Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

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

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android Multithreading & Networking

Agenda & Goals

- ❑ Understand what multithreading is and the need for it in Android development
- ❑ Be aware of the different approaches available in Android multithreading
- ❑ Be aware how to implement some of these approaches (including *AsyncTask*s) in an Android app.
- ❑ Understand JSON & Googles Gson
- ❑ Investigate the use of Volley in Android Networking
- ❑ Revisit our Donation Case Study

Background Processes in General

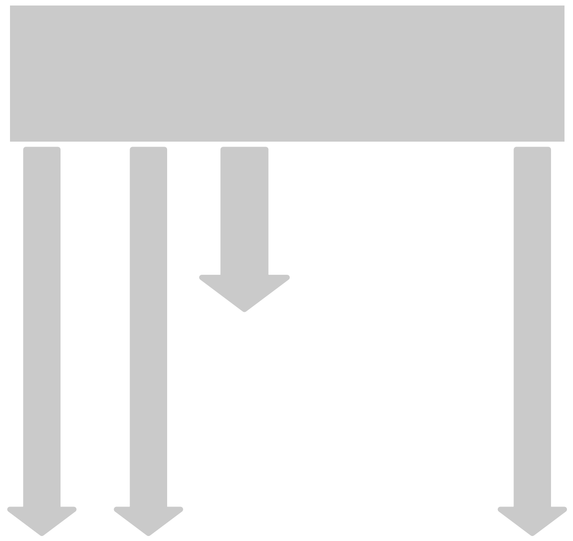
- One of the key features of Android (and iPhone) is the ability to run things in the background
  - Threads
    - Run something in the background while user interacts with UI
  - Services
    - Regularly or continuously perform actions that don’t require a UI

Threads

- ❑ Recall that Android ensures responsive apps by enforcing a 5 second limit on Activities
- ❑ Sometimes we need to do things that take longer than 5 seconds, or that can be done while the user does something else
- ❑ Activities, Services, and Broadcast Receivers run on the *main application thread*
- ❑ But we can start background/child threads to do other things for us

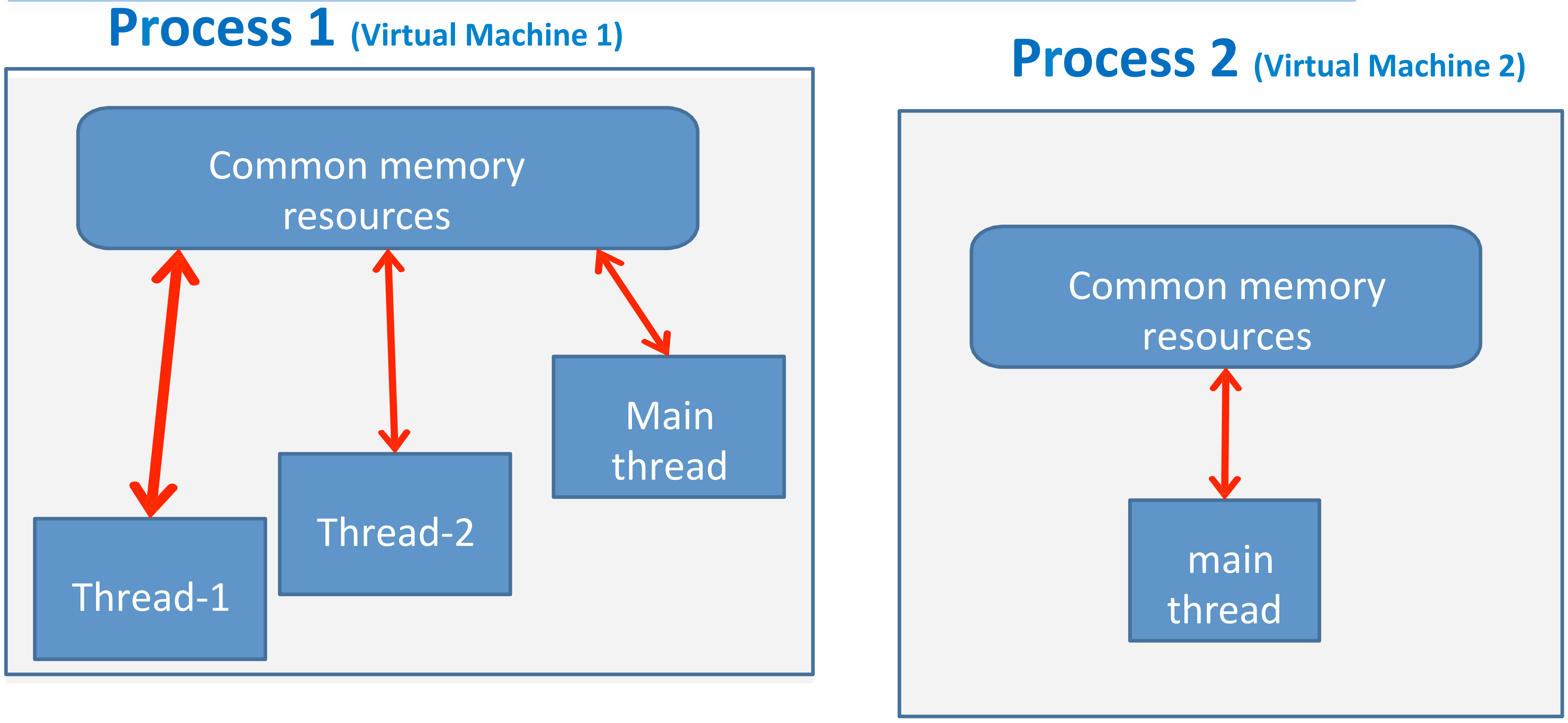
Threads

http://developer.android.com/reference/java/lang/Thread.html



- A Thread is a concurrent unit of execution.
- Each thread has its own call stack. The call stack is used on method calling, parameter passing, and storage for the called method’s local variables.
- Each virtual machine instance has at least one main thread.
- Threads in the same VM interact and synchronize by the use of shared objects and monitors associated with these objects.

Threads


Process 1 (Virtual Machine 1)
Common memory resources
Thread-1
Thread-2
Main thread
Process 2 (Virtual Machine 2)
Common memory resources
main thread

Android Thread Constraints

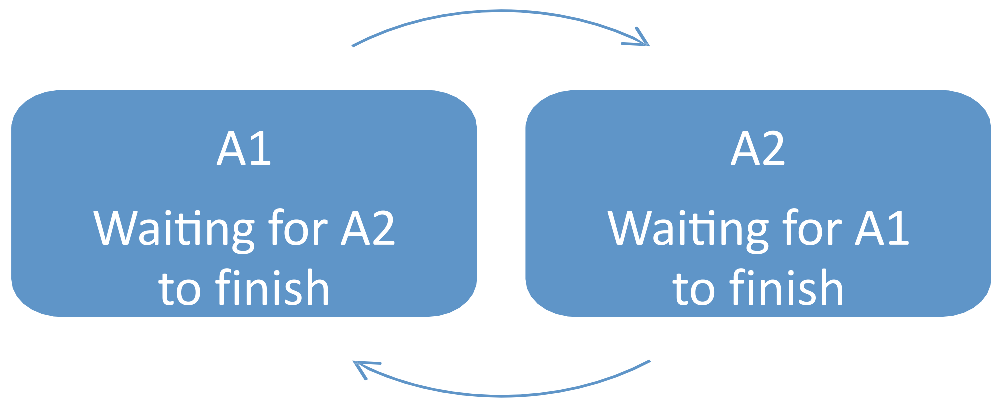
- Child threads ***cannot*** access UI elements (views); these elements must (and can only) be accessed through the main thread
- So what do you do?
  - You give results to the main thread and let it use the results

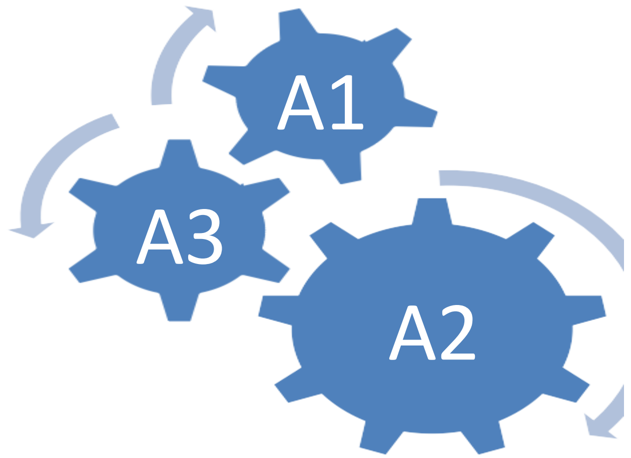
Advantages of Multithreading

- Threads share the process' resources but are able to execute independently.
- Applications responsibilities can be separated
  - main thread runs UI, and
  - slow tasks are sent to background threads.
- Threading provides an useful abstraction of *concurrent* execution.
- A multithreaded program operates *faster* on computer systems that have *multiple CPUs*.
  (Java 8 supports multi-core multi-threading)

Disadvantages

- Code tends to be more complex
- Need to detect, avoid, resolve deadlocks





Android‘s Approach to Slow Activities

Problem: An application may involve a time-consuming operation.
Goal: We want the **UI** to be responsive to the user in spite of heavy load.
Solution: Android offers two ways for dealing with this scenario:

1. Do expensive operations in a background ***service***, using *notifications* to inform users about next step
2. Do the slow work in a ***background thread***.

Using Threads: Interaction ***between*** Android threads is accomplished using
(a) a main thread ***Handler*** object and
(b) posting ***Runnable*** objects to the main view.

Thread Execution – Example

There are basically two main ways of having a **Thread** execute application code.

- Create a new class that *extends* **Thread** and override its **run()** method.

```
MyThread t = new MyThread();
t.start();
```

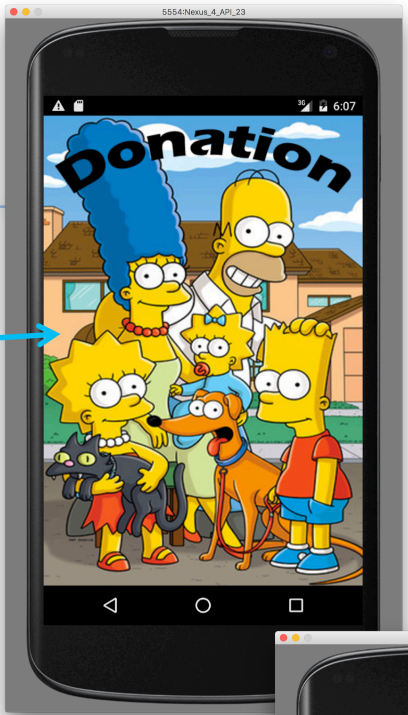
- Create a new **Thread** instance passing to it a **Runnable** object.

```
Runnable myRunnable1 = new
MyRunnableClass(); Thread t1 = new
Thread(myRunnable1); t1.start();
```

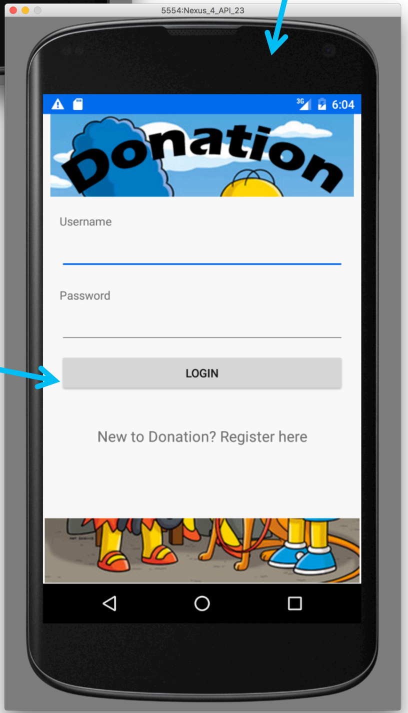
In both cases, the **start()** method must be called to actually execute the new Thread.

