Wykład
Usługi w Android

Programowanie Urządzeń Mobilnych

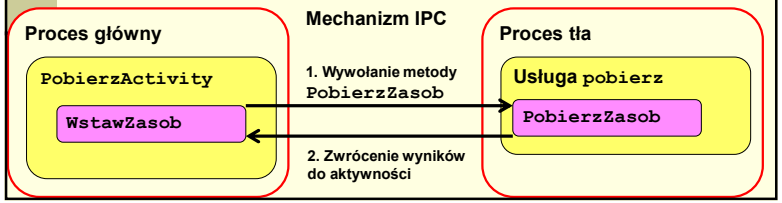
Dr inż. Damian Raczyński

Usługi podstawy

Usługi nie mają interfejsu graficznego, uruchomione są w tle w oddzielnym wątku lub procesie.

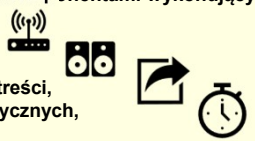
Aktywności wykorzystują usługi w celu wykonania długo trwających czynności (np. dostęp do zdalnego zasobu).

Aktywności i usługi współpracują z wykorzystaniem mechanizmów IPC (`AIDL` – Android Interface Definition Language)



Usługi podstawy

Usługi w systemie Android są komponentami wykonującymi długie czasowo operacje w tle np.:
- Operacje sieciowe,
- Odtwarzanie muzyki,
- Współpraca z dostawcami treści,
- wykonywanie zadań periodycznych,
- ... .



Usługę może uruchomić inny komponent Android'a.

Usługa może działać w tle pomimo tego, że użytkownik przełączy aktywną aplikację/aktywność.

Aktywność ściągająca plik

Usługa nie wspiera bezpośredniego dostępu przez interfejs użytkownika

Usługa pobierz

Usługi podstawy

Występują trzy podstawowe typy usług:
`Started Service` – wykonują pojedynczą operację i nie muszą zwracać wyników do procesu wywołującego bezpośrednio

`Bound Service` – dostarczają interfejsu typu klient-serwer (umożliwia on komunikację z usługą).

`Scheduled Service` (od API 21)–wywoływana w określonym czasie

Uruchamianie usługi typu `Started Service`:
inne komponenty uruchamiają usługę wykorzystując metodę `startService()`:

```
Intent intent = new Intent(this, ThreadedDownloadService.class);
intent.putExtra(„URL”, imageUrl);
startService(intent);
```

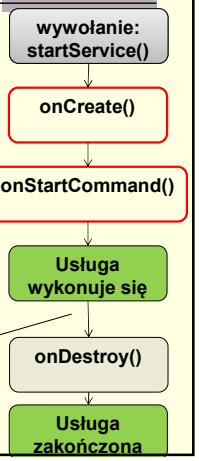
Usługi podstawy

`Started Service` – cykl życia

`startService()` powoduje wywołanie metody `onCreate()` i metody `onStartCommand()`

```
public class DownloadService extends Service{
        public int onStartCommand(Intent intent,
                int flags,
                int startId)
{
        ...
}
```

Usługa jest zatrzymywana przez siebie lub klienta



Usługi podstawy

Po wykonaniu operacji usługa może zostać zatrzymana.

Usługa może zatrzymać się wywołując metodę `stopSelf()`.

Inny komponent może zatrzymać usługę wywołując metodę `stopService()`.

Przykłady usług `Started Service`:
- Usługi SMS, MMS

Zarządzanie wiadomościami takie jak wysyłanie danych, tekstu, ...

- AlertService

Obsługuje przypomnienia związane z kalendarzem

Podsumowując: Usługi wywoływane są na ŻĄDANIE!

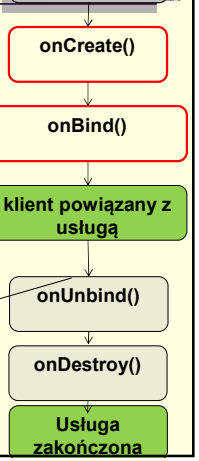
6

Usługi podstawy

`Bound Service` – cykl życia
Usługa `Bound Service` umożliwia aplikacji utworzenie długoterminowego połączenia z wykorzystaniem metody `bindService()`

```
Intent intent = new
Intent(IDownloadSync.class.getName());
bindService(intent, this.syncConnection,
Context.BIND_AUTO_CREATE);
```

klient wywołuje metodę unbindService()



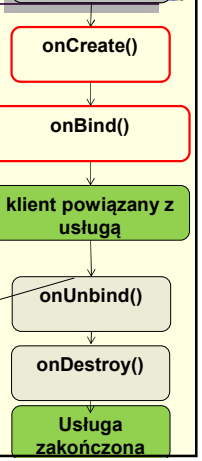
Usługi podstawy

`Bound Service` – cykl życia
Wywołanie metody `bindService()` powoduje wywołaniem metody `onCreate()` oraz `onBind()` usługi

```
public IBinder onBind(Intent intent){
        return this.binder;
}
```

Metoda zwraca IBinder, który definiuje API dla komunikacji z usługą.

klient wywołuje metodę unbindService()



Usługi podstawy

`Bound service` wykorzystuje interfejs typu klient-serwer, który umożliwia komponentom interakcje z usługą (wysyłanie żądań, pobieranie wyników z wykorzystaniem IPC)

```
interface DownloadCall{
        String downloadImage(in Uri uri);
}
```

`AIDL` – Android Interface Definition Language,
`Binder RPC` implementuje brokerów i szablony Proxy.

Usługi typu `Bound` są uruchomione dopóki inny komponent jest z nimi połączony (zbindowany).

W przypadku, gdy klient nie zamierza współpracować już z usługą, wywołuje metodę `unbindService()`. W przypadku, gdy żaden inny klient nie jest zbindowany z usługą, jest ona niszczona.

