Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Application Design

Agenda & Goals

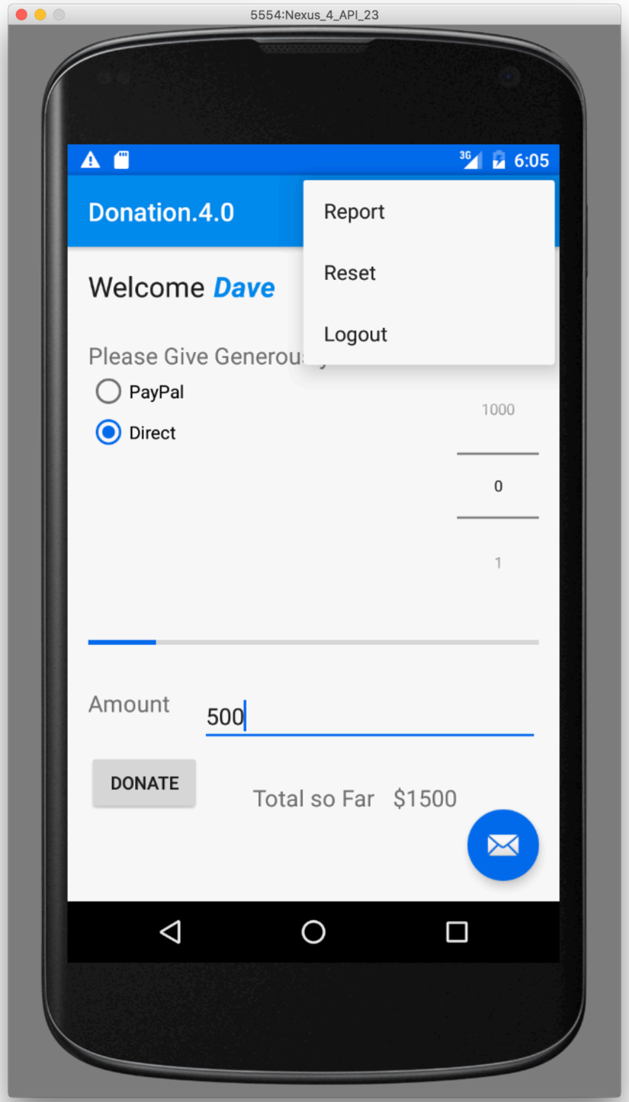
- ❑Application Design
- ❑Donation Data Model
- ❑More Menu Navigation
- ❑Creating and using Custom **Adapters**

Introduction – App Design

- The structure of an Android application is fairly rigidly defined. In order for things to work properly, you need to put certain files in the right places.
- As the complexity of an app increases, generally, so too does the design and structure of the app.
- From the developers perspective, it is important to try and maintain the rigid, highly organised, app structure, following well established guidelines and principles.
- Here, we try and follow these principles in refactoring our Donation App to include a Base Class and a Model.

Case Study

- ❑ *Donation* – an Android App to keep track of donations made to '*Homers Presidential Campaign* '.
- ❑ App Features
  - Accept donation via number picker or typed amount
  - Keep a running total of donations
  - Display report on donation amounts and types
  - Display running total on progress bar



