Java and Android Concurrency

Introduction to Android Programming

fausto.spoto@univr.it

git@bitbucket.org:spoto/java-and-android-concurrency.git

git@bitbucket.org:spoto/factorization-client.git

Android Programming is Difficult

Native Android programming is performed is Java. Most people think it is consequently simple UI design with little underlying logic

In reality, Android applications must

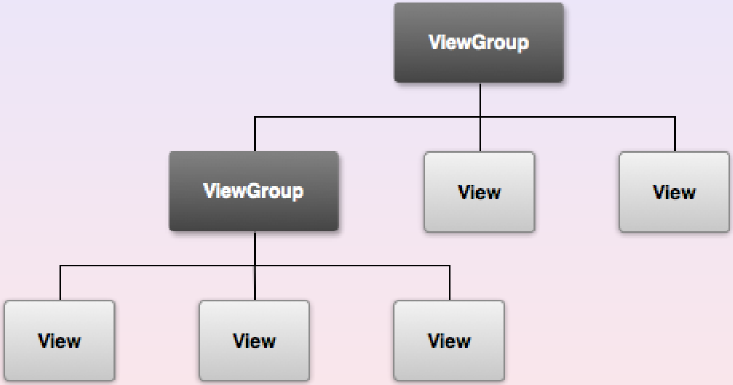
- support different devices and orientations
- make heavy use of concurrency
- deal with components that are created and destroyed by the framework

References

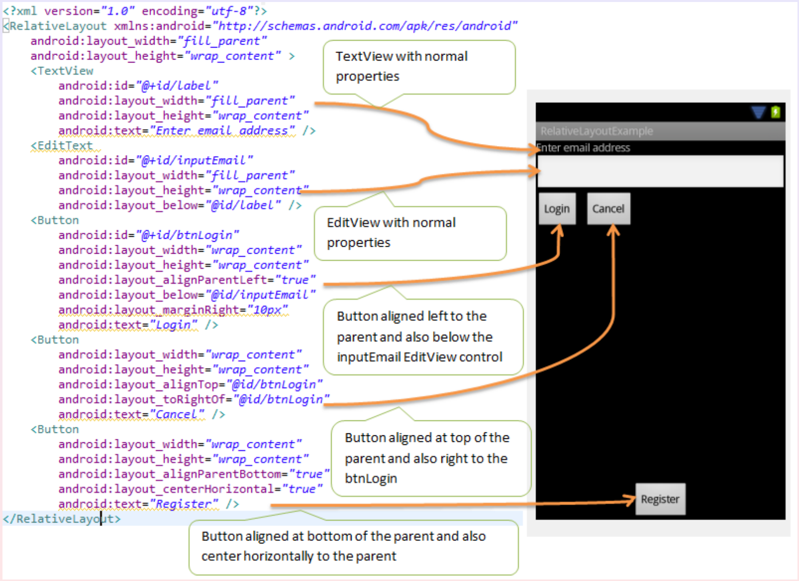
Some of this material has been taken from:

- `https://developer.android.com/training/index.html`
- *Head First Design Patterns*, 2004, O'Reilly Media
- *Android Programming: The Big Nerd Ranch Guide*, 2015, Financial Times/Prentice Hall

User Interfaces: The Composite Pattern



Example of Relative Layout



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Enter email address" />
    <EditText
        android:id="@+id/inputEmail"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label" />
    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@id/inputEmail"
        android:layout_marginRight="10px"
        android:text="Login" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@id/btnLogin"
        android:layout_toRightOf="@id/btnLogin"
        android:text="Cancel" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="Register" />
</RelativeLayout>
```

TextView with normal properties

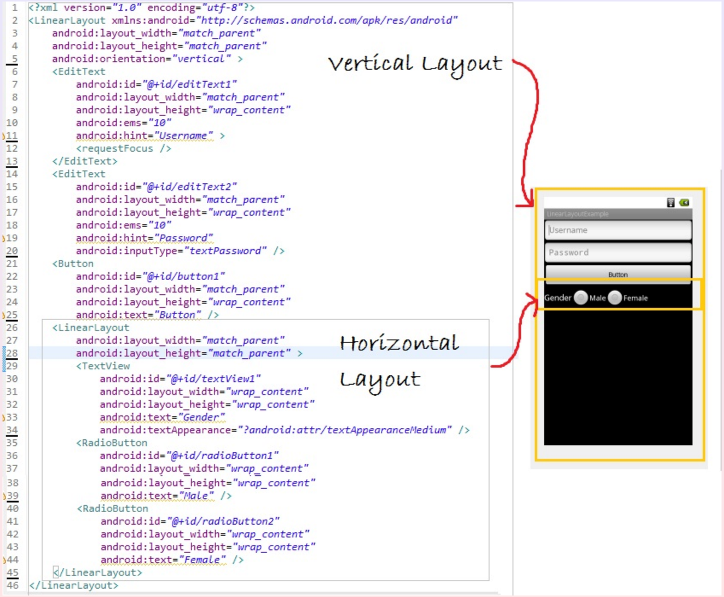
EditView with normal properties

Button aligned left to the parent and also below the inputEmail EditView control

Button aligned at top of the parent and also right to the btnLogin

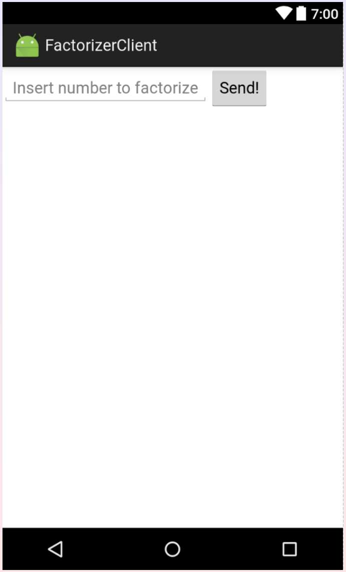
Button aligned at bottom of the parent and also center horizontally to the parent

Example of Linear Layout

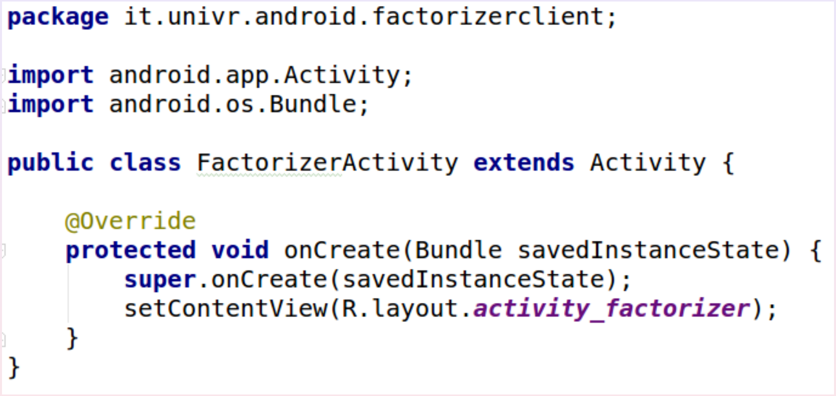


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Username" >
        <requestFocus />
    </EditText>
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Password"
        android:inputType="textPassword" />
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Gender"
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Male" />
        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Female" />
    </LinearLayout>
</LinearLayout>
```

The First Version of our Activity



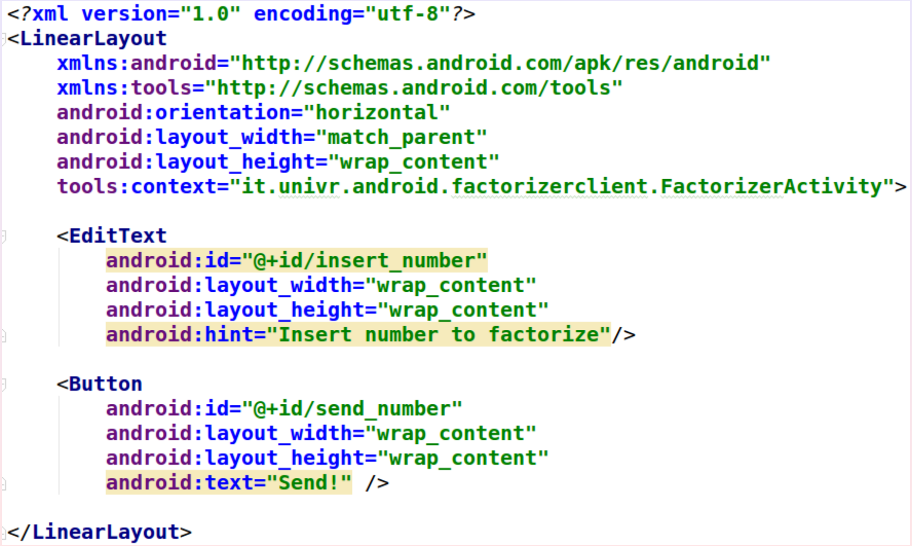
How an Activity Looks Like in Code



```java
package it.univr.android.factorizerclient;

import android.app.Activity;
import android.os.Bundle;

public class FactorizerActivity extends Activity {

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

User Interfaces: Declarative Definition in XML

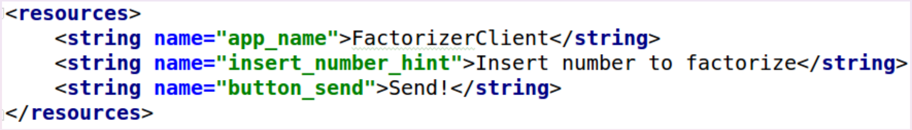
File `res/layout/activity_factorizer.xml`



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="it.univr.android.factorizerclient.FactorizerActivity">

    <EditText
        android:id="@+id/insert_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Insert number to factorize"/>

    <Button
        android:id="@+id/send_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send!" />

</LinearLayout>
```

