

Introduction to Android Hello World! Application

Çağatay Sönmez

07.12.2018

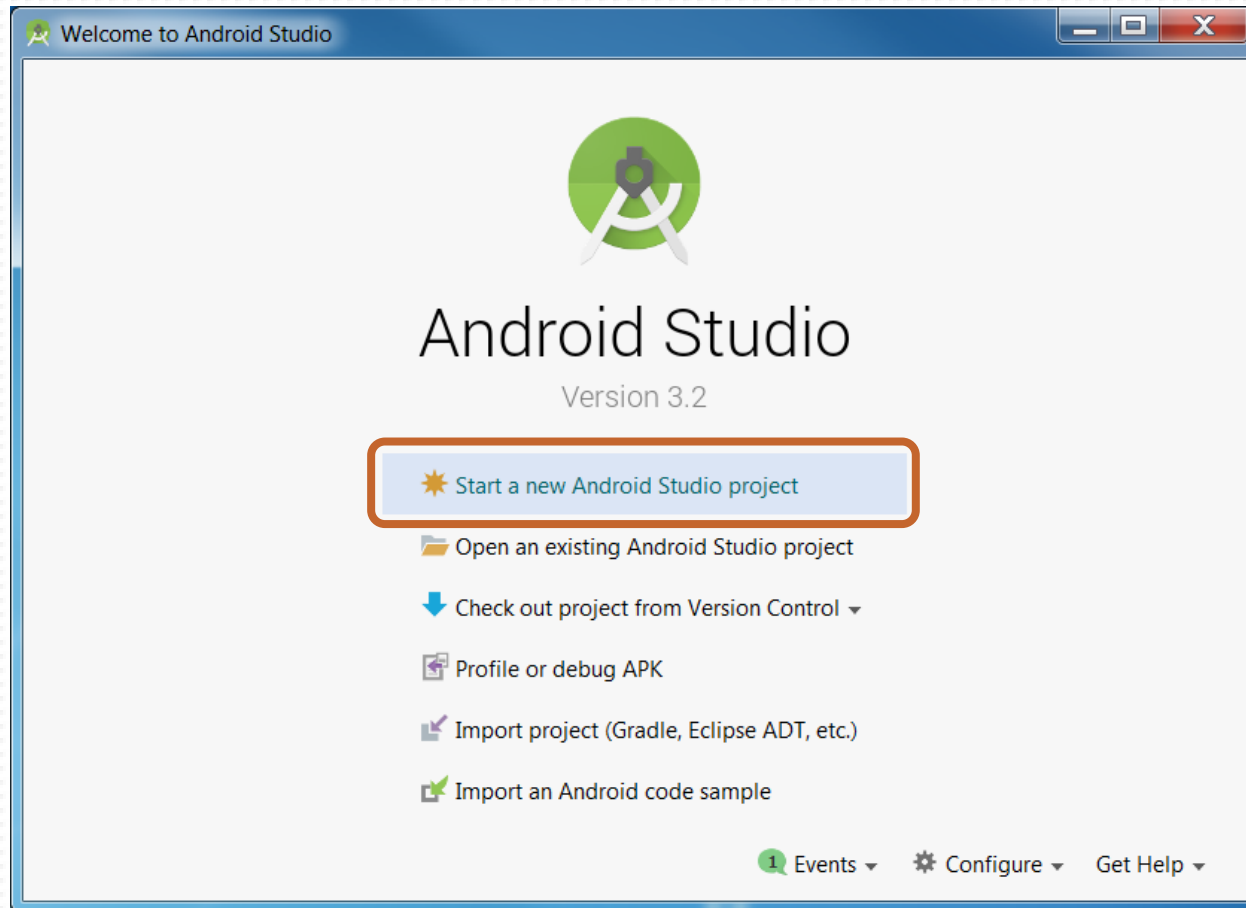
Agenda

- Creating a sample Android TV application
- Adding Activities
- Adding Services
- Adding BroadcastReceivers
- Running the app on the emulator
- Running the app on the device over USB
- Running the app on the device over network
- Profile the app performance
- Debugging the app

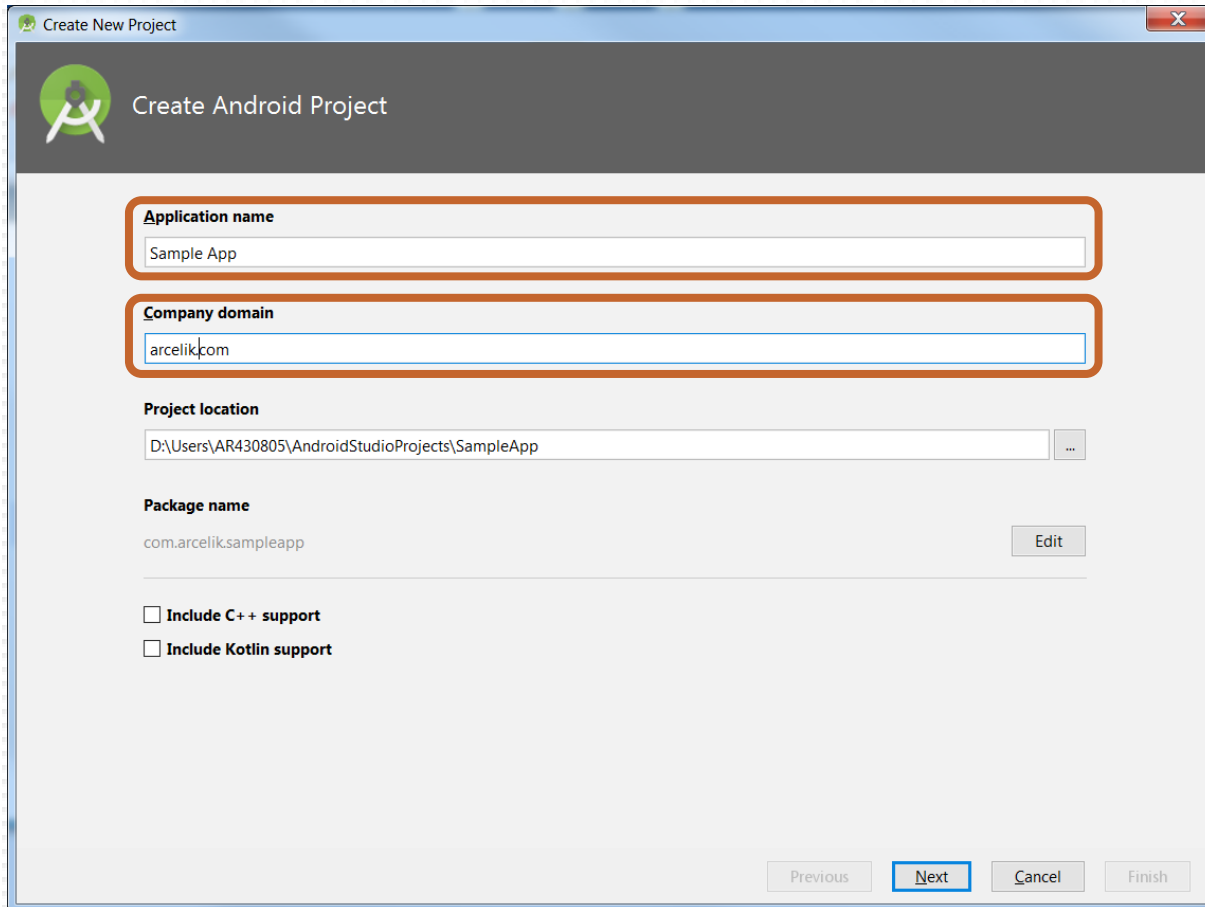
Materials

- Application source code can be found on GitHub
 - <https://github.com/CagataySonmez/Android-for-Beginners/tree/master/3-IntroductionToAndroid-HelloWorldApplication>
- Android Studio version 3.2.1 is used on this training
- Android Studio can be downloaded from the official website
 - <https://developer.android.com/studio/>
- Free courses can be found on Google Developers Training website
 - <https://developers.google.com/training/android/>

Creating Android TV App I



Creating Android TV App II



Create New Project

Create Android Project

Application name
Sample App

Company domain
arcelik.com

Project location
D:\Users\AR430805\AndroidStudioProjects\SampleApp

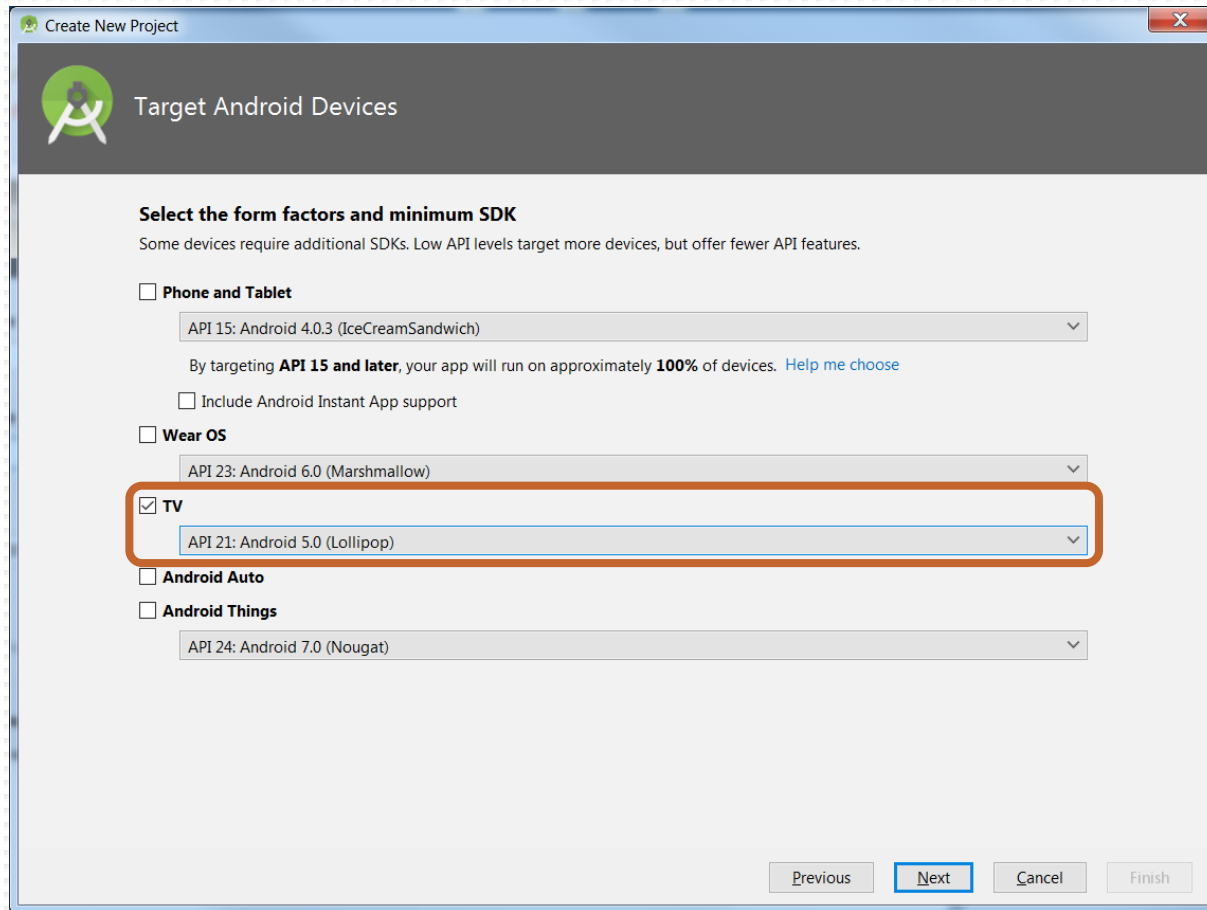
Package name
com.arcelik.sampleapp Edit

☐ Include C++ support


☐ Include Kotlin support

Previous Next Cancel Finish

Creating Android TV App III



Create New Project

 **Target Android Devices**

Select the form factors and minimum SDK
Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☐ **Phone and Tablet**
API 15: Android 4.0.3 (IceCreamSandwich) ▼
By targeting **API 15 and later**, your app will run on approximately **100%** of devices. [Help me choose](#)

☐ Include Android Instant App support

☐ **Wear OS**
API 23: Android 6.0 (Marshmallow) ▼

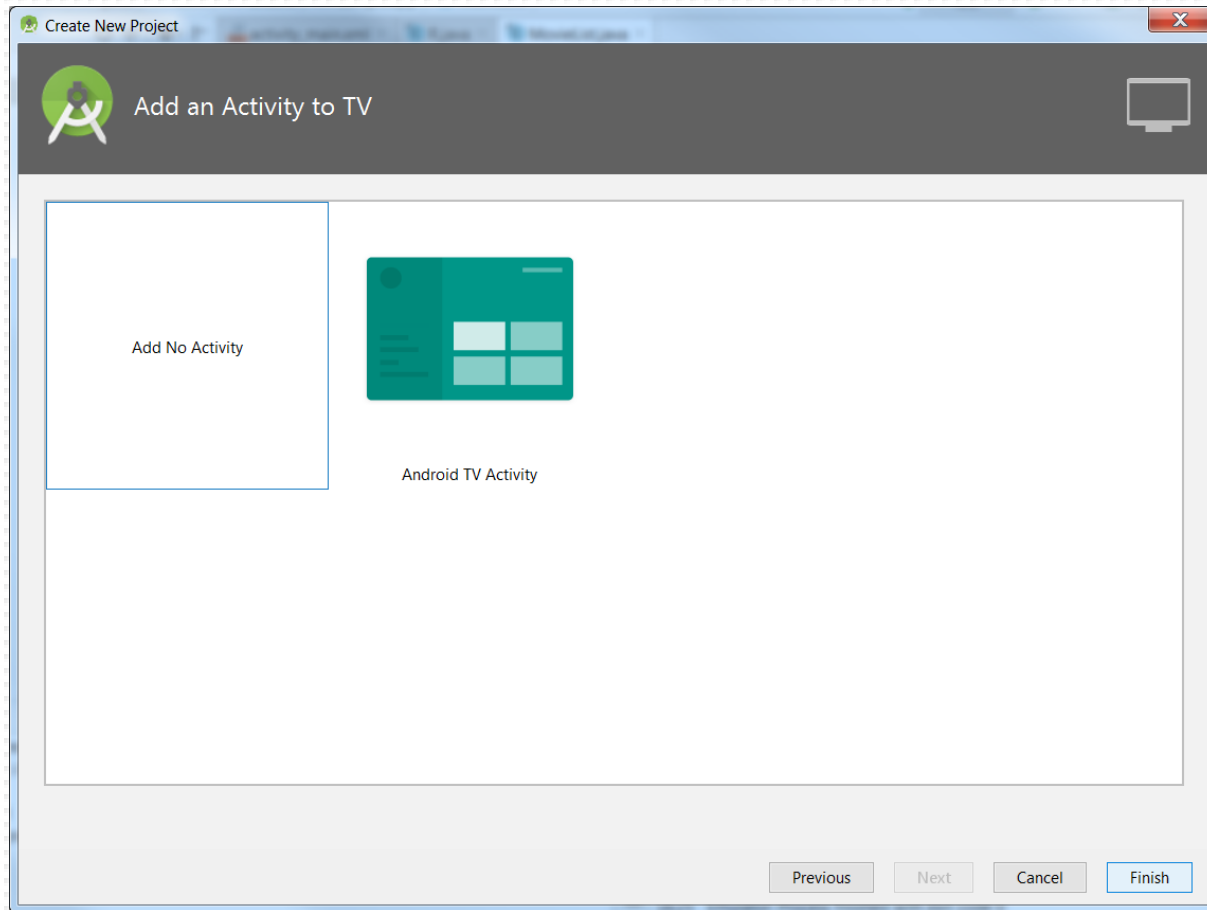
☒ **TV**
API 21: Android 5.0 (Lollipop) ▼

☐ **Android Auto**

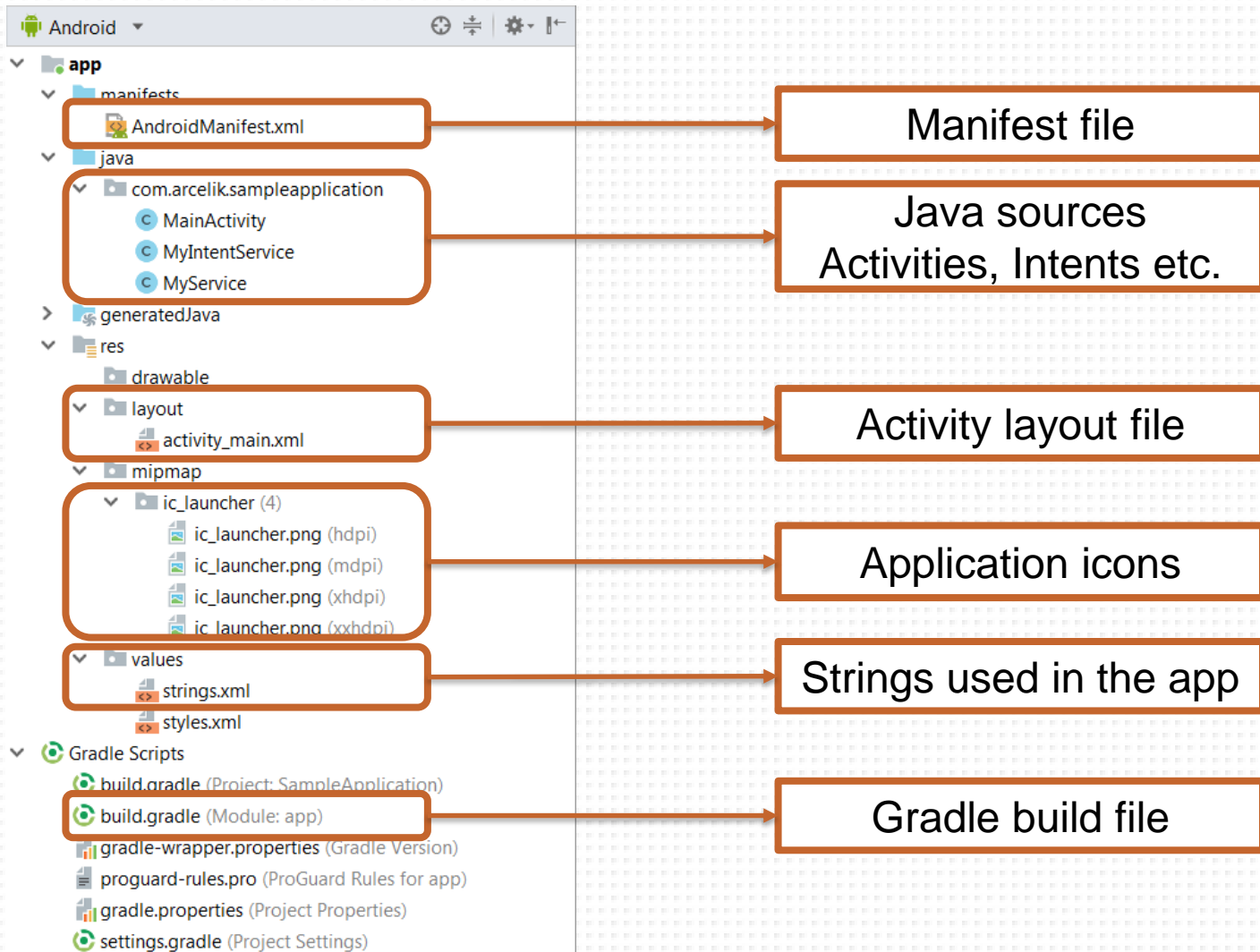
☐ **Android Things**
API 24: Android 7.0 (Nougat) ▼

