Beginner's Workshop

Paul Lammertsma
CTO, Pixplicity

.Pixplicity®

Agenda

10:00	Introduction
10:30	Lesson #1
11:30	Lesson #2
12:30	Lunch
13:30	Lesson #3
14:30	Lesson #4
15:30	Lesson #5
16:30	Homework

Contents

Contents

Assumed knowledge

- Java language and syntax
- Basic Object Oriented (OO) programming
- Basic event driven programming
- Writing tests
- Working with more complex concepts
  - Services, Broadcast Receivers, Databases, etc.
  - Performing background tasks

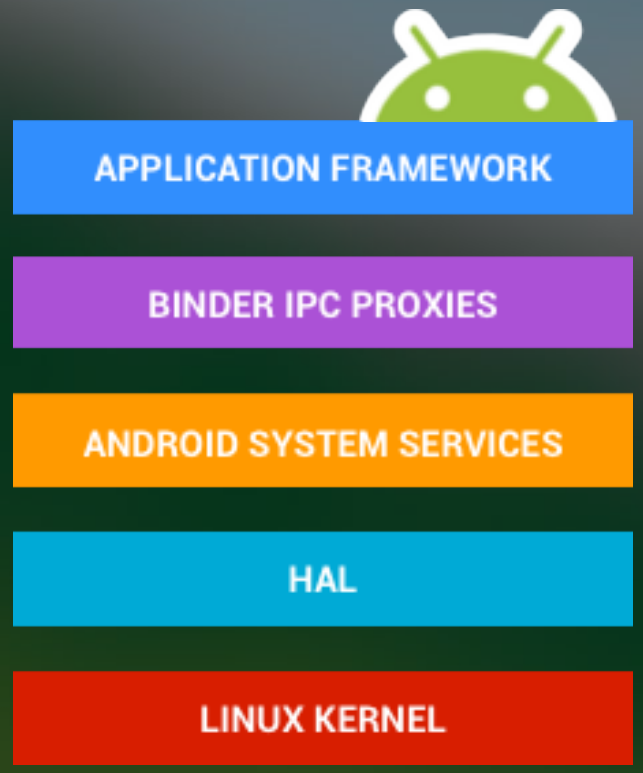
Homework!

Why Android?

- Truly open! (...truly open?)
- Component based
- Tons of built-in services
- Automatic management of application life cycle
- High-quality graphics and sounds
- Portability across a wide range of hardware

Android System

Android System



Your application interacts with the Application Framework.
These services in turn access the underlying hardware, using the Hardware Abstraction Layer, a standard interface for hardware vendors.
The hardware is ultimately controlled by device drivers based on the Linux kernel.

Some of those frameworks are Inter-Process Communication for hardware boundaries to interact with Android system services.
There are *media* services, such as Camera and MediaPlayer, and ***system*** services, such as ActivityManager and NotificationManager.

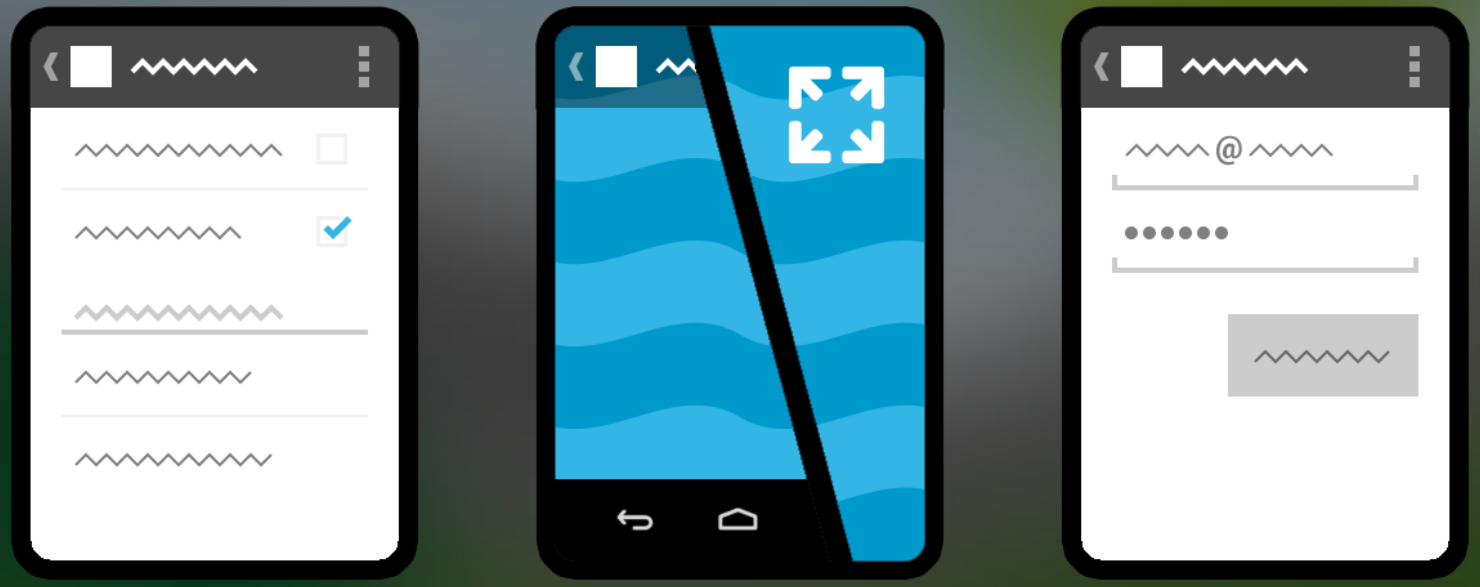
Android System

Building Blocks

- Activities, Views
- Intents
- Services
- Content Providers
- Broadcast Receivers
- Resources
- Safe and Secure

Activity

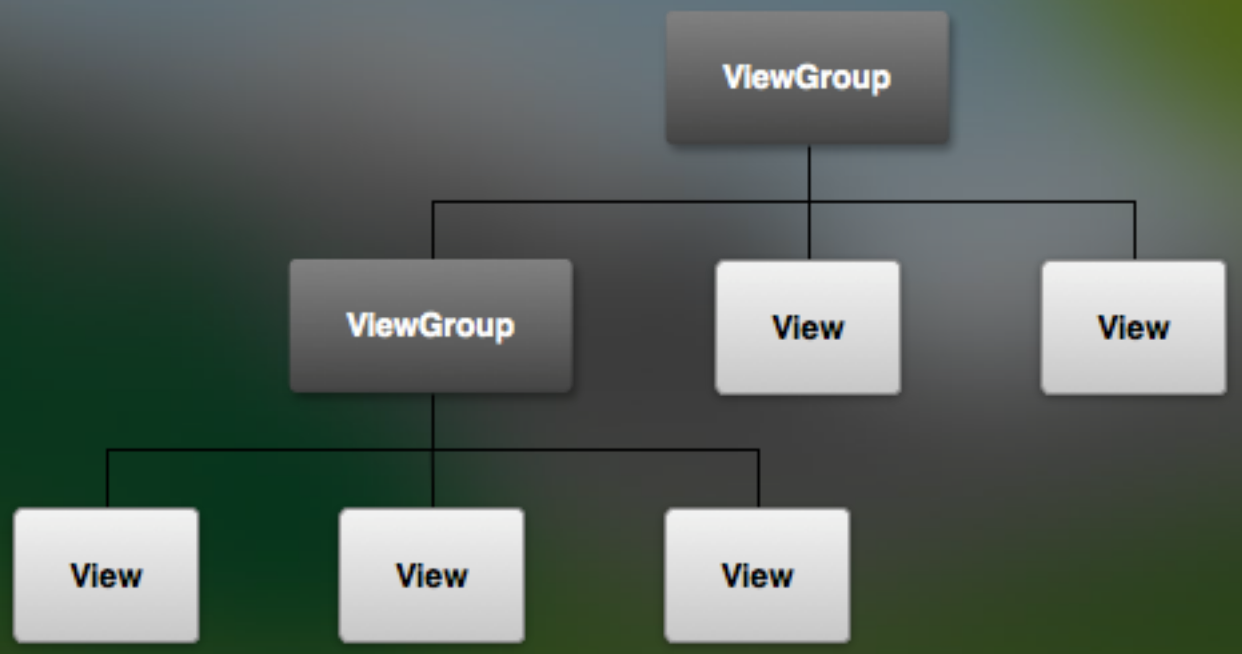
- Single screen with a user interface



- Excluding system interfaces (system bar, navigation)