String Resources

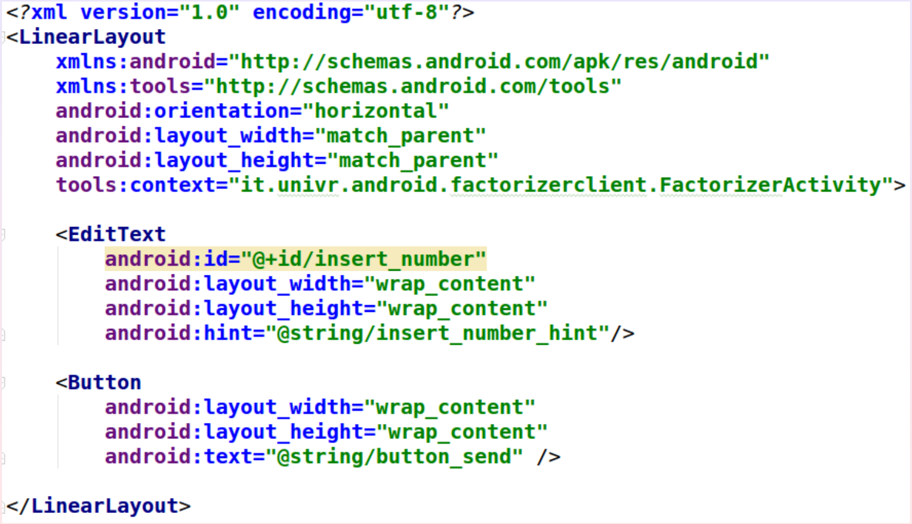
File `res/values/strings.xml`



```xml
<resources>
    <string name="app_name">FactorizerClient</string>
    <string name="insert_number_hint">Insert number to factorize</string>
    <string name="button_send">Send!</string>
</resources>
```

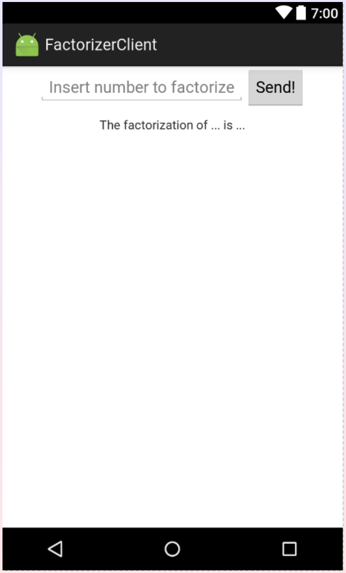
Using String Resources

File `res/layout/activity_factorizer.xml`



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="it.univr.android.factorizerclient.FactorizerActivity">

    <EditText
        android:id="@+id/insert_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/insert_number_hint"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />

</LinearLayout>
```

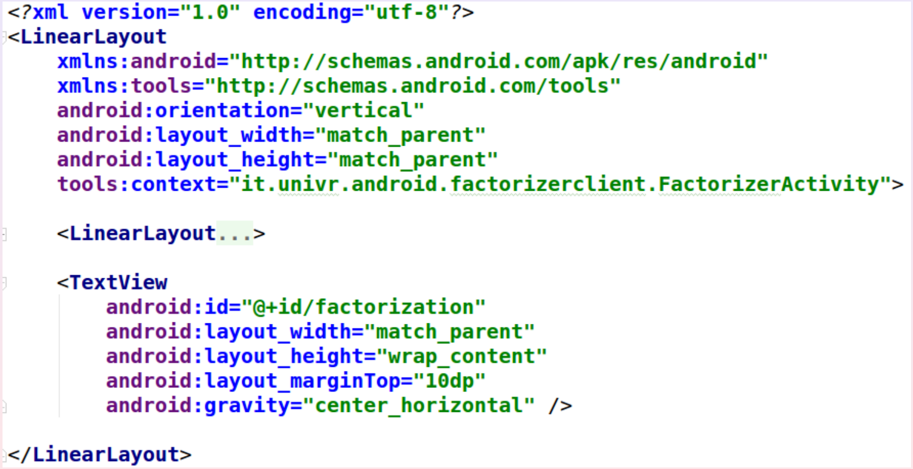
Add a Factorization Result and Center



The factorization of ... is ...

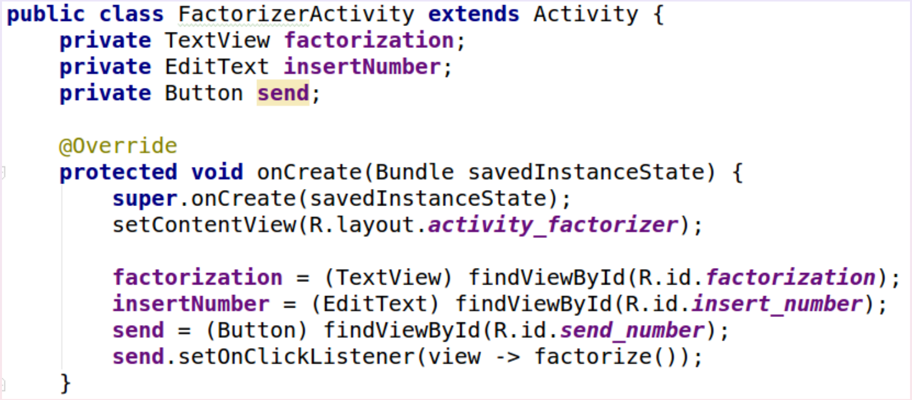
Add a Factorization Result and Center

File `res/layout/activity_factorizer.xml`



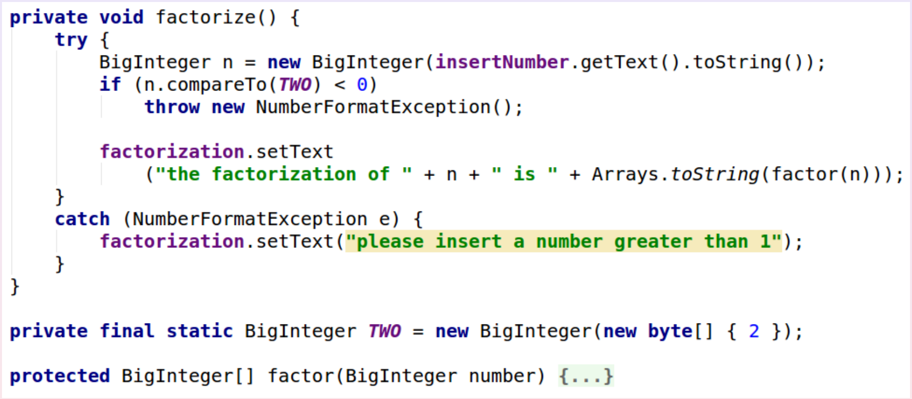
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="it.univr.android.factorizerclient.FactorizerActivity">

    <LinearLayout...>

    <TextView
        android:id="@+id/factorization"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_horizontal" />

</LinearLayout>
```

Wire the Calculation of the Factorization



```java
public class FactorizerActivity extends Activity {
    private TextView factorization;
    private EditText insertNumber;
    private Button send;

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

        factorization = (TextView) findViewById(R.id.factorization);
        insertNumber = (EditText) findViewById(R.id.insert_number);
        send = (Button) findViewById(R.id.send_number);
        send.setOnClickListener(view -> factorize());
    }
```

Wire the Calculation of the Factorization



```
private void factorize() {
    try {
        BigInteger n = new BigInteger(insertNumber.getText().toString());
        if (n.compareTo(TWO) < 0)
            throw new NumberFormatException();

        factorization.setText
            ("the factorization of " + n + " is " + Arrays.toString(factor(n)));
    }
    catch (NumberFormatException e) {
        factorization.setText("please insert a number greater than 1");
    }
}

private final static BigInteger TWO = new BigInteger(new byte[] { 2 });

protected BigInteger[] factor(BigInteger number) {...}
```

Useful Android Studio Tricks

Activate Java 8

Modify the module build.gradle as follows:

```
android {
  compileSdkVersion 25
  ...
  defaultConfig {
    ...
    jackOptions {
      enabled true
    }
  }
  ...
  compileOptions {
    targetCompatibility 1.8
    sourceCompatibility 1.8
  }
}
```

Useful Android Studio Tricks

Make Android Studio add all missing imports

- For Windows/Linux, go to File ⇒ Settings ⇒ Editor ⇒ General ⇒ Auto Import ⇒ Java and make the following changes:
  1. change *Insert imports on paste* value to *All*
  2. mark *Add unambigious imports on the fly* option as checked
- On a Mac, do the same thing in Android Studio ⇒ Preferences

Use @UiThread and @WorkerThread annotations

Modify the module build.gradle as follows:

```
dependencies {
  compile 'com.android.support:support-annotations:25.1.0'
  compile 'net.jcip:jcip-annotations:1.0'
  compile 'net.jcip:jcip-annotations:1.0'
}
```

Localizing the Application

Hardcoded strings cannot be localized:

```
factorization.setText
  ("the factorization of " + n + " is "
    + Arrays.toString(factor(n)));
```

Instead, we can use symbolic reference to string resources and provide distinct resource files for different countries. The first step is to replace all hardcoded strings with string resources:

res/values/strings.html

```
<string name="insert_at_least_2">
  please insert a number greater than 1</string>
<string name="factorization_message">
  the factorization of %1$s is %2$s</string>
```

