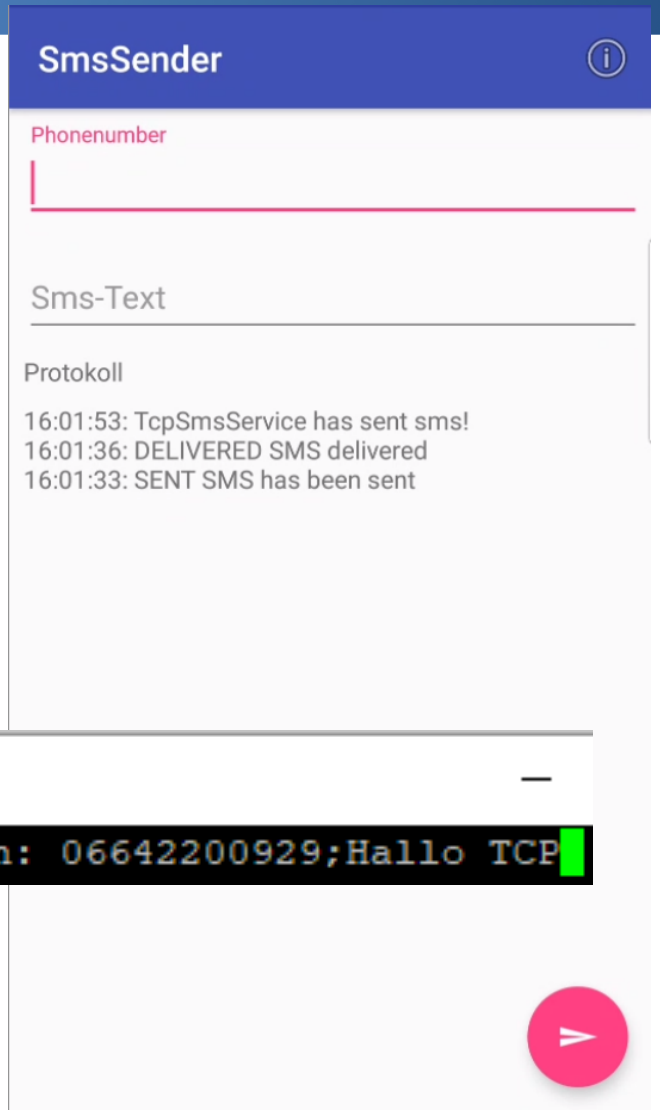
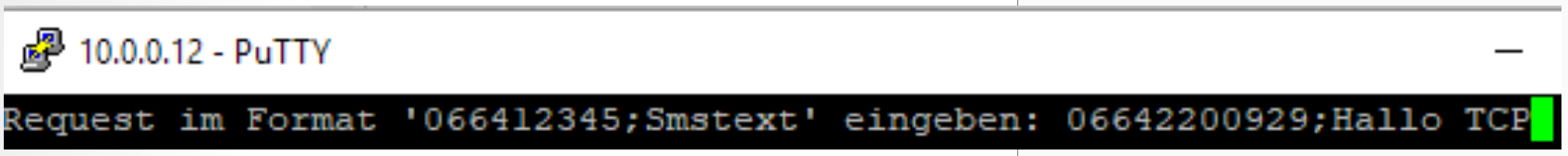
Service

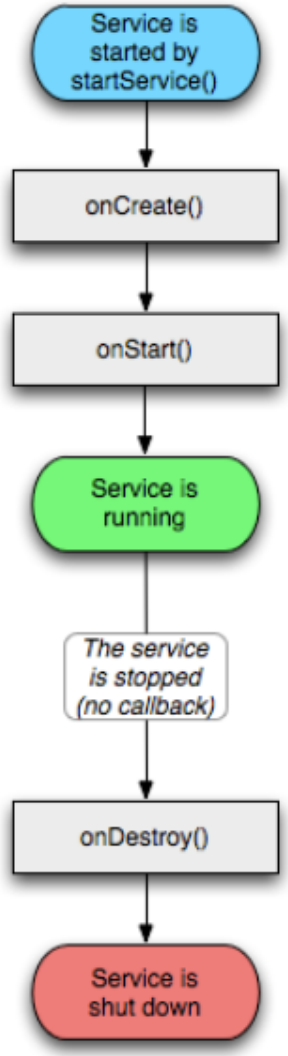




HTL LEONDING

Was ist ein Service

- Building-Block wie Activity
  - Ohne Benutzerinteraktion
  - Wiederverwendbarkeit
- Übernimmt Aufgaben, die nicht das UI betreffen
- Standardmäßig im MainThread(UI-Thread)
  - Erzeugt meist eigenen Thread
- Muss sparsam mit Ressourcen umgehen
  - Blockiert sonst die Vordergrundprozesse
  - Häufig ist Multithreading notwendig
- Lifecycle: create-start-destroy (kein stop)
- Weitere Infos unter: developer.android.com
  - http://developer.android.com/reference/android/app/Service.html

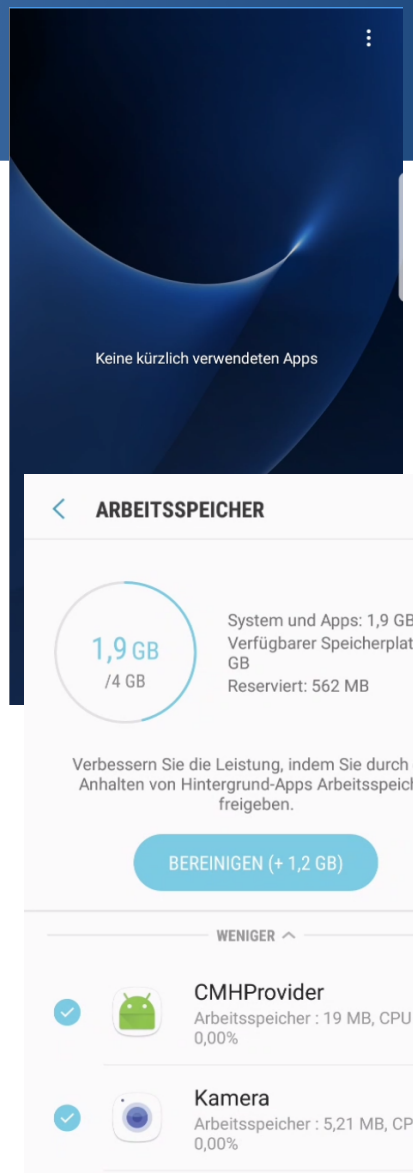


HTL LENDING

Services als Ressourcenfresser

- Gerät nach Neustart
  - Keine Apps geladen
- 1,9 GB durch System und Services belegt
- CPU-Auslastung natürlich auch nicht gering
- Akkuverbrauch

- Mit Android-O neu Richtlinien zum Thema Backgroundservices
  - https://developer.android.com/about/versions/oreo/android-8.0-changes