Multithreading – Our Splash Screen

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splash);
    Handler handler = new Handler();
    // run a thread after 2 seconds to start the home screen
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            // make sure we close the splash screen so the user
            // won't come back when it presses back key
            finish();

            if (!mIsBackButtonPressed) {
                // start the home screen if the back button wasn't pressed already
                Splash.this.startActivity(new Intent(Splash.this, Login.class));
            }
        }
    }, SPLASH_DURATION); // time in milliseconds to delay call to run()
}
```



2 secs later



Using the *AsyncTask* class

http://developer.android.com/reference/android/os/AsyncTask.html

- ❑ The AsyncTask class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
- ❑ An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.
- ❑ An asynchronous task is defined by

3 Generic Types	4 Main States	1 Auxiliary Method
Params, Progress, Result	**onPreExecute, doInBackground, onProgressUpdate onPostExecute**.	**publishProgress**

Using the *AsyncTask* class

```
AsyncTask  <Params,  Progress, Result>
```

AsyncTask's generic types	
Params:	the type of the input parameters sent to the task at execution.
Progress:	the type of the progress units published during the background computation.
Result:	the type of the result of the background computation.

- ❑Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type **Void**

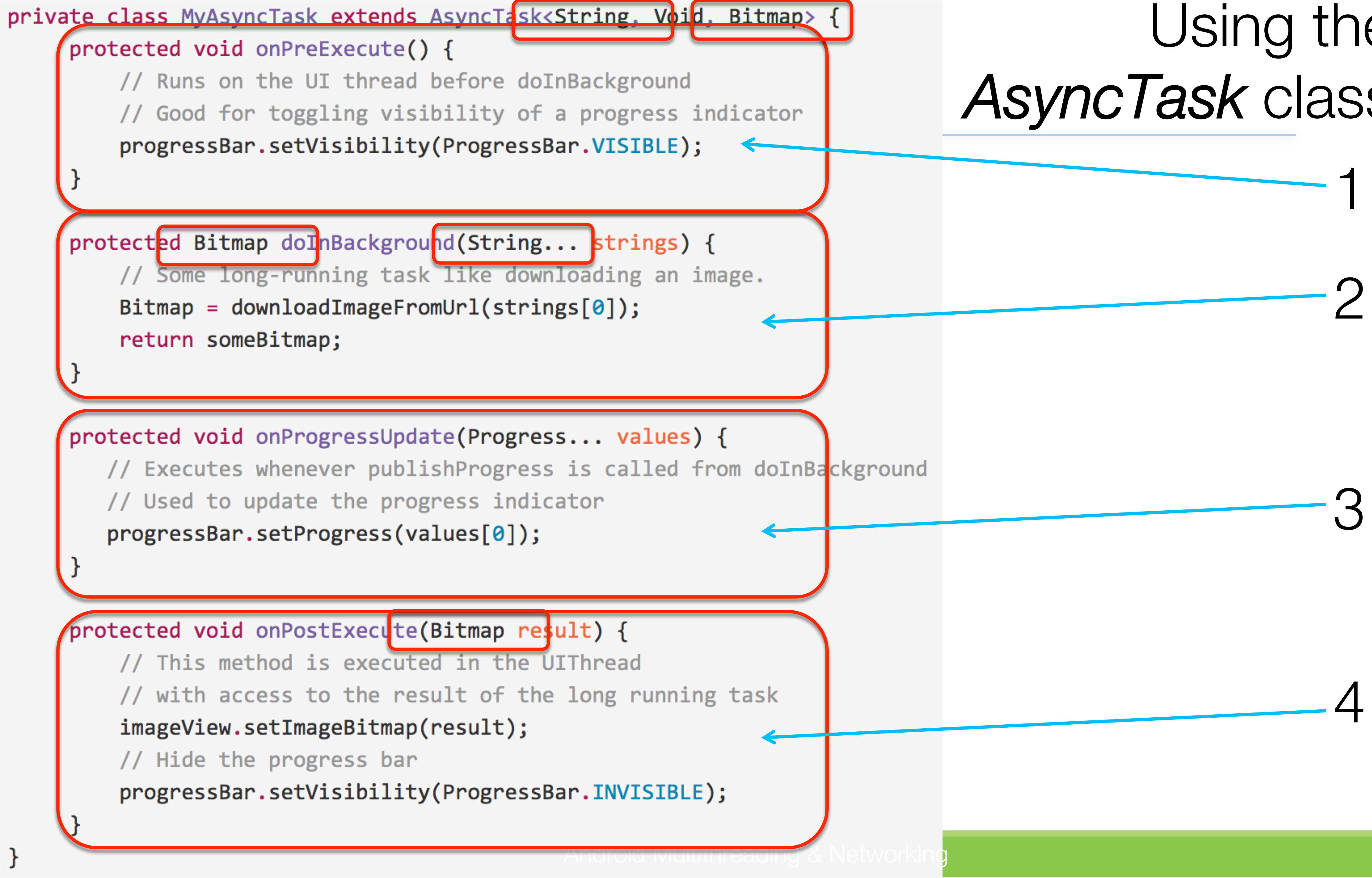
Note:

Syntax "`String ...`"   indicates (Varargs) array of String values, similar to "`String[]`"

Using the *AsyncTask* class

- onPreExecute
  - is invoked ***before*** the execution.
- onPostExecute
  - is invoked ***after*** the execution.
- doInBackground
  - ***the main operation***. Write your heavy operation here.
- onProgressUpdate
  - Indication to the user on the current progress. It is invoked every time publishProgress() is called.

Using the *AsyncTask* class

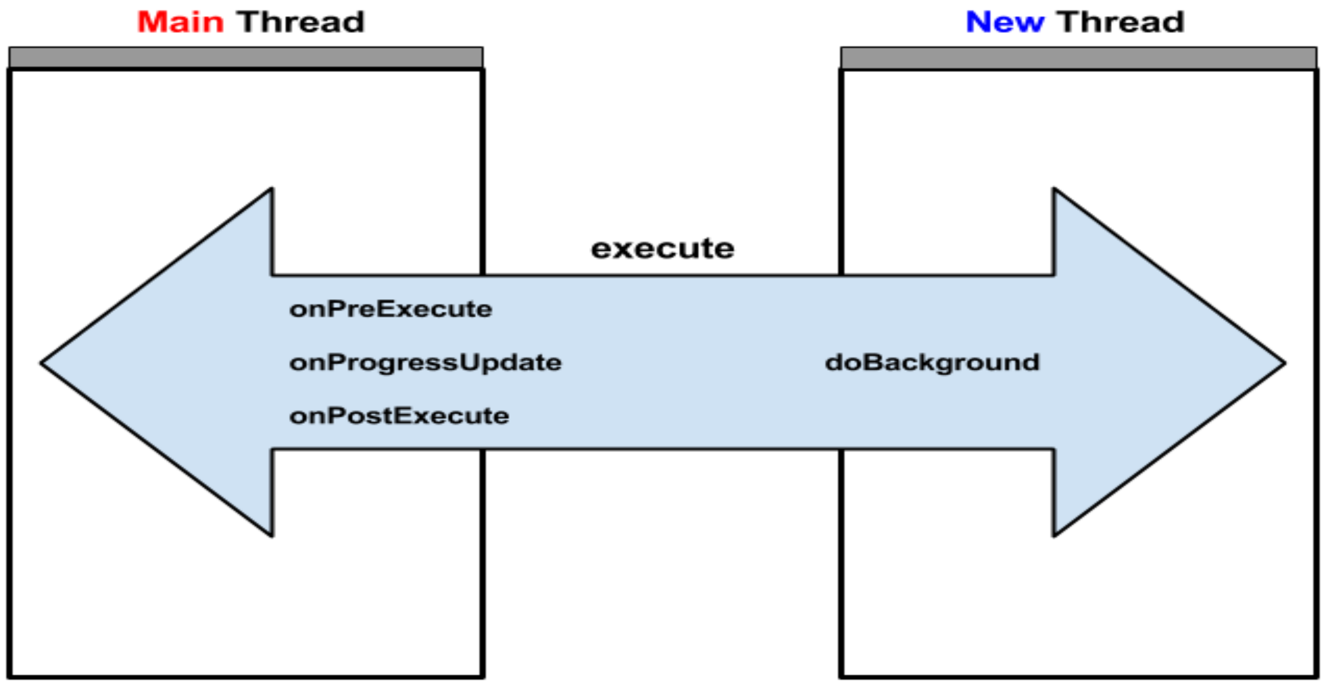


