Introduction to Xamarin.Android

Objectives

1. Create a Xamarin.Android project
2. Decompose an app into Activities
3. Build an Activity's UI
4. Write an Activity's behavior
5. Update your Android SDK

Demonstration

Preview the finished lab exercise

Create a Xamarin.Android project

Tasks

1. Survey Xamarin.Android features
2. Create a new project in your IDE

What is a Xamarin.Android app?

- *Xamarin.Android apps* are apps built with Xamarin's tools and libraries

UI is made from Xamarin's wrappers around the native Android views →

← Code is written in C#

Development environment

- Xamarin.Android apps are coded in C# and built with either Xamarin Studio or Visual Studio

```
var employees = new List<Employee>();
var seniors = from e in employees where e.Salary > 50000 select e;

var client = new HttpClient();
var result = await client.GetStringAsync("");
```

Supports latest C# features like generics, async/await, LINQ, lambda expressions, etc.

F# is also supported; however, this course will use C#.

C# idioms

- The Xamarin.Android bindings to Android libraries provide a familiar programming experience for C# developers

```
EditText input = new EditText(this);

String text = input.getText().toString();

input.addTextChangedListener(new TextWatcher() { ... });
```

Java uses get/set methods, listeners, etc.

```
var input = new EditText(this);

string text = input.Text;

input.TextChanged += (sender, e) => { ... };
```

Xamarin.Android uses properties and events

Libraries

- Xamarin.Android apps can use utility classes from three libraries

Xamarin provides C# wrappers for all Android Java libraries

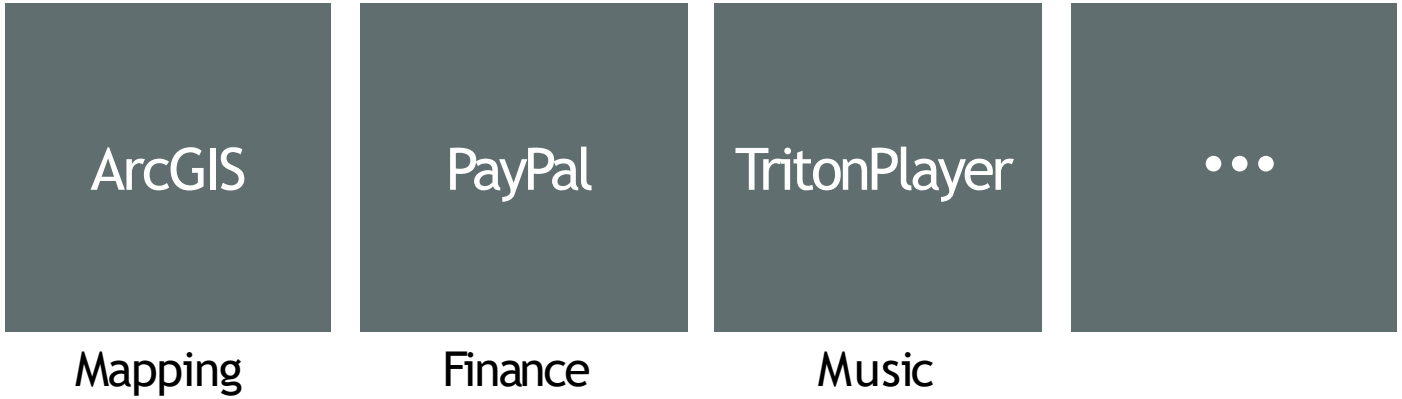
Xamarin provides C# wrappers for all Android APIs

Includes most .NET types but not the entire Mono library

When a new version of Android is released, the Xamarin wrappers are ready within days.

Third-party Java

- You can use JNI or a Bindings Library to incorporate third-party Java libraries into your Xamarin.Android app

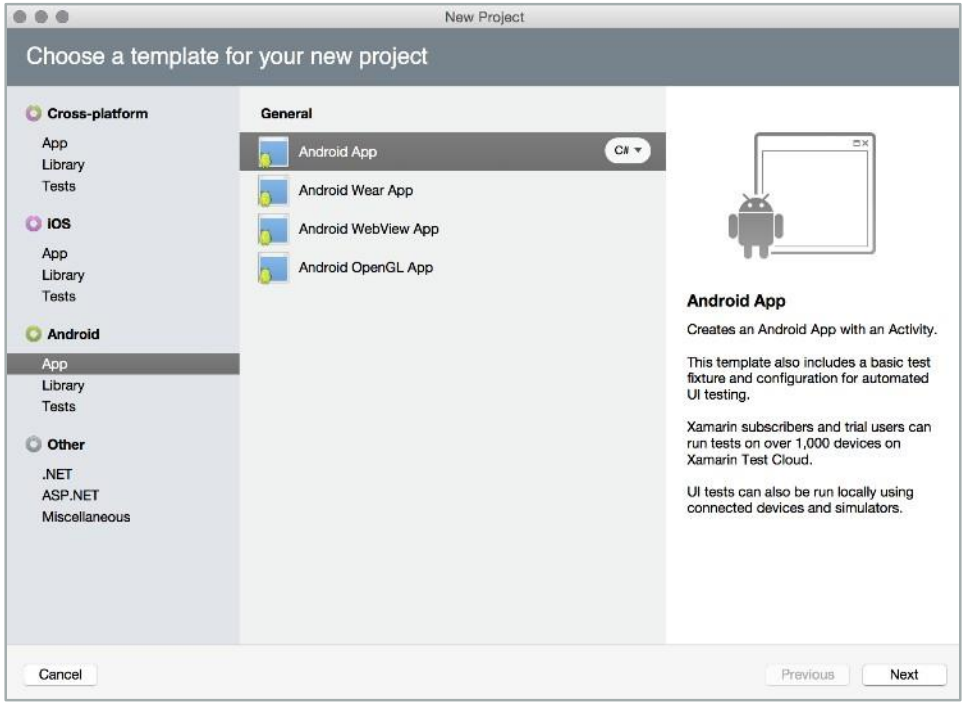


A Bindings Library is built on JNI and take some work to set up but is easier to use.

Xamarin.Android project templates

- Xamarin.Android includes several Android project templates

Xamarin Studio shown, Visual Studio has analogous templates →



Group Exercise

Create a Xamarin.Android project

Summary

1. Survey Xamarin.Android features
2. Create a new project in your IDE

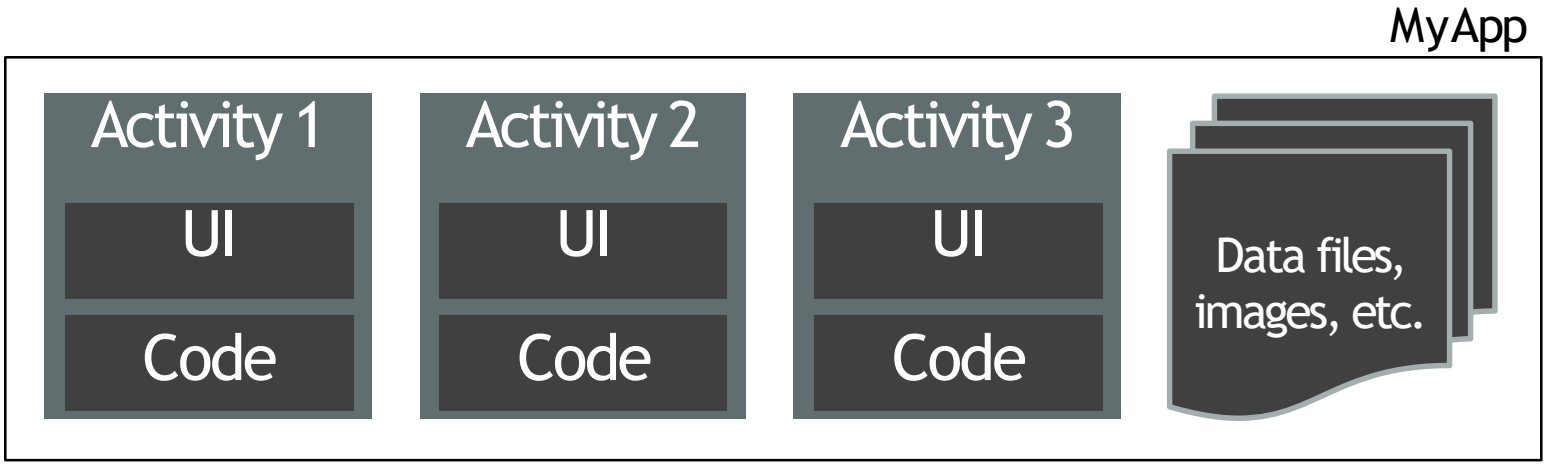
Decompose an app into Activities

Tasks

1. Define the concept of an Activity
2. Decompose an app into Activities

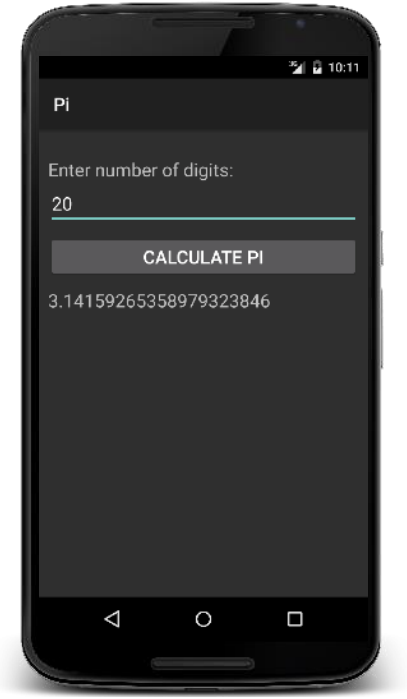
App structure

- An Android app is a collection of collaborating parts called *Activities*



What is an Activity?

