Android Binder Inter Process Communication (IPC) with AIDL

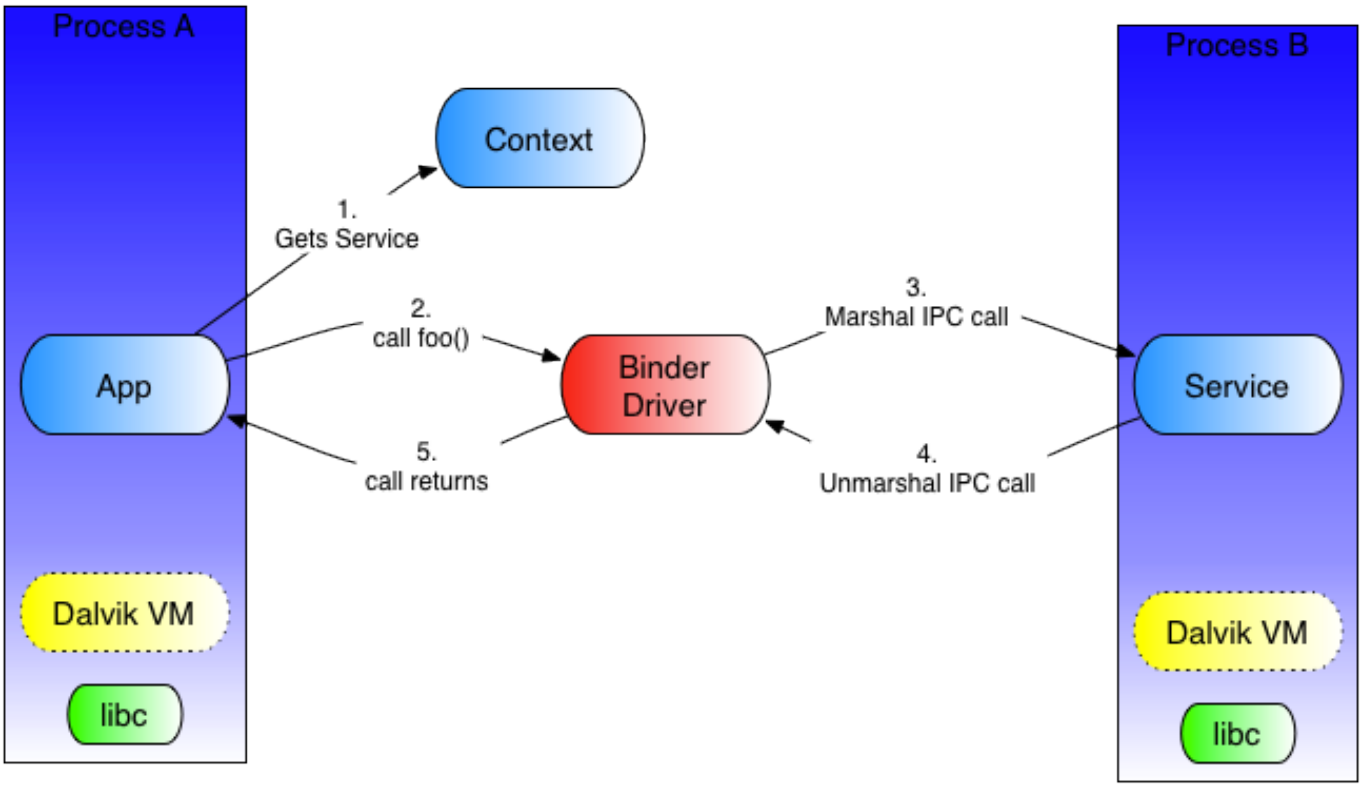
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Why IPC?

- Each Android application runs in a separate process
  - Android application-framework services also run in a separate process called `systemserver`
- Often we wish to make use of services offered by other applications, or we simply wish to expose our applications' capabilities to 3rd party apps
  - Even Android's `Intent`-based IPC-like mechanism (used for starting activities and services, as well as delivering events), is internally based on Binder
- By design (for security reasons), processes cannot directly access each other's data
- To cross the process boundaries, we need support of a inter-process communication transport mechanism, which handles passing of data from one process (caller) to another (callee)
  - Caller's data is *marshaled* into tokens that IPC understands, copied to callee's process, and finally *unmarshaled* into what callee expects
  - Callee's response is also *marshaled*, copied to caller's process where it is *unmarshaled* into what caller expects
  - Marshaling/unmarshaling is automatically provided by the IPC mechanism

2 What is Binder?



- Binder provides a lightweight remote procedure call (RPC) mechanism designed for high performance when performing in-process and cross-process calls (IPC)
- Binder-capable services are described in Android Interface Definition Language (AIDL), not unlike other IDL languages
- Since Binder is provided as a Linux driver, the services can be written in both C/C++ as well as Java

  - Most Android services are written in Java
- All caller calls go through Binder's `transact()` method, which automatically marshals the arguments and return values via `Parcel` objects
  - `Parcel` is a generic buffer of data (decomposed into primitives) that also maintains some meta-data about its contents - such as object references to ensure object identity across processes
  - Caller calls to `transact()` are by default synchronous - i.e. provide the same semantics as a local method call
    * On callee side, the Binder framework maintains a pool of transaction threads, which are used to handle the incoming IPC requests (unless the call is local, in which case the same thread is used)
    * Callee methods can be marked as `oneway`, in which case caller calls do not block (i.e. calls return immediately)
- Callee' mutable state needs to be thread-safe - since callee's can accept concurrent requests from multiple callers
- The Binder system also supports recursion across processes - i.e. behaves the same as recursion semantics when calling methods on local objects