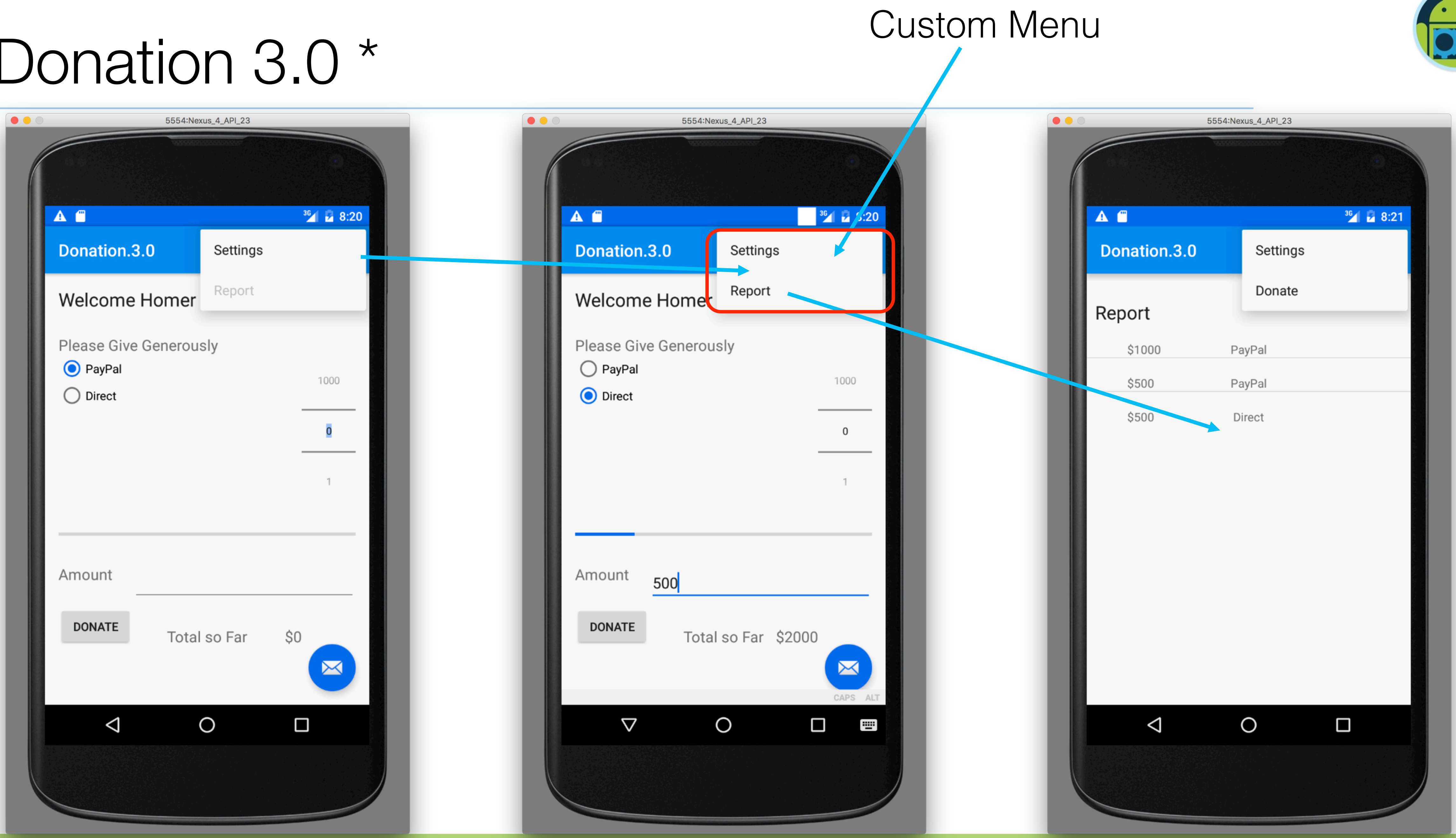
Donation.3.0

Introducing the Model

&

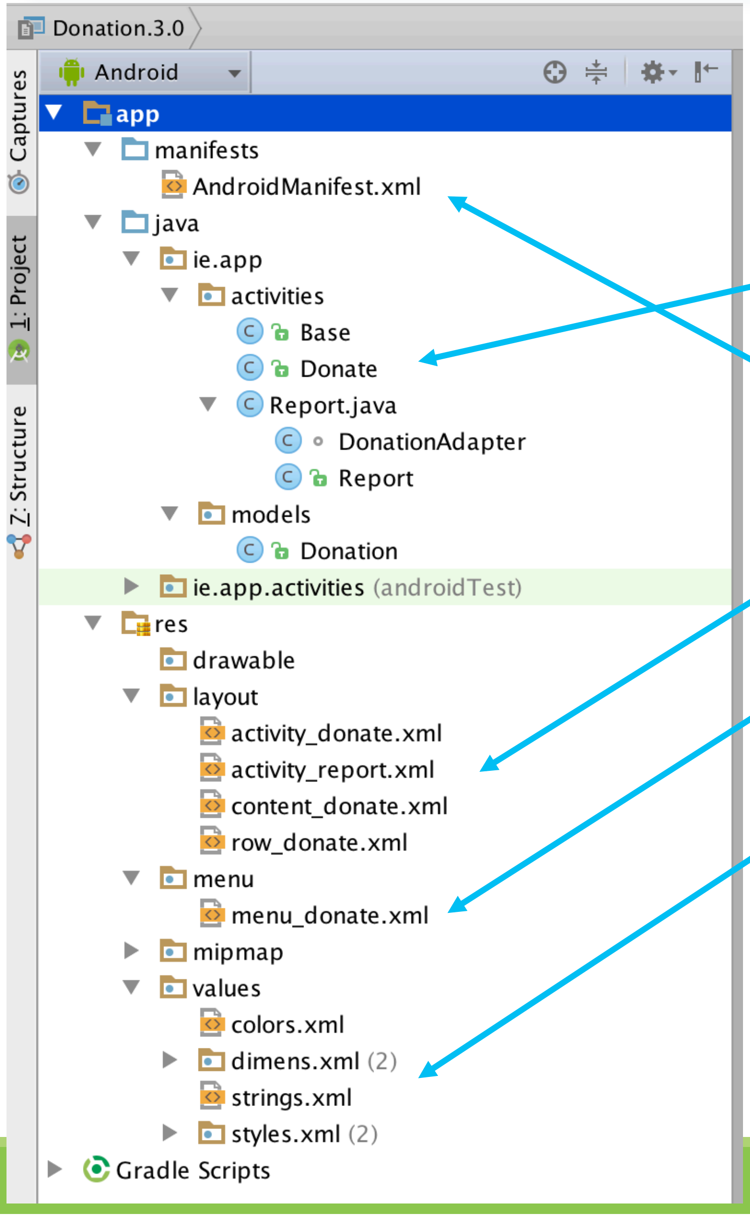
Base Class

Donation 3.0 *



$1000	PayPal
$500	PayPal
$500	Direct

Donation 3.0 – Project Structure *

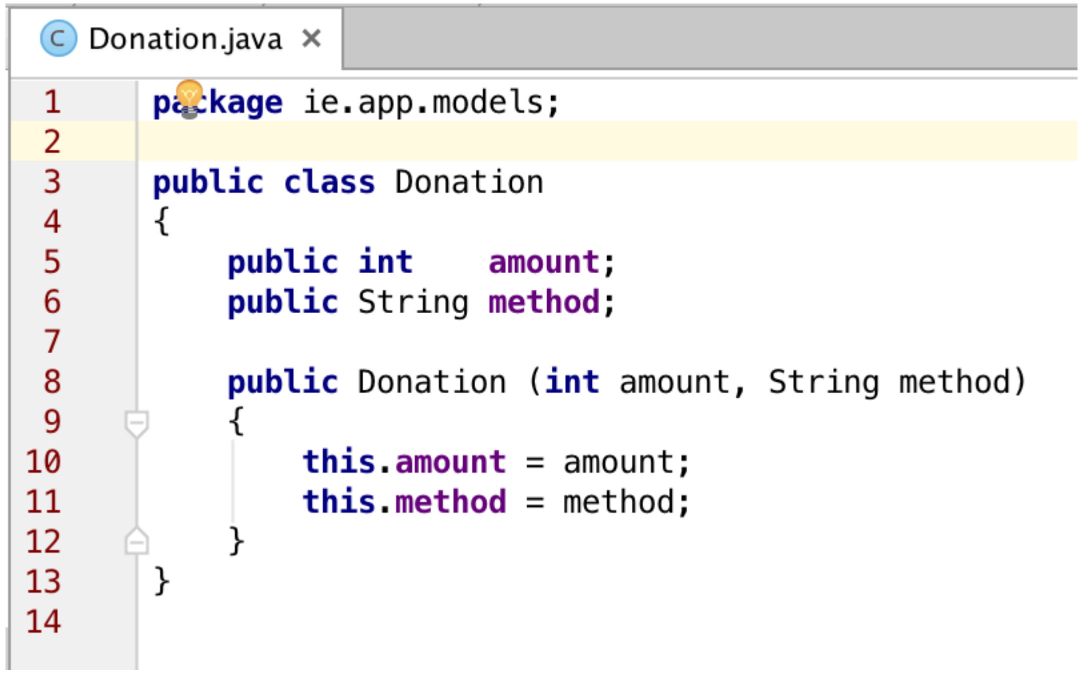


Donation.3.0

```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ ie.app
      ▼ activities
          Base
          Donate
        ▼ Report.java
            DonationAdapter
            Report
      ▼ models
          Donation
    ▶ ie.app.activities (androidTest)
  ▼ res
      drawable
    ▼ layout
        activity_donate.xml
        activity_report.xml
        content_donate.xml
        row_donate.xml
    ▼ menu
        menu_donate.xml
    ▶ mipmap
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
      ▶ styles.xml (2)
▶ Gradle Scripts
```

- 3 java source files
- 4 xml layouts
- 1 xml menu
- 6 xml files for resources
- 1 xml 'configuration' file