[Previous](#) [Next](#) [Cancel](#) [Finish](#)

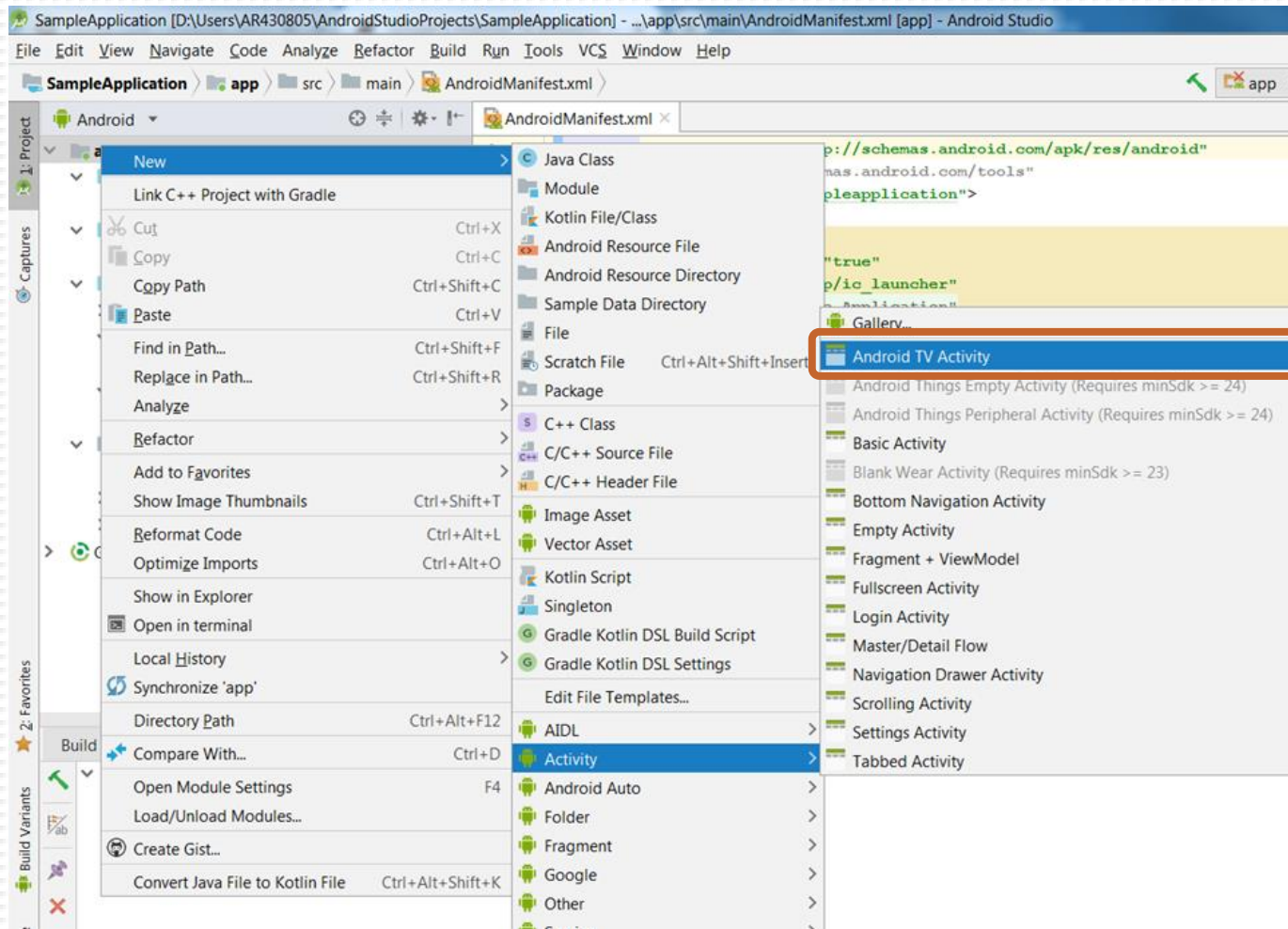
Creating Android TV App IV



Creating Android TV App V





Adding Activity I



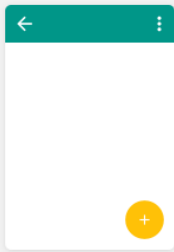
Adding Activity II

New Android Activity

 **Configure Activity**
Android Studio



Creates a new basic activity with an app bar.



Activity Name:

Layout Name:

Title:

☐ Launcher Activity

☐ Use a Fragment

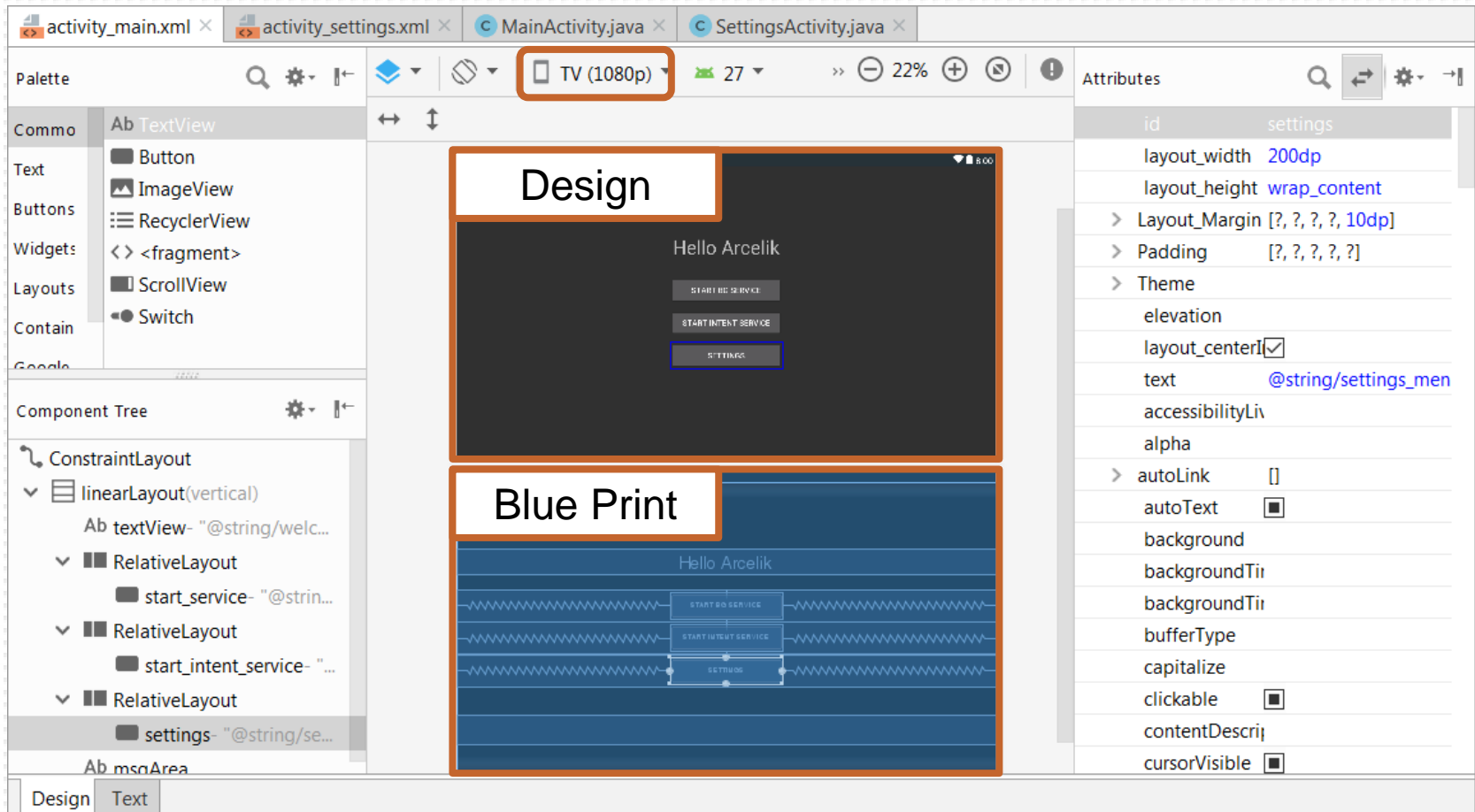
Hierarchical Parent: ...

Package name:

Source Language:

The name of the activity class to create

Activity Layout File



Strings.txt file I

- Do not embed texts to layout file!

AndroidManifest.xml x strings.xml x activity_main.xml x MainActivity.java x

Edit translations for all locales in the translations editor.

```
1 <resources>
2   <string name="app_name">Sample Application</string>
3   <string name="title_activity_main">MainActivity</string>
4   <string name="welcome_text">Hello Arcelik</string>
5   <string name="start_service_button">Start BG Service</string>
6   <string name="stop_service_button">Stop BG Service</string>
7   <string name="start_
8 </resources>
```

activity_main.xml x activity_settings.xml x

Attributes

- layout_height wrap_content
- > Layout_Margin [?, ?, 120dp, ?, 25dp]
- > Padding [?, ?, ?, ?]
- > Theme
- elevation
- text @string/welcome_text
- textAlignment center
- textSize @dimen/lb_details_des
- accessibilityLevel
- alpha
- > autoLink []

Strings.txt file II

- Do not use static texts in your source code!



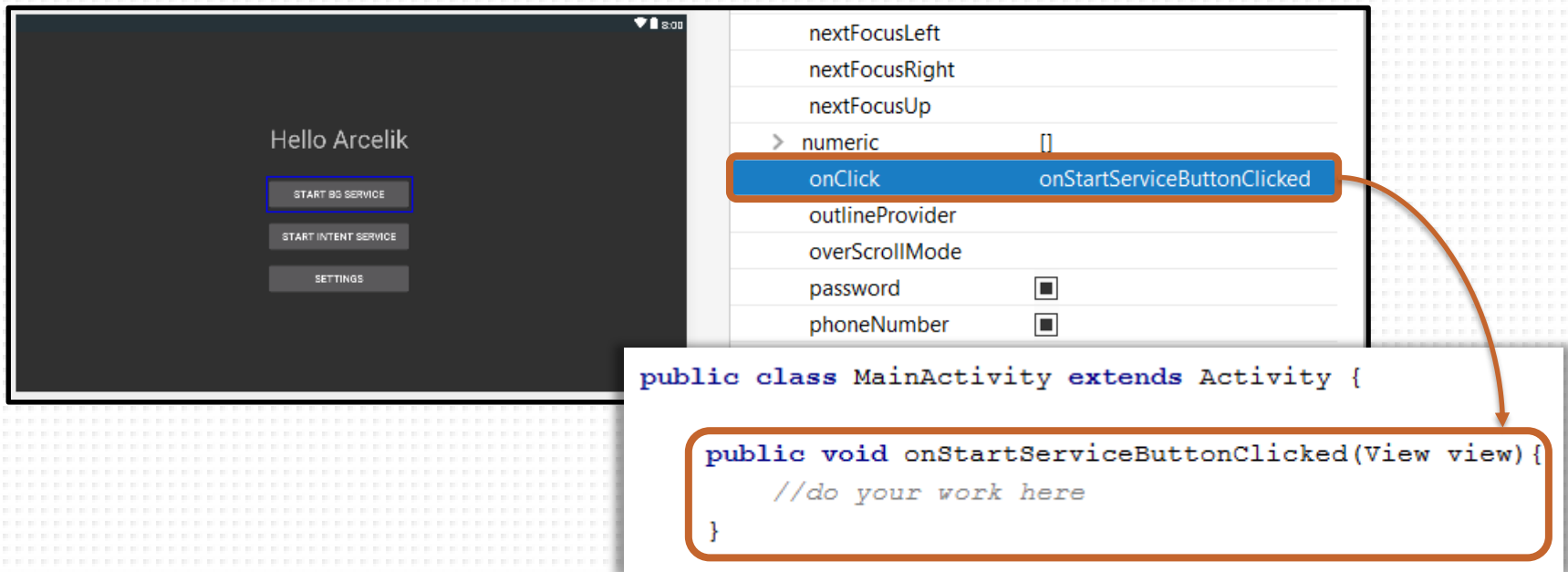
```
if (isMyServiceRunning (MyService.class))  
    ((Button) findViewById(R.id.start_service)).setText("Stop BG Service");
```



```
if (isMyServiceRunning (MyService.class))  
    ((Button) findViewById(R.id.start_service)).setText(R.string.stop_service_button);
```

Handling Click Events of Button I

- There are many options to handle click events of button
- Option 1: Using onClick property in xml layout file
- The callback function's signature cannot be changed!



The image shows a screenshot of an Android application interface on the left and its XML layout on the right. The interface displays "Hello Arcelik" and three buttons: "START BG SERVICE", "START INTENT SERVICE", and "SETTINGS". The XML layout on the right lists various properties for a button, with the `onClick` property highlighted in blue and its value `onStartServiceButtonClicked` shown. An orange arrow points from this value to a code snippet box below. The code snippet shows the implementation of the `onStartServiceButtonClicked` method in `MainActivity`.

```
public class MainActivity extends Activity {  
  
    public void onStartServiceButtonClicked(View view) {  
        //do your work here  
    }  
}
```

Handling Click Events of Button II

- Option 2: Using an View.OnClickListener via an **anonymous inner class**
- The system executes the code in onClick on the main thread!

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

    // set onclick listener of start_service button
    Button startService = (Button)findViewById(R.id.start_service);
    startService.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //Do your work here
        }
    });
}
```

Handling Click Events of Button III

- Option 3: Using a View.OnClickListener via an anonymous inner class which can be reusable

```
private View.OnClickListener startServiceOnClickListener = new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        //Do your work here  
    }  
};  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // set onclick listener of start_service button  
    Button startService = (Button)findViewById(R.id.start_service);  
    startService.setOnClickListener(startServiceOnClickListener);  
}
```


Handling Click Events of Button IV

- Option 4: Using your own class by implementing View.OnClickListener Interface

```
class StartServiceButtonClick implements View.OnClickListener {  
    @Override  
    public void onClick(View v) {  
        //Do your work here  
    }  
}  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // set onclick listener of start_service button  
    Button startService = (Button)findViewById(R.id.start_service);  
    startService.setOnClickListener(new StartServiceButtonClick());  
}
```

Starting Another (Settings) Activity

1. Add a button click listener
2. Create an explicit Intent when the button is clicked
3. Start Activity via created Intent

```
// Handle onClickListener of settings button.  
Button settings = (Button)findViewById(R.id.settings);  
settings.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Intent intent = new Intent( packageContext: MainActivity.this, SettingsActivity.class);  
        startActivity(intent);  
    }  
});
```

Read/Write Application Settings

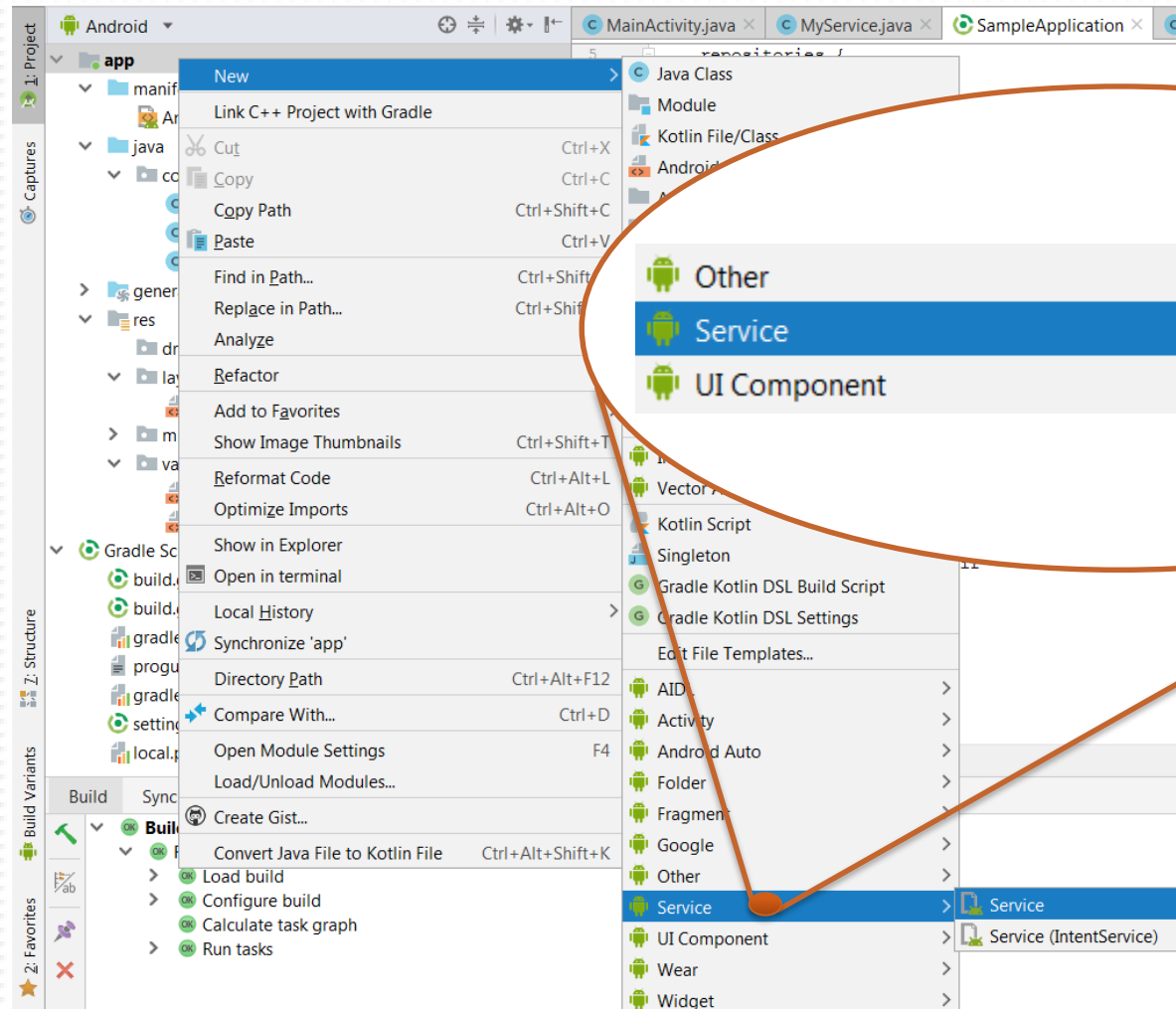
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_settings);

    //get last saved value from preferences
    SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
    String lastSavedDate = sharedPref.getString(S: "last_saved_date", s1: "unsaved!");

    //Use last saved date

    // Handle onClickListener of save button.
    Button save = (Button)findViewById(R.id.save);
    save.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String lastSavedDate = getShortDate();
            SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
            SharedPreferences.Editor editor = sharedPref.edit();
            editor.putString(S: "last_saved_date", lastSavedDate);
            editor.commit();
        }
    });
}
```