3 What is AIDL?

- Android Interface Definition Language is a Android-specific language for defining Binder-based service interafaces
- AIDL follows Java-like interface syntax and allows us to declare our "business" methods
- Each Binder-based service is defined in a separate `.aidl` file, typically named `IFooService.aidl`, and saved in the `src/` directory

  src/com/example/app/IFooService.aidl

```
package com.example.app;
import com.example.app.Bar;
interface IFooService {
    void save(inout Bar bar);
    Bar getById(int id);
    void delete(in Bar bar);
    List<Bar> getAll();
}
```

- The `aidl` build tool (part of Android SDK) is used to extract a real Java interface (along with a `Stub` providing Android's `android.os.IBinder`) from each `.aidl` file and place it into our `gen/` directory

  gen/com/example/app/IFooService.java

```
package com.example.app;
public interface IFooService extends android.os.IInterface
{
    public static abstract class Stub extends android.os.Binder
        implements com.example.app.IFooService {
        ...
        public static com.example.app.IFooService asInterface(
            android.os.IBinder obj) {
            ...
        }
        public android.os.IBinder asBinder() {
            return this;
        }
        ...
    }
    void save(com.example.app.Bar bar) throws android.os.RemoteException;
    com.example.app.Bar getById(int id) throws android.os.RemoteException;
    void delete(com.example.app.Bar bar) throws android.os.RemoteException;
    java.util.List<Bar> getAll() throws android.os.RemoteException;
}
```

Note
Eclipse ADT automatically calls `aidl` for each `.aidl` file that it finds in our `src/` directory

- AIDL supports the following types:
  - `null`
  - `boolean`, `boolean[]`, `byte`, `byte[]`, `char[]`, `int`, `int[]`, `long`, `long[]`, `float`, `float[]`, `double`, `double[]`
  - `java.lang.CharSequence`, `java.lang.String`
  - `java.io.FileDescriptor`
    - Gets transferred as a dup of the original file descriptor - while the fd is different, it points to the same underlying stream and position
  - `java.io.Serializable` (not efficient)
  - `java.util.List` (of supported types, including generic definitions)
    - Internally, Binder always uses `java.util.ArrayList` as the concrete implementation
  - `java.util.Map` (of supported types)
    - Internally, Binder always uses `java.util.HashMap` as the concrete implementation
  - `java.lang.Object[]` (supporting objects of the same type defined here, but also primitive wrappers)
  - `android.util.SparseArray`, `android.util.SparseBooleanArray`
  - `android.os.Bundle`
  - `android.os.IBinder`, `android.os.IInterface`
    - The contents of these objects are not actually transferred - instead a special token serving as a self-reference is written
    - Reading these objects from a parcel returns a handle to the original object that was written
  - `android.os.Parcelable` (allowing for custom types):
    src/com/example/app/Bar.java

```
package com.example.app;

import android.os.Parcel;
import android.os.Parcelable;

public class Bar implements Parcelable {
    private int id;
    private String data;

    public Bar(Parcel parcel) {
        this.id = parcel.readInt();
        this.data = parcel.readString();
    }

    // getters and setters omitted
    ...

    public void writeToParcel(Parcel parcel, int flags) {
        parcel.writeInt(this.id);
        parcel.writeString(this.data);
    }

    public static final Parcelable.Creator<Bar> CREATOR = new Parcelable.Creator<Bar>()  ↩
        {
        public Bar createFromParcel(Parcel in) {
            return new Bar(in);
        }
        public Bar[] newArray(int size) {
```

```
            return new Bar[size];
        }
    };
}
```

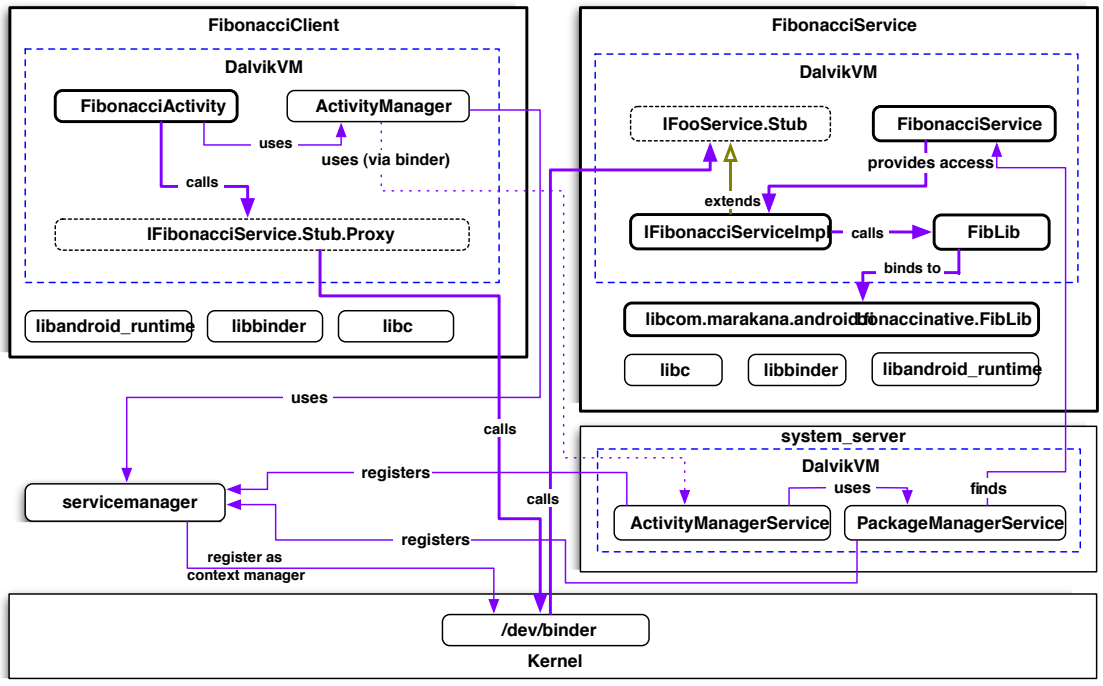
* These custom classes have to be declared in their own (simplified) `.aidl` files
  src/com/example/app/Bar.aidl

```
package com.example.app;
parcelable Bar;
```

> **Note**
> AIDL-interfaces have to `import` parcelable custom classes even if they are in the same package. In the case of the previous example, `src/com/example/app/IFooService.aidl` would have to `import com.example.app.Bar;` if it makes any references to `com.example.app.Bar` even though they are in the same package.