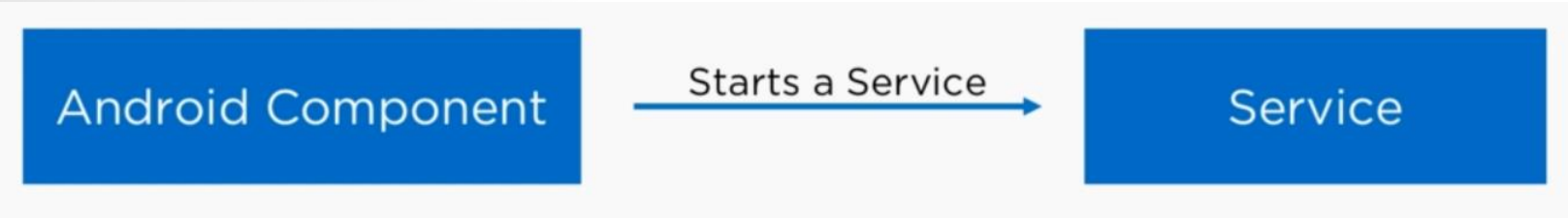
HTL LEONDING

Lokale Services ⇔ Remote Services

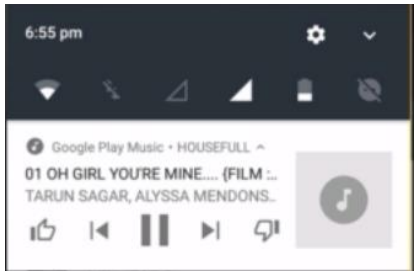
- Lokale Services
  - Im Prozessraum der Activity
  - Kommunikation mit Activity
    - Binder und ServiceConnection
    - Definition der Serviceaufgaben im Binder

- Remote Services
  - Über Prozessgrenzen verfügbar
  - Kommunikation
    - Broadcast
    - Binder
    - AIDL

Started Service



- Kann von beliebiger Komponente gestartet werden
  - Activity, BroadcastReceiver, Service, ContentProvider
- Läuft weiter, auch wenn startende Komponente beendet wurde
- One-Way-Kommunikation Component ➔ Service
- Beendet mit stopSelf() oder stopService()
- Gefahr vom Memory-Leaks
- Zwei Typen
  - Foreground-Services mit UI über Notification
  - Background-Services ohne Benutzerinteraktion



HTL LEONDING

Bound Service

- Component ist an den Service „gebunden“
  - Binder als Interface zur Kommunikation mit dem Service
  - Two-Way-Communication
- Starten mit erstem bind()
- Werden mit letztem unBind() beendet
- Kommunikation Activity ➔ Service
  - Lokaler Service ➔ Methodenaufruf
  - Remote Bound Service ➔ Handler/Message

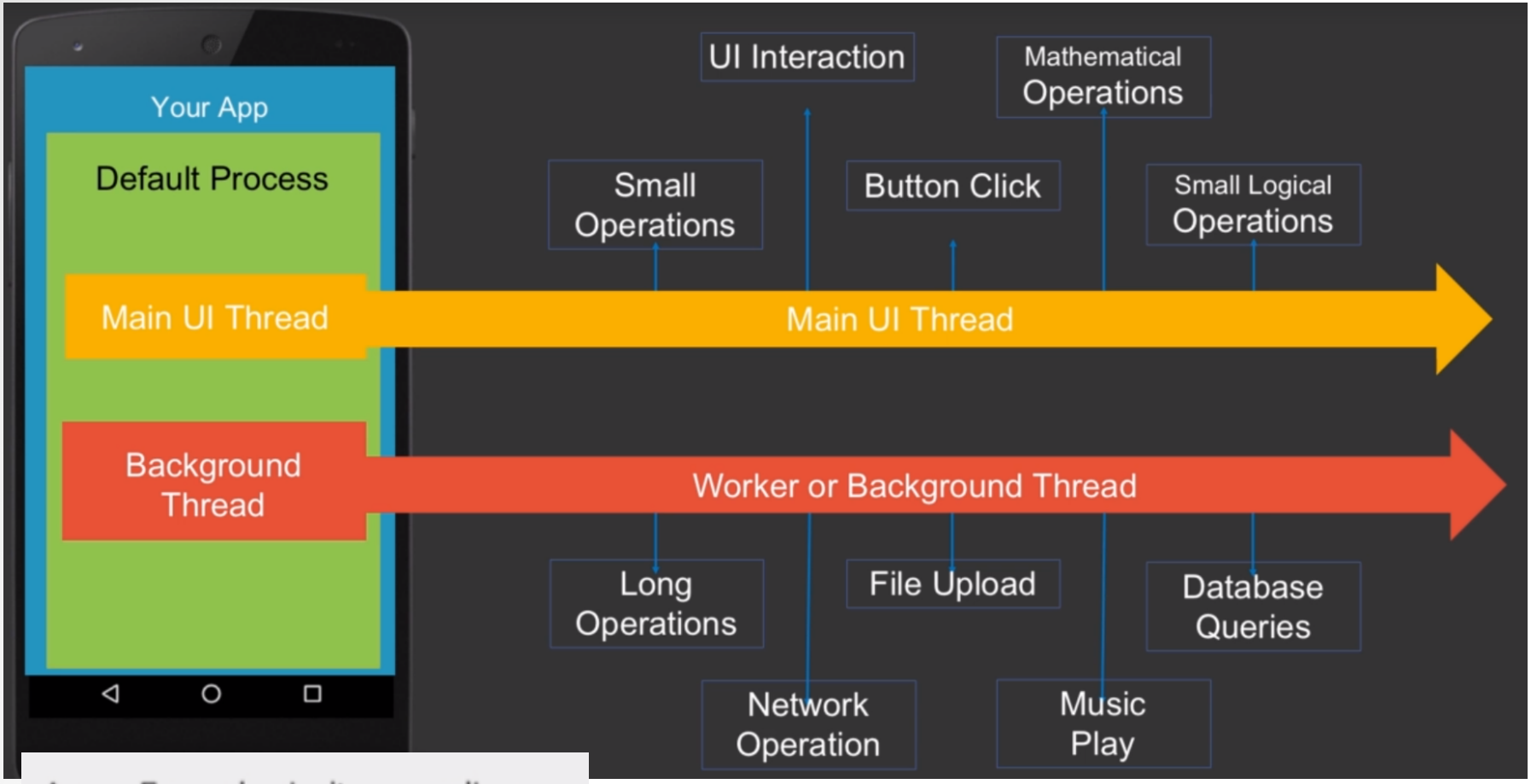
IntentService

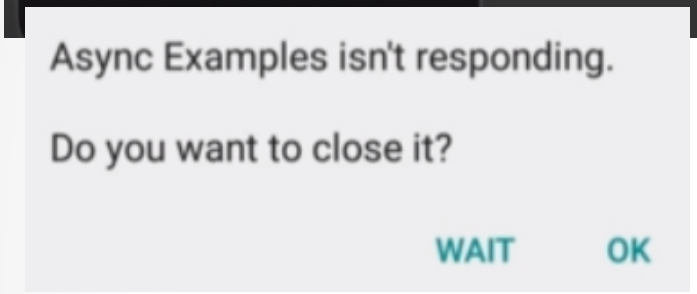
- Einfachste Form von Service mit Hintergrundtask (automatisch)
  - Download von Daten
  - Verschicken einer SMS
- Fire and forget
- Verarbeitet Anforderungen und verwaltet diese in Queue
- Wird über Intent aufgerufen
- Beendet sich nach getaner Arbeit selbst