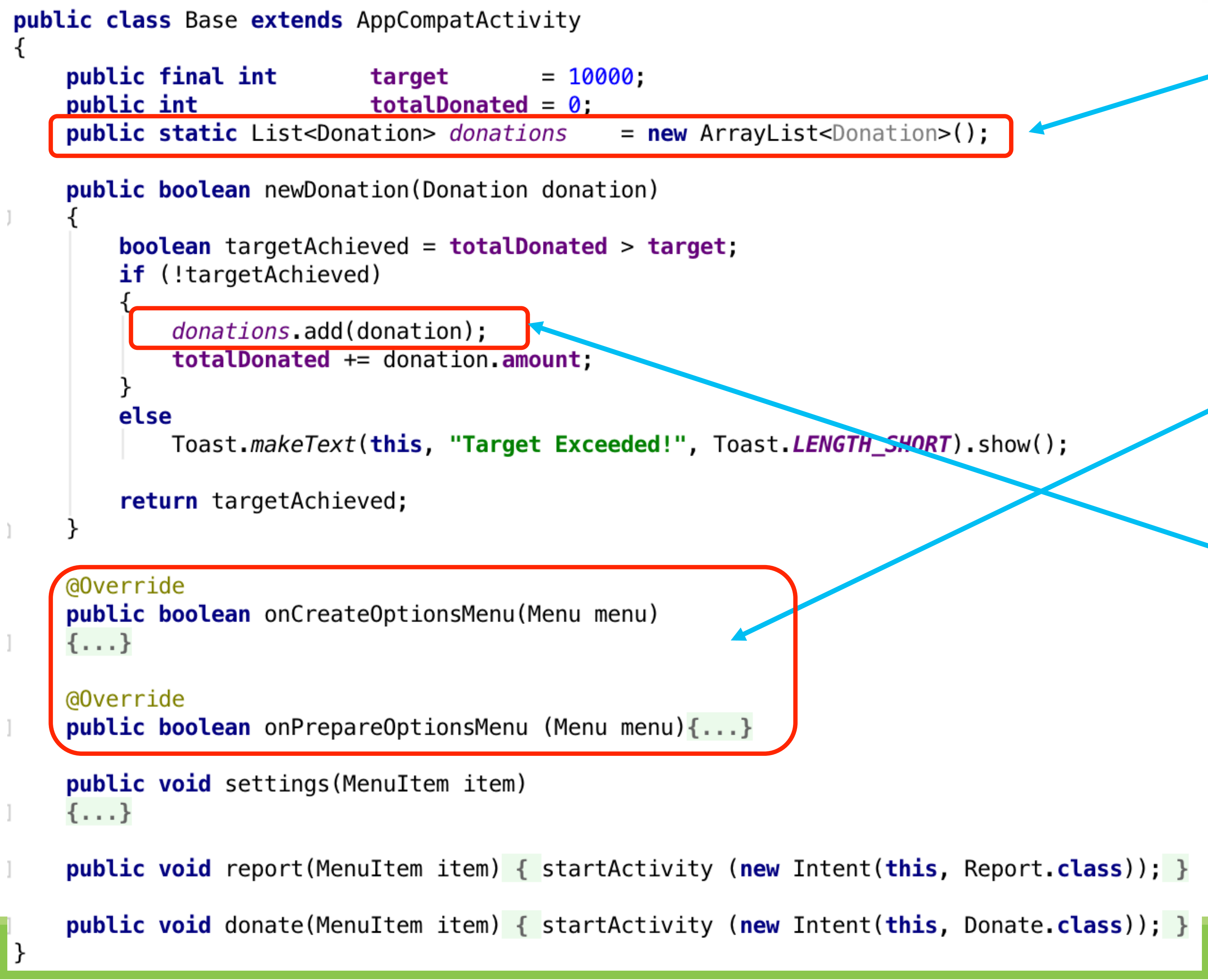
Donation 3.0 - Model



```
package ie.app.models;

public class Donation
{
    public int    amount;
    public String method;

    public Donation (int amount, String method)
    {
        this.amount = amount;
        this.method = method;
    }
}
```

We'll refactor this class in Donation 4.0 to include an 'id'

Donation 3.0 – Base Class *



```
public class Base extends AppCompatActivity
{
    public final int        target       = 10000;
    public int              totalDonated = 0;
    public static List<Donation> donations     = new ArrayList<Donation>();

    public boolean newDonation(Donation donation)
    {
        boolean targetAchieved = totalDonated > target;
        if (!targetAchieved)
        {
            donations.add(donation);
            totalDonated += donation.amount;
        }
        else
            Toast.makeText(this, "Target Exceeded!", Toast.LENGTH_SHORT).show();

        return targetAchieved;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {...}

    @Override
    public boolean onPrepareOptionsMenu (Menu menu){...}

    public void settings(MenuItem item)
    {...}

    public void report(MenuItem item) { startActivity (new Intent(this, Report.class)); }

    public void donate(MenuItem item) { startActivity (new Intent(this, Donate.class)); }
}
```

Our List of Donations

We'll take a closer look at these methods in "Menus Part 2"

Adding a 'donation'

Why a ‘Base’ Class?? *

- *Green* Programming – Reduce, Reuse, Recycle
  - Reduce the amount of code we need to implement the functionality required (Code Redundancy)
  - Reuse common code throughout the app/project where possible/appropriate
  - Recycle existing code for use in other apps/projects

