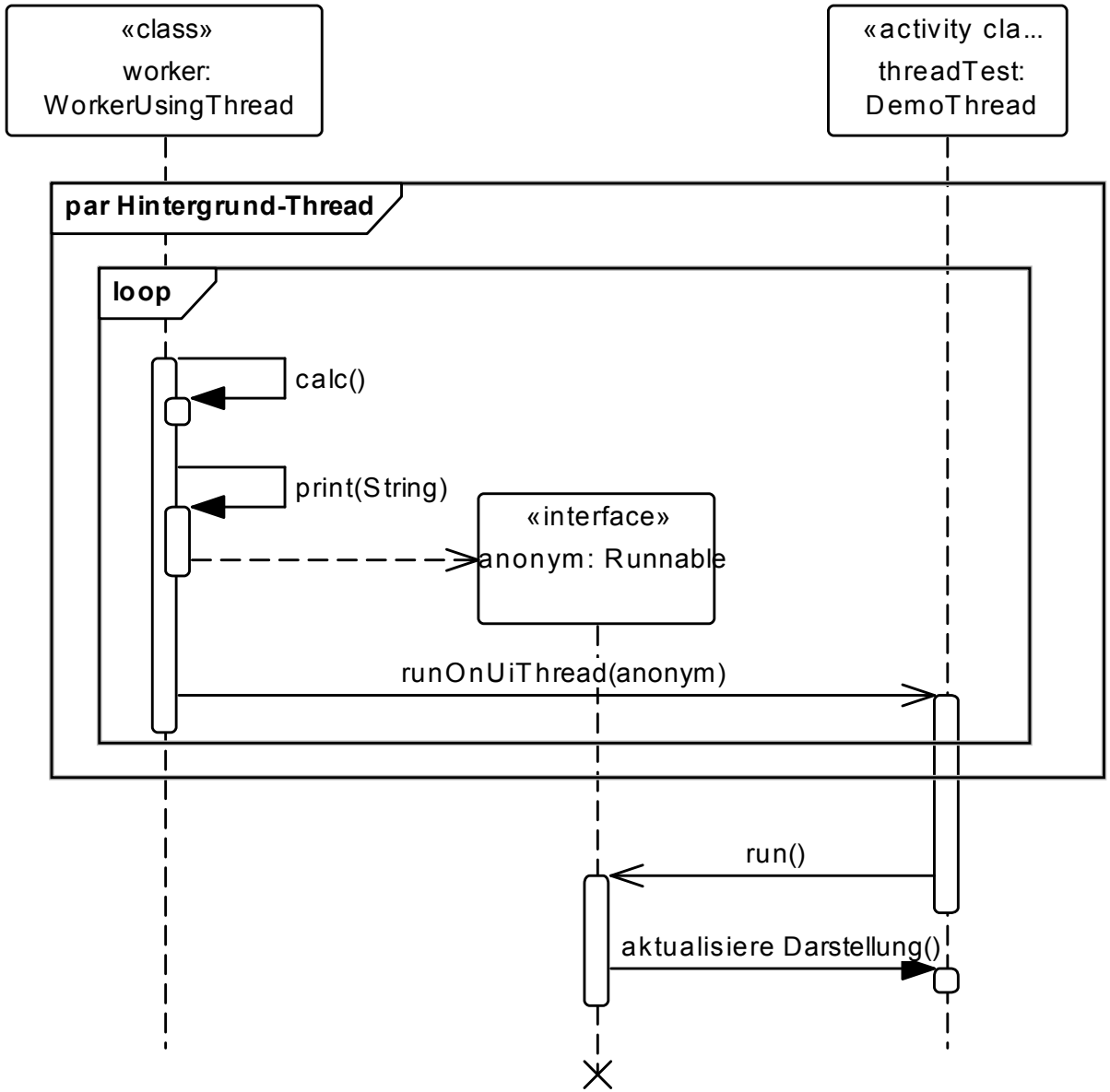


Abbildung: Android verbietet es von einer Nebenläufigkeit direkt auf die Benutzeroberfläche zuzugreifen

- Die Klasse `WorkerUsingThread` implementiert das Interface `Runnable`.
- Dazu müssen wird die Schnittstellenfunktion `run()` implementieren, die dann in einem innerhalb dieser Klasse angelegten Thread ausgeführt wird.

```
104     class WorkerUsingThread implements Runnable
105     {
106         private volatile boolean running = false;
107         private Thread thread;
108         private String threadName = "worker thread";
109
110         private void print(final String s)
111         {
112             runOnUiThread(new Runnable()
113             {
114                 @Override
115                 public void run()
116                 {
117                     textViewWorkerThread.setText(s);
118                 }
119             });
120         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Das Schlüsselwort **`volatile`** benötigen wir, da die Variable `running` unerwartet von anderen Programmteilen geändert werden kann (siehe unten).

Android verbietet es, von einer Nebenläufigkeit auf die Benutzeroberfläche direkt zuzugreifen. Da die Funktion `print` nebenläufig ausgeführt wird, darf diese nicht direkt auf die Benutzeroberfläche zugreifen, sondern delegiert den Zugriff an die Funktion `runOnUiThread`.

Dieser Funktion wird ein Objekt vom Type `Runnable` übergeben. Hierfür wird das Interface `Runnable` implizit mit einer anonymen Klasse implementiert, die die Schnittstellenfunktion `run()` implementiert. Die Funktion `run()` wird vom UI-Thread ausgeführt und aktualisiert den Text der `View`.

```
122         @Override
123         public void run()
124         {
125             int i = 0;
126             while (running)
127             {
128                 i++;
129                 print(String.valueOf(i));
130                 try
131                 {
132                     Thread.sleep(100);
133                 }
134                 catch (InterruptedException e)
135                 {
136                     e.printStackTrace();
137                 }
138             }
139             print(getString(R.string.workerThread) + "
                    endet mit " + i);
140         }
```

Diese Funktion läuft innerhalb des Threads.

Die Verarbeitung des Threads pausiert für 100 ms.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

```
142         void start()
143         {
144             logger.log("Starting " + threadName + "...");
145             running = true;
146             thread = new Thread(this);
147             thread.setName(threadName);
148             thread.start();
149             logger.log("... " + threadName + " started");
150         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Da unsere eigene Klasse `WorkerUsingThread` das Interface Runnable implementiert, können wir die aktuelle Instanz direkt an den neu zu erstellenden Thread mit **`this`** übergeben. Der Thread führt dann unsere `run()`-Funktion aus.

Anhand dieses Namens können wir den Thread im Monitor (ADM, DDMS) identifizieren.

```
152         void stop()
153         {
154             if (!running)
155             {
156                 logger.log(threadName + " not running");
157             }
158             else
159             {
160                 logger.log("Stopping " + threadName + "
                        ...");
161                 running = false;
162                 while (true)
163                 {
164                     try
165                     {
166                         thread.join();
167                         logger.log("... " + threadName +
                                " stopped");
168                         break;
169                     }
170                     catch (InterruptedException e)
171                     {
172                         e.printStackTrace();
173                     }
174                 }
175             }
176         }
177     }
```

Von dieser Zeile ist keine der folgenden Zeilen abhängig. Daher könnte der Compiler die Reihenfolge dieser Zeile und der nachfolgenden Zeilen vertauschen. Damit der Compiler das nicht macht, muss `running` als **`volatile`** definiert sein.

Warten bis der Thread beendet ist.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Inhaltsverzeichnis

- Die Klasse `WorkerUsingAsyncTask` erweitert die parametrierbare (generische) Klasse `AsyncTask`.
- `AsyncTask` erlaubt einen bequemen Weg zur Ausführung von nebenläufigen Arbeiten im Hintergrund und der Kommunikation des Hintergrundthreads mit dem Thread der Benutzschnittstelle (UI).

Verwendung der Parameter der generischen Klasse `AsyncTask<Long, String, Long>`:

1. Parameter (hier `Long`): Gibt den Typ der Daten an, die an den Hintergrundprozess übergeben werden (siehe `doInBackground`).
2. Parameter (hier `String`): Gibt den Typ der Daten an, die veröffentlicht werden (siehe `publishProgress` und `onProgressUpdate`).
3. Parameter (hier `Long`): Gibt den Typ des Ergebnisses der Berechnung des Hintergrundprozesses an (siehe Rückgabewert von `doInBackground`).

Die Nebenläufigkeit kann auf zwei Wegen beendet werden:

1. Aufruf von `cancel()` von außerhalb der nebenläufig ausgeführten Funktion `doInBackground`.
   - Dieses Abbruchereignis muss innerhalb von `doInBackground` mittels der Funktion `isCancelled()` abgefragt werden. Der Programmierer muss dann dafür sorgen, dass `doInBackground` über **`return`** verlassen wird.
   - Auf einen über **`return`** zurückgelieferten Wert `wert` kann dann außerhalb der Nebenläufigkeit mittels `onCancelled(Long wert)` zugegriffen werden.
2. Die Nebenläufigkeit beendet sich nach Ermittlung des Berechnungsergebnisses selbst, ohne Aufruf von `cancel()`.
   - Dies geschieht einfach über das Verlassen von `doInBackground` über **`return`**.
   - Auf einen über **`return`** zurückgelieferten Wert `wert` kann dann außerhalb der Nebenläufigkeit mittels `onPostExecute(Long wert)` zugegriffen werden.

```
179     class WorkerUsingAsyncTask extends AsyncTask<
            Long, String, Long>
180     {
181         private long zaehler = 0;
182         private final TextView textView;
183
184         WorkerUsingAsyncTask(TextView textView)
185         {
186             this.textView = textView;
187         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

`WorkerUsingAsyncTask` erweitert die generische Klasse `AsyncTask` und definiert dabei die Klassenparameter:

1. `Long`: Gibt den Typ der Daten an, die an den Hintergrundprozess übergeben werden.
2. `String`: Gibt den Typ der Daten an, die veröffentlicht werden.
3. `Long`: Gibt den Typ des Ergebnisses der Berechnung des Hintergrundprozesses an.