Service ⇔ AsyncTask

- AsyncTask lagert länger dauernde Aufgaben in den Hintergrund aus
  - Laufen im Prozessumfeld der Activity
  - Sind eng an die erzeugende Komponente gekoppelt
- Service
  - Lose Kopplung an die erzeugende Komponente
  - Kann in eigenem Prozessumfeld laufen
  - Verwendet meist Threading für Hintergrundaufgaben

Process – Main-Thread – Background-Thread

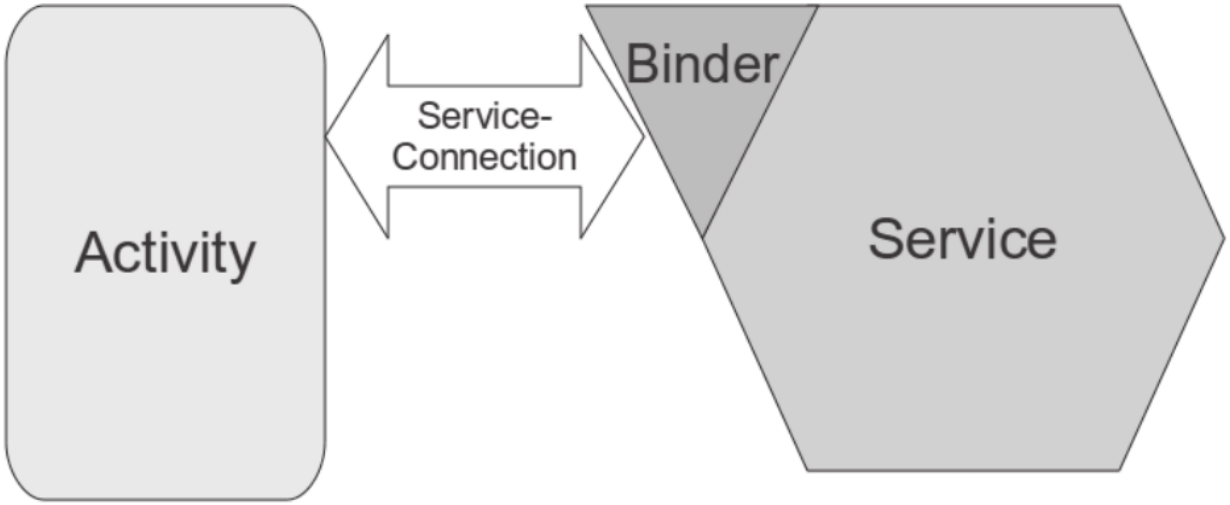




HTL LEONDING

Lokaler Service mit Binder

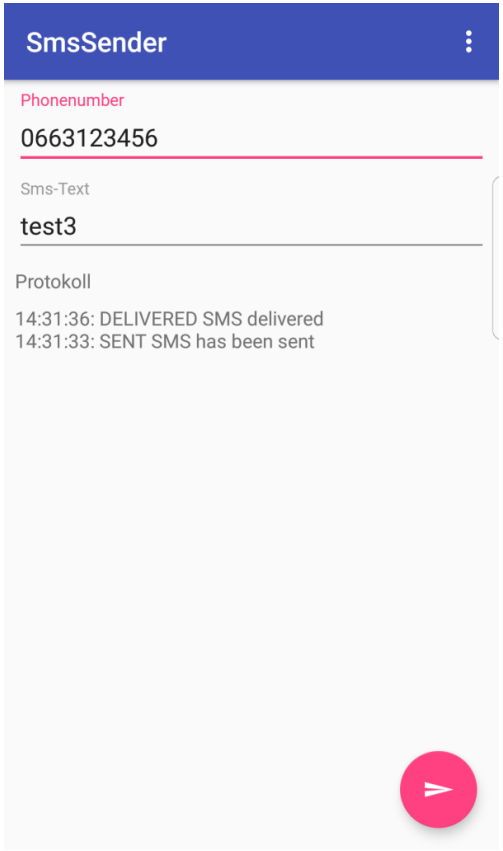
- Binder definiert Schnittstelle zu Service



Service: Klasse anlegen
Service: Binder definieren
Activity: ServiceConnection definieren
Activity: mit Service verbinden
Service: Methode onBind() überschreiben

Sms versenden in Service auslagern

- Funktionalität bleibt gleich
- Sms wird vom Service verschickt
- BroadcastReceiver werden im Service registriert/abgemeldet
- Benutzer wird per Toast informiert
- Wird später erweitert, damit Service auch Sinn macht



SmsSender – Service Service Anfang

Übung

HTL LEONDING

Template für gebundenen Service

```
class SmsSenderService : Service() {
    private val LOG_TAG = SmsSenderService::class.java.simpleName
    private val smsSenderServiceBinder = SmsSenderServiceBinder()

    // Systemmeldungen über den Versand der SMS per BroadcastReceiver empfangen
    internal var smsSentReceiver: BroadcastReceiver? = null
    internal var smsDeliveredReceiver: BroadcastReceiver? = null
    private var activityNotificationHandler: Handler? = null

    override fun onBind(intent: Intent): IBinder? {...}
    override fun onUnbind(intent: Intent): Boolean {...}
    override fun onCreate() {...}
    override fun onDestroy() {...}

    /** BroadcastReceiver für die PendingIntents des Empfangs ...*/
    private fun createBroadcastReceivers() {...}

    /** Sms per SmsManager verschicken. Für die Actions SMS_SENT und ...*/
    fun sendSMS(phoneNo: String, msg: String) {...}

    fun setNotificationHandler(handler : Handler){...}

    /** Statusmeldungen über das Versenden der SMS an die Activity ...*/
    fun sendNotificationToActivity(notificationText: String) {...}

    /** Binder implementiert die Schnittstellenmethoden des ...*/
    inner class SmsSenderServiceBinder : Binder() {...}
```

Binderklasse

- Retourniert Zugriff auf Serviceklasse
- Alternativ Interface mit benötigten Methoden

```
inner class SmsSenderServiceBinder : Binder() {

    fun getService() : SmsSenderService{
        return this@SmsSenderService
    }

}
```

Service – onBind()/onUnbind()

```
override fun onBind(intent: Intent): IBinder? {
    Toast.makeText( context: this,  text: "SmsSenderService onBind()", Toast.LENGTH_SHORT).show()
    Log.d(LOG_TAG,  msg: "onBind()")
    return smsSenderServiceBinder
}
override fun onUnbind(intent: Intent): Boolean {
    Toast.makeText( context: this,  text: "SmsSenderService onUnbind()", Toast.LENGTH_SHORT).show(
    Log.d(LOG_TAG,  msg: "SmsSenderService onUnbind()")
    activityNotificationHandler = null
    return super.onUnbind(intent)
}
```

Service – onCreate()/onDestroy()

- BroadcastReceiver registrieren/freigeben

```
override fun onCreate() {
    Toast.makeText( context: this,  text: "SmsSenderService onCreate()", Toast.LENGTH_SHORT).show()
    Log.d(LOG_TAG,  msg: "onCreate()")
    super.onCreate()
    createBroadcastReceivers()
    // Beide Broadcastreceiver für die pending intents (sent, delivered) registrieren
    registerReceiver(smsSentReceiver, IntentFilter( action: "SMS_SENT"))
    registerReceiver(smsDeliveredReceiver,IntentFilter( action: "SMS_DELIVERED"))
}
override fun onDestroy() {
    Log.d(LOG_TAG,  msg: "onDestroy()")
    Toast.makeText( context: this,  text: "SmsSenderService onDestroy()", Toast.LENGTH_SHORT).show()
    unregisterReceiver(smsSentReceiver)
    unregisterReceiver(smsDeliveredReceiver)
    super.onDestroy()
}
```

