warwick C<>ding

by Wesley Haigh

Lecture 3

Lists and Objects

Recap

- Java is the programming language used in Android for all of the logic and backend processes.
- Android is the MVC (Model-View-Controller) framework.
- You can get controls in java by calling **findViewById(*id*)**

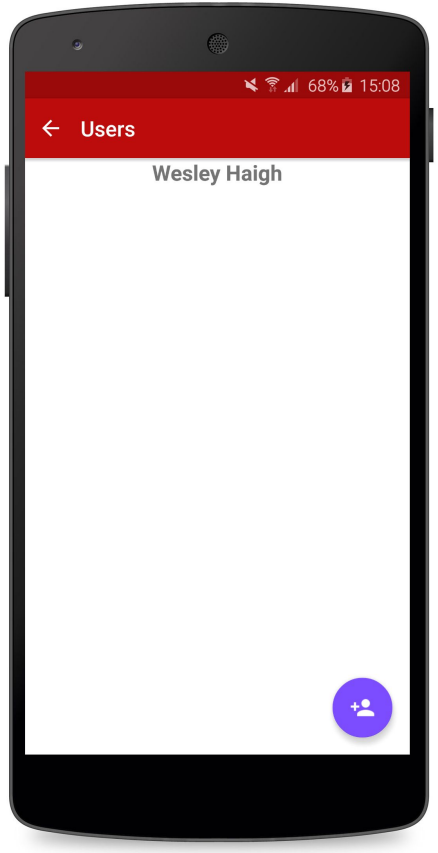
```
Button startButton = (Button) findViewById(R.id.startButton);
```

- You can change and get properties of these controls:

```
_password = (EditText) findViewById(R.id.password);
String inputtedText = _password.getText().toString();
```

Recap Exercise

- Have the MainActivity pass what is typed into the password field.
- Display it in UsersActivity in a TextView at the top.
- To do this you will need to look at adding things to intents and getting things from them too.



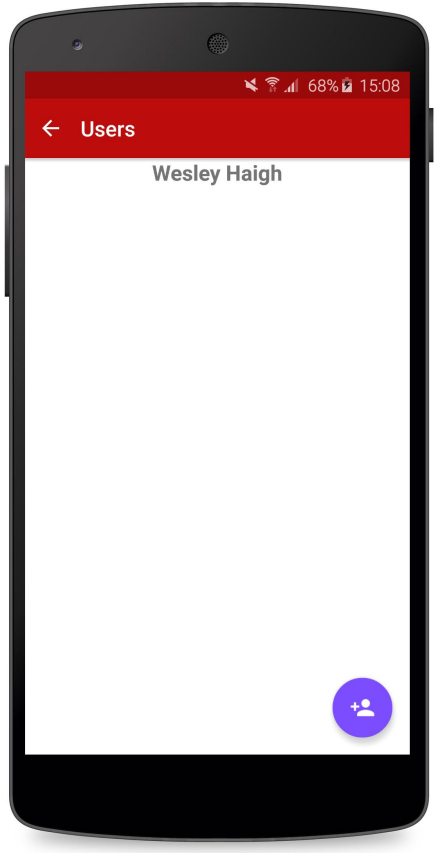
Recap Exercise

- In MainActivity.java

```
Intent intent = new Intent(this, UsersActivity.class);
intent.putExtra("Key", input);
startActivity(intent);
```

- In UsersActivity.java

```
if (getIntent().getExtras() != null) {
    String input = getIntent().getExtras().getString("Key");
}
```



Java Objects

Object Oriented Programming

Model Classes

- Model classes are the core of Object Oriented Programming.
- They are classes that “**Model**” things in the real (or programming) world. For example: could have an animal class that stores the height, weight etc.
- Chapter 4 of my Java tutorial goes into more detail of model classes.

User Class

- We are going to create a User class.
- Create a new **package** in com.warwickcodingapp called **ModelClasses**
- Create a class in that folder called **User**

```
package com.warwickcodingapp.ModelClasses;

/**
 * Created by Wesley on 08/10/2015.
 */
public class User {

}
```

User Class

- Add a “name” property to the class and a getName and setName function.

```
package com.warwickcodingapp.ModelClasses;

public class User {
    private String _name;

    public User() {
    }

    public String getName() {
        return _name;
    }

    public void setName(String name) {
        _name = name;
    }
}
```

User Class

- We can now create User objects and they can have a name.
- Let’s create a user object, give it a name and display it in a TextView in UsersActivity.

```
TextView displayName = (TextView) findViewById(R.id.displayName);
User user = new User();
user.setName("John Smith");
displayName.setText(user.getName());
```