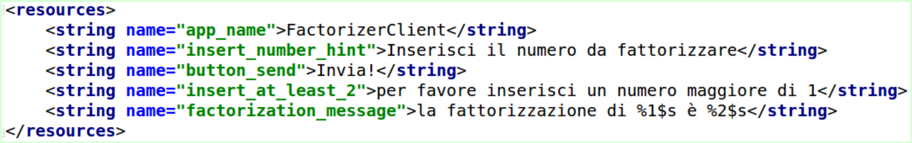
Using String Resources Only

```
private void factorize() {
  try {
    BigInteger n = new BigInteger(insertNumber.getText().toString()
    if (n.compareTo(TWO) < 0)
      throw new NumberFormatException();

    factorization.setText(getResources().getString
      (R.string.factorization_message,
       n, Arrays.toString(factor(n))));  // <- arguments
  }
  catch (NumberFormatException e) {
    factorization.setText
      (getResources().getString(R.string.insert_at_least_2));
  }
}
```

Localizing String Resources

File `res/values/strings.xml` is used by default, but one can provide distinct versions of the same file for different countries:

res/values-it/strings.xml



```xml
<resources>
    <string name="app_name">FactorizerClient</string>
    <string name="insert_number_hint">Inserisci il numero da fattorizzare</string>
    <string name="button_send">Invia!</string>
    <string name="insert_at_least_2">per favore inserisci un numero maggiore di 1</string>
    <string name="factorization_message">la fattorizzazione di %1$s è %2$s</string>
</resources>
```

Android will automatically select at runtime the right resource file according to the country set on the device when the app is running

git checkout local

Switch to tag `local` to see the application code as it is up to this point:

```
git checkout local
```

Later, you can switch to other tags or come back to the latest version of the code:

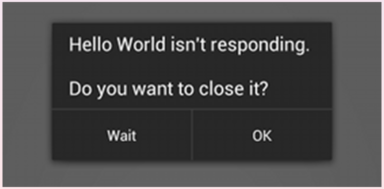
```
git checkout master
```

Factorization Might Be Expensive

Computing the factorization might take many seconds

Method `onCreate()` is called in the EDT, hence it is a `@UiThread` method:

- while the factorization is in progress, the UI freezes
- if this takes too long, an Application Not Responding (ANR) message might appear, allowing the user to stop the app



Hello World isn't responding.

Do you want to close it?

We Need a Worker Thread

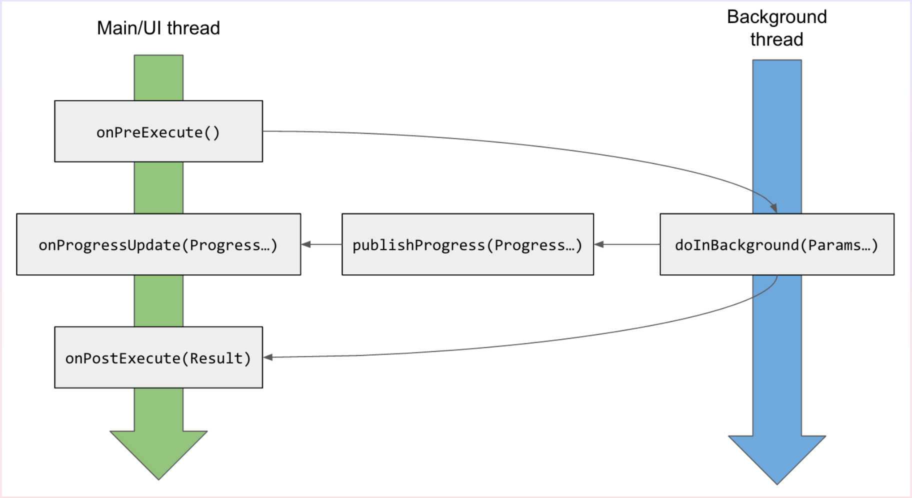
Android has the `java.lang.Thread` class, but

1. threads have no native way of reporting their work back to the EDT
2. threads have no native support for progress updates
3. threads cannot specify an executor
4. threads do not increase the rank of the application

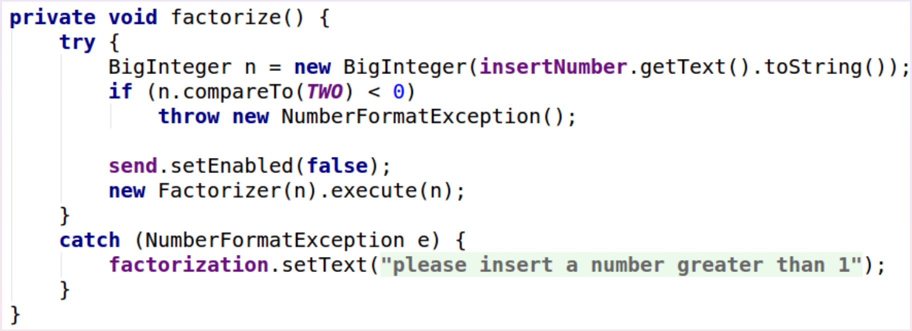
Rather than threads, Android uses other specific classes. The simplest class is `android.os.AsyncTask<Params, Progress, Result>`:

```
public class AsyncTask<Params, Progress, Result> {
  @UiThread AsyncTask<...> execute(Params... params);
  @UiThread void on PreExecute();
  @WorkerThread Result doInBackground(Params... params);
  @UiThread void onProgressUpdate(Progress... values);
  @UiThread void onPostExecute(Result result);
}
```

AsyncTask

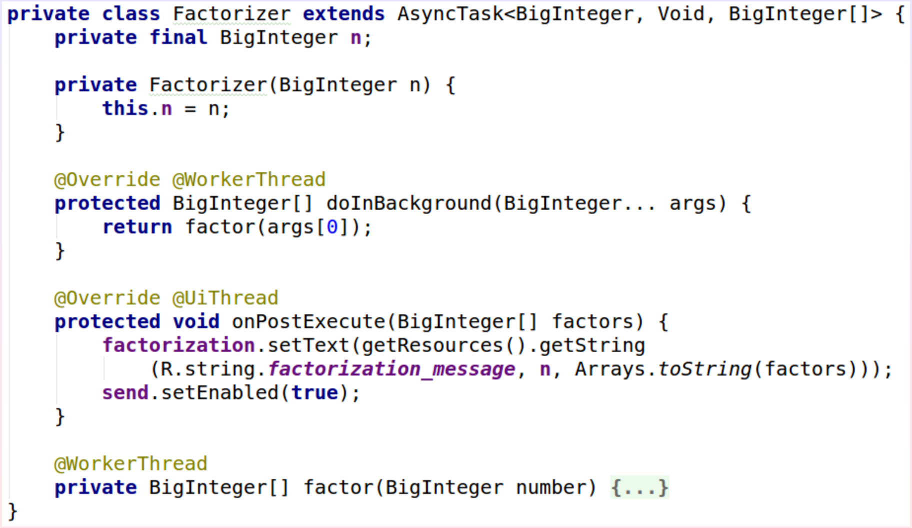

Main/UI thread
Background thread
onPreExecute()
onProgressUpdate(Progress…)
publishProgress(Progress…)
doInBackground(Params…)
onPostExecute(Result)

Spawning the AsyncTask



```java
private void factorize() {
    try {
        BigInteger n = new BigInteger(insertNumber.getText().toString());
        if (n.compareTo(TWO) < 0)
            throw new NumberFormatException();

        send.setEnabled(false);
        new Factorizer(n).execute(n);
    }
    catch (NumberFormatException e) {
        factorization.setText("please insert a number greater than 1");
    }
}
```

The Factorizing AsyncTask



```
private class Factorizer extends AsyncTask<BigInteger, Void, BigInteger[]> {
    private final BigInteger n;

    private Factorizer(BigInteger n) {
        this.n = n;
    }

    @Override @WorkerThread
    protected BigInteger[] doInBackground(BigInteger... args) {
        return factor(args[0]);
    }

    @Override @UiThread
    protected void onPostExecute(BigInteger[] factors) {
        factorization.setText(getResources().getString
            (R.string.factorization_message, n, Arrays.toString(factors)));
        send.setEnabled(true);
    }

    @WorkerThread
    private BigInteger[] factor(BigInteger number) {...}
}
```