```
189         @Override
190         protected Long doInBackground(Long... params)
191         {
192             Long limit = (params.length>0)? params[0] : null;
193 
194             for (;;)
195             {
196                 if (isCancelled())
197                 {
198                     break;
199                 }
200                 else
201                 {
202                     zaehler++;
203                     publishProgress(String.valueOf(zaehler));
204 
205                     if(limit!=null && limit.equals(zaehler))
206                     {
207                         break;
208                     }
209 
210                     try
211                     {
212                         Thread.sleep(100);
213                     }
214                     catch (InterruptedException e)
215                     {
216                         e.printStackTrace();
217                     }
218                 }
219             }
220             return zaehler;
221         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Das optionale Argument gibt die Begrenzung des Zählers an. Sobald der Zähler den Wert `param[0]` erreicht hat, beendet sich die Nebenläufigkeit ohne Aufruf von `AsyncTask.cancel()`.

Diese Funktion wird in der Nebenläufigkeit ausgeführt. Sie dient dazu, die in der Nebenläufigkeit berechneten Daten in den Thread der Benutzeroberfläche zu transportieren, um diese dort darzustellen. Innerhalb des Threads der Benutzeroberfläche nimmt die Funktion `onProgressUpdate` die darzustellenden Daten entgegen und zeigt sie an.

Bei Erreichen der Begrenzung wird die Nebenläufigkeit verlassen.

Die Verarbeitung des Threads pausiert 100 ms.

```
224         protected void onProgressUpdate(String...
                progress)
225         {
226             super.onProgressUpdate(progress);
227             textView.setText(progress[0]);
228         }
229
230         @Override
231         protected void onPreExecute()
232         {
233             super.onPreExecute();
234             logger.log("WorkerAsync.onPreExecute " +
                    getStatus().toString());
235         }
236
237         @Override
238         protected void onPostExecute(Long result)
239         {
240             super.onPostExecute(result);
241             logger.log("WorkerAsync.onPostExecute " +
                    getStatus().toString() + " " + result);
242             textView.setText(getString(R.string.
                    workerAsyncTask1) + " endet mit " + result
                    );
243         }
```

Diese Funktion wird im Thread der Benutzeroberfläche ausgeführt und darf auf die `View`-Elemente zugreifen.

Diese Funktion wird (neuerdings) nur aufgerufen, wenn die Funktion `doInBackground` nicht mit `cancel()` beendet wurde.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

```
245         @Override
246         protected void onCancelled()
247         {
248             super.onCancelled();
249             logger.log("WorkerAsync.onCancelled");
250         }
251
252         @Override
253         protected void onCancelled(Long result)
254         {
255             super.onCancelled(result);
256             logger.log("WorkerAsync.onCancelled(Long) "
                    + result);
257             textView.setText(getString(R.string.
                    workerAsyncTask1) + " endet mit " + result
                    );
258         }
259     }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Diese Funktion wird aufgerufen, wenn die Funktion `doInBackground` mit `cancel()` beendet wurde. Das Berechnungsergebnis der Nebenläufigkeit wird als Argument übergeben.

Inhaltsverzeichnis

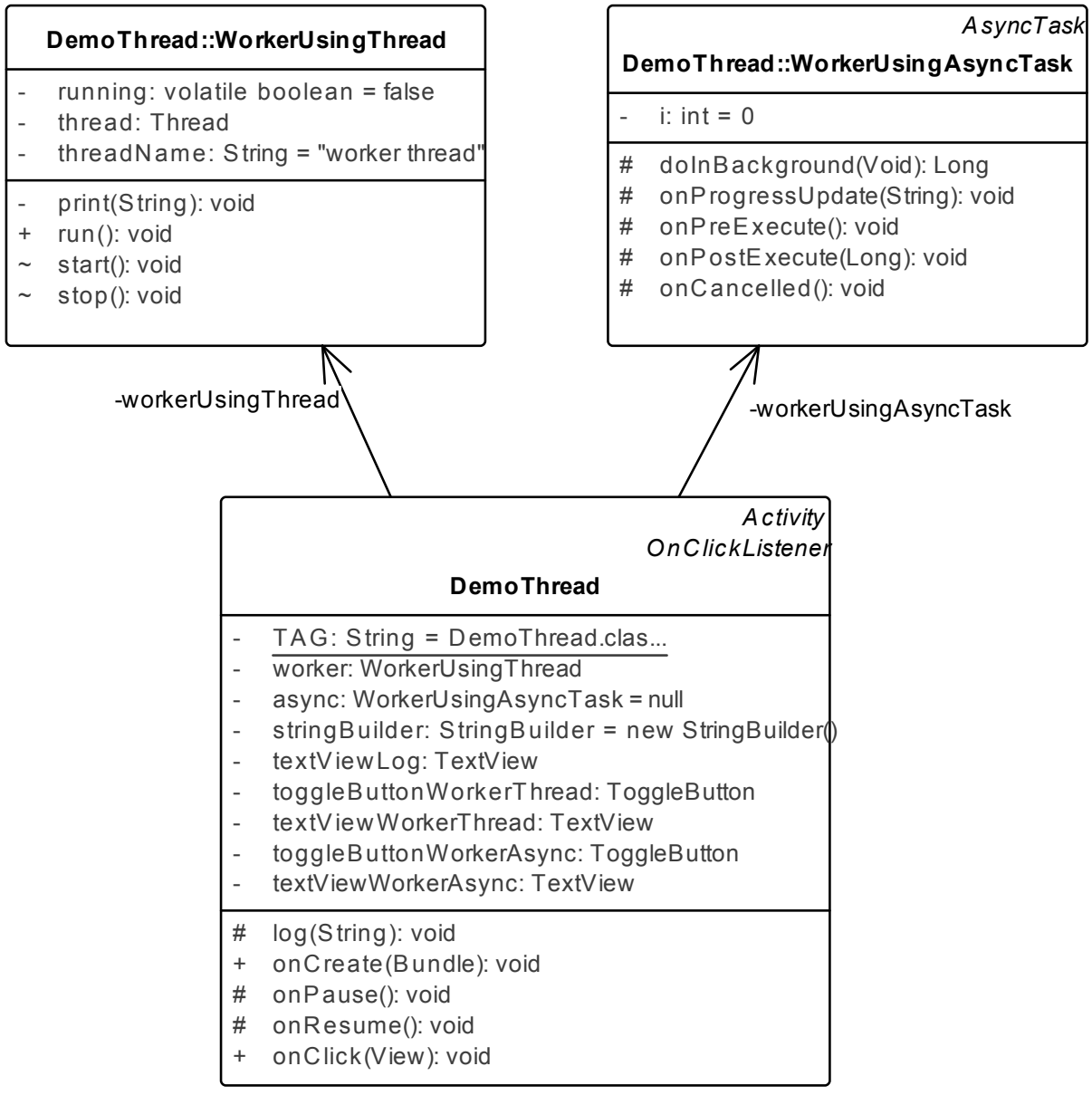


Abbildung: Komposition der DemoThread-Aktivität

Benutzeroberfläche der Aktivität `DemoLayout`:

```
15 public class DemoThread extends Activity implements View.OnClickListener
16 {
17     private WorkerUsingThread workerUsingThread;
18     private WorkerUsingAsyncTask workerUsingAsyncTask1 = null;
19     private WorkerUsingAsyncTask workerUsingAsyncTask2 = null;
20     private TextView textViewLog;
21     private ToggleButton toggleButtonWorkerThread;
22     private TextView textViewWorkerThread;
23     private ToggleButton toggleButtonWorkerAsync1;
24     private TextView textViewWorkerAsync1;
25     private Button buttonWorkerAsync2;
26     private TextView textViewWorkerAsync2;
27     private Logger logger;
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

```
29      @Override
30      protected void onCreate(Bundle savedInstanceState)
31      {
32          super.onCreate(savedInstanceState);
33          setContentView(R.layout.activity_demo_thread);
34          setTitle(this.getClass().getSimpleName());
35          textViewLog = (TextView) findViewById(R.id.textViewLog);
36          logger = new Logger(this.getClass().getSimpleName(), textViewLog, "");
37          toggleButtonWorkerThread = (ToggleButton) findViewById(R.id.
                toggleButtonWorkerThread);
38          toggleButtonWorkerThread.setOnClickListener(this);
39          textViewWorkerThread = (TextView) findViewById(R.id.textViewWorkerThread);
40          toggleButtonWorkerAsync1 = (ToggleButton) findViewById(R.id.
                toggleButtonWorkerAsyncTask1);
41          toggleButtonWorkerAsync1.setOnClickListener(this);
42          textViewWorkerAsync1 = (TextView) findViewById(R.id.textViewWorkerAsyncTask1);
43          buttonWorkerAsync2 = (Button) findViewById(R.id.buttonWorkerAsyncTask2);
44          buttonWorkerAsync2.setOnClickListener(this);
45          textViewWorkerAsync2 = (TextView) findViewById(R.id.textViewWorkerAsyncTask2);
46          logger.log("onCreate");
47      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

```
49      @Override
50      protected void onPause()
51      {
52          super.onPause();
53          logger.log("onPause");
54          if (workerUsingThread != null) { workerUsingThread.stop(); }
55          if (workerUsingAsyncTask1 != null) { workerUsingAsyncTask1.
                cancel(true); }
56          if (workerUsingAsyncTask2 != null) { workerUsingAsyncTask2.
                cancel(true); }
57      }
58
59      @Override
60      protected void onResume()
61      {
62          super.onResume();
63          logger.log("onResume");
64          workerUsingThread = new WorkerUsingThread();
65          if (toggleButtonWorkerThread.isChecked()) { workerUsingThread
                .start(); }
66          if (toggleButtonWorkerAsync1.isChecked())
67          {
68              workerUsingAsyncTask1 = new WorkerUsingAsyncTask(
                    textViewWorkerAsync1);
69              workerUsingAsyncTask1.execute();
70          }
71      }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Sobald die Aktivität in den Hintergrund geht, werden alle Nebenläufigkeiten gestoppt.