MainActivity – Service starten und beenden

```
override public fun onResume() {
    super.onResume()
    Log.d(LOG_TAG,  msg: "onResume() SmsService gebunden")
    val smsServiceIntent = Intent( packageContext: this, SmsSenderService::class.java)
    bindService(smsServiceIntent, smsSenderServiceConnection, Context.BIND_AUTO_CREATE)
}
override public fun onPause() {
    Log.d(LOG_TAG,  msg: "onPause() SmsService beendet")
    unbindService(smsSenderServiceConnection)
    super.onPause()
}
```

MainActivity – Service verwalten

```
internal var smsSenderService: SmsSenderService? = null
// ServiceConnection repräsentiert die Verbindung zum Service
private val smsSenderServiceConnection = object : ServiceConnection {
    override fun onServiceConnected(name: ComponentName, binder: IBinder) {
        Toast.makeText(applicationContext, text: "ServiceConnection, onServiceConnected",
                Toast.LENGTH_SHORT).show()
        val smsSenderServiceBinder : SmsSenderService.SmsSenderServiceBinder = binder
                as SmsSenderService.SmsSenderServiceBinder
        smsSenderServiceBinder?.getService().setNotificationHandler(logFromServicesNotificationHandler)
        smsSenderService = smsSenderServiceBinder?.getService()
    }
    override fun onServiceDisconnected(name: ComponentName) {

        Toast.makeText(applicationContext, text: "ServiceConnection, onServiceDisconnected", Toast.LENGTH_SHORT).show()
        smsSenderService = null
    }
}
```

MainActivity – Verwendung des Service

- Anlegen einer ServiceConnection
  - Verbindet sich über Binder

```
public class MainActivity extends Activity implements View.OnClickListener {
    private static final String LOG_TAG = MainActivity.class.getSimpleName();
    // Binder bilden das Schnittstellenobjekt (Contract) zum Service
    SmsSenderService.SmsSenderServiceBinder smsSenderServiceBinder;
    // ServiceConnection repräsentiert die Verbindung zum Service
    private ServiceConnection smsSenderServiceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder binder) {
            smsSenderServiceBinder = ((SmsSenderService.SmsSenderServiceBinder) binder);
        }
        @Override
        public void onServiceDisconnected(ComponentName name) { smsSenderServiceBinder = null;
    };
```

MainActivity – Service nutzen

```
if (isPhoneNumberOk && isTextOk){
    smsSenderService?.sendSMS(phoneNo, text)
    Snackbar.make(view,  text: "SMS sent", Snackbar.LENGTH_LONG).show()
}
```

Manifest – Service eintragen

```
        </activity>
        <service android:name=".SmsSenderService" />
    </application>
```

Dokumentation über LogCat

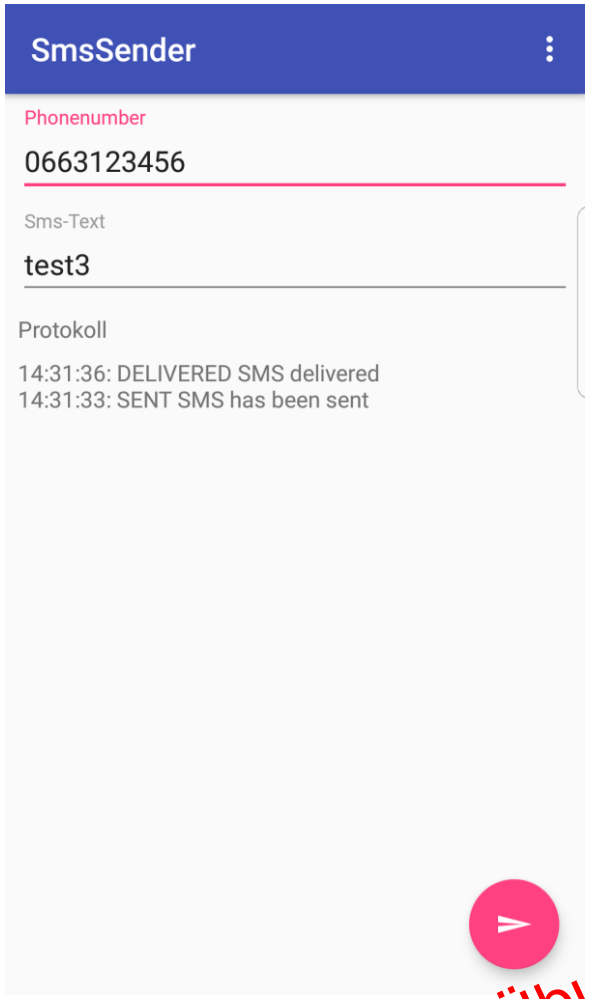
logcat

```
11-29 13:02:20.683  23531-23531/htl.at.smssender D/MainActivity:  onResume() SmsService gebunden
11-29 13:02:20.723  23531-23531/htl.at.smssender D/SmsSenderService:  onCreate()
11-29 13:02:20.728  23531-23531/htl.at.smssender D/SmsSenderService:  onBind()
11-29 13:02:20.843  23531-23531/htl.at.smssender D/libEGL:  loaded /system/lib/egl/libEGL_mali.so
11-29 13:02:20.868  23531-23531/htl.at.smssender D/libEGL:  loaded /system/lib/egl/libGLESv1_CM_mali.so
11-29 13:02:20.873  23531-23531/htl.at.smssender D/libEGL:  loaded /system/lib/egl/libGLESv2_mali.so
11-29 13:02:20.883  23531-23531/htl.at.smssender E/:  Device driver API match
    Device driver API version: 23
    User space API version: 23
11-29 13:02:20.883  23531-23531/htl.at.smssender E/:  mali: REVISION=Linux-r3p2-01rel3 BUILD_DATE=Wed Oct  9
11-29 13:02:21.018  23531-23531/htl.at.smssender D/OpenGLRenderer:  Enabling debug mode 0
11-29 13:02:33.418  23531-23531/htl.at.smssender D/SmsSenderService:  sendSMS()
11-29 13:02:33.418  23531-23531/htl.at.smssender D/SmsSenderService:  _sendSMS(06642200929,hxhl)
11-29 13:02:36.147  23531-23531/htl.at.smssender D/SmsSenderService:  Broadcast smsSentReceiver
11-29 13:02:45.115  23531-23531/htl.at.smssender D/SmsSenderService:  Broadcast smsDeliveredReceiver
11-29 13:02:53.095  23531-23531/htl.at.smssender I/Choreographer:  Skipped 32 frames!  The application may b
11-29 13:02:53.100  23531-23531/htl.at.smssender D/MainActivity:  onPause() SmsService beendet
11-29 13:02:53.170  23531-23531/htl.at.smssender D/SmsSenderService:  onUnbind()
11-29 13:02:53.170  23531-23531/htl.at.smssender D/SmsSenderService:  onDestroy()
```