Adding Service

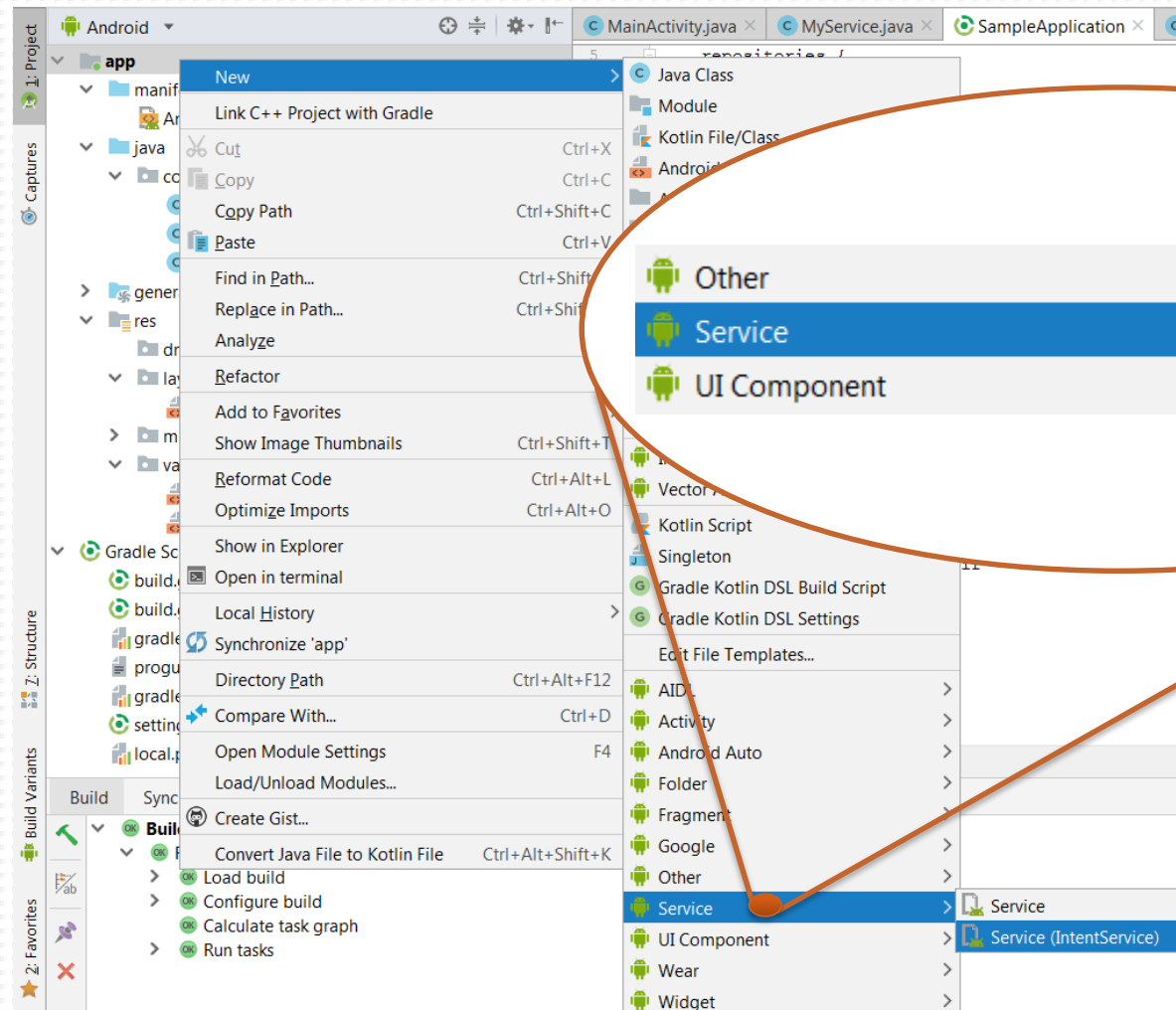


Toogle the Service

1. Add a button click listener
2. Create an explicit Intent when the button is clicked
3. Start or stop the Service via created Intent

```
// Click this button to start service.
Button startService = (Button)findViewById(R.id.start_service);
startService.setOnClickListener((v) -> {
    Intent intent = new Intent( packageContext: MainActivity.this, MyService.class);
    if(isMyServiceRunning(MyService.class)) {
        stopService(intent);
        ((Button)findViewById(R.id.start_service)).setText("Start BG Service");
    }
    else {
        startService(intent);
        ((Button)findViewById(R.id.start_service)).setText("Stop BG Service");
    }
});
```

Adding Intent Service




Starting Intent Service

1. Add a button click listener
2. Create an explicit Intent when the button is clicked
3. Set Intent action and extra data
4. Start the Intent Service via created Intent

```
// Click this button to start intent service.
Button startIntentService = (Button)findViewById(R.id.start_intent_service);
startIntentService.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this, MyIntentService.class);
        intent.setAction(MyIntentService.ACTION_SEND_NOTIFICATION);
        intent.putExtra(MyIntentService.NOTIFICATION_METHOD, value: "TOAST");
        startService(intent);
    }
});
```

Adding Broadcast Receiver I

New Android Component

 **Configure Component**
Android Studio

Creates a new broadcast receiver component and adds it to your Android manifest.

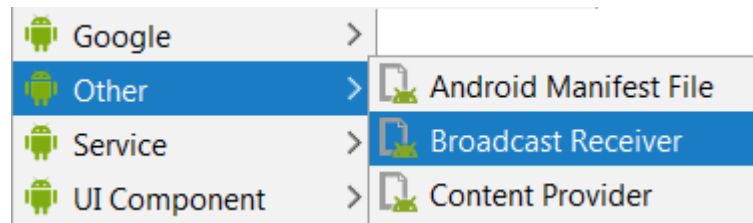
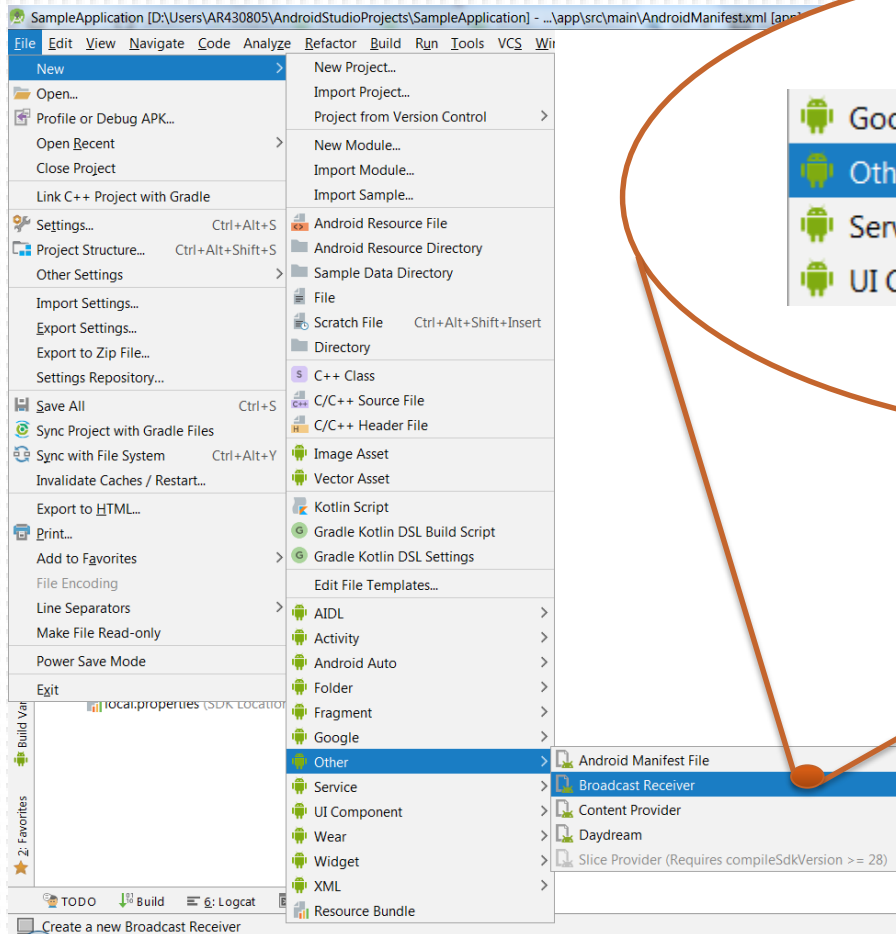
Class Name:

☒ Exported

☒ Enabled

Source Language: ▼

Adding Broadcast Receiver II



Sending Broadcast

1. Declare related permission in manifest file

```
<permission  
    android:name="com.arcelik.sampleapplication.permission.NOTIFICATION"  
    android:protectionLevel="dangerous">  
</permission>  
  
<uses-permission android:name="com.arcelik.sampleapplication.permission.NOTIFICATION" />
```

2. Send broadcast message in your application

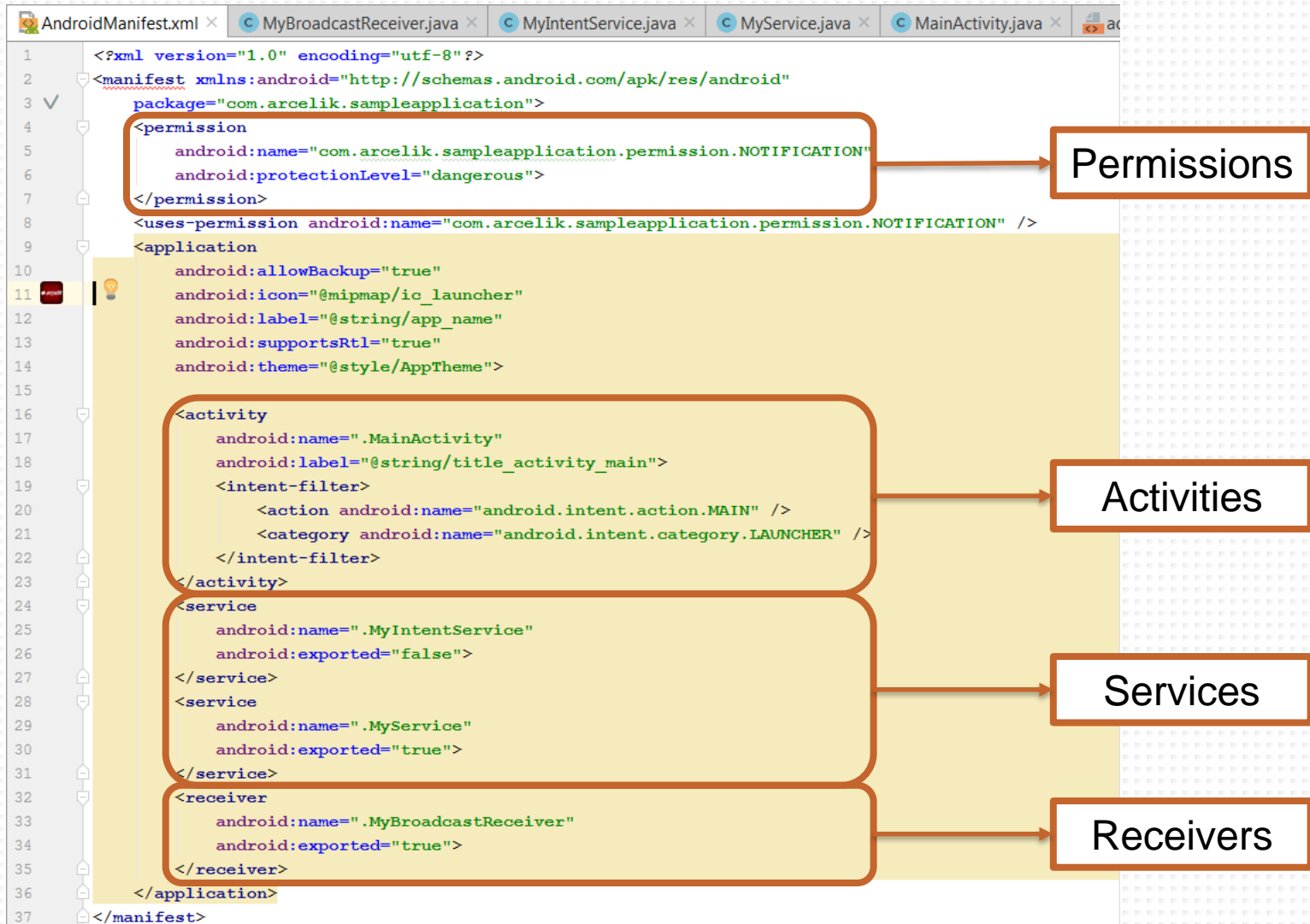
```
//Send broadcast  
Log.d(TAG_SERVICE, msg: "Sending message to activity: " + notification);  
final Intent intent = new Intent(BROADCAST_INTENT);  
intent.putExtra( name: "msg", value: dateAsStr + ":" + notification);  
sendBroadcast(intent, BROADCAST_PERMISSION);
```

Receiving Broadcast

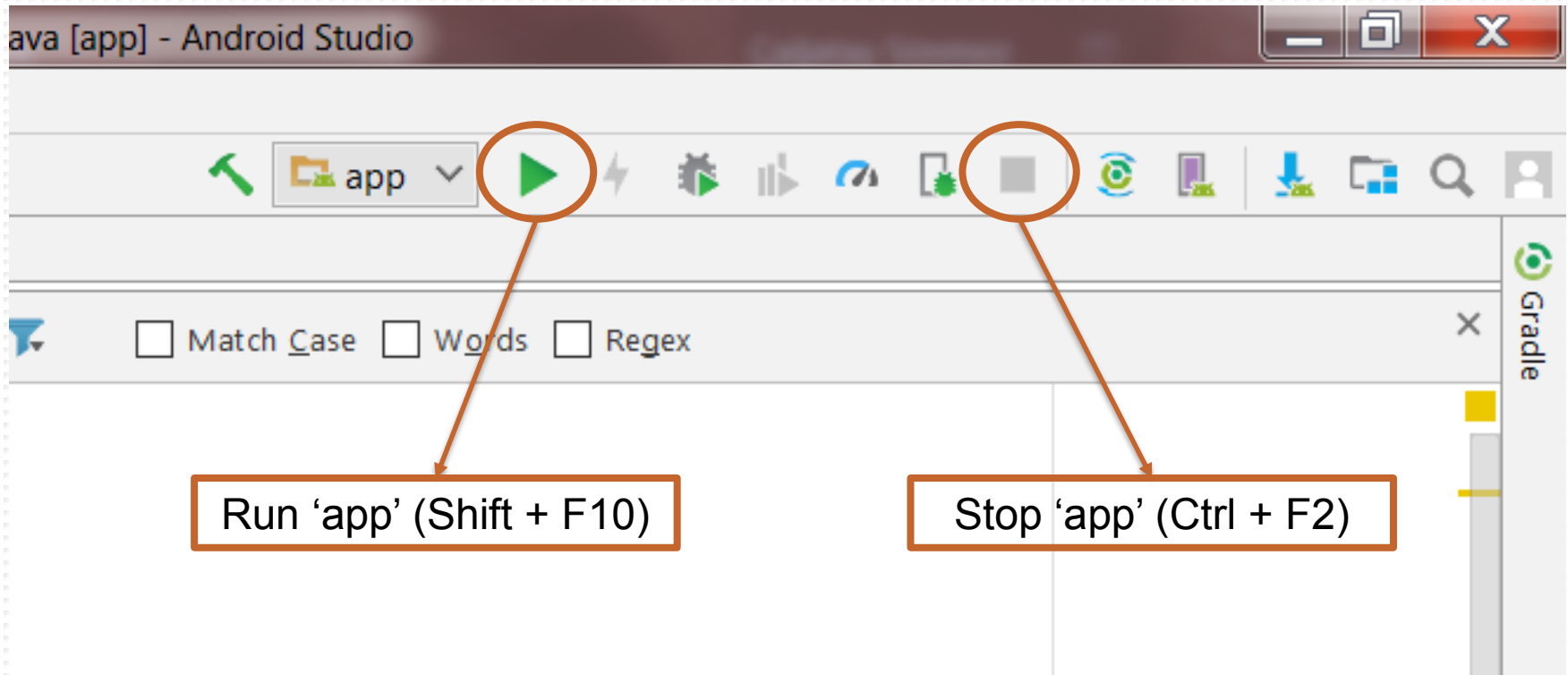
1. Create an IntentFilter to catch proper broadcast event
2. Create BroadcastReceiver and override onReceive method
3. Do your job with Intent provided by the receiver

```
public class MainActivity extends Activity {  
  
    //Use broadcast receiver to get broadcast messages  
    final IntentFilter myFilter = new IntentFilter(MyService.BROADCAST_INTENT);  
    private MyBroadcastReceiver mReceiver = new MyBroadcastReceiver() {  
        @Override  
        public void onReceive(Context context, Intent intent) {  
            final TextView responseFromService = (TextView)findViewById(R.id.msgArea);  
            responseFromService.setText(intent.getCharSequenceExtra( name: "msg" ));  
        }  
    };  
};
```

