XebiCon 2013

Clean Code in Android Apps

It's just Java, right?

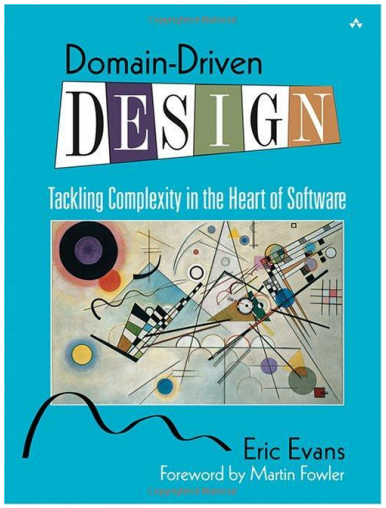
Well, kind of...

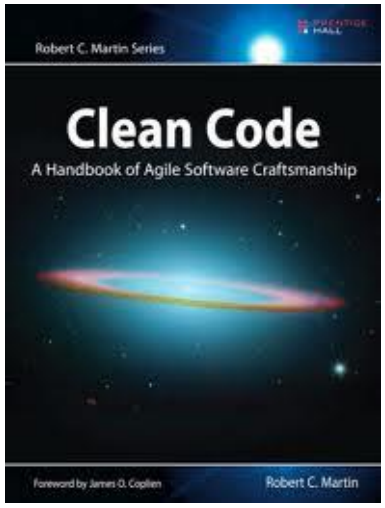
Conventional Wisdom still valid...

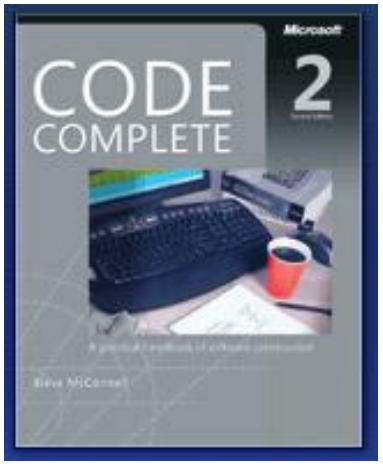
SOLID principles

Don't Repeat Yourself

Keep It Simple, Stupid







...but question your assumptions

Your app can outlive its Linux process.

Your static variables will vanish.

The GC hits much harder than on the desktop.

Devices are varied and, occasionally, broken.

Creating Intents

Don't scatter string references everywhere:

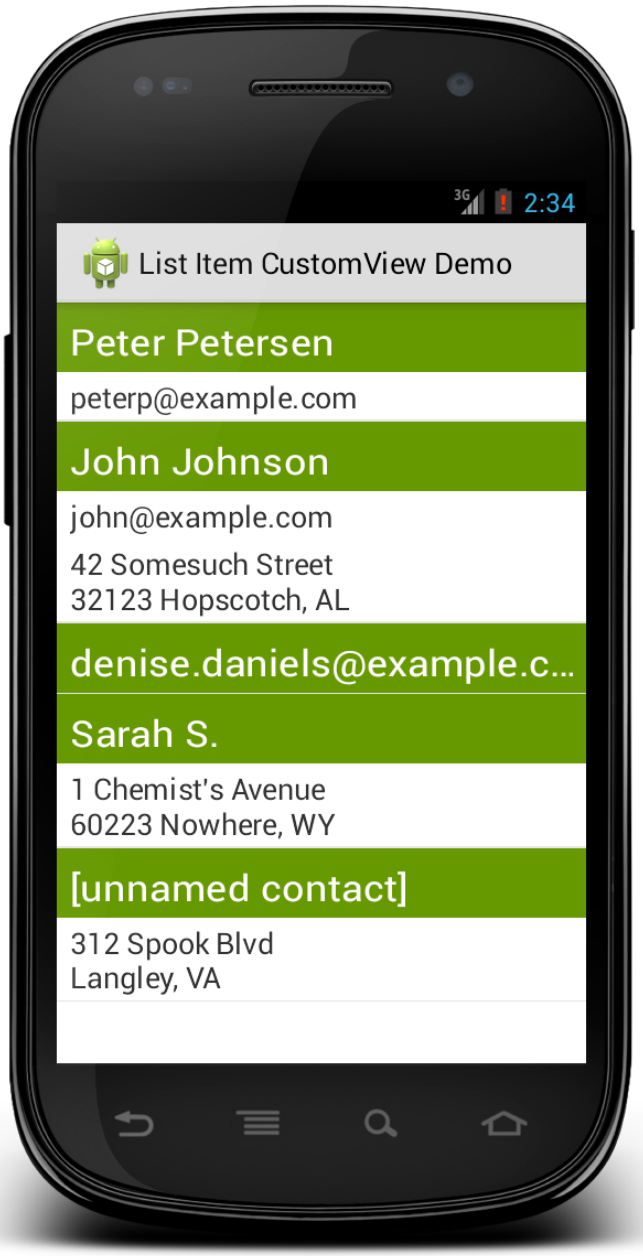
```
// in calling Activity
public void onClick( View v ){
   Intent it = new Intent(Target.class);
   it.putExtra("name", "Joe");
   it.putExtra("send_enabled", true);

   this.startActivity(it);
}
```