- This creates a User object and sets its name to “John Smith”

Exercise: Objects

- Add the following properties (with the type given) to the User class
  - Age (int)
  - Gender (boolean, true will be male)
  - Email (String)
  - Location (String)
  - Rating (float)
  - Profile Picture (Bitmap)

ArrayLists

- ArrayLists are like arrays for more complex things (like our User objects)
- Defining an arraylist is simple

```
ArrayList<User> users = new ArrayList<>();
```

- Inside the <> you have what it is a list of (eg. here it is a list of User objects)
- ArrayLists are dynamic. They are not fixed in size once they are defined.
- Basic functions of an ArrayList are:
    - .add(*object you want to add*)
    - .get(*index of object*)
    - .remove(*index or the object itself*)

ArrayLists

- ArrayLists are like arrays for more complex things (like our User objects)
- Defining an arraylist is simple

```
ArrayList<User> users = new ArrayList<>();
```

- Inside the <> you have what it is a list of (eg. here it is a list of User objects)
- ArrayLists are dynamic. They are not fixed in size once they are defined.
- Basic functions of an ArrayList are:
    - .add(*object you want to add*)
    - .get(*index of object*)
    - .remove(*index or the object itself*)

ListViews

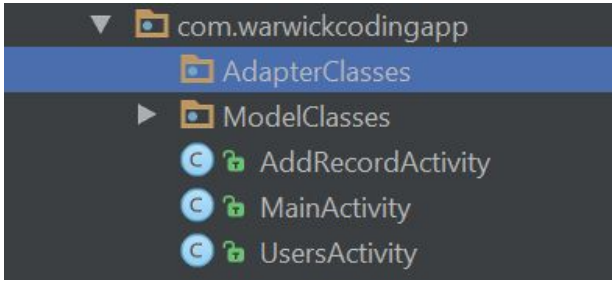
- In Android if you want to display a list of information you use a ListView

```
<ListView
    android:id="@+id/userList"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

- ListViews make use of another xml to define how each item will look
- Create an xml file in the **layout** folder called **listview_user_item.xml**
- Make the layout look like the image below

ListViews

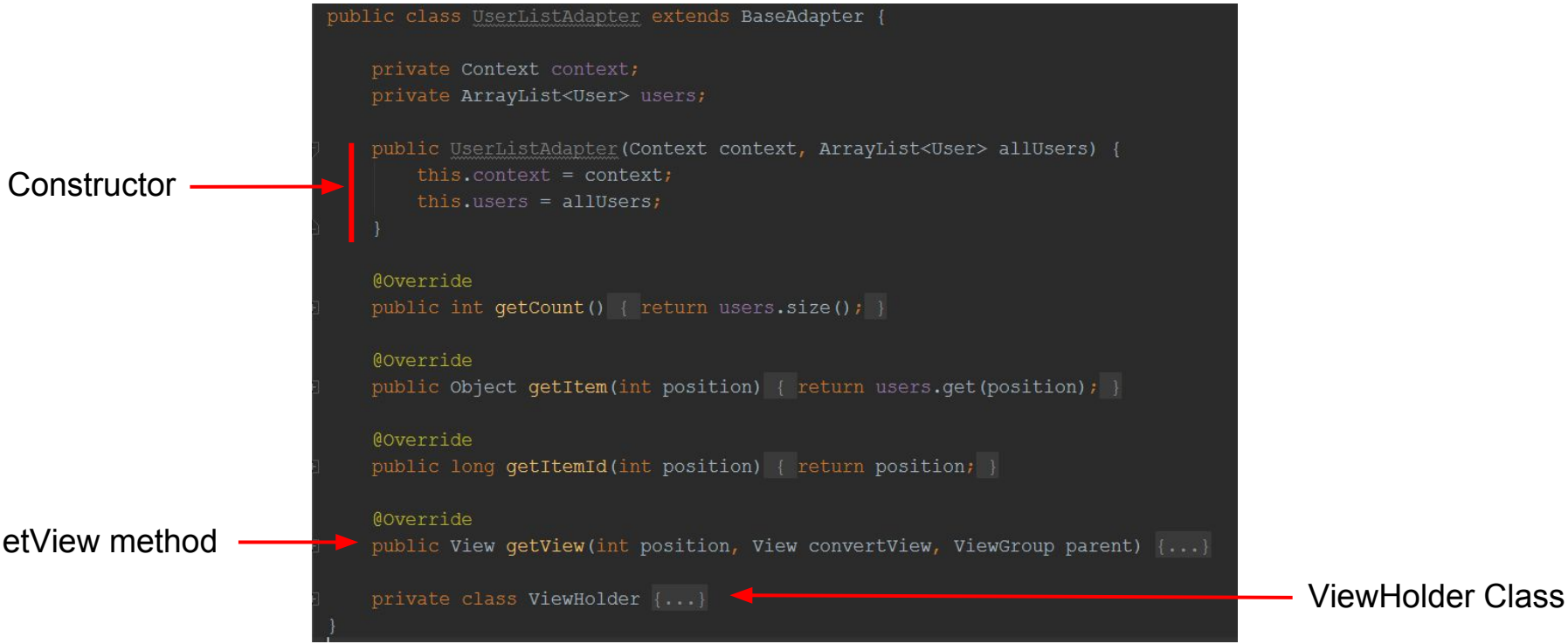
- Along with the layout you need an **Adapter Class** this controls how the list of objects gets translated into the layout.
- Create a package called **AdapterClasses**



- Create a Java file called **UserListAdapter**
- Copy the code from the file **ListAdapter.java** in the Google Drive folder

ListViews

- Let’s dissect this class:




```
public class UserListAdapter extends BaseAdapter {

    private Context context;
    private ArrayList<User> users;

    public UserListAdapter(Context context, ArrayList<User> allUsers) {
        this.context = context;
        this.users = allUsers;
    }

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

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

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

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {...}

    private class ViewHolder {...}
}
```

Adapters: The getView Method

- The getView method is called anytime the listview detects that a new item in it is coming into view.
- Split into 2 parts, “**initialisation**” and “**personalisation**”.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
        convertView = mInflater.inflate(R.layout.listview_user_item, null);
        holder.shopImage = (ImageView) convertView.findViewById(R.id.shopImage);
        holder.price = (TextView) convertView.findViewById(R.id.price);
        holder.coins = (TextView) convertView.findViewById(R.id.coins);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }

    ShopItem s = shopItems.get(position);
    if (s != null) {
        String coins = NumberFormat.getNumberInstance(Locale.US).format(s.getCoins());
        coins = coins.replace(",", " ");
        holder.coins.setText(s.getItemCost());
        holder.price.setText(coins + " " + context.getResources().getString(R.string.shop_textView_coins));
        holder.shopImage.setImageResource(s.getItemImage());
    }

    return convertView;
}
```

- In initialisation you check to see if there is already a view for the adapter to use and if not then you create one.
- In personalisation you set what goes into the view

getView: Initialisation

- if **convertView** is null then
  - create a **ViewHolder**
  - inflate a view and put it in **convertView**
  - get hold of all controls that you will want to personalise and store them in the **ViewHolder**.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
        convertView = mInflater.inflate(R.layout.listview_user_item, null);
        holder.shopImage = (ImageView) convertView.findViewById(R.id.shopImage);
        holder.price = (TextView) convertView.findViewById(R.id.price);
        holder.coins = (TextView) convertView.findViewById(R.id.coins);
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }

    ShopItem s = shopItems.get(position);
    if (s != null) {
        String coins = NumberFormat.getNumberInstance(Locale.US).format(s.getCoins());
        coins = coins.replace(",", " ");
        holder.coins.setText(s.getItemCost());
        holder.price.setText(coins + " " + context.getResources().getString(R.string.shop_textView_coins));
        holder.shopImage.setImageResource(s.getItemImage());
    }

    return convertView;
}
```

ViewHolders

- ViewHolders are a class that keep hold controls for each entry in a list.
- There will be **1** ViewHolder attached to each view in a list.
- It has variables corresponding to all the controls that you will wish to alter
- Why use a ViewHolder?
  - This is an issue of memory and performance
  - Every time an entry in a list comes into view it has to be created on screen, calling findViewById() every time can affect performance.