git checkout asynctask

Switch to tag asynctask to see the application code as it is up to this point:

```
git checkout asynctask
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Epic Fail

1. Add logging at the beginning of `doInBackground` and at the end of `onPostExecute`
2. Ask for the factorization of 1234533345678912, which takes around 10 seconds
3. rotate the device
4. verify on the logs that the computation has finished

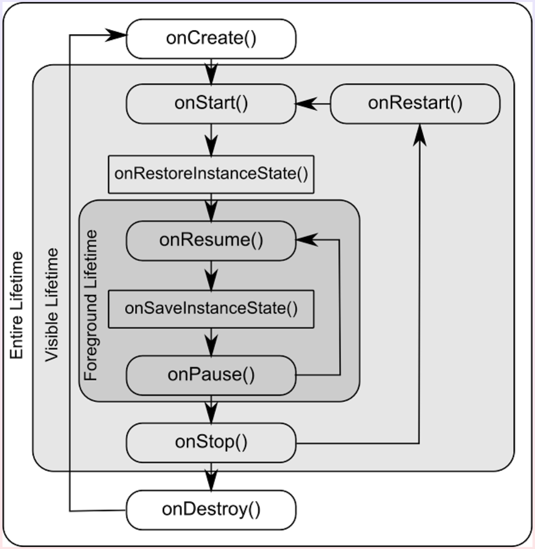
Where is the result gone?

Activities Die

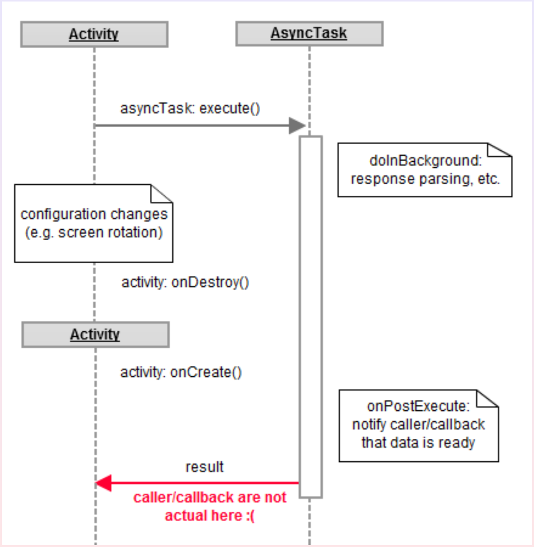
“I see dead activities”

Never talk to dead activities

Activities: UI Screens with a Lifecycle



The Problem: Do not Report Back to a Dead Activity


Activity
AsyncTask
asyncTask: execute()
doInBackground: response parsing, etc.
configuration changes (e.g. screen rotation)
activity: onDestroy()
Activity
activity: onCreate()
onPostExecute: notify caller/callback that data is ready
result
caller/callback are not actual here :(

Where Did We Go Wrong?

We have violated the Single Responsibility Principle

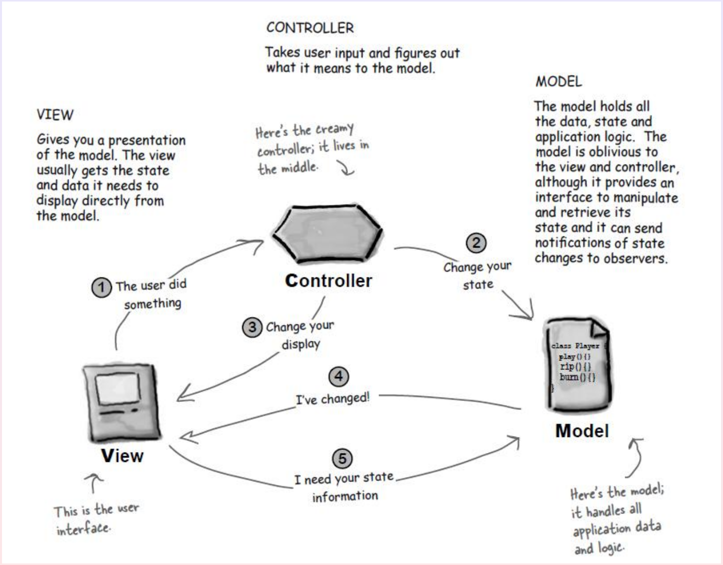
A class should have a single responsibility, hence it should have a single reason to change

An activity has the resposnibility of being a graphical view. It is not

- a controller computing factorizations
- a model storing the result of the last factorization

We need to move away concerns from the activity. Let us go back to the MVC pattern

The Model/View/Controller Design Pattern


CONTROLLER
Takes user input and figures out what it means to the model.
MODEL
The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.
VIEW
Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.
Here's the creamy controller; it lives in the middle.
Controller
1 The user did something
2 Change your state
3 Change your display
4 I've changed!
5 I need your state information
View
Model
class Player {
play() {}
rip() {}
burn() {}
}
This is the user interface.
Here's the model; it handles all application data and logic.

The MVC Triple (Identical to That for Swing)

```
@ThreadSafe
public class MVC {
  public final Model model;
  public final Controller controller;
  private final List<View> views = new CopyOnWriteArrayList<>();

  public MVC(Model model, Controller controller) {
    this.model = model;
    this.controller = controller;

    model.setMVC(this);
    controller.setMVC(this);
  }
```

The MVC Triple (Identical to That for Swing)

```
  public void register(View view) { views.add(view); }

  public void unregister(View view) { views.remove(view); }

  public interface ViewTask {
    void process(View view);
  }

  public void forEachView(ViewTask task) {
    for (View view: views)
      task.process(view);
  }
}
```

The Model Stores the Last Factorization

```
@ThreadSafe
public class Model {
  private MVC mvc;
  private BigInteger n;
  private BigInteger[] factors;

  public void setMVC(MVC mvc) { this.mvc = mvc; }

  @UiThread public void storeFactorization(BigInteger n, BigInteger[] factors) {
    this.n = n;
    this.factors = factors.clone();
    mvc.forEachView(View::onModelChanged);
  }

  @UiThread public BigInteger getLastFactorizedNumber() {
    return n;
  }

  @UiThread public BigInteger[] getLastFactorization() {
    return factors.clone();
  }
}
```

The Controller Performs the Factorization

```
@ThreadSafe public class Controller {
  private MVC mvc;
  public void setMVC(MVC mvc) { this.mvc = mvc; }

  @UiThread public void factorize(BigInteger n) {
    new Factorizer(n).execute(n);
  }

  private class Factorizer extends AsyncTask<BigInteger, Void, BigInteger[]> {
    private final BigInteger n;
    @UiThread private Factorizer(BigInteger n) { this.n = n; }

    @Override @WorkerThread
    protected BigInteger[] doInBackground(BigInteger... args) {
      return factor(args[0]);
    }

    @Override @UiThread
    protected void onPostExecute(BigInteger[] factors) {
      mvc.model.storeFactorization(n, factors);
    }

    @WorkerThread private BigInteger[] factor(BigInteger number) { ... }
  }
}
```

The View Reflects Model Changes

```
public interface View {
  @UiThread void onModelChanged();
}
```

The FactorizerActivity is Our View Now

```
public class FactorizerActivity extends Activity implements View {
  ...
  @Override @UiThread protected void onStart() {
    super.onStart();
    mvc.register(this);
    onModelChanged();
  }

  @Override @UiThread protected void onStop() {
    mvc.unregister(this); // this allows dead activities to be garbage collected
    super.onStop();
  }

  @Override @UiThread public void onModelChanged() {
    BigInteger n = mvc.model.getLastFactorizedNumber();
    if (n == null) // no factorization up to now
      return;

    BigInteger[] factors = mvc.model.getLastFactorization();
    factorization.setText(getResources().getString
      (R.string.factorization_message, n, Arrays.toString(factors)));
    send.setEnabled(true);
  }
```

The FactorizerActivity is Our View Now

```
public class FactorizerActivity extends Activity implements View {
  ...
  @UiThread private void factorize() {
    try {
      BigInteger n = new BigInteger(insertNumber.getText().toString());
      if (n.compareTo(TWO) < 0)
        throw new NumberFormatException();

      send.setEnabled(false);
      mvc.controller.factorize(n);
    }
    catch (NumberFormatException e) {
      factorization.setText(getResources().getString(R.string.insert_at_least_2));
    }
  }
}
```

Where do We Create and Store the MVC Triple?

A running Android application has an `android.app.Application` context where shared, application-wide *global* data can be stored

1. redefine it into our specific application class:

```
public class FactorizerApplication extends Application {
  private MVC mvc;

  @Override public void onCreate() {
    super.onCreate();
    mvc = new MVC(new Model(), new Controller());
  }

  public MVC getMVC() {
    return mvc;
  }
}
```

2. specify that class for our app, inside `AndroidManifest.xml`:

```
<application
  android:name=".FactorizerApplication"
  ... >
```

git checkout mvc

Switch to tag `mvc` to see the application code as it is up to this point:

```
git checkout mvc
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Query a Remote Factorization Server

Instead of letting our little phone compute the factorization, let us query a remote factorization server, such as that implemented through a servlet on Heroku

Separation of concerns rocks

Thanks to separation of concerns, this just amounts to modifying the controller, by letting its `factor()` method contact the servlet instead of perform the factorization

The New Controller

```
@WorkerThread private BigInteger[] factor(BigInteger number) {
  try {
    URL url = new URL(SERVER + number);
    URLConnection conn = url.openConnection();
    String answer = "", line;
    BufferedReader in = null;
    try {
      in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
      while ((line = in.readLine()) != null)
        answer += line;
    }
    finally {
      if (in != null)
        in.close();
    }
    answer = answer.substring(1, answer.length() - 1);
    String[] numbers = answer.split(",");
    BigInteger[] result = new BigInteger[numbers.length];
    for (int pos = 0; pos < numbers.length; pos++)
      result[pos] = new BigInteger(numbers[pos].trim());
    return result;
  }
  catch (IOException e) { return new BigInteger[0]; }
}
```

Give the App the Right to Access the Internet

Add to `AndroidManifest.xml`:

```
<uses-permission
  android:name="android.permission.INTERNET" />
<uses-permission
  android:name="android.permission.ACCESS_NETWORK_STATE" />
```

git checkout remote

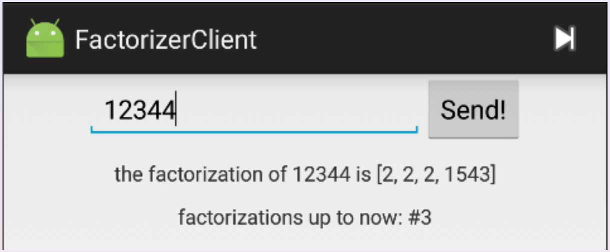
Switch to tag `remote` to see the application code as it is up to this point:

```
git checkout remote
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Add a Factorizations Counter Turned On-Off from Menu



the factorization of 12344 is [2, 2, 2, 1543]

factorizations up to now: #3

This require two extra pieces of information:

- if the counter should be shown or not
- how many factorizations have been performed up to now

View-Specific Information

These pieces of information are related to the status and the history of the view, they are *not* related to the data model

They should be put in the view, not in the model

```
public class FactorizerActivity extends Activity implements View {
  ...
  // view state
  private boolean isCountOn;
  private int factorizationsCount;
```

Let the Activity Report the Counter if Required