Use a static factory method in target Activity:

```
public static Intent newIntent(
     String arg1, boolean arg2 ){
   Intent it = new Intent(Target.class);
   it.putExtra("name", arg1);
   it.putExtra("send_enabled", arg2);
   return it;
}

// in class CallingActivity
public void onClick( View v ){
   startActivity(
     TargetActivity.newIntent("Joe", true)
   );
}
```

Custom Views

Use Custom Views to to consistently display domain objects

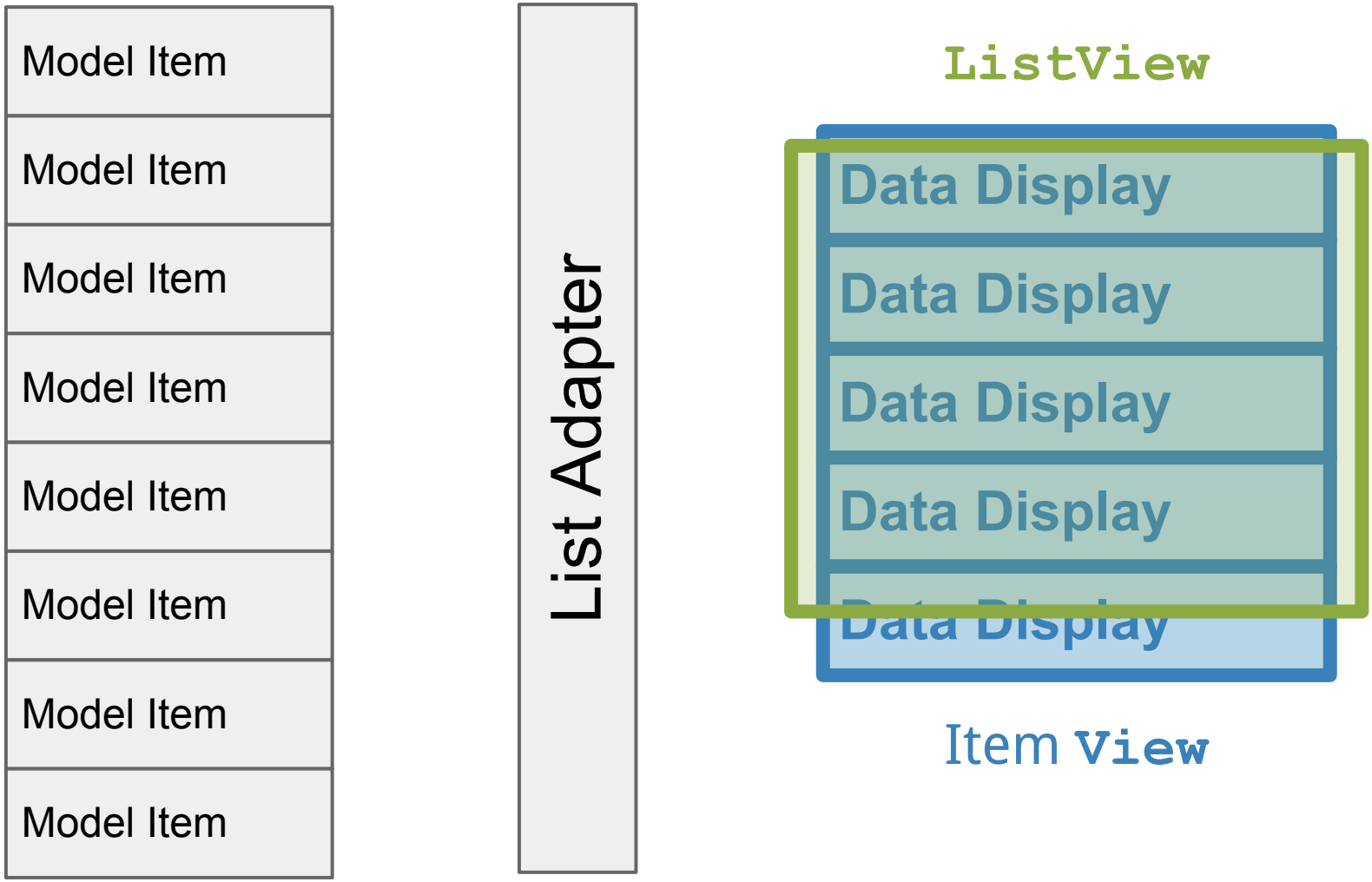

2:34
List Item CustomView Demo
Peter Petersen
peterp@example.com
John Johnson
john@example.com
42 Somesuch Street
32123 Hopscotch, AL
denise.daniels@example.c…
Sarah S.
1 Chemist's Avenue
60223 Nowhere, WY
[unnamed contact]
312 Spook Blvd
Langley, VA