9

Usługi podstawy

Przykłady:

- `BluetoothHeadsetService` (usługa wspierająca zestawy słuchawkowe bluetooth)

- `MediaPlaybackService` (umożliwia odtwarzanie muzyki w tle),

- Wymiana wiadomości E-mail (zarządza operacjami związanymi z wiadomościami e-mail – np. synchronizacja skrzynki pocztowej na telefonie ze stanem konta pocztowego).

10

Usługi podstawy

Implementacja Usług `Started Service`

Implementacja usług tego typu jest podobna do Aktywności.

Klasa usługi musi dziedziczyć po `Service`,

zawierać implementację metod związanych z cyklem życia,

Zawierać odpowiedni wpis w pliku manifestu.

```
public class Service extends ...{
        public void onCreate();
        public int onStartCommand(Intent intent, int flags, int startId);
        public abstract IBinder onBind(Intent intent);
        public boolean onUnbind(Intent intent);
        protected void onDestroy();
...
}
```

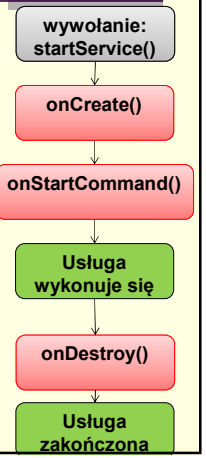
Usługi podstawy

Implementacja Usług Started Service

`onCreate()` – wywoływana, w przypadku gdy usługa jest tworzona (przed `onStartCommand`). W przypadku, gdy usługa już działa metoda nie zostanie wywołana.

`onStartCommand()` – wywoływana za każdym razem, gdy usługa dostaje „zlecenie" wywołane metodą `startService()`.

`onDestroy()` – wywoływana w przypadku, gdy usługa jest zakańczana w celu zwolnienia zasobów.



Usługi podstawy

Implementacja Usług `Started Service`

Usługa typu `Started Service` jest aktywowana poprzez wywołanie metody `Context.startService()`

Intencja identyfikuje usługę, z którą będzie odbywała się komunikacja (i do której zostaną przekazane parametry przez `extras`) w celu przekazania informacji co dokładnie wykonać.

```
Intent intent = new Intent(this, ThreadedDownloadService.class);
intent.putExtra(„URL”, imageUrl);
startService(intent);
```


Usługi podstawy

metoda `startService()` nie jest metodą blokującą.

Jeżeli usługa nie jest jeszcze uruchomiona, zostaje uruchamiana i otrzymuje intencję poprzez metodę `onStartCommand()`

Uruchomiona usługa zazwyczaj wykonuje pojedynczą operację, po czym kończy swoje działanie

```
public class DownloadService extends Service{
        public int onStartCommand(Intent intent, int flags, int startId)
        {
                ...
        }
}
```

Usługi podstawy

Usługi typu `Started Services` nie zwracają wyników do metody, która je wywołała, ale zwracają wartości do systemu Android poprzez metodę `onStartCommand()`

Wartości mają znaczenie gdy Android zabije usługę przed zakończeniem wykonania (np. za mało pamięci):
`START_STICKY` – nie przekazuj ponownie intencji do `onStartCommand()` (przekazuje `null`) (przydatne np. dla odtwarzaczy multimediów)
`START_REDELIVER_INTENT` – uruchom usługę poprzez `onStartCommand()` dostarczając tą samą intencję, która została dostarczona poprzednio (przydatne np. dla usługi ściągającej plik)
`START_NOT_STICKY` – Usługa powinna pozostać zatrzymana dopóki nie zostanie uruchomiona z kodu aplikacji

```
public class DownloadService extends Service{
public int onStartCommand(Intent intent, int flags, int startId)
{

        return ...;
}
}
```

Usługi podstawy

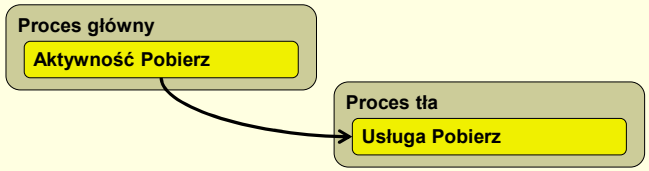
Należy poinformować Android o tym, że dany element jest usługą wprowadzając do pliku manifestu znaczniki `<service>` (jako element znacznika `<application>`)

element `android:name` ma odnosić się do klasy usługi

```
usługa MMS:
<service android:name=„.transaction.TransactionService”
android:exported=„true”/>

<service android:name=„.transaction.SmsReceiverService”
android:exported=„true”/>


Usługa odtwarzacza muzyki:
<service android:name=„com.android.music.MediaPlaybackService”
android:exported=„false”/>
```

Usługi podstawy

W przypadku, gdy usługa ma się wykonywać w innym procesie należy wykorzystać atrybut:
`android:process=„:myProcess"`




```
<service android:name=„com.android.music.MediaPlaybackService"
android:process=„:myProcess"/>
```


Usługi podstawy

`IntentService` wykonuje następujące zadania:

Tworzy domyślny wątek roboczy, który obsługuje wszystkie intencje dostarczane do metody `onStartCommand()`,

Tworzy kolejkę dostarczonych elementów tak, aby metoda `onHandleIntent` obsługiwała wyłącznie jedną intencję w danym czasie (nie trzeba się martwić o wielowątkowość),

Zatrzymuje usługę po tym, jak zostaną obsłużone wszystkie żądania (nie trzeba wywoływać `stopSelf()`).