- All good for improving Design

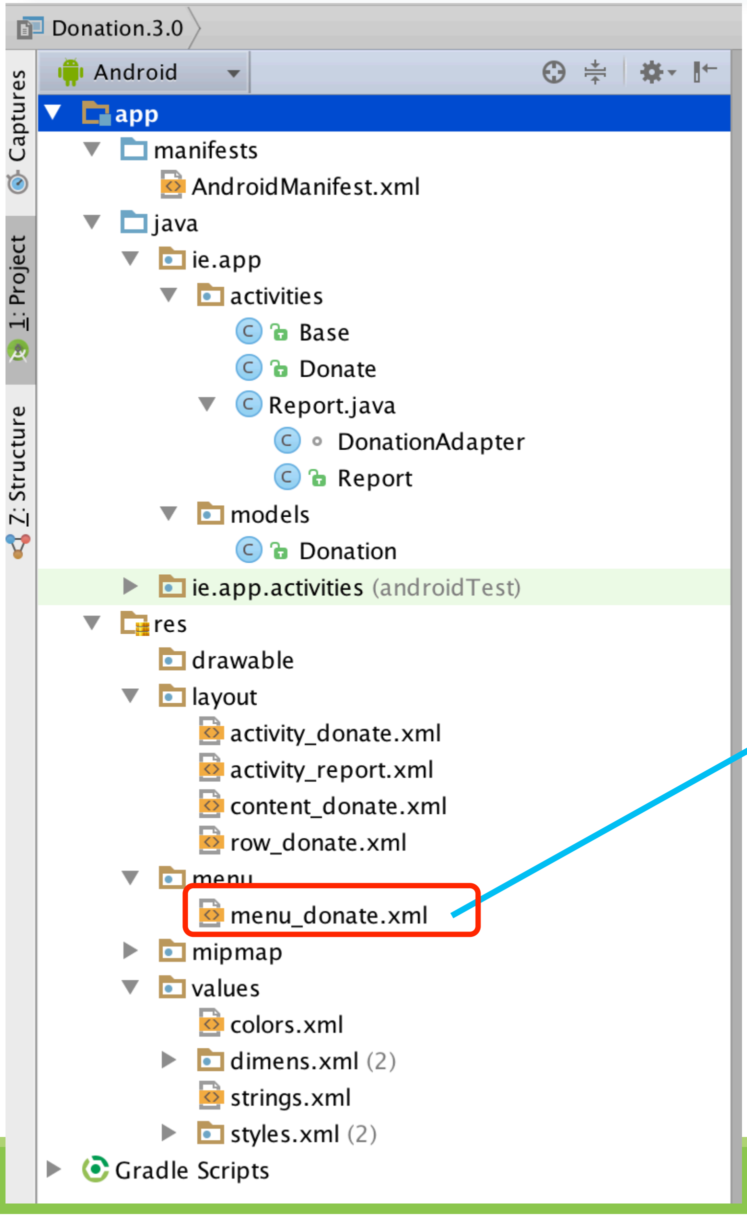
Donation.3.0

Using Menus
Part 2

Enabling/Disabling Menu Items on the fly

- There may be times where you don't want all your menu options available to the user under certain situations
  - e.g – if you've no donations, why let them see the report?
- You can modify the options menu at runtime by overriding the `onPrepareOptionsMenu()` method
  - called each and every time the user presses the *MENU* button.

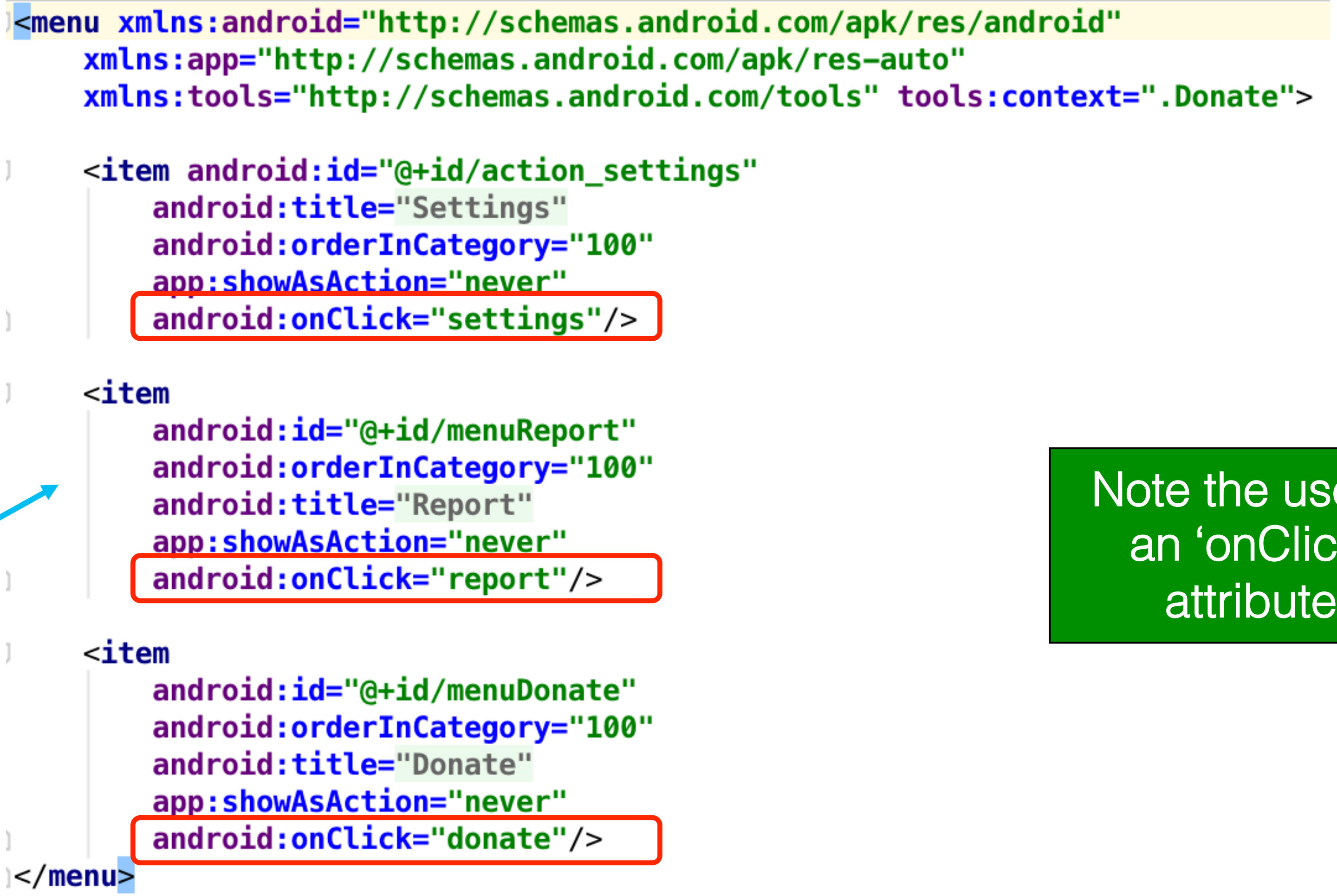
Menus in *Donation 3.0* *


Donation.3.0

Android

```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ ie.app
      ▼ activities
          Base
          Donate
        ▼ Report.java
            DonationAdapter
            Report
      ▼ models
          Donation
    ▶ ie.app.activities (androidTest)
  ▼ res
      drawable
    ▼ layout
        activity_donate.xml
        activity_report.xml
        content_donate.xml
        row_donate.xml
    ▼ menu
        menu_donate.xml
    ▶ mipmap
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
      ▶ styles.xml (2)
▶ Gradle Scripts
```

Menu Specification



```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".Donate">

    <item android:id="@+id/action_settings"
        android:title="Settings"
        android:orderInCategory="100"
        app:showAsAction="never"
        android:onClick="settings"/>

    <item
        android:id="@+id/menuReport"
        android:orderInCategory="100"
        android:title="Report"
        app:showAsAction="never"
        android:onClick="report"/>

    <item
        android:id="@+id/menuDonate"
        android:orderInCategory="100"
        android:title="Donate"
        app:showAsAction="never"
        android:onClick="donate"/>
</menu>
```