Sobald die Aktivität in den Vordergrund geht, werden die Nebenläufigkeiten bei Bedarf gestartet.

```
73      @Override
74      public void onClick(View v)
75      {
76          if (v == toggleButtonWorkerThread)
77          {
78              logger.log("onClick toggleButtonThread " +
                    toggleButtonWorkerThread.isChecked());
79              if (toggleButtonWorkerThread.isChecked()) { workerUsingThread
                    .start(); }
80              else { workerUsingThread.stop(); }
81          }
82          else if (v == toggleButtonWorkerAsync1)
83          {
84              logger.log("onClick toggleButtonAsync1 " +
                    toggleButtonWorkerAsync1.isChecked());
85              if (toggleButtonWorkerAsync1.isChecked())
86              {
87                  workerUsingAsyncTask1 = new WorkerUsingAsyncTask(
                        textViewWorkerAsync1);
88                  workerUsingAsyncTask1.execute();
89              }
90              else
91              {
92                  workerUsingAsyncTask1.cancel(true);
93                  workerUsingAsyncTask1 = null;
94              }
95          }
96          else if (v == buttonWorkerAsync2)
97          {
98              logger.log("onClick toggleButtonAsync2");
99              workerUsingAsyncTask2 = new WorkerUsingAsyncTask(
                    textViewWorkerAsync2);
100             workerUsingAsyncTask2.execute(Long.valueOf(10));
101         }
102     }
```

Über die Schalter werden die Nebenläufigkeiten gesteuert.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoThread.java (Ausschnitt)

Inhaltsverzeichnis

Übung: Speed-Reader 2

Diese Übung erweitert Speed-Reader 1 um eine automatische Anzeige der einzelnen Worte:

1. Anstelle der manuellen Anzeige des nächsten Wortes, sollen die einzelnen Worte automatisch mit einer bestimmten Frequenz angezeigt werden.
2. Beim Drücken des Schalters S3 beginnt jetzt die automatische Anzeige der einzelnen Worte mit einer bestimmten Frequenz.
3. Verwenden Sie für S3 einen `ToggleButton`.
4. Die Frequenz wird mittels eines Schiebereglers R unterhalb des Textfensters T2 gesteuert.
5. Die Schalter müssen wieder treffend beschriftet werden.

Inhaltsverzeichnis

Service

- Ein Service
  - wird von einer Aktivität oder einer anderen Applikationskomponente gestartet
  - und kann im Hintergrund laufen,
  - selbst wenn der Benutzer nicht mit der Applikation interagiert
  - oder sogar zu einer anderen Applikation wechselt.
- Ein Service besitzt keine grafische Benutzerschnittstelle.

Beispiele für Aufgaben, die in einem Service laufen:

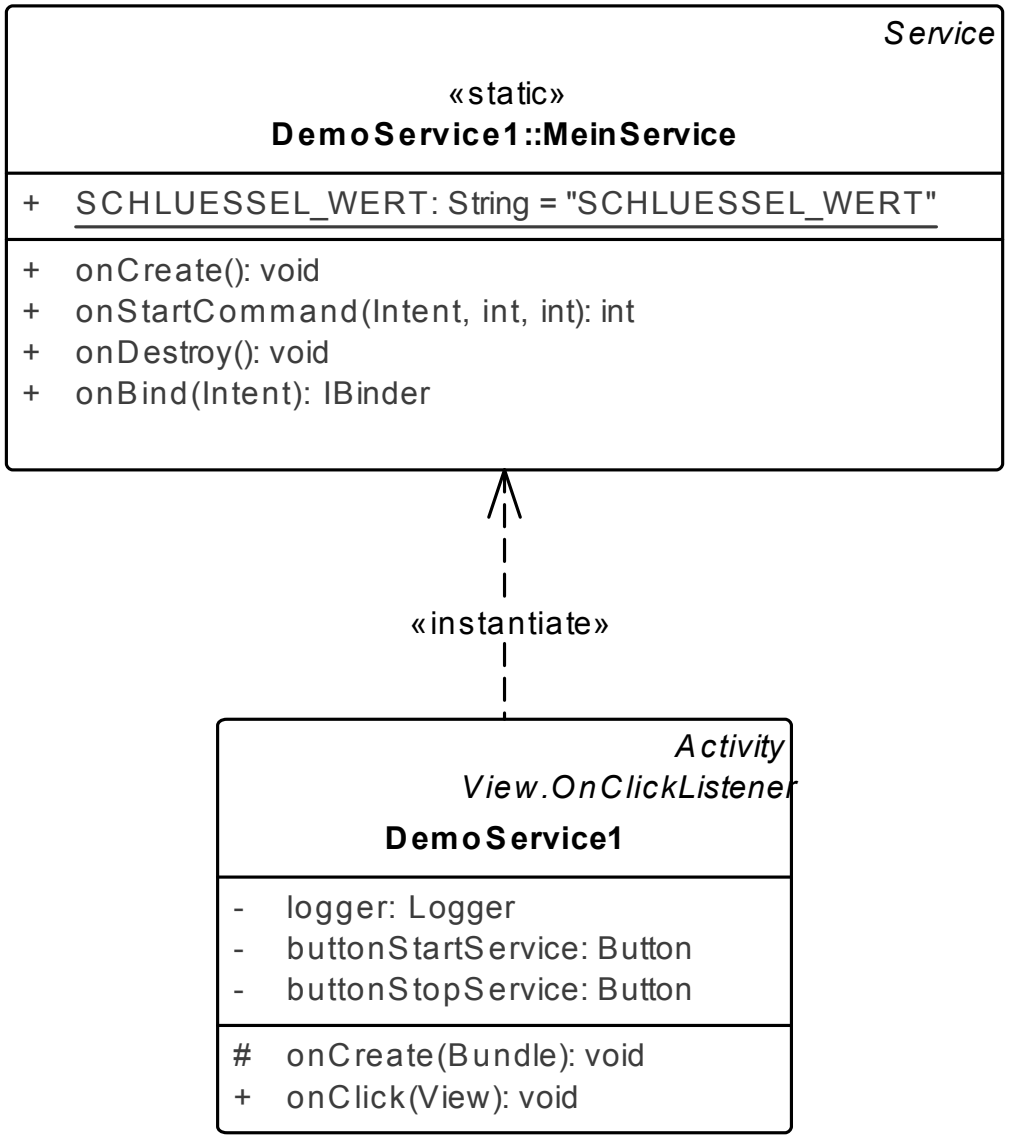
- Datei im Hintergrund herunterladen
- Musik abspielen
- Sensordaten erfassen

Service vs. Thread

- Beide Techniken, Service und Thread, sind dazu geeignet, Aufgaben im Hintergrund auszuführen.
- Im Gegensatz zum Thread ist der Service dafür vorgesehen, die Hintergrundaufgabe auch dann auszuführen, wenn der Benutzer nicht mit der Applikation interagiert.

Inhaltsverzeichnis

- In Beispiel *DemoService1* startet eine Aktivität einen Service,
- übergibt dem Service beim Starten einen Parameter
- und beendet den Service.


Service
«static»
DemoService1::MeinService
+ SCHLUESSEL_WERT: String = "SCHLUESSEL_WERT"
+ onCreate(): void
+ onStartCommand(Intent, int, int): int
+ onDestroy(): void
+ onBind(Intent): IBinder
«instantiate»
Activity
View.OnClickListener
DemoService1
- logger: Logger
- buttonStartService: Button
- buttonStopService: Button
onCreate(Bundle): void
+ onClick(View): void

Ein Service muss im Android-Manifest eingetragen werden:

```
<service
  android:name=".DemoService1$MeinService"
  android:exported="false">
</service>
```

Das Attribut `android:exported` gibt an, ob andere Applikationen auf den Service zugreifen dürfen.

```java
package de.tmahr.android.unterricht.demos;

import android.app.Activity;
import android.app.Service;
import android.content.Intent;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import de.tmahr.android.unterricht.demos.logger.Logger;

public class DemoService1 extends Activity implements View.OnClickListener
{
    private Logger logger;
    private Button buttonStartService;
    private Button buttonStopService;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_demo_service1);

        TextView textViewLogger = (TextView)findViewById(R.id.textViewLogger);
        logger = new Logger(this.getClass().getSimpleName(),textViewLogger,"");

        buttonStartService = (Button)findViewById(R.id.buttonServiceStart);
        buttonStartService.setOnClickListener(this);
        buttonStopService= (Button)findViewById(R.id.buttonStopNutzen);
        buttonStopService.setOnClickListener(this);
    }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService1.java (Ausschnitt)

```
36      @Override
37      public void onClick(View v)
38      {
39          if(v==buttonStartService)
40          {
41              logger.log("onClick: Schalter \"" + buttonStartService
                    .getText() + "\"");
42              Intent intent = new Intent(this, MeinService.class);
43              intent.putExtra(MeinService.SCHLUESSEL_WERT, 12);
44              startService(intent);
45          }
46          else if(v==buttonStopService)
47          {
48              logger.log("onClick: Schalter \"" + buttonStopService.
                    getText() + "\"");
49              Intent intent = new Intent(this, MeinService.class);
50              stopService(intent);
51          }
52      }
```

Dieser Intent gibt den zu startenden Service an.

Übergabeparameter an den Service.

Startet den Service.

Dieser Intent gibt den zu beendenden Service an.

Beendet den Service, unabhängig davon wie oft `startService` aufgerufen wurde.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService1.java (Ausschnitt)

```
54      public static class MeinService extends Service
55      {
56          public static String SCHLUESSEL_WERT = "SCHLUESSEL_WERT";
57
58          @Override
59          public void onCreate()
60          {
61              super.onCreate();
62              Log.d(DemoService1.class.getSimpleName(), getClass().
                    getSimpleName()+".onCreate");
63          }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService1.java (Ausschnitt)

Der Service erweitert die abstrakte Klasse `Service`.