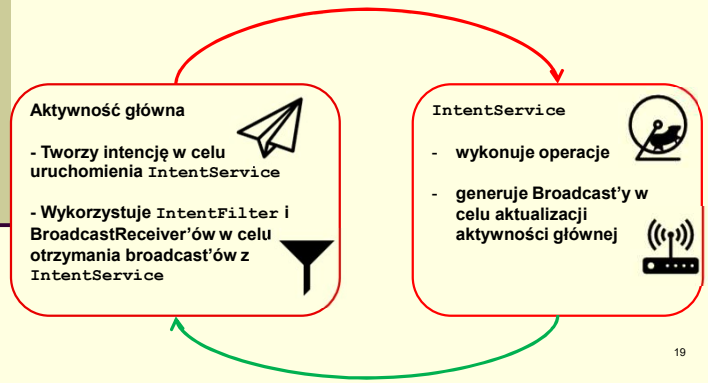
Dostarcza standardowej implementacji `onBind` (zwraca `null`),

Dostarcza standardowej implementacji `onStartCommand` (wysyła intencję do kolejki elementów)

Samemu należy zaimplementować tylko `onHandleIntent();`

Usługi podstawy

`IntentService`
Najczęściej wykorzystywaną klasą usługi jest `IntentService`



Usługi podstawy

`IntentService` jest klasą bazową dla usług obsługujących asynchroniczne żądania (wyrażone poprzez intencje).

```
public class PrzykladService extends IntentService{
        protected void onHandleIntent(Intent intent){
                ...
        }
}
```

`IntentService` wywołuje tą metodę z domyślnego wątku z intencją, która uruchomiła usługę

Usługi podstawy

Przykład:

```
public class ThreadedDownloadService extends IntentService{

public void onHandleIntent(Intent intent){
       String
downloadedType=intent.getCharSequenceExtra(„DOWNLOAD_TYPE”).
                                                toString();
if(downloadType.equals(„messenger”))
       threadMessengerDownload(intent);
else if (downloadType.equals(„pending_intent”))
       threadPendingIntentDownload(intent);
else if (downloadType.equals(„asynctask”)
       asyncTaskDownload(intent);
...
}

}
```

Dziedziczy po IntentService

Metoda cyklu życia, ściąga obrazek z wykorzystaniem różnych mechanizmów

Manifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.root.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <service android:name=".Usluga1" android:exported="false"/>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

MainActivity:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b= (Button) findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
             Intent i= new Intent(getApplication(), Usluga1.class);
                i.putExtra("info", "tekst");
                startService(i);
            }
        });

    }
}
```

Started Service

Usluga1:

```
public class Usluga1 extends IntentService {
    public Usluga1() {
        super("usluga");
    }
    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        String tekst = intent.getStringExtra("info");
        Log.d("Wyklad", tekst);
    }
}
```

```
plication D/Wyklad: tekst
plication D/Wyklad: tekst
plication D/Wyklad: tekst
plication D/Wyklad: tekst
```

24

Usługi podstawy

Wyświetlanie okienka `Toast`

Okienka Toast mogą być wyświetlane wyłącznie z wątków, które posiadają przydzielonego `Handler'a/Looper'a`.

Usługi domyślnie nie posiadają kolejki komunikatów – należy stworzyć `hanlder'a` samemu.

Started Service

Manifest:

```
<service android:name=".SendInfo"  android:exported="false"/>
```

Started Service

Aktywność:

```
public class MainActivity extends AppCompatActivity {
    Context c=this;
       @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b = (Button) findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {
              @Override
              public void onClick(View v) {
                  String a = ((EditText)
findViewById(R.id.editText)).getText().toString();
                  Intent intent;
                  intent = new Intent(c, SendInfo.class);
                  intent.putExtra("info", a);
                  intent.putExtra("dlugosc", Toast.LENGTH_LONG);
                  startService(intent);
              }
          });
    }
}
```

Started Service

Usługa:

```
public class SendInfo extends IntentService {
    private Handler mHandler;
    public String tekst;
    int dlugosc;
    @Override
    public void onCreate(){
        super.onCreate();
        mHandler = new Handler();
    }
    public SendInfo() {
        super("SendInfo");
    }
    ...
```

Started Service

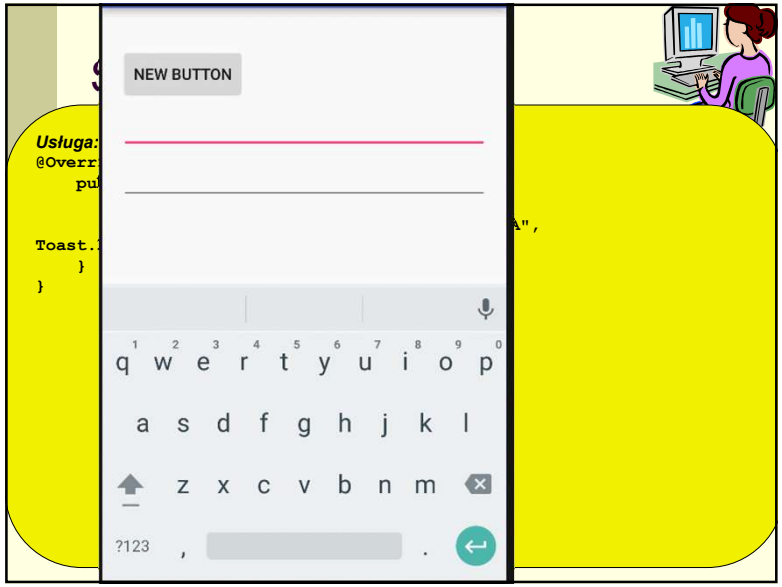
Usługa:

```
@Override
    protected void onHandleIntent(Intent intent) {
        tekst = intent.getStringExtra("info");
        dlugosc = intent.getIntExtra("dlugosc", Toast.LENGTH_SHORT);
        mHandler.post(new Runnable(){
            @Override
            public void run(){
                Toast.makeText(SendInfo.this,
                        tekst,
(dlugosc==Toast.LENGTH_SHORT)?Toast.LENGTH_SHORT:Toast.LENGTH_LONG
                ).show();
            }});
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    ...
```

Started Service

Usługa:

```
@Override
    public void onDestroy(){
        super.onDestroy();
        Toast.makeText(this, "USLUGA ZATRZYMANA",
Toast.LENGTH_LONG).show();
    }
}
```



Started Service

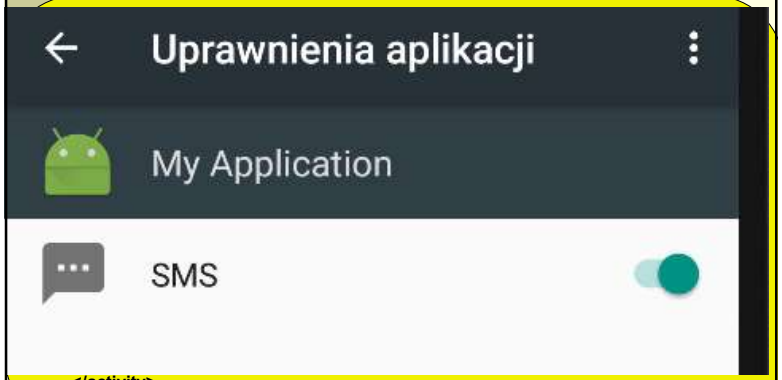
Przykład – usługa wysyłająca SMS

32

Started Service

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.administrator.myapplication" >

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

    <application
        ...>
        <service android:name=".SendSMS" android:process=":myProcess"
android:exported="false"/>

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

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

Started Service




```
</activity>
</application>
```

```
package com.example.administrator.myapplication;

import android.app.IntentService;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.Looper;
import android.support.annotation.Nullable;
import android.telephony.SmsManager;


public class SendSMS extends IntentService {
    private Looper serviceLooper;
     public SendSMS() {
         super("SendSMS");
     }
     @Override
     protected void onHandleIntent(Intent intent) {
         String numer = intent.getStringExtra("numer");
         String tresc = intent.getStringExtra("tresc");
         SmsManager sms = SmsManager.getDefault();
         sms.sendTextMessage(numer, null, tresc, null, null);
     }
}
```

```
Context c=this;
       @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b = (Button) findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    String a = ((EditText)
findViewById(R.id.editText)).getText().toString();
                    String b = ((EditText)
findViewById(R.id.editText2)).getText().toString();
                    Intent intent;
                    intent = new Intent(c, SendSMS.class);
                    intent.putExtra("numer", a);
                    intent.putExtra("tresc", b);
                    startService(intent);
                }
            });
    }
```

Started Service

Implementacja:

dziedziczyć po `Service`

Zaimplementować metody cyklu życia

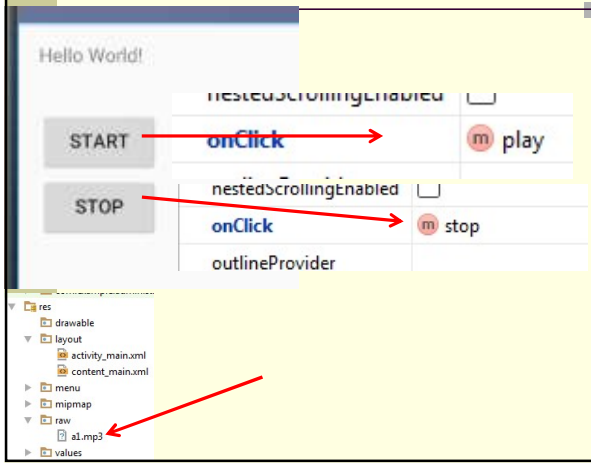
`onBind()` i `onUnbind()` nie są w tym typie potrzebne, ale należy dostarczyć ich implementację (która nic nie robi).

zamieścić odpowiednie wpisy w `AndroidManifest`

Sposób wymagany w przypadku, gdy wymagana jest wielozadaniowa obsługa żądań (`IntentService` obsługiwał jedno żądanie w danym czasie).

37

Started Service



38

Usługi podstawy

Aktywność główna:

```
public class MainActivity extends AppCompatActivity {
. . .

  public void play(View src){
   Intent intent = new Intent(MainActivity.this, MusicService.class);
   intent.putExtra("SongID", R.raw.a1);
   startService(intent);
  }

  public void stop(View src){
   Intent intent = new Intent (MainActivity.this,MusicService.class);
   stopService(intent);
  }
. . .

}
```

Usługi podstawy

Klasa MusicService:

```
public class MusicService extends Service {
  MediaPlayer player;
  public int onStartCommand(Intent intent, int flags, int startid){
    player= MediaPlayer.create(this, intent.getIntExtra("SongID", 0));
    player.setLooping(false);
    player.start();
    return START_NOT_STICKY;
  }
  public void onDestroy(){player.stop();}
  @Nullable
  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }
}
```

Usługi podstawy

AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.administrator.myapplication" >

 <application
...
   <service android:exported="true" android:name=".MusicService"/>
 </application>
</manifest>
```

41

Started Service

Wysyłanie wiadomości z usługi do aktywności

Android dostarcza klasę `Messenger`, która pozwala na "obudowanie" obiektu klasy `Handler` aktywności i przekazanie jej do innego komponentu.

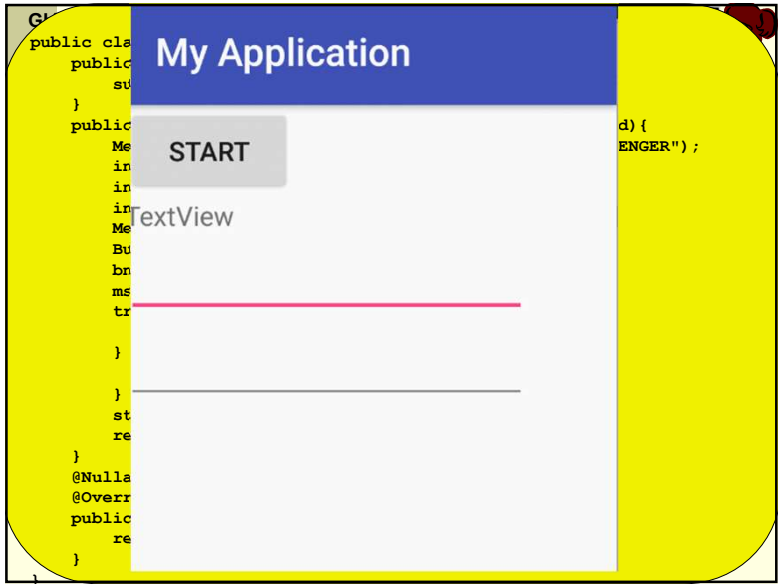
Umożliwia to korzystanie z kolejki komunikatów aktywności w dowolnym innym komponencie.

```
Messenger messenger = new Messenger(handler);
```