Main Layout XML

<ListView

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://.../android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ListView android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</FrameLayout>
```

ListView Quick Recap



Model class

```
public class Contact {
   private final String name;
   private final String email;
   private final String address1;
   private final String address2;

   // Constructor and getters
}
```

Item View XML


```
<?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">

    <TextView
      android:id="@+id/contact_name"
      style="@style/contact_title_field">
      </TextView>

    <TextView
      android:id="@+id/contact_email"
      style="@style/contact_detail_field">
      </TextView>

    <TextView
      android:id="@+id/contact_address"
      style="@style/contact_detail_field">
      </TextView>
</LinearLayout>
```

ContactListAdapter.java

```
public class ContactListAdapter extends BaseAdapter {

    public View getView(
        int position, View convertView, ViewGroup parent) {

        final Contact item = getItem(position);
        final View view = (convertView == null)
          ? inflater.inflate(R.layout.list_item, null)
          : convertView;

        TextView nameView = ((TextView) view.findViewById(R.id.contact_name));
        if (item.getName() != null) {
            nameView.setText(item.getName());
        } else if (item.getEmail() != null) {
            nameView.setText(item.getEmail());
        } else {
            nameView.setText(R.string.unidentified);
        }

        TextView emailView = (TextView) view.findViewById(R.id.contact_email);
        if (item.getEmail() != null) {
             emailView.setText(item.getEmail());
             emailView.setVisibility(item.getName() == null ? View.GONE : View.VISIBLE);
        } else {
             emailView.setVisibility(View.GONE);
        }

        TextView addressView = (TextView) view.findViewById(R.id.contact_address);
        if (item.getAddressLines() != null) {
             addressView.setText(item.getAddressLines());
             addressView.setVisibility(View.VISIBLE);
        } else {
             addressView.setVisibility(View.GONE);
        }

        return view;
    }
}
```

public View getView(int, View, ViewGroup)

```
public class ContactListAdapter extends BaseAdapter {

    public View getView(
        int position, View convertView, ViewGroup parent) {

        final Contact item = getItem(position);
        final View view = (convertView == null)
          ? inflater.inflate(R.layout.list_item, null)
          : convertView;

        TextView nameView = ((TextView) view.findViewById(R.id.contact_name));
        if (item.getName() != null) {
            nameView.setText(item.getName());
        } else if (item.getEmail() != null) {
            nameView.setText(item.getEmail());
        } else {
            nameView.setText(R.string.unidentified);
        }

        TextView emailView = (TextView) view.findViewById(R.id.contact_email);
        if (item.getEmail() != null) {
             emailView.setText(item.getEmail());
             emailView.setVisibility(item.getName() == null ? View.GONE : View.VISIBLE);
        } else {
             emailView.setVisibility(View.GONE);
        }

        TextView addressView = (TextView) view.findViewById(R.id.contact_address);
        if (item.getAddressLines() != null) {
             addressView.setText(item.getAddressLines());
             addressView.setVisibility(View.VISIBLE);
        } else {
             addressView.setVisibility(View.GONE);
        }

        return view;
    }
}
```

Create or recycle view

Fill "name" field

Fill "email" field

Fill "address" field

Return view

Create or Recycle View

```
public View getView( int position,
          View convertView, ViewGroup parent) {

       final Contact item = getItem(position);