Über diesen Schlüssel wird dem Service beim Starten ein Wert übergeben.

Erstellt den Service

```
65          @Override
66          public int onStartCommand(Intent intent, int flags, int
                startId)
67          {
68              StringBuilder sb = new StringBuilder();
69              sb.append(getClass().getSimpleName()).append(".
                    onStartCommand: ");
70
71              if(intent!=null)
72              {
73                  int wert = intent.getIntExtra(SCHLUESSEL_WERT,0);
74                  sb.append("wert=").append(wert);
75              }
76              sb.append(", startId=").append(startId);
77
78              Log.d(DemoService1.class.getSimpleName(),sb.toString()
                    );
79              return START_STICKY;
80          }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService1.java (Ausschnitt)

Wird jedesmal aufgerufen, wenn ein Aufrufer des Services `startService` aufruft.

Der Intent kann `null` sein, falls der Prozess neu gestartet wird.

Liest den vom Aufrufer übergebenen Wert aus.

Gibt die Anzahl der Aufrufe von `startService` an.

Der Rückgabewert legt die Art und Weise des Neustarts fest: Falls der Prozess beendet wird, versucht Android den Service nach einem Neustart des Prozesses neu zu starten.

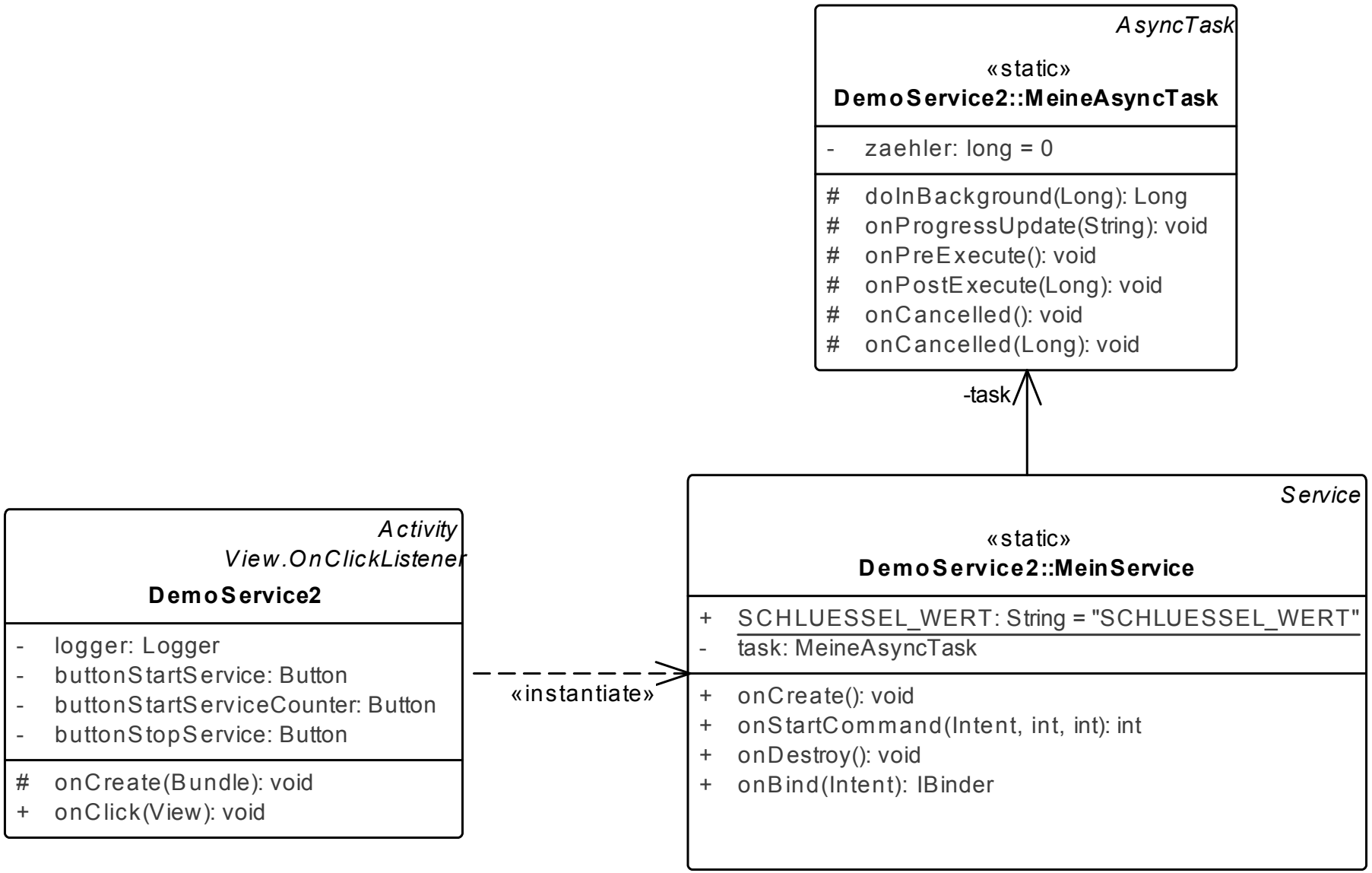
```
82          @Override
83          public void onDestroy()
84          {
85              Log.d(DemoService1.class.getSimpleName(), getClass().
                    getSimpleName()+".onDestroy");
86              super.onDestroy();
87          }
88
89          @Override
90          public IBinder onBind(Intent intent)
91          {
92              Log.d(DemoService1.class.getSimpleName(), getClass().
                    getSimpleName()+".onBind");
93              return null;
94          }
95      }
96  }
```

Wird aufgerufen, wenn der Service beendet wird.

Abstrakte Funktion muss implementiert werden, wird aber in diesem Beispiel nicht benötigt.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService1.java (Ausschnitt)

- Der Service aus dem letzten Beispiel hat noch keine Aufgabe erledigt.
- Daher wird im nächsten Beispiel, *DemoService2*, der Service um eine Hintergrundaufgabe ergänzt.
- Diese Aufgabe wird in einem extra Thread ausgeführt.


AsyncTask
«static»
DemoService2::MeineAsyncTask
- zaehler: long = 0
doInBackground(Long): Long
onProgressUpdate(String): void
onPreExecute(): void
onPostExecute(Long): void
onCancelled(): void
onCancelled(Long): void
-task
Service
«static»
DemoService2::MeinService
+ SCHLUESSEL_WERT: String = "SCHLUESSEL_WERT"
- task: MeineAsyncTask
+ onCreate(): void
+ onStartCommand(Intent, int, int): int
+ onDestroy(): void
+ onBind(Intent): IBinder
Activity
View.OnClickListener
DemoService2
- logger: Logger
- buttonStartService: Button
- buttonStartServiceCounter: Button
- buttonStopService: Button
onCreate(Bundle): void
+ onClick(View): void
«instantiate»

```
39
40      @Override
41      public void onClick(View v)
42      {
43          Intent intent = new Intent(this, MeinService.class);
44          if(v==buttonStartService)
45          {
46              logger.log("onClick: Schalter \"" +
                    buttonStartService.getText() + "\"");
47              intent.putExtra(MeinService.SCHLUESSEL_WERT, 0);
48              startService(intent);
49          }
50          else if(v==buttonStartServiceCounter)
51          {
52              logger.log("onClick: Schalter \"" +
                    buttonStartServiceCounter.getText() + "\"");
53              intent.putExtra(MeinService.SCHLUESSEL_WERT, 10);
54              startService(intent);
55          }
56          else if(v==buttonStopService)
57          {
58              logger.log("onClick: Schalter \"" +
                    buttonStopService.getText() + "\"");
59              stopService(intent);
60          }
61      }
```

Dieser Intent gibt den betreffenden Service an.

Übergibt den Wert 0 an den Service. Der Service durchläuft die Schleife des Hintergrund-Threads bis zum Aufruf von `stopService`.

Übergibt den Wert 10 an den Service. Der Service durchläuft die Schleife des Hintergrund-Threads 10 mal.

Beendet den Service, unabhängig davon wie oft `startService` aufgerufen wurde.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService2.java (Ausschnitt)

```
63      public static class MeinService extends Service
64      {
65          public static String SCHLUESSEL_WERT = "
                SCHLUESSEL_WERT";
66          private MeineAsyncTask task;
67
68          @Override
69          public void onCreate()
70          {
71              super.onCreate();
72              Log.d(DemoService2.class.getSimpleName(), getClass
                    ().getSimpleName()+".onCreate");
73          }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService2.java (Ausschnitt)

Der Service nutzt für die nebenläufige Ausführung der Berechnung eine erweiterte `AsyncTask`.

```
75          @Override
76          public int onStartCommand(Intent intent, int flags, int
                startId)
77          {
78              int wert=0;
79              StringBuilder sb = new StringBuilder();
80              sb.append(getClass().getSimpleName()).append(".
                    onStartCommand: ");
81              if(intent!=null)
82              {
83                  wert = intent.getIntExtra(SCHLUESSEL_WERT,0);
84                  sb.append("wert=").append(wert);
85              }
86              sb.append(", startId=").append(startId);
87              Log.d(DemoService2.class.getSimpleName(), sb.toString());
88
89              if(task!=null)
90              {
91                  task.cancel(false);
92              }
93              task = new MeineAsyncTask();
94              task.execute(Long.valueOf(wert));
95
96              return START_STICKY;
97          }
98
99          @Override
100         public void onDestroy()
101         {
102             Log.d(DemoService2.class.getSimpleName(), getClass().
                    getSimpleName()+".onDestroy");
103             task.cancel(false);
104             super.onDestroy();
105         }
```

Liest den vom Aufrufer übergebenen Wert aus.

Falls die AsyncTask bereits ausgeführt wird, wird sie hier beendet.

Instanziiert die erweiterte `AsyncTask`.

Startet die Nebenläufigkeit mit dem vom Aufrufer übergebenen Wert. Dieser Wert bestimmt die Ausführung der Nebenläufigkeit.