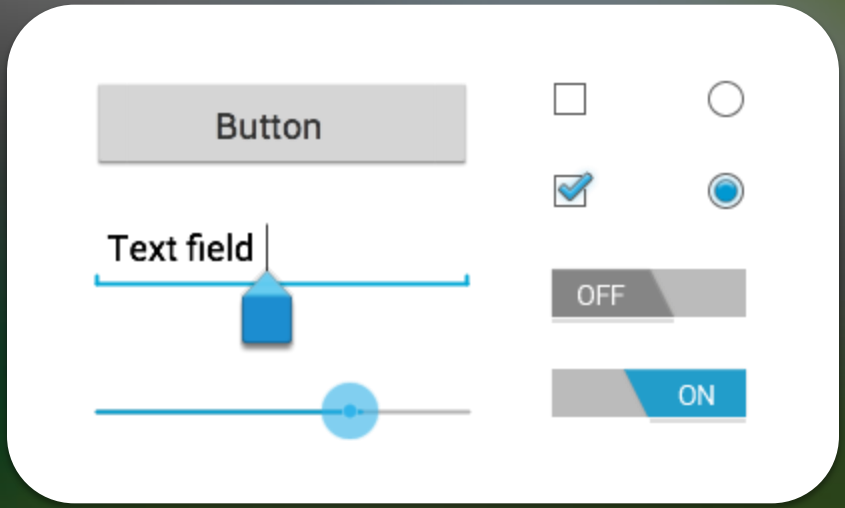
User Interface

- Hierarchy of View and ViewGroup objects



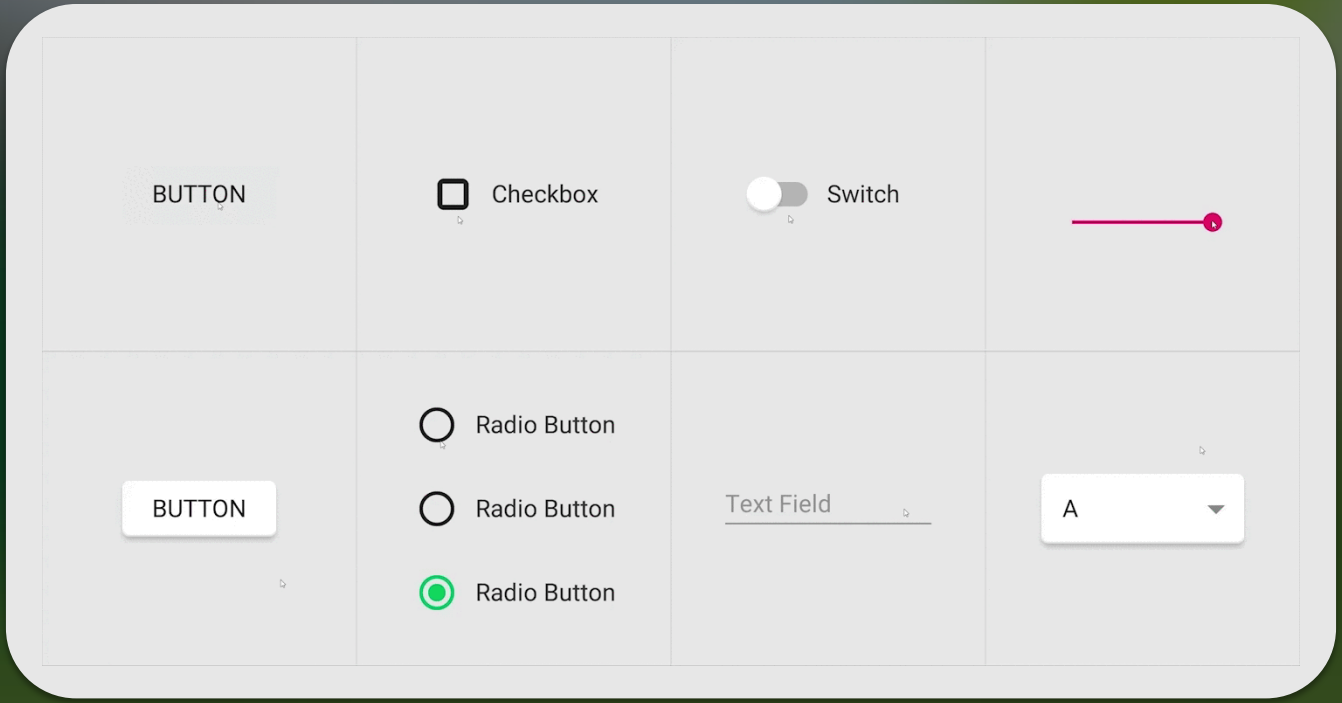
Views / “widgets”

- User interface components
  - Platform native vs. Material/AppCompat



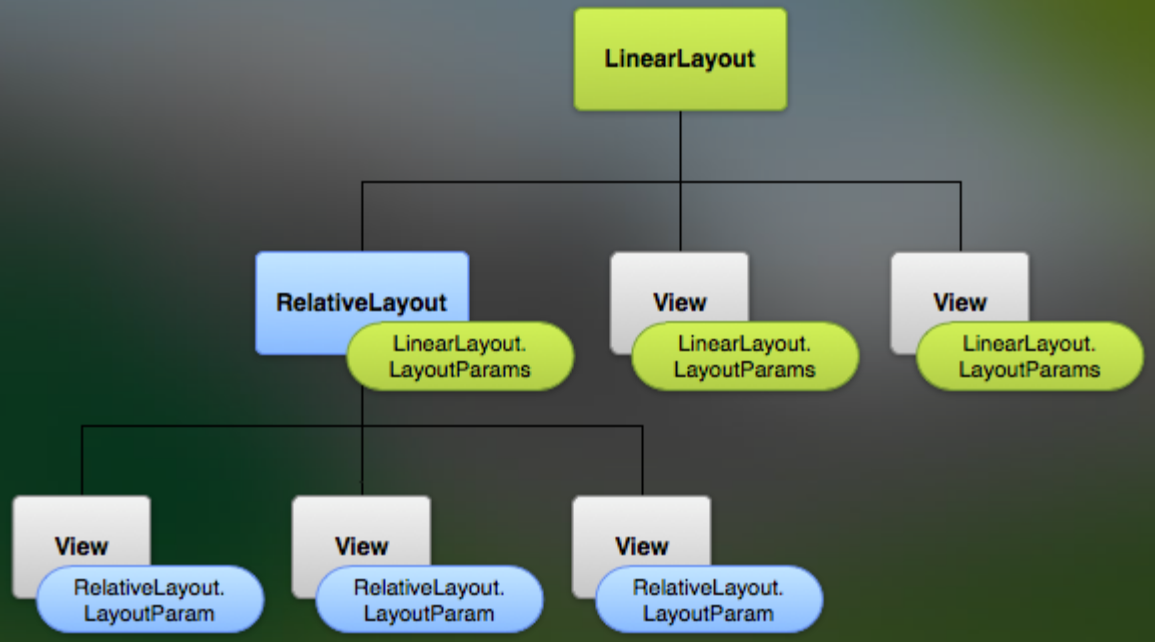
Views / “widgets”

- User interface components
  - Platform native vs. Material/AppCompat



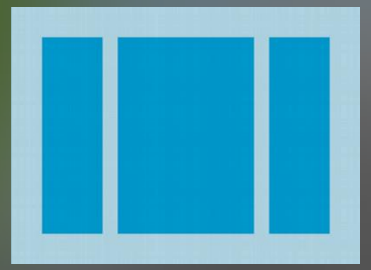
Layouts

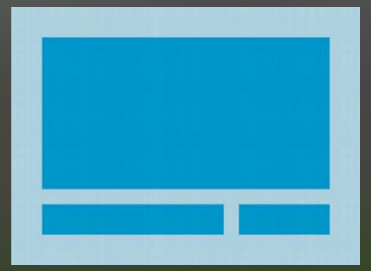
- Implementation of a user interface

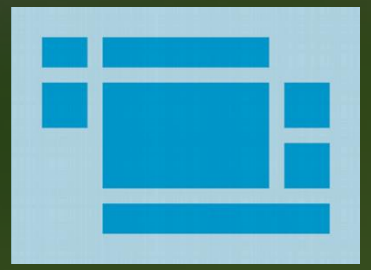


Layouts

- LinearLayout
  - horizontal / vertical layout
- RelativeLayout
  - below, leftOf, alignTop to other views or parent
- GridLayout
  - columns / rows
- FrameLayout
  - single frame / stacked items