```
public class FactorizerActivity extends Activity implements View {
  ...
  @Override @UiThread
  public void onModelChanged() {
    ...
    updateCounter();
  }

  @UiThread private void updateCounter() {
    if (isCountOn)
      counter.setText(getResources().getString
        (R.string.factorizations_up_to_now, factorizationsCount));
      else
        counter.setText("");
  }

  @UiThread private void factorize() {
    ...
    factorizationsCount++;
    ...
  }
}
```

Specify the Menu Item

res/menu/activity_factorizer.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/menu_item_show_counter"
    android:title="@string/show_counter"
    android:icon="@android:drawable/ic_media_next"
    android:showAsAction="ifRoom|withText" />
</menu>
```

Add the Menu to the Activity

```
public class FactorizerActivity extends Activity implements View {
  ...
  @Override @UiThread
  public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    getMenuInflater().inflate(R.menu.activity_factorizer, menu);
    return true; // show the menu
  }

  @Override @UiThread
  public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.menu_item_show_counter) {
      isCountOn = !isCountOn;
      updateCounter();
      return true;
    }
    else
      return super.onOptionsItemSelected(item);
  }
```

The Dead Activity Problem Strikes Again

Play with the app for a while, turn the counter on, then rotate the device

The factorization counter gets hidden and reset to 0

The view instance state fields get reset to their default value at activity destruction/recreation

We need a way to save the view instance state at destruction time and to recover it at recreation

Save and Recover the Instance State of the View

```
public class FactorizerActivity extends Activity implements View {
  ...
  private final static String TAG = FactorizerActivity.class.getName();

  @Override @UiThread
  protected void onCreate(Bundle savedInstanceState) {
    ...
    if (savedInstanceState != null) {
      isCountOn = savedInstanceState.getBoolean(TAG + "isCountOn");
      factorizationsCount = savedInstanceState.getInt(TAG + "factorizationsCount");
    }
  }

  @Override @UiThread
  protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putBoolean(TAG + "isCountOn", isCountOn);
    outState.putInt(TAG + "factorizationsCount", factorizationsCount);
  }
  ...
}
```

git checkout menu

Switch to tag `menu` to see the application code as it is up to this point:

```
git checkout menu
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Exercise

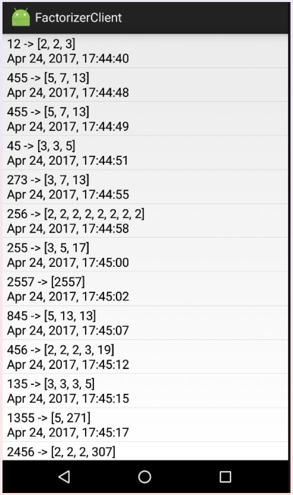
A Chat Client in Android

1. Write a new Android appplication whose only activity has
   - an edit text view for inserting the author
   - an edit text view for inserting a message
   - a send button to send the author/message pair to the chat servlet at `https://mysterious-escarpment-70352.herokuapp.com/` through the `AddMessage?author=AA&text=TT` path
2. Verify that messages are actually stored in the server, by pointing a browser to the `ListMessages?howmany=XX` path
3. Add a menu to the app, with an item that reloads the last 10 messages from the server and reports them inside another text view

Use the MVC design pattern

Addition of a Second Activity

Let us add a menu button that starts a new activity, showing the list and time of the latest factorizations performed with the application:



FactorizerClient

```
12 -> [2, 2, 3]
Apr 24, 2017, 17:44:40
455 -> [5, 7, 13]
Apr 24, 2017, 17:44:48
455 -> [5, 7, 13]
Apr 24, 2017, 17:44:49
45 -> [3, 3, 5]
Apr 24, 2017, 17:44:51
273 -> [3, 7, 13]
Apr 24, 2017, 17:44:55
256 -> [2, 2, 2, 2, 2, 2, 2, 2]
Apr 24, 2017, 17:44:58
255 -> [3, 5, 17]
Apr 24, 2017, 17:45:00
2557 -> [2557]
Apr 24, 2017, 17:45:02
845 -> [5, 13, 13]
Apr 24, 2017, 17:45:07
456 -> [2, 2, 2, 3, 19]
Apr 24, 2017, 17:45:12
135 -> [3, 3, 3, 5]
Apr 24, 2017, 17:45:15
1355 -> [5, 271]
Apr 24, 2017, 17:45:17
2456 -> [2, 2, 2, 307]
```

Modifications to the Model

```
@Immutable public static class Factorization {
  private final BigInteger n;
  private final BigInteger[] factors;
  private final Date when;
  private final static DateFormat format
    = new SimpleDateFormat("MMM d, yyyy, HH:mm:ss");

  private Factorization(BigInteger n, BigInteger[] factors) {
    this.n = n;
    this.factors = factors.clone();
    this.when = new Date();
  }

  public BigInteger getFactorizedNumber() { return n; }

  public BigInteger[] getFactors() { return factors.clone(); }

  @Override public String toString() {
    return n + " -> " + Arrays.toString(factors) + "\n" + format.format(when);
  }
}
```

Modifications to the Model

```
@ThreadSafe public class Model { ...
  private final @GuardedBy("itself") LinkedList<Factorization> factorizations
    = new LinkedList<>();
  private final static int MAX_FACTORIZATIONS = 20;

  public void storeFactorization(BigInteger n, BigInteger[] factors) {
    synchronized (factorizations) {
      if (factorizations.size() >= MAX_FACTORIZATIONS)
        factorizations.removeFirst();
      factorizations.add(new Factorization(n, factors));
    }

    mvc.forEachView(View::onModelChanged);
  }

  public Factorization getLastFactorization() {
    synchronized (factorizations) {
      return factorizations.isEmpty() ? null : factorizations.getLast();
    }
  }

  public Factorization[] getFactorizations() {
    synchronized (factorizations) {
      return factorizations.toArray(new Factorization[factorizations.size()]);
    }
```

Posting Runnables on the EDT

Since the model has been made thread-safe without thread confinement, calls to `forEachView` happen on any thread now:

```
public void storeFactorization(BigInteger n, BigInteger[] factors)
  ...
  mvc.forEachView(View::onModelChanged);
}
```

Hence we must post them to the EDT now: in the `MVC` triple we edit:

```
public void forEachView(ViewTask task) {
  // run a Runnable in the UI thread
  new Handler(Looper.getMainLooper()).post(() -> {
    for (View view: views)
      task.process(view);
  });
}
```

Addition of a New Menu Item

In file `res/menu/activity_fractorizer.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/menu_item_show_counter"
    android:title="@string/show_counter"
    android:icon="@android:drawable/ic_media_next"
    android:showAsAction="ifRoom|withText" />
  <item
    android:id="@+id/menu_item_show_last_factorizations"
    android:title="@string/show_factorizations_list"
    android:icon="@android:drawable/ic_menu_recent_history"
    android:showAsAction="ifRoom|withText" />
</menu>
```

Addition of a New Menu Handler

In file FactorizerActivity.java:

```
@Override @UiThread
public boolean onOptionsItemSelected(MenuItem item) {
  if (item.getItemId() == R.id.menu_item_show_counter) { ... }
  else if (item.getItemId() == R.id.menu_item_show_last_factorizations) {
    HistoryActivity.start(this); // replaces the current activity
    return true;
  }
  else
    return super.onOptionsItemSelected(item);
}
```

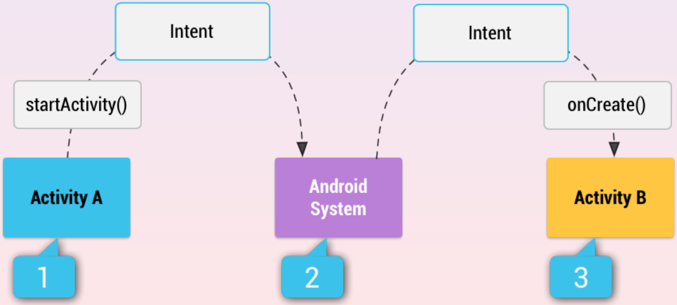
The New Activity

```
public class HistoryActivity extends ListActivity implements View {
  private MVC mvc;

  public static void start(Context parent) {
    parent.startActivity(new Intent(parent, HistoryActivity.class));
  }

  @Override @UiThread
  protected void onCreate(Bundle savedInstanceState) { like FactorizerActivity }

  @Override @UiThread
  protected void onStart() { like FactorizerActivity }

  @Override @UiThread
  protected void onStop() { like FactorizerActivity }

  @Override @UiThread
  public void onModelChanged() {
    ArrayAdapter<Model.Factorization> adapter = new ArrayAdapter<>
      (this, android.R.layout.simple_list_item_1, mvc.model.getFactorizations());
    setListAdapter(adapter);
  }
}
```

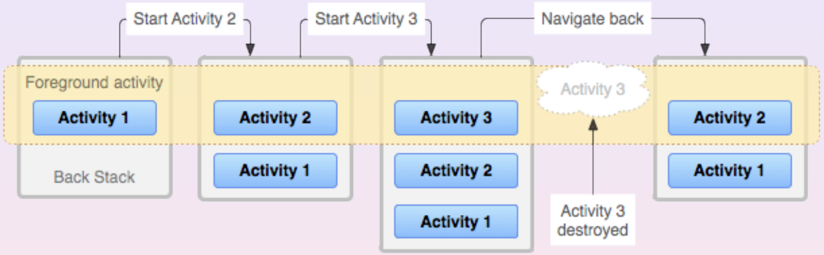
Starting Activities through Intents

Activities can be started in order to respond to the need of fulfilling an *intent*. Intents in Android specify a goal to be achieved. Intent resolution is a complex and abstract process. Here, we just use the simplest intent: one that explicitly specifies the activity that must be run:

```
context.startActivity(new Intent(context, ActivityClass.class))
```