```java
private class MyAsyncTask extends AsyncTask<String, Void, Bitmap> {
    protected void onPreExecute() {
        // Runs on the UI thread before doInBackground
        // Good for toggling visibility of a progress indicator
        progressBar.setVisibility(ProgressBar.VISIBLE);
    }

    protected Bitmap doInBackground(String... strings) {
        // Some long-running task like downloading an image.
        Bitmap = downloadImageFromUrl(strings[0]);
        return someBitmap;
    }

    protected void onProgressUpdate(Progress... values) {
       // Executes whenever publishProgress is called from doInBackground
       // Used to update the progress indicator
       progressBar.setProgress(values[0]);
    }

    protected void onPostExecute(Bitmap result) {
        // This method is executed in the UIThread
        // with access to the result of the long running task
        imageView.setImageBitmap(result);
        // Hide the progress bar
        progressBar.setVisibility(ProgressBar.INVISIBLE);
    }
}
```

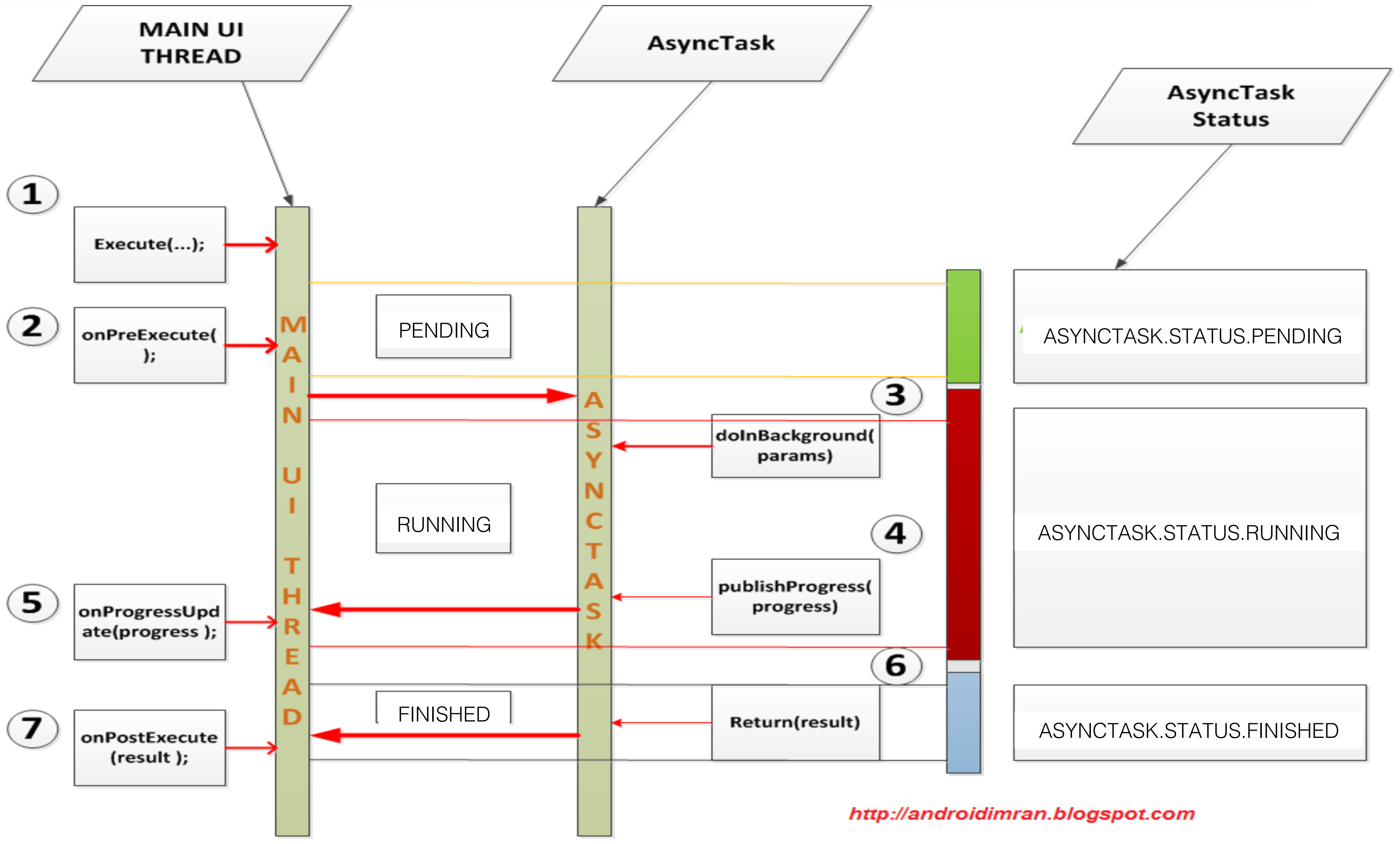
Using the *AsyncTask* class

AsyncTask's methods
onPreExecute(), invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
doInBackground(Params...), invoked on the background thread immediately after *onPreExecute*() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use *publishProgress(Progress...)* to publish one or more units of progress. These values are published on the UI thread, in the *onProgressUpdate(Progress...)* step.
onProgressUpdate(Progress...), invoked on the UI thread after a call to *publishProgress(Progress...)*. The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

The Asynchronous Calls



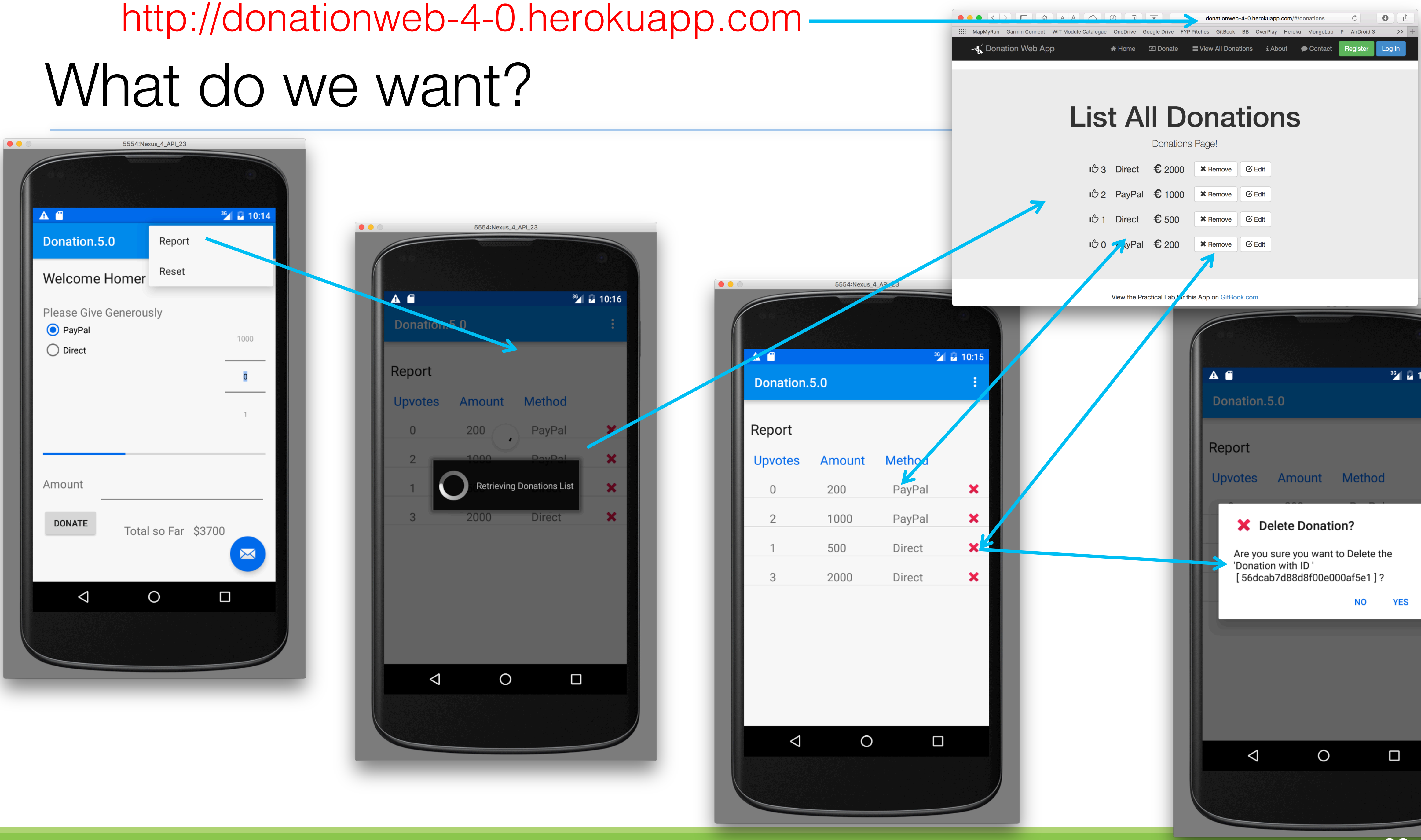
AsyncTask Lifecycle



Let’s look at our Donation Example

http://donationweb-4-0.herokuapp.com

What do we want?



List All Donations

Donations Page!

3	Direct	€ 2000	Remove	Edit
2	PayPal	€ 1000	Remove	Edit
1	Direct	€ 500	Remove	Edit
0	PayPal	€ 200	Remove	Edit

View the Practical Lab for this App on GitBook.com

Welcome Homer

Please Give Generously

PayPal

Direct

1000

0

1

Amount

Total so Far $3700

Report

Upvotes	Amount	Method
0	200	PayPal
2	1000	PayPal
1	500	Direct
3	2000	Direct

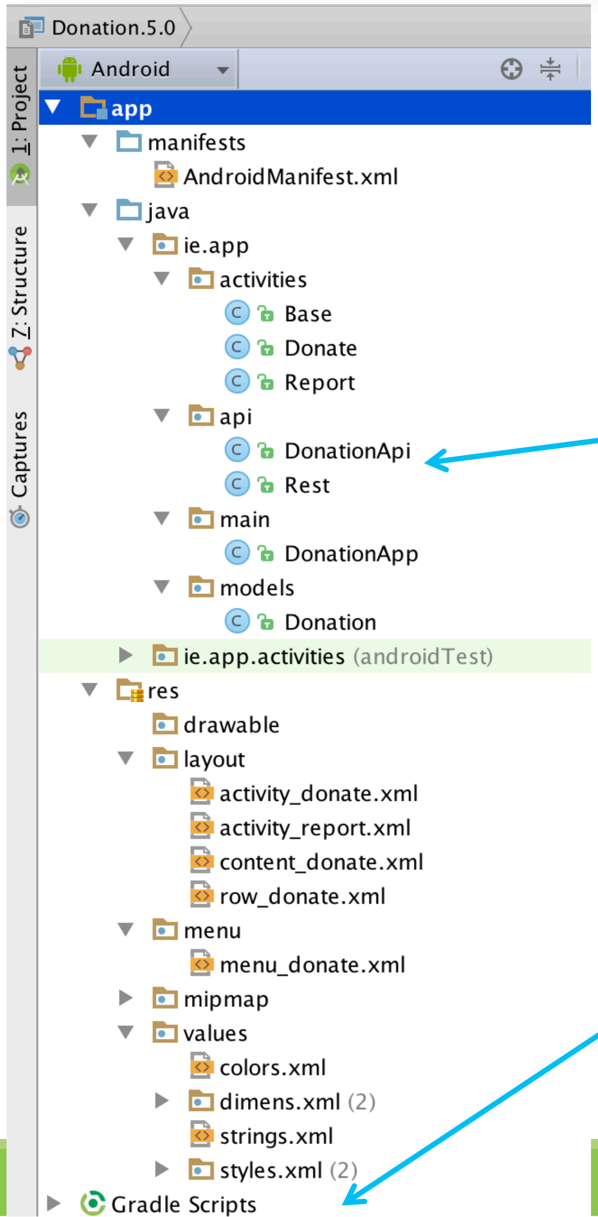
Retrieving Donations List

Delete Donation?

Are you sure you want to Delete the 'Donation with ID ' [56dcab7d88d8f00e000af5e1] ?

NO YES

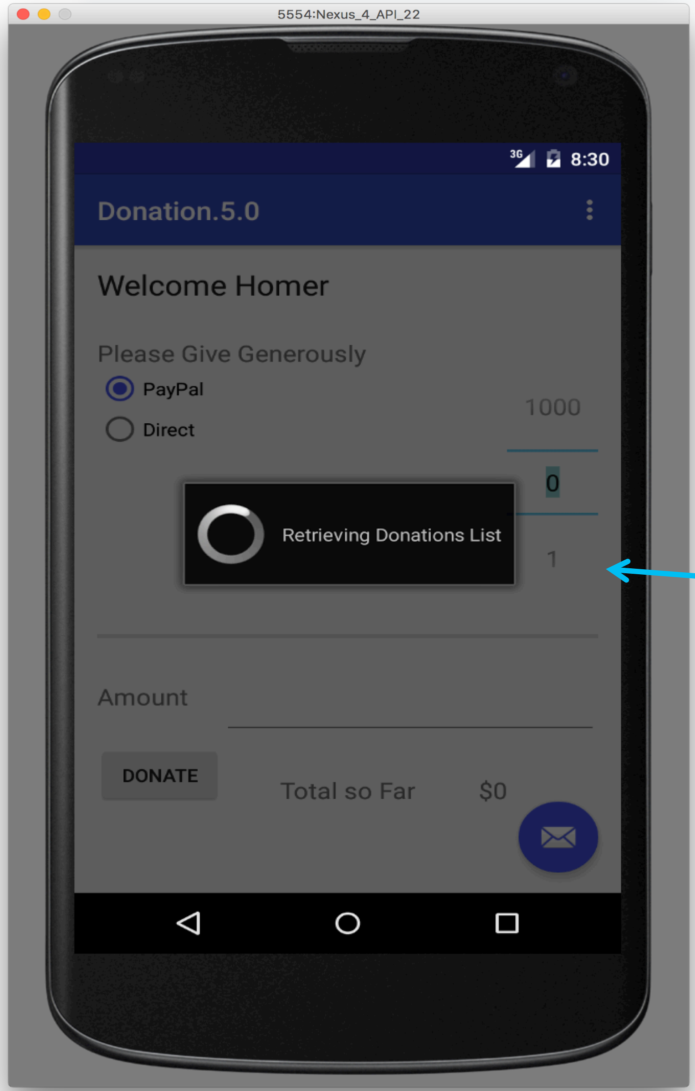
Donation 5.0 Project Structure

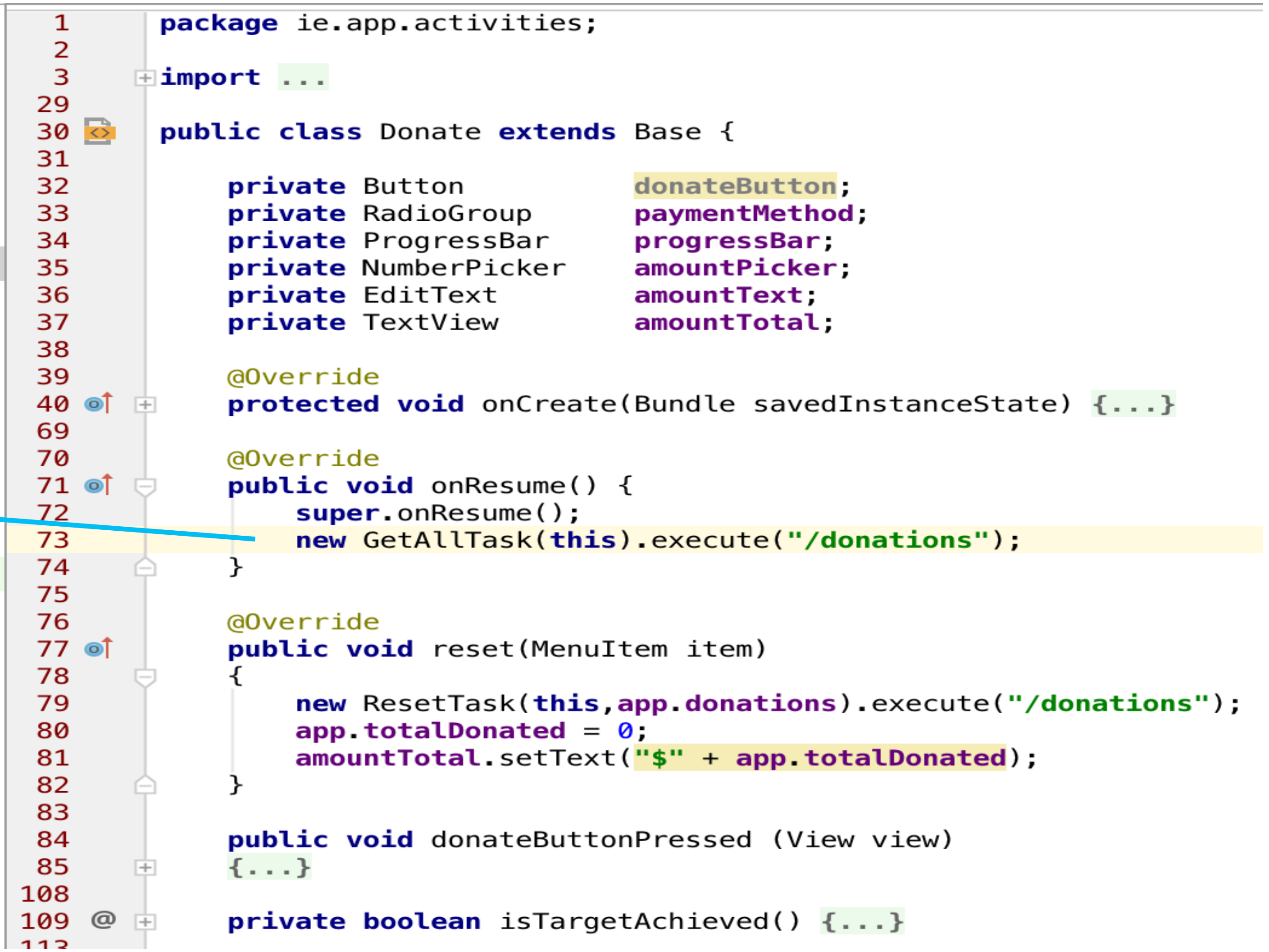