Note the use of an ‘onClick’ attribute

Donation 3.0 Menu Event Handler *

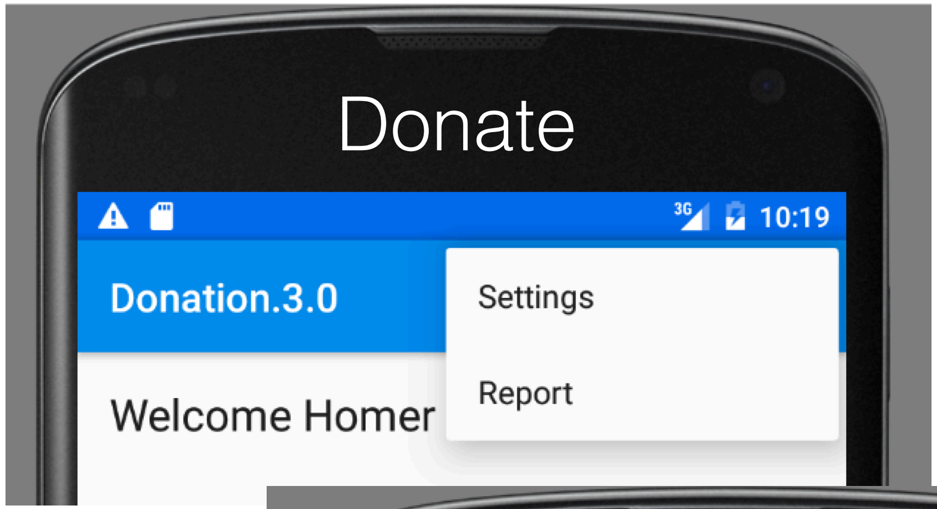
Menu Specification

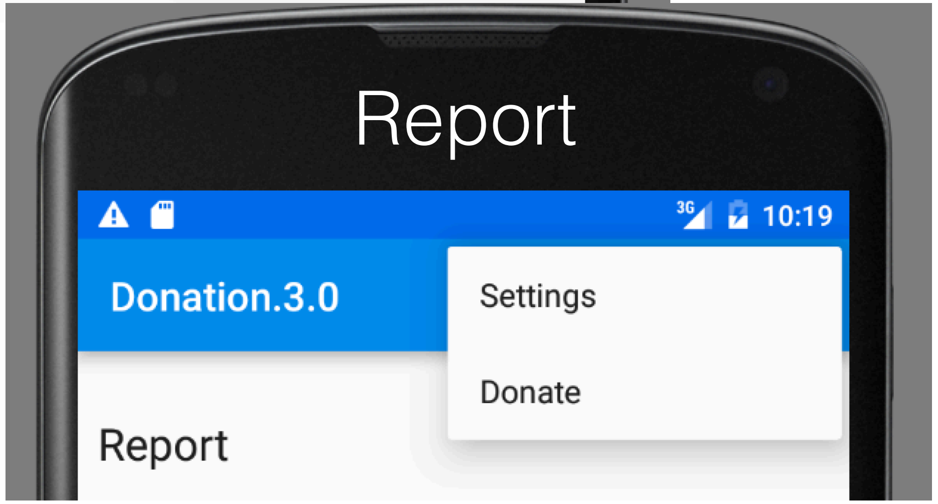
```java
public class Base extends AppCompatActivity
{
    public final int        target       = 10000;
    public int              totalDonated = 0;
    public static List<Donation> donations     = new ArrayList<Donation>();

    public boolean newDonation(Donation donation)
    {...}

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {...}

    @Override
    public boolean onPrepareOptionsMenu (Menu menu){...}

    public void settings(MenuItem item)
    {
        Toast.makeText(this, "Settings Selected", Toast.LENGTH_SHORT).show
    }

    public void report(MenuItem item)
    {
        startActivity (new Intent(this, Report.class));
    }

    public void donate(MenuItem item)
    {
        startActivity (new Intent(this, Donate.class));
    }
```

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".Donate">

    <item android:id="@+id/action_settings"
        android:title="Settings"
        android:orderInCategory="100"
        app:showAsAction="never"
        android:onClick="settings"/>

    <item
        android:id="@+id/menuReport"
        android:orderInCategory="100"
        android:title="Report"
        app:showAsAction="never"
        android:onClick="report"/>

    <item
        android:id="@+id/menuDonate"
        android:orderInCategory="100"
        android:title="Donate"
        app:showAsAction="never"
        android:onClick="donate"/>
</menu>
```

Donation 3.0 – `onPrepareOptionsMenu()`

```
@Override
public boolean onPrepareOptionsMenu (Menu menu){
    super.onPrepareOptionsMenu(menu);
    MenuItem report = menu.findItem(R.id.menuReport);
    MenuItem donate = menu.findItem(R.id.menuDonate);

    if(donations.isEmpty())
        report.setEnabled(false);
    else
        report.setEnabled(true);

    if(this instanceof Donate){
        donate.setVisible(false);
        if(!donations.isEmpty())
            report.setVisible(true);
    }
    else {
        report.setVisible(false);
        donate.setVisible(true);
    }

    return true;
}
```





Donation.3.0

Using ArrayAdapters

&

ListViews

Introducing Adapters

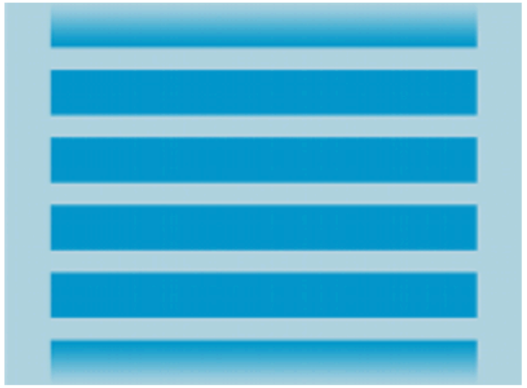
- **`Adapters`** are bridging classes that bind data to **`Views`** (eg ListViews) used in the UI.
  - Responsible for creating the child Views used to represent each item within the parent View, and providing access to the underlying data
- Views that support adapter binding must extend the **`AdapterView`** abstract class.
  - You can create your own `AdapterView`-derived controls and create new custom `Adapter` classes to bind to them.
- Android supplies a set of `Adapters` that pump data into native UI controls and layouts (next slide)

Building Layouts with an Adapter

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses `AdapterView` to populate the layout with views at runtime. A subclass of the `AdapterView` class uses an `Adapter` to bind data to its layout. The `Adapter` behaves as a middleman between the data source and the `AdapterView` layout—the `Adapter` retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the `AdapterView` layout.

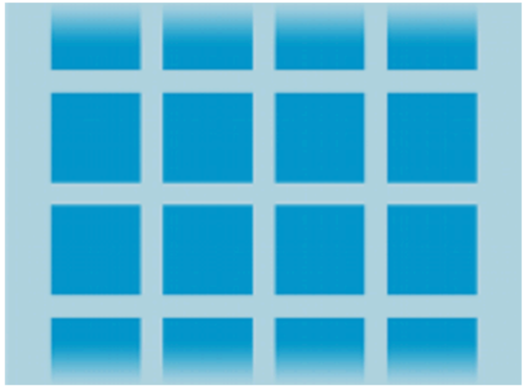
Common layouts backed by an adapter include:

List View



Displays a scrolling single column list.

Grid View



Displays a scrolling grid of columns and rows.

Building Layouts with an Adapter

- Because **`Adapters`** are responsible for supplying the data AND for creating the Views that represent each item, they can radically modify the appearance and functionality of the controls they're bound to.
- Most Commonly used Adapters
  - **`ArrayAdapter`**
    - uses generics to bind an **`AdapterView`** to an array of objects of the specified class.
    - By default, uses the **`toString()`** of each object to create & populate `TextViews`.
    - Other constructors available for more complex layouts (as we will see later on)
    - Can extend the class to use alternatives to simple `TextViews` (as we will see later on)
- See also **`SimpleCursorAdapter`** – attaches Views specified within a layout to the columns of Cursors returned from Content Provider queries.

Filling an Adapter View with Data

- You can populate an AdapterView such as ListView or GridView by binding the AdapterView instance to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                                    android.R.layout.simple_list_item_1,
                                    numbers);
```