Beendet die AsyncTask.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService2.java (Ausschnitt)

```
115     static class MeineAsyncTask extends AsyncTask<Long, String,
            Long>
116     {
117         private long zaehler = 0;
118
119         @Override
120         protected Long doInBackground(Long... params)
121         {
122             Log.d(DemoService2.class.getSimpleName(), getClass().
                    getSimpleName()+".doInBackground");
123             Long limit = (params.length>0)? params[0] : null;
124
125             for (;;)
126             {
127                 if (isCancelled()) { break; }
128                 else
129                 {
130                     zaehler++;
131                     publishProgress(String.valueOf(zaehler));
132                     if(limit!=null && limit.equals(zaehler)) {
133                         break;
134                     }
135                     try { Thread.sleep(500); }
136                     catch (InterruptedException e) { e.
                            printStackTrace(); }
137                 }
138             }
139             return zaehler;
140         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoService2.java (Ausschnitt)

Das optionale Argument gibt die Begrenzung des Zählers an. Sobald der Zähler den Wert `param[0]` erreicht hat, beendet sich die Nebenläufigkeit ohne Aufruf von `AsyncTask.cancel()`.

Diese Funktion wird in der Nebenläufigkeit ausgeführt. Sie dient dazu, die in der Nebenläufigkeit berechneten Daten in den Thread der Benutzeroberfläche zu transportieren, um diese dort darzustellen. Innerhalb des Threads der Benutzeroberfläche nimmt die Funktion `onProgressUpdate` die darzustellenden Daten entgegen und stellt zeigt sie an.

Bei Erreichen der Begrenzung wird die Nebenläufigkeit verlassen.

Die Verarbeitung des Threads pausiert 500 ms.

Inhaltsverzeichnis

Übung

Erweitern Sie Beispiel *DemoService2* um folgende Funktionen:

1. Falls der Service gerade im Hintergrund eine Task ausführt, soll der Service bei einem weiteren Aufruf von `onStartCommand` eine weitere Task starten.
2. Optional: Der Benutzer soll die Anzahl der Schleifendurchläufe und die Geschwindigkeit der Task einstellen können.

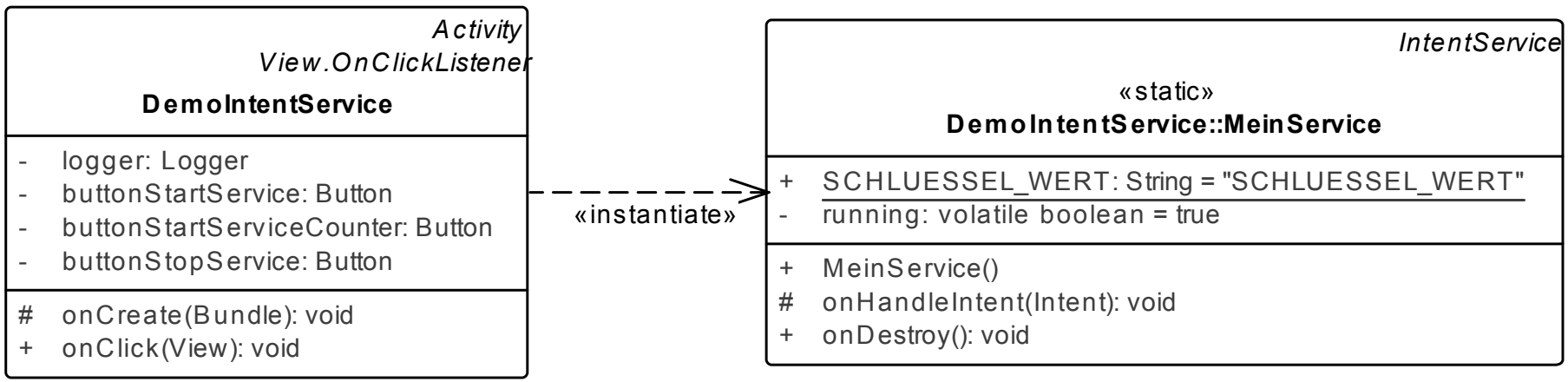
Inhaltsverzeichnis

Ein `IntentService`

- erstellt einen Thread und führt in diesem Thread alle Intents aus, die mittels `onStartCommand` beauftragt wurden,
- puffert die beauftragten Intents,
- beendet den Service, nachdem alle Intents abgearbeitet wurden.

Bei einem `IntentService` muss man nur zwei Funktionen zur Verfügung stellen:

1. Im Konstruktor muss man den Namen des zu erstellenden Threads an den Konstruktor von `IntentService` übergeben.
2. In `void onHandleIntent(Intent intent)` wird der Intent bearbeitet.


Activity
View.OnClickListener
DemoIntentService
- logger: Logger
- buttonStartService: Button
- buttonStartServiceCounter: Button
- buttonStopService: Button
onCreate(Bundle): void
+ onClick(View): void
«instantiate»
IntentService
«static»
DemoIntentService::MeinService
+ SCHLUESSEL_WERT: String = "SCHLUESSEL_WERT"
- running: volatile boolean = true
+ MeinService()
# onHandleIntent(Intent): void
+ onDestroy(): void

```
62      public static class MeinService extends
            IntentService
63      {
64          public static String SCHLUESSEL_WERT = "
                SCHLUESSEL_WERT";
65          private volatile boolean running = true;
66
67          public MeinService()
68          {
69              super(DemoIntentService.class.
                    getSimpleName()+"Thread");
70          }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoIntentService.java
(Ausschnitt)

Der Service erweitert die Basisklasse `IntentService`.

Darüber lässt sich der Thread beenden.

Im Konstruktor wird der Name des Threads festgelegt.

```
72          @Override
73          protected void onHandleIntent(Intent intent)
74          {
75              long zaehler = 0;
76              int maxZaehler = intent.getIntExtra(SCHLUESSEL_WERT
                    ,0);
77              Log.d(DemoIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".onHandleIntent,
                    maxZaehler=" + maxZaehler);
78              running = true;
79
80              while(running)
81              {
82                  Log.d(DemoIntentService.class.getSimpleName(),
                        getClass().getSimpleName()+".onHandleIntent,
                        Intent=" + intent.hashCode() + ", zaehler=" +
                         zaehler);
83                  if(maxZaehler!=0 && zaehler>=maxZaehler) { break
                        ; }
84                  zaehler++;
85
86                  try { Thread.sleep(500); }
87                  catch (InterruptedException e)
88                  {
89                      e.printStackTrace();
90                      Thread.currentThread().interrupt();
91                  }
92              }
93          }
```

Wird innerhalb des Threads aufgerufen. Nach dem Verlassen dieser Funktion wird der Service beendet.

Falls `maxZaehler=0` kann man die Schleife nur über `running=false` verlassen.

Falls `maxZaehler>0` wird die Schleife verlassen, sobald der Zähler die Schwelle erreicht hat.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoIntentService.java (Ausschnitt)

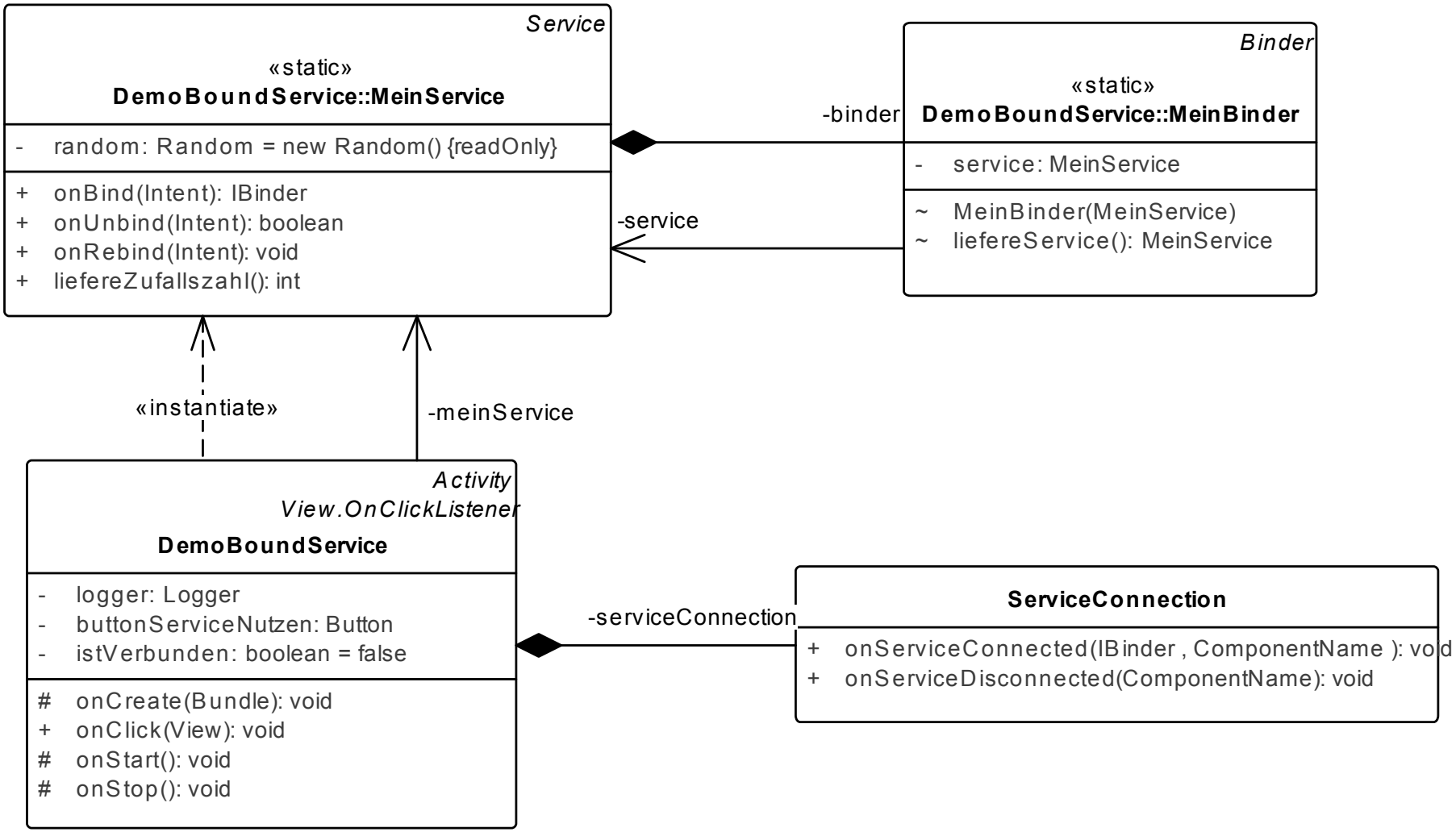
```
95          @Override
96          public void onDestroy()
97          {
98              Log.d(DemoIntentService.class.
                    getSimpleName(), getClass().
                    getSimpleName()+".onDestroy");
99              running = false;
100             super.onDestroy();
101         }
102     }
103 }
```