```
Donation.5.0
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ ie.app
      ▼ activities
          Base
          Donate
          Report
      ▼ api
          DonationApi
          Rest
      ▼ main
          DonationApp
      ▼ models
          Donation
    ▶ ie.app.activities (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
```

- api classes for calling REST service (we'll investigate these classes in our networking section)
- Googles gson for parsing JSON strings included in build

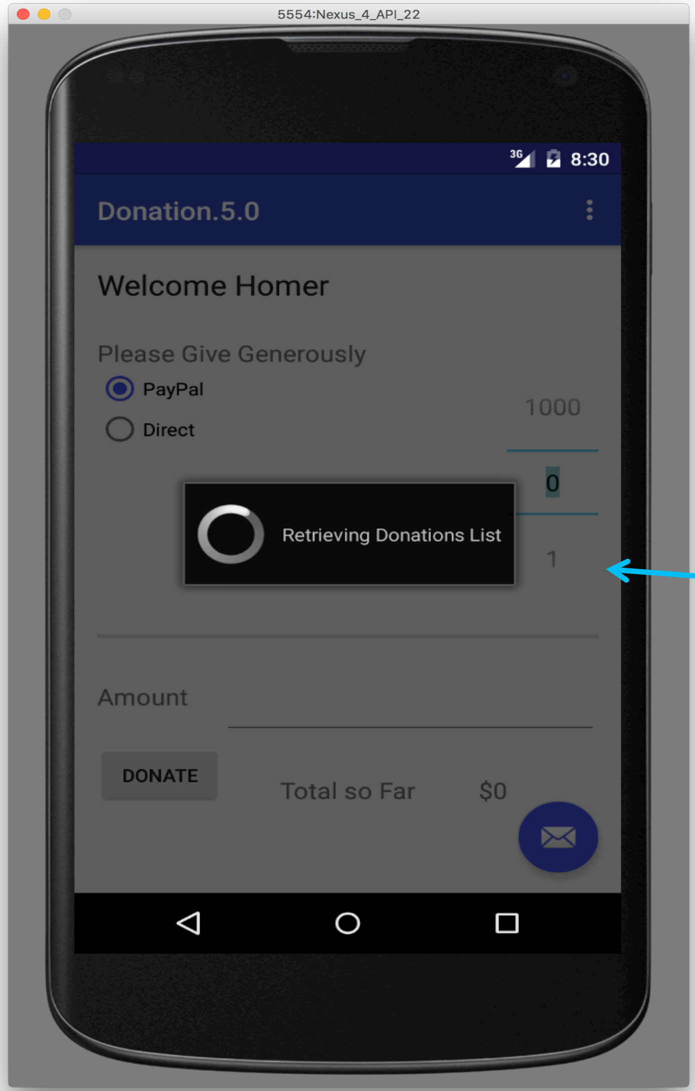
Donation 5.0 – Donate Activity

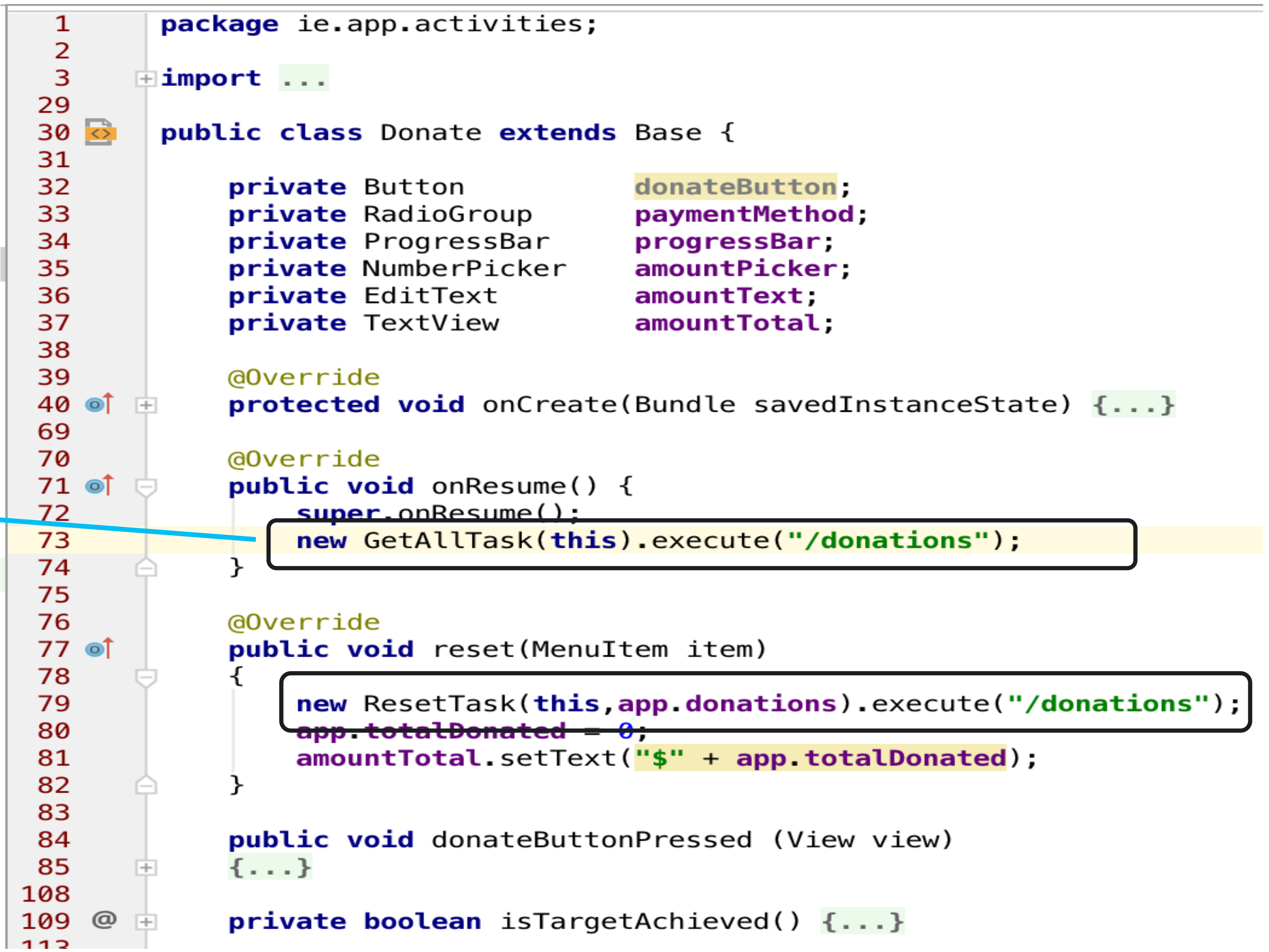