       final View view = (convertView == null)
       ? inflater.inflate(R.layout.list_item, null)
       : convertView;
```

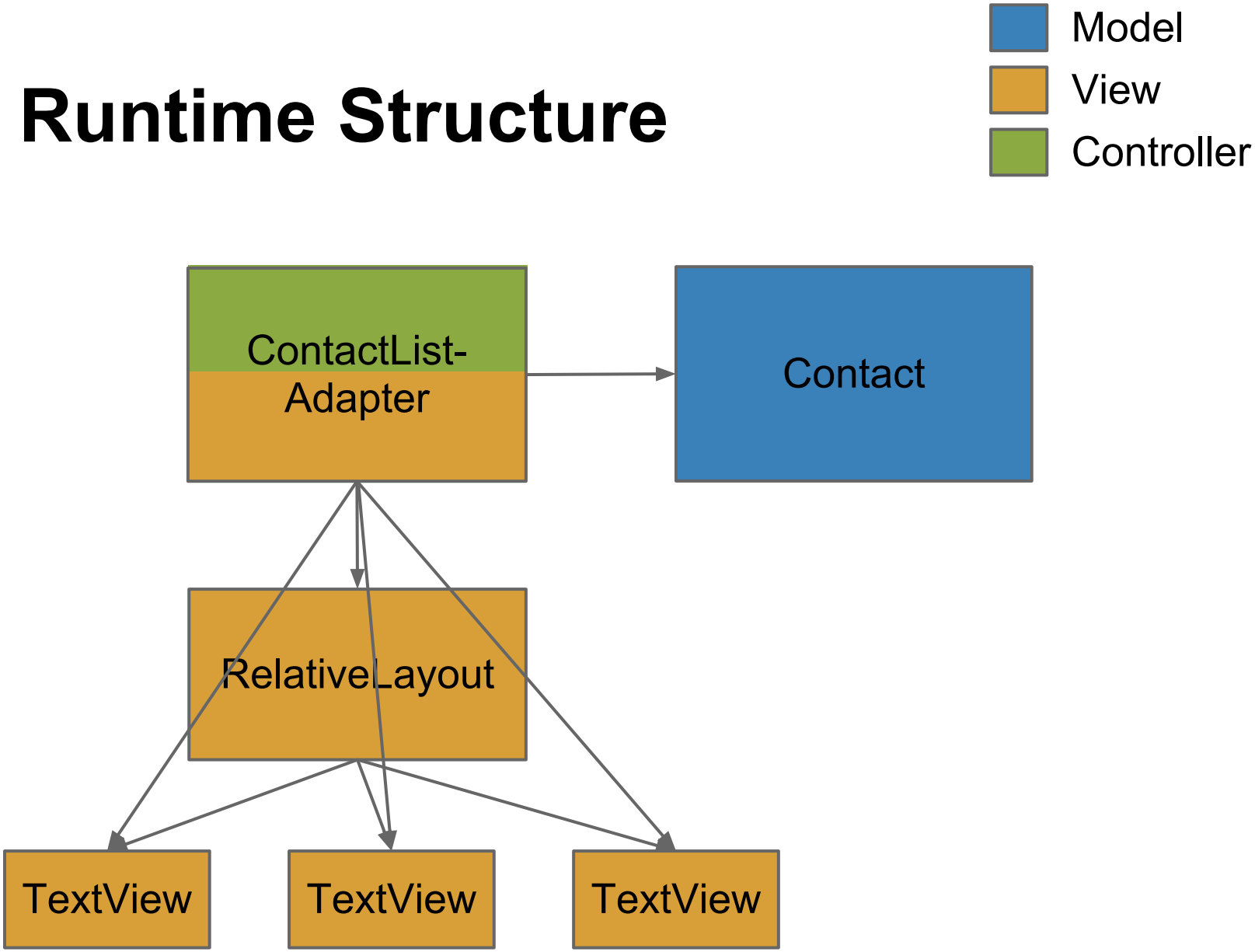
Fill "Name" field

```
TextView nameView = ((TextView)
      view.findViewById( R.id.contact_name ));

if (item.getName() != null) {
   nameView.setText(item.getName());
} else if (item.getEmail() != null) {
   nameView.setText(item.getEmail());
} else {
   nameView.setText(R.string.unidentified);
}

 // Continued
```

Fill "email" field

```
TextView emailView = (TextView)
      view.findViewById(R.id.contact_email);

if (item.getEmail() != null) {
   emailView.setText(item.getEmail());
   emailView.setVisibility(
        item.getName() == null
        ? View.GONE : View.VISIBLE);
} else {
   emailView.setVisibility(View.GONE);
}
```

Runtime Structure



ViewHolder Pattern?

What about the ViewHolder pattern?

```
public class ViewHolder {
    public final TextView nameView;
    public final TextView emailView;
    public final TextView addressView;

    public ViewHolder( View v ) {
        nameView = (cast) v.findViewById(R.id.name);
        emailView = (cast) v.findVBI(R.id.email);
        addressView = (cast) v.findVBI(R.id.addr);