- The arguments for this constructor are:
  - Your app Context
  - The layout that contains a TextView for each string in the array
  - The string array (*numbers*)
- Then simply call setAdapter() on your ListView:

Donation.2.0

```
listView = (ListView) findViewById(R.id.reportList);
listView.setAdapter(adapter);
```

Handling Click Events

- ❑ You can respond to click events on each item in an AdapterView by implementing the AdapterView.OnItemClickListener interface

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

- ❑ We won't be covering this in our Case Study, but would be desirable to see in your project

Donation.3.0

Custom Adapters

Customizing the ArrayAdapter *

- By default, the **`ArrayAdapter`** uses the **`toString()`** of the object array it's binding, to populate the **`TextView`** available within the specified layout.
- Generally, you customize the layout to display more complex views by..
  - Extending the **`ArrayAdapter`** class with a type-specific variation, eg

```
class DonationAdapter extends ArrayAdapter<Donation>
```

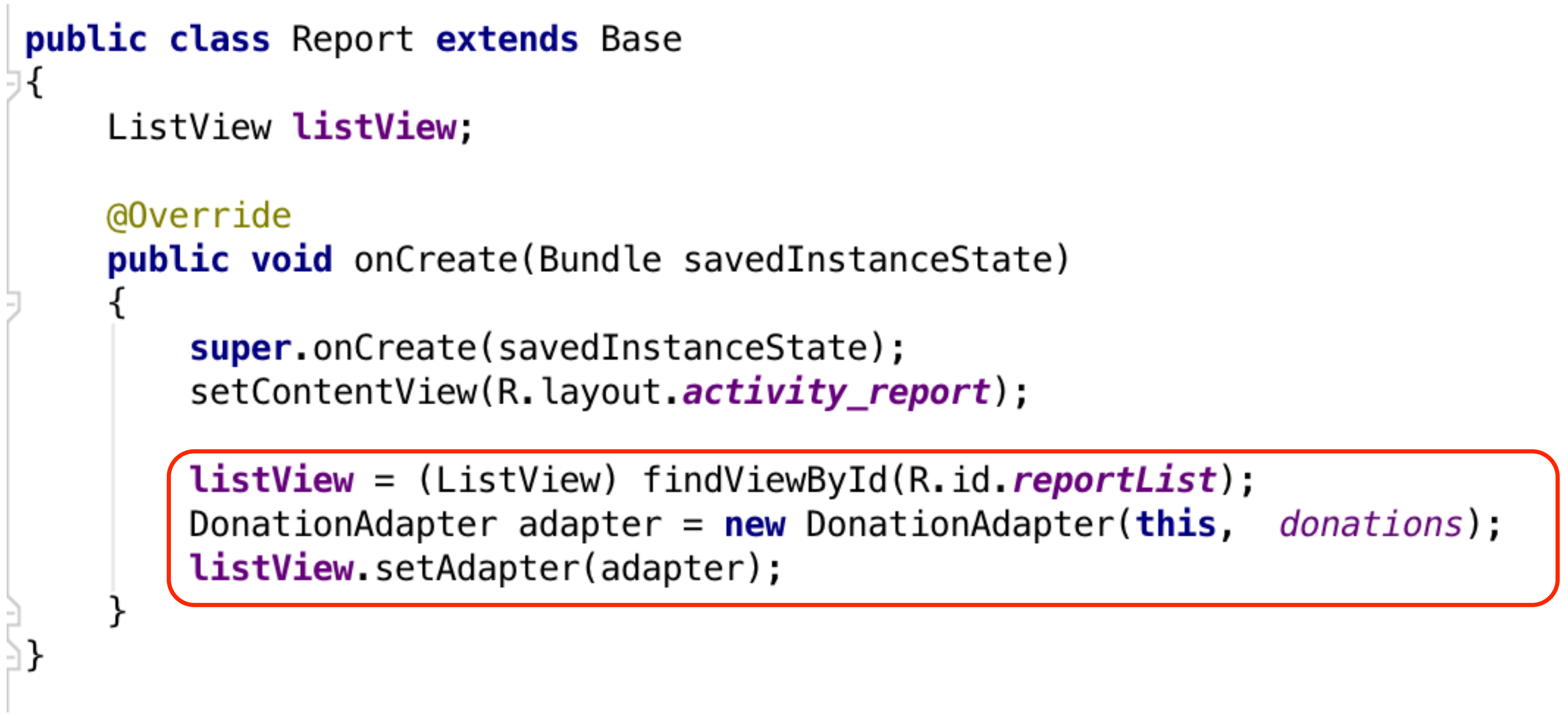
  - Override the **`getView()`** method to assign object properties to layout `View` objects. (see our case study example next)

The `getView()` Method

- Used to construct, inflate, and populate the View that will be displayed within the parent **`AdapterView`** class (eg a ListView) which is being bound to the underlying array using this adapter.
- Receives parameters that describes
  - The position of the item to be displayed
  - The **`View`** being updated (or `null`)
  - The **`ViewGroup`** into which this new **`View`** will be placed
- Returns the new populated **`View`** instance as a result