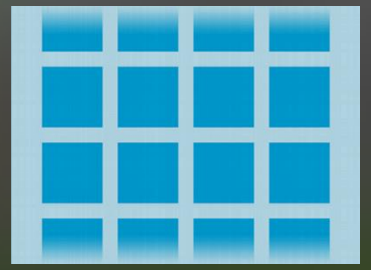






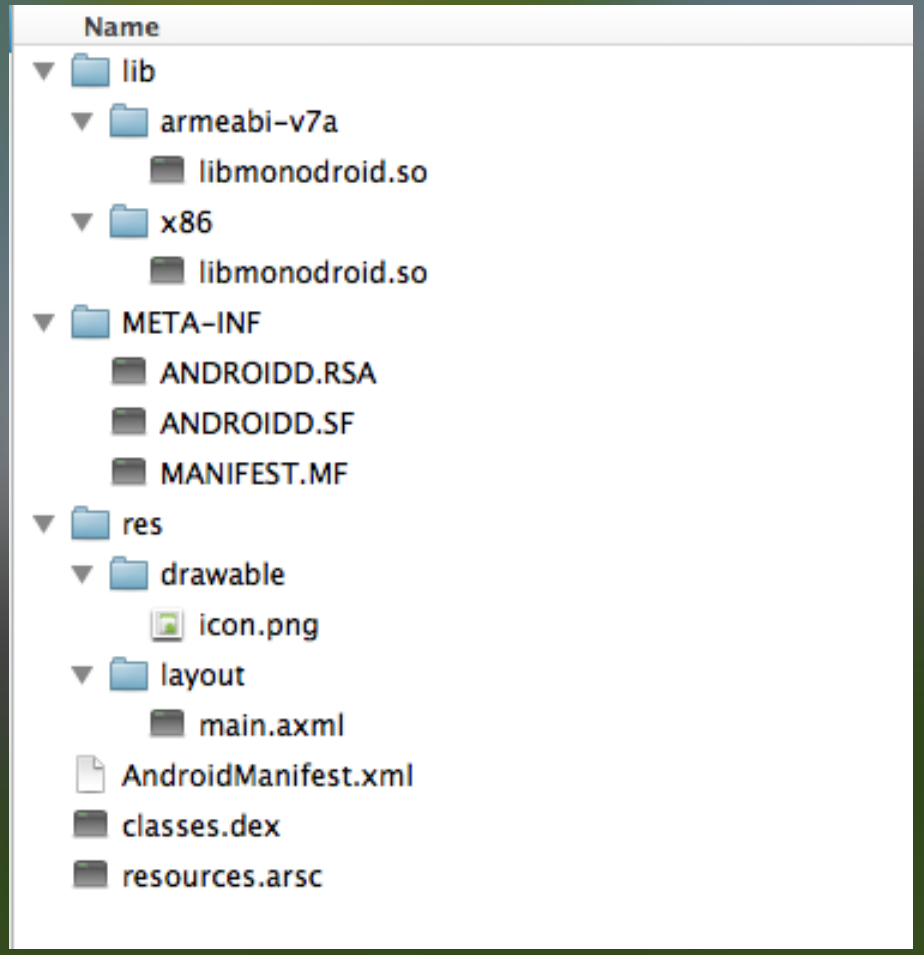
Layouts with Adapters

- Binds data to a layout dynamically using an Adapter
- ListView
  - scrolling single column list
- GridView
  - scrolling grid of columns and rows



Contents of an APK

- Packaged as APK-file (ZIP file)
- Manifest
- Signing information
- Compiled code & resources
- Binary resources and assets



Contents of an APK

Native libraries by CPU architecture

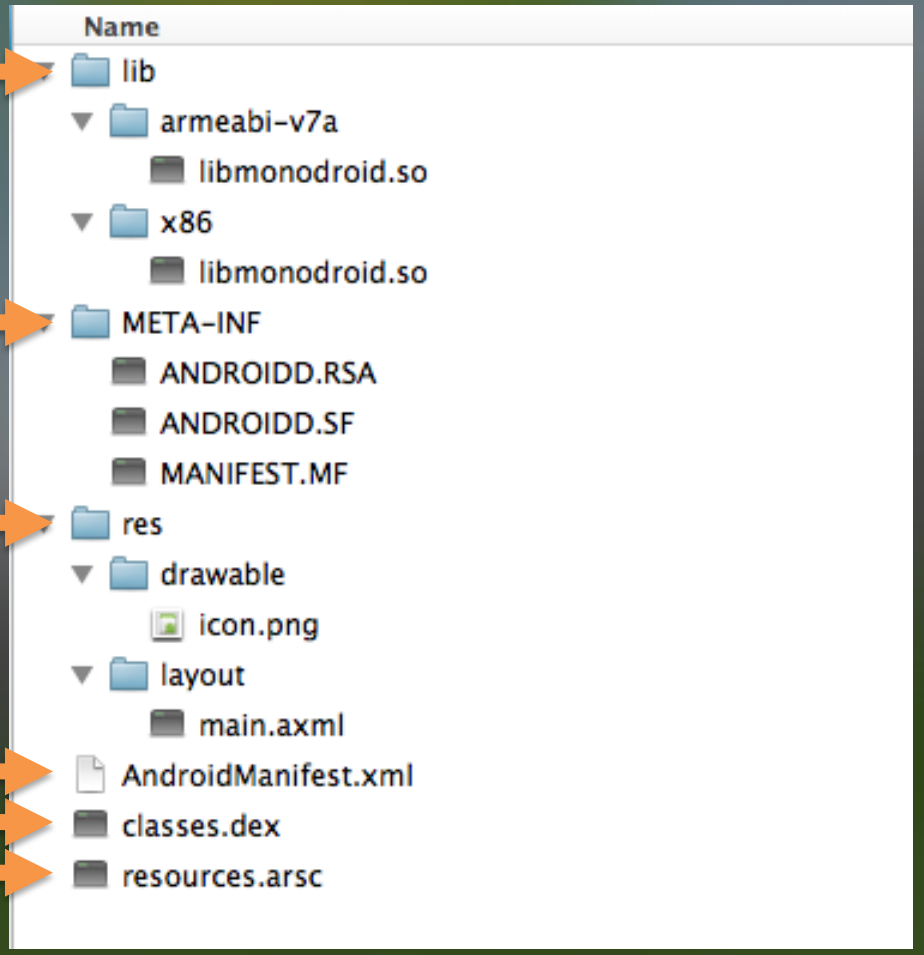
Certificate and SHA-1 file digests

Uncompiled resource files

Android's manifest that describes app

Java classes compiled into dex format

Pre-compiled resources (e.g. binary XML)



AndroidManifest.xml

- Application information (version number, version code)
- Minimum and target SDK Version
- App permissions and device constraints
- Activities (with intent filters) `<activity>`
- Services `<service>`
- Broadcast Receivers `<receiver>`
- Content Providers `<provider>`

Course Guide

Make sure you have the course guide open:

https://pixplicity.gitbooks.io/android-beginners-workshop/

Online | PDF | ePub | Mobi

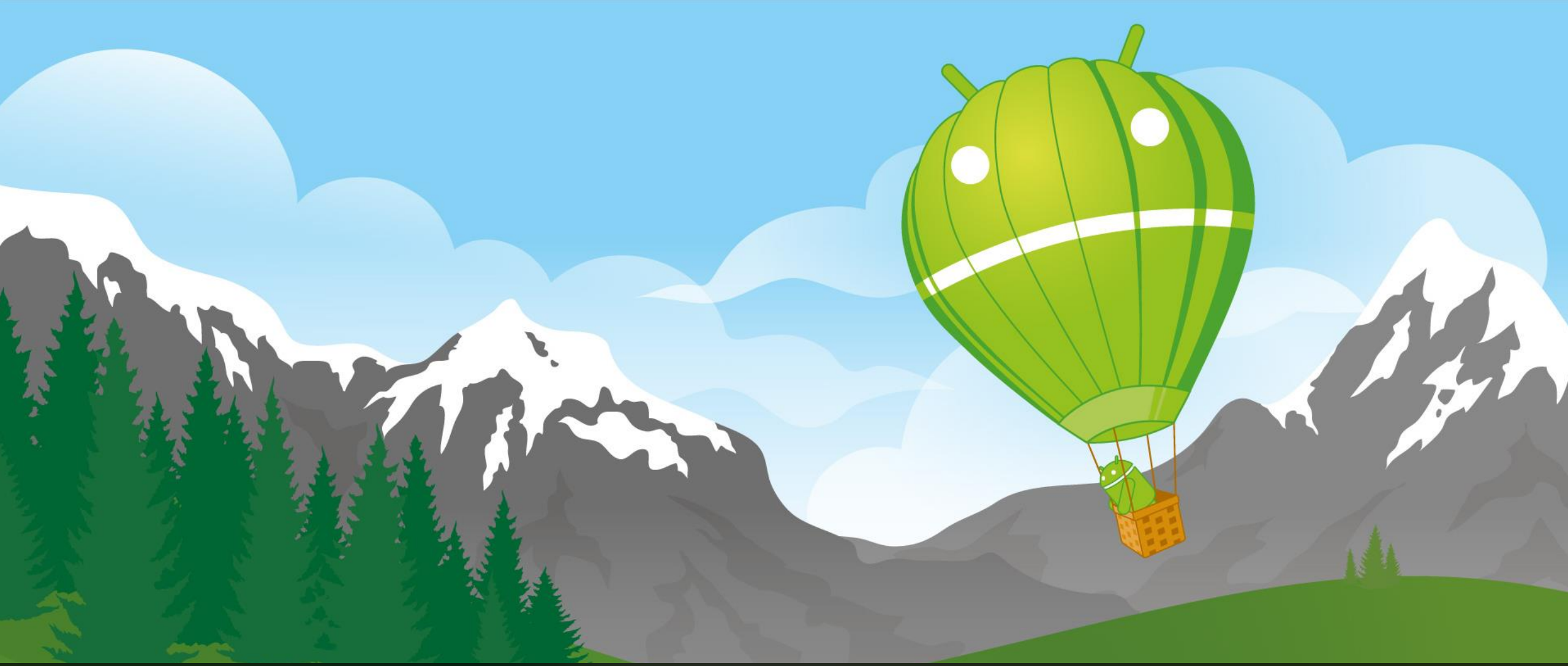
Legend

Course Guide:

...