- AIDL-defined methods can take zero or more parameters, and must return a value or `void`
  - All non-primitive parameters require a *directional tag* indicating which way the data goes: one of: `in`, `out`, or `inout`
    * Direction for primitives is always `in` (can be omitted)
    * The direction tag tells binder when to marshal the data, so its use has direct consequences on performance
- All `.aidl` comments are copied over to the generated Java interface (except for comments before the import and package statements).
- Static fields are not supported in `.aidl` files

4 Building a Binder-based Service and Client



- To demonstrate an Binder-based service and client, we'll create three separate projects:

1. `FibonacciCommon` library project - to define our AIDL interface as well as custom types for parameters and return values
2. `FibonacciService` project - where we implement our AIDL interface and expose it to the clients
3. `FibonacciClient` project - where we connect to our AIDL-defined service and use it

5 FibonacciCommon - Define AIDL Interface and Custom Types

- We start by creating a new Android (library) project, which will host the common API files (an AIDL interface as well as custom types for parameters and return values) shared by the service and its clients
  - Project Name: `FibonacciCommon`
  - Build Target: Android 2.2 (API 8)
  - Package Name: `com.marakana.android.fibonaccicommon`
  - Min SDK Version: 8
  - No need to specify Application name or an activity
- To turn this into a *library project* we need to access project properties → `Android` → `Library` and check `Is Library`
  - We could also manually add `android.library=true` to `FibonacciCommon/default.properties` and refresh the project
- Since library projects are never turned into actual applications (APKs)
  - We can simplify our manifest file:
    FibonacciCommon/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.marakana.android.fibonaccicommon" android:versionCode="1"
        android:versionName="1.0">
</manifest>
```

  - And we can remove everything from `FibonacciCommon/res/` directory (e.g. `rm -fr FibonacciCommon/res/*`)
- We are now ready to create our AIDL interface
  FibonacciCommon/src/com/marakana/android/fibonaccicommon/IFibonacciService.aidl

```
package com.marakana.android.fibonaccicommon;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;

interface IFibonacciService {
    long fibJR(in long n);
    long fibJI(in long n);
    long fibNR(in long n);
    long fibNI(in long n);
    FibonacciResponse fib(in FibonacciRequest request);
}
```

- Our interface clearly depends on two custom Java types, which we have to not only implement in Java, but define in their own `.aidl` files
  FibonacciCommon/src/com/marakana/android/fibonaccicommon/FibonacciRequest.aidl

```
package com.marakana.android.fibonaccicommon;

parcelable FibonacciRequest;
```

FibonacciCommon/src/com/marakana/android/fibonaccicommon/FibonacciRequest.java

```
package com.marakana.android.fibonaccicommon;

import android.os.Parcel;
import android.os.Parcelable;

public class FibonacciRequest implements Parcelable {

    public static final int RECURSIVE_JAVA_TYPE = 1;

    public static final int ITERATIVE_JAVA_TYPE = 2;

    public static final int RECURSIVE_NATIVE_TYPE = 3;

    public static final int ITERATIVE_NATIVE_TYPE = 4;

    private final long n;

    private final int type;

    public FibonacciRequest(long n, int type) {
        this.n = n;
        if (type < RECURSIVE_JAVA_TYPE || type > ITERATIVE_NATIVE_TYPE) {
            throw new IllegalArgumentException("Invalid type: " + type);
        }
        this.type = type;
    }

    private FibonacciRequest(Parcel parcel) {
        this(parcel.readLong(), parcel.readInt());
    }

    public long getN() {
        return n;
    }

    public int getType() {
        return type;
    }

    public int describeContents() {
        return 0;
    }

    public void writeToParcel(Parcel parcel, int flags) {
        parcel.writeLong(this.n);
        parcel.writeInt(this.type);
    }

    public static final Parcelable.Creator<FibonacciRequest> CREATOR = new Parcelable. ↩
        Creator<FibonacciRequest>() {
        public FibonacciRequest createFromParcel(Parcel in) {
            return new FibonacciRequest(in);
        }

        public FibonacciRequest[] newArray(int size) {
            return new FibonacciRequest[size];
        }
    };
}
```

FibonacciCommon/src/com/marakana/android/fibonaccicommon/FibonacciResponse.aidl

```
package com.marakana.android.fibonaccicommon;

parcelable FibonacciResponse;
```

FibonacciCommon/src/com/marakana/android/fibonaccicommon/FibonacciResponse.java

```
package com.marakana.android.fibonaccicommon;

import android.os.Parcel;
import android.os.Parcelable;

public class FibonacciResponse implements Parcelable {

    private final long result;

    private final long timeInMillis;

    public FibonacciResponse(long result, long timeInMillis) {
        this.result = result;
        this.timeInMillis = timeInMillis;
    }

    public FibonacciResponse(Parcel parcel) {
        this(parcel.readLong(), parcel.readLong());
    }

    public long getResult() {
        return result;
    }

    public long getTimeInMillis() {
        return timeInMillis;
    }

    public int describeContents() {
        return 0;
    }

    public void writeToParcel(Parcel parcel, int flags) {
        parcel.writeLong(this.result);
        parcel.writeLong(this.timeInMillis);
    }

    public static final Parcelable.Creator<FibonacciResponse> CREATOR = new Parcelable. ↩
        Creator<FibonacciResponse>() {
        public FibonacciResponse createFromParcel(Parcel in) {
            return new FibonacciResponse(in);
        }

        public FibonacciResponse[] newArray(int size) {
            return new FibonacciResponse[size];
        }
    };
}
```

- Finally we are now ready to take a look at our generated Java interface

FibonacciCommon/gen/com/marakana/android/fibonaccicommon/IFibonacciService.java