42

```
public class MainActivity extends AppCompatActivity {
    private Handler h = new Handler() {
        public void handleMessage(Message msg) {
            TextView t = (TextView) findViewById(R.id.textView);
            int wynik = msg.getData().getInt("wynik");
            t.setText(String.valueOf(wynik));
        }
    };
 @Override
 protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   Button b= (Button) findViewById(R.id.button);
   b.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
     EditText et1= (EditText) findViewById(R.id.editText);
     EditText et2= (EditText) findViewById(R.id.editText2);
     int a= Integer.parseInt(et1.getText().toString());
     int b= Integer.parseInt(et2.getText().toString());
     Intent i= new Intent(getApplication(), Usluga2.class);
     i.putExtra("op1", a);
     i.putExtra("op2", b);
     i.putExtra("MESSENGER", new Messenger(h));
     startService(i);
 } }); }}
```

```
public class Usluga2 extends Service {
    public void onCreate(){
        super.onCreate();
    }
    public int onStartCommand(Intent i, int f, int startId){
        Messenger m = (Messenger) i.getExtras().get("MESSENGER");
        int a= (Integer) i.getExtras().getInt("op1");
        int b= (Integer) i.getExtras().getInt("op2");
        int wynik=a+b;
        Message msg= new Message();
        Bundle bnd = new Bundle();
        bnd.putInt("wynik", wynik);
        msg.setData(bnd);
        try {
            m.send(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        stopSelf();
        return START_NOT_STICKY;
    }
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```



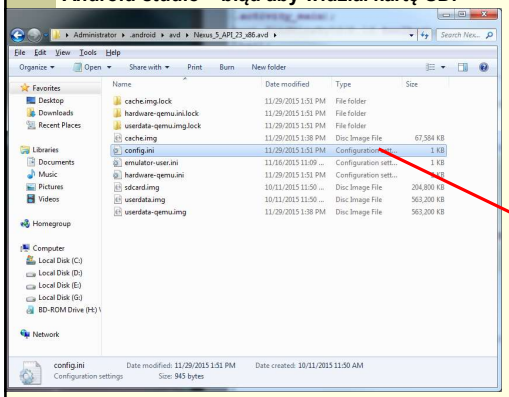
Started Service

Przykład 2 – ściąganie obrazka

- **Klient wysyła Intencję poprzez wywołanie metody `startService()`,**
- **Usługa pobierania (`DownloadService`) jest uruchamiana na żądanie,**
- **`DownloadService` obsługuje każdą intencję w wątku roboczym i zatrzymuje się w przypadku, gdy nie ma więcej pracy,**
- **Implementacja związana z kolejką komunikatów.**

Started Service

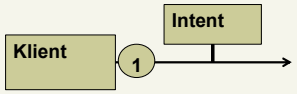
Android studio – błąd aby widział kartę SD:



Dodać na końcu pliku wpis: hw.sdCard=yes

Started Service

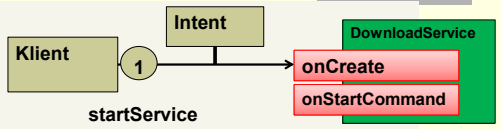
1. Klient wysyła Intencję poprzez wywołanie startService()



startService

Started Service

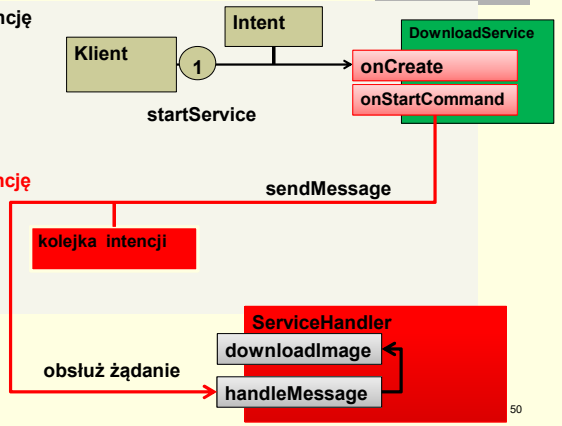
1. Klient wysyła Intencję poprzez wywołanie startService()
2. DownloadService jest uruchamiana (na żądanie)



49

Started Service

1. Klient wysyła Intencję poprzez wywołanie startService()
2. DownloadService jest uruchamiana (na żądanie)
3. DownloadService obsługuje każdą Intencję i zatrzymuje się gdy kolejka zadań jest pusta



50

```
public class MainActivity extends AppCompatActivity {
  Handler handler;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
  ..
    handler = new Handler(){
      public void handleMessage(Message msg){
        Bundle data= msg.getData();
        String pathname= data.getString("PATHNAME");
        if(msg.arg1!=0 || pathname==null){
          Toast.makeText(MainActivity.this,"BLAD" Toast.LENGTH_LONG).show()
        } else {
          ImageView x = (ImageView)findViewById(R.id.imageView);
          Drawable d = Drawable.createFromPath(pathname);
          x.setImageDrawable(d);
        }
      }
    };
  }
. . .
  public void onClick(View v){
    Intent intent = new Intent(MainActivity.this, DownloadService.class);
    EditText editText = (EditText) findViewById(R.id.editText);
    intent.setData(Uri.parse(editText.getText().toString()));
    intent.putExtra("MESSENGER", new Messenger(handler));
    startService(intent);
  }
```

Usługi podstawy

```
private class DownloadTask extends AsyncTask<String,Integer,String> {
  private Context context;
  public boolean trwa=true;
  @Override
  protected void onPostExecute(String unused)    {
    trwa=false;
  }
  public DownloadTask(Context context) {
    this.context = context;
  }
  @Override
  protected String doInBackground(String... sUrl) {
        ...
  }
}
```

52