        v.setTag( this );
    }

}
```

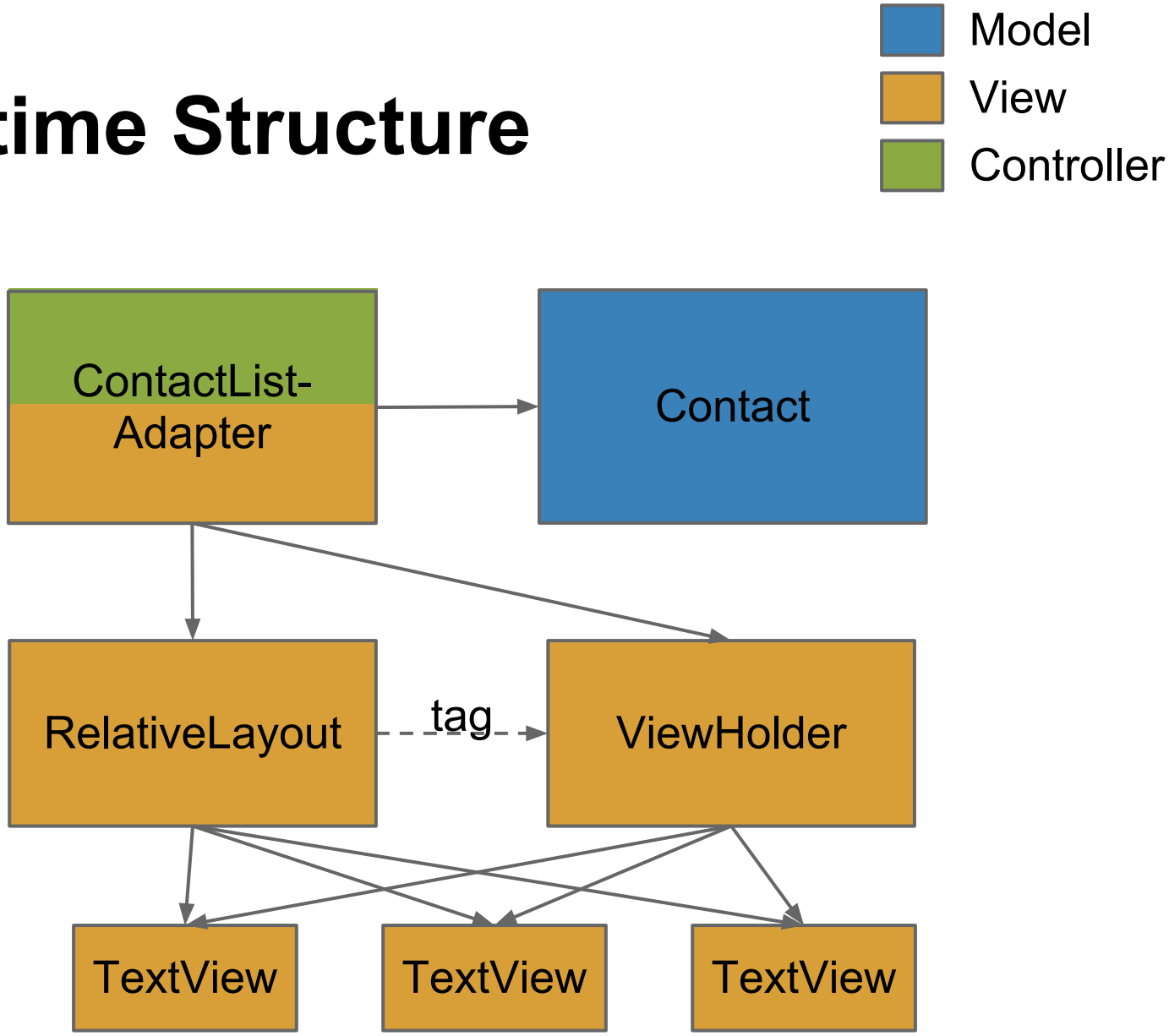
Create or Recycle View

```
public View getView( int position,
          View convertView, ViewGroup parent) {

    final ViewHolder holder;
    final Contact item = getItem(position);


    if (convertView == null {
        holder = new ViewHolder(
        inflater.inflate(
                R.layout.list_item, null));
    } else {
        holder = (cast) convertView.getTag();
    }
```

Fill "Name" field

```
if (item.getName() != null) {
    holder.nameView.setText( item.getName() );

} else if (item.getEmail() != null) {
    holder.nameView.setText( item.getEmail() );

} else {
    holder.nameView.setText(
        R.string.unidentified);
}
```

Runtime Structure



Can we do better?

Create or Recycle, and init, view

```
public View getView( int position,
          View convertView, ViewGroup parent) {

       final Contact item = getItem(position);

       final ItemView view = (ItemView)
           (convertView == null)
           ? inflater.inflate(R.layout.list_item)
           : convertView;
       view.show( item );

       return view;
}
```

Item View, original

<LinearLayout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.androi
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">

    <TextView
      android:id="@+id/contact_name"
      style="@style/contact_title_field">
      </TextView>

    <TextView
      android:id="@+id/contact_email"
      style="@style/contact_detail_field">
      </TextView>

    <TextView
      android:id="@+id/contact_address"
      style="@style/contact_detail_field">
      </TextView>
</LinearLayout>
```

Item View, new

<com.myapp.ItemView

```
<?xml version="1.0" encoding="utf-8"?>
<com.myapp.ItemView xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

    <TextView
      android:id="@+id/contact_name"
      style="@style/contact_title_field">
      </TextView>

    <TextView
      android:id="@+id/contact_email"
      style="@style/contact_detail_field">
      </TextView>

    <TextView
      android:id="@+id/contact_address"
      style="@style/contact_detail_field">
      </TextView>
</com.myapp.ItemView>
```