```
package com.marakana.android.fibonaccicommon;
public interface IFibonacciService extends android.os.IInterface
```

```
{
    public static abstract class Stub extends android.os.Binder
        implements com.marakana.android.fibonacci.IFibonacciService {
        ...
        public static com.marakana.android.fibonacci.IFibonacciService asInterface(
            android.os.IBinder obj) {
            ...
        }
        public android.os.IBinder asBinder() {
            return this;
        }
        ...
    }

    public long fibJR(long n) throws android.os.RemoteException;
    public long fibJI(long n) throws android.os.RemoteException;
    public long fibNR(long n) throws android.os.RemoteException;
    public long fibNI(long n) throws android.os.RemoteException;
    public com.marakana.android.fibonaccicommon.FibonacciResponse fib(
        com.marakana.android.fibonaccicommon.FibonacciRequest request)
        throws android.os.RemoteException;
}
```

6 FibonacciService - Implement AIDL Interface and Expose It To Our Clients

- We start by creating a new Android project, which will host the our AIDL Service implementation as well as provide a mechanism to access (i.e. bind to) our service implementation
  - Project Name: `FibonacciService`
  - Build Target: Android 2.2 (API 8)
  - Package Name: `com.marakana.android.fibonacciservice`
  - Application name: Fibonacci Service
  - Min SDK Version: 8
  - No need to specify an Android activity
- We need to link this project to the `FibonacciCommon` in order to be able to access the common APIs: project properties → `Android` → `Library` → `Add...` → `FibonacciCommon`
  - As the result, `FibonacciService/default.properties` now has `android.library.reference.1=../Fibona` and `FibonacciService/.classpath` and `FibonacciService/.project` also link to `FibonacciCommon`
- Our service will make use of the `com.marakana.android.fibonaccinative.FibLib`, which provides the actual implementation of the Fibonacci algorithms
- We copy (or move) this Java class (as well as the `jni/` implementation) from the `FibonacciNative` project
  - Don't forget to run `ndk-build` under `FibonacciService/` in order to generate the required native library

7 Implement AIDL Interface

- We are now ready to implement our AIDL-defined interface by extending from the auto-generated `com.marakana.android.fib` (which in turn extends from `android.os.Binder`)

  FibonacciService/src/com/marakana/android/fibonacciservice/IFibonacciServiceImpl.java

```
package com.marakana.android.fibonacciservice;

import android.util.Log;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;
import com.marakana.android.fibonaccicommon.IFibonacciService;
import com.marakana.android.fibonaccinative.FibLib;

public class IFibonacciServiceImpl extends IFibonacciService.Stub {
    private static final String TAG = "IFibonacciServiceImpl";

    public long fibJI(long n) {
        Log.d(TAG, "fibJI()");
        return FibLib.fibJI(n);
    }

    public long fibJR(long n) {
        Log.d(TAG, "fibJR()");
        return FibLib.fibJR(n);
    }

    public long fibNI(long n) {
        Log.d(TAG, "fibNI()");
        return FibLib.fibNI(n);
    }

    public long fibNR(long n) {
        Log.d(TAG, "fibNR()");
        return FibLib.fibNR(n);
    }

    public FibonacciResponse fib(FibonacciRequest request) {
        Log.d(TAG, "fib()");
        long timeInMillis = System.currentTimeMillis();
        long result;
        switch (request.getType()) {
            case FibonacciRequest.ITERATIVE_JAVA_TYPE:
                result = this.fibJI(request.getN());
                break;
            case FibonacciRequest.RECURSIVE_JAVA_TYPE:
                result = this.fibJR(request.getN());
                break;
            case FibonacciRequest.ITERATIVE_NATIVE_TYPE:
                result = this.fibNI(request.getN());
                break;
            case FibonacciRequest.RECURSIVE_NATIVE_TYPE:
                result = this.fibNR(request.getN());
                break;
            default:
                return null;
        }
        timeInMillis = System.currentTimeMillis() - timeInMillis;
        return new FibonacciResponse(result, timeInMillis);
    }
}
```

8 Expose our AIDL-defined Service Implementation to Clients

- In order for clients (callers) to use our service, they first need to bind to it.
- But in order for them to *bind* to it, we first need to expose it via our own `android.app.Service`'s `onBind(Intent)` implementation

FibonacciService/src/com/marakana/android/fibonacciservice/FibonacciService.java

```
package com.marakana.android.fibonacciservice;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class FibonacciService extends Service { // ❶

    private static final String TAG = "FibonacciService";

    private IFibonacciServiceImpl service; // ❷

    @Override
    public void onCreate() {
        super.onCreate();
        this.service = new IFibonacciServiceImpl(); // ❸
        Log.d(TAG, "onCreate()'ed"); // ❹
    }

    @Override
    public IBinder onBind(Intent intent) {
        Log.d(TAG, "onBind()'ed"); // ❺
        return this.service; // ❻
    }

    @Override
    public boolean onUnbind(Intent intent) {
        Log.d(TAG, "onUnbind()'ed"); // ❼
        return super.onUnbind(intent);
    }

