# Asynchrone Verarbeitung

# Inhaltsübersicht



1. Einführung in die Entwicklung mobiler Anwendungen
2. Erste grafische Oberflächen und Benutzerinteraktionen
3. Weiterführende Konzepte mobiler Plattformen
4. Standorbezogene Dienste, Sensoren und Kamera
5. Dauerhaftes Speichern von Daten (Persistenz)
6. Responsive Design, Weiterführende Interaktionsmuster
7. Asynchrone Verarbeitung

# Main-Thread (auch UI-Thread genannt)



- Der Main-Thread ist der Hauptausführungsstrang einer Anwendung
- Der Main-Thread ist verantwortlich für das Management des User Interfaces:
  - Verteilen von Benutzerevents an die entsprechenden Komponenten (zB Touch-Gesten)
  - Darstellung der Benutzeroberfläche
- Zwei wichtige “Regeln” für die Android Entwicklung:
  1. Den Main-Thread nicht unnötig blockieren durch langwierige Verarbeitungen
  2. UI-Elemente dürfen nicht außerhalb des Main-Threads bearbeitet werden (UI-Thread)

# Threading Beispiel 1



```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

- Ziel des Beispiels:
  - Auslagern einer langwierigen Verarbeitung in einen Hintergrundthread (Download eines Bildes)
  - Nach dem Download soll das Bild angezeigt werden.
- Regel 1: Langwierige Verarbeitung auslagern (OK)
- Regel 2: Zugriff auf UI-Element außerhalb des Main-Threads (Verstoß)

# Threading Beispiel 2



```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png")
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

- UI Verarbeitungslogik kann wieder zurück an den Main-Thread gegeben werden:
  - View.post(Runnable)
  - View.postDelayed(Runnable,long)
  - Activity.runOnUiThread(Runnable)

# AsyncTask



- Der Beispielcode aus dem vorhergehenden Beispiel funktioniert, ist aber sehr sperrig zu lesen und schreiben
- AsyncTask stellt eine einfache Abstraktion dar, um ähnliche Problemstellungen eleganter zu lösen
- AsyncTask lagert langwierige Verarbeitungen in den Hintergrund aus (siehe ***doInBackground()***) und gibt das Ergebniss wieder an den Main-Thread zurück (siehe ***onPostExecute()***)
- AsynTask sollte für kurzweiligere Aufgaben verwendet werden (einige Sekunden)

# Threading Beispiel 3



```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

- Erweiterung der Klasse AsyncTask:
  - Die langwierige Verarbeitung wird in den Hintergrund ausgelagert
  - Die Manipulation der UI wird wieder an den Main-Thread übergeben

# Anwenden von AsyncTask



- AsyncTask wird über 3 Generics definiert
  - Params: Übergabeparameter für den Hintergrundthread
  - Progress: Werte welche kontinuierlich über den Verlauf informieren
  - Result: Das Ergebnis der Verarbeitung
- Nicht alle Generics müssen gesetzt sein, void kann ebenfalls als Datentyp angegeben werden
- AsyncTask wird über die Methode ***execute(Params)*** gestartet
- Das Ende des AsyncTasks wird über das Ereignis ***onPostExecute(Result)*** eingeleitet, dieses wird innerhalb des Main-Threads ausgeführt
- Der Verarbeitungsverlauf wird über Ereignisse an die Methode ***onProgressUpdate(Progress)*** wiedergegeben

# AsyncTask Beispiel



```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Deklaration der Generics für Params, Progress und Result