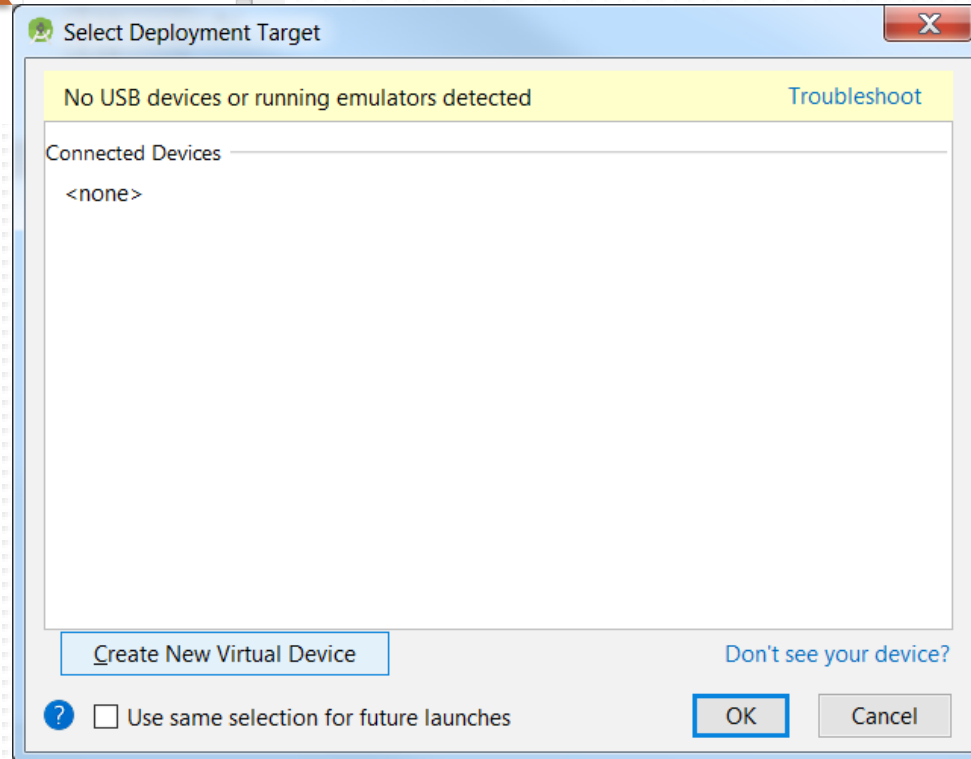
Manifest File



Running an Android Application I

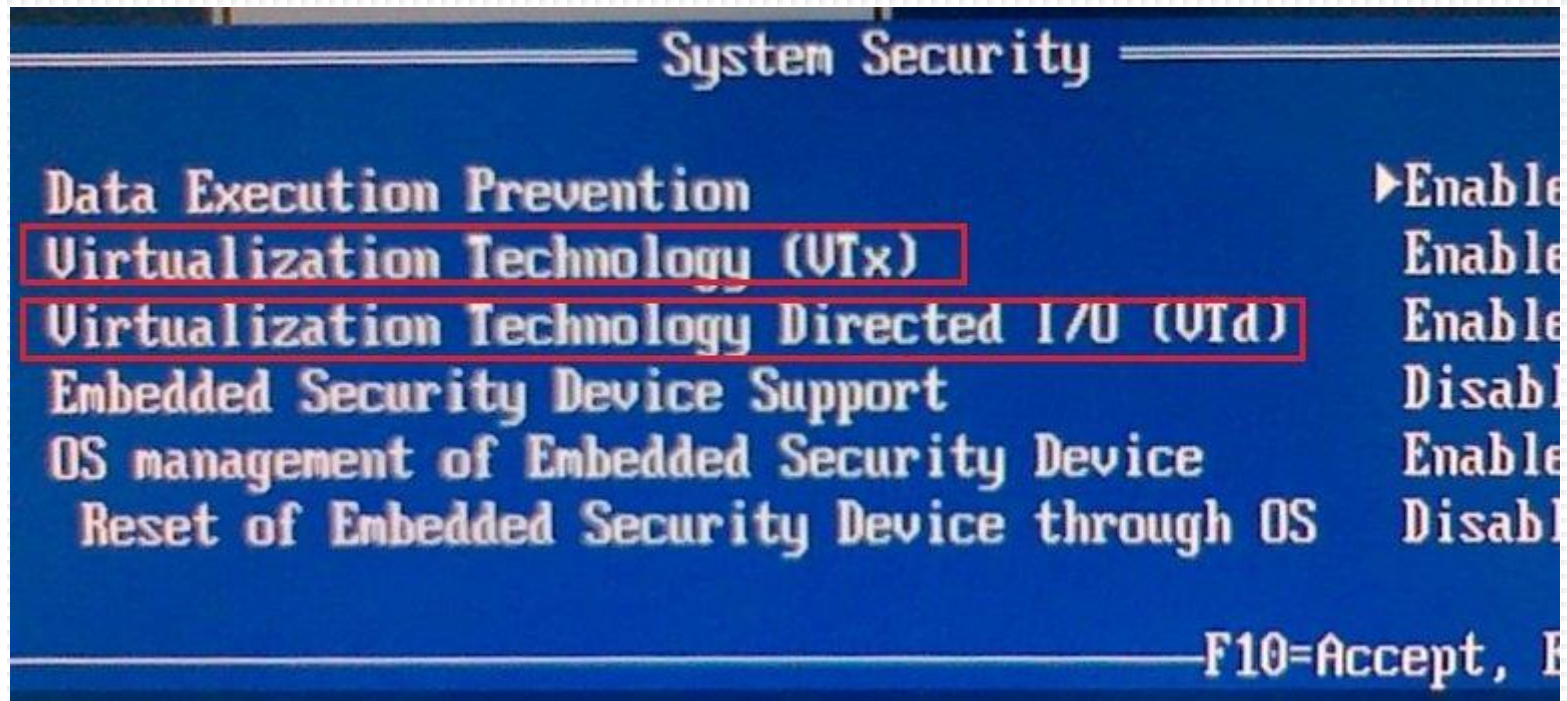


Running an Android Application II

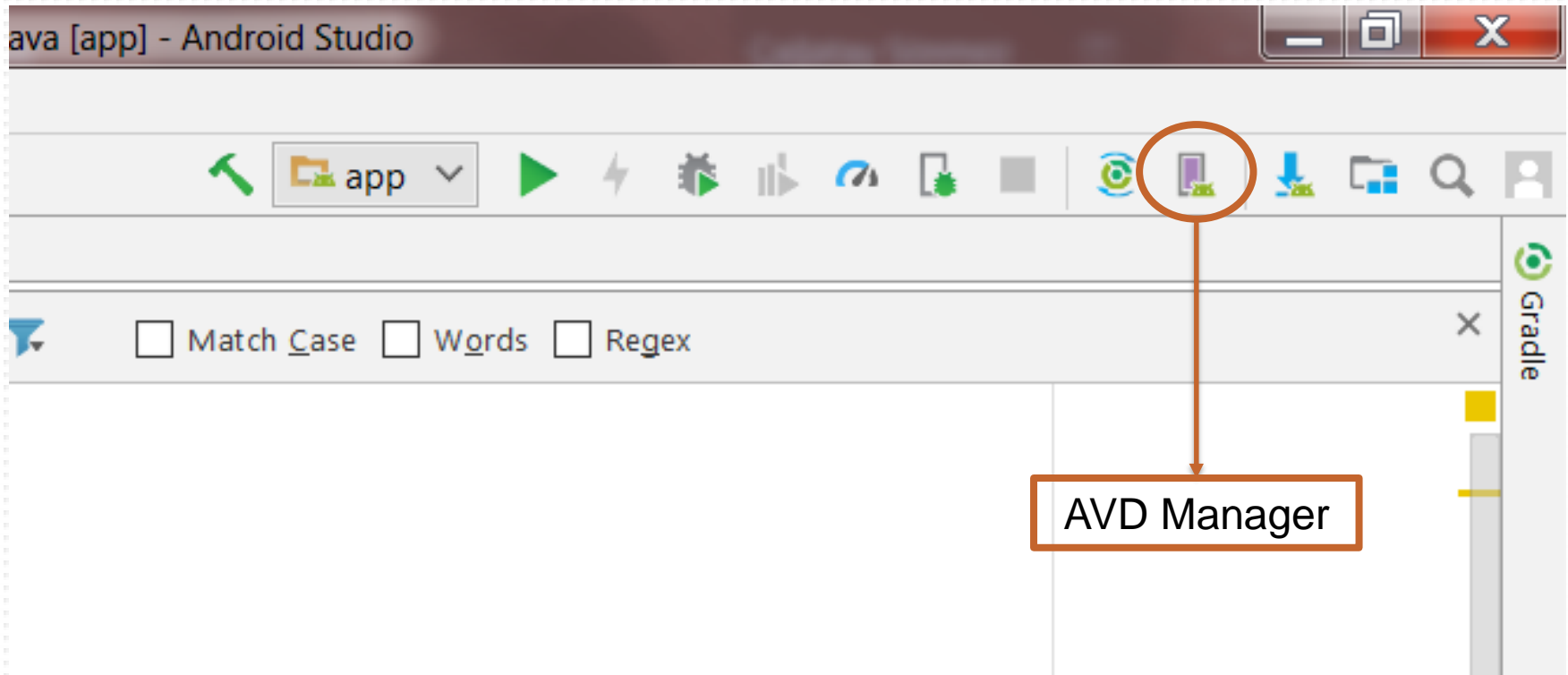


Running App on Emulator

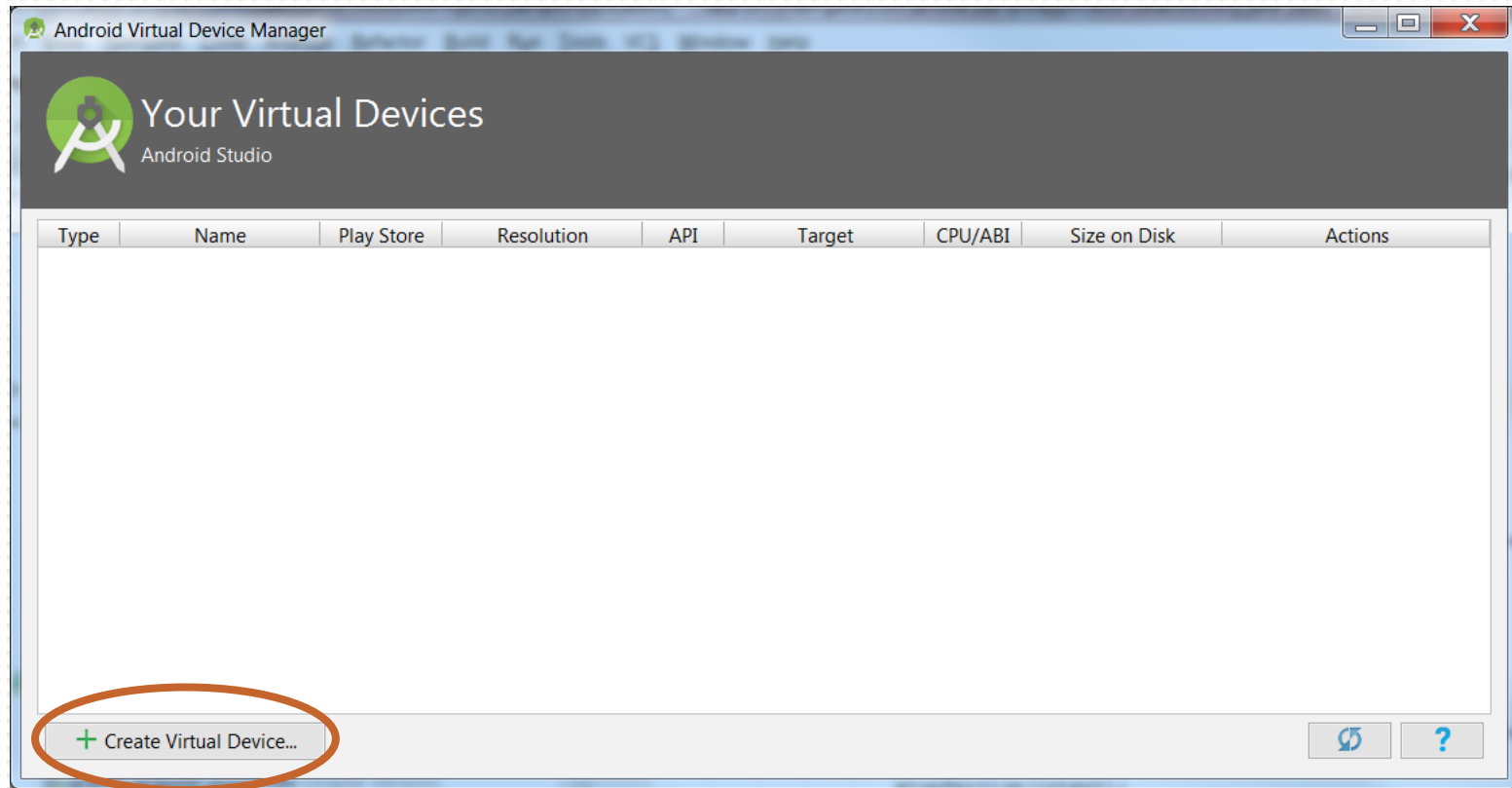
- To use emulator, enable Intel Virtualization Technology or AMD-V depending on the brand of the processor



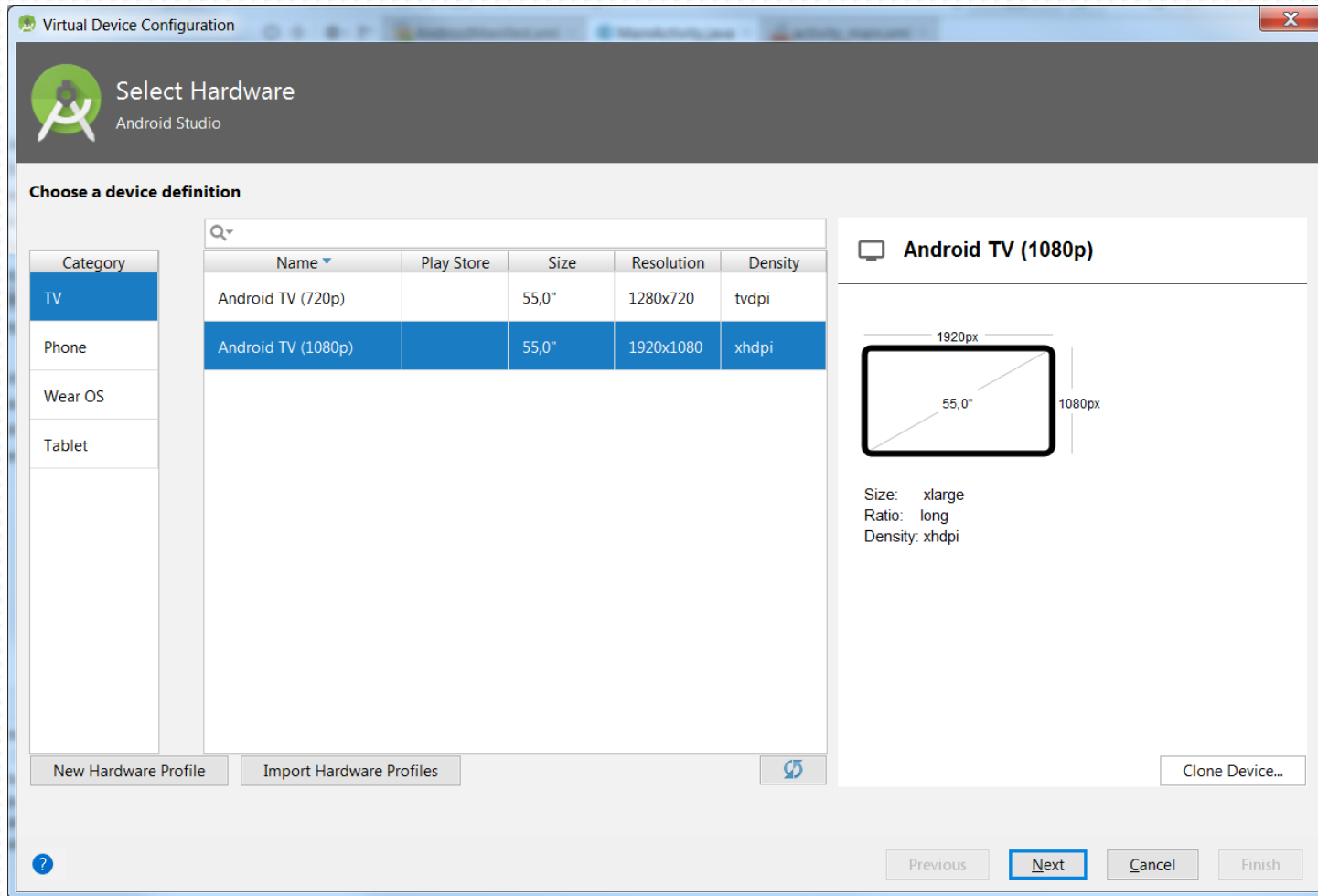
Creating Virtual Device I



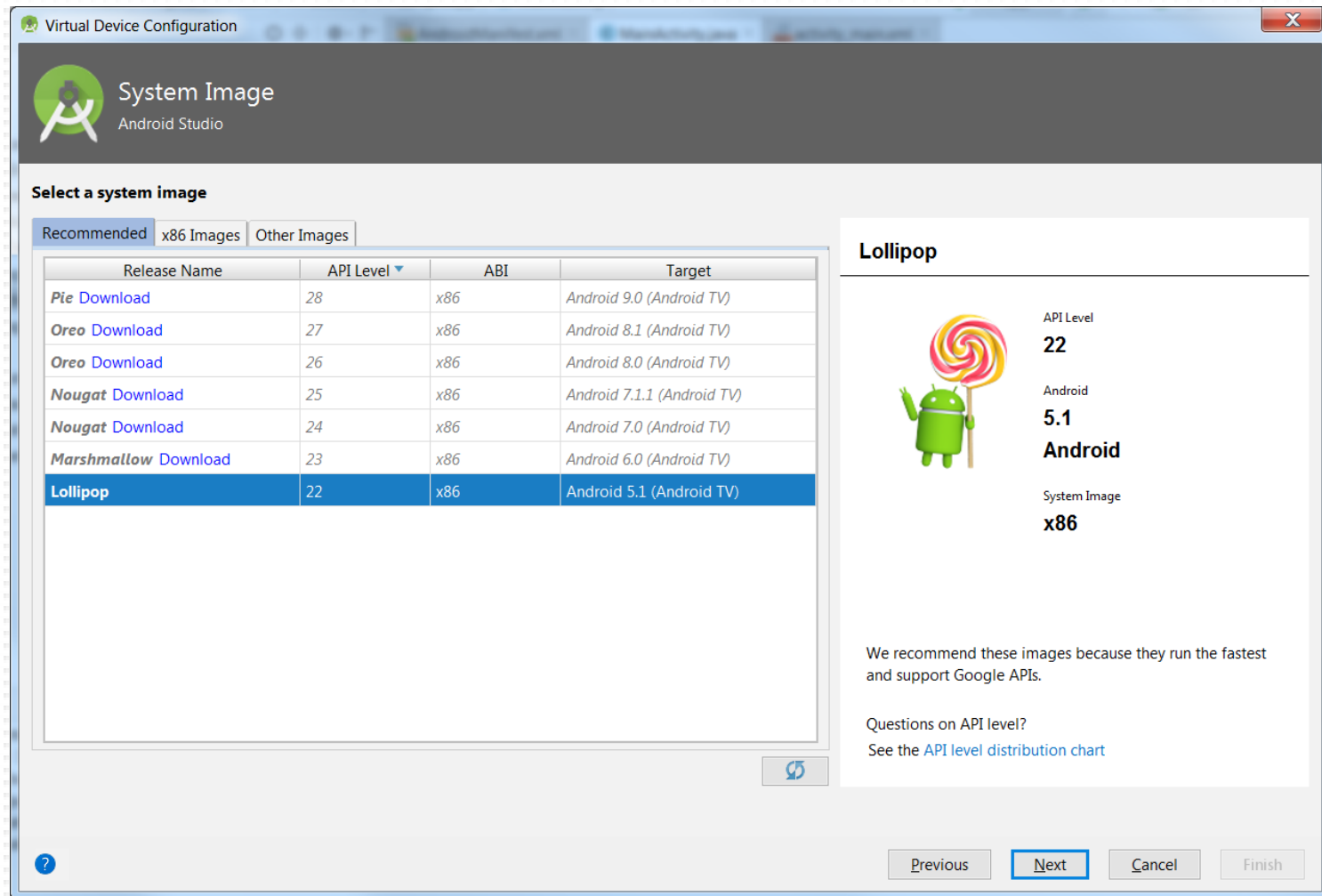
Creating Virtual Device II




Creating Virtual Device III



Creating Virtual Device IV





Creating Virtual Device V




Android Virtual Device (AVD)
Android Studio


Verify Configuration

AVD Name

 Android TV (1080p)	55.0 1920x1080 xhdpi	Change...
 Lollipop	Android 5.1 x86	Change...