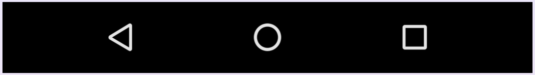
The Activity Back Stack



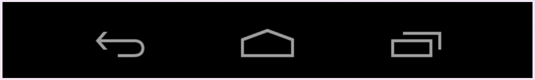
Activities in the back stack are kept until they are explicitly destroyed through the back button of the phone, unless the system needs to reclaim memory, in which case they can be destroyed earlier

Android Buttons

Android devices normally have three soft buttons:



back home tasks



back: destroy the current activity, go back to the previous one in the back stack, which becomes the new current activity. If instead the current activity was the only one in the back stack, destroy the whole application as well

home: show the home screen. If the user comes back to the application later, show the current activity again

git checkout history

Switch to tag history to see the application code as it is up to this point:

```
git checkout history
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Exercise

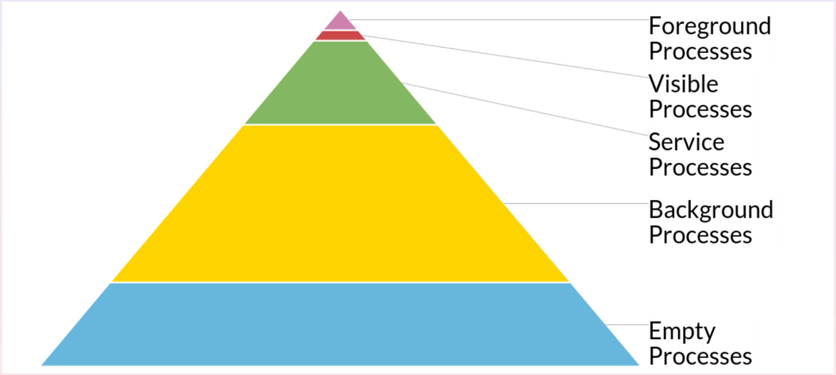
A Flickr Client in Android

1. obtain your Flickr API key at `https://www.flickr.com/services/api/misc.api_keys.html`
2. Write a new Android appplication with two activities:
   - the first activity allows the user to insert a search string. When the `Send!` button is clicked, the second activity is shown instead
   - the second activity lists the names and URL of the last 50 Flickr pictures related to that search. For that, use the Flickr API method `https://api.flickr.com/services/rest?method=flickr.photos.search&api_key=KEY&text=string&extras=url_z,description,tags&per_page=50` and parse the resulting XML
   - you can see the documentation of the above API method at `https://www.flickr.com/services/api/flickr.photos.search.html`

Use the MVC design pattern

Android Priority Pyramid

Android destroyes applications to reclaim memory, starting with applications whose components are closer to the bottom of the following pyramid:



Android Components

Android applications are composed of four kinds of components:

activities	views interacting with the user
services	high-priority background processes
content providers	abstract presentations of a data source
broadcast receivers	listeners to external events

The priority of a running Android application is the highest priority among those of its active components

Threads and ASyncTasks are not components

Background threads do not contribute to the determination of the priority of an Android application. A running thread or ASyncTask gives an application the *background processes* priority, which is very low

Delegating background tasks to threads or ASyncTasks may lead to the OS destroying the app although its background threads are doing useful work

Running Background Processes inside a Component

The priority of a background task can be increased if it is run inside a service component. The Android library provides a simplified service implementation, called IntentService:

1. the client calls `startService(intent)`, where `intent` must target the intent service and specify the task
2. the intent is put into an intent queue
3. intents are removed from the queue and executed sequentially on a worker thread

`IntentService`

Tasks are executed sequentially. There is a single executor per intent service instance. The executor has middle priority

`ASyncTasks`

Tasks are executed sequentially. There is a single executor per application. This choice has changed over time and can be modified by the programmer. The executor has low priority

Adding an Intent to the Intent Queue

```
public class FactorizationService extends IntentService {
  private final static String ACTION_FACTORIZE = "factorize";
  private final static String PARAM_N = "n";

  // called by the OS. Must be public and with no args
  public FactorizationService() {
    super("factorization service");
  }

  static void factorize(Context context, BigInteger n) {
    // pack the task into the intent, target the FactorizationService
    Intent intent = new Intent(context, FactorizationService.class);
    intent.setAction(ACTION_FACTORIZE);
    intent.putExtra(PARAM_N, n);

    // put the intent in the queue
    context.startService(intent);
  }
```

Running a Task from the Queue

```
@WorkerThread
protected void onHandleIntent(Intent intent) {
  switch (intent.getAction()) {
    case ACTION_FACTORIZE:
      BigInteger n = (BigInteger) intent.getSerializableExtra(PARAM_N);
      BigInteger[] factors = factor(n); // our old friend, calls the servlet
      MVC mvc = ((FactorizerApplication) getApplication()).getMVC();
      mvc.model.storeFactorization(n, factors); // runs on the worker thread
      break;
  }
}
```

The Controller Calls the IntentService Now

```
public class Controller {
  private MVC mvc; // unused, maybe in the future....

  public void setMVC(MVC mvc) {
    this.mvc = mvc;
  }

  @UiThread
  public void factorize(Context context, BigInteger n) {
    FactorizationService.factorize(context, n);
  }
}
```

Register the Service in the `AndroidManifest.xml`

```
<application
  ...
  <service android:name=".controller.FactorizationService" />
</application>
```

Parallel Intent Service

Remember that an intent service has a single executor per instance:

- tasks are executed in sequence, on a single worker thread

What if we want to run more tasks in parallel, on distinct worker threads?

1. extend `ExecutorIntentService` instead of `IntentService`
2. implement its method

   `protected ExecutorService mkExecutorService()`
3. tasks will be scheduled on the executor service returned by the above method

Implementing ExecutorIntentService

```
public class FactorizationService extends ExecutorIntentService {
  ...
  @Override
  protected ExecutorService mkExecutorService() {
    return Executors.newFixedThreadPool(10);
  }

  ...
}
```

git checkout intent_service

Switch to tag intent_service to see the application code as it is up to this point:

```
git checkout intent_service
```

Later, you can switch to other tags or come back to the latest version of the code:

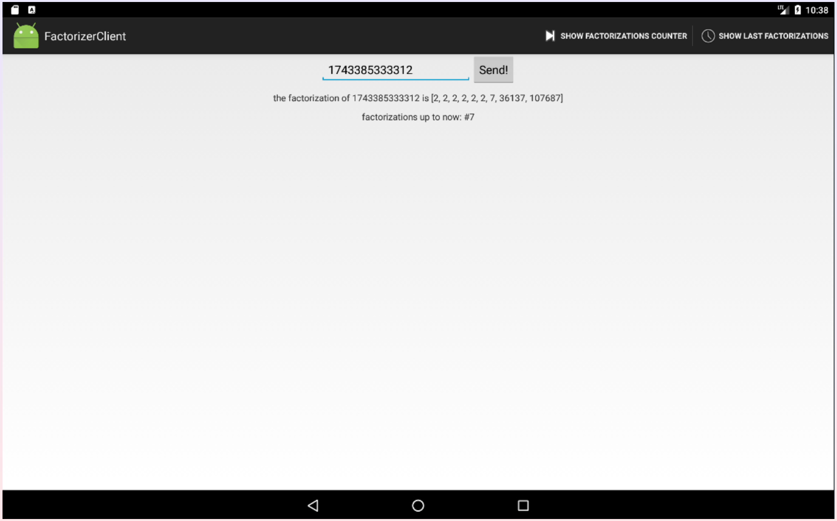
```
git checkout master
```

Exercise

A Flickr Client in Android

Modify your Flickr client so that search and parsing of the XML is performed in a background task supported by an `IntentService`

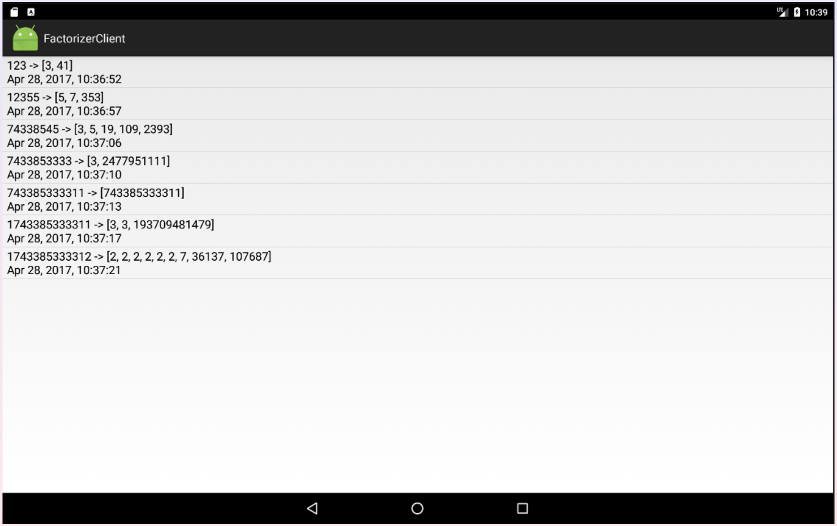
Let’s Go Tablet!



the factorization of 1743385333312 is [2, 2, 2, 2, 2, 2, 7, 36137, 107687]

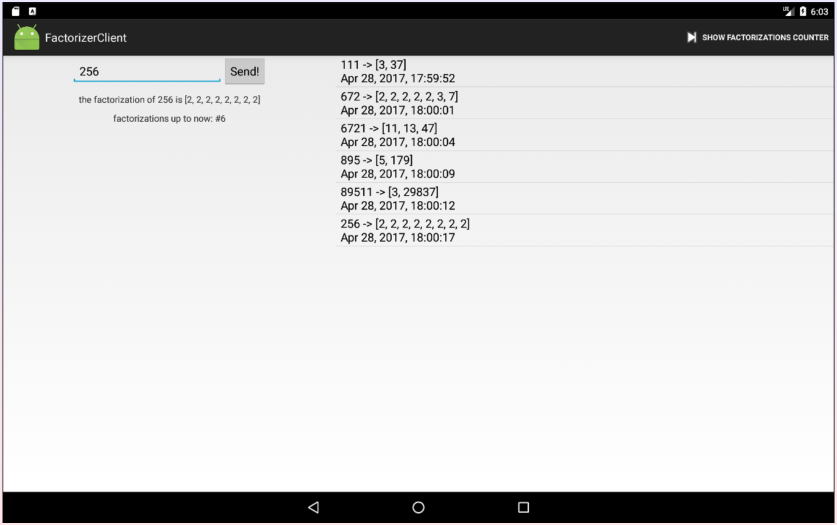
factorizations up to now: #7

Let's Go Tablet!