Rules

- Ask questions
- Talk to others
- Work together
- Take breaks



Lesson #1

Android Studio & projects

Pixplicity®

Android Studio

Getting Started

Course Guide:
Getting Started

1. Java SDK
2. Android Studio
3. Git client (optional)
4. Ensure everything's up-to-date
5. Prepare a device

Lesson 1

Course Guide:
Lesson 1

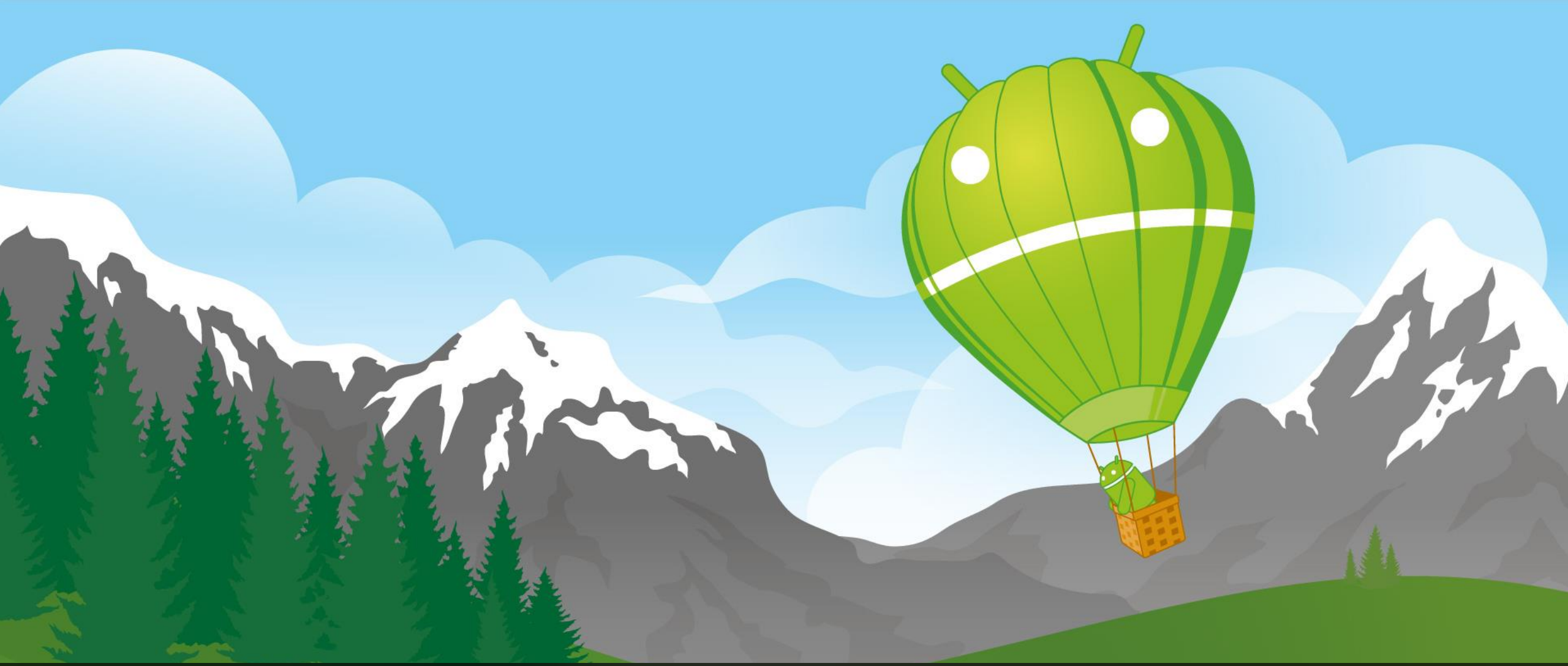
1. Importing an Android project in Android Studio
2. The Android Project structure
   - Gradle build files
   - AndroidManifest.xml
   - MainActivity
   - Resources & layouts

Lesson 1

- New Project Wizard
- Import sample
- SDK Manager
- Run on emulator
- Gradle files
- Building with `gradlew`

Lesson 1

- AndroidManifest.xml
  - Package, permissions, application, activity
- MainActivity
  - `onCreate()`
  - `setContentView()`
  - Referenced layout
- Resource folders
  - String resources



Lesson #2

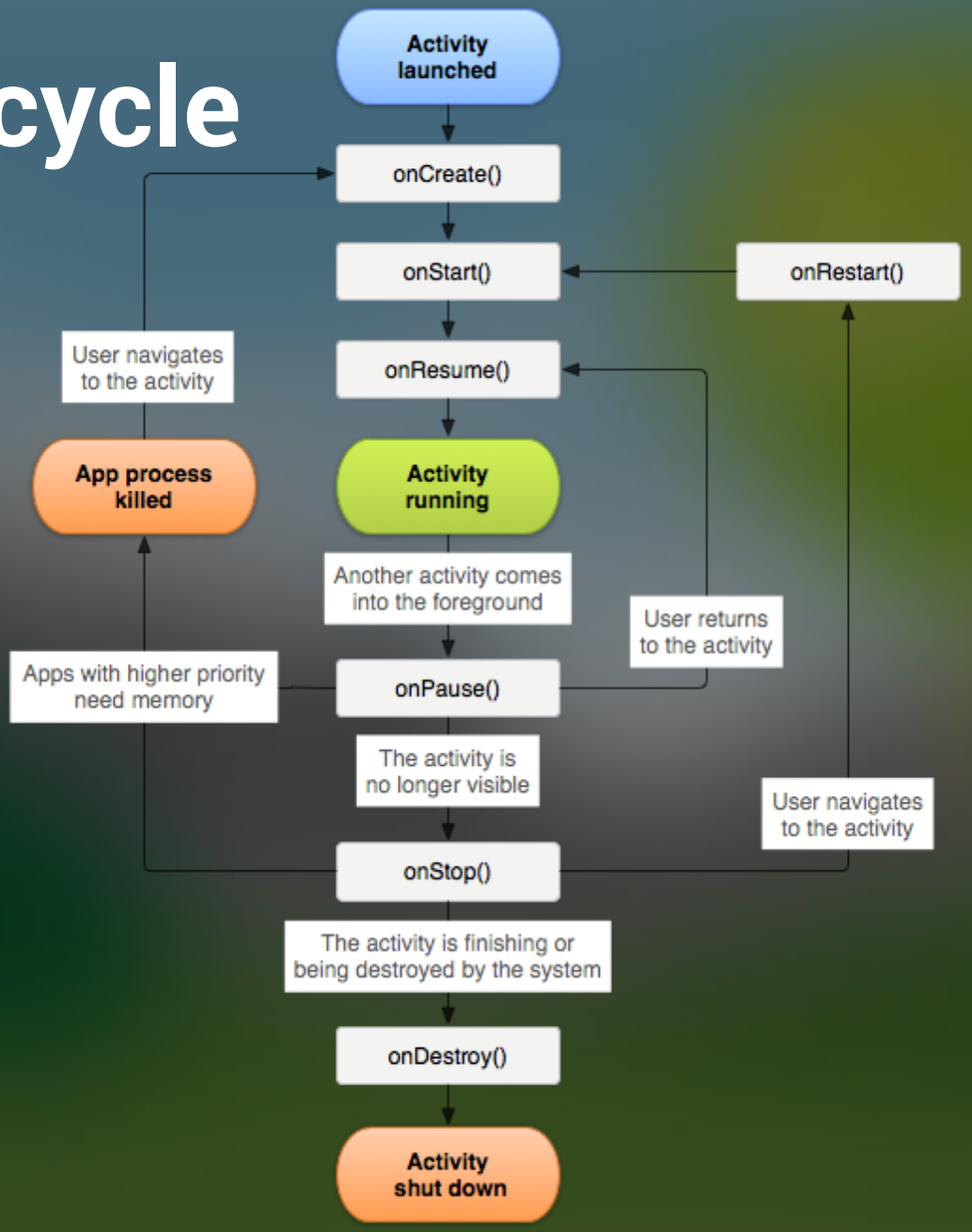
Activities & Views

.Pixplicity®

Activities

- Invoked by Intents
- Have a lifecycle
  - **Resumed** in the foreground & has user focus (“running”)
  - **Paused** visible, but another activity has focus
  - **Stopped** completely obscured in the background
- Can be killed to free up memory

Activity lifecycle



Events & Listeners

```
button.setOnClickListener(

    new View.OnClickListener() {
        public void onClick(View v) {
            //…
        }
    }

);
```