- An *Activity* defines the UI and behavior for a single task

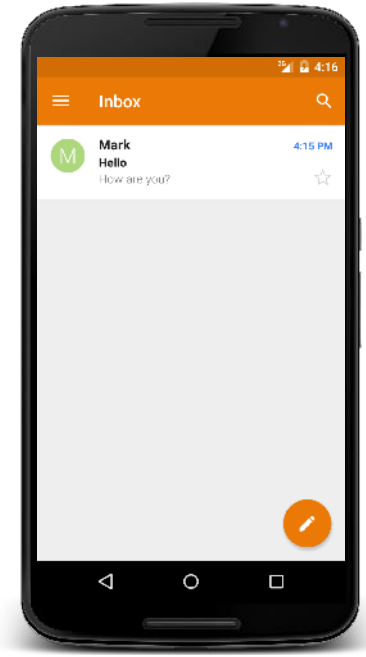


The "Pi" Activity has UI and coded behavior

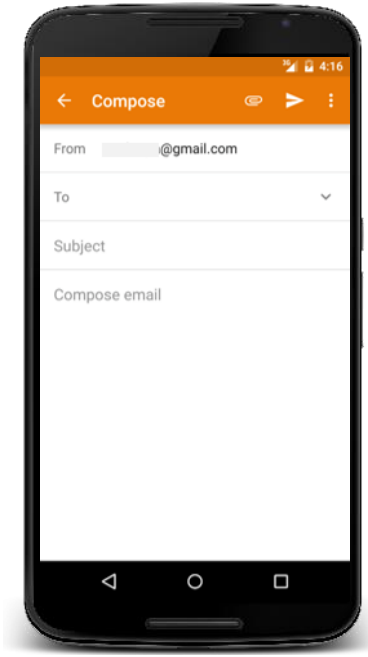
```
void OnClick(object sender, EventArgs e)
{
  int digits =int.Parse(input.Text);

  string result =CalculatePi(digits);

  output.Text =result;
}
```

Activity example: Email

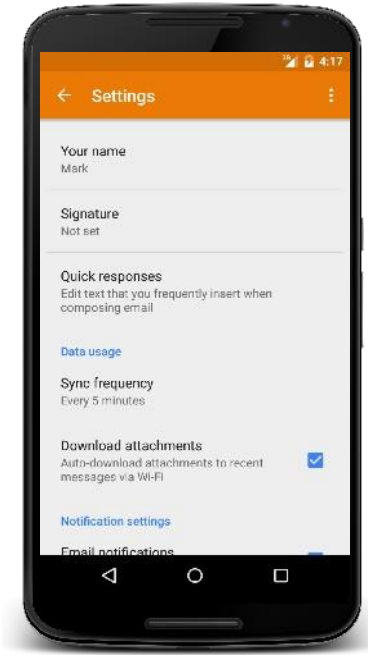
- The Email app has several activities



Messages Activity



Compose Activity



Settings Activity

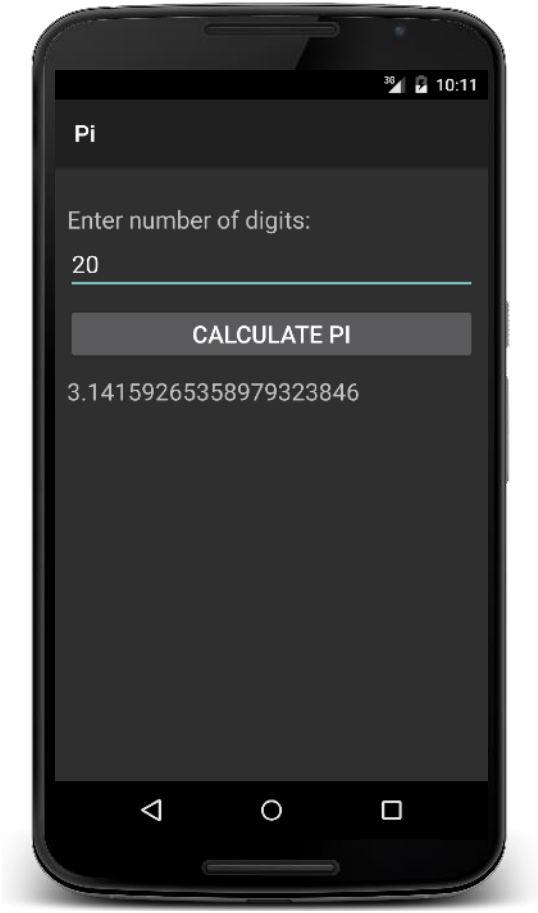
Summary

1. Define the concept of an Activity
2. Decompose an app into Activities

Build an Activity's UI

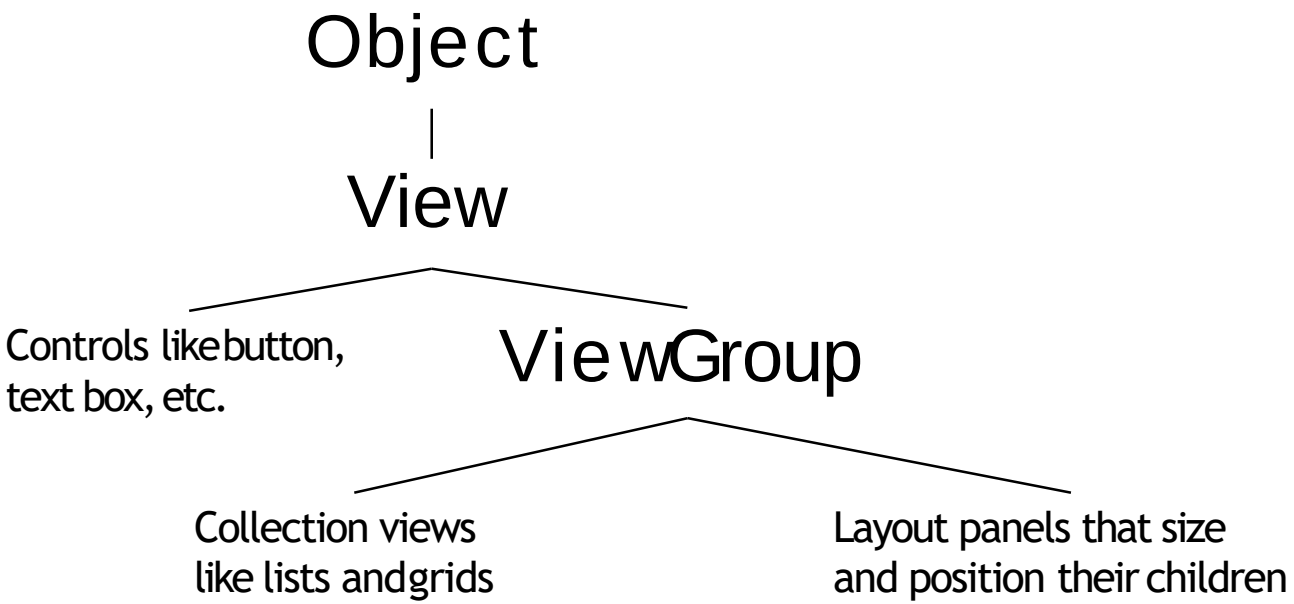
Tasks

1. Add Views to a Layout in XML
2. Use the Designer tool



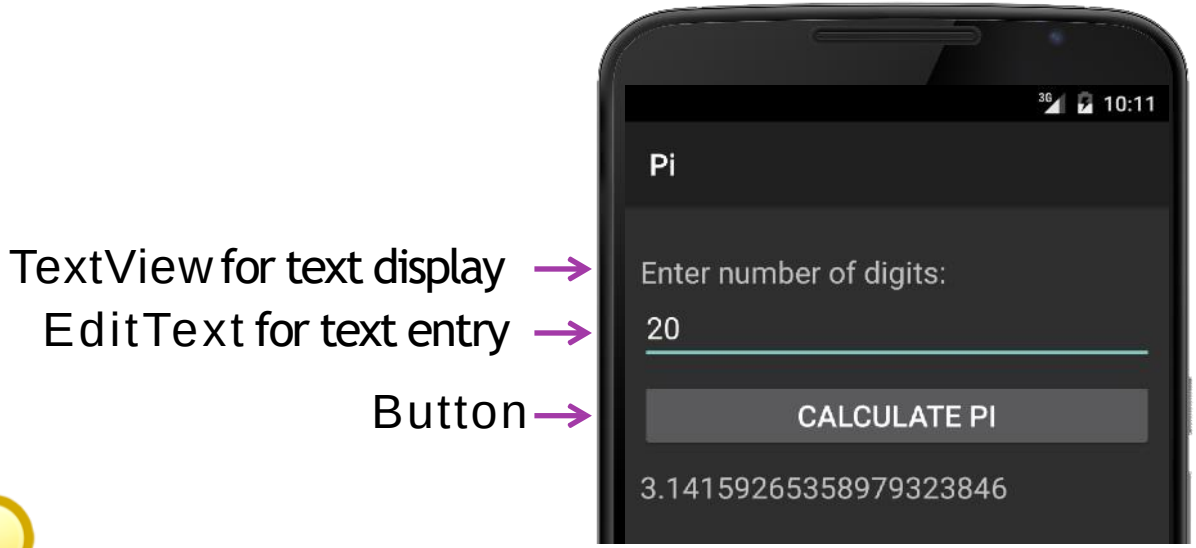
UI elements

- An Android UI is composed of Views and ViewGroups



What is a View?