```
123 -> [3, 41]
Apr 28, 2017, 10:36:52
12355 -> [5, 7, 353]
Apr 28, 2017, 10:36:57
74338545 -> [3, 5, 19, 109, 2393]
Apr 28, 2017, 10:37:06
7433853333 -> [3, 2477951111]
Apr 28, 2017, 10:37:10
743385333311 -> [743385333311]
Apr 28, 2017, 10:37:13
1743385333311 -> [3, 3, 193709481479]
Apr 28, 2017, 10:37:17
1743385333312 -> [2, 2, 2, 2, 2, 2, 7, 36137, 107687]
Apr 28, 2017, 10:37:21
```

How the Application Should Look in a Tablet



the factorization of 256 is [2, 2, 2, 2, 2, 2, 2, 2]

factorizations up to now: #6

```
111 -> [3, 37]
Apr 28, 2017, 17:59:52
672 -> [2, 2, 2, 2, 2, 3, 7]
Apr 28, 2017, 18:00:01
6721 -> [11, 13, 47]
Apr 28, 2017, 18:00:04
895 -> [5, 179]
Apr 28, 2017, 18:00:09
89511 -> [3, 29837]
Apr 28, 2017, 18:00:12
256 -> [2, 2, 2, 2, 2, 2, 2, 2]
Apr 28, 2017, 18:00:17
```

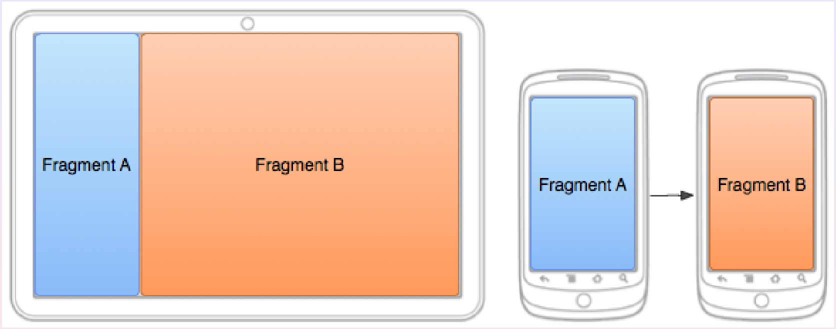
Why Tablets Are Different

- plenty of room
- uncomfortable widget positions
- more functions expected

Solutions

- ship two versions of the app ⇒ maintanance headache
- let activities behave differently on different configurations ⇒ spaghetti code
- split UI screens into composable and redefinable fragments

The Master/Detail Approach



- in a tablet, there is an activity that always contains two fragments
- in a phone, there is an activity that contains a swappable fragment

Hence There Is a Single Activity Now

```
public class MainActivity extends Activity {

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

But there are two layout files:

phone	tablet
res/layout/activity_main.xml	res/layout_large/activity_main.xml

Note that the activity is not a view of the MVC triple anymore

The Phone Layout: res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="it.univr.android.factorizerclient.view.MainActivity">

  <it.univr.android.factorizerclient.view.PhoneView
     android:id="@+id/phone_view"
     android:layout_width="match_parent"
     android:layout_height="match_parent" />

</FrameLayout>
```

A PhoneView is a custom widget that will host a swappable fragment

The Tablet Layout: `res/layout-large/activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout ...
  <it.univr.android.factorizerclient.view.TabletView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <fragment android:id="@+id/factorizer_fragment"
      android:name="it.univr.android.factorizerclient.view.FactorizerFragment"
      android:layout_width="0dp"
      android:layout_height="match_parent" android:layout_weight="2" />

    <fragment android:id="@+id/history_fragment"
      android:name="it.univr.android.factorizerclient.view.HistoryFragment"
      android:layout_width="0dp"
      android:layout_height="match_parent" android:layout_weight="3" />

  </it.univr.android.factorizerclient.view.TabletView>
</FrameLayout>
```

A `TabletView` is a custom widget that hosts two fragments

How to Define a Custom Widget: TabletView 1/4

A custom widget can be defined by subclassing a widget class:

```
public class TabletView extends LinearLayout implements View {

  /**
   * These two constructors must exist to let the view be recreated at
   * configuration change or inflated from XML.
   */

  public TabletView(Context context) {
    super(context);
  }

  public TabletView(Context context, AttributeSet attrs) {
    super(context, attrs);
  }
```

How to Define a Custom Widget: TabletView 2/4

The two fragments are statically wired at two fixed identifiers:

```
private FragmentManager getFragmentManager() {
  return ((Activity) getContext()).getFragmentManager();
}

// an AbstractFragment is a superinterface of both kinds of fragments
private AbstractFragment getFactorizerFragment() {
  return (AbstractFragment) getFragmentManager()
    .findFragmentById(R.id.factorizer_fragment);
}

private AbstractFragment getHistoryFragment() {
  return (AbstractFragment) getFragmentManager()
    .findFragmentById(R.id.history_fragment);
}
```

How to Define a Custom Widget: TabletView 3/4

The widget is attached and detached from the MVC triple

```
private MVC mvc;

@Override
protected void onAttachedToWindow() {
  super.onAttachedToWindow();
  mvc = ((FactorizerApplication) getContext().getApplicationContext()).getMVC();
  mvc.register(this);
}

@Override
protected void onDetachedFromWindow() {
  mvc.unregister(this);
  super.onDetachedFromWindow();
}
```

How to Define a Custom Widget: TabletView 4/4

The custom widget implements the MVC `View` interface:

```
@Override
public void onModelChanged() {
  // delegation to both fragments
  getFactorizerFragment().onModelChanged();
  getHistoryFragment().onModelChanged();
}

@Override
public void showHistory() {
  // nothing to do, this widget always shows history
}
```

Another Custom Widget: PhoneView 1/4

A custom widget can be defined by subclassing a widget class:

```
public class PhoneView extends FrameLayout implements View {

  /**
   * These two constructors must exist to let the view be recreated at
   * configuration change or inflated from XML.
   */

  public PhoneView(Context context) {
    super(context);
  }

  public PhoneView(Context context, AttributeSet attrs) {
    super(context, attrs);
  }
```

Another Custom Widget: PhoneView 2/4

The only fragment is dynamically bound at a fixed identifier:

```
private FragmentManager getFragmentManager() {
  return ((Activity) getContext()).getFragmentManager();
}

// an AbstractFragment is a superinterface of both kinds of fragments
private AbstractFragment getFragment() {
  return (AbstractFragment) getFragmentManager()
    .findFragmentById(R.id.phone_view);
}
```

Another Custom Widget: PhoneView 3/4

The widget is attached and detached from the MVC triple

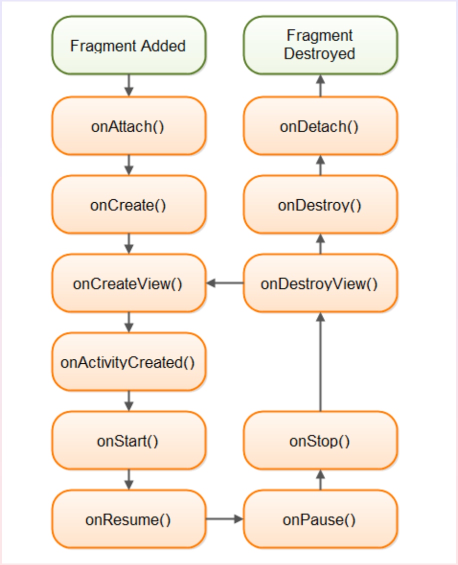
```
private MVC mvc;

@Override
protected void onAttachedToWindow() {
  super.onAttachedToWindow();
  mvc = ((FactorizerApplication) getContext().getApplicationContext()).getMVC();
  mvc.register(this);

  // at the beginning, show the factorizer fragment
  if (getFragment() == null)
    getFragmentManager().beginTransaction()
      .add(R.id.phone_view, new FactorizerFragment())
      .commit();
}

@Override
protected void onDetachedFromWindow() {
  mvc.unregister(this);
  super.onDetachedFromWindow();
}
```