Startup orientation


Portrait


Landscape

Emulated Performance


Graphics:

Device Frame ☒ Enable Device Frame

[Show Advanced Settings](#)

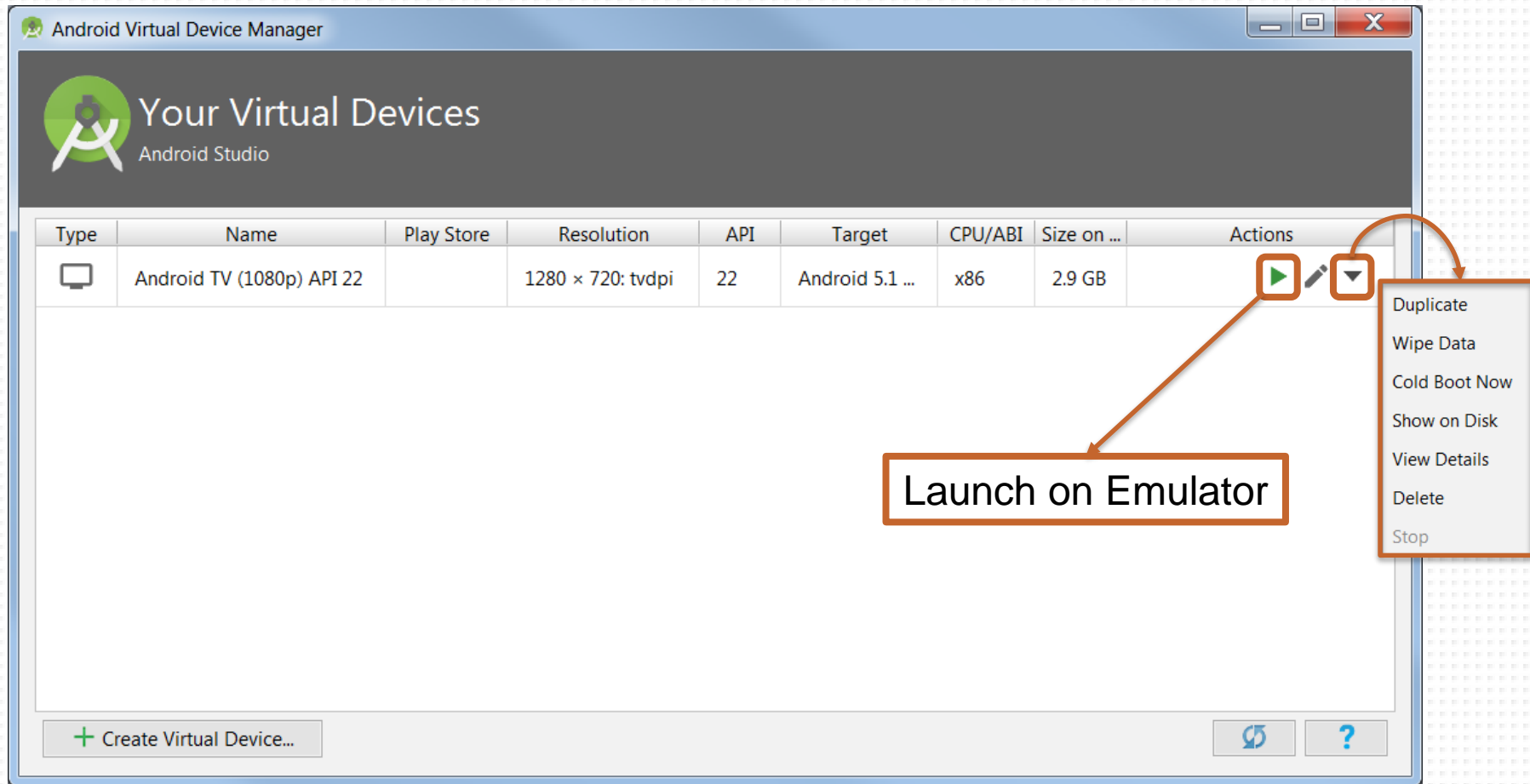
AVD Name

The name of this AVD.

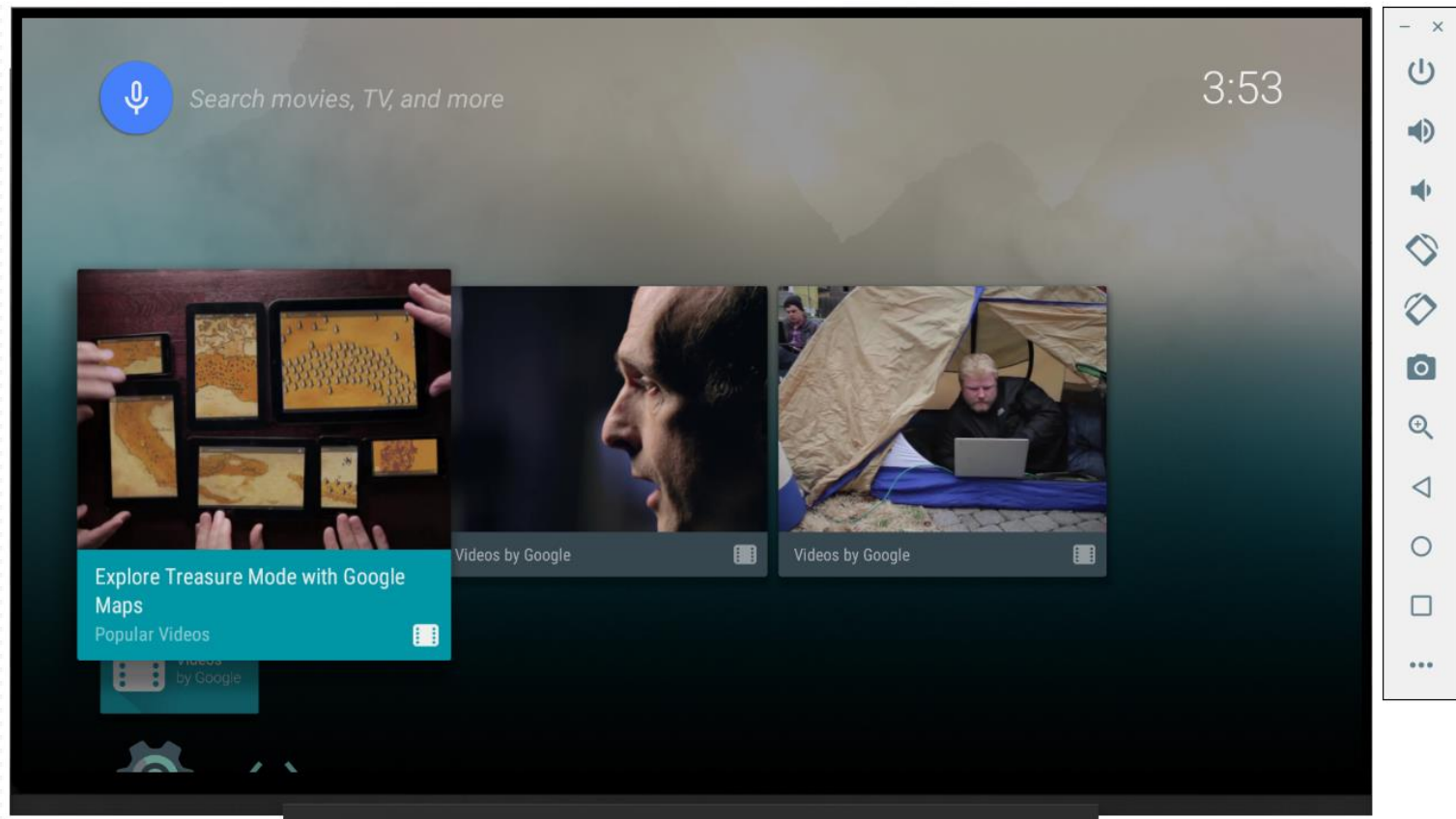


[Previous](#) [Next](#) [Cancel](#) [Finish](#)