- A *View* is a user-interface component with on-screen visuals and (typically) behavior such as events

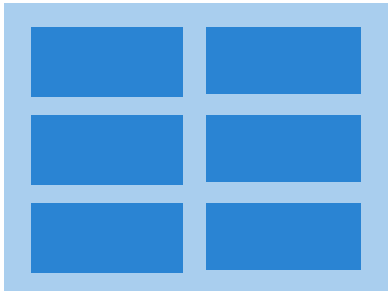


Views are also called *widgets*. Many are defined in the `Android.Widget` namespace.

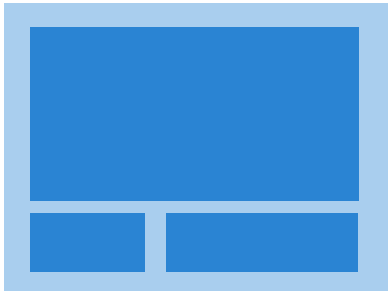
What is a layout?

- A *layout* is a container that manages a collection of child views and calculates their size/position on screen

LinearLayout
Single row or column



GridLayout
Rows and columns



RelativeLayout
You specify how each is positioned relative to neighbors

We will use only LinearLayout in this course.

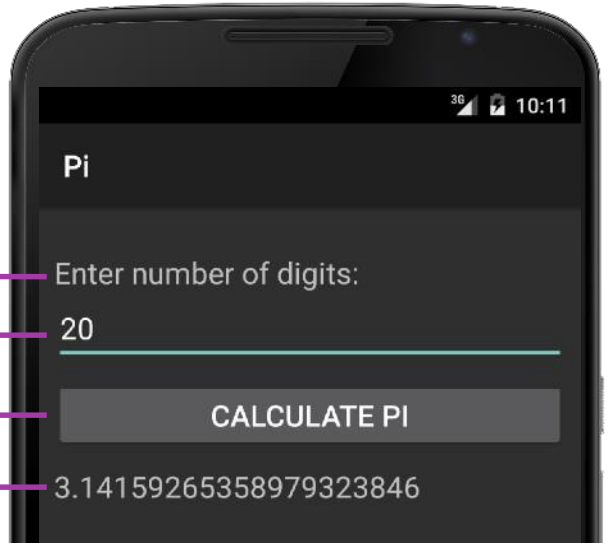
What is a layout file

- UI Views are typically created in an XML *layout file* (.axml)

Pi.axml

```
<LinearLayout ... >
  <TextView ... />
  <EditText ... />
  <Button ... />
  <TextView ... />
</LinearLayout ... >
```

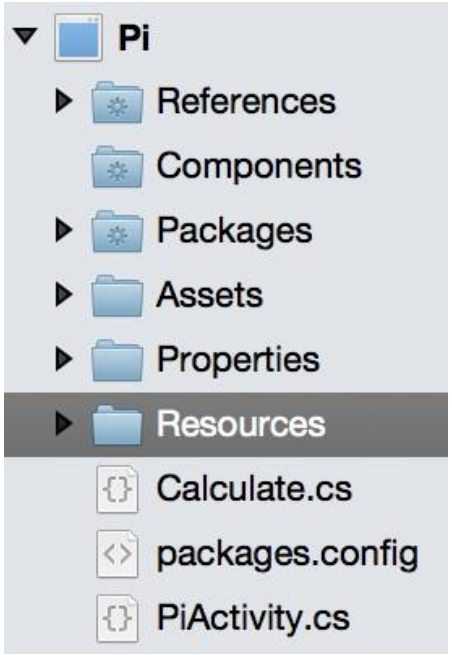
Child views are nested inside a layout panel



What are Resources?

- *Resources* are non-code files packaged with your app

Placed in the app's Resources folder



Note: Xamarin Studio shown, Visual Studio is similar

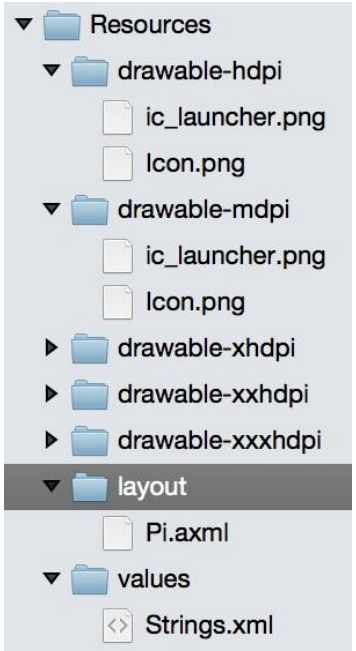
Where to define your layout files

- Layout files are a Resource and must be placed in the `layout` folder

Images in many sizes for screens of different pixel densities →

Layout files →

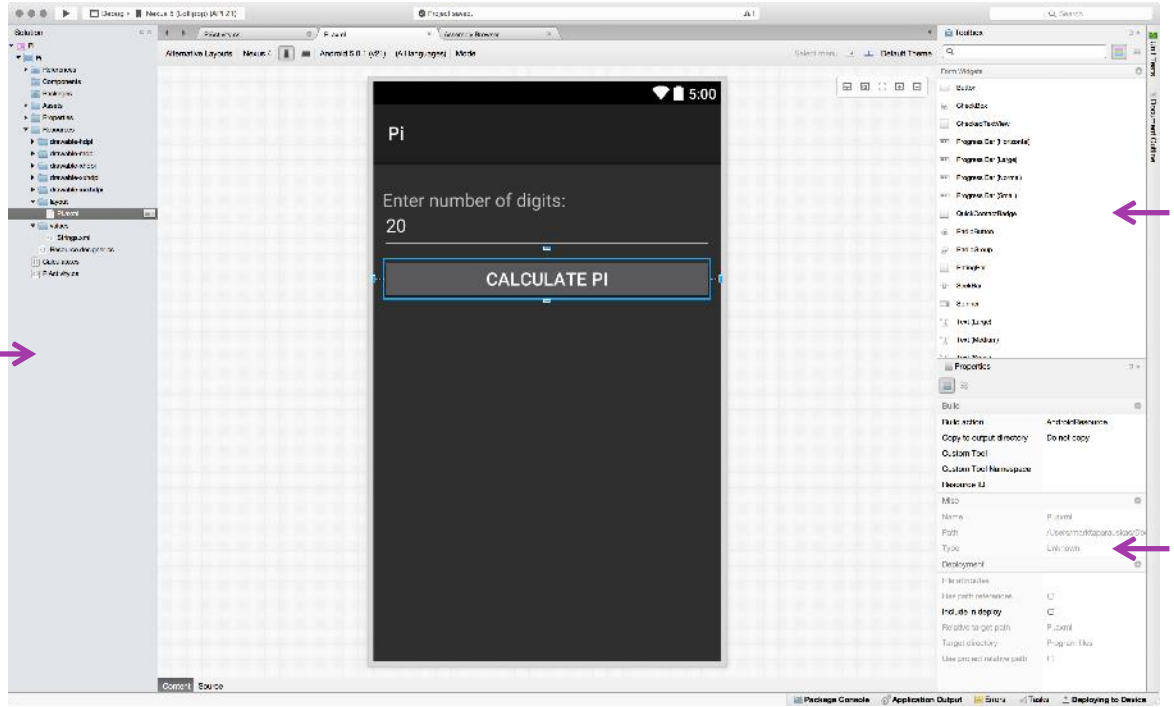
Strings, colors, etc. →



Note: Xamarin Studio shown, Visual Studio is similar

UI Designer

- Xamarin provides a UI design tool for creating and editing layout XML



Available for Xamarin Studio and Visual Studio

Toolbox to add new Views with drag-drop

Property editor

View attributes

- XML attributes are used to set properties on the underlying objects

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" andro id:orientation="vertical">
  <TextView android:text="Enter number of digits:" ... />
  ...
</LinearLayout>
```


TextView, EditText, and Button have a text attribute that sets their Text property

LinearLayout has an orientation that sets its Orientation property

Attributes names do not always match the underlying property names. See the Android documentation on each class (e.g. TextView) for a table of the XML attribute names.

Android namespace

- View attributes must be prefixed with the Android namespace when defined in XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical">
  <TextView android:text="Enter number of digits:" ... />
  ...
</LinearLayout>
```

Prefix with namespace

Prefix with namespace

Android does not require the prefix on Elements so it is common practice to omit it.

View sizing [required]

- LinearLayout requires layout_width and layout_height on every view

Failure to set width and height yields a runtime exception

View sizing [automatic]

- There are two special values you can use to specify width and height

```
<LinearLayout ... >
  ...
  <TextView android:layout_width="match_parent" android:layout_height="wrap_content" ... />
  ...
</LinearLayout>
```

same size as
the parent view

just large enough to
fit around its content

match_parent is the replacement for the equivalent-but-deprecated fill_parent

Group Exercise

Add views to a layout file manually and with the Designer tool

Fixed units-of-measure

- You can use `px` (screen pixel), `pt` (1/72"), `in` (inch), and `mm` for sizing but they are not recommended since they do not adapt to different displays

```
<Button android:layout_width="100px" ... />
```

Always occupies 100 physical pixels, so it will be different size on different screens

What is a density-independent pixel?

- A *density-independent pixel* (dp) is an abstract unit of measure that maps to physical pixels at runtime based on screen density

```
<Button android:layout_width="100dp" ... />
```

The goal is for this to occupy about the same area on-screen regardless of the device's screen density. On a high-resolution screen, this would occupy more than 100 physical pixels.