Lesson 2

Course Guide:
Lesson 2

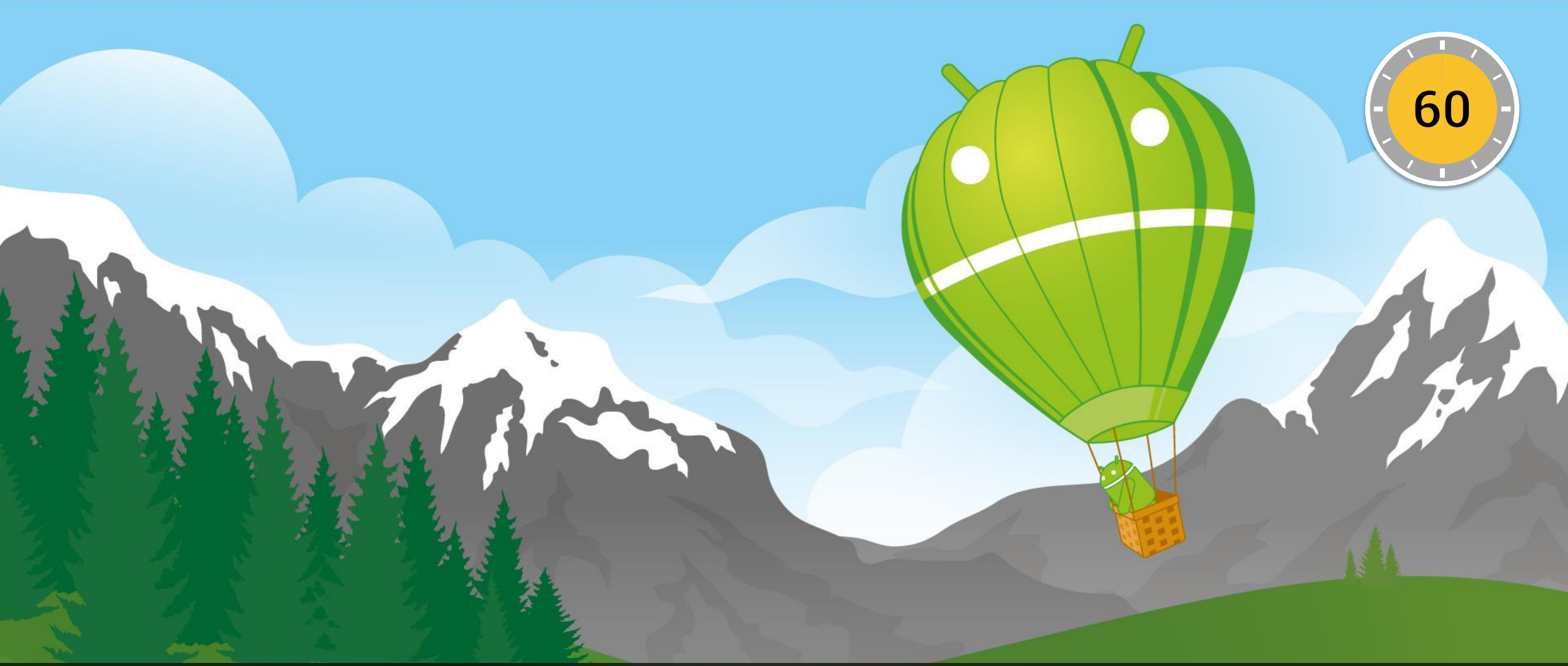
1. Activity launch intent filter
2. The activity life cycle
3. Logcat
4. Click listeners

Lesson 2

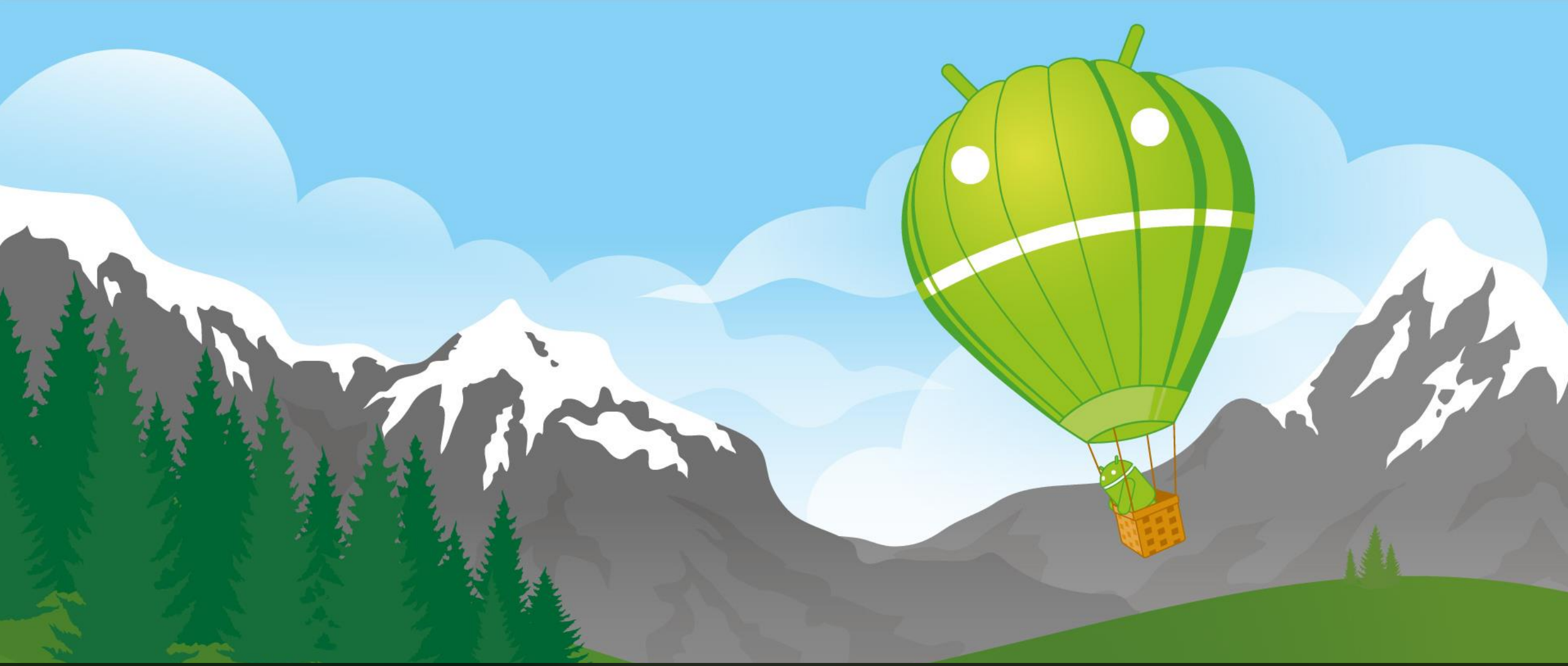
- AndroidManifest.xml
  - Activity's launch intent filter
- MainActivity
  - `onCreate()`, `onResume()`, `onPause()`
  - Annotations
- Logcat

Lesson 2

- activity_main.xml
  - Design, Text and Preview
  - Add “Person Name” TextView
  - Add Button
  - android:id
- Referencing layouts from code
- Registering an OnClickListener



Lunch!



Lesson #3

Intents, Tasks & Activity Back Stack

Pixplicity®

Intents

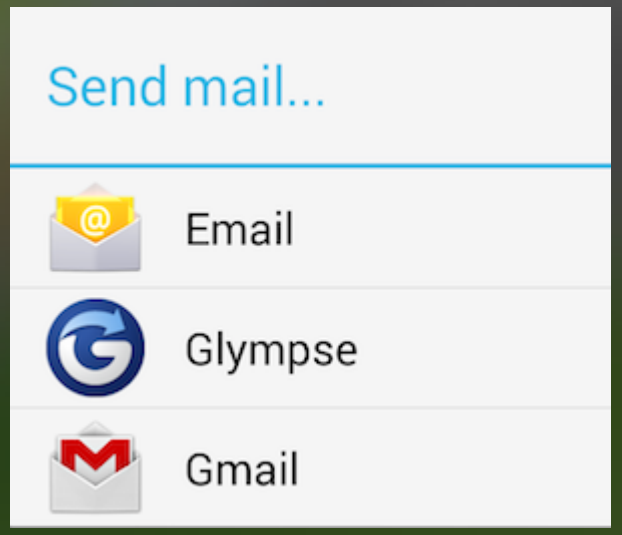
- Navigate to other Activities
- Not just your own Activities
- Pass information along
- Receive information back
- Explicit vs. implicit

Implicit intents

- By Action
  - Intent.ACTION_SEND
  - android.media.action.IMAGE_CAPTURE
- By Uri
  - https://plus.google.com/+PaulLammertsma
- By Mime type
  - `"text/plain"`
- A combination

Implicit intents

```
Intent intent = new Intent(
        Intent.ACTION_SENDTO,
        Uri.parse("mailto:paul@pixplicity.com"));
intent.putExtra(Intent.EXTRA_SUBJECT,
        "Hi there!");
intent.putExtra(Intent.EXTRA_TEXT,
        "I learned a lot from the workshop!");
startActivity(Intent.createChooser(intent,
        "Send mail…"));
```