    @Override
    public void onDestroy() {
        Log.d(TAG, "onDestroy()'ed");
        this.service = null;
        super.onDestroy();
    }
}
```

❶ We create yet another "service" object by extending from `android.app.Service`. The purpose of `FibonacciService` object is to provide access to our Binder-based `IFibonacciServiceImpl` object.

❷ Here we simply declare a local reference to `IFibonacciServiceImpl`, which will act as a singleton (i.e. all clients will share a single instance). Since our `IFibonacciServiceImpl` does not require any special initialization, we could instantiate it at this point, but we choose to delay this until the `onCreate()` method.

❸ Now we instantiate our `IFibonacciServiceImpl` that we'll be providing to our clients (in the `onBind(Intent)` method). If our `IFibonacciServiceImpl` required access to the `Context` (which it doesn't) we could pass a reference to `this` (i.e. `android.app.Service`, which implements `android.content.Context`) at this point. Many Binder-based services use `Context` in order to access other platform functionality.

❻ This is where we provide access to our `IFibonacciServiceImpl` object to our clients. By design, we chose to have only one instance of `IFibonacciServiceImpl` (so all clients share it) but we could also provide each client with their own instance of `IFibonacciServiceImpl`.

❹, ❺, ❼ We just add some logging calls to make it easy to track the life-cycle of our service.

- Finally, we register our `FibonacciService` in our `AndroidManifest.xml`, so that clients can find it

FibonacciService/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.marakana.android.fibonacciservice" android:versionCode="1"
        android:versionName="1.0">
        <uses-sdk android:minSdkVersion="8" />
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <service android:name=".FibonacciService">
                        <intent-filter>
                                <action android:name="com.marakana.android.fibonaccicommon ↩
                                    .IFibonacciService" /> <!-- ❶ -->
                        </intent-filter>
                </service>
        </application>
</manifest>
```

❶ The name of this action is arbitrary, but it is a common convention to use the fully-qualified name of our AIDL-derived interface.

9 FibonacciClient - Using AIDL-defined Binder-based Services

- We start by creating a new Android project, which will server as the client of the AIDL Service we previously implemented
  - Project Name: `FibonacciClient`
  - Build Target: Android 2.2 (API 8)
  - Package Name: `com.marakana.android.fibonacciclient`
  - Application name: Fibonacci Client
  - Create activity: `FibonacciActivity`
    - We'll repurpose most of this activity's code from `FibonacciNative`
  - Min SDK Version: 8
- We need to link this project to the `FibonacciCommon` in order to be able to access the common APIs: project properties → `Android` → `Library` → `Add...` → `FibonacciCommon`
  - As the result, `FibonacciClient/default.properties` now has `android.library.reference.1=../Fibonac` and `FibonacciClient/.classpath` and `FibonacciClient/.project` also link to `FibonacciCommon`
  - As an alternative, we could've avoided creating `FibonacciCommon` in the first place
    - `FibonacciService` and `FibonacciClient` could have each had a copy of: `IFibonacciService.aidl`, `FibonacciRequest.aidl`, `FibonacciResponse.aidl`, `FibonacciResult.java`, and `FibonacciResponse`
    - But we don't like duplicating source code (even though the binaries do get duplicated at runtime)
- Our client will make use of the string resources and layout definition from `FibonacciNative` application

FibonacciClient/res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="hello">Get Your Fibonacci Here!</string>
        <string name="app_name">Fibonacci Client</string>
        <string name="input_hint">Enter N</string>
        <string name="input_error">Numbers only!</string>
        <string name="button_text">Get Fib Result</string>
```

```
        <string name="progress_text">Calculating...</string>
        <string name="fib_error">Failed to get Fibonacci result</string>
        <string name="type_fib_jr">fibJR</string>
        <string name="type_fib_ji">fibJI</string>
        <string name="type_fib_nr">fibNR</string>
        <string name="type_fib_ni">fibNI</string>
</resources>
```

FibonacciClient/res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView android:text="@string/hello" android:layout_height="wrap_content"
                android:layout_width="fill_parent" android:textSize="25sp" android:gravity ↩
                    ="center"/>
        <EditText android:layout_height="wrap_content"
                android:layout_width="match_parent" android:id="@+id/input"
                android:hint="@string/input_hint" android:inputType="number"
                android:gravity="right" />
        <RadioGroup android:orientation="horizontal"
                android:layout_width="match_parent" android:id="@+id/type"
                android:layout_height="wrap_content">
                <RadioButton android:layout_height="wrap_content"
                        android:checked="true" android:id="@+id/type_fib_jr" android:text= ↩
                            "@string/type_fib_jr"
                        android:layout_width="match_parent" android:layout_weight="1" />
                <RadioButton android:layout_height="wrap_content"
                        android:id="@+id/type_fib_ji" android:text="@string/type_fib_ji"
                        android:layout_width="match_parent" android:layout_weight="1" />
                <RadioButton android:layout_height="wrap_content"
                        android:id="@+id/type_fib_nr" android:text="@string/type_fib_nr"
                        android:layout_width="match_parent" android:layout_weight="1" />
                <RadioButton android:layout_height="wrap_content"
                        android:id="@+id/type_fib_ni" android:text="@string/type_fib_ni"
                        android:layout_width="match_parent" android:layout_weight="1" />
        </RadioGroup>
        <Button android:text="@string/button_text" android:id="@+id/button"
                android:layout_width="match_parent" android:layout_height="wrap_content" / ↩
                    >
        <TextView android:id="@+id/output" android:layout_width="match_parent"
                android:layout_height="match_parent" android:textSize="20sp"  ↩
                    android:gravity="center|top"/>
</LinearLayout>
```