Baseline density

- Android chose a baseline density of 160dpi, so 1dp=1px on a 160dpi screen

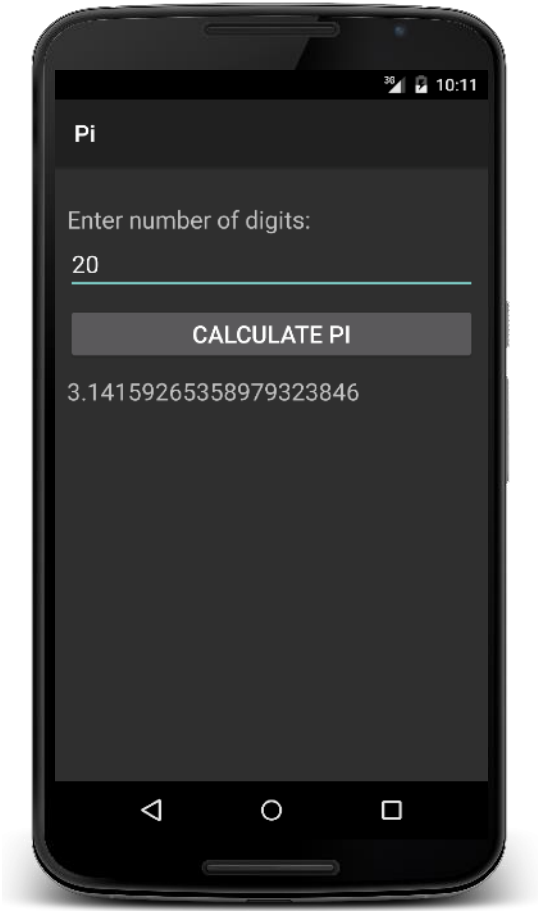
```
<Button android:layout_width="100dp" ... />
```

On a 160dpi screen, this would occupy 100 physical pixels

The baseline density is derived from the screen of the G1, the first Android device.

Summary

1. Add Views to a Layout in XML
2. Use the Designer tool



Write an Activity's behavior

Tasks

1. Designate a Main Activity
2. See how the MainActivity is listed in the app Manifest
3. Load an Activity's UI
4. Access Views from code

How to define an Activity

- An Activity has an XML layout file and a C# source file to drive the logic

Pi.axml

```
<LinearLayout . . . >
   <TextView    . . . >
   <EditText    . . . >
   <Button      . . . >
   <TextView    . . . >
</LinearLayout>
```

UI layout file

PiActivity.cs

```
[Activity]
public class PiActivity : Activity
{
   . . .
   . . .
}
```

C# class must inherit from Activity and be decorated with the ActivityAttribute

Main Activity

- An app uses the `ActivityAttribute` to designate an Activity as an entry point

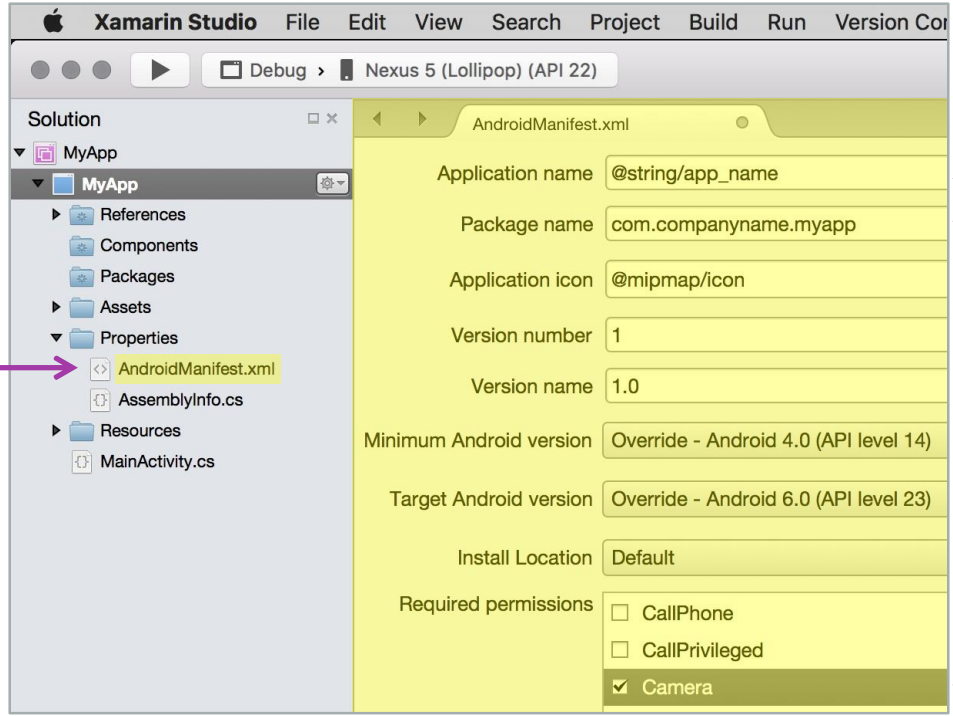
Only one activity can be marked as the main entry point

```
[Activity(MainLauncher = true)]
public class PiActivity : Activity
{
    ...
}
```

What is the AppManifest?

- An app's manifest describes the app to the Android OS

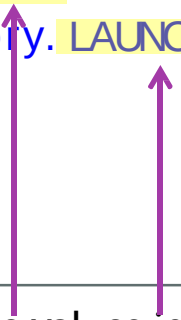
Every app must have a manifest and it must be named AndroidManifest.xml



Main Activity and the Manifest

- The Manifest tells Android which is your app's main Activity

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



The MainLauncher property in the ActivityAttribute creates these values in the Manifest. Android uses these to determine the app entry point and to list this activity on the launcher screen.

Activity initialization

- Override Activity.OnCreate to do your initialization

Must call base or get an exception

```
[Activity(MainLauncher =true)]
public class PiActivity : Activity
{
   protected override void OnCreate(Bundle bundle)
   {
      base.OnCreate(bundle);
      . . .
   }
   . . .
}
```

How to identify a layout file

- The build process auto-generates a Resource.Layout classthat contains an identifier for each of your layout files

Resource.designer.cs

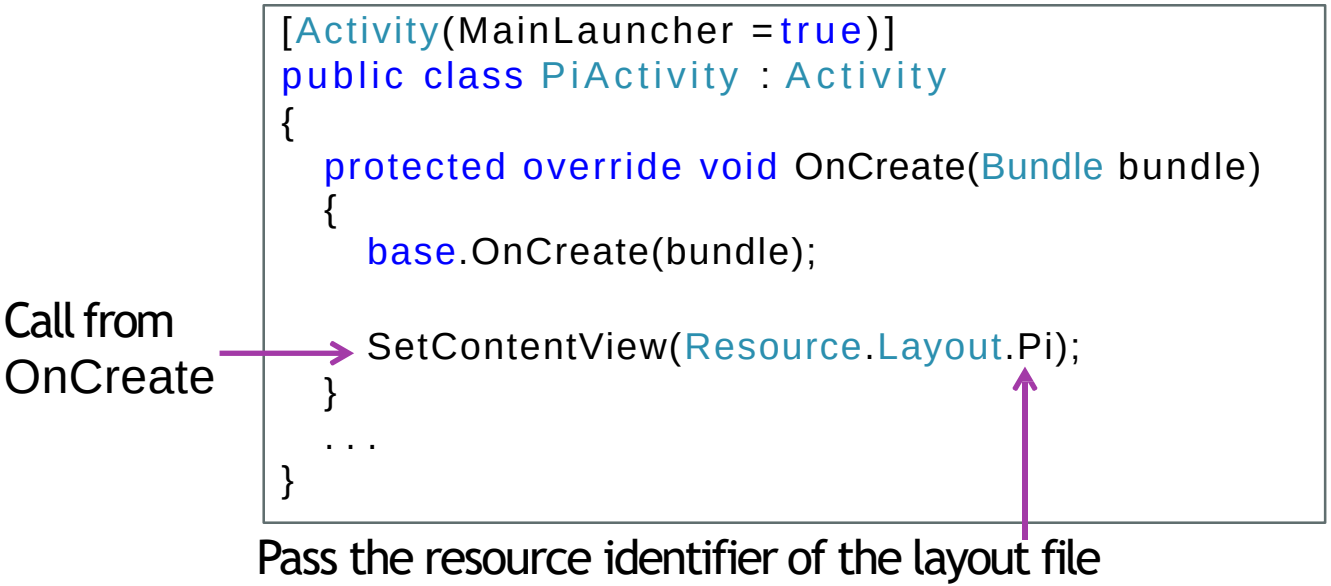
```
public partial class Resource
{
   public partial class Layout
   {
      public const int Pi = 2130903040;
      ...
   }
   ...
}
```

Use Resource.Layout.Pi to refer to this layout in code

The generated field matches the filename

UI Creation

- The `Activity.SetContentView` method instantiates all the Views in a layout file and loads them as the Activity's UI



What is an Id?