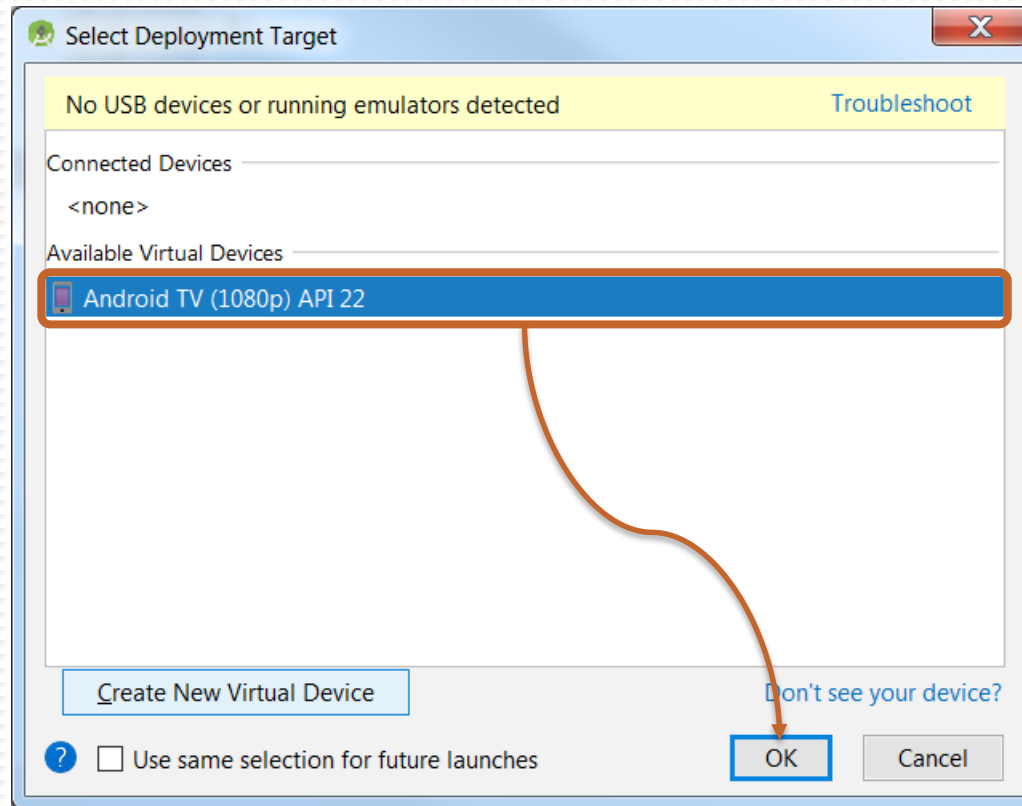
Launching Virtual Device on Emulator I



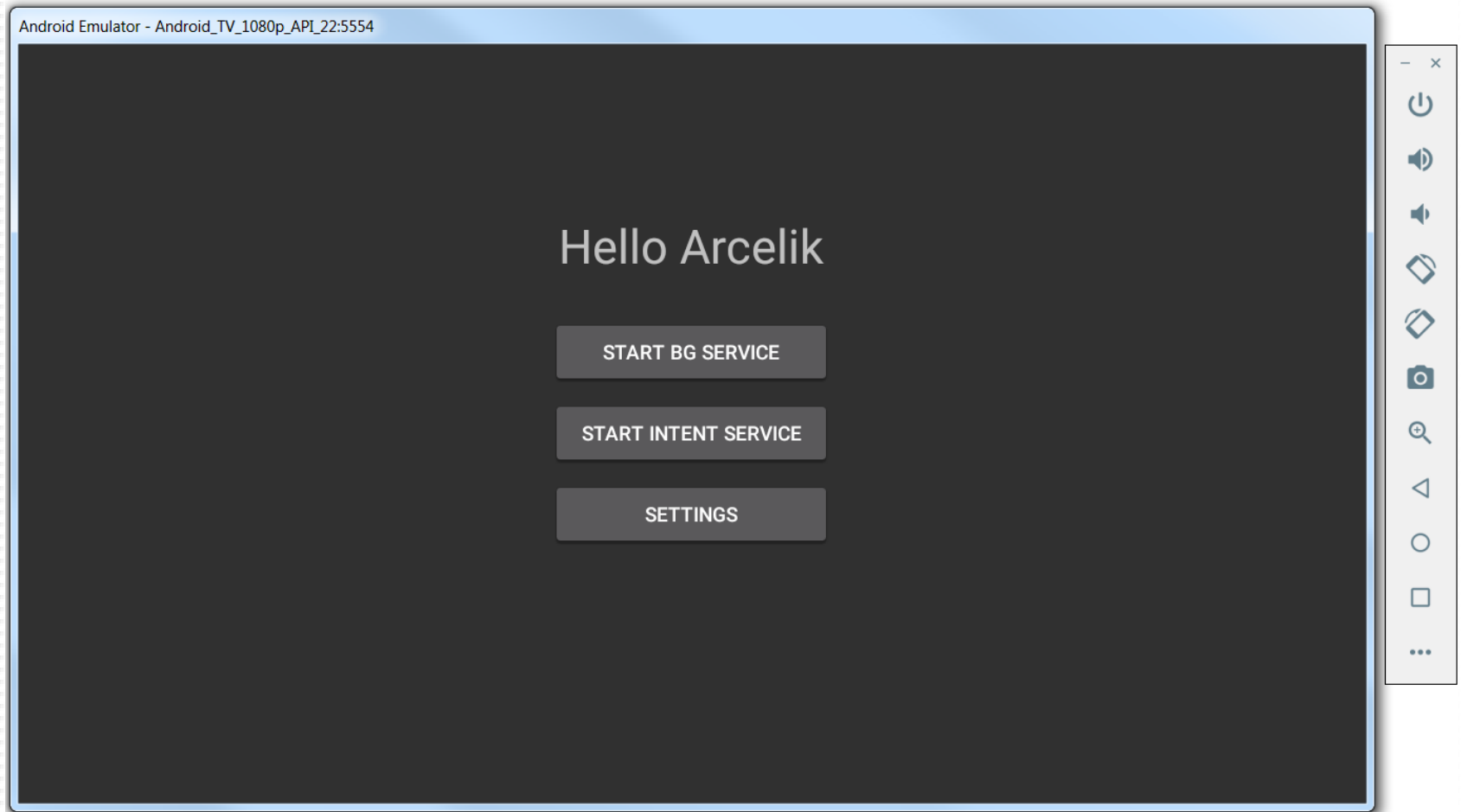
Launching Virtual Device on Emulator II



Running App on Emulator I



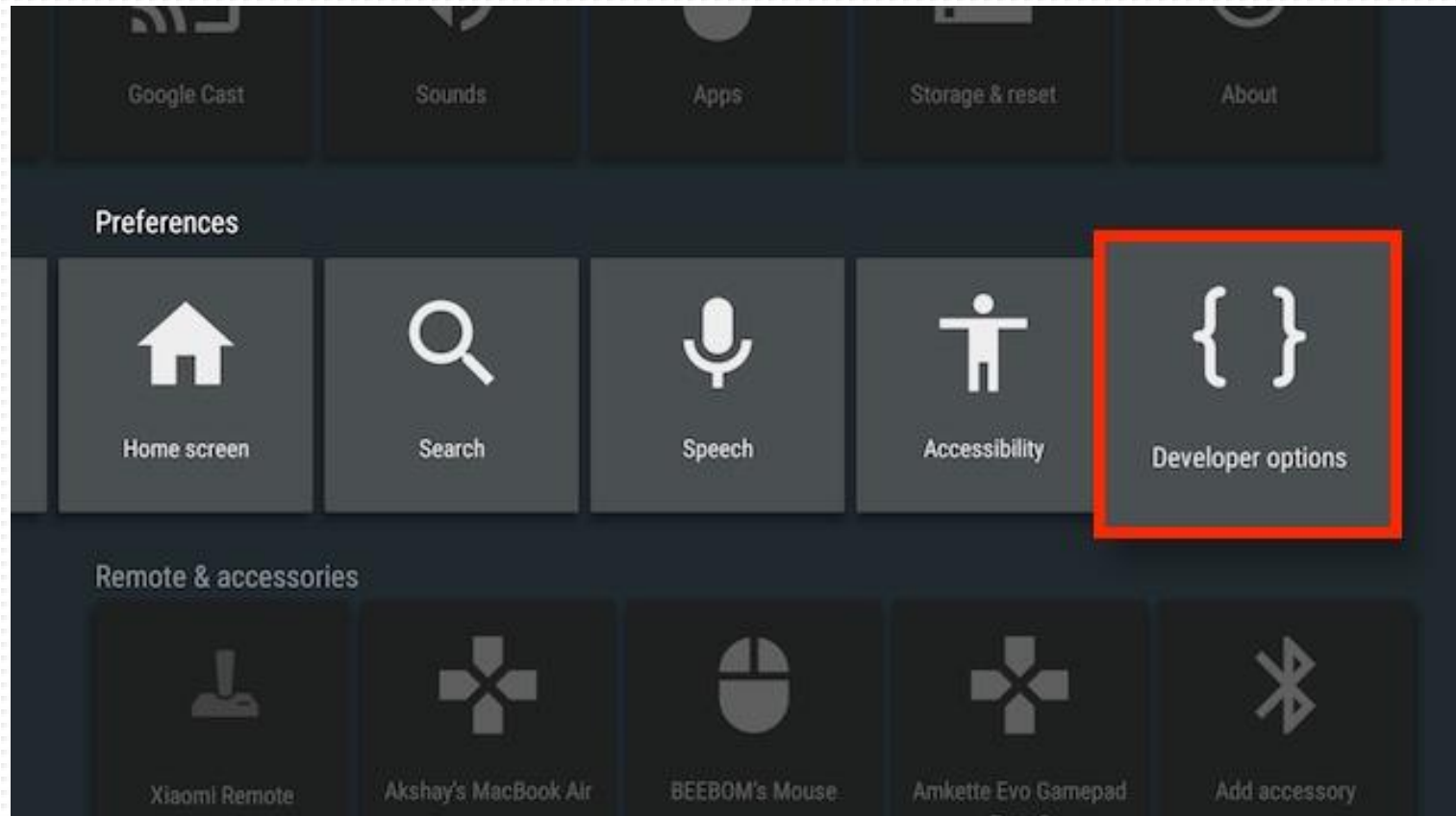
Running App on Emulator II



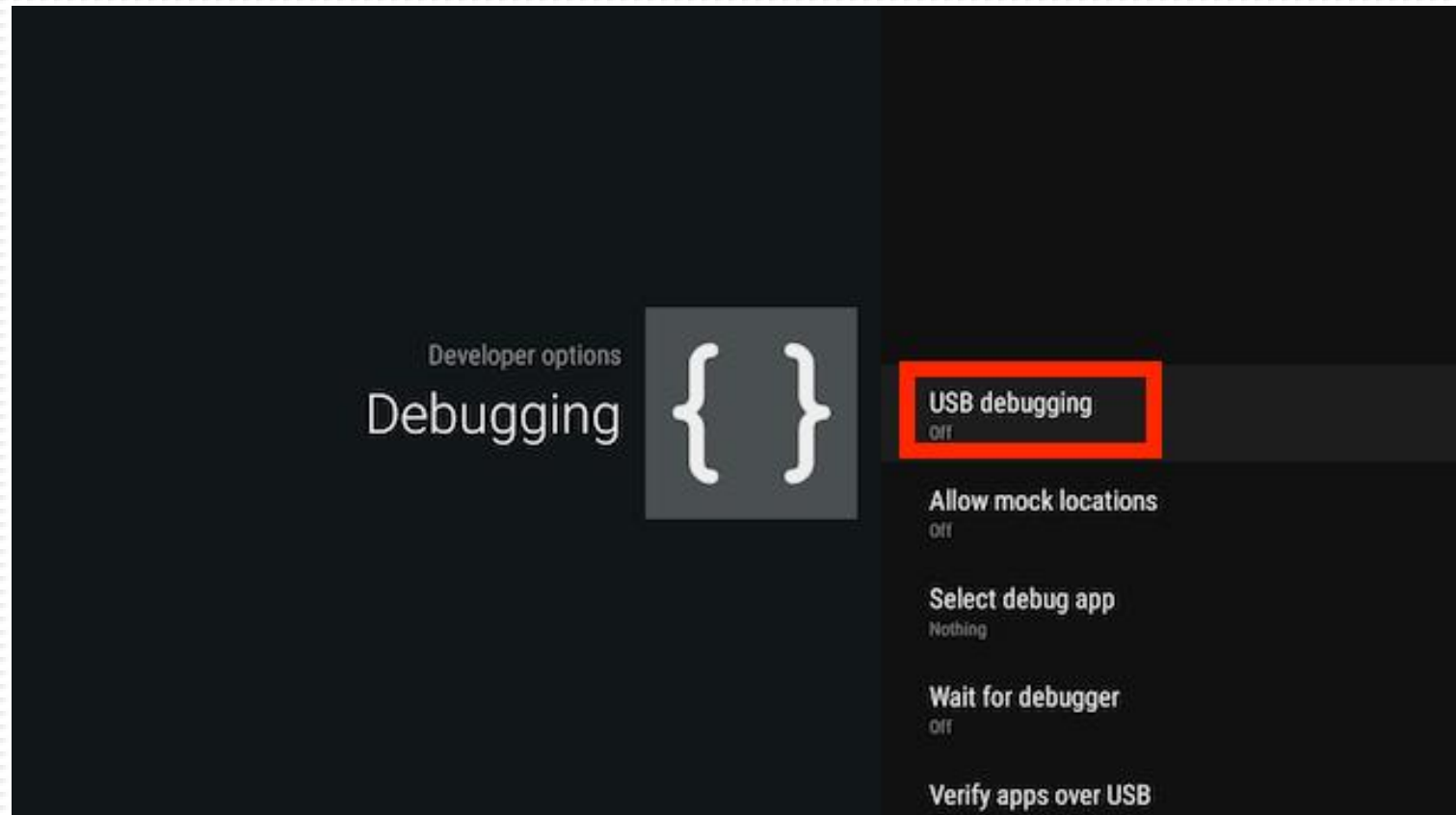
Running App on Device

- Android Debug Bridge (adb) tool is used to communicate with a device
- adb can be used with
 - Network
 - USB
- To use adb with a device connected over USB, you must enable **USB debugging** in the device system settings
- To use adb with a device connected over network, you must enable **ADB debugging** in the device system settings

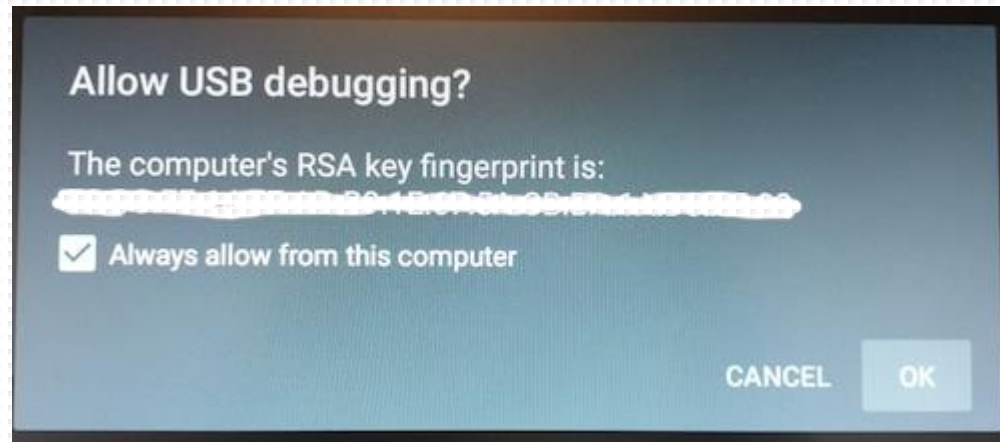
Connect Device (Android TV) over USB I



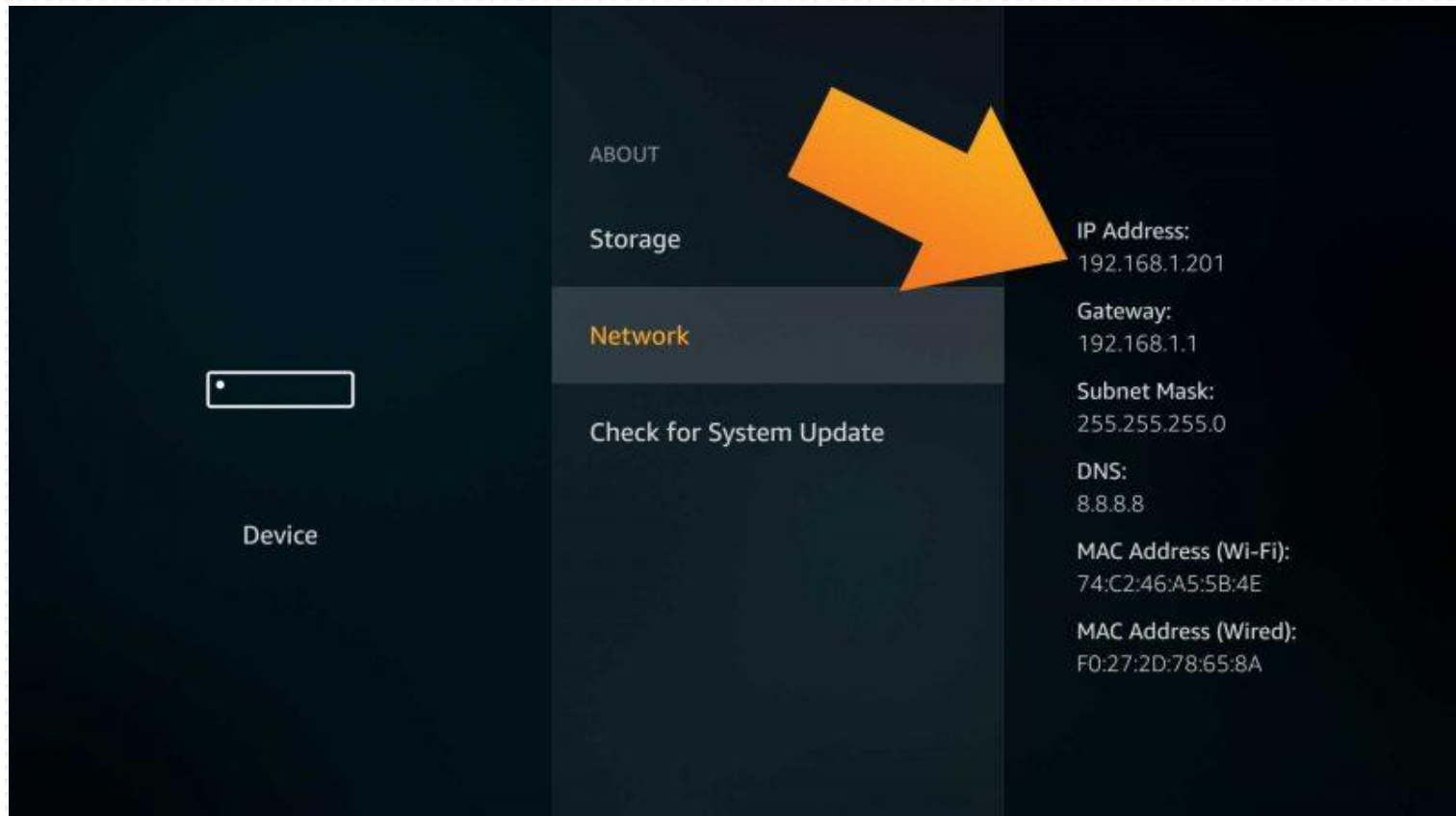
Connect Device (Android TV) over USB II



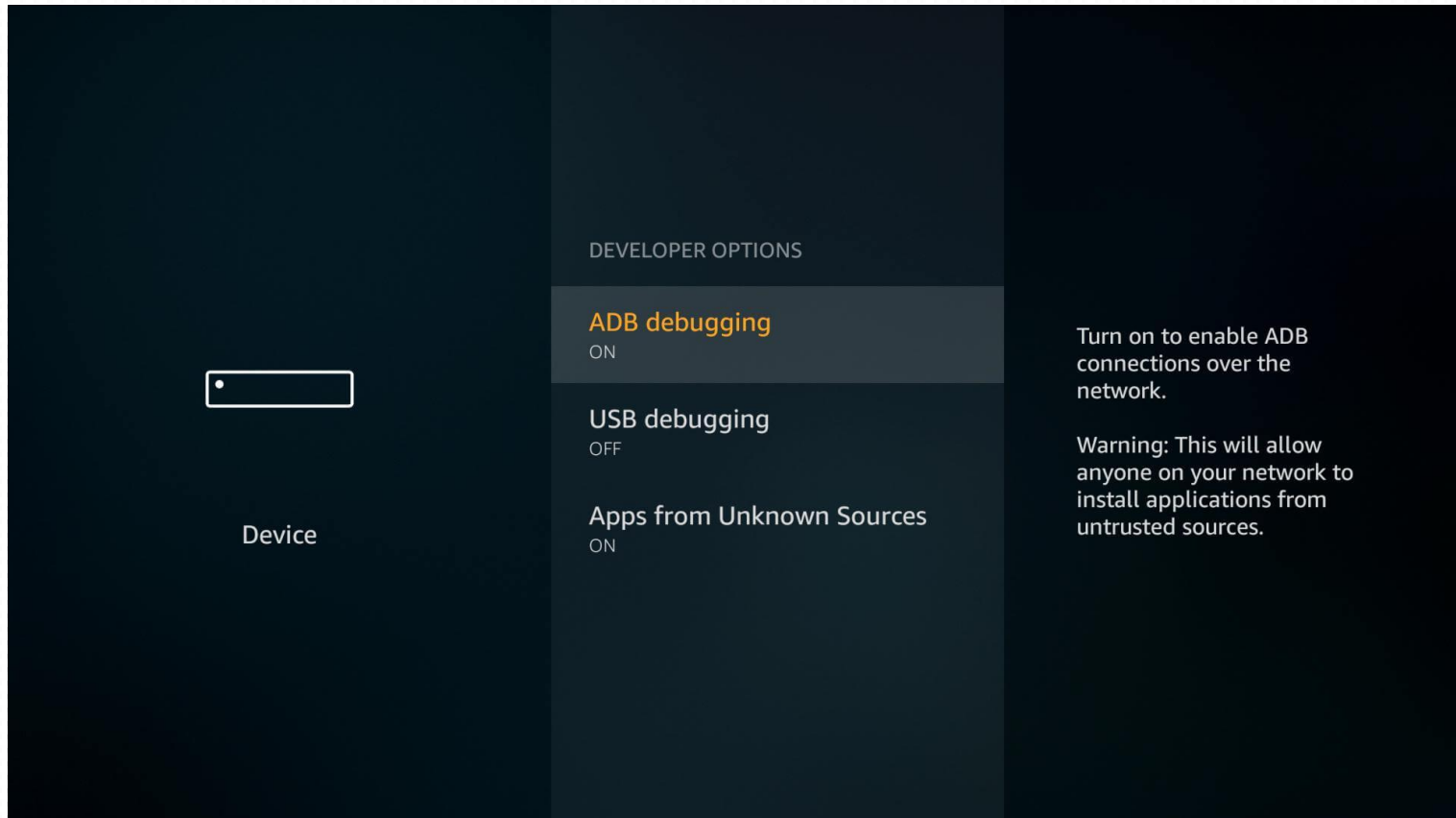
Connect Device (Android TV) over USB III



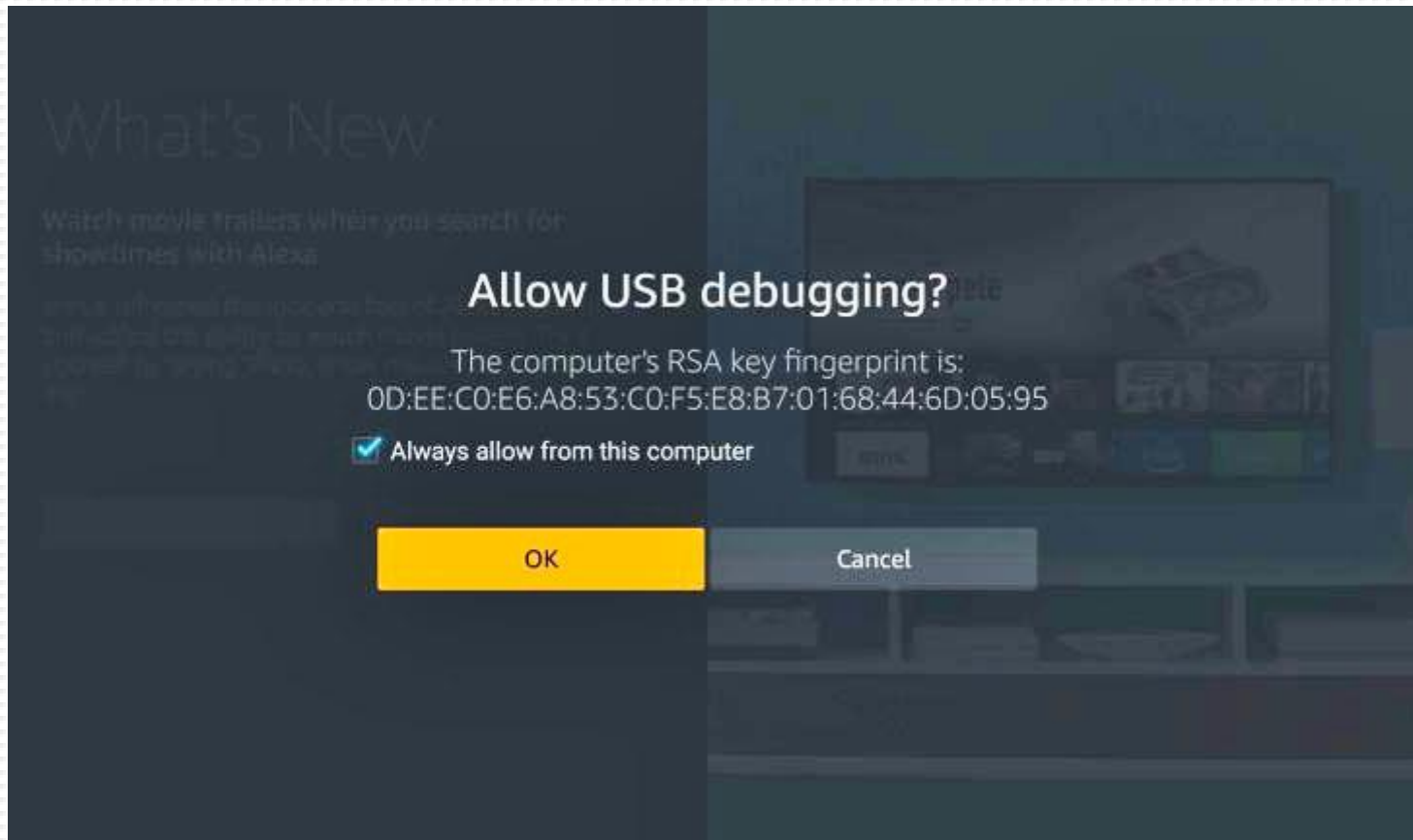
Connect Device (Fire TV) over Network I



Connect Device (Fire TV) over Network II



Connect Device (Fire TV) over Network III



Connect Device (Fire TV) over Network IV

Terminal

```
+ D:\Users\AR430805\AndroidStudioProjects\SampleApplication>cd D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools
X D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb start-server
* daemon not running; starting now at tcp:5037
* daemon started successfully

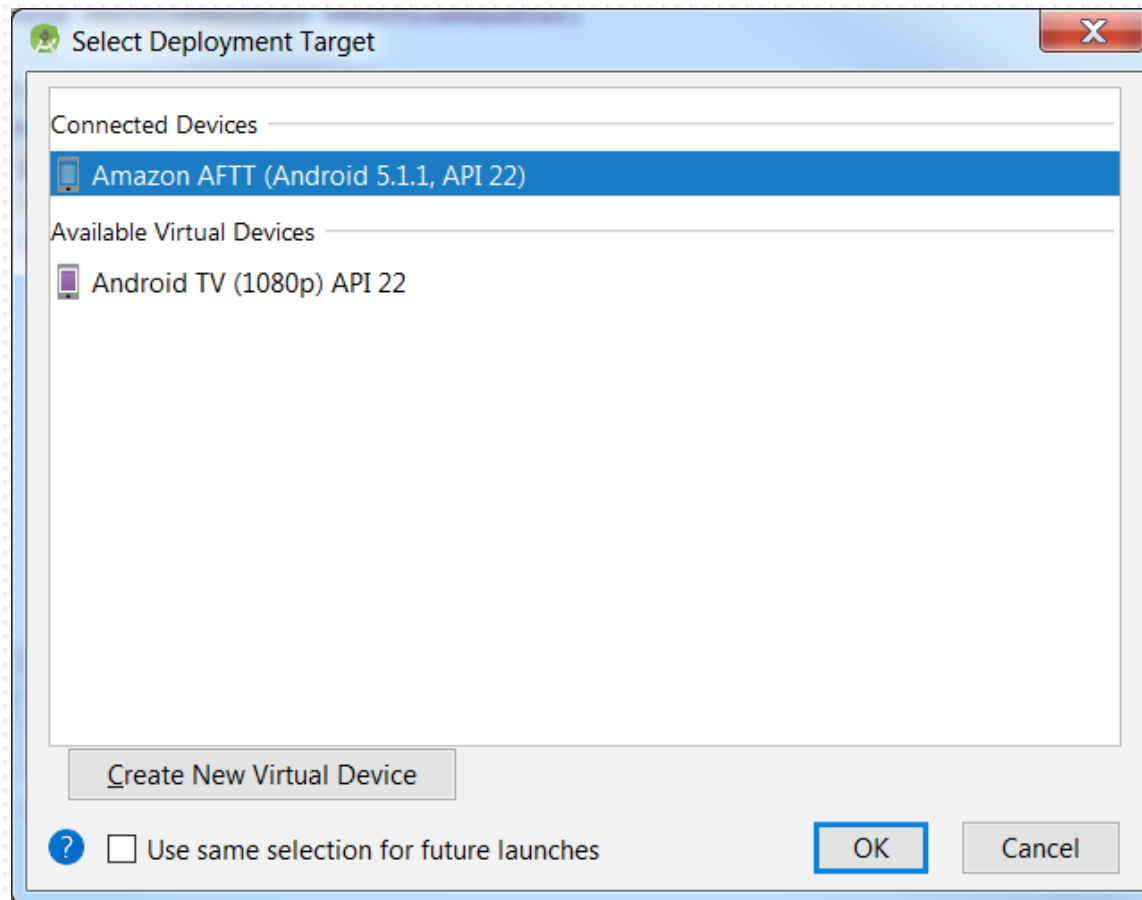
D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb connect 192.168.1.119
^C
D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb connect 192.168.0.119
failed to authenticate to 192.168.0.119:5555

D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb connect 192.168.0.119
already connected to 192.168.0.119:5555

D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb devices
List of devices attached
192.168.0.119:5555    device
192.168.1.119:5555    offline

D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>
```