- We are now ready to implement our client

FibonacciClient/src/com/marakana/android/fibonacciclient/FibonacciActivity.java

```
package com.marakana.android.fibonacciclient;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.text.TextUtils;
```

```
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;
import com.marakana.android.fibonaccicommon.IFibonacciService;

public class FibonacciActivity extends Activity implements OnClickListener,  ↩
    ServiceConnection {

    private static final String TAG = "FibonacciActivity";

    private EditText input; // our input n

    private Button button; // trigger for fibonacci calcualtion

    private RadioGroup type; // fibonacci implementation type

    private TextView output; // destination for fibonacci result

    private IFibonacciService service; // reference to our service

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        // connect to our UI elements
        this.input = (EditText)super.findViewById(R.id.input);
        this.button = (Button)super.findViewById(R.id.button);
        this.type = (RadioGroup)super.findViewById(R.id.type);
        this.output = (TextView)super.findViewById(R.id.output);
        // request button click call-backs via onClick(View) method
        this.button.setOnClickListener(this);
        // the button will be enabled once we connect to the service
        this.button.setEnabled(false);
    }

    @Override
    protected void onResume() {
        Log.d(TAG, "onResume()'ed");
        super.onResume();
        // Bind to our FibonacciService service, by looking it up by its name
        // and passing ourselves as the ServiceConnection object
        // We'll get the actual IFibonacciService via a callback to
        // onServiceConnected() below
        if (!super.bindService(new Intent(IFibonacciService.class.getName()), this,
                BIND_AUTO_CREATE)) {
            Log.w(TAG, "Failed to bind to service");
        }
    }

    @Override
    protected void onPause() {
        Log.d(TAG, "onPause()'ed");
        super.onPause();
        // No need to keep the service bound (and alive) any longer than
```

```
        // necessary
        super.unbindService(this);
    }

    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d(TAG, "onServiceConnected()'ed to " + name);
        // finally we can get to our IFibonacciService
        this.service = IFibonacciService.Stub.asInterface(service);
        // enable the button, because the IFibonacciService is initialized
        this.button.setEnabled(true);
    }

    public void onServiceDisconnected(ComponentName name) {
        Log.d(TAG, "onServiceDisconnected()'ed to " + name);
        // our IFibonacciService service is no longer connected
        this.service = null;
        // disabled the button, since we cannot use IFibonacciService
        this.button.setEnabled(false);
    }

    // handle button clicks
    public void onClick(View view) {
        // parse n from input (or report errors)
        final long n;
        String s = this.input.getText().toString();
        if (TextUtils.isEmpty(s)) {
            return;
        }
        try {
            n = Long.parseLong(s);
        } catch (NumberFormatException e) {
            this.input.setError(super.getText(R.string.input_error));
            return;
        }

        // build the request object
        int type;
        switch (FibonacciActivity.this.type.getCheckedRadioButtonId()) {
            case R.id.type_fib_jr:
                type = FibonacciRequest.RECURSIVE_JAVA_TYPE;
                break;
            case R.id.type_fib_ji:
                type = FibonacciRequest.ITERATIVE_JAVA_TYPE;
                break;
            case R.id.type_fib_nr:
                type = FibonacciRequest.RECURSIVE_NATIVE_TYPE;
                break;
            case R.id.type_fib_ni:
                type = FibonacciRequest.ITERATIVE_NATIVE_TYPE;
                break;
            default:
                return;
        }
        final FibonacciRequest request = new FibonacciRequest(n, type);

        // showing the user that the calculation is in progress
        final ProgressDialog dialog = ProgressDialog.show(this, "", super
                .getText(R.string.progress_text), true);
        // since the calculation can take a long time, we do it in a separate
        // thread to avoid blocking the UI
        new AsyncTask<Void, Void, String>() {
            @Override
```

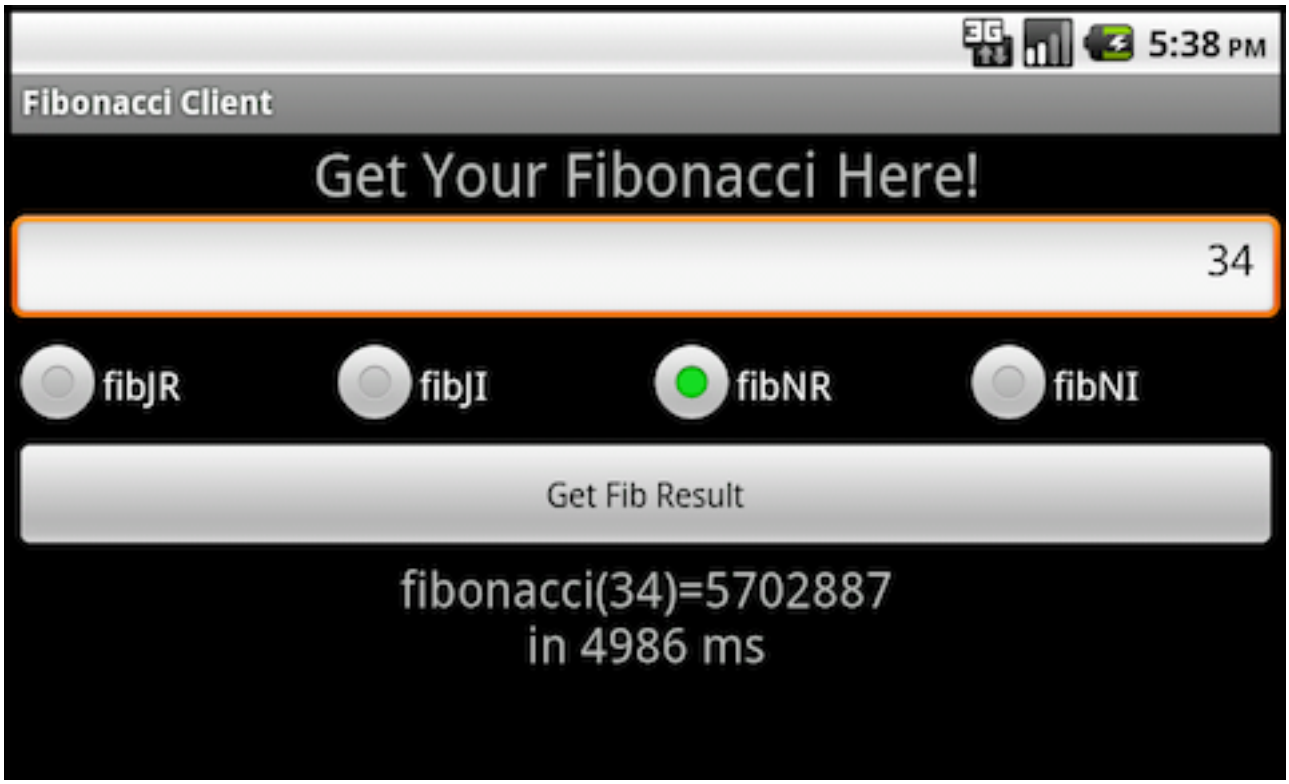
```
            protected String doInBackground(Void... params) {
                // this method runs in a background thread
                try {
                    FibonacciResponse response = FibonacciActivity.this.service.fib( ↩
                        request);
                    // generate the result
                    return String.format("fibonacci(%d)=%d\nin %d ms", n, response. ↩
                        getResult(),
                            response.getTimeInMillis());
                } catch (RemoteException e) {
                    Log.wtf(TAG, "Failed to communicate with the service", e);
                    return null;
                }
            }

            @Override
            protected void onPostExecute(String result) {
                // get rid of the dialog
                dialog.dismiss();
                if (result == null) {
                    // handle error
                    Toast.makeText(FibonacciActivity.this, R.string.fib_error, Toast. ↩
                        LENGTH_SHORT)
                            .show();
                } else {
                    // show the result to the user
                    FibonacciActivity.this.output.setText(result);
                }
            }
        }.execute(); // run our AsyncTask
    }
}
```