- The View class defines an Id property that is used to uniquely identify an instance of a View

```
namespace Android.Views
{
   public class View
   {
      public virtual int Id { get; set; }
      ...
   }
}
```

Notice that the type is int, not string

How to set an Id

- Set the Id of a View in XML using the id attribute and the syntax @+id

Set an id in the XML

```
<EditText android:id="@+id/digitsInput" ... />
```

Build tool generates a integer field and loads the integer into the View's Id

```
public partial class Resource
{
   public partial class Id
   {
      public const int digitsInput = 2131034113;
      ...
   }
   ...
}
```

Resource.designer.cs

How to access views from code

- Use `Activity.FindViewById` to lookup a View in an Activity's UI

```
[Activity(MainLauncher = true)]
public class PiActivity : Activity
{
  protected override void OnCreate(Bundle bundle)
  {
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Pi);

    var et = FindViewById<EditText>(Resource.Id.digitsInput);
    ...
  }
  ...
}
```

Individual Exercise

Implement an Activity's behavior and run your app in an emulator

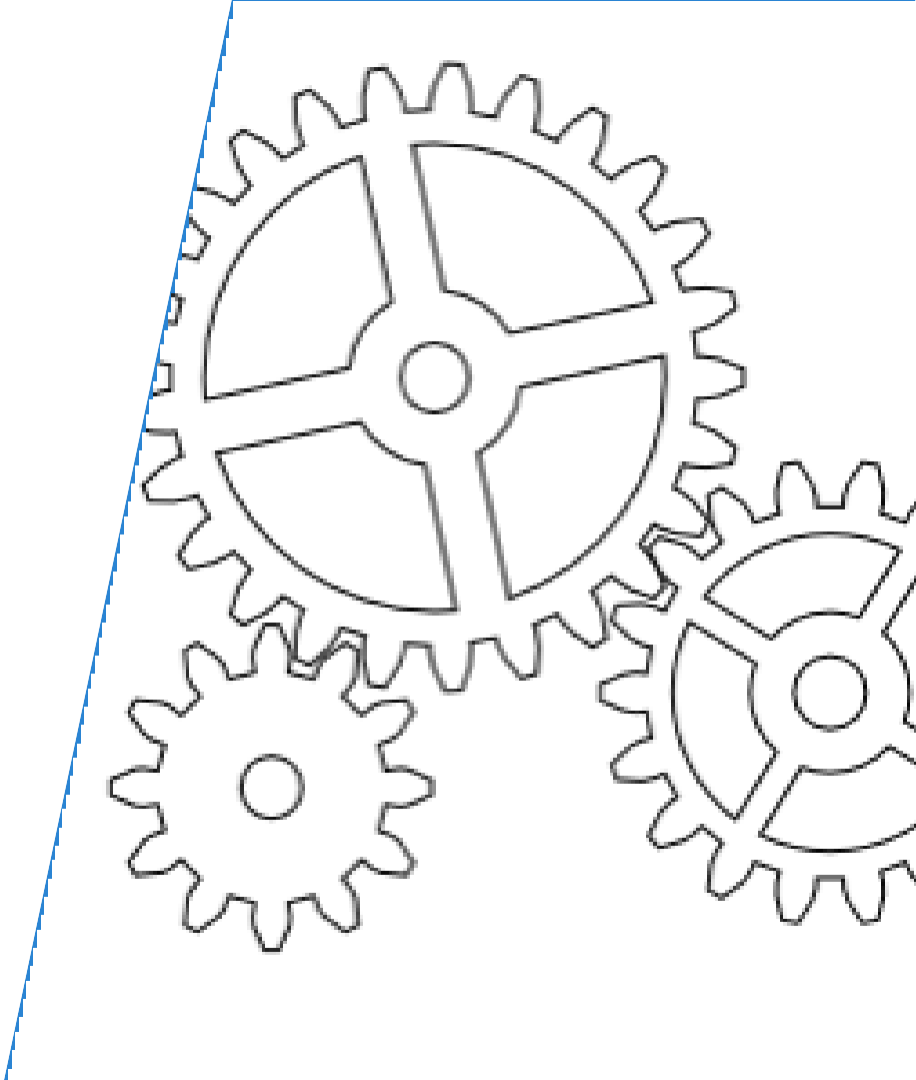
Summary

1. Designate a Main Activity
2. See how the MainActivity is listed in the app Manifest
3. Load an Activity's UI
4. Access Views from code

Update your Android SDK

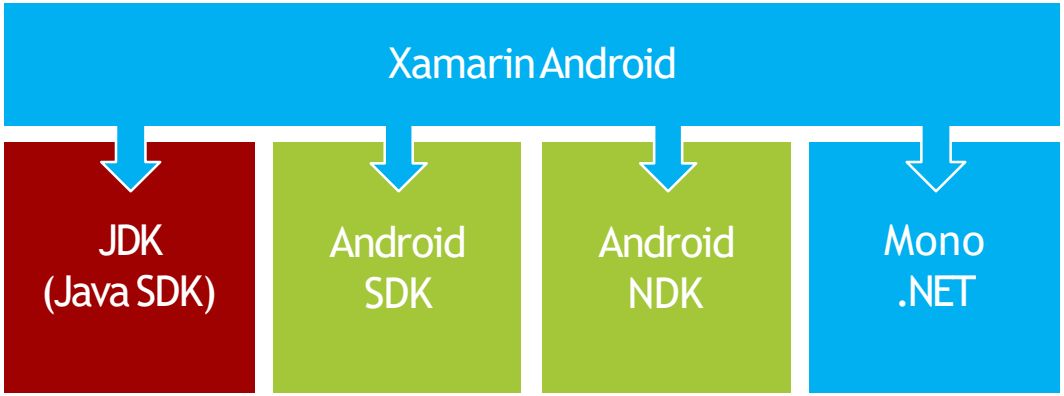
Tasks

1. Review native Android development
2. Understand the Xamarin.Android development process
3. Update your Android Tools
4. Update your Android Platform SDK



Motivation

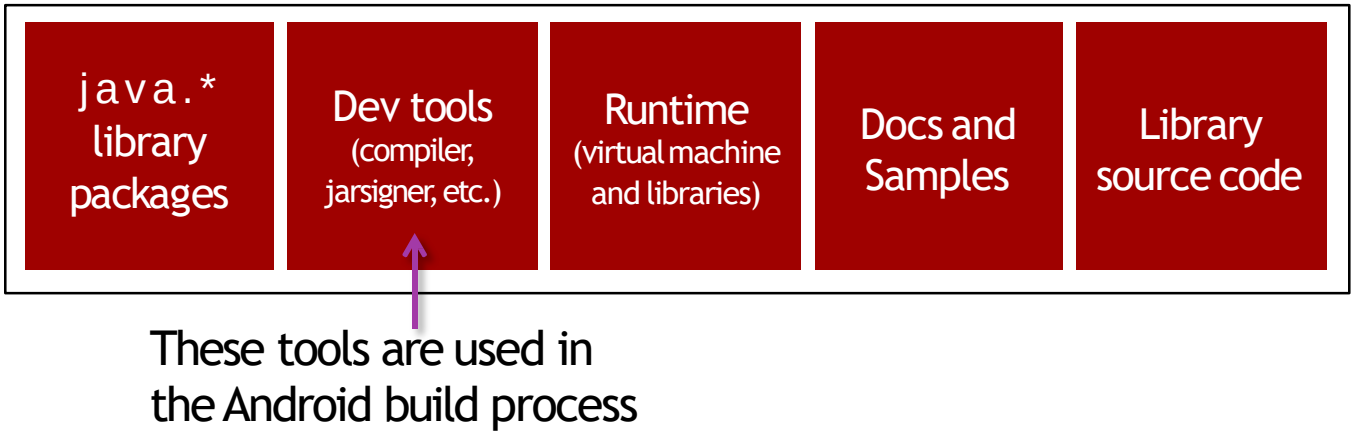
- Xamarin.Android uses native Android tools and libraries



You need to install updates to target new Android versions

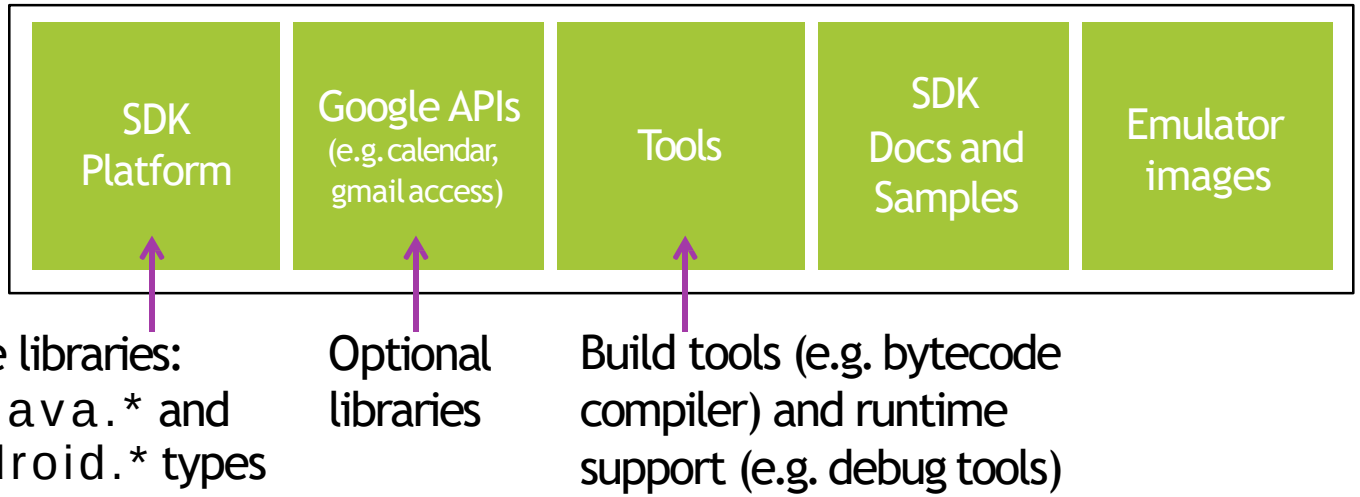
What is the JDK?

- The *Java SDK* (JDK) is the collection of libraries and tools needed to build and run Java applications



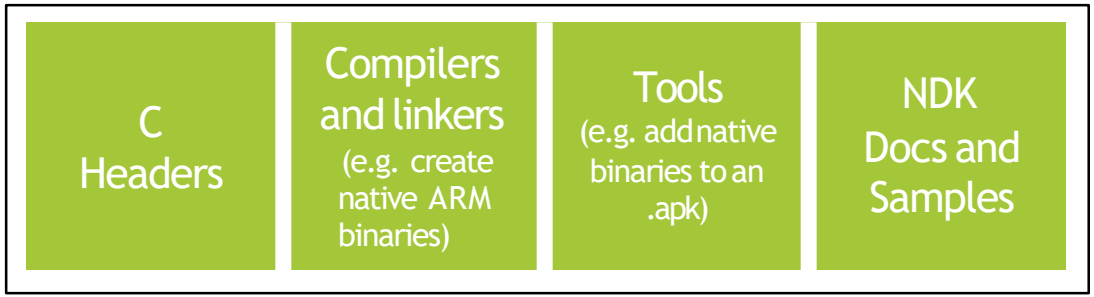
What is the AndroidSDK?