```java
package ie.app.activities;

import ...

public class Donate extends Base {

    private Button          donateButton;
    private RadioGroup      paymentMethod;
    private ProgressBar     progressBar;
    private NumberPicker    amountPicker;
    private EditText        amountText;
    private TextView        amountTotal;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public void onResume() {
        super.onResume();
        new GetAllTask(this).execute("/donations");
    }

    @Override
    public void reset(MenuItem item)
    {
        new ResetTask(this,app.donations).execute("/donations");
        app.totalDonated = 0;
        amountTotal.setText("$" + app.totalDonated);
    }

    public void donateButtonPressed (View view)
    {...}

    private boolean isTargetAchieved() {...}
```

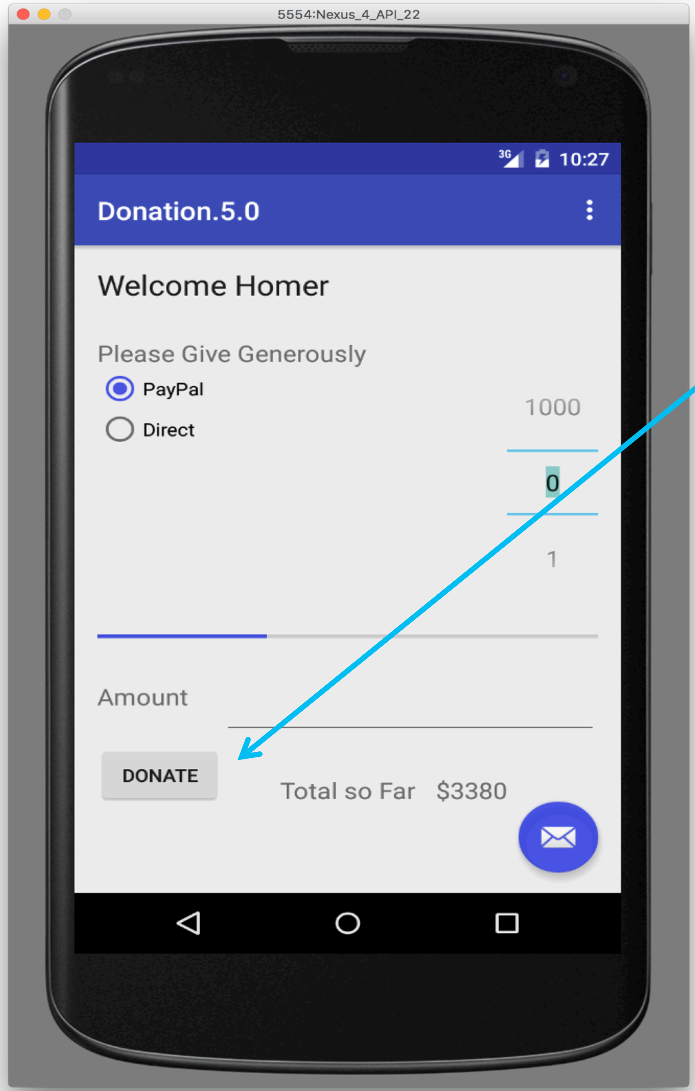
Donation 5.0 – Donate Activity





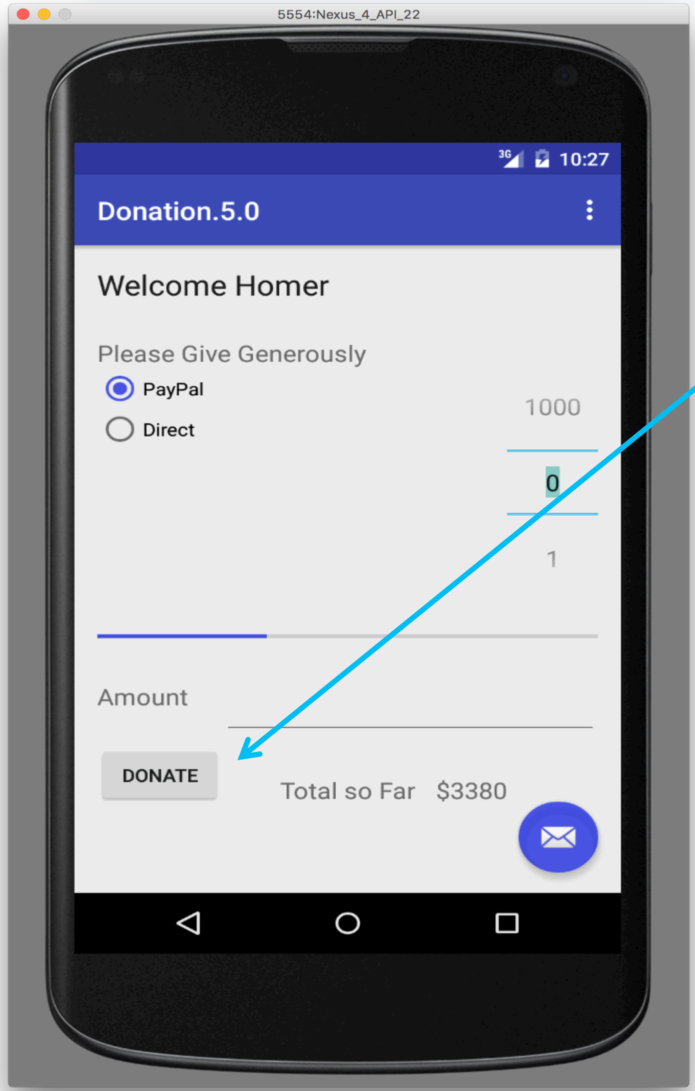
```java
package ie.app.activities;

import ...

public class Donate extends Base {

      private Button             donateButton;
      private RadioGroup         paymentMethod;
      private ProgressBar        progressBar;
      private NumberPicker       amountPicker;
      private EditText           amountText;
      private TextView           amountTotal;

      @Override
      protected void onCreate(Bundle savedInstanceState) {...}

      @Override
      public void onResume() {
          super.onResume();
          new GetAllTask(this).execute("/donations");
      }

      @Override
      public void reset(MenuItem item)
      {
          new ResetTask(this,app.donations).execute("/donations");
          app.totalDonated = 0;
          amountTotal.setText("$" + app.totalDonated);
      }

      public void donateButtonPressed (View view)
      {...}

      private boolean isTargetAchieved() {...}
```

Donation 5.0 – Donate Activity



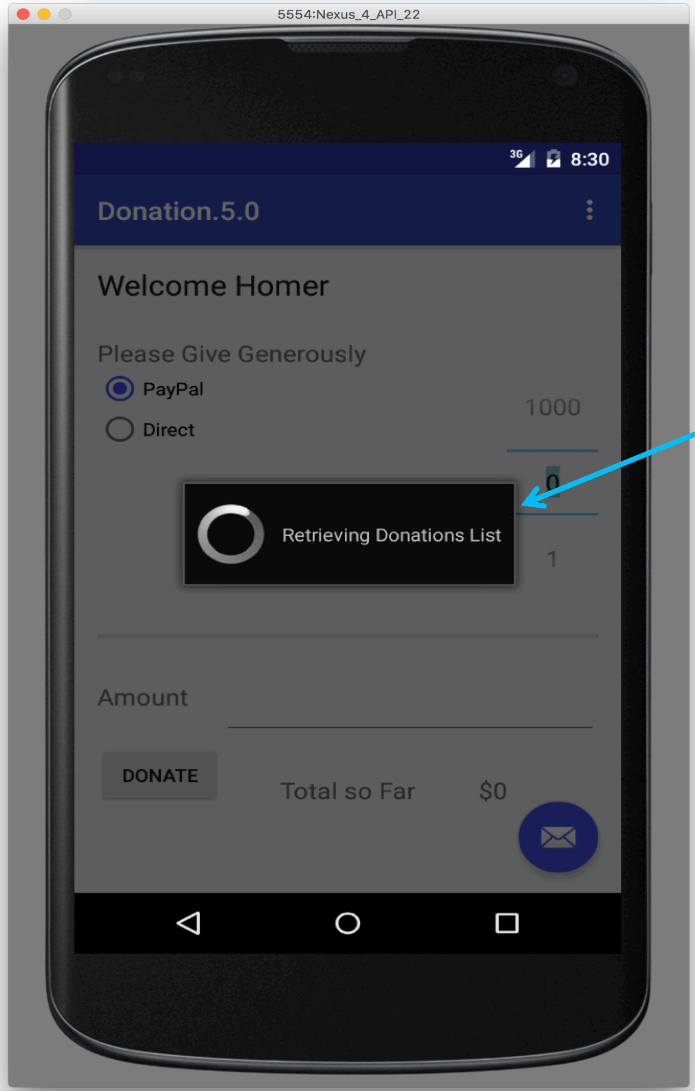
```
public void donateButtonPressed (View view)
{
      String method = paymentMethod.getCheckedRadioButtonId() == R.id.PayPal ? "PayPal" : "Direct"
      int donatedAmount =  amountPicker.getValue();
      if (donatedAmount == 0)
      {
            String text = amountText.getText().toString();
            if (!text.equals(""))
                  donatedAmount = Integer.parseInt(text);
      }
      if (donatedAmount > 0)
      {
            if(!isTargetAchieved()){
                  new InsertTask(this).execute("/donations",new Donation(donatedAmount, method,0));
                  new GetAllTask(this).execute("/donations");