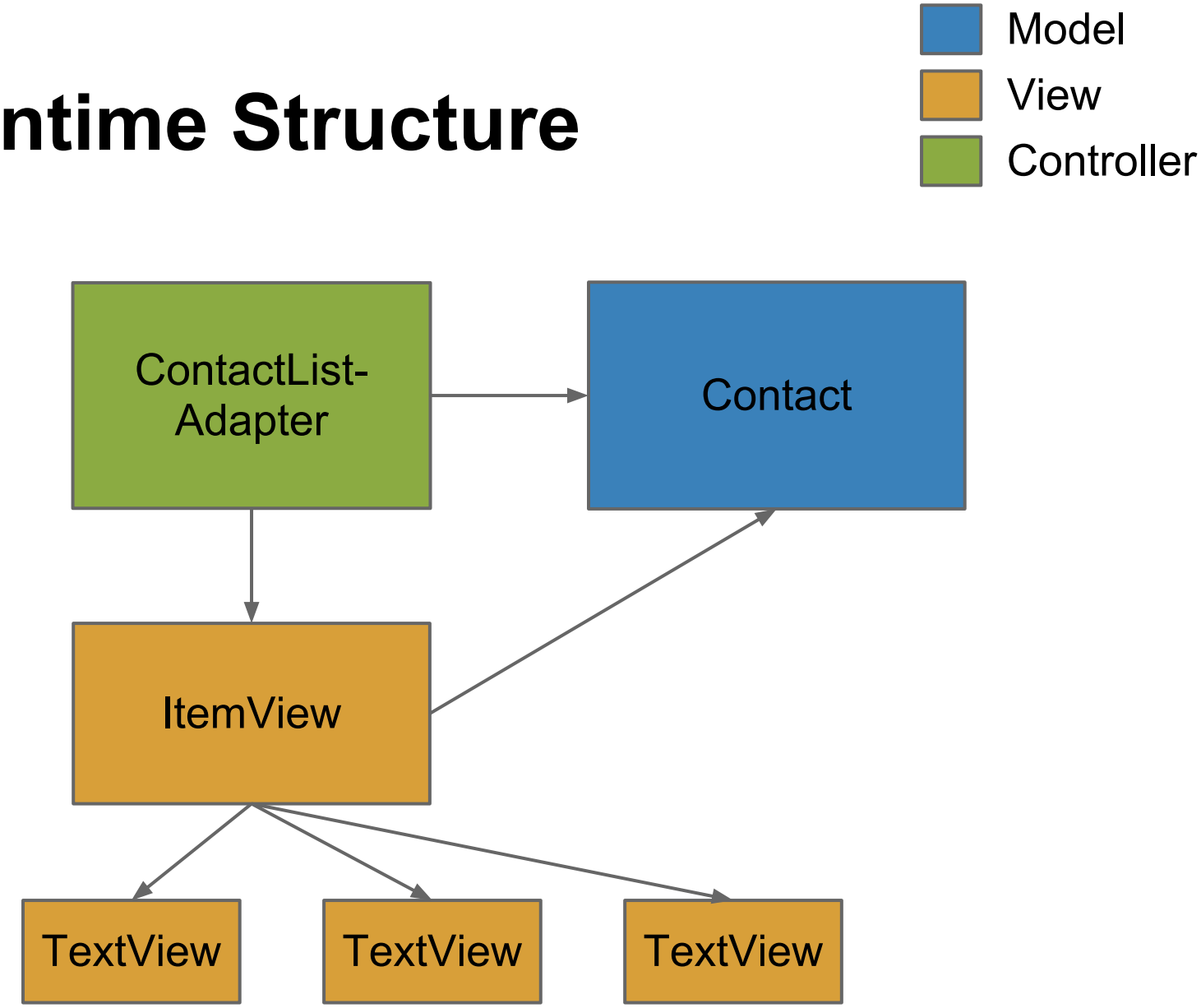
ItemView.java

```
package com.myapp;
public class ItemView extends LinearLayout {

    private TextView nameView;
    private TextView emailView;
    private TextView addressView;

    // expose superclass constructors (3)

    @Override
    protected void onFinishInflate() {
      super.onFinishInflate();
      nameView = (cast) findViewById(R.id.name);
      emailView = (cast) findViewById(R.id.email);
      addressView = (cast) findVBI(R.id.address);
   }
```

ItemView.java

```
@Override
public void show( Contact item ) {
    String name = item.getName();
    String email = item.getEmail();

    if (name != null) {
        nameView.setText(name);
    } else if (email != null) {
        nameView.setText(email);
    } else {
        nameView.setText(R.string.unidentified);
    }

    // etcetera
}
```

Runtime Structure



Not just composite views


07:24
Rekeningen
Betaalrekeningen
Saldo
Hr A. van Dijk
NL20 INGB 0001 2345 67
€733,04
Spaarrekeningen
Saldo
Spaarrekening
NL20 INGB 0001 2345 67
€555,90
Creditcards
Creditcard
HR A. VAN DIJK
kaartnr. 1234 **** **** 1234

Multiple Screens

One for all, all for one

Support multiple screen sizes while maintaining DRY principle.