- The *Android SDK* contains the APIs and tools needed to create and run a native Android app



What is the Android NDK?

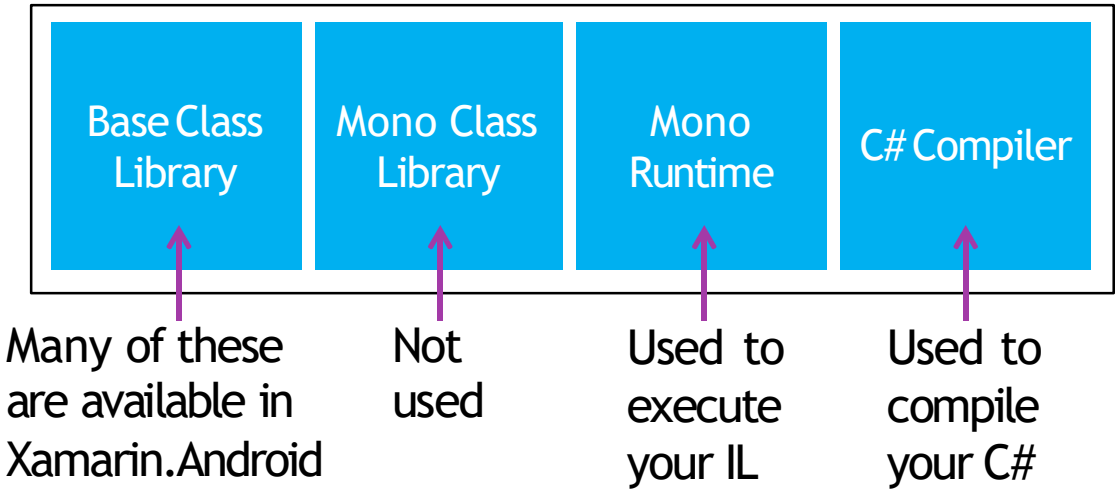
- The *Android NDK* is a collection of code and tools that let you write part of your native Android app in a language like C and C++



Writing part of your app in C/C++ is rare. It will increase complexity but may not increase performance. It can be useful in games or to reuse an existing C/C++ codebase.

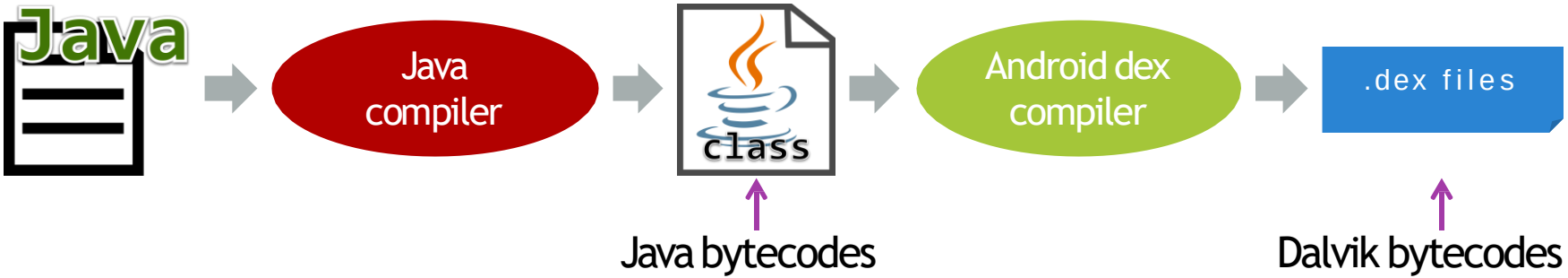
What is Mono?

- Mono is an open-source implementation of the .NET Framework; several parts are used in Xamarin.Android development



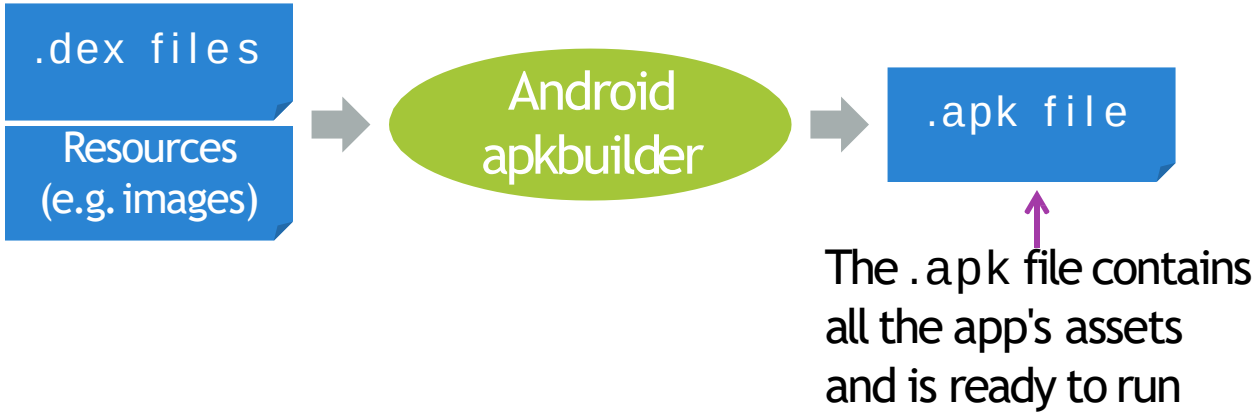
Native compilation

- Java source is compiled into *Dalvik bytecodes* for deployment (bytecodes are analogous to .NET Intermediate Language)



Native packaging

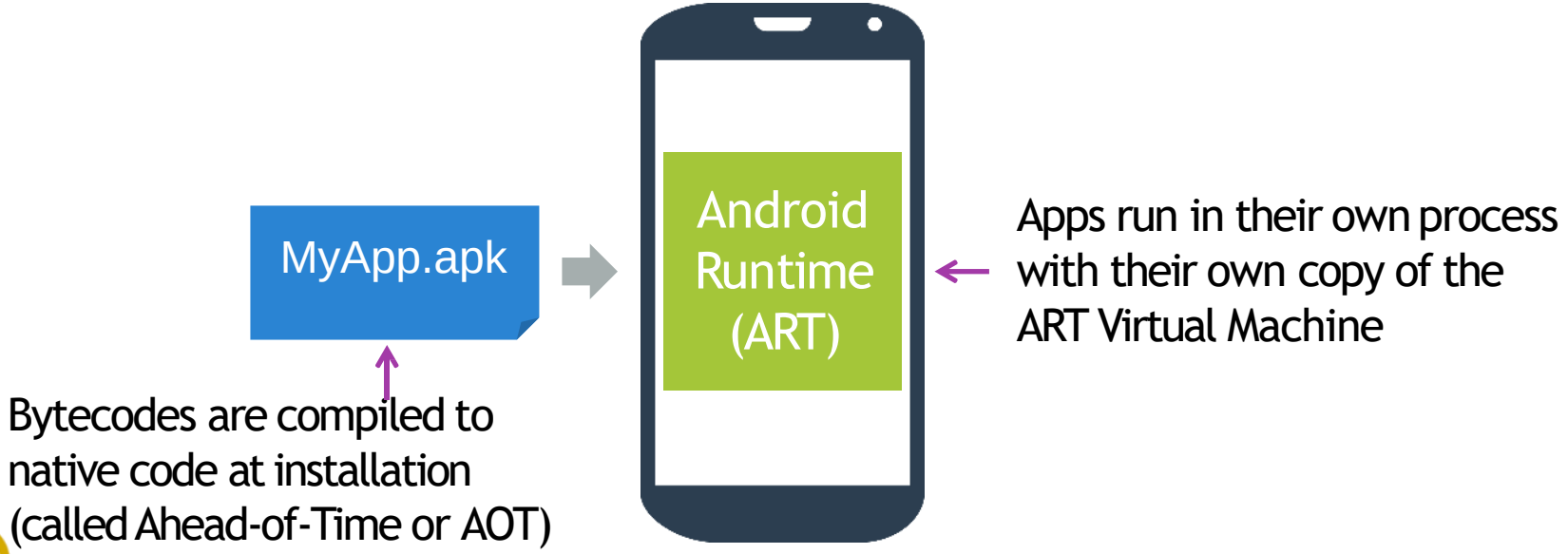
- An app's bytecodes, images, data files, etc. are combined into an Application Package (`.apk` file) for deployment



For upload to the Play store, there are two more steps that are not shown: signing with jarsigner and optimizing the layout of the file with zipalign.

Native execution

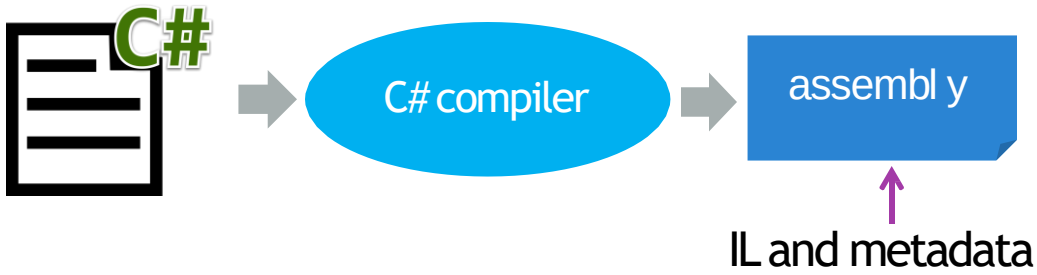
- The Android Runtime (ART) is the execution engine for Android apps



Android versions before 5.0 used the Dalvik VM which did runtime bytecode translation.