                  progressBar.setProgress(app.totalDonated);
                  String totalDonatedStr = "$" + app.totalDonated;
                  amountTotal.setText(totalDonatedStr);
            }
            else
                  Toast.makeText(this, "Target Exceeded!",Toast.LENGTH_SHORT).show();
      }
}
```

Donation 5.0 – Donate Activity



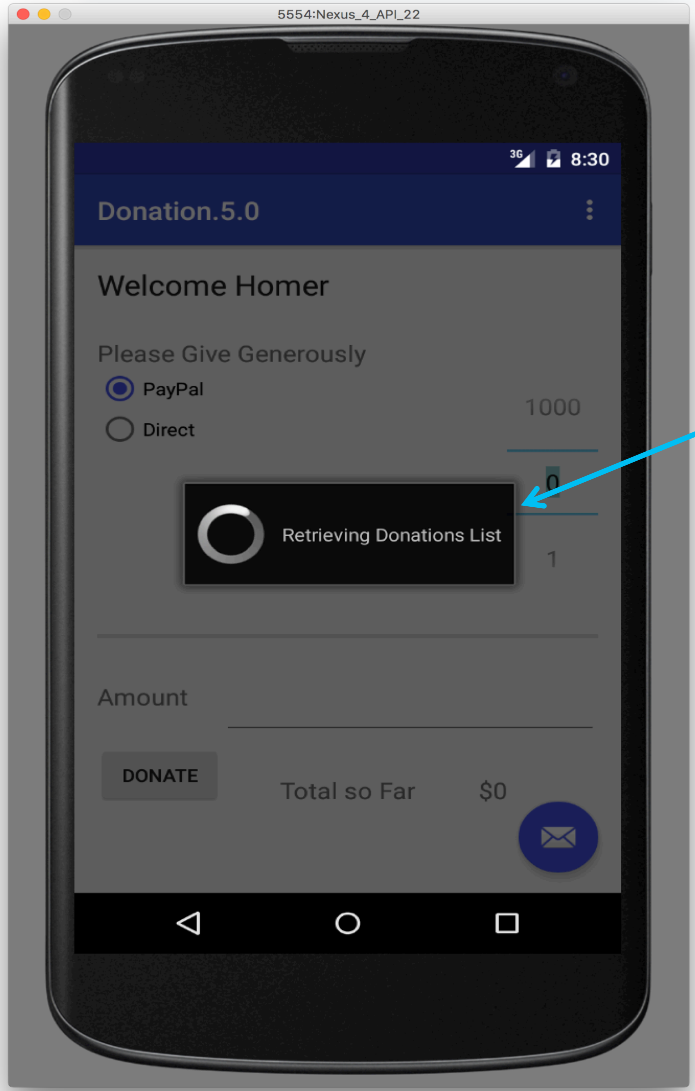
```java
public void donateButtonPressed (View view)
{
    String method = paymentMethod.getCheckedRadioButtonId() == R.id.PayPal ? "PayPal" : "Direct"
    int donatedAmount =  amountPicker.getValue();
    if (donatedAmount == 0)
    {
        String text = amountText.getText().toString();
        if (!text.equals(""))
            donatedAmount = Integer.parseInt(text);
    }
    if (donatedAmount > 0)
    {
        if(!isTargetAchieved()){
            new InsertTask(this).execute("/donations",new Donation(donatedAmount, method,0));
            new GetAllTask(this).execute("/donations");

            progressBar.setProgress(app.totalDonated);
            String totalDonatedStr = "$" + app.totalDonated;
            amountTotal.setText(totalDonatedStr);
        }
        else
            Toast.makeText(this, "Target Exceeded!",Toast.LENGTH_SHORT).show();
    }
}
```

Donation 5.0 – AsyncTask : GetAllTask



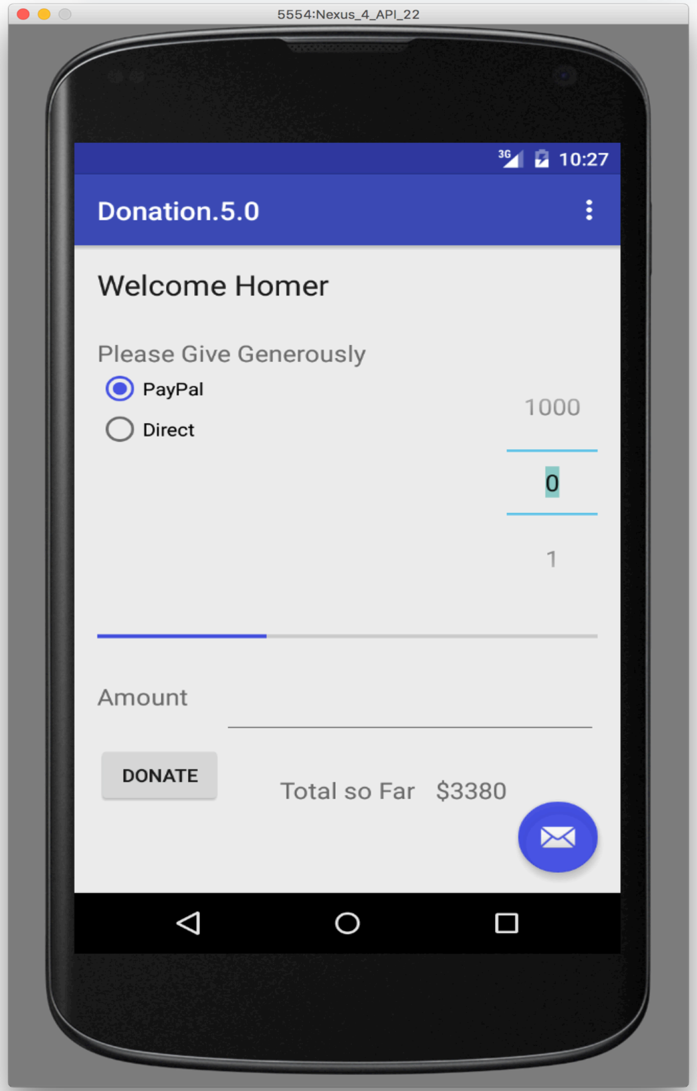
```java
private class GetAllTask extends AsyncTask<String, Void, List<Donation>> {

    protected ProgressDialog         dialog;
    protected Context                context;

    public GetAllTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        this.dialog = new ProgressDialog(context, 1);
        this.dialog.setMessage("Retrieving Donations List");
        this.dialog.show();
    }

    @Override
    protected List<Donation> doInBackground(String... params) {
        try {
            Log.v("donate", "Donation App Getting All Donations");
            return (List<Donation>) DonationApi.getAll((String) params[0]);
        }
        catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(List<Donation> result) {
        super.onPostExecute(result);
        app.totalDonated = 0;
        app.donations = result;
        for (Donation d : app.donations)
            app.totalDonated += d.amount;

        progressBar.setProgress(app.totalDonated);
        amountTotal.setText("$" + app.totalDonated);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – AsyncTask : GetAllTask



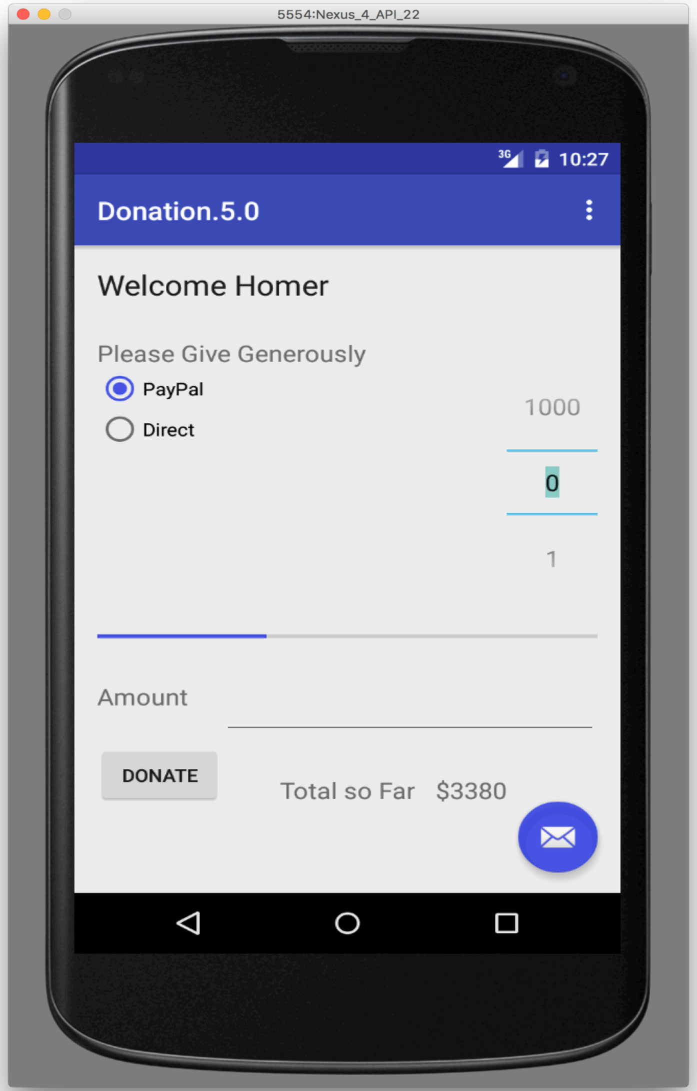
```
private class GetAllTask extends AsyncTask<String, Void, List<Donation>> {

    protected ProgressDialog        dialog;
    protected Context               context;

    public GetAllTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        this.dialog = new ProgressDialog(context, 1);
        this.dialog.setMessage("Retrieving Donations List");
        this.dialog.show();
    }

    @Override
    protected List<Donation> doInBackground(String... params) {
        try {
            Log.v("donate", "Donation App Getting All Donations");
            return (List<Donation>) DonationApi.getAll((String) params[0]);
        }
        catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(List<Donation> result) {
        super.onPostExecute(result);
        app.totalDonated = 0;
        app.donations = result;
        for (Donation d : app.donations)
            app.totalDonated += d.amount;

        progressBar.setProgress(app.totalDonated);
        amountTotal.setText("$" + app.totalDonated);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – AsyncTask : InsertTask



```java
private class InsertTask extends AsyncTask<Object, Void, String> {

    protected ProgressDialog dialog;
    protected Context context;

    public InsertTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        this.dialog = new ProgressDialog(context, 1);
        this.dialog.setMessage("Saving Donation....");
        this.dialog.show();
    }