How to reuse layouts:

-Fragments (self-contained)

-Use a Callback interface

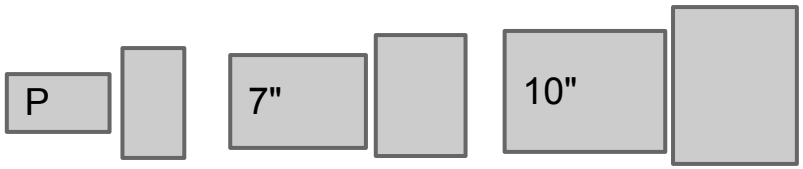
-Make use of resource qualifiers

```java
public class AccountListFragment extends ListFragment{

    public static interface Callback {
        void onAccountSelected(final Account account);
    }

    public static AccountListFragment newInstance(final String param) {
        AccountListFragment fragment =new AccountListFragment();
        Bundle args = new Bundle();
        args.putString("param1", param);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onAttach(Activity activity) {
        this.callback = (Callback) activity;
    }

    @Override
    public View onCreateView(...){
         return inflater.inflate(R.layout.account_fragment, null);
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        String param1 = getArguments().getString("param1");
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id){
       Account account = getListAdapter().getItem(position);
       callback.onAccountSelected(account);
    }
```

Resource Qualifiers

```
<TextView
    android:layout_marginLeft="@dimen/ui_edge_gap"
    android:maxLength="@integer/company_id_maxlength"
    android:text="@string/contactpicker"
    android:background="@color/list_row_background"
    style="@style/InlineError"/>
```