Diese Methode wird im Hintergrundthread ausgeführt

Der Verarbeitungsverlauf (Progress) wird an diese Methode weitergeleitet

Letztlich wird das Ergebnis (Result) innerhalb dieser Methode verarbeitet

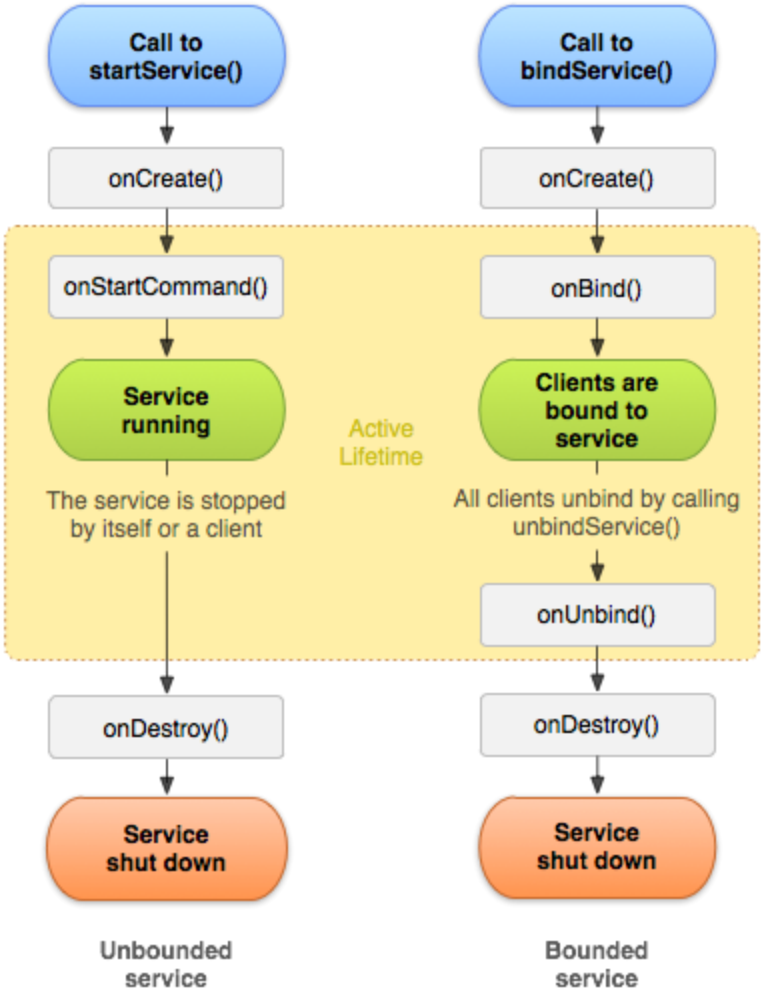
# Service Komponente



- Zweierlei Services sind zu unterscheiden:
  - Started Services: Arbeit, welche in den Hintergrund ausgelagert werden soll, solange bis explizit gestoppt wird oder sich selbst stoppt: zB Abspielen eines MP3s
    - ***Context.startService(Intent)*** zum Starten des Services
    - ***Context.stopService(Intent)*** zum expliziten Stoppen des Services
    - ***Service.stopSelf()***, der Service kann sich selbst stoppen
  - Bound Services: Ein Hintergrunddienst, welcher dem Client-Server Modell entspricht und mehrere Clients bedienen kann.
    - ***Context.bindService(Intent,ServiceConnection,int)*** Clientseitiger Zugriff auf den Service
    - ***Context.unbindService(ServiceConnection)*** Verbindung Auflösen

# Started vs Bound Services (Lebenszyklus)






- Alle Services leiten den Beginn durch ***onCreate()*** und das Ende ihres Lebenszykluses durch ***onDestroy()*** ein
- ***onStartCommand()*** wird nur für Started Services aufgerufen
- ***onBind()*** und ***onUnbind()*** nur von Bound Services

# Ausführung des Services



- Services werden standardmäßig nicht in einem eigenen Thread ausgeführt
- Services werden standardmäßig nicht in einem eigenen Prozess ausgeführt

- Ausführung in eigenem Thread:
  - Implementierung des Threadhandlings im Service selbst
  - Implementierung der abgeleiteten Klasse IntentService (ähnlich AsyncTask einfacher)
- Ausführung in eigenem Prozess:
  - Eintrag im AndroidManifest.xml

```
<service android:name=".app.MessengerService"
        android:process=":remote" />
```

# Beispiel IntentService



```
public class IntentServiceExample extends IntentService {

    public IntentServiceExample() {
        super("IntentServiceExample");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i("SERVICE_EXAMPLE","Start Processing in the Backgro  d...");

        // DO WORK HERE!
        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {}

        Log.i("        E_EXAMPLE","Finished Processing in the Background...")

    }
}
```

IntentService vereinfacht die Ausführung im Hintergrund (Thread)

Die Methode ***onHandleIntent()*** muss implementiert werden. Innerhalb dieser Methode soll die tatsächlich Arbeit geleistet werden

Beispielhaft wird nur 4 Sekunden geschlafen...