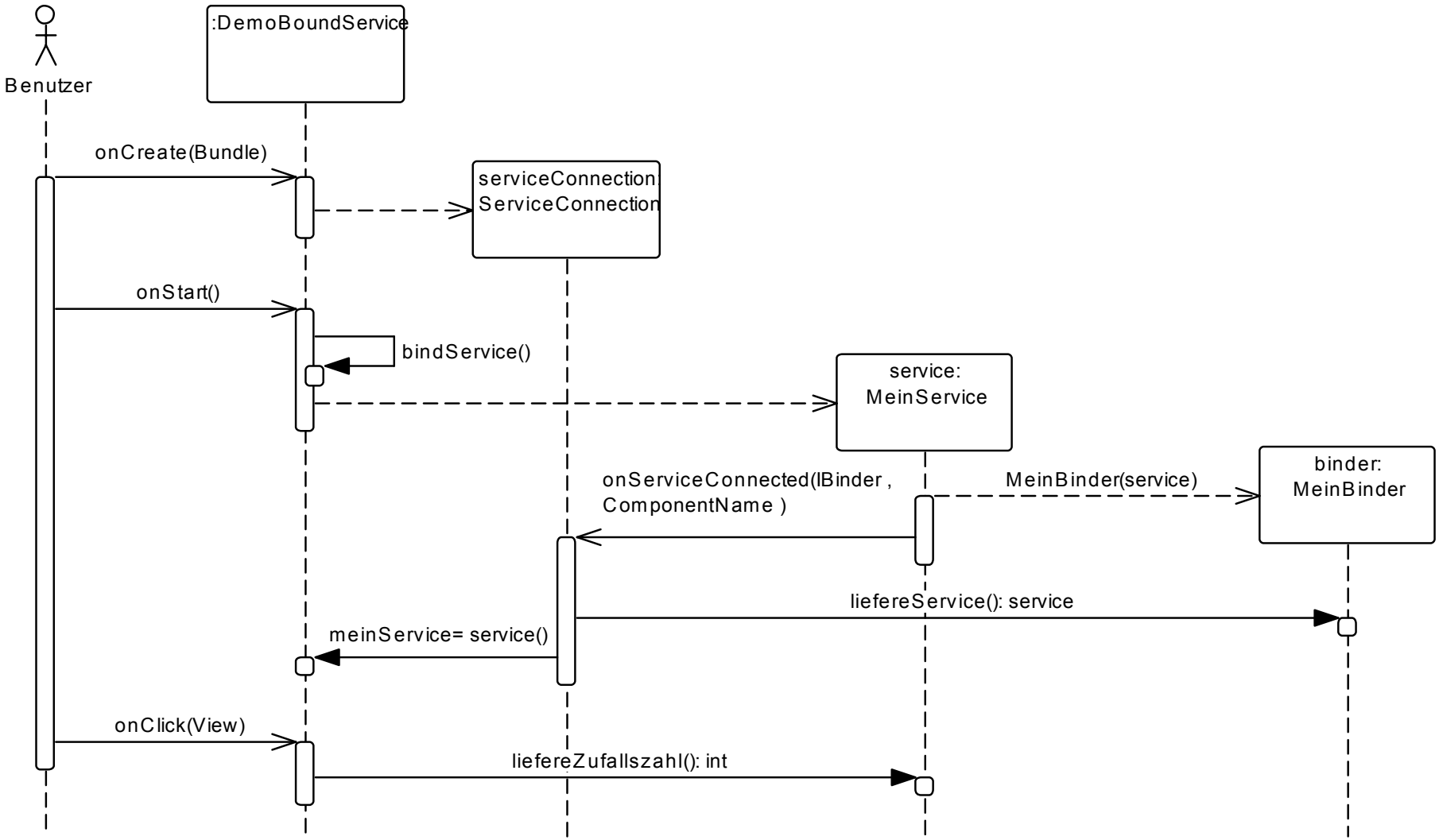
Stellt sicher, dass die im Thread ausgeführte Schleife verlassen werden kann.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoIntentService.java
(Ausschnitt)

Inhaltsverzeichnis

- Eine Applikationskomponente, z.B. eine Aktivität kann sich mittels des Befehls `bindService` an einen Service binden und mit dem Service interagieren.
- Im folgenden Beispiel wird eine Aktivität an einen Service gebunden und fragt von diesem Service Zufallszahlen ab.


Service
«static»
DemoBoundService::MeinService
- random: Random = new Random() {readOnly}
+ onBind(Intent): IBinder
+ onUnbind(Intent): boolean
+ onRebind(Intent): void
+ liefereZufallszahl(): int
Binder
«static»
DemoBoundService::MeinBinder
- service: MeinService
~ MeinBinder(MeinService)
~ liefereService(): MeinService
-binder
-service
«instantiate»
-meinService
Activity
View.OnClickListener
DemoBoundService
- logger: Logger
- buttonServiceNutzen: Button
- istVerbunden: boolean = false
onCreate(Bundle): void
+ onClick(View): void
# onStart(): void
# onStop(): void
-serviceConnection
ServiceConnection
+ onServiceConnected(IBinder , ComponentName): void
+ onServiceDisconnected(ComponentName): void


Benutzer
:DemoBoundService
onCreate(Bundle)
serviceConnection
ServiceConnection
onStart()
bindService()
service:
MeinService
onServiceConnected(IBinder ,
ComponentName)
MeinBinder(service)
binder:
MeinBinder
liefereService(): service
meinService= service()
onClick(View)
liefereZufallszahl(): int

```java
package de.tmahr.android.unterricht.demos;

import android.app.Activity;
import android.app.Service;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Binder;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import java.util.Random;

import de.tmahr.android.unterricht.demos.logger.Logger;

public class DemoBoundService extends Activity implements View.
    OnClickListener
{
    private Logger logger;
    private Button buttonServiceNutzen;
    private MeinService meinService;
    private boolean istVerbunden = false;
    private ServiceConnection serviceConnection;
```

Über diesen Schalter kann der Benutzer den Service nutzen.

Über dieses Objekt wird die Aktivität darüber informiert, dass sich die Aktivität mit einem Service verbunden hat.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java (Ausschnitt)

```
29      @Override
30      protected void onCreate(Bundle savedInstanceState)
31      {
32          super.onCreate(savedInstanceState);
33          setContentView(R.layout.activity_demo_bound_service);
34          setTitle(getClass().getSimpleName());
35
36          TextView textViewLogger = (TextView) findViewById(R.id.
                textViewLogger);
37          logger = new Logger(this.getClass().getSimpleName(),
                textViewLogger, "");
38
39          buttonServiceNutzen = (Button) findViewById(R.id.
                buttonServiceStart);
40          buttonServiceNutzen.setOnClickListener(this);
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java (Ausschnitt)

```
42          serviceConnection = new ServiceConnection()
43          {
44              @Override
45              public void onServiceConnected(ComponentName name,
                    IBinder binder)
46              {
47                  logger.log("onServiceConnected");
48                  MeinBinder meinBinder = (MeinBinder) binder;
49                  meinService = meinBinder.liefereService();
50                  istVerbunden = true;
51              }
52
53              @Override
54              public void onServiceDisconnected(ComponentName
                    name)
55              {
56                  logger.log("onServiceDisconnected");
57                  meinService = null;
58                  istVerbunden = false;
59              }
60          };
61      }
```

Wird aufgerufen, wenn sich die Aktivität mit dem Service verbunden hat.

Dieser `binder` wurde innerhalb des Service erstellt und ist vom Typ `MeinBinder`.

Der `binder` liefert den Service zurück.

Wird aufgerufen, wenn sich die Aktivität mit dem Service getrennt hat.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java (Ausschnitt)

```
63      @Override
64      public void onClick(View v)
65      {
66          if (v == buttonServiceNutzen)
67          {
68              if (istVerbunden)
69              {
70                  int wert = meinService.
                        liefereZufallszahl();
71                  logger.log("Service liefert " +
                        wert);
72              }
73          }
74      }
```

Hier wird der Service genutzt.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java
(Ausschnitt)

```
76      @Override
77      protected void onStart()
78      {
79          super.onStart();
80          logger.log("onStart");
81          Intent intent = new Intent(this,
                MeinService.class);
82          bindService(intent, serviceConnection,
                Context.BIND_AUTO_CREATE);
83      }
84
85      @Override
86      protected void onStop()
87      {
88          super.onStop();
89          logger.log("onStop");
90          if (istVerbunden)
91          {
92              unbindService(serviceConnection);
93              istVerbunden = false;
94          }
95      }
```

Aktivität mit dem Service verbinden.

Aktivität vom Service trennen.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java
(Ausschnitt)

```
97      public static class MeinService extends Service
98      {
99          private final IBinder binder = new MeinBinder(this);
100         private final Random random = new Random();
101
102         @Override
103         public IBinder onBind(Intent intent)
104         {
105             Log.d(DemoBoundService.class.getSimpleName(), getClass().
                    getSimpleName() + ".onBind");
106             return binder;
107         }
108
109         @Override
110         public boolean onUnbind(Intent intent)
111         {
112             Log.d(DemoBoundService.class.getSimpleName(), getClass().
                    getSimpleName() + ".onUnbind");
113             return super.onUnbind(intent);
114         }
115
116         @Override
117         public void onRebind(Intent intent)
118         {
119             Log.d(DemoBoundService.class.getSimpleName(), getClass().
                    getSimpleName() + ".onRebind");
120             super.onRebind(intent);
121         }
122
123         public int liefereZufallszahl()
124         {
125             Log.d(DemoBoundService.class.getSimpleName(), getClass().
                    getSimpleName() + ".liefereZufallszahl");
126             return random.nextInt(100);
127         }
128     }
```

Der Service erstellt einen `MeinBinder` und übergibt diesem die Referenz auf den Service.

Wenn eine Verbindung erstellt wurde, liefert diese Funktion das `MeinBinder`-Objekt zurück.

Der Service erstellt eine Zufallszahl und liefert diese zurück.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java (Ausschnitt)

```
130     private static class MeinBinder extends Binder
131     {
132         private MeinService service;
133
134         MeinBinder(MeinService service)
135         {
136             this.service = service;
137         }
138
139         MeinService liefereService()
140         {
141             return service;
142         }
143     }
144 }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundService.java
(Ausschnitt)

Die Klasse erweitert `Binder` um eine Referenz auf `MeinService`.