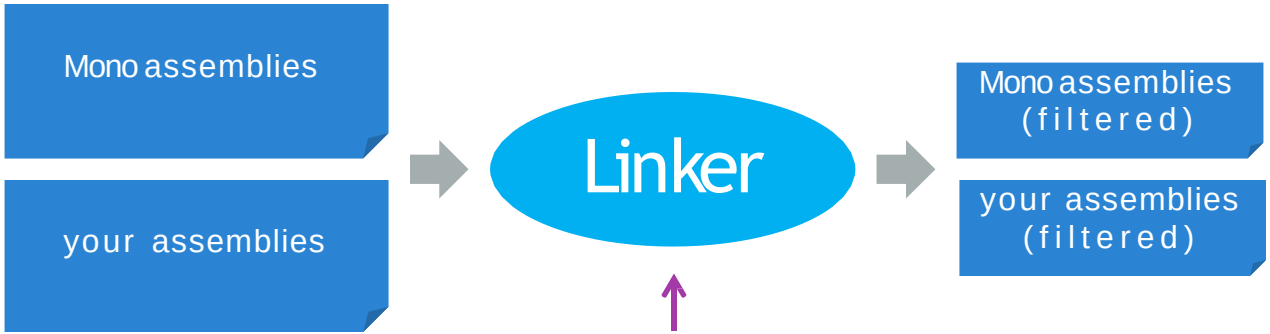
Xamarin.Android compilation

- C# code in Xamarin.Android apps is compiled to .NET Intermediate Language (IL)



Xamarin.Android linking

- The Xamarin.Android *linker* removes unused IL to reduce the size of your app for deployment



Determines which class members are used in your app and includes only those members in the output

Project settings and code Attributes let you control which assemblies are linked. Dynamic code should not use the linker (e.g. members accessed via reflection).

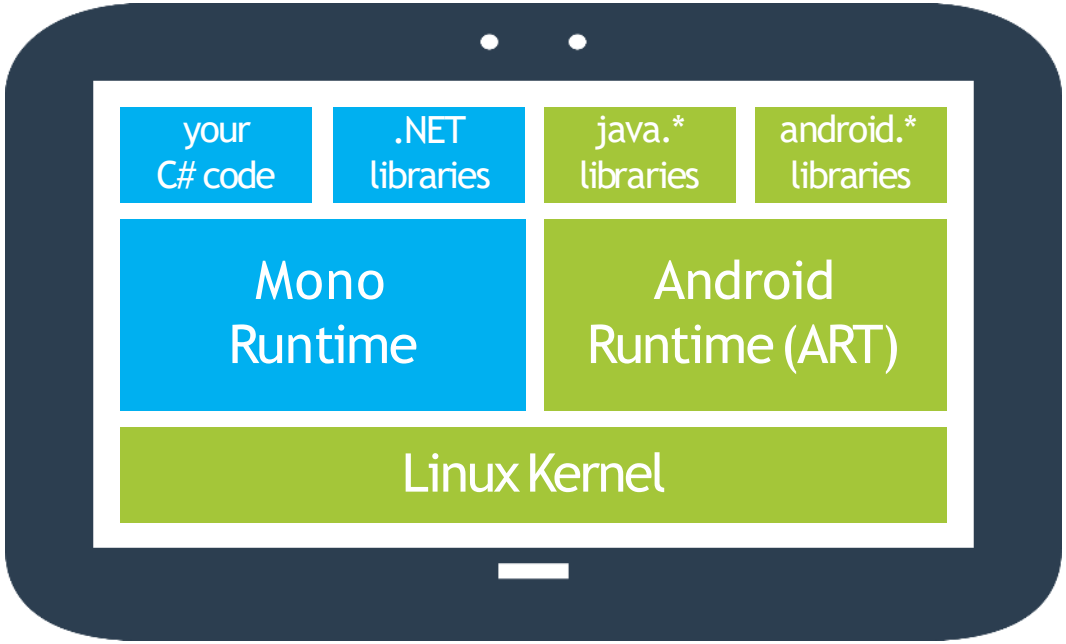
Xamarin.Android and the Mono VM

- Xamarin.Android apps have the Mono Runtime packaged in their .apk file because it is needed to execute IL

The Xamarin build tools add the Mono VM to your application package

Xamarin.Android execution

- Mono and ART VMs run side-by-side to execute a Xamarin.Android app

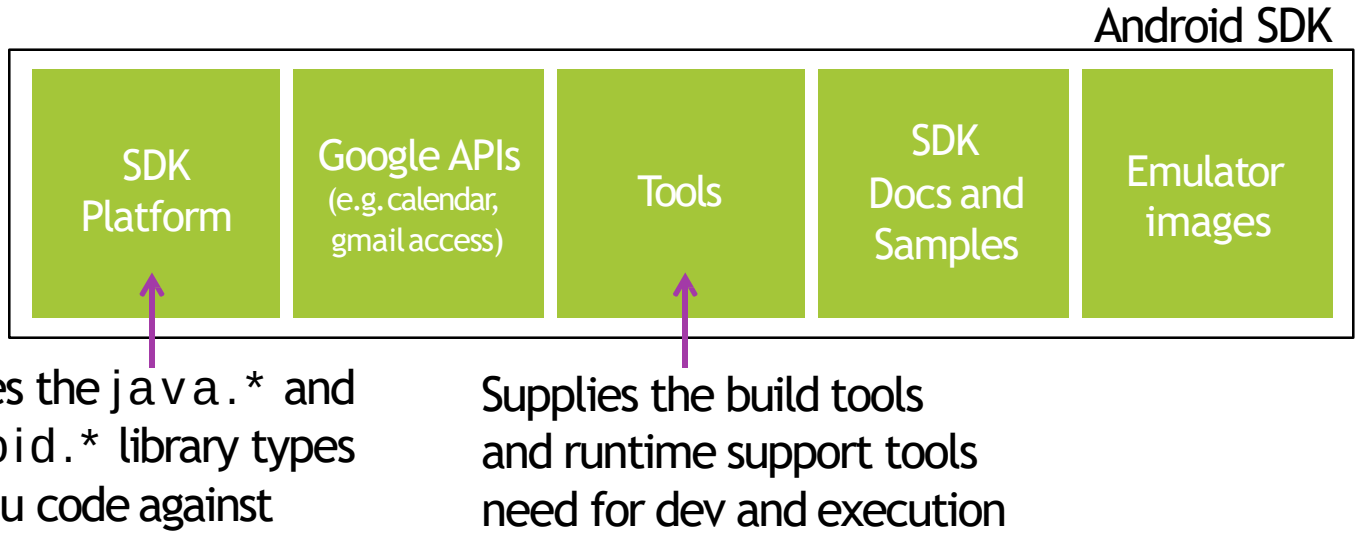


Xamarin.Android installation

- The *Xamarin unified installer* (http://xamarin.com/download) loads nearly everything you need to develop and run Xamarin.Android apps

Android SDK updates

- You need to manually update your Android SDK Platform and Tools so you can build against the latest versions of Android



Android versions

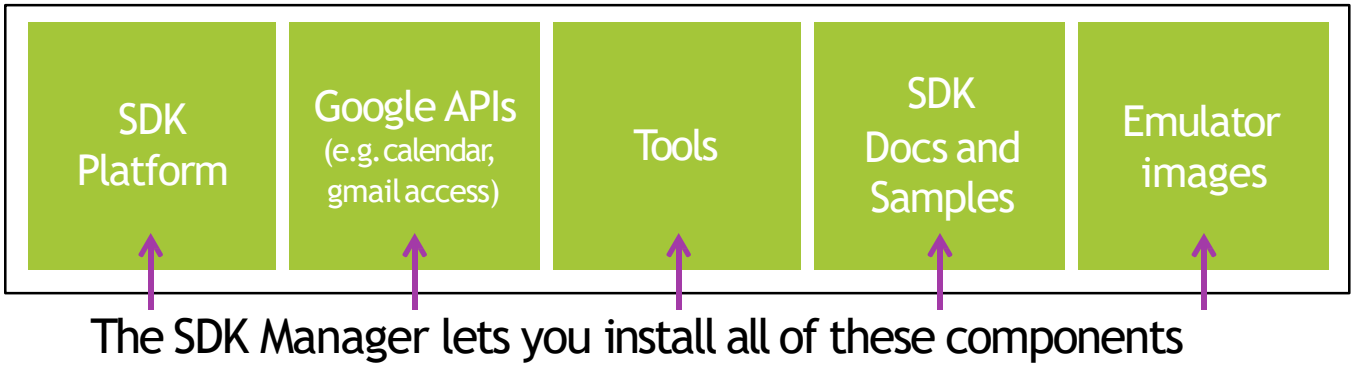
- Android versions are identified via a code name and two numbers

Code Name	Version	API Level
Marshmallow	6.0	23
Lollipop	5.1	22
Lollipop	5.0	21
Kit Kat (watch)	4.4W	20
Kit Kat	4.4	19
Jelly Bean	4.3	18
Jelly Bean	4.2.2	17
Jelly Bean	4.2	17
…	…	…

Level identifies the combination of libraries, manifest elements, permissions, etc. that you code against as a developer

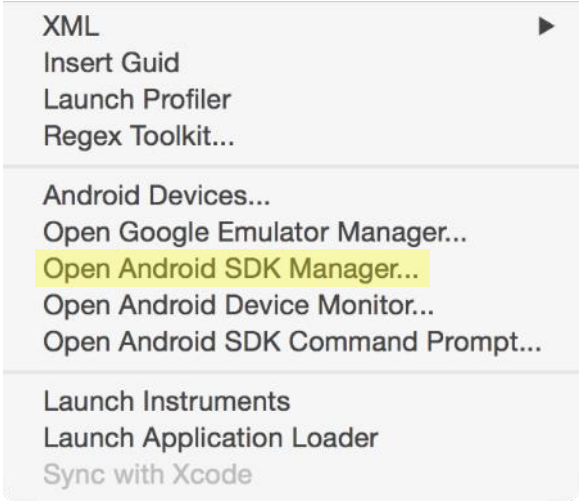
What is Android SDK Manager?