```
protected String doInBackground(String... sUrl) {
  InputStream input = null;    OutputStream output = null;
  HttpURLConnection connection = null;
  try {
    URL url = new URL(sUrl[0]);
    connection = (HttpURLConnection) url.openConnection();
    connection.connect();
    if (connection.getResponseCode()!= HttpURLConnection.HTTP_OK)   {
      return "KOD: " + connection.getResponseCode()
      + " " + connection.getResponseMessage();
    }
    int fileLength = connection.getContentLength();
    input = connection.getInputStream();
    output = new FileOutputStream("/sdcard/file_name.png");
    byte data[] = new byte[4096];  long total = 0;  int count;
    while ((count = input.read(data)) != -1) {
      if (isCancelled()) { input.close();  return null;   }
      total += count;
      if (fileLength>0)publishProgress((int)(total*100/ fileLength));
      output.write(data, 0, count);
    }
  } catch (Exception e) { return e.toString(); } finally {
  try { if (output != null) output.close();
        if (input != null)  input.close();
  } catch (IOException ignored) {}
  if (connection != null) connection.disconnect();
    }    return null; }
```

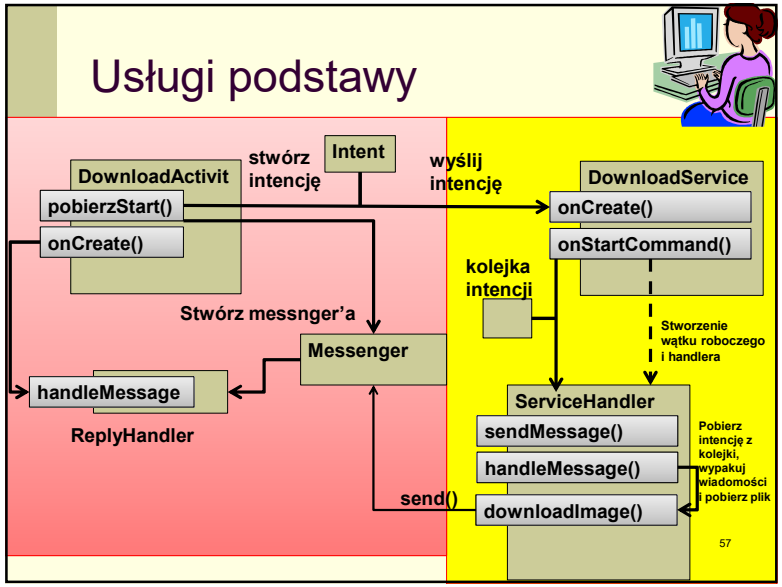
```
public class DownloadService extends Service {
  private volatile Looper mServiceLooper;
  private volatile ServiceHandler mServiceHandler;
  private final class ServiceHandler extends Handler
  {
    public ServiceHandler(Looper looper) {super(looper);}
    public void handleMessage(Message msg) {
      downloadImage((Intent) msg.obj);
      stopSelf(msg.arg1);
  }
  public void downloadImage(Intent intent) {
    Message msg = Message.obtain();//Zwraca obiekt wiadomości z puli globalnej
    msg.arg1=0;
    DownloadTask downloadTask=new DownloadTask(DownloadService.this);
    downloadTask.execute(intent.getDataString());
    while(downloadTask.trwa);
    Bundle bundle = new Bundle();
    String pathname="/sdcard/file_name.png";
    bundle.putString("PATHNAME", pathname);
    msg.setData(bundle);
    Messenger messenger=(Messenger) intent.getExtras().get("MESSENGER");
    try { messenger.send(msg); } catch (RemoteException e) {
              e.printStackTrace();
    }
  }
}
```

```
public class DownloadService extends Service {
...
    public void onCreate(){
        super.onCreate();
        HandlerThread thread= new HandlerThread("DownloadService");
        thread.start();
        mServiceLooper= thread.getLooper();
        mServiceHandler= new ServiceHandler(mServiceLooper);
    }

    public int onStartCommand(Intent intent, int f, int startId){
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1=startId;
        msg.obj=intent;
        mServiceHandler.sendMessage(msg);
        return START_NOT_STICKY;
    }
    public void onDestroy(){
        mServiceLooper.quit();
    }
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Manifest:

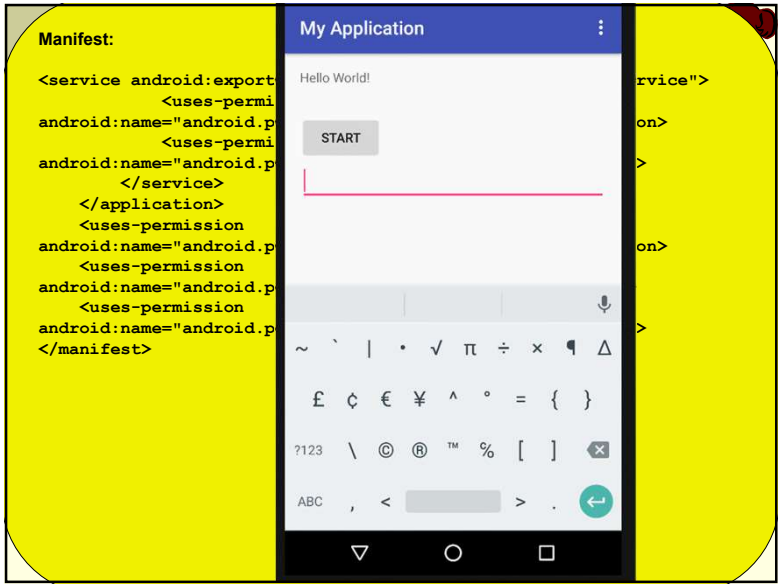
```
<service android:exported="true" android:name=".DownloadService">
            <uses-permission
android:name="android.permission.INTERNET"></uses-permission>
            <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        </service>
    </application>
    <uses-permission
android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Usługi podstawy



Manifest:

```
<service android:export
            <uses-permi
android:name="android.p
            <uses-permi
android:name="android.p
        </service>
    </application>
    <uses-permission
android:name="android.p
    <uses-permission
android:name="android.p
    <uses-permission
android:name="android.p
</manifest>
```



Bound service

`Bound service` wymaga dostarczenia interfejsu `IBinder`, który klient wykorzystuje do komunikacji z usługą.
Metody definiowania interfejsu:

Dziedziczenie po klasie `Binder`
W przypadku, gdy usługa jest dostępna tylko dla własnej aplikacji i wykonuje się w tym samym procesie, powinno się tworzyć interfejs poprzez dziedziczenie po klasie `Binder`. Interfejs jest przekazywany metodą `onBind()`.
Klient otrzymuje obiekt klasy `Binder` i wykorzystuje go do wywoływania publicznych metod obiektu `Binder` lub `Service`.

Wykorzystanie `Messenger'a`
W przypadku, gdy interfejs musi działać pomiędzy różnymi procesami należy wykorzystać `Messenger'a`. Usługa definiuje `Handler'a`, który odpowiada na różne typy wiadomości.
Dodatkowo klient może zdefiniować `Messenger'a` tak, aby usługa przesyłała informacje zwrotne.

59

Bound service

Wykorzystanie `AIDL`
Android Interface Definition Language (AIDL) wykorzystuje technikę podobną do Messenger'a umożliwiając obsługę wielu żądań jednocześnie (Messenger tworzy kolejkę żądań obsługiwanych w jednym wątku). W celu wykorzystania AIDL należy stworzyć plik .aidl definiujący interfejs programistyczny.

60

Bound service

Dziedziczenie po `Binder` (metoda działa wyłącznie dla usługi w tym samym procesie co aplikacja). Lista kroków:

1. Należy utworzyć instancję klasy `Binder` w usłudze, która spełnia jeden z warunków:
   - Zawiera zbiór publicznych metod, które klient może wywołać,
   - Zwraca instancję bieżącej usługi, która zawiera zbiór publicznych metod możliwych do wykonania przez klienta,
   - Zwraca instancję innej klasy zawartej w usłudze, która zawiera listę publicznych metod dla klienta.

2. Zwrócić instancję `Binder` w metodzie `onBind()` (metoda typu callback)

3. W kliencie – odczytać instancję `Binder'a` z metody `onServiceConnected` (metoda typu callback interfejsu `ServiceConnection`)

61

Bound service

`Bound service`
Usługa i klient muszą być umieszczone w tej samej aplikacji i procesie.

Klasa dziedzicząca po `Binder` dostarcza metodę `getService()` w celu pobrania bieżącej instancji usługi lokalnej (umożliwia to klientowi wywoływanie publicznych metod usługi).

`IBinder` - interfejs dla wywoływania metod obiektów zdalnych.

`ServiceConnection` – interfejs do monitorowania stanu usługi. Definiuje metody wywoływane w konkretnych sytuacjach:

- `onServiceConnected(ComponentName` name, `IBinder` service) – metoda wywoływana gdy połączenie z usługą zostało nawiązane (`IBinder` kanału komunikacji).

- `onServiceDisconnected(ComponentName` name) – wywoływana, gdy połączenie z usługą zostało zerwane.

```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >

        <service android:name=".LocalService"/>
```

```
public class LocalService extends Service {
    private final IBinder mBinder = new LocalBinder();
    public class LocalBinder extends Binder {
        LocalService getService() {
            return LocalService.this;
        }    }
    @Override
    public IBinder onBind(Intent intent) { return mBinder;  }
    public String getC(String tekst) {
        char x[] = tekst.toCharArray();
        for(int i=0; i<x.length; i++) {
            int n =  x[i];
            n += 3;
            x[i] = (char) n;
        }
        return new String(x);    }
    public String getE(String tekst) {
        char x[] = tekst.toCharArray();
        for(int i=0; i<x.length; i++) {
            int n =  x[i];
            n -= 3;
            x[i] = (char) n;
        }
        return new String(x);
    }}
```

Bound service

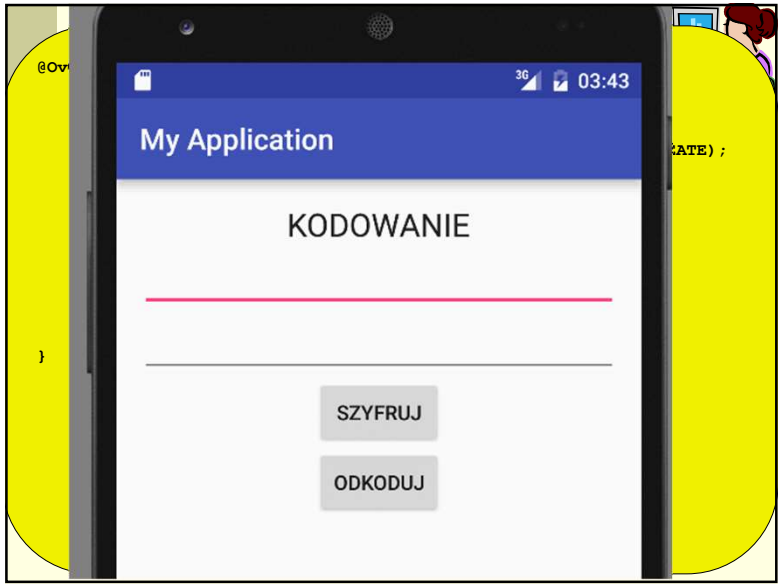
```
public class MainActivity extends AppCompatActivity {
  LocalService mService;
  boolean mBound=false;
  private ServiceConnection mConnection = new ServiceConnection(){
  @Override
  public void onServiceConnected(ComponentName className,IBinder service){
    LocalService.LocalBinder binder = (LocalService.LocalBinder) service;
    mService = binder.getService();
    mBound = true;
  }
  @Override
  public void onServiceDisconnected(ComponentName arg0) {
    mBound = false;
  }
};
```

65

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b1= (Button) findViewById(R.id.button);
        Button b2= (Button) findViewById(R.id.button2);
        b1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mBound) {
                    EditText e1 = (EditText)
findViewById(R.id.editText);
                    EditText e2 = (EditText)
findViewById(R.id.editText2);

e2.setText(mService.getC(e1.getText().toString()));
                }
            }
        });
```

```
b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mBound) {
                    EditText e1 = (EditText)
findViewById(R.id.editText);
                    EditText e2 = (EditText)
findViewById(R.id.editText2);

e2.setText(mService.getE(e1.getText().toString()));
                }
            }
        });

    }
```

```
@Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }
    @Override
    protected void onStop() {
        super.onStop();
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }
}
```