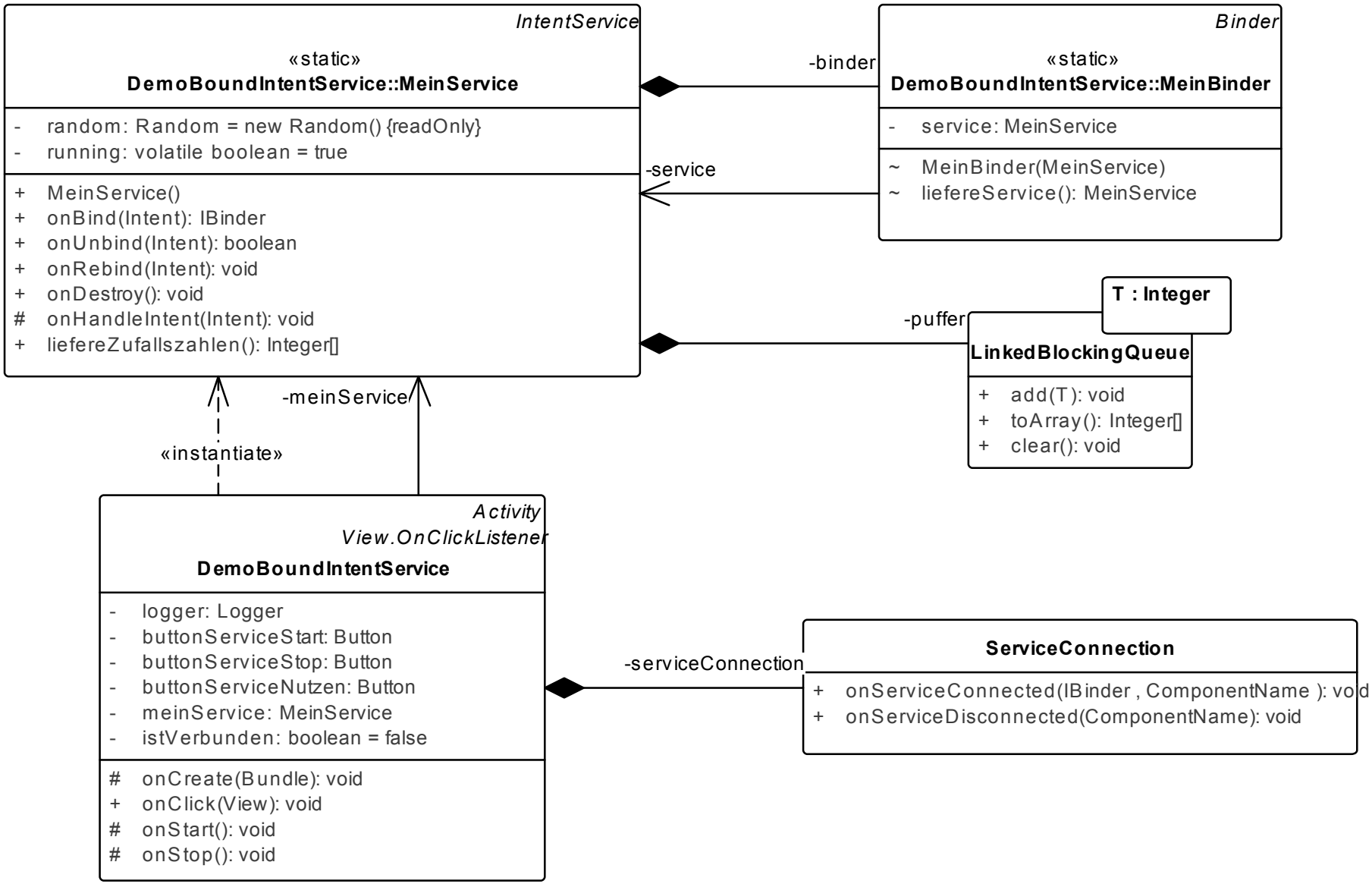
Der Service wird dem Konstruktor übergeben

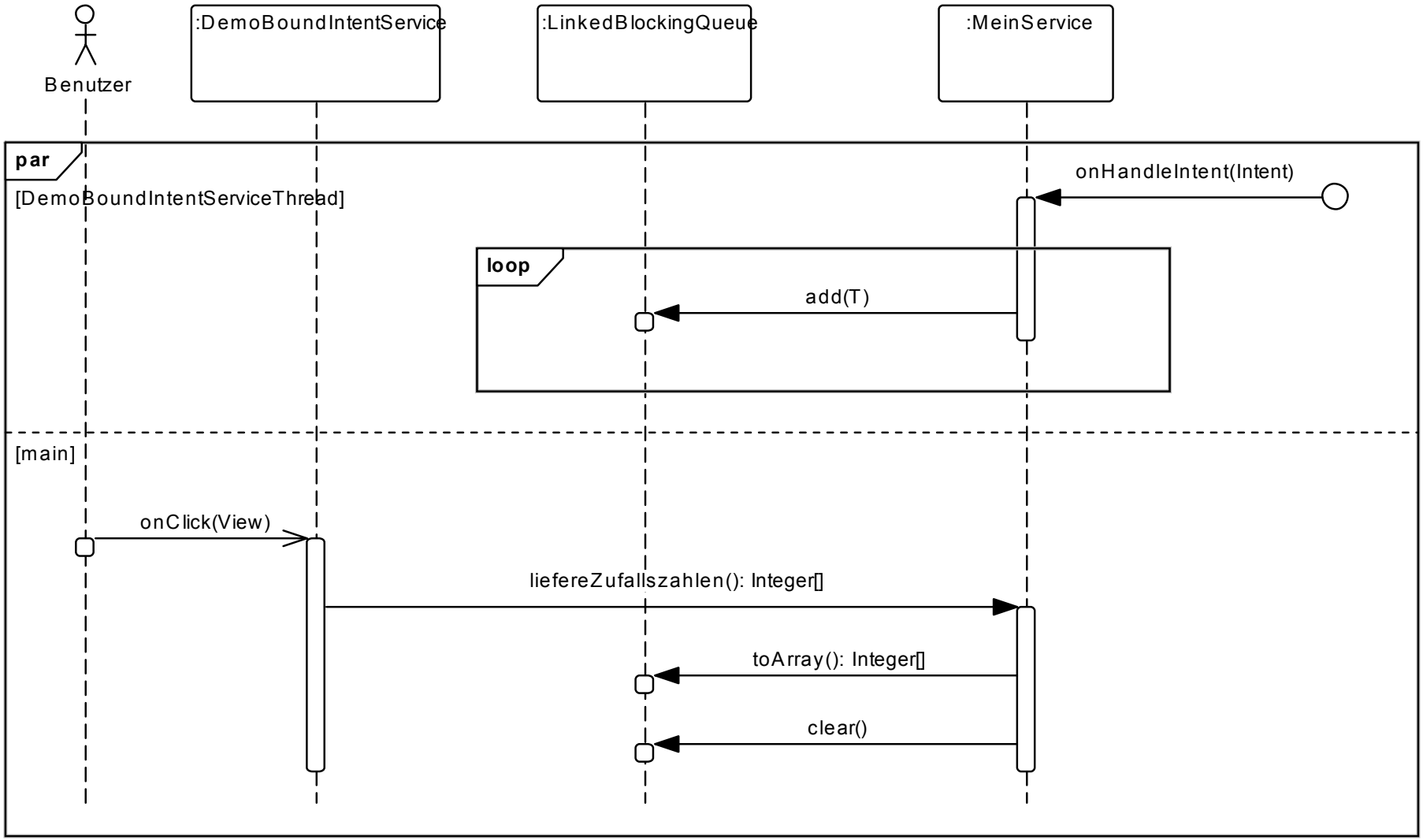
und hier zurückgeliefert.

Inhaltsverzeichnis

Im Gegensatz zum letzten Beispiel soll der Service im folgenden Beispiel nebenläufig arbeiten:

- Eine Aktivität startet einen Service und verbindet sich mit dem Service.
- Der Service erweitert IntentService und erstellt in einem Thread Zufallszahlen.
- Der Benutzer ruft die erstellten Zufallszahlen ab und stellt sie dar. Dabei wird der Zufallszahlenpuffer des Service geleert.
- Da zwei Threads auf die Zufallszahlen zugreifen werden die Zufallszahlen in einer Thread-sicheren `LinkedBlockingQueue` gespeichert.
- Der Service wird erste gestoppt, wenn er ungebunden ist und `stopService` aufgerufen wurde.