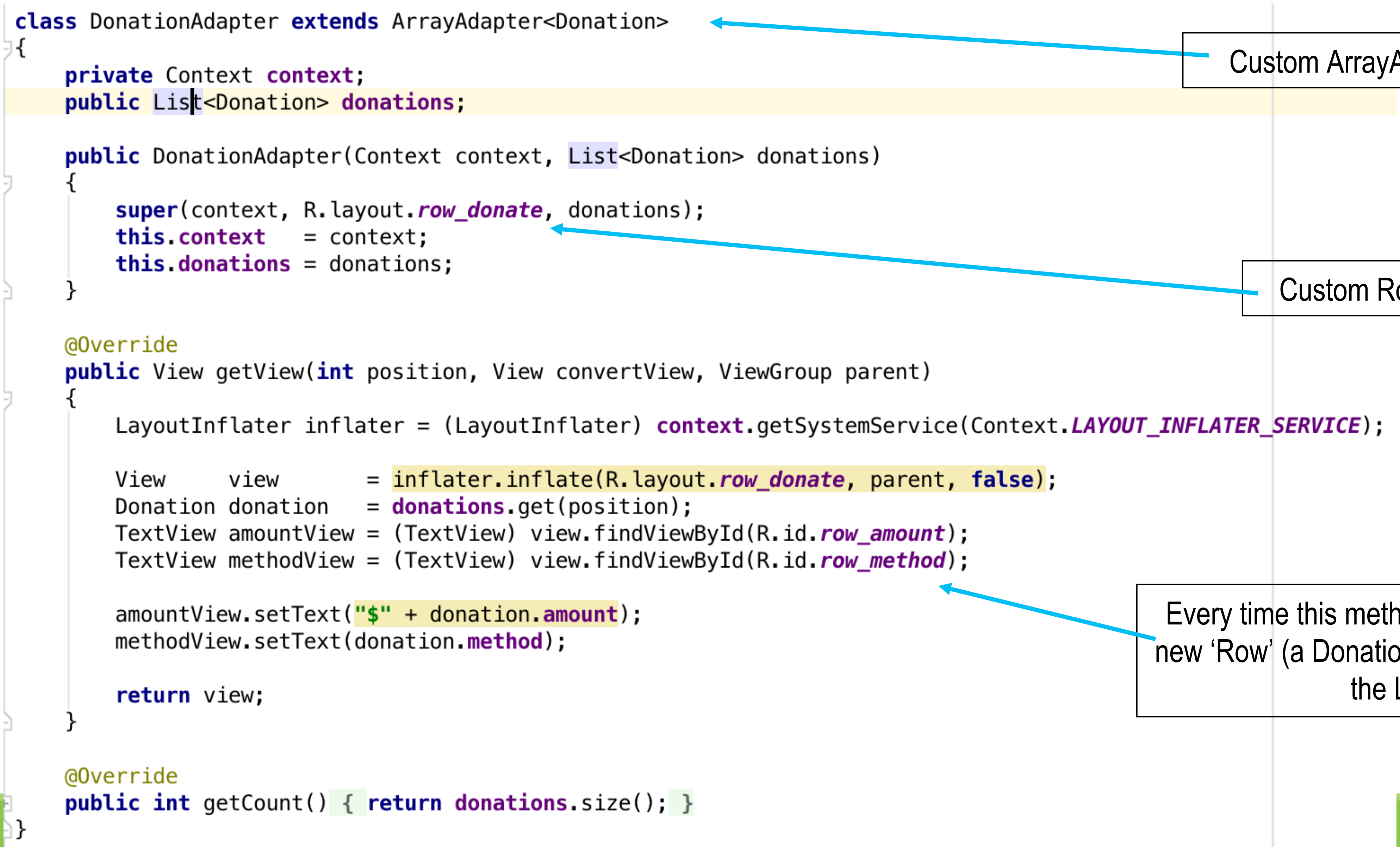
- A call to **`getItem()`** will return the value (object) stored at the specified index in the underlying array.

Donation 3.0 – Report Activity *



```
public class Report extends Base
{
    ListView listView;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_report);

        listView = (ListView) findViewById(R.id.reportList);
        DonationAdapter adapter = new DonationAdapter(this,  donations);
        listView.setAdapter(adapter);
    }
}
```