Running App on Device (Fire TV)



Installing App on Device (Fire TV)

- You can also install external apk files to Android devices
- "Apps from Unknown Sourced" option should be 'ON' in developer options menu

Terminal

+

```
D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb start-server
```

×

```
D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb connect 192.168.0.116  
connected to 192.168.0.116:5555
```

```
D:\Users\AR430805\AppData\Local\Android\sdk\platform-tools>adb install -r D:\lifecell_v1.09.apk  
[ 40%] /data/local/tmp/lifecell_v1.09.apk
```

6: Logcat

Terminal

Build

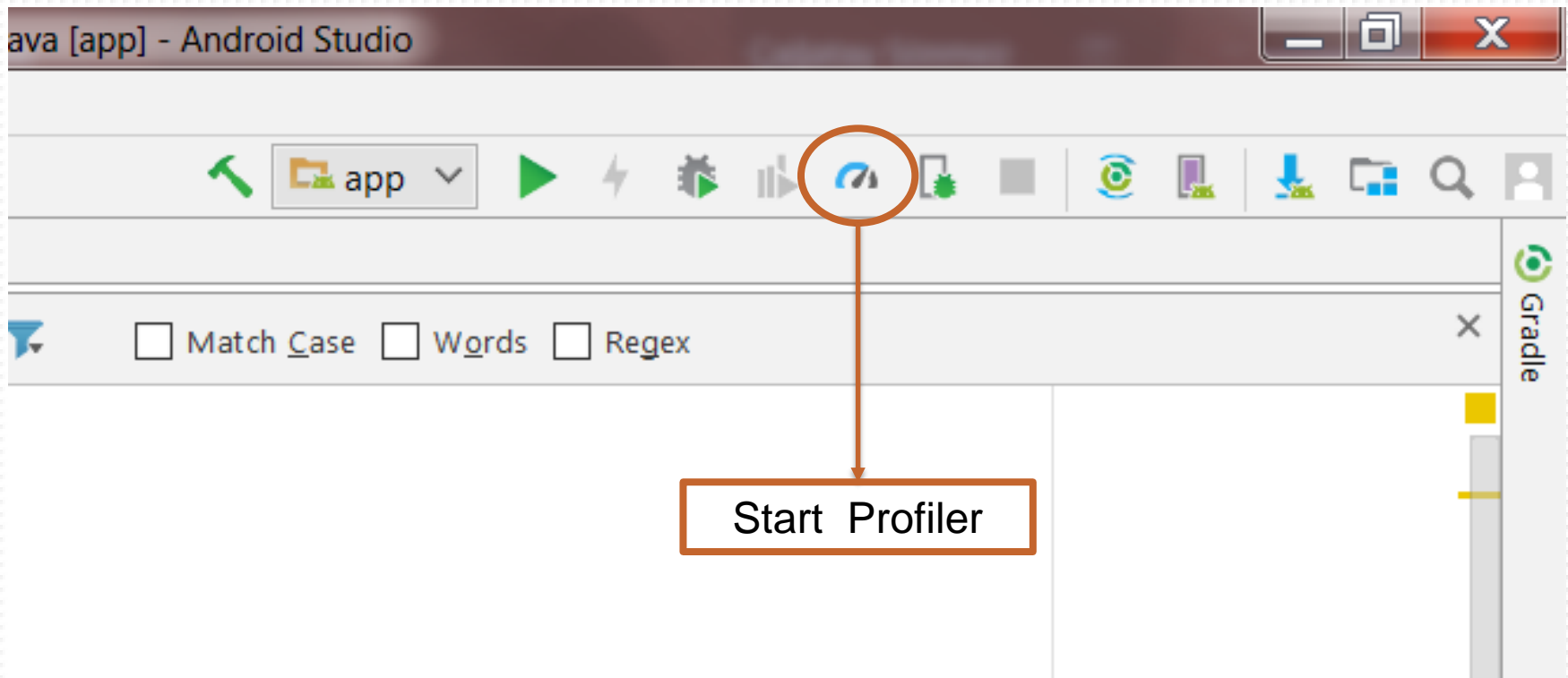
TODO

adb install options

Command	Action
adb install test.apk	
adb install -l test.apk	forward lock application
adb install -r test.apk	replace existing application
adb install -t test.apk	allow test packages
adb install -s test.apk	install application on sdcard
adb install -d test.apk	allow version code downgrade
adb install -p test.apk	partial application install

Monitoring App Performance

- Use Profiler to monitor the resource usage of your application



Profiler

Stop Session

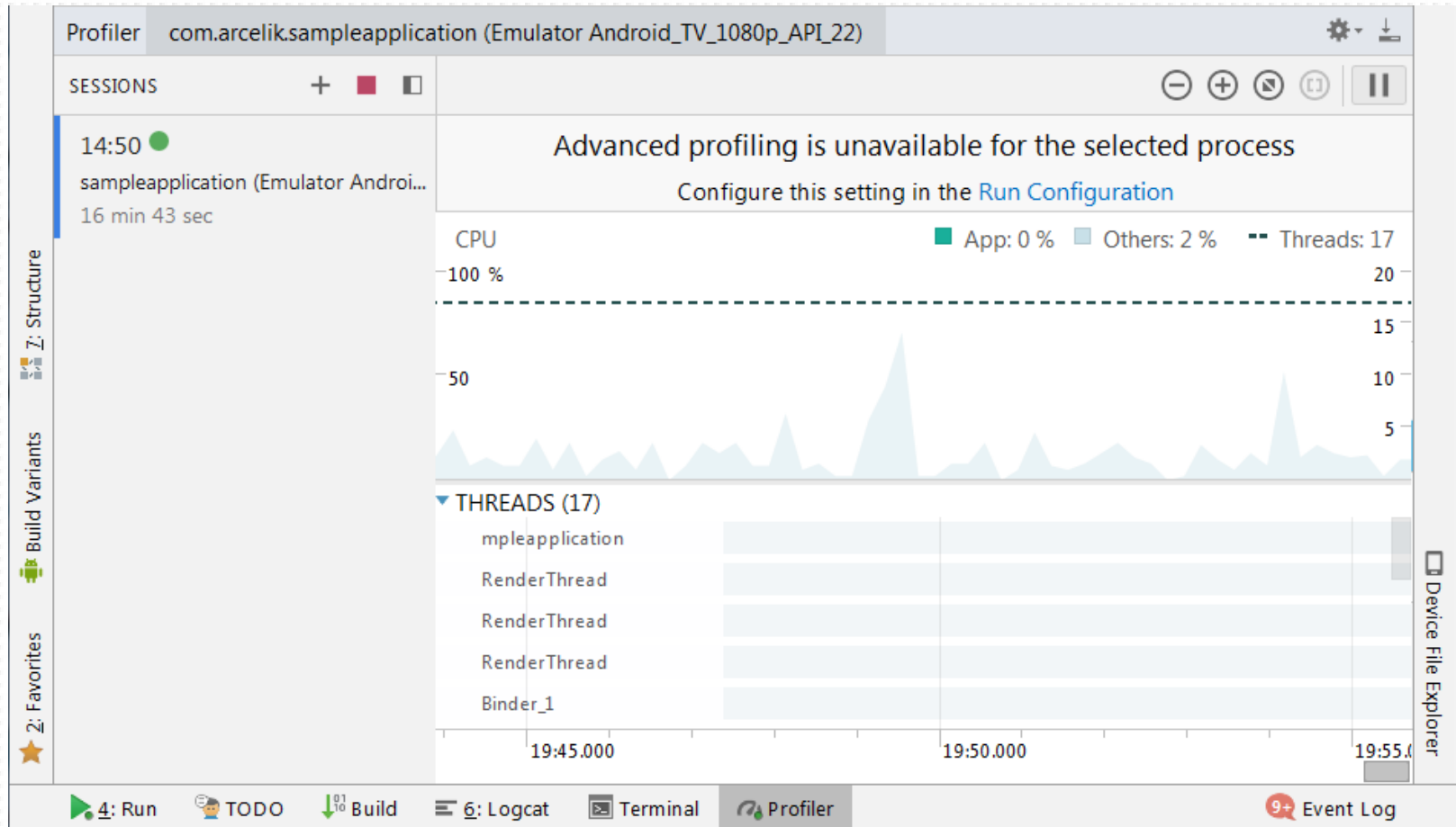
Pause/Resume Live Monitoring

The screenshot displays the Android Studio Profiler interface for the process `com.arcelik.sampleapplication (Emulator Android_TV_1080p_API_22)`. The interface is divided into several sections:

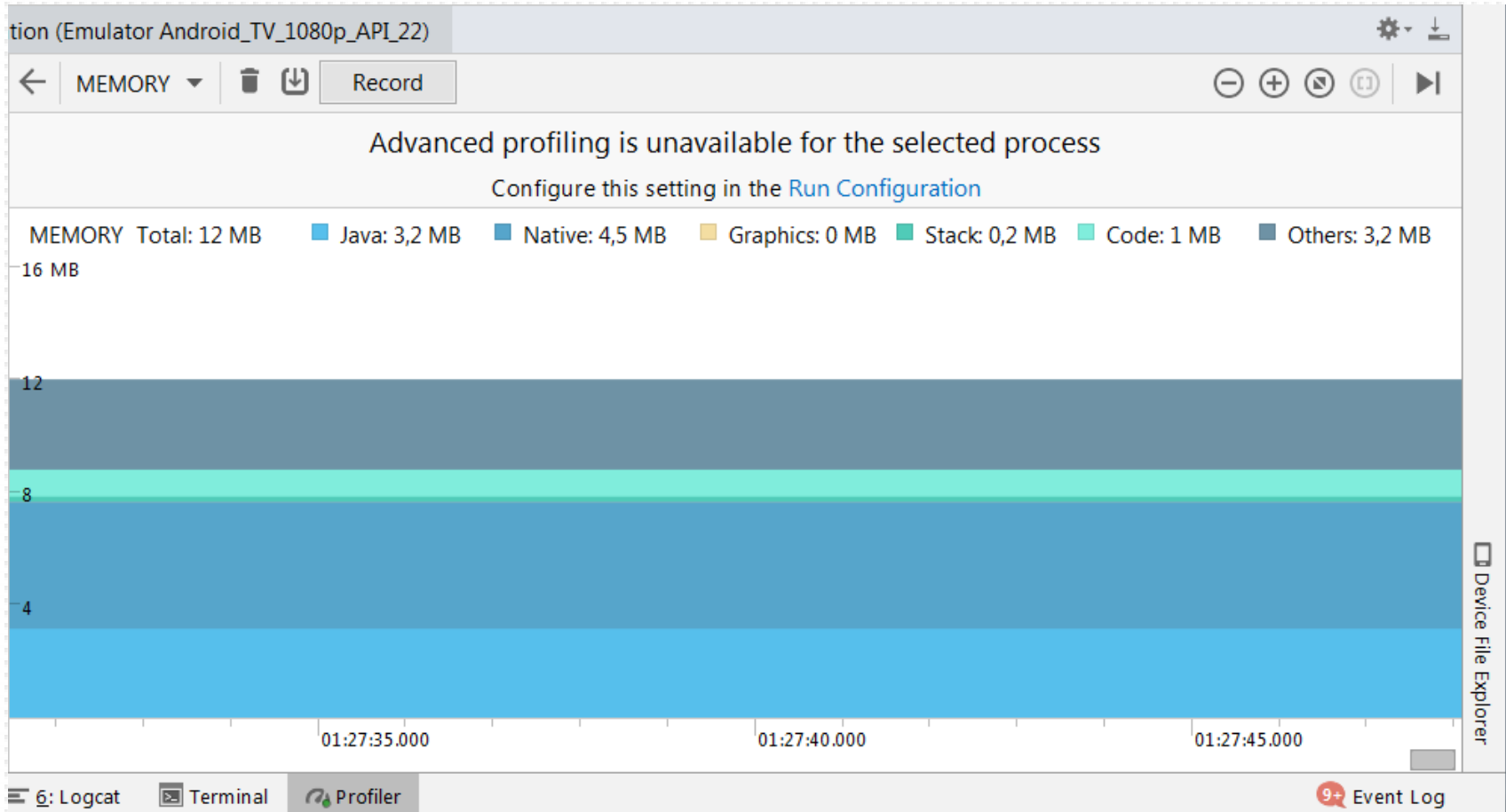
- SESSIONS:** A list of sessions with a red square icon for stopping the session and a pause icon for pausing/resuming live monitoring.
- Advanced profiling is unavailable for the selected process:** A message indicating that advanced profiling is not supported for this process, with a link to [Configure this setting in the Run Configuration](#).
- CPU:** A graph showing CPU usage over time, with a peak of 2%.
- MEMORY:** A graph showing memory usage over time, with a peak of 11.9 MB.
- NETWORK:** A graph showing network usage over time, with a peak of 4 B/s.
- Energy profiler unavailable:** A message indicating that the energy profiler is not supported for this device, with a link to [Learn More](#).

The bottom of the interface features a toolbar with icons for Run, TODO, Build, Logcat, Terminal, Profiler, and Event Log.

Profiler - CPU Usage




Profiler - Memory Usage



Profiler - Java Heap

tion (Emulator Android_TV_1080p_API_22)

← MEMORY ▾  Record

Dump Java Heap

Advanced profiling is unavailable for the selected process
Configure this setting in the [Run Configuration](#)

MEMORY 32 MB

Graphics: 0 MB Stack: 0,2 MB Code: 1,3 MB Others: 3,1 MB

8 16 24 32 MB

800 12.900 13.000 13.100 13.200 13.300 13.400 13.500

Dump (00.556)

Instances

Heap Dump app heap ▾ Arrange by page ▾ 12.935 - 13.491 ▾

Package Name	Allocations	Shallo...	Retain...
> android	3.154	161.896	668.050
> com	1.806	39.528	81.484
> android	1.789	38.680	73.284
> arcelik	17	848	8.200
> sampleapplication	17	848	8.200
MyService	6	288	6.258
MyIntentService	1	52	846
MainActivity	1	272	800
MyService\$ServiceHandler	6	192	192
MainActivity\$1	1	20	80
MainActivity\$3	1	12	12

Instance View

Instance	Depth	Shallow Size	Retained Size
> MyService@316209472 (0x12d8f940)	2	48	1.242
> MyService@314582112 (0x12c02460)	2	48	1.242
> MyService@316371488 (0x12db7220)	2	48	1.242
> MyService@315697376 (0x12d128e0)	2	48	1.242
> MyService@316187984 (0x12d8a550)	2	48	1.242
> MyService@315207888 (0x12c9b0d0)		48	48
> mThread = {ActivityThread}	0	192	9.804
> mBase = {ContextImpl}		128	128
> mThread = {HandlerThread}		100	100
> mApplication = {Application}	1	28	68
> mActivityManager = {ActivityManagerProxy}	2	12	44
> mServiceHandler = {MyService\$ServiceHandler}		32	32
> mToken = {Binder}		32	32
> mClassName = {String}		24	24

References

Reference	Depth	Shallow S...	Retained ...
> HandlerThread@314939872 (0x12c599e0)		100	100
> mThread in Looper@314949728 (0x12c5c060)		20	20
> mThread in MyService@315207888 (0x12c9b0d0)		48	48

References

Debugging

- Android Studio provides a debugger to debug application
 - View logs
 - Attach the debugger to a running app
 - Use breakpoints
 - Analyze stack trace
 - View on-device files
 - Take screenshot
 - Record a video

Viewing System Logs

The screenshot displays the Android Studio interface with the Logcat window at the bottom. The main editor shows the `MyService.java` file, which includes a `Log.d` statement. The Logcat window is filtered by the application package `com.arcelik.sampleapplic`, the log level `Verbose`, and the tag `SAMPLE_APP_SERVICE`. A log entry is visible, showing the output of the `Log.d` statement. Annotations with orange boxes and arrows point to various elements: 'App Filter' points to the package name filter; 'Debug Level' points to the 'Verbose' level dropdown; 'Search' points to the tag filter; 'Printing Log' points to the `Log.d` statement in the code; and 'Logcat' points to the Logcat window header.

```
import android.widget.Toast;

public class MyService extends Service {
    public static final String TAG_SERVICE = "SAMPLE_APP_SERVICE";

    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            Log.d(TAG_SERVICE, msg: "handleMessage: " + msg);

            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            try {
                // ...
            } catch (InterruptedException e) {
                // ...
            }
        }
    }

    @Override
    public void onCreate() {
        mServiceLooper = Looper.getLooper(this);
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }

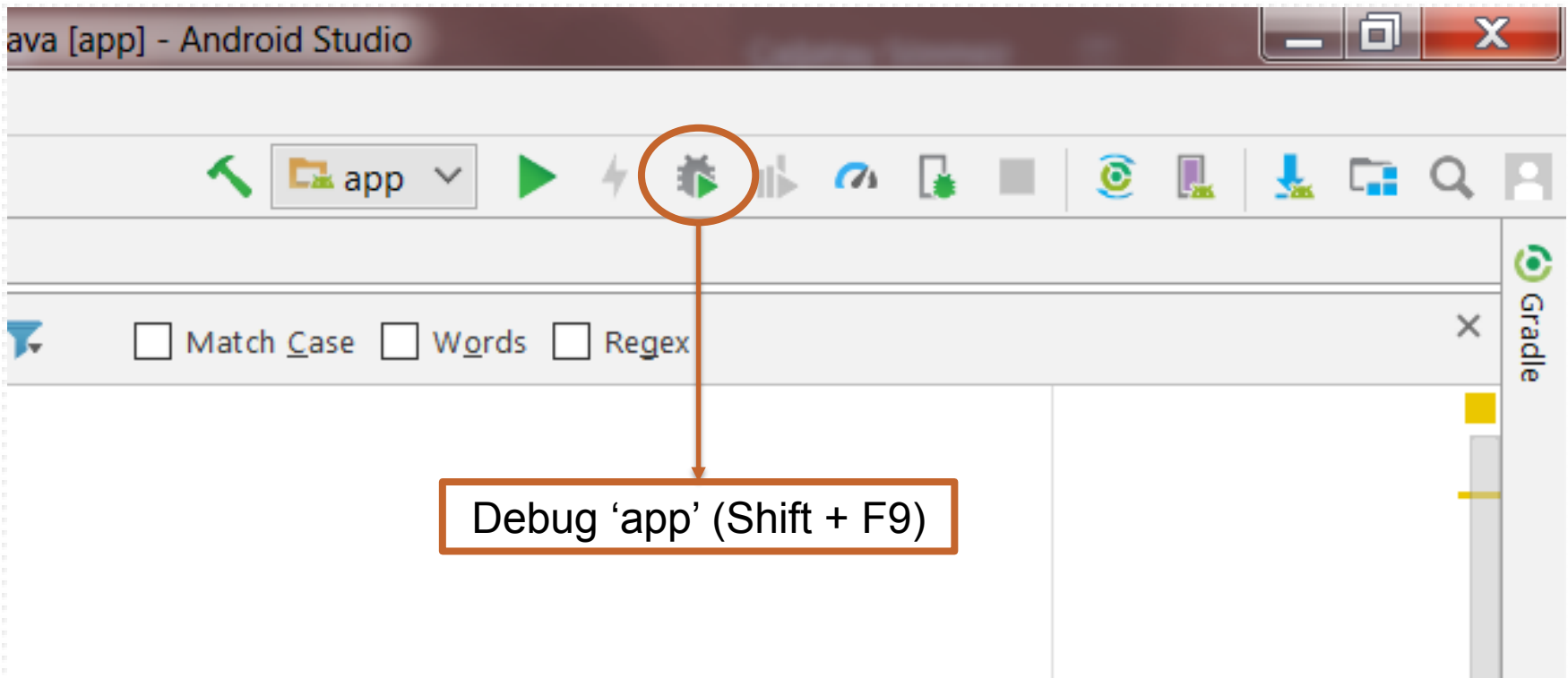
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        mServiceHandler.handleMessage(new Message());
        return START_STICKY;
    }
}
```