SmsSender - ServiceEnde

SMS protokollieren

- Der Versand der SMS soll über eine mehrzeilige TextView protokolliert werden
- SMS sollen in Zukunft auch in einem Hintergrundthread verschickt werden können



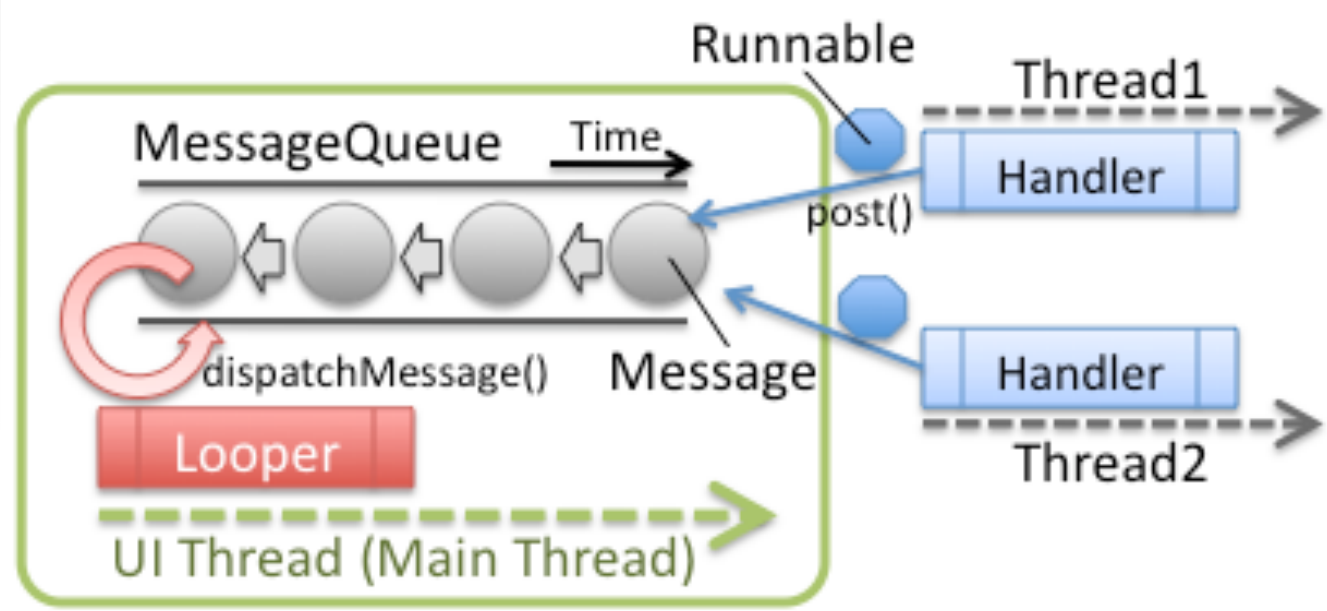
Übung

SmsSender - UI Callback Anfang

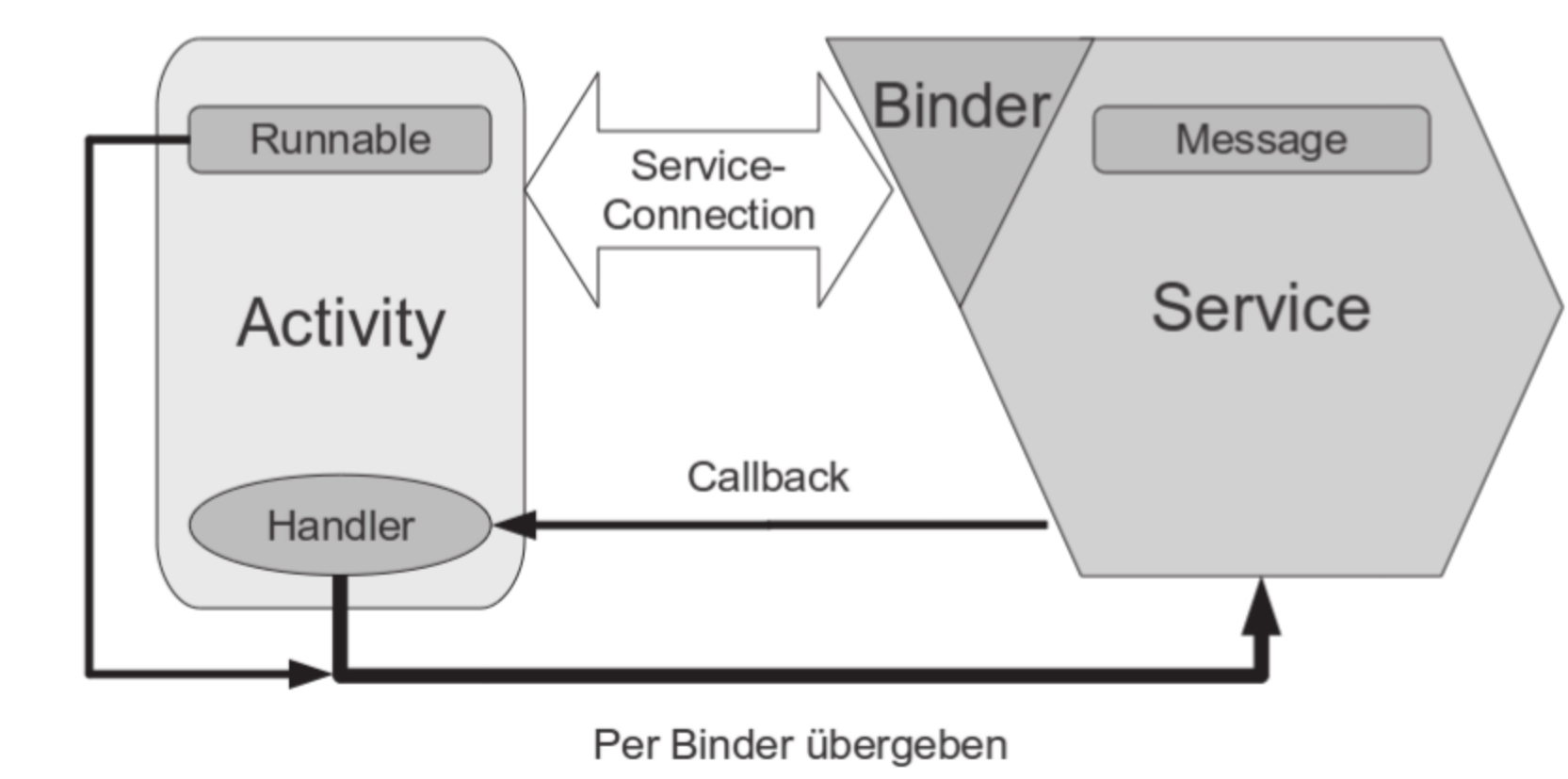
HTL LEONDING

Callback vom Service ➔ Activity

- Direkter Zugriff auf UI-Komponenten ist nur dem Mainthread erlaubt
  - Analog zu WinForms, WPF, Swing, Java FX, …