    @Override
    protected String doInBackground(Object... params) {

        String res = null;
        try {
            Log.v("donate", "Donation App Inserting");
            res = DonationApi.insert((String) params[0], (Donation) params[1]);
        }

        catch(Exception e)
        {
            Log.v("donate","ERROR : " + e);
            e.printStackTrace();
        }
        return res;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

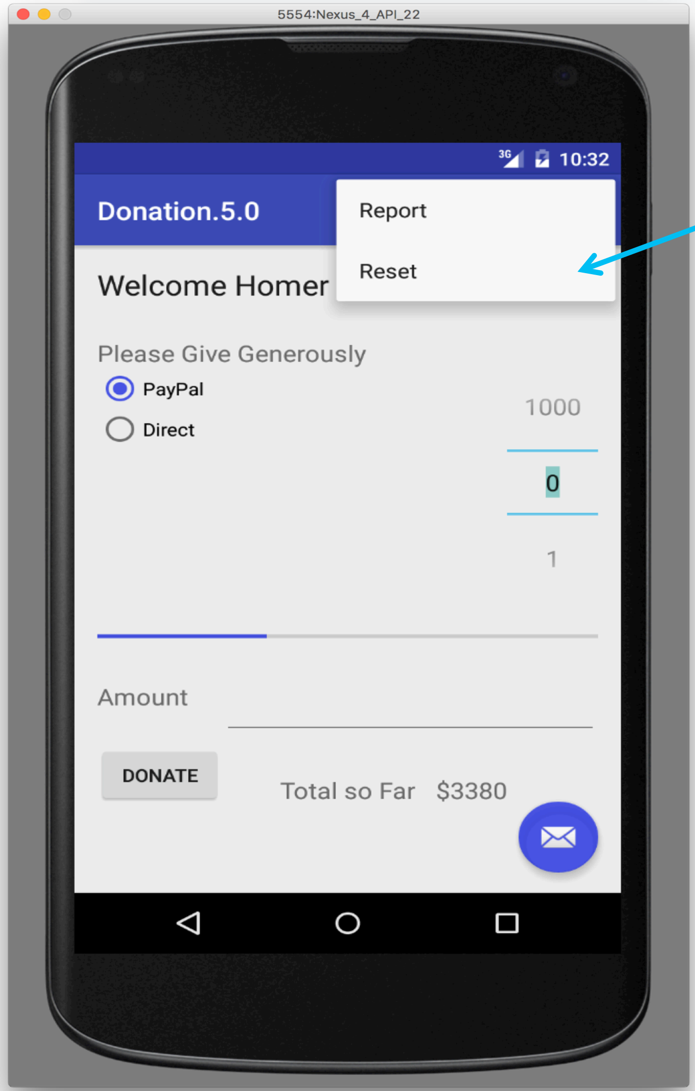
Donation 5.0 – AsyncTask : InsertTask



```java
private class InsertTask extends AsyncTask<Object, Void, String> {

    protected ProgressDialog dialog;
    protected Context context;

    public InsertTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        this.dialog = new ProgressDialog(context, 1);
        this.dialog.setMessage("Saving Donation....");
        this.dialog.show();
    }

    @Override
    protected String doInBackground(Object... params) {

        String res = null;
        try {
            Log.v("donate", "Donation App Inserting");
            res = DonationApi.insert((String) params[0], (Donation) params[1]);
        }

        catch(Exception e)
        {
            Log.v("donate","ERROR : " + e);
            e.printStackTrace();
        }
        return res;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

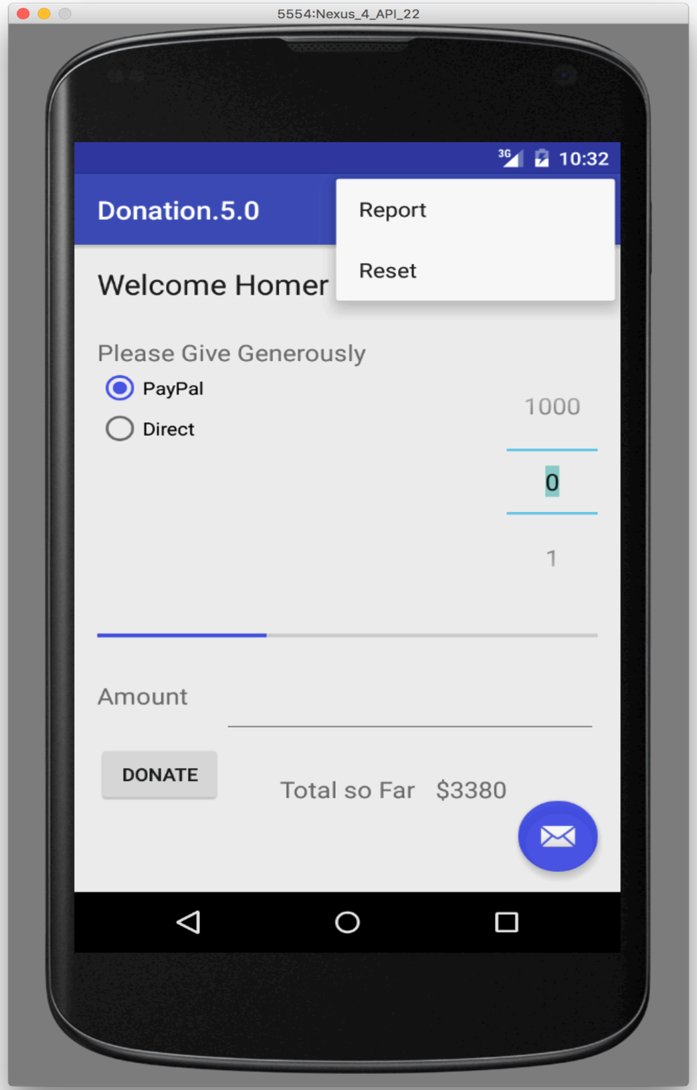
Donation 5.0 – ResetTask

```
@Override
public void reset(MenuItem item)
{
    new ResetTask(this,app.donations).execute("/donations");
    app.totalDonated = 0;
    amountTotal.setText("$" + app.totalDonated);
}
```



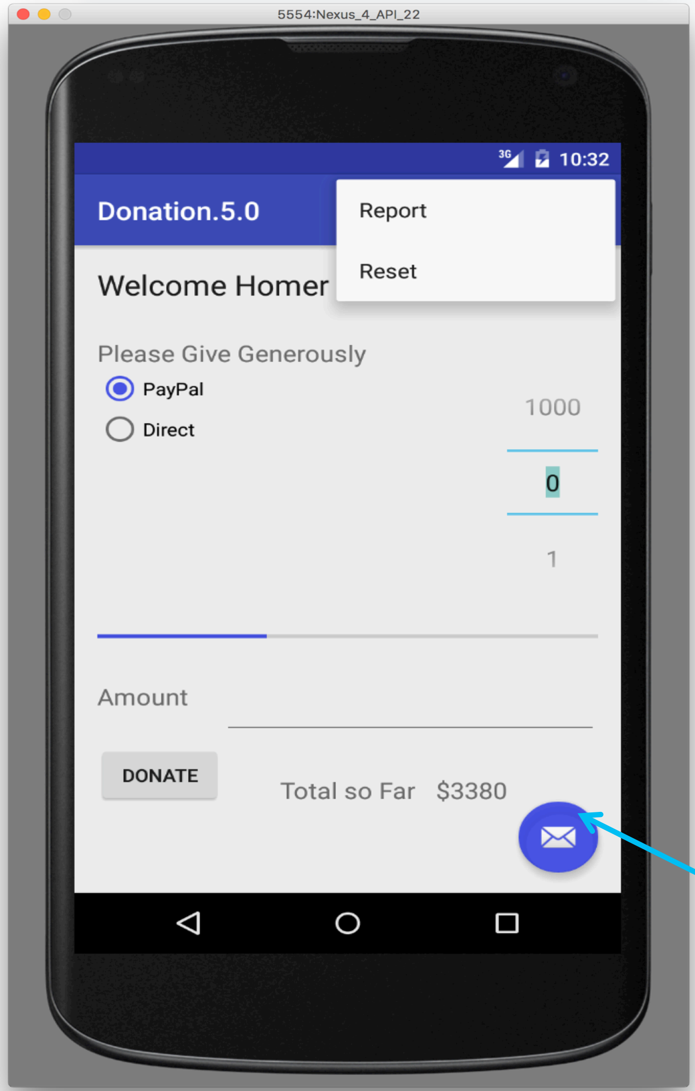
Donation 5.0 – ResetTask

```
@Override
public void reset(MenuItem item)
{
    new ResetTask(this,app.donations).execute("/donations");
    app.totalDonated = 0;
    amountTotal.setText("$" + app.totalDonated);
}
```



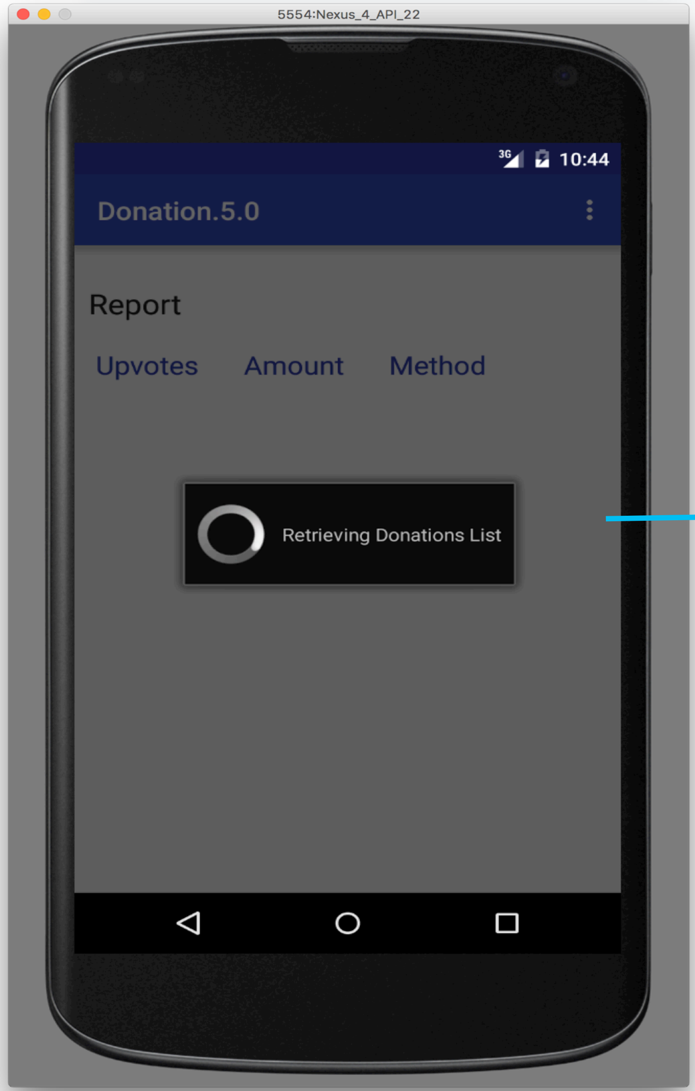
```
private class ResetTask extends AsyncTask<Object, Void, String> {