- The *Android SDK Manager* is a tool from Google that lets you install new (and old) versions of the Android SDK

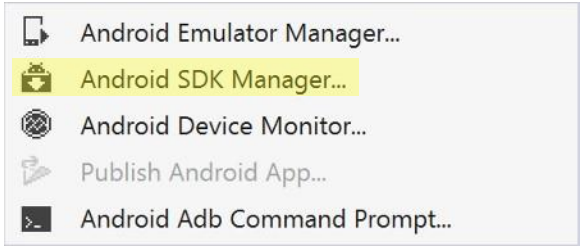


How to launchAndroid SDK Manager

- Xamarin Studio and Visual Studio menu entries launch the Android SDK Manager



Xamarin Studio Tools menu

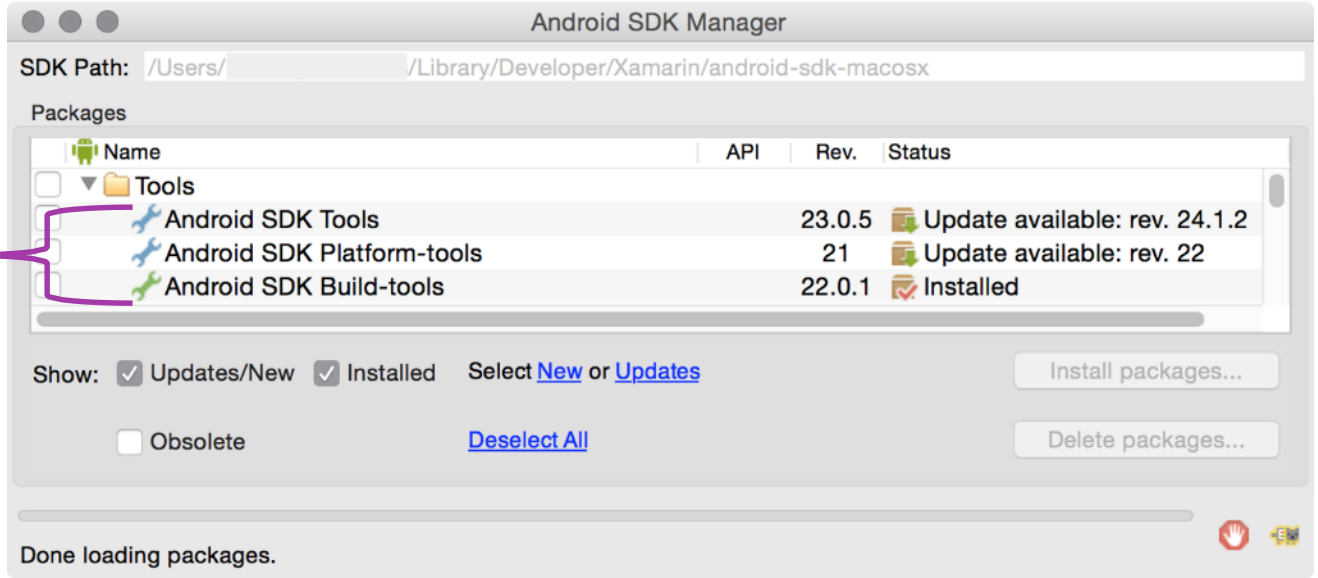


Visual Studio Tools > Android menu

Updating tools

- Android splits the SDK tools into three parts that can be updated separately; you should keep all three categories up-to-date

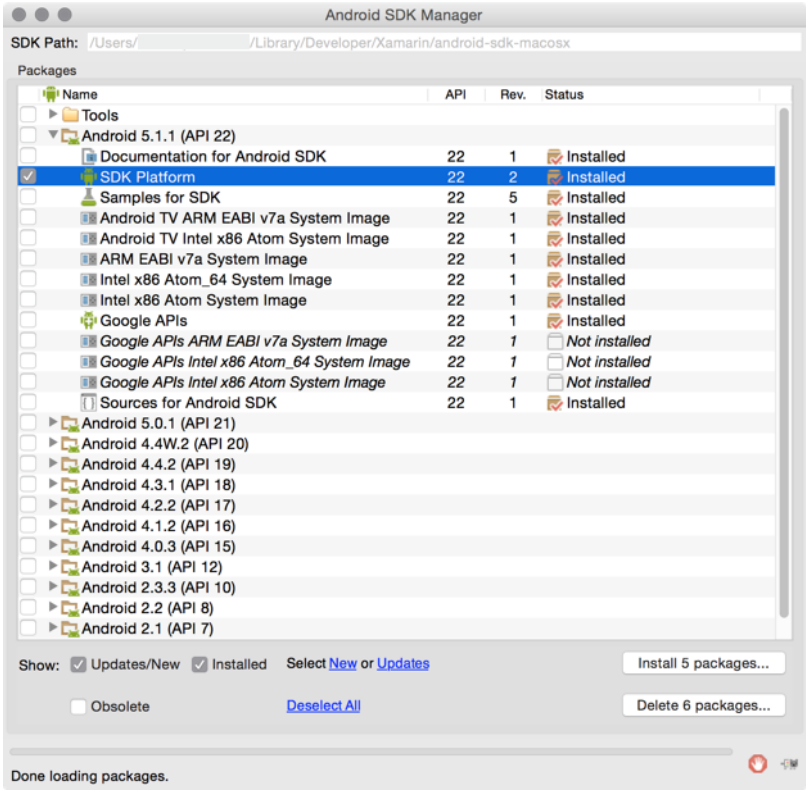
Update all of these. The SDK manager tells you when updates are ready.



Updating platform versions

- Use the SDK Manager to install the platform versions you would like to compile against

Install the SDK Platform for the versions you need

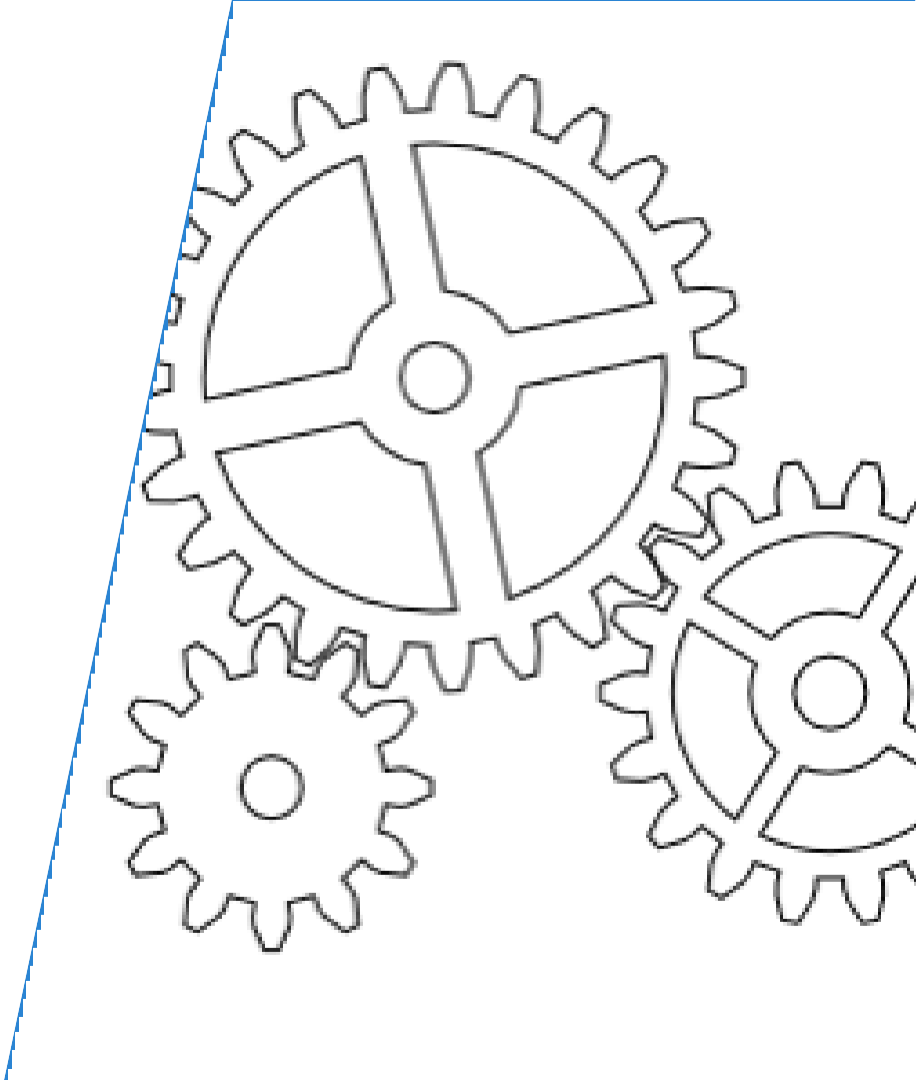


Group Exercise

Update Tools and SDK Platform

Summary

1. Review native Android development
2. Understand the Xamarin.Android development process
3. Update your Android Tools
4. Update your Android Platform SDK



Next Steps

- This class has shown you how to build a Xamarin.Android app with one Activity
- In AND102 we will look at how to create multiple Activities and get them to work together by passing arguments and retrieving results

Thank You!