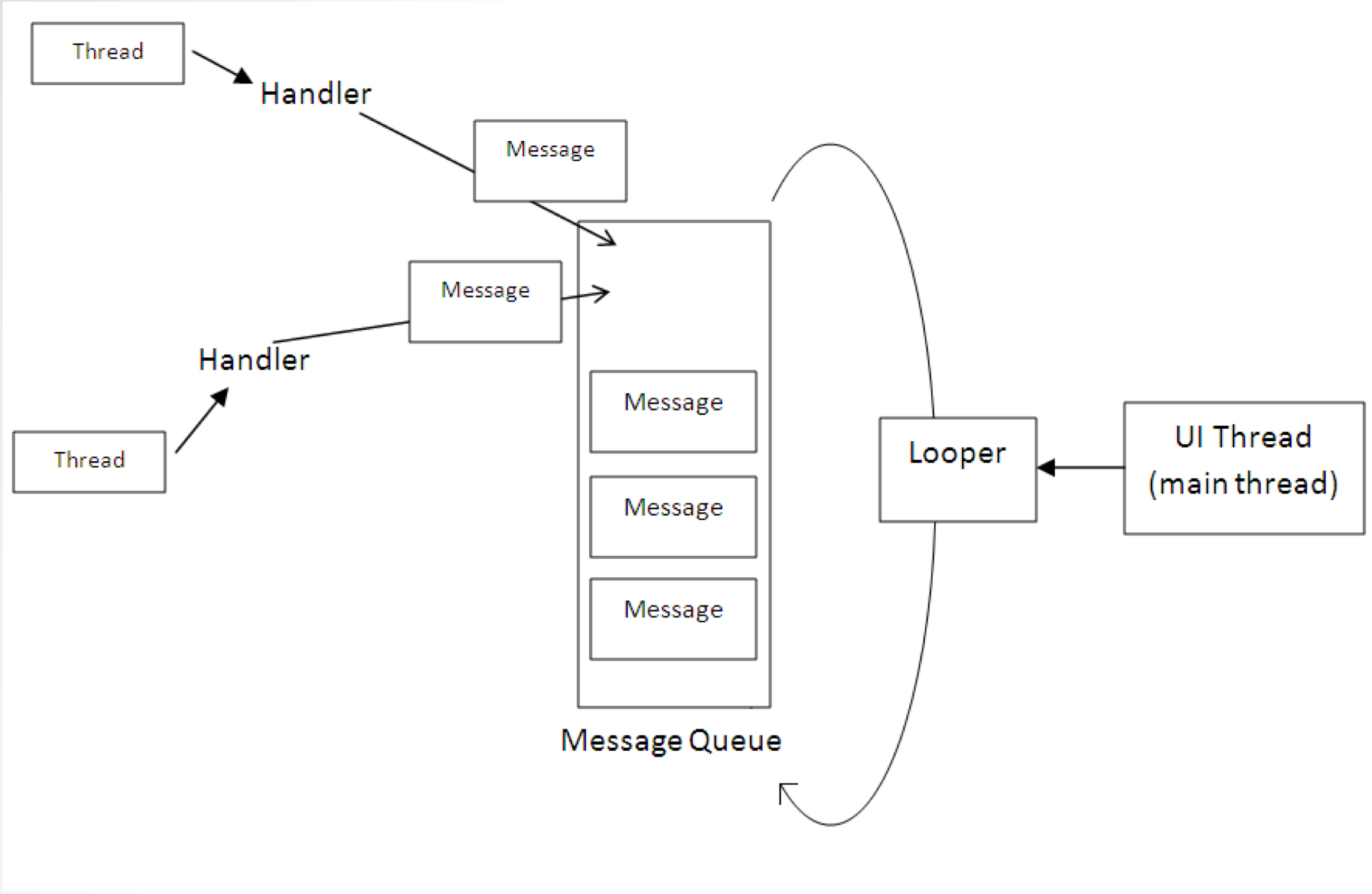
Zusammenhang mit Binder



Aktualisierung des UI mit Handler

- Callback-Mechanismus liefert Anforderung an Mainthread
- Handler stellt diese Brücke mit zwei möglichen Implementierungen her
  - post(Runnable uiUpdater)
    - Verzögerte Bearbeitung mit postDelayed() und postAtTime() möglich
  - Überschreiben von handleMessage() in eigenem Handler
    - sendMessage(Message msg) kann auch Daten übermitteln
    - Ebenfalls verzögert möglich

Multithreading mit Handler und Looper



MainActivity – Callback registrieren

```
private val logFromServicesNotificationHandler = object : Handler() {
    override fun handleMessage(msg: Message) {
        Log.d(LOG_TAG,  msg: "handleMessage()")
        val bundle : Bundle! = msg.data
        val logString : String! = bundle.getString( key: "LogString")
        addToLogView(logString)
        Toast.makeText(applicationContext,  text: "Handler, handleMessage: " + logString!!,
                Toast.LENGTH_SHORT).show()
        super.handleMessage(msg)
    }
}
```

Service – Handler verwalten

- Wenn Binding aufgelöst wird ➔ Handler löschen

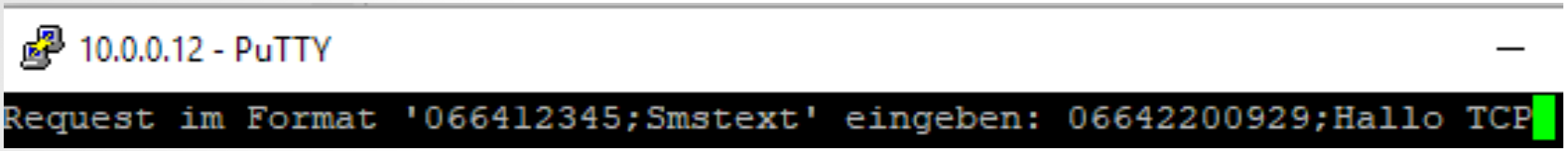
```
private var activityNotificationHandler: Handler? = null
fun setNotificationHandler(handler : Handler){
    activityNotificationHandler = handler
}
override fun onUnbind(intent: Intent): Boolean {
    Toast.makeText( context: this,  text: "SmsSenderService onUnbind()", Toast.LENGTH_SHORT).show()
    Log.d(LOG_TAG,  msg: "SmsSenderService onUnbind()")
    activityNotificationHandler = null
    return super.onUnbind(intent)
}
```

Über Binder Handler registrieren

```
override fun onServiceConnected(name: ComponentName, binder: IBinder) {
    Toast.makeText(applicationContext,  text: "ServiceConnection, onServiceConnected",
                Toast.LENGTH_SHORT).show()
    val smsSenderServiceBinder : SmsSenderService.SmsSenderServiceBinder  = binder
            as SmsSenderService.SmsSenderServiceBinder
    smsSenderServiceBinder?.getService()
            .setNotificationHandler(logFromServicesNotificationHandler)
    smsSenderService = smsSenderServiceBinder?.getService()
}
```

SmsSender - UI Callback Ende

SMS-Aufträge über TCP annehmen



SmsSender – SocketServer Anfang

Übung

HTL LEONDING

IP-Adresse des Smartphones ermitteln

- Steuerung über OptionsMenu

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="at.htl.smssender.MainActivity">
    <item
        android:id="@+id/menuActionIp"
        app:showAsAction="always"
        android:icon="@android:drawable/ic_menu_info_details"
        android:title="@string/menu_ip"/>
</menu>
```

MainActivity - Optionsauswahl verarbeiten

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menuQuit:
            finish();
            return true;
        case R.id.menuActionIp:
            AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
            alertDialogBuilder.setTitle(getString(R.string.txt_ip_address_handy));
            alertDialogBuilder
                    .setMessage(getLocalIpAddress())
                    .setCancelable(false)
                    .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            dialog.cancel();
                        }
                    });
            AlertDialog ipDialog = alertDialogBuilder.create();
            ipDialog.show();
            return true;
    }
    return false;
}
```