Logcat

Emulator Android_TV_... com.arcelik.sampleapplic Verbose SAMPLE_APP_SERVICE

10-30 16:01:19.025 4915-4937/com.arcelik.sampleapplication D/SAMPLE_APP_SERVICE: handleMessage: { when=0 what=0 arg1=1 target=com.arcelik.sampleapplication.MyService\$ServiceHa

Starting Debugger



Debug 'app' (Shift + F9)

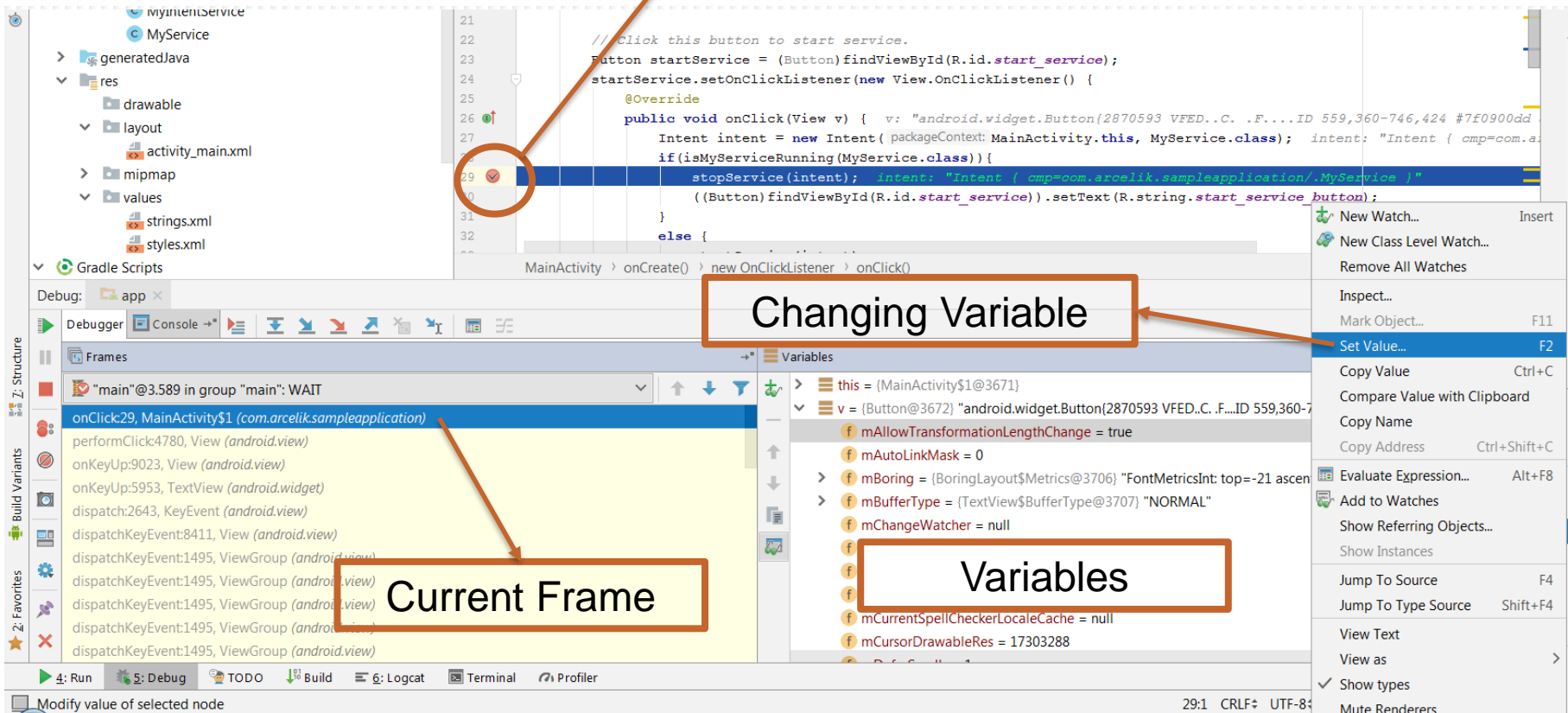
Using Breakpoint I

Break Point

Changing Variable

Current Frame

Variables



Using Breakpoint II

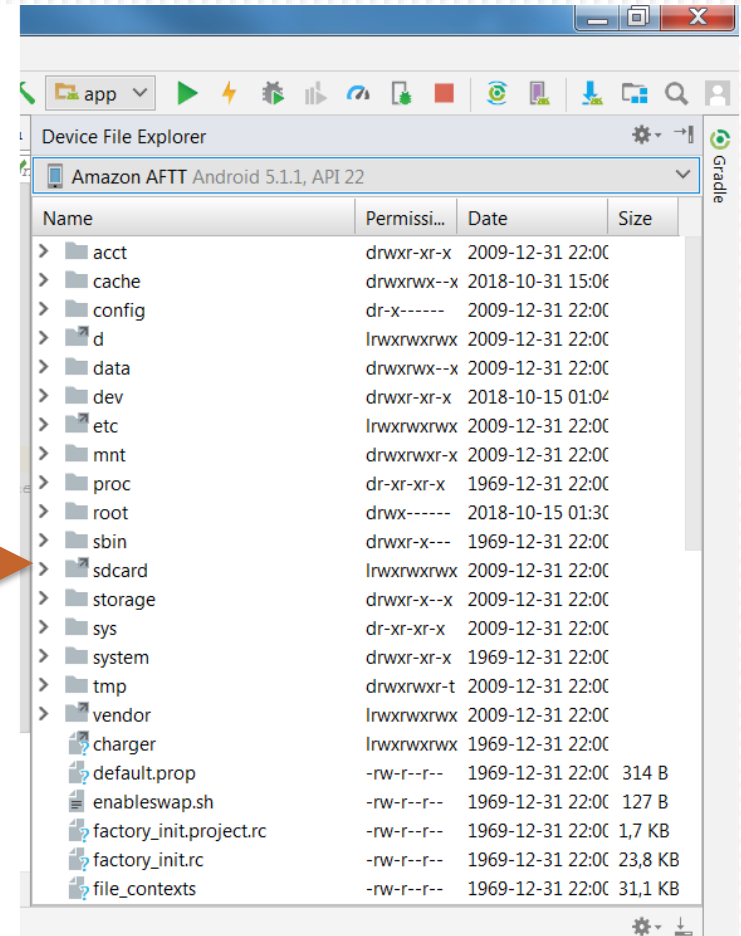
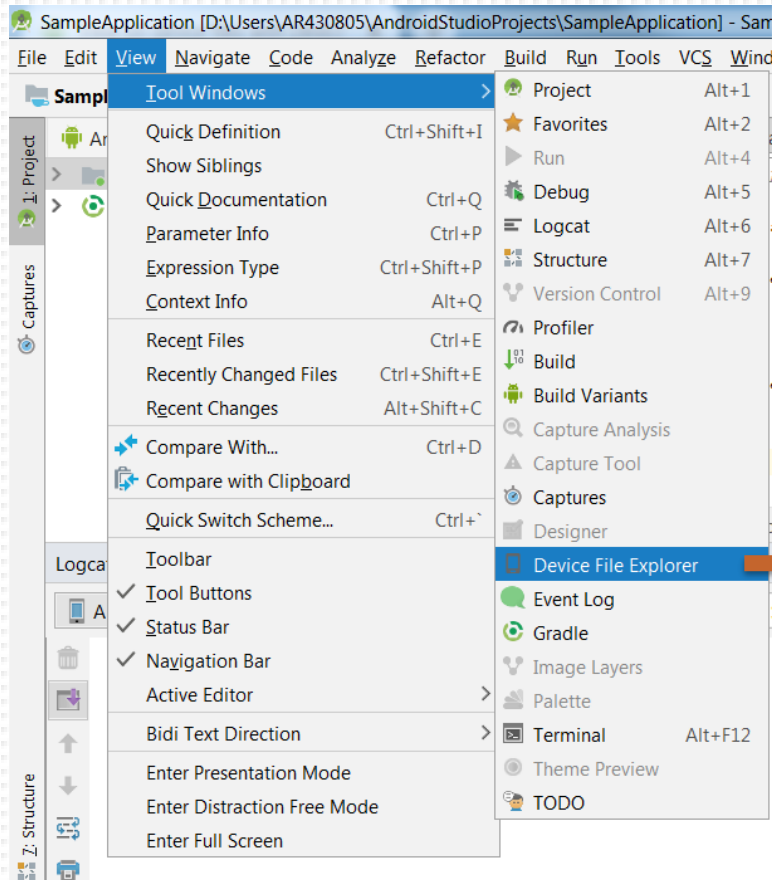
The image shows a screenshot of the Android Studio debugger interface. Several icons in the top toolbar are circled in orange, with arrows pointing to labels in orange-bordered boxes:

- Resume Program**: Points to the green play button icon.
- Pause Program**: Points to the orange pause button icon.
- Step Over (F8)**: Points to the blue icon with a downward arrow and a horizontal line.
- Step Into (F7)**: Points to the blue icon with a downward arrow and a vertical line.
- Step Out (Shift + F8)**: Points to the blue icon with an upward arrow and a horizontal line.

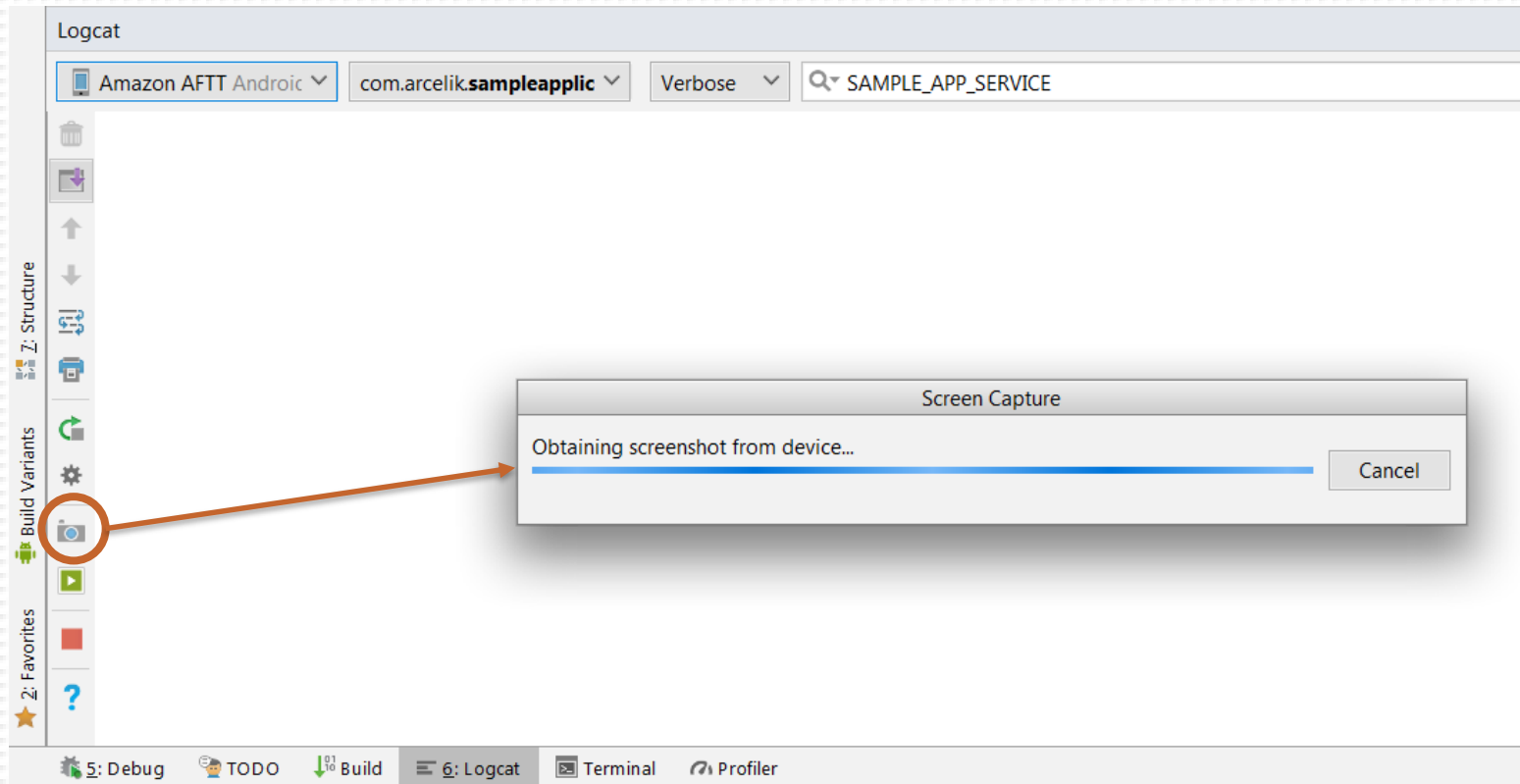
The interface also shows the following components:

- Debugger** and **Console** tabs at the top.
- Frames** panel on the left, showing a stack of frames.
- Current Frame**: `"main"@3.589 in group "main": WAIT`.
- Method Call Stack**:
 - `onClick:29, MainActivity$1 (com.arcelik.sampleapplication)` (highlighted in blue)
 - `performClick:4780, View (android.view)`
 - `onKeyUp:9023, View (android.view)`

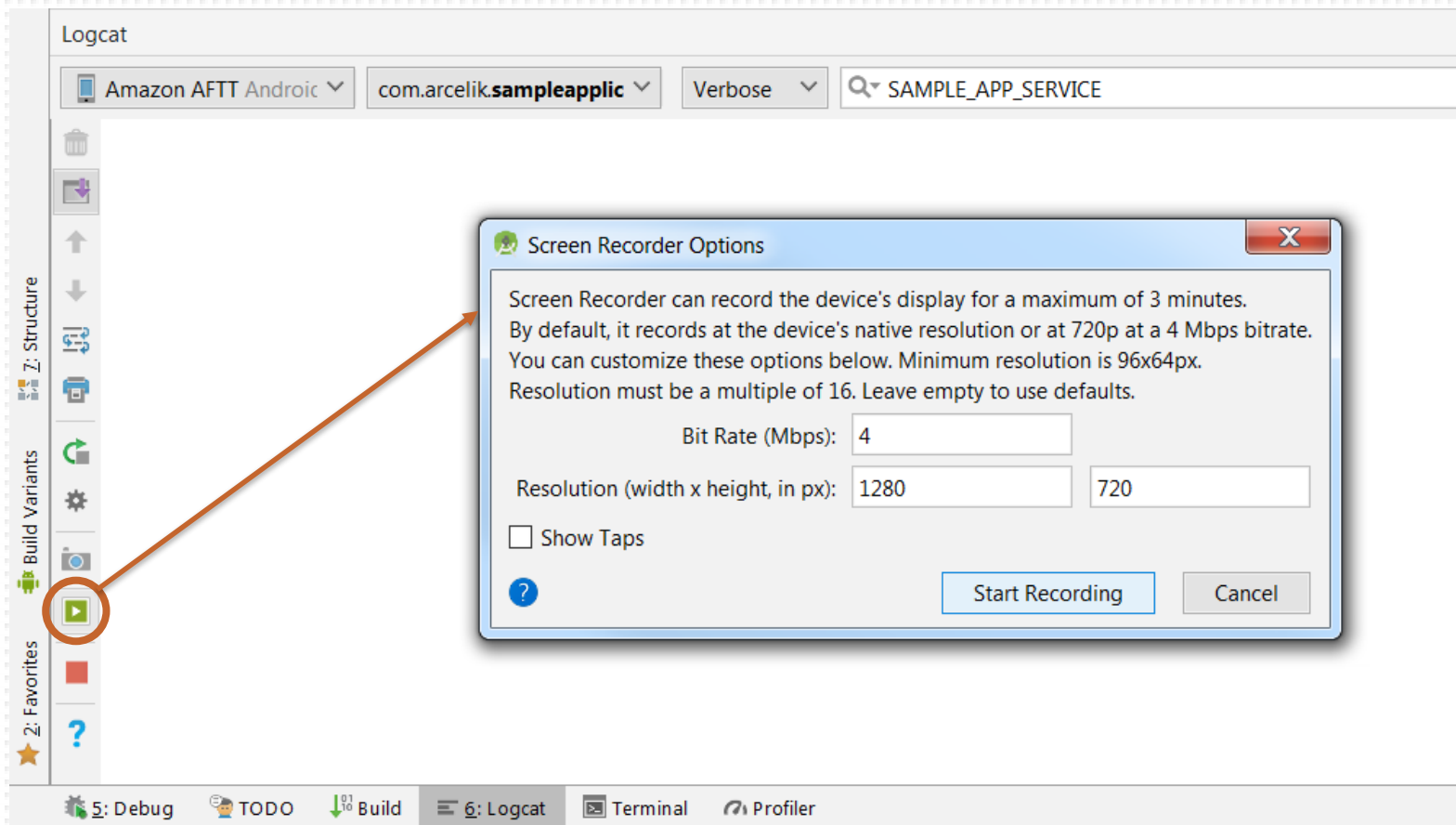
View On-Device Files



Take screenshot



Record Video



QUESTIONS?