Resource Qualifiers



res/layout/login.xml

res/layout-xlarge/login.xml

res/layout-sw600dp/login.xml

res/layout-land-xlarge/login.xml

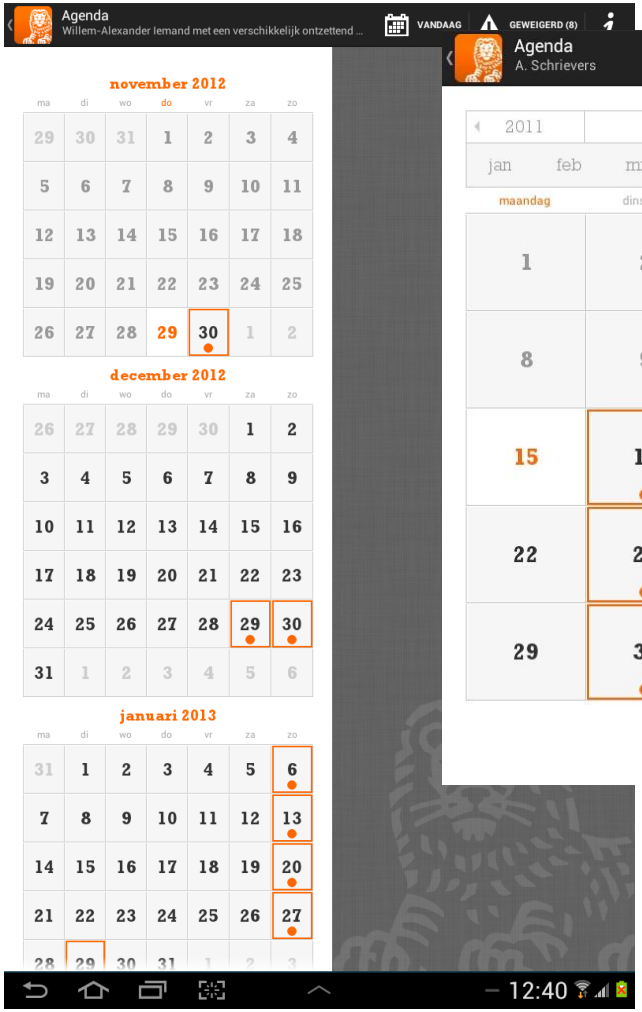
res/layout-port/login.xml

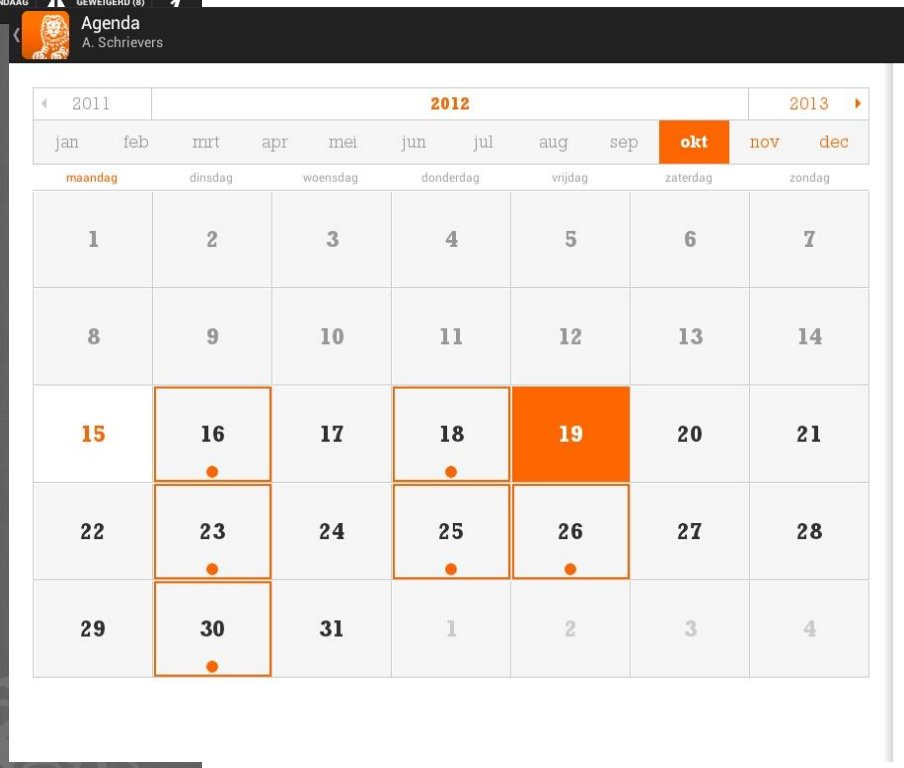
res/values-sw720dp/dimensions.xml

res/values-XXX/colors.xml ..

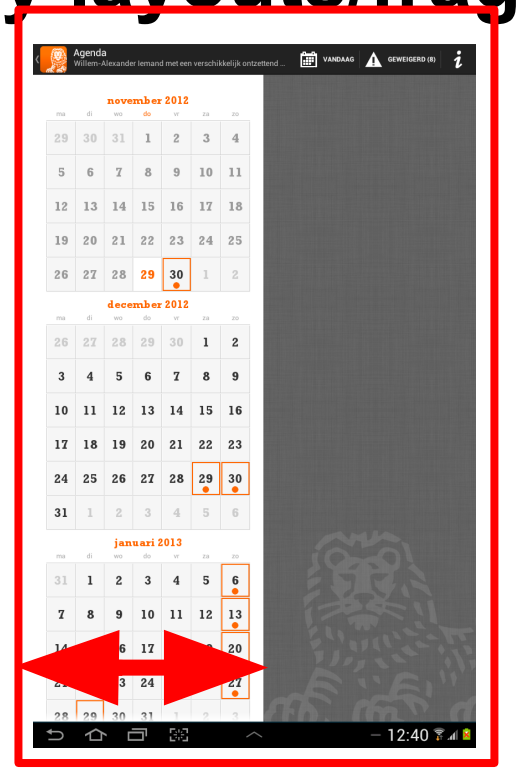
res/values-XXX/styles.xml ..

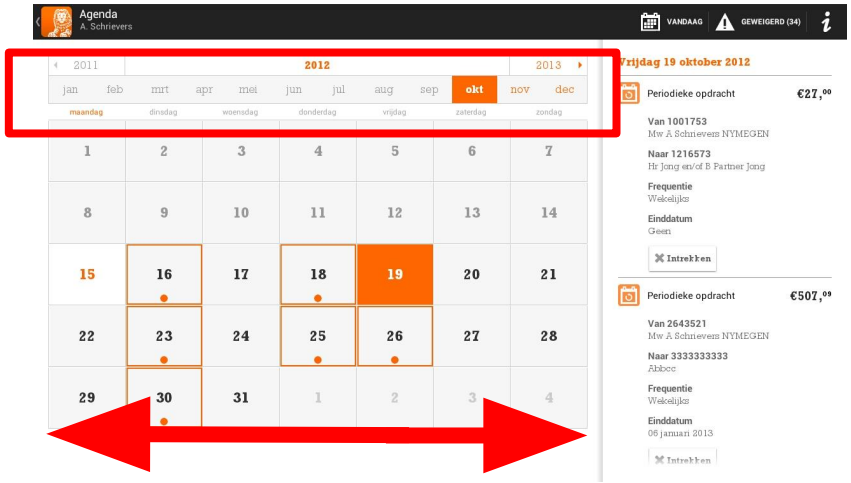
How many layouts/fragments/activities?





How many layouts/fragments/activities?





1 activity: AgendaActivity.java
1 fragment: DatePickerFragment.java

3 layouts: res/layout/agendaActivity.xml //simple: only a placeholder for fragment
res/layout-xlarge/agendaActivity.xml //simple: only a 2 panel holder for fragments
res/layout/**datepicker.xml** //this one is complex!
2 dimensions: res/values-xlarge-land/dimension.xml //only a width
res/values-xlarge-port/dimension.xml //only a width
1 extra style: res/values-xlarge-land /style.xml //visibilty of extra control in datepicker.xml

Resource naming

- drawable resources: images and selectors
  icon_foo_enabled.png
  icon_foo_disabled.png
  icon_foo.xml
- layouts, styles and values: context, subject
  dimen/global__screen_edge_margin
  string/login__error_invalid_credentials
  style/login__password

Pick a pattern and be consistent

Summary

Use static factory methods (intent, fragment)

Use custom views to clarify program structure

Smart reuse of views using qualifiers

ViewGroup Example:
github.com/xebia/xebicon-2013__cc-in-aa

Session Survey:

bit.ly/xc_clean