Lokale IP-Adresse ermitteln

```
public String getLocalIpAddress() {
    try {
        List<NetworkInterface> interfaces = Collections
                .list(NetworkInterface.getNetworkInterfaces());
        for (NetworkInterface intf : interfaces) {
            List<InetAddress> addrs = Collections.list(intf
                    .getInetAddresses());
            for (InetAddress addr : addrs) {
                if (!addr.isLoopbackAddress()  && addr instanceof Inet4Address) {
                    String sAddr = addr.getHostAddress().toUpperCase(Locale.GERMAN);
                    return sAddr;
                }
            }
        }
    } catch (Exception ex) {
        Log.e(LOG_TAG, "getLocalIpAddress() Exception: " + ex.toString());
    }
    return "";
}
```

Permission im Manifest nicht vergessen

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="htl.at.smssender" >
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="Inter"

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic
```

```
android.permission.INTERNAL_SYSTEM_WINDOW
android.permission.INTERNET
android.permission.SET_POINTER_SPEED
Dot, semicolon and some other keys will also close this lookup and be inserted into edito
```

Dialog liefert richtiges Ergebnis

HTL LEONDING

MainActivity – TcpServer binden/unbinden

- Automatisch mit Appstart

```
override public fun onResume() {
    super.onResume()
    Log.d(LOG_TAG, msg: "onResume() SmsService gebunden")
    val smsServiceIntent = Intent( packageContext: this, SmsSenderService::class.java)
    bindService(smsServiceIntent, smsSenderServiceConnection, Context.BIND_AUTO_CREATE)
    val tcpSmsServiceIntent = Intent( packageContext: this, TcpSmsService::class.java)
    bindService(tcpSmsServiceIntent, tcpSmsServiceConnection, Context.BIND_AUTO_CREATE)
}
override public fun onPause() {
    Log.d(LOG_TAG, msg: "onPause() SmsService beendet")
    unbindService(smsSenderServiceConnection)
    unbindService(tcpSmsServiceConnection)
    super.onPause()
}
```

Service – Tcp-Server starten/stoppen

- Service verwaltet Referenz auf TcpServer

```
override fun onCreate() {
    Log.d(LOG_TAG,  msg: "TcpSmsService onCreate()")
    val smsServiceIntent = Intent( packageContext: this, SmsSenderService::class.java)
    bindService(smsServiceIntent, smsSenderServiceConnection, Context.BIND_AUTO_CREATE)
    Log.d(LOG_TAG,  msg: "startTcpSmsServer()")
    tcpServerThread.start()
    super.onCreate()
}

override fun onDestroy() {
    Log.d(LOG_TAG,  msg: "TcpSmsService onDestroy()")
    Log.d(LOG_TAG,  msg: "Stop Serverthread()")
    unbindService(smsSenderServiceConnection)
    if (serverSocket != null)
        try {
            serverSocket?.close()
        } catch (e: IOException) {
            Log.e(LOG_TAG,  msg: "Exception in close serversocket: " + e.localizedMessage)
        }
    super.onDestroy()
}
```

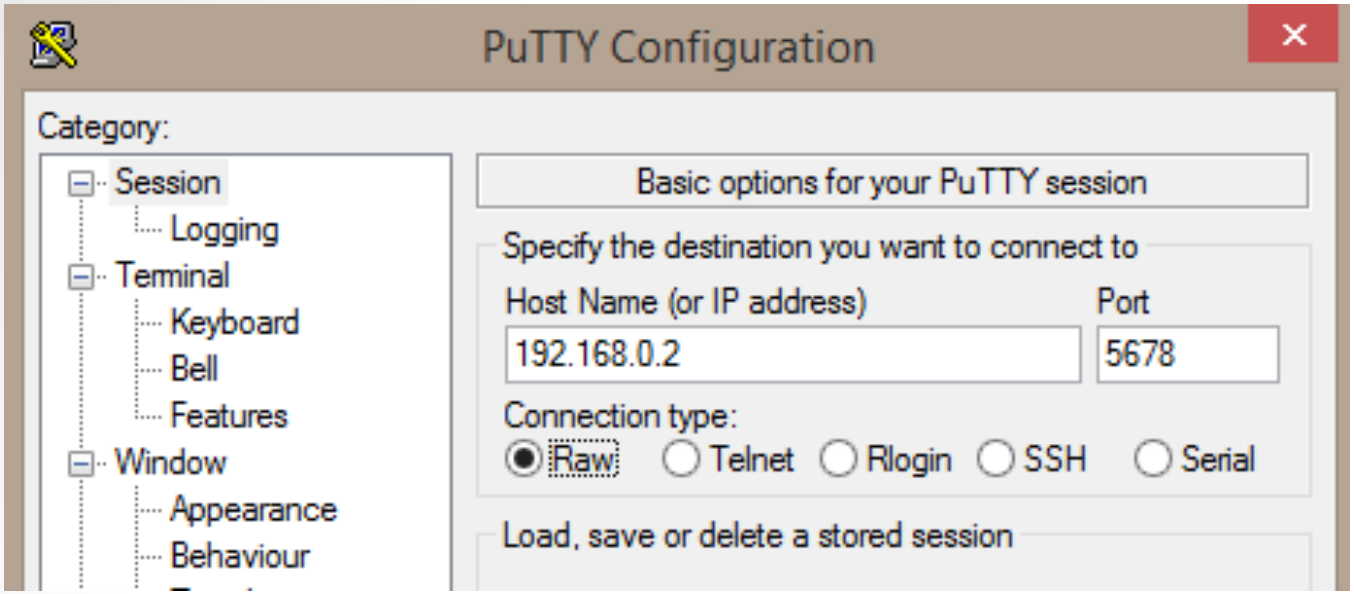
Tcp-Server - ServerThread

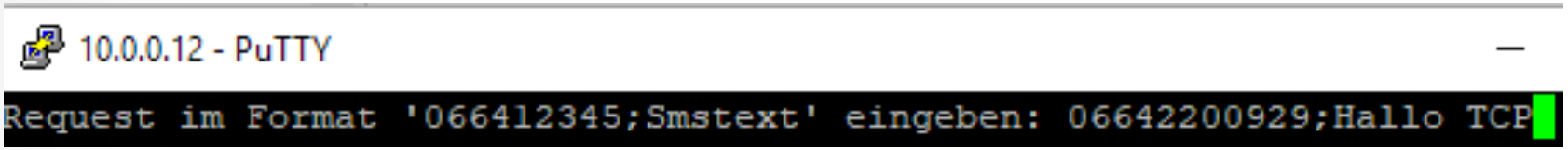
```
internal var tcpServerThread = Thread(Runnable {
    var isAborted = false
    Log.d(LOG_TAG,  msg: "start IpListener")
    try {
        serverSocket = ServerSocket(TCP_PORT)
        while (!isAborted /*!Thread.currentThread().isInterrupted() */) {
            try {