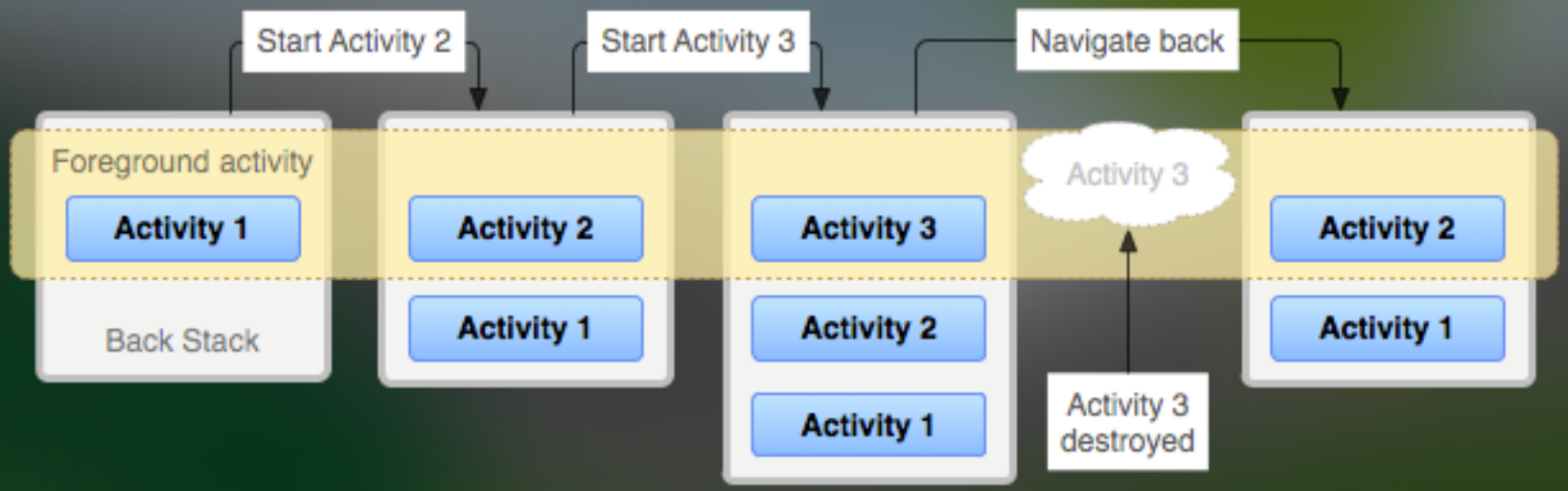


Explicit intents

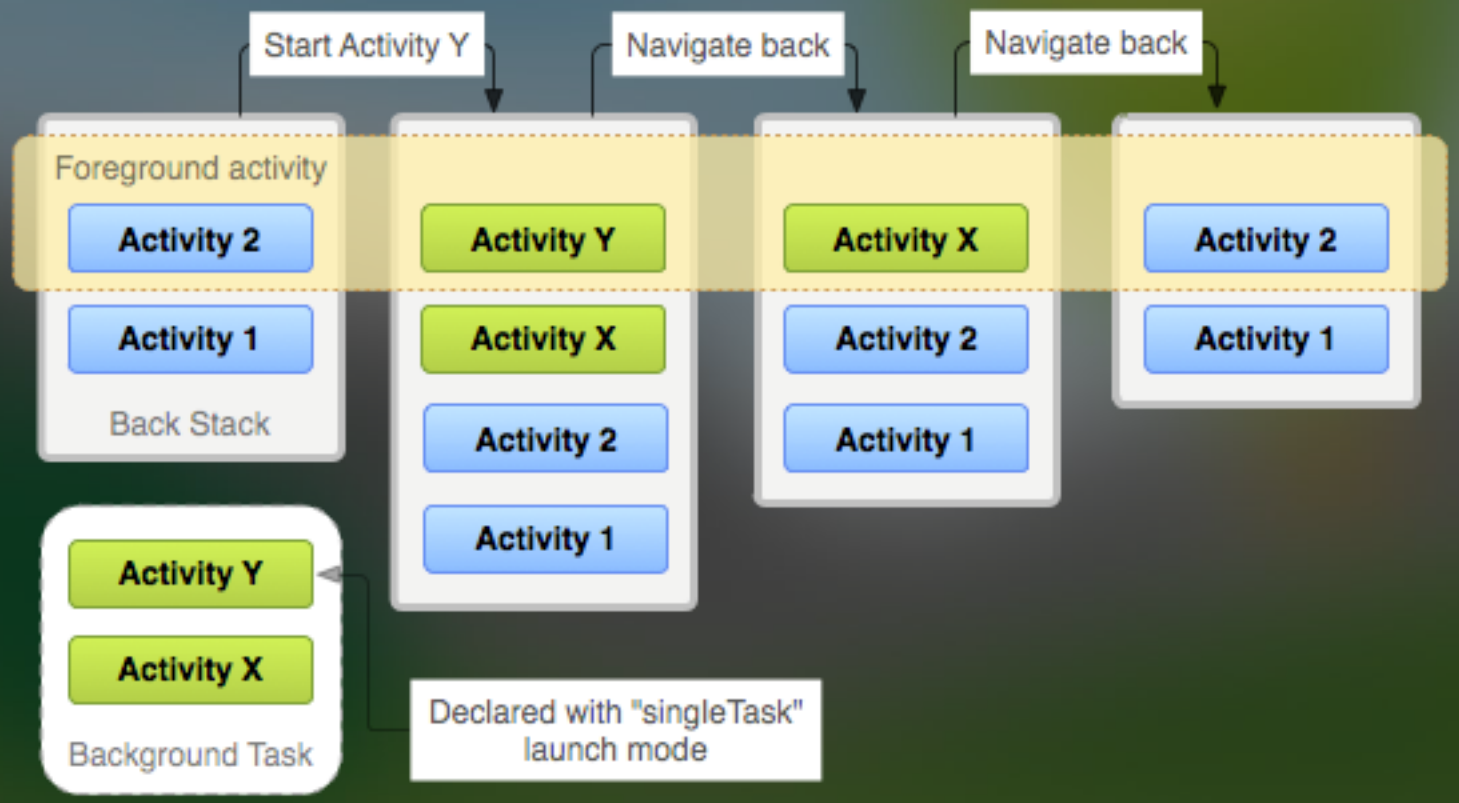
```
Intent intent = new Intent(
        ActivityA.this,
        ActivityB.class);

// (add data to the intent)

startActivity(intent);
```

Do *not* create an instance;
Android will take care of this for you

Tasks & Activity Back Stack



Tasks & Activity Back Stack



Lesson 3

Course Guide:
Lesson 3

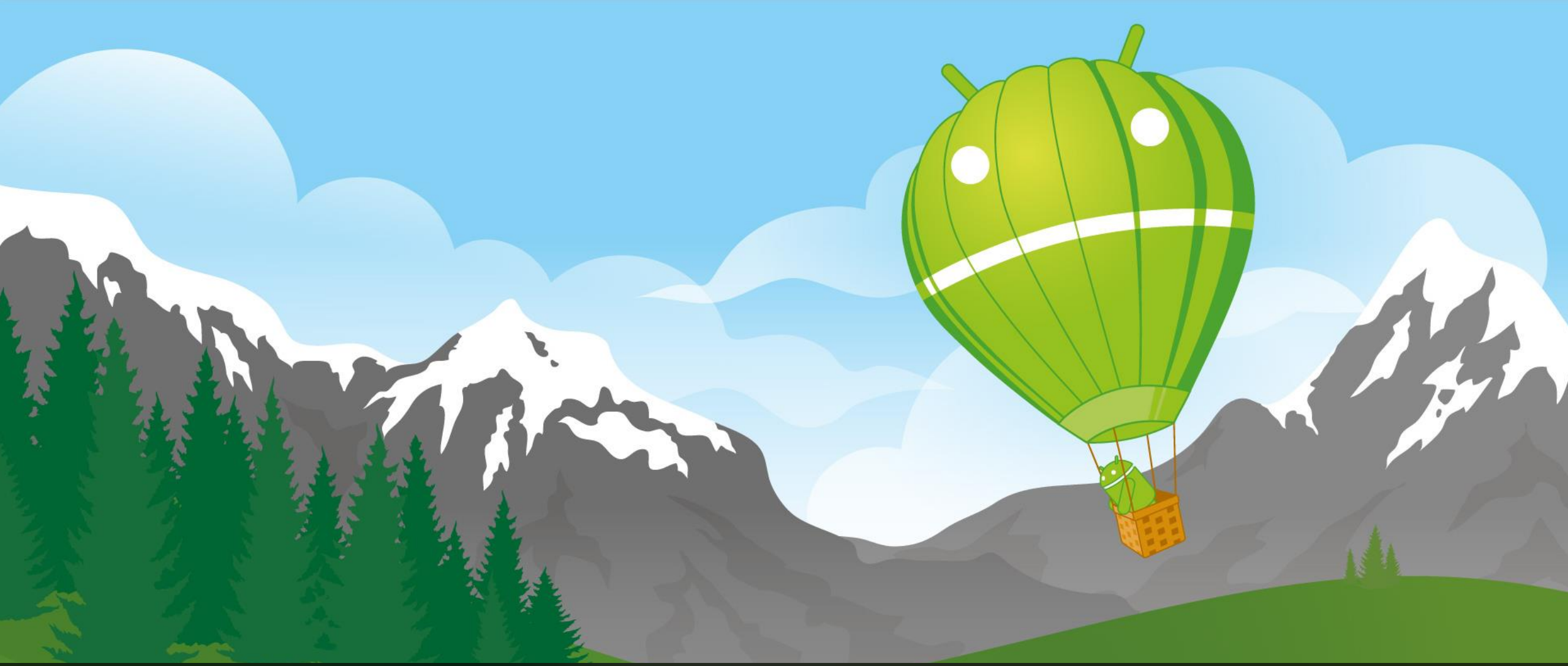
1. Explicit intents
2. Implicit intents
3. The activity back stack
4. Exchanging data between activities

Lesson 3

- Debugging crashes
- Breakpoint in SecondActivity.onCreate()
- Retreive email and subject from EditTexts
- Create explicit intent to another Activity
- Pass along data in Intent extras
- Purpose of onSaveInstanceState()

Lesson 3

- Implicit intent to send an email
- Inspecting the activity stack using `adb shell dumpsys activity`



Lesson #4

ListViews & Adapters

Pixplicity®

ListViews & Adapters

- **ListView**
  Component to display a (long) list of scrollable things
- **Adapter**
  Pulls content from a source
  - E.g. an array or database query
  - Each item shows as a View in the list

RecyclerView

Homework!