Donation 3.0 - DonationAdapter class



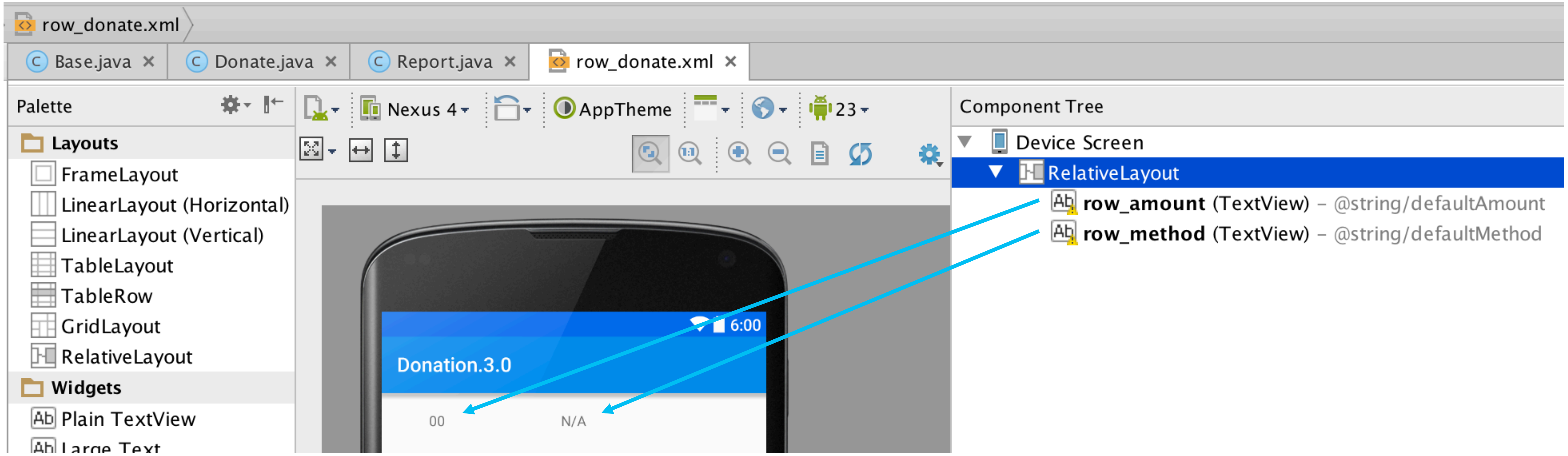
```java
class DonationAdapter extends ArrayAdapter<Donation>
{
    private Context context;
    public List<Donation> donations;

    public DonationAdapter(Context context, List<Donation> donations)
    {
        super(context, R.layout.row_donate, donations);
        this.context   = context;
        this.donations = donations;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View     view        = inflater.inflate(R.layout.row_donate, parent, false);
        Donation donation    = donations.get(position);
        TextView amountView = (TextView) view.findViewById(R.id.row_amount);
        TextView methodView = (TextView) view.findViewById(R.id.row_method);

        amountView.setText("$" + donation.amount);
        methodView.setText(donation.method);

        return view;
    }

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

Custom ArrayAdapter of type ‘Donation’

Custom Row Layout

Every time this method is called we create a new ‘Row’ (a Donation from our List) to add to the ListView

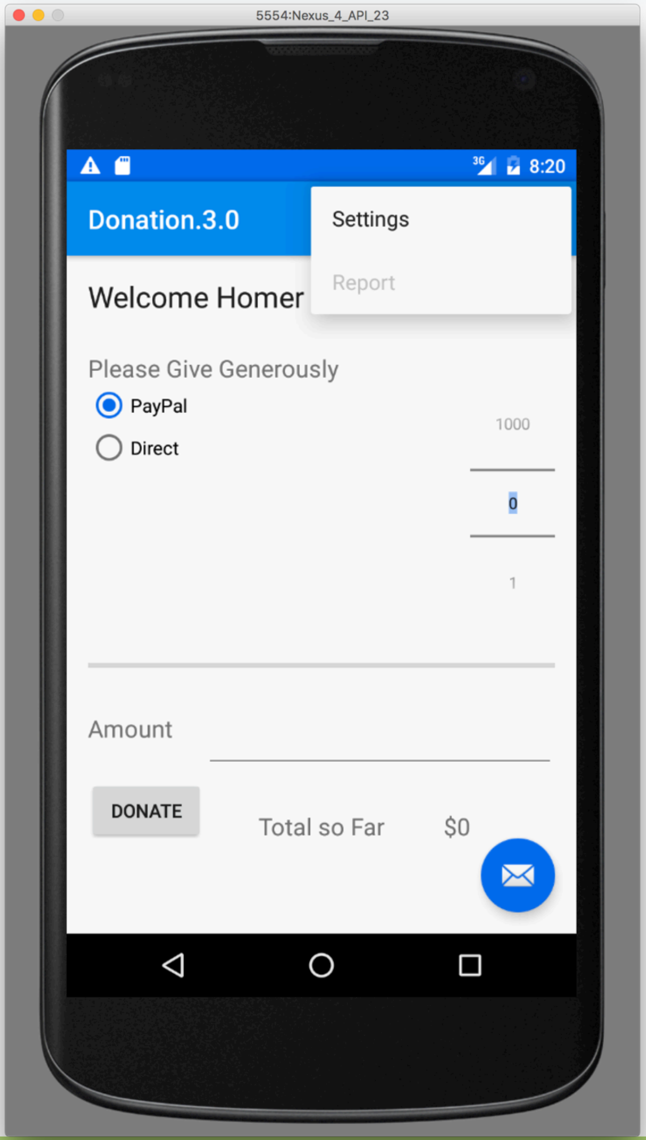
Donation 3.0 - row_donate.xml

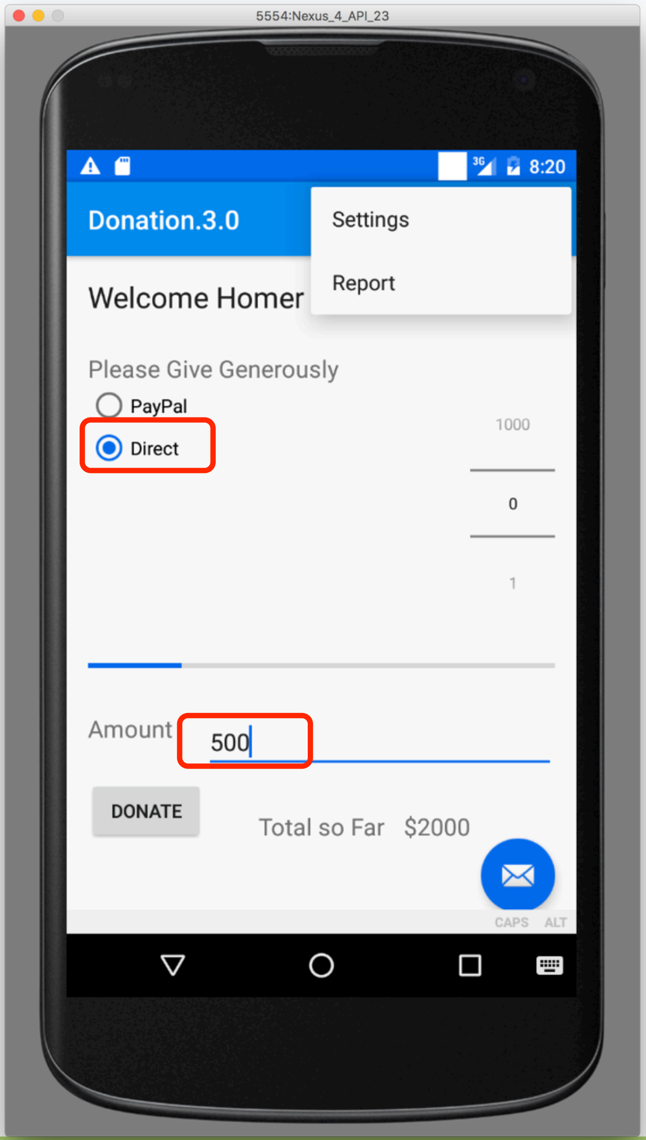


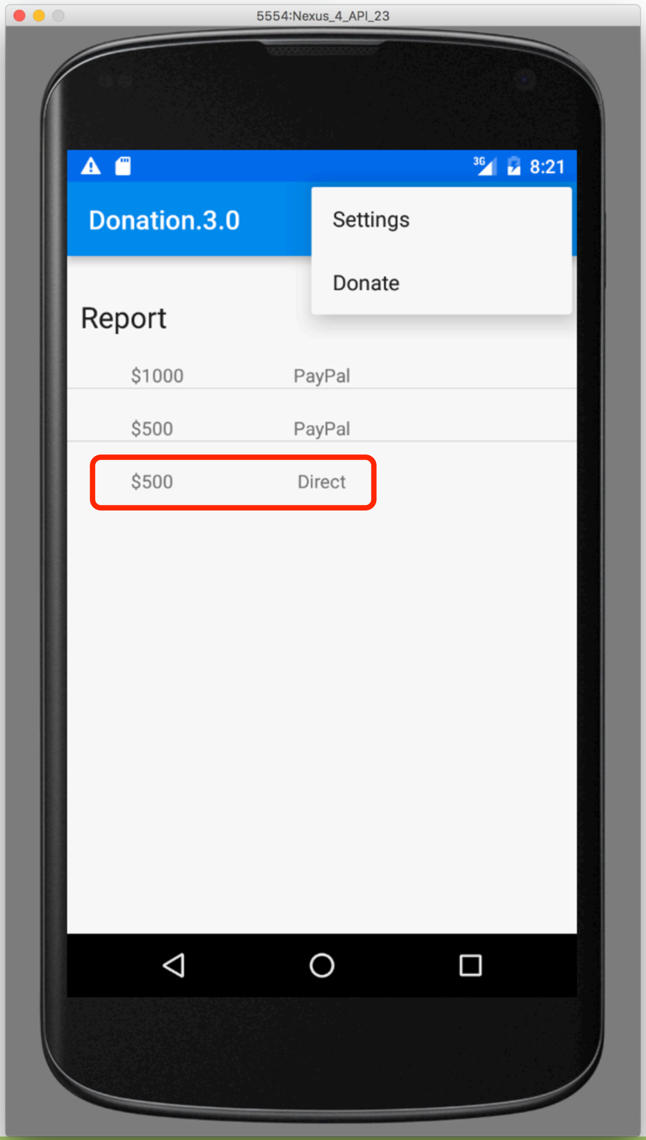
Component Tree
- Device Screen
  - RelativeLayout
    - **row_amount** (TextView) – @string/defaultAmount
    - **row_method** (TextView) – @string/defaultMethod

Each time `getView()` is called, it creates a new '`Row`' and binds the individual Views (widgets) above, to each element of the object array in the `ArrayAdapter`.

Resulting ListView (inside our Report) *







$1000	PayPal
$500	PayPal
$500	Direct

Summary

- ❑We looked at Application Structure and Design
- ❑We revisited the Structure of our App and introduced a ‘Donation Model’ and Base class
- ❑We looked at more Menu Navigation
- ❑We Created and used Custom Adapters

Questions?