- Our activity should already be registered in our `AndroidManifest.xml` file

  FibonacciClient/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        android:versionCode="1" android:versionName="1.0"
        package="com.marakana.android.fibonacciclient">
        <uses-sdk android:minSdkVersion="8" />
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name="com.marakana.android.fibonacciclient. ↩
                    FibonacciActivity"
                        android:label="@string/app_name">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER"  ↩
                                    />
                        </intent-filter>
                </activity>
        </application>
</manifest>
```

- And the result should look like



10 Async-IPC via Binder

- Binder allows for the asynchronous communication between the client and its service via the `oneway` declaration on the AIDL interface
- Of course, we still care about the result, so generally async calls are used with call-backs - typically through listeners
- When clients provide a reference to themselves as call-back listeners, then the roles reverse at the time the listeners are called: clients' listeners become the services, and services become the clients to those listeners
- This is best explained via an example (based on Fibonacci)

10.1 FibonacciCommon - Defining a `oneway` AIDL Service

- First, we need a listener, which itself is a `oneway` AIDL-defined "service":

  FibonacciCommon/src/com/marakana/android/fibonaccicommon/IFibonacciServiceResponseListener.aidl:

```
package com.marakana.android.fibonaccicommon;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;

oneway interface IFibonacciServiceResponseListener {
    void onResponse(in FibonacciResponse response);
}
```

- Now we can create a our `oneway` (i.e. asynchronous) interface:

  FibonacciCommon/src/com/marakana/android/fibonaccicommon/IFibonacciService.aidl:

```
package com.marakana.android.fibonaccicommon;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;
import com.marakana.android.fibonaccicommon.IFibonacciServiceResponseListener;

oneway interface IFibonacciService {
    void fib(in FibonacciRequest request, in IFibonacciServiceResponseListener listener);
}
```

10.2 FibonacciService - Implementing our async AIDL service

- The implementation of our service invokes the listener, as opposed to returning a result:

  FibonacciService/src/com/marakana/android/fibonacciservice/IFibonacciServiceImpl.java:

```
package com.marakana.android.fibonacciservice;

import android.os.RemoteException;
import android.util.Log;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;
import com.marakana.android.fibonaccicommon.IFibonacciService;
import com.marakana.android.fibonaccicommon.IFibonacciServiceResponseListener;
import com.marakana.android.fibonaccinative.FibLib;

public class IFibonacciServiceImpl extends IFibonacciService.Stub {
    private static final String TAG = "IFibonacciServiceImpl";

    @Override
    public void fib(FibonacciRequest request, IFibonacciServiceResponseListener listener)
            throws RemoteException {
        long n = request.getN();
        Log.d(TAG, "fib(" + n + ")");
        long timeInMillis = System.currentTimeMillis();
        long result;
        switch (request.getType()) {
            case FibonacciRequest.ITERATIVE_JAVA_TYPE:
                result = FibLib.fibJI(n);
                break;
            case FibonacciRequest.RECURSIVE_JAVA_TYPE:
                result = FibLib.fibJR(n);
                break;
            case FibonacciRequest.ITERATIVE_NATIVE_TYPE:
                result = FibLib.fibNI(n);
                break;
            case FibonacciRequest.RECURSIVE_NATIVE_TYPE:
                result = FibLib.fibNR(n);
                break;
            default:
                result = 0;
        }
        timeInMillis = System.currentTimeMillis() - timeInMillis;
        Log.d(TAG, String.format("Got fib(%d) = %d in %d ms", n, result, timeInMillis));
        listener.onResponse(new FibonacciResponse(result, timeInMillis));
    }
}
```

Note
The service will not block waiting for the listener to return, because the listener itself is also `oneway`.

10.3 FibonacciClient - Implementing our async AIDL client

- Finally, we implement our client, which itself has to also implement a listener as a Binder service:

FibonacciClient/src/com/marakana/android/fibonacciclient/FibonacciActivity.java:

```
package com.marakana.android.fibonacciclient;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteException;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioGroup;
import android.widget.TextView;

import com.marakana.android.fibonaccicommon.FibonacciRequest;
import com.marakana.android.fibonaccicommon.FibonacciResponse;
import com.marakana.android.fibonaccicommon.IFibonacciService;
import com.marakana.android.fibonaccicommon.IFibonacciServiceResponseListener;

public class FibonacciActivity extends Activity implements OnClickListener,  ↩
    ServiceConnection {

    private static final String TAG = "FibonacciActivity";

    // the id of a message to our response handler
    private static final int RESPONSE_MESSAGE_ID = 1;