- More advanced and flexible than
  - For instance, animations!
- Still easy to use
- From the **support library**
- Sorry! Won’t be covered in this workshop

```
compile 'com.android.support:recyclerview-v7:23.0.+'
```

Adapter contract

```
int getCount();
```

How many items are in the data set represented by this Adapter

```
Object getItem(int position);
```

Get the data item associated with the specified position in the data set

```
long getItemId(int position);
```

Get the row id associated with the specified position in the list

```
View getView(int position, View convertView,
             ViewGroup parent);
```

Return a view for the given position, given a recycled convertView

Lesson 4

Course Guide:
Lesson 4

1. ListViews
2. RecyclerViews
3. Adapters

Lesson 4

- ListView & RecyclerView are efficient for large data sets
- Grasping the concept of Adapters
- Array resources
- Creating ArrayAdapters
- Hierarchy of types
- ListView & ListActivity
  - onListItemClicked()

04.01 Lesson 4

```
public class ListViewInLayoutActivity extends Activity {

    private ListView listView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_listview_in_layout);
        listView = (ListView) findViewById(R.id.listview);
        String[] entries = getResources().getStringArray(R.array.animals);
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
                this, android.R.layout.simple_list_item_1, entries);
        listView.setAdapter(arrayAdapter);
    }

}
```

04.02 Lesson 4

```
public class ListViewActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String[] entries = getResources().getStringArray(R.array.animals);
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
                this, android.R.layout.simple_list_item_1, entries);
        getListView().setAdapter(arrayAdapter);
    }

}
```

Lesson 4

- Custom adapters
  - CustomAdapter extends BaseAdapter
- Android Studio’s auto-completion
- Using the LayoutInflater

04.03 Lesson 4

```
public class CustomAdapter extends BaseAdapter {

    private final Context mContext;

    private final List<Animal> mData = new ArrayList<Animal>();

    public CustomAdapter(Context context) {
        mContext = context;
        mData.add(new Animal(
            "Bear",
            "Mammal",
            R.drawable.bear_thumb,
            R.drawable.bear,
            "http://a-z-animals.com/animals/bear/"));
        …
    }
```

04.03 Lesson 4

```
@Override
public int getCount() {
    return mData.size();
}

@Override
public Object getItem(int position) {
    return mData.get(position);
}

@Override
public long getItemId(int position) {
    return position;
}
```

04.03 Lesson 4

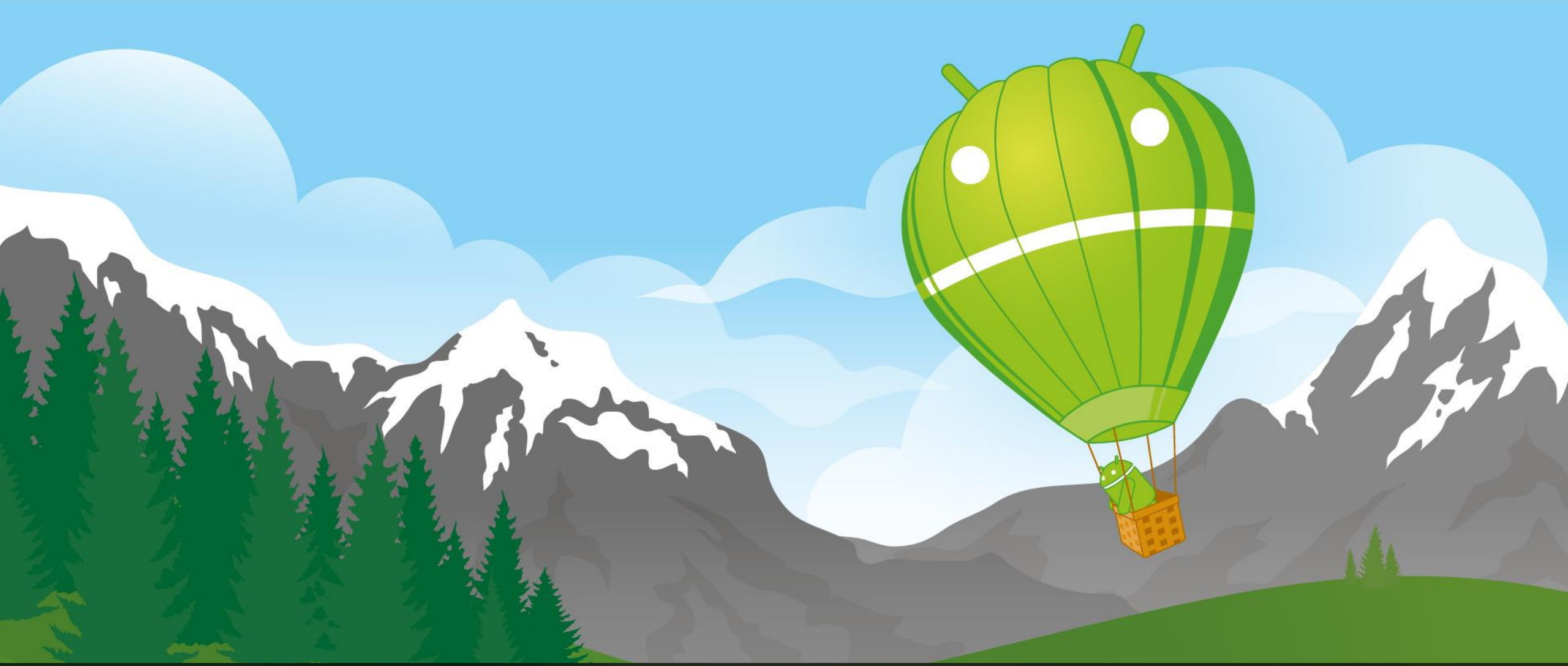
```
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Animal animal = mData.get(position);

        if (convertView == null) {
            LayoutInflater layoutInflater = (LayoutInflater)
mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            // If you have created your own custom layout you can replace it here
            convertView = layoutInflater.inflate(R.layout.custom_layout, null, false);
        }

        ImageView imageView = …
        TextView textView = …

        return convertView;
    }
```

04.03 Lesson 4

```
public class CustomAdapterActivity extends ListActivity {

    private CustomAdapter mCustomAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCustomAdapter = new CustomAdapter(this);
        getListView().setAdapter(mCustomAdapter);
    }
```