```
private class ViewHolder {
    TextView price;
    TextView coins;
    ImageView shopImage;
}
```

```
private class ViewHolder {
    TextView name;
    TextView email;
    ImageView profilePicture;
}
```

getView: Initialisation

- Need to change what the LayoutInflator is inflating to **listview_user_item**
- Need to change the relevant names and ids of the views that are stored in the **ViewHolder**

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);

        convertView = mInflater.inflate(R.layout.listview_user_item, null);
        holder.profilePicture = (ImageView) convertView.findViewById(R.id.profilePicture);
        holder.name = (TextView) convertView.findViewById(R.id.name);
        holder.email = (TextView) convertView.findViewById(R.id.email);

        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }

    ShopItem s = shopItems.get(position);
    if (s != null) {
        String coins = NumberFormat.getNumberInstance(Locale.US).format(s.getCoins());
        coins = coins.replace(",", " ");
        holder.coins.setText(s.getItemCost());
        holder.price.setText(coins + " " + context.getResources().getString(R.string.shop_textView_coins));
        holder.shopImage.setImageResource(s.getItemImage());
    }

    return convertView;
}
```

getView: Personalisation

- This is the section that is used to customise how that particular entry in the listview looks.
- Step 1: get the relevant **Object** (eg. a User Object)
- Step 2: check it isn't null (good practise)

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);

        convertView = mInflater.inflate(R.layout.listview_user_item, null);
        holder.profilePicture = (ImageView) convertView.findViewById(R.id.profilePicture);
        holder.name = (TextView) convertView.findViewById(R.id.name);
        holder.email = (TextView) convertView.findViewById(R.id.email);

        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
    ShopItem s = shopItems.get(position);
    if (s != null) {
        String coins = NumberFormat.getNumberInstance(Locale.US).format(s.getCoins());
        coins = coins.replace(",", " ");
        holder.coins.setText(s.getItemCost());
        holder.price.setText(coins + " " + context.getResources().getString(R.string.shop_textView_coins));
        holder.shopImage.setImageResource(s.getItemImage());
    }
    return convertView;
}
```

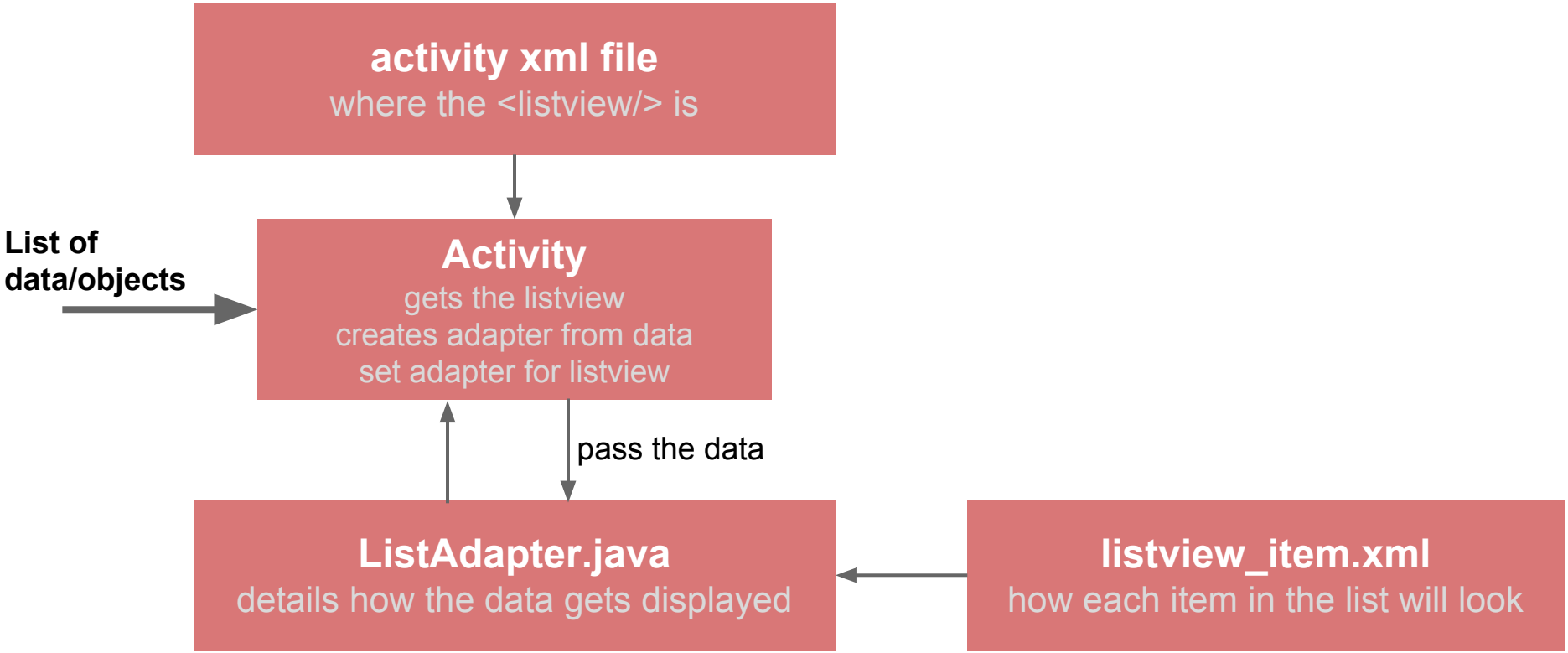
- Step 3: alter the views stored in the ViewHolder so that the reflect the various properties in the object

getView: Personalisation

- Need to get the User Object in the **users** list at the position in the **position** variable.
- Then set the relevant properties of the controls in the ViewHolder (Text, Image etc.)

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView == null) {
        holder = new ViewHolder();
        LayoutInflater mInflater = (LayoutInflater)
                context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);

        convertView = mInflater.inflate(R.layout.listview_user_item, null);
        holder.profilePicture = (ImageView) convertView.findViewById(R.id.profilePicture);
        holder.name = (TextView) convertView.findViewById(R.id.name);
        holder.email = (TextView) convertView.findViewById(R.id.email);

        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }

    User u = users.get(position);
    if (u != null) {
        holder.name.setText(u.getName());
        holder.email.setText(u.getEmail());
        if (u.getProfilePicture() != null)
            holder.profilePicture.setImageBitmap(u.getProfilePicture());
        else
            holder.profilePicture.setImageResource(R.drawable.profilepicture_placeholder);
    }

    return convertView;
}
```