    // the id of a progress dialog that we'll be creating
    private static final int PROGRESS_DIALOG_ID = 1;

    private EditText input; // our input n

    private Button button; // trigger for fibonacci calcualtion

    private RadioGroup type; // fibonacci implementation type

    private TextView output; // destination for fibonacci result

    private IFibonacciService service; // reference to our service

    // the responsibility of the responseHandler is to take messages
    // from the responseListener (defined below) and display their content
    // in the UI thread
```

```
    private final Handler responseHandler = new Handler() {
        @Override
        public void handleMessage(Message message) {
            switch (message.what) {
                case RESPONSE_MESSAGE_ID:
                    Log.d(TAG, "Handling response");
                    FibonacciActivity.this.output.setText((String) message.obj);
                    FibonacciActivity.this.removeDialog(PROGRESS_DIALOG_ID);
                    break;
            }
        }
    };

    // the responsibility of the responseListener is to receive call-backs
    // from the service when our FibonacciResponse is available
    private final IFibonacciServiceResponseListener responseListener = new  ↩
        IFibonacciServiceResponseListener.Stub() {

        // this method is executed on one of the pooled binder threads
        @Override
        public void onResponse(FibonacciResponse response) throws RemoteException {
            String result = String.format("%d in %d ms", response.getResult(),
                    response.getTimeInMillis());
            Log.d(TAG, "Got response: " + result);
            // since we cannot update the UI from a non-UI thread,
            // we'll send the result to the responseHandler (defined above)
            Message message = FibonacciActivity.this.responseHandler.obtainMessage(
                    RESPONSE_MESSAGE_ID, result);
            FibonacciActivity.this.responseHandler.sendMessage(message);
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.main);
        // connect to our UI elements
        this.input = (EditText) super.findViewById(R.id.input);
        this.button = (Button) super.findViewById(R.id.button);
        this.type = (RadioGroup) super.findViewById(R.id.type);
        this.output = (TextView) super.findViewById(R.id.output);
        // request button click call-backs via onClick(View) method
        this.button.setOnClickListener(this);
        // the button will be enabled once we connect to the service
        this.button.setEnabled(false);
    }

    @Override
    protected void onStart() {
        Log.d(TAG, "onStart()'ed");
        super.onStart();
        // Bind to our FibonacciService service, by looking it up by its name
        // and passing ourselves as the ServiceConnection object
        // We'll get the actual IFibonacciService via a callback to
        // onServiceConnected() below
        if (!super.bindService(new Intent(IFibonacciService.class.getName()), this,
                BIND_AUTO_CREATE)) {
            Log.w(TAG, "Failed to bind to service");
        }
    }

    @Override
```

```
    protected void onStop() {
        Log.d(TAG, "onStop()'ed");
        super.onStop();
        // No need to keep the service bound (and alive) any longer than
        // necessary
        super.unbindService(this);
    }

    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d(TAG, "onServiceConnected()'ed to " + name);
        // finally we can get to our IFibonacciService
        this.service = IFibonacciService.Stub.asInterface(service);
        // enable the button, because the IFibonacciService is initialized
        this.button.setEnabled(true);
    }

    public void onServiceDisconnected(ComponentName name) {
        Log.d(TAG, "onServiceDisconnected()'ed to " + name);
        // our IFibonacciService service is no longer connected
        this.service = null;
        // disabled the button, since we cannot use IFibonacciService
        this.button.setEnabled(false);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case PROGRESS_DIALOG_ID:
                // this dialog will be opened in onClick(...) and
                // dismissed/removed by responseHandler.handleMessage(...)
                ProgressDialog dialog = new ProgressDialog(this);
                dialog.setMessage(super.getText(R.string.progress_text));
                dialog.setIndeterminate(true);
                return dialog;
            default:
                return super.onCreateDialog(id);
        }
    }

    // handle button clicks
    public void onClick(View view) {
        // parse n from input (or report errors)
        final long n;
        String s = this.input.getText().toString();
        if (TextUtils.isEmpty(s)) {
            return;
        }
        try {
            n = Long.parseLong(s);
        } catch (NumberFormatException e) {
            this.input.setError(super.getText(R.string.input_error));
            return;
        }

        // build the request object
        int type;
        switch (FibonacciActivity.this.type.getCheckedRadioButtonId()) {
            case R.id.type_fib_jr:
                type = FibonacciRequest.RECURSIVE_JAVA_TYPE;
                break;
            case R.id.type_fib_ji:
                type = FibonacciRequest.ITERATIVE_JAVA_TYPE;
```

```
                break;
            case R.id.type_fib_nr:
                type = FibonacciRequest.RECURSIVE_NATIVE_TYPE;
                break;
            case R.id.type_fib_ni:
                type = FibonacciRequest.ITERATIVE_NATIVE_TYPE;
                break;
            default:
                return;
        }

        FibonacciRequest request = new FibonacciRequest(n, type);
        try {
            Log.d(TAG, "Submitting request...");
            // submit the request; the response will come to responseListener
            this.service.fib(request, this.responseListener);
            Log.d(TAG, "Submited request");
            // this dialog will be dismissed/removed by responseHandler
            super.showDialog(PROGRESS_DIALOG_ID);
        } catch (RemoteException e) {
            Log.wtf(TAG, "Failed to communicate with the service", e);
        }
    }
}
```

11 Lab: Binder-based Service with AIDL

Create an AIDL-described `ILogService` that provides the following functionality:

```
package com.marakana.android.logservice;
public interface ILogService {
    public void log(LogMessage logMessage);
}
```

where `LogMessage` is defined as follows:

```
package com.marakana.android.logservice;
public class LogMessage ... {
    ...
    public LogMessage(int priority, String tag, String message) {
        ...
    }
    ...
}
```

Create a simple Android client that allows the user to submit a `LogMessage` request to the remote `ILogService` running in a separate process.

Tip
Your implementation can simply use `android.util.Log.println(int priority, String tag, String msg)` to do the logging.