IntentService
«static»
DemoBoundIntentService::MeinService
- random: Random = new Random() {readOnly}
- running: volatile boolean = true
+ MeinService()
+ onBind(Intent): IBinder
+ onUnbind(Intent): boolean
+ onRebind(Intent): void
+ onDestroy(): void
# onHandleIntent(Intent): void
+ liefereZufallszahlen(): Integer[]
-binder
Binder
«static»
DemoBoundIntentService::MeinBinder
- service: MeinService
~ MeinBinder(MeinService)
~ liefereService(): MeinService
-service
-puffer
T : Integer
LinkedBlockingQueue
+ add(T): void
+ toArray(): Integer[]
+ clear(): void
-meinService
«instantiate»
Activity
View.OnClickListener
DemoBoundIntentService
- logger: Logger
- buttonServiceStart: Button
- buttonServiceStop: Button
- buttonServiceNutzen: Button
- meinService: MeinService
- istVerbunden: boolean = false
onCreate(Bundle): void
+ onClick(View): void
# onStart(): void
# onStop(): void
-serviceConnection
ServiceConnection
+ onServiceConnected(IBinder , ComponentName): void
+ onServiceDisconnected(ComponentName): void


Benutzer
:DemoBoundIntentService
:LinkedBlockingQueue
:MeinService
par
[DemoBoundIntentServiceThread]
onHandleIntent(Intent)
loop
add(T)
[main]
onClick(View)
liefereZufallszahlen(): Integer[]
toArray(): Integer[]
clear()

```
1  package de.tmahr.android.unterricht.demos;
2
3  import android.app.Activity;
4  import android.app.IntentService;
5  import android.content.ComponentName;
6  import android.content.Context;
7  import android.content.Intent;
8  import android.content.ServiceConnection;
9  import android.os.Binder;
10 import android.os.Bundle;
11 import android.os.IBinder;
12 import android.util.Log;
13 import android.view.View;
14 import android.widget.Button;
15 import android.widget.TextView;
16
17 import java.util.Random;
18 import java.util.concurrent.LinkedBlockingQueue;
19
20 import de.tmahr.android.unterricht.demos.logger.Logger;
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

```
22 public class DemoBoundIntentService extends Activity
       implements View.OnClickListener
23 {
24     private Logger logger;
25     private Button buttonServiceStart;
26     private Button buttonServiceStop;
27     private Button buttonServiceNutzen;
28     private MeinService meinService;
29     private boolean istVerbunden = false;
30     private ServiceConnection serviceConnection;
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java
(Ausschnitt)

Über diesen Schalter kann der Benutzer den Service starten.

Über diesen Schalter kann der Benutzer den Service beenden.

Über diesen Schalter kann der Benutzer den Service nutzen.

Über dieses Objekt wird die Aktivität darüber informiert, dass sich die Aktivität mit einem Service verbunden hat.

```
32      @Override
33      protected void onCreate(Bundle savedInstanceState)
34      {
35          super.onCreate(savedInstanceState);
36          setContentView(R.layout.activity_demo_bound_intent_service);
37          setTitle(getClass().getSimpleName());
38
39          TextView textViewLogger = (TextView) findViewById(R.id.textViewLogger);
40          logger = new Logger(this.getClass().getSimpleName(), textViewLogger, ""
                );
41
42          buttonServiceStart = (Button) findViewById(R.id.buttonServiceStart);
43          buttonServiceStart.setOnClickListener(this);
44          buttonServiceStop = (Button) findViewById(R.id.buttonServiceStop);
45          buttonServiceStop.setOnClickListener(this);
46          buttonServiceNutzen = (Button) findViewById(R.id.buttonServiceNutzen);
47          buttonServiceNutzen.setOnClickListener(this);
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

```
49          serviceConnection = new ServiceConnection()
50          {
51              @Override
52              public void onServiceConnected(ComponentName
                    className, IBinder service)
53              {
54                  logger.log("onServiceConnected");
55                  MeinBinder meinBinder = (MeinBinder) service;
56                  meinService = meinBinder.liefereService();
57                  istVerbunden = true;
58              }
59
60              @Override
61              public void onServiceDisconnected(ComponentName
                    arg0)
62              {
63                  logger.log("onServiceDisconnected");
64                  meinService = null;
65                  istVerbunden = false;
66              }
67          };
68      }
```

Wird aufgerufen, wenn sich die Aktivität mit dem Service verbunden hat.

Dieser `binder` wurde innerhalb des Service erstellt und ist vom Typ `MeinBinder`.

Der `binder` liefert den Service zurück.

Wird aufgerufen, wenn sich die Aktivität mit dem Service getrennt hat.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

```
70      @Override
71      public void onClick(View v)
72      {
73          if(v==buttonServiceStart)
74          {
75              logger.log("onClick: Schalter \"" + buttonServiceStart.getText
                    () +"\"");
76              Intent intent = new Intent(this, MeinService.class);
77              startService(intent);
78          }
79          if(v==buttonServiceStop)
80          {
81              logger.log("onClick: Schalter \"" + buttonServiceStop.getText
                    () + "\"");
82              Intent intent = new Intent(this, MeinService.class);
83              stopService(intent);
84          }
85          else if(v==buttonServiceNutzen)
86          {
87              logger.log("onClick: Schalter \"" + buttonServiceNutzen.
                    getText() + "\"");
88              if(istVerbunden)
89              {
90                  Integer[] werte = meinService.liefereZufallszahlen();
91                  StringBuilder sb = new StringBuilder();
92                  for(Integer i : werte)
93                  {
94                      sb.append(i).append(" ");
95                  }
96                  logger.log("Service liefert " + sb.toString());
97              }
98          }
99      }
```

Service starten.

Service beenden.

Zufallszahlen abholen und darstellen.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

```
101     @Override
102     protected void onStart()
103     {
104         super.onStart();
105         logger.log("onStart");
106         Intent intent = new Intent(this,
                MeinService.class);
107         bindService(intent, serviceConnection,
                Context.BIND_AUTO_CREATE);
108     }
109
110     @Override
111     protected void onStop()
112     {
113         super.onStop();
114         logger.log("onStop");
115         if (istVerbunden)
116         {
117             unbindService(serviceConnection);
118             istVerbunden = false;
119         }
120     }
```

Aktivität mit dem Service verbinden.

Aktivität vom Service trennen.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java
(Ausschnitt)

```
122     public static class MeinService extends IntentService
123     {
124         private final IBinder binder = new MeinBinder(this);
125         private final Random random = new Random();
126         private volatile boolean running = true;
127         private LinkedBlockingQueue<Integer> puffer = new
                LinkedBlockingQueue<>();
128
129         public MeinService()
130         {
131             super(DemoBoundIntentService.class.getSimpleName()+"Thread
                    ");
132         }
133
134         @Override
135         public IBinder onBind(Intent intent)
136         {
137             Log.d(DemoBoundIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".onBind");
138             return binder;
139         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

Der Service erstellt einen `MeinBinder` und übergibt diesem die Referenz auf den Service.

Thread-sicheren Puffer für die Zufallszahlen anlegen.

Wenn eine Verbindung erstellt wurde, liefert diese Funktion das `MeinBinder`-Objekt zurück.

```
141         @Override
142         public boolean onUnbind(Intent intent)
143         {
144             Log.d(DemoBoundIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".onUnbind");
145             return super.onUnbind(intent);
146         }
147
148         @Override
149         public void onRebind(Intent intent)
150         {
151             Log.d(DemoBoundIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".onRebind");
152             super.onRebind(intent);
153         }
154
155         @Override
156         public void onDestroy()
157         {
158             Log.d(DemoBoundIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".onDestroy");
159             running = false;
160             super.onDestroy();
161         }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

Stellt sicher, dass die im Thread ausgeführte Schleife verlassen werden kann.

```
163         @Override
164         protected void onHandleIntent(Intent intent)
165         {
166             Log.d(DemoBoundIntentService.class.getSimpleName(),
                   getClass().getSimpleName()+".onHandleIntent");
167             running = true;
168
169             while(running)
170             {
171                 int wert = random.nextInt(1000);
172                 puffer.add(wert);
173                 Log.d(DemoBoundIntentService.class.getSimpleName(),
                       getClass().getSimpleName()+".onHandleIntent, anzahl
                       =" + puffer.size() + ", wert=" + wert);
174
175                 try { Thread.sleep(500); }
176                 catch (InterruptedException e)
177                 {
178                     e.printStackTrace();
179                     Thread.currentThread().interrupt();
180                 }
181             }
182         }
```

Diese Schleife läuft nebenläufig.

Erstellt eine Zufallszahl.

und legt sie im Puffer ab.

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

```
184         public Integer[] liefereZufallszahlen()
185         {
186             Log.d(DemoBoundIntentService.class.getSimpleName(),
                    getClass().getSimpleName()+".liefereZufallszahlen");
187             Integer[] ret = puffer.toArray(new Integer[0]);
188             puffer.clear();
189             return ret;
190         }
191     }
192
193     private static class MeinBinder extends Binder
194     {
195         private MeinService service;
196
197         MeinBinder(MeinService service)
198         {
199             this.service = service;
200         }
201
202         MeinService liefereService()
203         {
204             return service;
205         }
206     }
207 }
```

Demos/app/src/main/java/de/tmahr/android/unterricht/demos/DemoBoundIntentService.java (Ausschnitt)

Der Service liefert die gepufferten Zufallszahlen zurück und löscht den Puffer.

Zufallszahlen in Array umwandeln.

Puffer löschen.

Die Klasse erweitert `Binder` um eine Referenz auf `MeinService`.

Der Service wird dem Konstruktor übergeben

und hier zurückgeliefert.

Inhaltsverzeichnis

Übung: Primzahlen

Erstellen Sie eine Applikation zur Berechnung von Primzahlen:

1. Der Benutzer legt den Zahlenbereich fest, in dem nach Primzahlen gesucht werden soll.
2. Die Primzahlensuche soll von einem Service angeboten werden.
3. Optional: Gleichzeitig sollen mehrere Primzahlensuchen durchgeführt werden können.