04.03 Lesson 4

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    Animal animal = (Animal) mCustomAdapter.getItem(position);
    Uri uri = Uri.parse(animal.infoUrl);
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(intent);
}
```



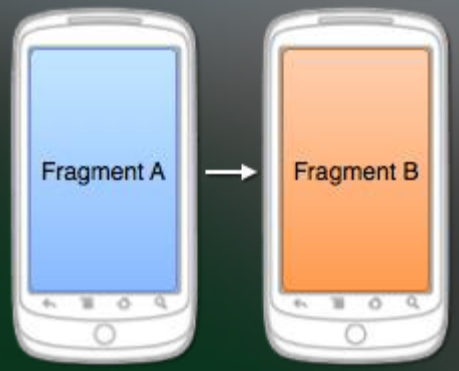
Lesson #5

All together now! An image viewing app

Pixplicity®

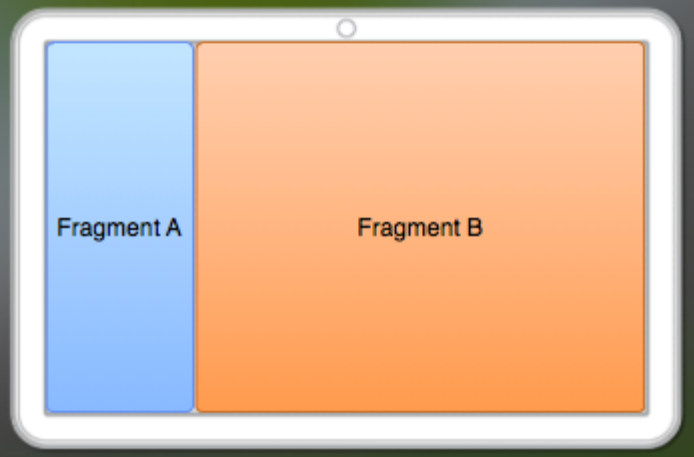
Reusing fragments

Action starts Activity B



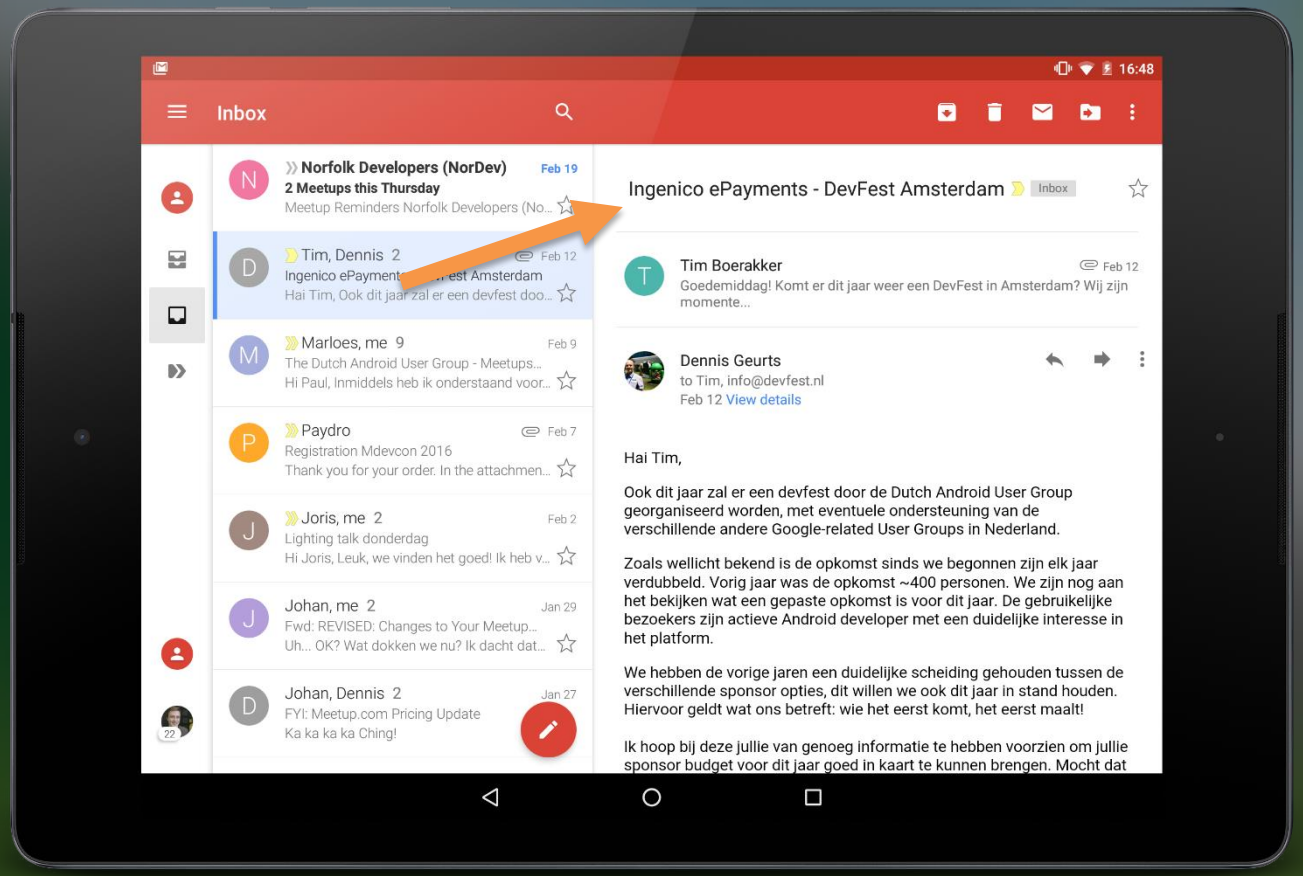
Activity A contains Fragment A
Activity B contains Fragment B

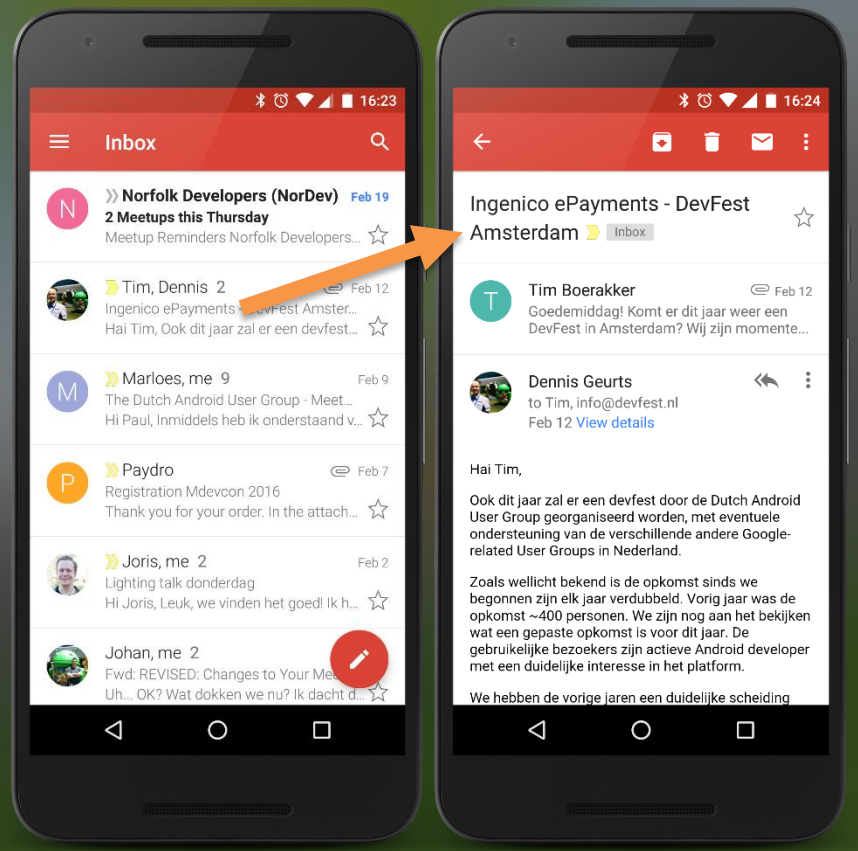
Action updates Fragment B



Activity contains
Fragment A & B

Reusing fragments

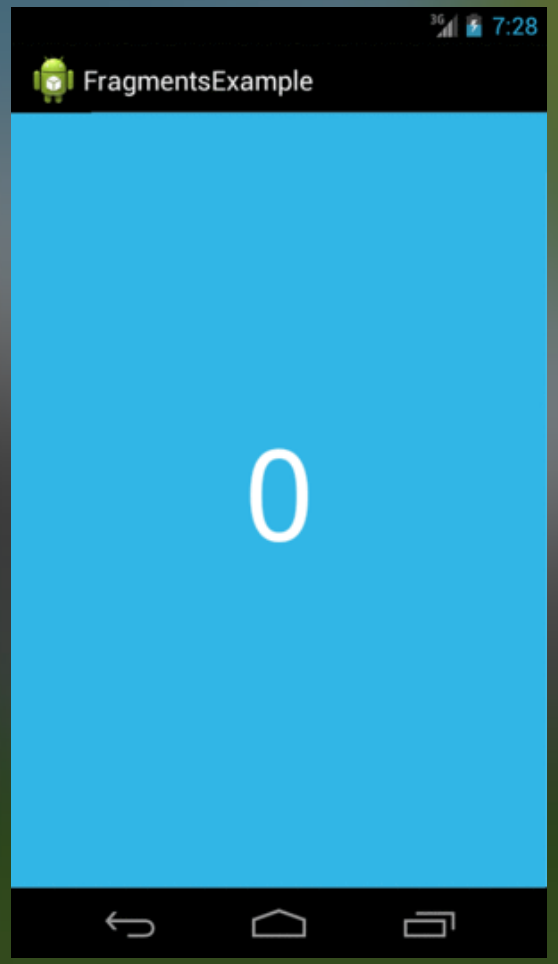




Fragments

- Reusable & modular
- Have their own lifecycle
- Interactions go through the FragmentManager
  - `activity.getFragmentManager()`
- Can be added to a layout through
  - XML, as `<fragment>`, or
  - Code, using FragmentTransactions

ViewPager



ViewPager

- Similar to a ListView:
  - Adapter provides the data
  - In theory, unlimited contents
- Items can be Views or Fragments
  - For fragments, use a matching adapter:
    - FragmentPagerAdapter
    - FragmentStatePagerAdapter

Lesson 5

Course Guide:
Lesson 5

1. ViewPager
2. Fragments
3. MediaPlayer

Lesson 5

- Fragments and ViewPagers
- The support package
- Create AnimalFragment
- Implement AnimalPagerAdapter

Homework

- Basic event driven programming
- Writing tests
- Working with more complex concepts
  - Services, Broadcast Receivers, Databases, etc.
  - Performing background tasks

Beginner's Workshop

Paul Lammertsma
CTO, Pixplicity

.Pixplicity®