Another Custom Widget: PhoneView 4/4

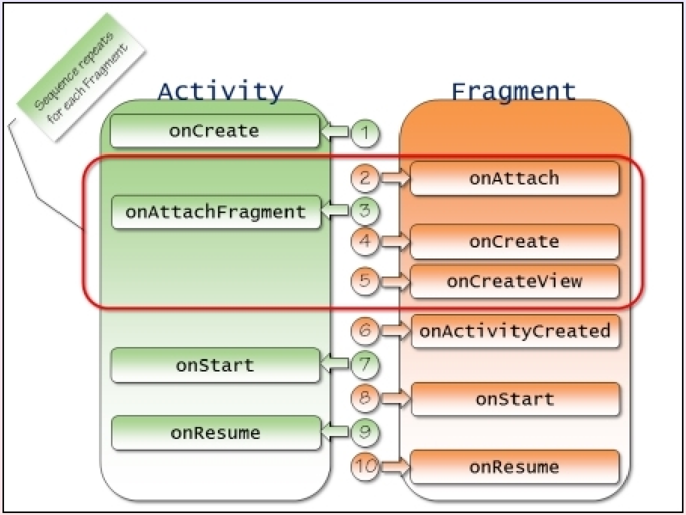
The custom widget implements the MVC View interface:

```
@Override
public void onModelChanged() {
  // delegation to its only fragment
  getFragment().onModelChanged();
}

@Override
public void showHistory() {
  // if required to show the history, replaces
  // the only fragment with a new HistoryFragment
  getFragmentManager().beginTransaction()
    .replace(R.id.phone_view, new HistoryFragment())
    .addToBackStack(null)
    .commit();
}
```

Fragments Have a Lifecycle



Activity and Fragment Lifecycles Are Related



The FactorizerFragment 1/4

Most of its code has been copied from the old FactorizerActivity

```
public class FactorizerFragment extends Fragment implements AbstractFragment {
  private final static String TAG = FactorizerFragment.class.getName();
  private MVC mvc;
  private TextView factorization;
  private TextView counter;
  private EditText insertNumber;
  private Button send;

  // view state
  private boolean isCountOn;
  private int factorizationsCount;

  @Override @UiThread
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true); // this fragment uses menus
  }
```

The FactorizerFragment 2/4

```
// called when its widgets must be created
@Override @UiThread
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedI
  View view = inflater.inflate(R.layout.fragment_factorizer, container, false);
  factorization = (TextView) view.findViewById(R.id.factorization);
  counter = (TextView) view.findViewById(R.id.counter);
  insertNumber = (EditText) view.findViewById(R.id.insert_number);
  send = (Button) view.findViewById(R.id.send_number);
  send.setOnClickListener(__ -> factorize());

  if (savedInstanceState != null) {
    isCountOn = savedInstanceState.getBoolean(TAG + "isCountOn");
    factorizationsCount = savedInstanceState.getInt(TAG + "factorizationsCount");
  }

  return view;
}

@Override @UiThread
public void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  outState.putBoolean(TAG + "isCountOn", isCountOn);
  outState.putInt(TAG + "factorizationsCount", factorizationsCount);
}
```

The FactorizerFragment 3/4

```
// called when the parent activity is ready
@Override @UiThread
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
  super.onActivityCreated(savedInstanceState);
  // we can safely call getActivity() here
  mvc = ((FactorizerApplication) getActivity().getApplication()).getMVC();
  onModelChanged(); // force redraw at start-up
}

@Override @UiThread
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
  super.onCreateOptionsMenu(menu, inflater);
  inflater.inflate(R.menu.fragment_factorizer, menu);
}
```

phone	tablet
res/menu/fragment_factorizer.xml	res/menu_large/fragment_factorizer.xml
two menu items	one menu item (no show history)

The FactorizerFragment 4/4

```
@Override @UiThread
public boolean onOptionsItemSelected(MenuItem item) {
  if (item.getItemId() == R.id.menu_item_show_counter) {
    isCountOn = !isCountOn; updateCounter(); return true;
  }
  else if (item.getItemId() == R.id.menu_item_show_last_factorizations) {
    mvc.controller.showHistory(); return true;
  }
  else
    return super.onOptionsItemSelected(item);
}

@Override @UiThread
public void onModelChanged() {
  Factorization fact = mvc.model.getLastFactorization();
  if (fact != null) {
    factorization.setText(getResources().getString
      (R.string.factorization_message,
        fact.getFactorizedNumber(), Arrays.toString(fact.getFactors())));
    send.setEnabled(true);
    updateCounter();
  }
}
```

The HistoryFragment

```
public class HistoryFragment extends ListFragment implements AbstractFragment {
  private MVC mvc;

  @Override @UiThread
  public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    mvc = ((FactorizerApplication) getActivity().getApplication()).getMVC();
    onModelChanged(); // force redraw at start-up
  }

  @Override @UiThread
  public void onModelChanged() {
    ArrayAdapter<Factorization> adapter = new ArrayAdapter<>
      (this.getActivity(), android.R.layout.simple_list_item_1,
       mvc.model.getFactorizations());
    setListAdapter(adapter);
  }
}
```

The Controller Must React to showHistory() Now

```
public class Controller {
  private MVC mvc;

  public void setMVC(MVC mvc) {
    this.mvc = mvc;
  }

  @UiThread
  public void factorize(Context context, BigInteger n) {
    FactorizationService.factorize(context, n);
  }

  @UiThread
  public void showHistory() {
    // delegation to all registered views
    mvc.forEachView(View::showHistory);
  }
}
```

git checkout master_detail

Switch to tag `master_detail` to see the application code as it is up to this point:

```
git checkout master_detail
```

Later, you can switch to other tags or come back to the latest version of the code:

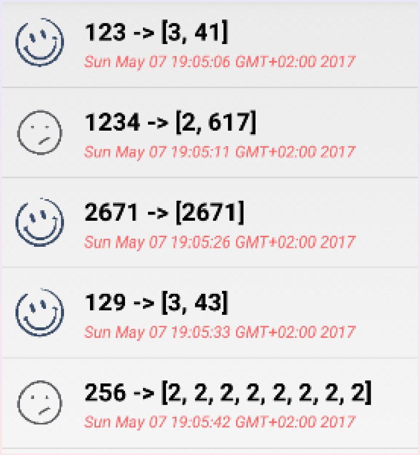
```
git checkout master
```

Exercise

A Flickr Client in Android

Modify your Flickr client so that it uses two fragments: one for the search form and another for the list of pictures. Use the master/detail approach, in order to show both fragments together on a tablet

Customizing the `HistoryFragment`'s View



123 -> [3, 41]
Sun May 07 19:05:06 GMT+02:00 2017

1234 -> [2, 617]
Sun May 07 19:05:11 GMT+02:00 2017

2671 -> [2671]
Sun May 07 19:05:26 GMT+02:00 2017

129 -> [3, 43]
Sun May 07 19:05:33 GMT+02:00 2017

256 -> [2, 2, 2, 2, 2, 2, 2, 2]
Sun May 07 19:05:42 GMT+02:00 2017

Since the same `HistoryFragment` is used for phone and tablet, this modification will have effect in both cases

We Need a Custom Adapter

In `HistoryFragment.java`:

```
@Override @UiThread
public void onModelChanged() {
  setListAdapter(new HistoryAdapter());
}

private class HistoryAdapter extends ArrayAdapter<Factorization> {
  ...
}
```

Each Item: res/layout/fragment_history_item.xml

```
<LinearLayout ...
  android:orientation="horizontal"
  android:layout_width="match_parent" android:layout_height="match_parent"
  android:padding="10dp">

  <ImageView android:id="@+id/icon"
    android:layout_width="36dp" android:layout_height="36dp" ... />

  <LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <TextView android:id="@+id/factorization"
      android:layout_width="match_parent" android:layout_height="wrap_content"
      android:textStyle="bold" android:textSize="18dp"
      android:textColor="#000000" ... />

    <TextView android:id="@+id/when"
      android:layout_width="match_parent" android:layout_height="wrap_content"
      android:textStyle="italic" android:textSize="12dp"
      android:textColor="#ff6666" ... />

  </LinearLayout>
</LinearLayout>
```

Creation of the Adapter's Views

```
private class HistoryAdapter extends ArrayAdapter<Factorization> {
 private final Factorization[] factorizations = mvc.model.getFactorizations();

 private HistoryAdapter() {
  super(getActivity(),R.layout.fragment_history_item,mvc.model.getFactorizations())
 }

 @Override
 public View getView(int position, View convertView, ViewGroup parent) {
  View row = convertView;
  if (row == null) { // we cannot recycle a preview list item
   LayoutInflater inflater = getActivity().getLayoutInflater();
   row = inflater.inflate(R.layout.fragment_history_item, parent, false);
  }
  Factorization fact = factorizations[position];
  ((ImageView) row.findViewById(R.id.icon)).setImageResource
   (fact.getFactorizedNumber().getLowestSetBit() == 0 ?
    R.drawable.even : R.drawable.odd);
  ((TextView) row.findViewById(R.id.factorization)).setText
   (fact.getFactorizedNumber() + " -> " + Arrays.toString(fact.getFactors()));
  ((TextView) row.findViewById(R.id.when)).setText(fact.getWhen().toString());
  return row;
 }
}
```

git checkout custom_item

Switch to tag `custom_item` to see the application code as it is up to this point:

```
git checkout custom_item
```

Later, you can switch to other tags or come back to the latest version of the code:

```
git checkout master
```

Exercises

A Flickr Client in Android, with Custom List Items

Modify your Flickr client so that the list of images is shown with a custom adapter. Namely, the title of the picture should be in boldface and the URL should be below, small and in italic

A Flickr Client in Android, with Image Preview

Modify your Flickr client so that the list of images reports a preview on its left, that is, a small preview image, at low-resolution. The image must be downloaded from the Flickr site. For that, you need to ask for the `url_s` extra in the query sent to the web service and access the URL reported in the reply

You cannot download the image in `getView()`

Ask the controller, instead. Once it will have downloaded the picture, it will modify the model and this will trigger a new `onModelChanged()` event