                Log.d(LOG_TAG,  msg: "TCP-Server wartet auf Verbindung")
                // Unterbrechung des Wartens auf Client erfolgt durch Schliessen des Sockets
                val clientSocket : Socket?  = serverSocket?.accept() // Auf Clientanfrage warten
                Log.d(LOG_TAG,  msg: "Anfrage von Client akzeptiert")
                val commThread = TcpSmsRequest(clientSocket)
                Thread(commThread).start()
            } catch (e: Exception) {
                Log.d(LOG_TAG,  msg: "Exception in accept(): " + e.localizedMessage)
                isAborted = true
            }
        }
    } catch (e: Exception) {
        Log.e(LOG_TAG,  msg: "Exception in create serversocket: " + e.localizedMessage)
    }
    Log.d(LOG_TAG,  msg: "Tcp-Server beendet")
})
```

ClientThread - Überblick

```
inner class TcpSmsRequest(private val clientSocket: Socket) : Runnable {
    override fun run() {
        try {
            val input = BufferedReader(InputStreamReader(clientSocket.getInputStream()))
            val output = BufferedWriter(OutputStreamWriter(clientSocket.getOutputStream()))
            output.write( str: "Request im Format '066412345;Smstext' eingeben: ")
            output.flush()
            val request : String! = input.readLine()
            if (request == null || request.length == 0) {
                Log.e(LOG_TAG, msg: "request is null or empty!")
                return
            } else {
                Log.d(LOG_TAG, msg: "Request: $request")
            }
            val elements : Array<String> = request.split(";".toRegex()).dropLastWhile { it.isEmpty() }.toTypedArray()
            if (elements.size < 2) {
                Log.e(LOG_TAG, msg: "Request hat nur " + elements.size + " Elemente!")
                return
            }
            val phoneNumber : String = elements[0]
            val smsText : String = elements[1]
            Log.d(LOG_TAG, msg: "Phonenumber: $phoneNumber, Text:$smsText")
            smsSenderService.sendSMS(phoneNumber, smsText)
            val response = "Sms sent"
            output.write(response)
            output.flush()
            output.close()
            input.close()
            sendNotificationToActivity( notificationText: "TcpSmsService has sent sms!")
        } catch (e: IOException) {
            Log.e(LOG_TAG, msg: "Communication-Thread Communication Exception: " + e.message)
        }
    }
}
```

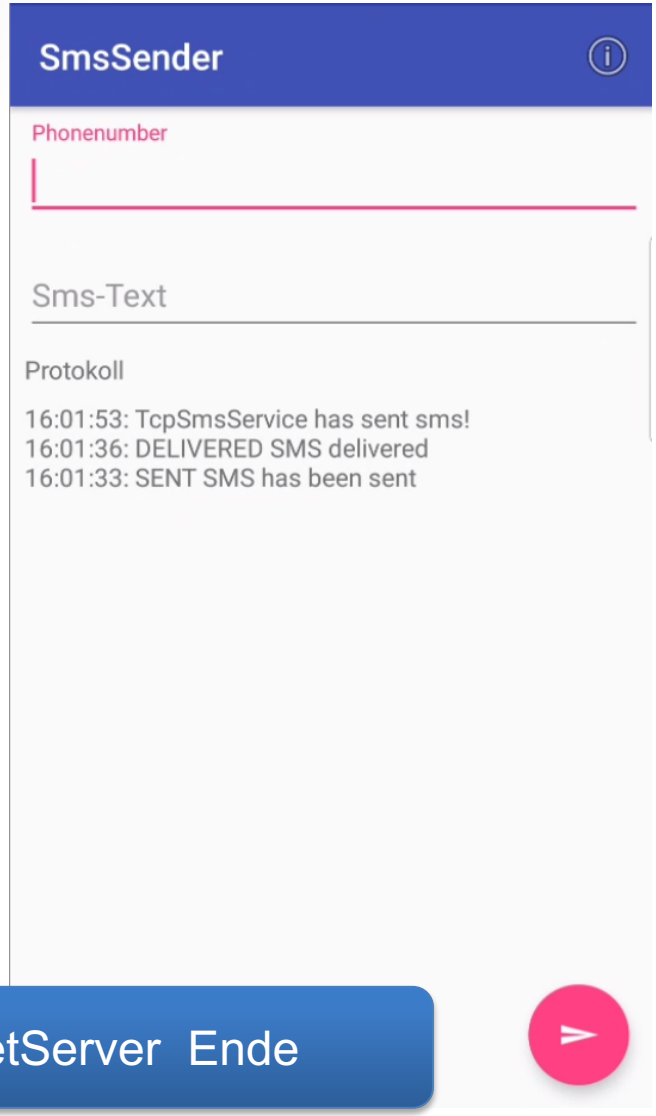
Test mit putty





HTL LEONDING

Protokoll des Versands am Handy

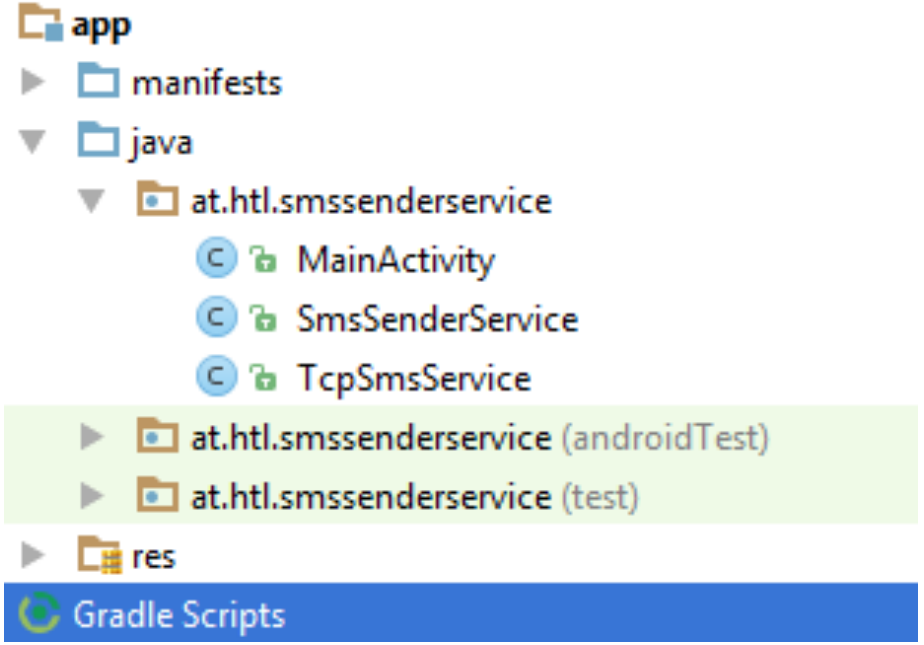


SmsSender – SocketServer Ende

HTL LEONDING

Übung Service

- TcpSmsService als eigenen Service definieren
  - TCP-Server als ServerThread
  - Pro Request ein Clientthread
- Verwendet SmsSenderService
  - Leitet dessen Messages an Activity weiter
  - Schickt Meldung, dass Versand über TCP erfolgte



Übung

HTL LEONDING