Bound service

Bound service – wykorzystanie `Messenger'a`
Gdy usługa musi komunikować się z procesem zdalnym należy wykorzystać obiekt `Messenger`.
Kroki:

1. **Usługa musi implementować `Hanlder'a` (otrzymuje callback od klienta)**
2. **Usługa wykorzystuje `Handlera` w celu stworzenia `Messenger` (który dostarcza `Handlera` usługi)**
3. **Messenger tworzy `IBinder'a` (który jest zwracany do klienta z `onBind()`)**
4. **Klient używa `IBinder` w celu utworzenia `Messenger'a` (odwołującego się do `Handler'a` usługi) – klient wykorzystuje go w celu przekazywania wiadomości do usługi**
5. **Usługa otrzymuje wiadomości poprzez `Handler'a` (metoda `handleMessage()`)**

70

Bound service - schemat

`bindService()`

- Komponent aplikacji wywołuje metodę w celu połączenia się z usługą

`onBind()`

- metoda wywoływana przez system. Zwraca obiekt IBinder umożliwiający interakcję

71

Bound service

Proces bindowania klienta z usługą przebiega asynchronicznie - powrót z metody `bindService()` następuje natychmiastowo (bez wrócenia obiektu `IBinder` do klienta).

W celu otrzymania obiektu `IBinder`, klient musi utworzyć instancję interfesu `ServiceConnection` i przekazać go jako parametr metody `bindService()`.

Obiekt `ServiceConnection` będzie zawierał metody, które wywołane zostaną przez system w celu dostarczenia obiektu `IBinder`.

72

Bound service

Implementacja ServiceConnection metody: onServiceConnected, onServiceDisconnected

unbindService() - w celu usunięcia połączenia z usługą

Wywołanie metody bindService z przekazanym obiektem ServiceConnection

System wywołuje onServiceConnected (od tego momentu możliwa jest komunikacja z usługą)

Android manifest:

```
...
<service
  android:name=".RemoteService"/>
...
```

```
public class RemoteService extends Service {
  private double a(double x, double y) {return x+y;}
  private double b(double x, double y) {return x*y;}
  class HandlerUslugi extends Handler {
    @Override
    public void handleMessage(Message msg) {
      Bundle dane = msg.getData();
      double w = 0;
      switch (msg.what) {
        case 1:
          w=a(dane.getDouble("arg1"),dane.getDouble("arg2")); break;
        case 2:
          w=b(dane.getDouble("arg1"),dane.getDouble("arg2")); break;
        default:
          w = 0;
      }
      Intent intent = new Intent(Intent.ACTION_ANSWER);
      intent.putExtra("wynik", String.valueOf(w));
      intent.setPackage("com.example.administrator.myapplication");
      sendBroadcast(intent);
      super.handleMessage(msg);
    }
  }
  final Messenger mMessenger = new Messenger(new HandlerUslugi());
    @Override
    public IBinder onBind(Intent i) {return mMessenger.getBinder();}
}
```

```
public class MainActivity extends AppCompatActivity {
  private BroadcastReceiver result = new BroadcastReceiver() {
    @Override
    public void onReceive(Context c, Intent i) {
      Bundle extras = i.getExtras();
      String extraStr=extras.getString("wynik");
      if(extras!=null) {
        TextView t = (TextView) findViewById(R.id.textView2);
        t.setText(extraStr);
      }
    }
  };
  Messenger mService = null;
  boolean mBound;
  private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,IBinder
service) {
      mService = new Messenger(service);
      mBound = true;
    }
    public void onServiceDisconnected(ComponentName className) {
      mService = null;
      mBound = false;
    }
  };
```

```
public void dodaj(double a, double b) {
  if (!mBound) return;
  Message msg = Message.obtain(null, 1, 0, 0);
  Bundle x = new Bundle();
  x.putDouble("arg1", a);
  x.putDouble("arg2", b);
  msg.setData(x);
  try {  mService.send(msg); } catch (RemoteException e) { /*...*/}
}
public void mnoz(double a, double b) {
  if (!mBound) return;
  Message msg = Message.obtain(null, 2, 0, 0);
  Bundle x = new Bundle();
  x.putDouble("arg1", a);
  x.putDouble("arg2", b);
  msg.setData(x);
  try { mService.send(msg);} catch (RemoteException e) {/*...*/ }
}
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Button b1= (Button) findViewById(R.id.button);
  Button b2= (Button) findViewById(R.id.button2);
```

```
b1.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    if (mBound) {
      EditText e1 = (EditText) findViewById(R.id.editText);
      EditText e2 = (EditText) findViewById(R.id.editText2);
      dodaj(Double.valueOf(e1.getText().toString()),
      Double.valueOf(e2.getText().toString()));
    }
  }
});
b2.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    if (mBound) {
      EditText e1 = (EditText) findViewById(R.id.editText);
      EditText e2 = (EditText) findViewById(R.id.editText2);
      mnoz(Double.valueOf(e1.getText().toString()),
      Double.valueOf(e2.getText().toString()));
    }
  }
});
IntentFilter filter=new IntentFilter(Intent.ACTION_ANSWER);
registerReceiver(result, filter);
}
```

```
@Override
protected void onStart() {
  super.onStart();
  bindService(new Intent(this, RemoteService.class),
              mConnection,
              Context.BIND_AUTO_CREATE);
  }
  @Override
  protected void onStop() {
    super.onStop();
    if (mBound) {
      unbindService(mConnection);
      mBound = false;
    }
  }

}
```