# Inter Process Communication (IPC)



- Services können auch innerhalb eines eigenen Prozesses Ablaufen
- Die Kommunikation zwischen zwei unabhängigen Prozessen funktioniert über Messages in Android
  - Messenger: fungieren als Kommunikationskanäle zwischen den Komponenten
  - Messages: sind Objekte mit Schlüssel/Wert Paaren, welche zwischen Komponenten (in unterschiedlichen Prozesses) ausgetauscht werden könnnen.
  - ServiceConnection: Eine ServiceConnection stellt einen ***Binder*** zur Verfügung
  - Binder: Stellt Verbindung zum Service zur Verfügung

# IPC Beispiel: Service Implementierung



```
private final Messenger messenger = new Messenger(new MessageHandler());

private class MessageHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        Bundle b = msg.getData();
        if (b.containsKey("greet")) {
            String greet = b.getString("greet");

            Messenger sender = msg.replyTo; // Who send the message to us
            Message answer = Message.obtain(); // Create an answer for the sender of the message

            Bundle b2 = new Bundle();
            b2.putString("greet_back",greet + " auch zurück vom MessagingService");
            answer.setData(b2);

            try {
                sender.send(answer); // send back t
            } catch (RemoteException e) {}
        }
    }
}

@Override
public IBinder onBind(Intent intent) {
    return messenger.getBinder();
}
```

Messenger wird initialisert mit Handler zur Verarbeitung von Nachrichten von Clients

Der Handler erhält eine Nachricht vom Client. Die Nachricht enthält eine Gruß. Es wird eine Antwort an den Client zurückgesendet. **replyTo** der Message kann dazu verwendet werden

Ein Binder wird zurückgegeben, welcher von Clients zum Nachrichtenaustausch verwendet werden kann

# IPC Beispiel: Activity Implementierung 1



```
Messenger serviceMessenger;
Messenger messenger = new Messenger(new MessagingHandler());

private ServiceConnection messageConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder service
        serviceMessenger = new Messenger(service);
    }

    @Override
    public void onServiceDisconnected(ComponentName compone
    }
};

public class MessagingHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        Bundle b = msg.getData();
        if (b.containsKey("greet_back")) {
            String greetBack = b.getString("greet_back");
            answer.setText(greetBack);
        }
    }
};
```

In der Activity sind zwei Messenger verfügbar. Einer zum Versenden von Nachrichten, einer zum Erhalten von Nachrichten

Die Verbindung zum Service wird über den Binder hergestellt

Erhaltene Nachrichten vom Service werden in einer TextView angezeigt

# IPC Beispiel: Activity Implementierung 2



```
private TextView answer; // Answers from message service

public void bindMessagingService(View v) {
    bindService(new Intent(this,MessagingService.class),messageConnection, Context.BIND_AUTO_CREATE);
}

public void unbindMessagingService(View v) {
    unbindService(messageConnection);
}

public void greet(View v) {
    Message greet = Message.obtain();

    Bundle b = new Bundle();
    b.putString("greet", "Hallo");

    greet.setData(b);
    greet.replyTo = messenger;

    try {
        serviceMessenger.send(greet);
    } catch (RemoteException e) {}

}
```

Verbindung zum Service Anmelden bzw. Abmelden

Ein Gruß wird vorbereitet, welcher an den Service versendet werden soll. Über replyTo wird dem Service mitgeteilt, an wen seine Antwort übergeben werden soll

# Thread oder Service?



- Service
  - Eine Komponente, welche im Hintergrund ausgeführt werden kann, auch wenn der Benutzer nicht mit der Anwendung interagiert
- Thread (AsyncTask, Handler, ...)
  - Arbeit welche ausserhalb des Main-Thread bearbeitet werden muss, aber nur während der Benutzer mit der Anwendung interagiert

- Wichtiger Hinweis:
  - Standardmäßig werden Services im Mainthread ausgeführt. Über das Manifest kann dies verändert werden

# „Take-Away“ für diese Einheit



- Regeln für den Main-Thread
- Arbeiten mit Hintergrundprozessen (AsynTask)
- Auslagern von Arbeit in den Hintergrund IntentService
- Nachrichtenaustausch zwischen Prozessen (IPC)