Putting it all Together



Using the Adapter

- In **UserActivity.java**
- Variables:

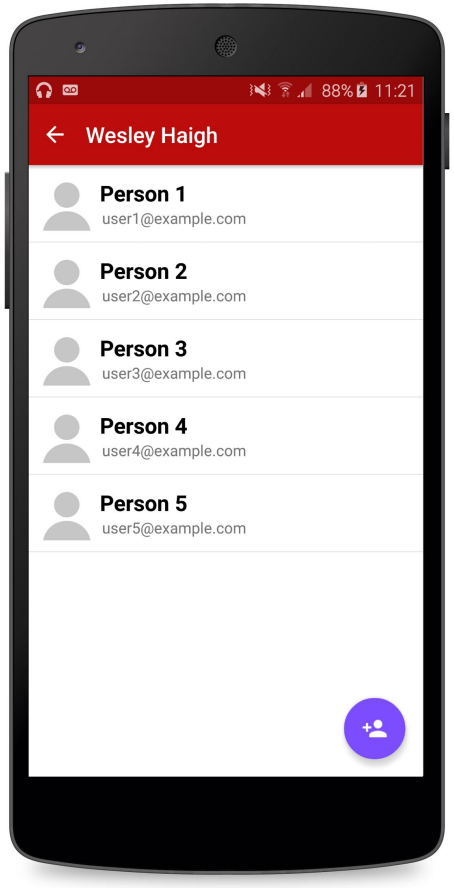
```
private ListView _userList;
private UserListAdapter _userListAdapter;
```

- Going to make a function called **initialiseList()** and **getXMLControls()**

```
public void getXMLControls() {
    _userList = (ListView) findViewById(R.id.userList);
}

private void initialiseList() {
    ArrayList<User> users = new ArrayList<>();
    for (int i = 1; i<=5; i++) {
        User user = new User();
        user.setName("Person " + i);
        user.setEmail("user" + i + "@example.com");
        users.add(user);
    }
    _userListAdapter = new UserListAdapter(this, users);
    _userList.setAdapter(_userListAdapter);
}
```

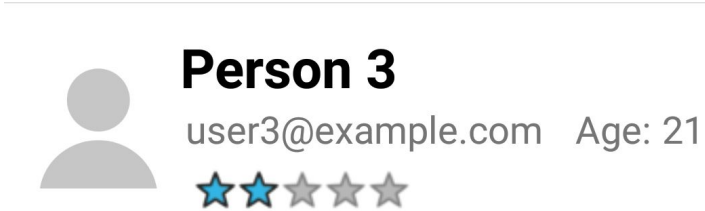
Using the Adapter

- Make sure you are calling **getXMLControls()** and **initialiseList()** in the **onCreate()** function!
- Run the program and put in the password
- You should now see a list of people on the UserActivity



End of Lecture Exercise

- Change the items in the listview so that they display the age, and rating of the user as well.
- Use a **<RatingBar/>** to create the ratings try looking up the properties of a rating bar
- Will need to modify:
  - listview_user_item.xml
  - UserListAdapter.java
  - UserActivity.java



warwick

C<>ding