    protected ProgressDialog        dialog;
    protected Context               context;
    protected List<Donation>        donations;

    public ResetTask(Context context, List<Donation> donations)
    {...}

    @Override
    protected void onPreExecute() {...}

    @Override
    protected String doInBackground(Object... params) {

        String res = null;
        try {
                res = DonationApi.deleteAll((String)params[0]);
        }

        catch(Exception e)
        {
            Log.v("donate"," RESET ERROR : " + e);
            e.printStackTrace();
        }
        return res;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        app.totalDonated = 0;
        progressBar.setProgress(app.totalDonated);
        amountTotal.setText("$" + app.totalDonated);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – ResetTask



```java
private class ResetTask extends AsyncTask<Object, Void, String> {

    protected ProgressDialog         dialog;
    protected Context                context;
    protected List<Donation>         donations;

    public ResetTask(Context context, List<Donation> donations)
    {...}

    @Override
    protected void onPreExecute() {...}

    @Override
    protected String doInBackground(Object... params) {

        String res = null;
        try {
            res = DonationApi.deleteAll((String)params[0]);
        }

        catch(Exception e)
        {
            Log.v("donate"," RESET ERROR : " + e);
            e.printStackTrace();
        }
        return res;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        app.totalDonated = 0;
        progressBar.setProgress(app.totalDonated);
        amountTotal.setText("$" + app.totalDonated);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report Activity


5554:Nexus_4_API_22

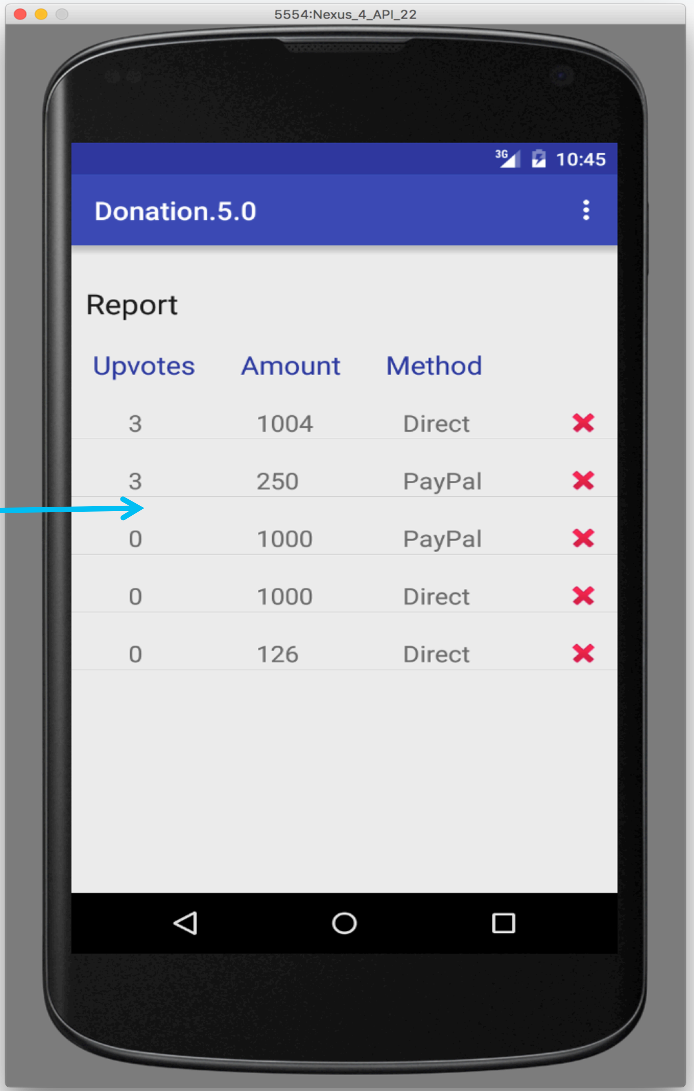
10:44

Donation.5.0

Report

Upvotes Amount Method

Retrieving Donations List


5554:Nexus_4_API_22

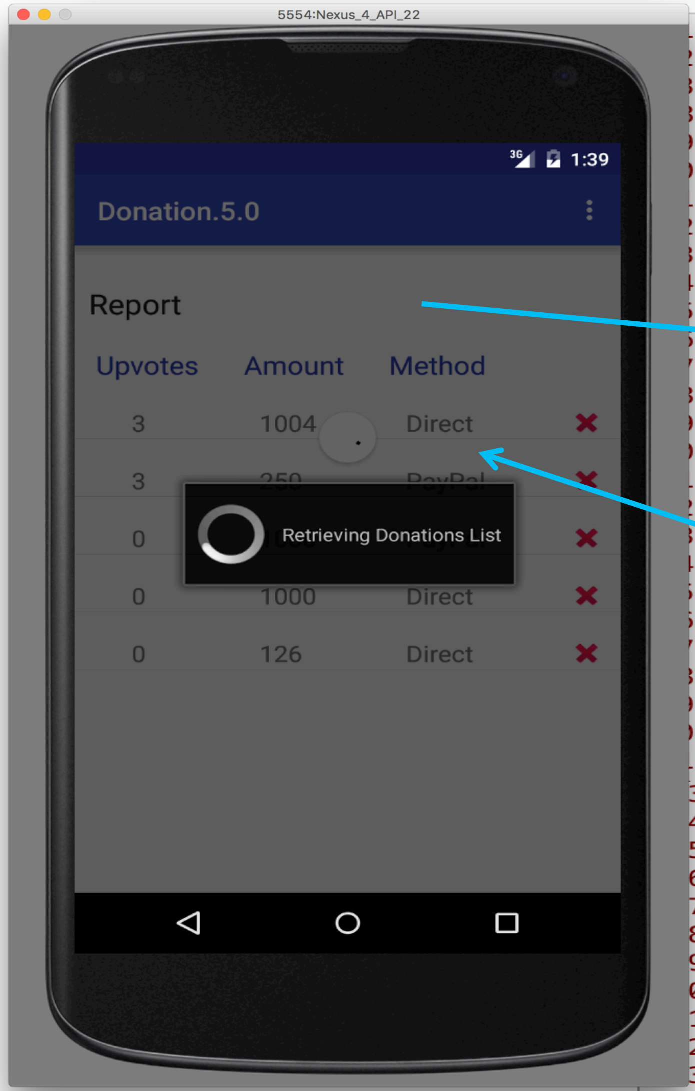
10:45

Donation.5.0

Report

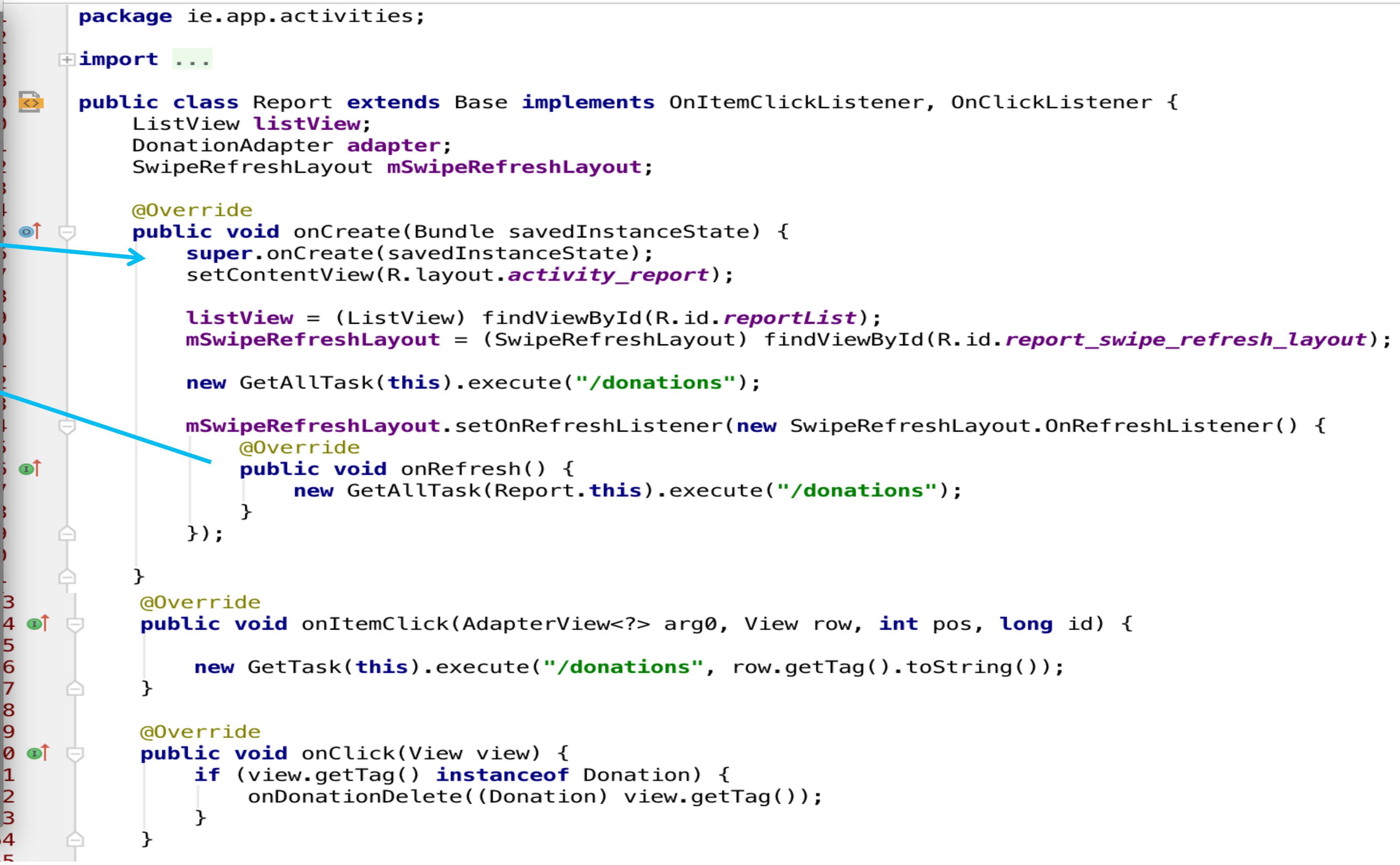
Upvotes	Amount	Method	
3	1004	Direct	✗
3	250	PayPal	✗
0	1000	PayPal	✗
0	1000	Direct	✗
0	126	Direct	✗

Donation 5.0 – Report Activity



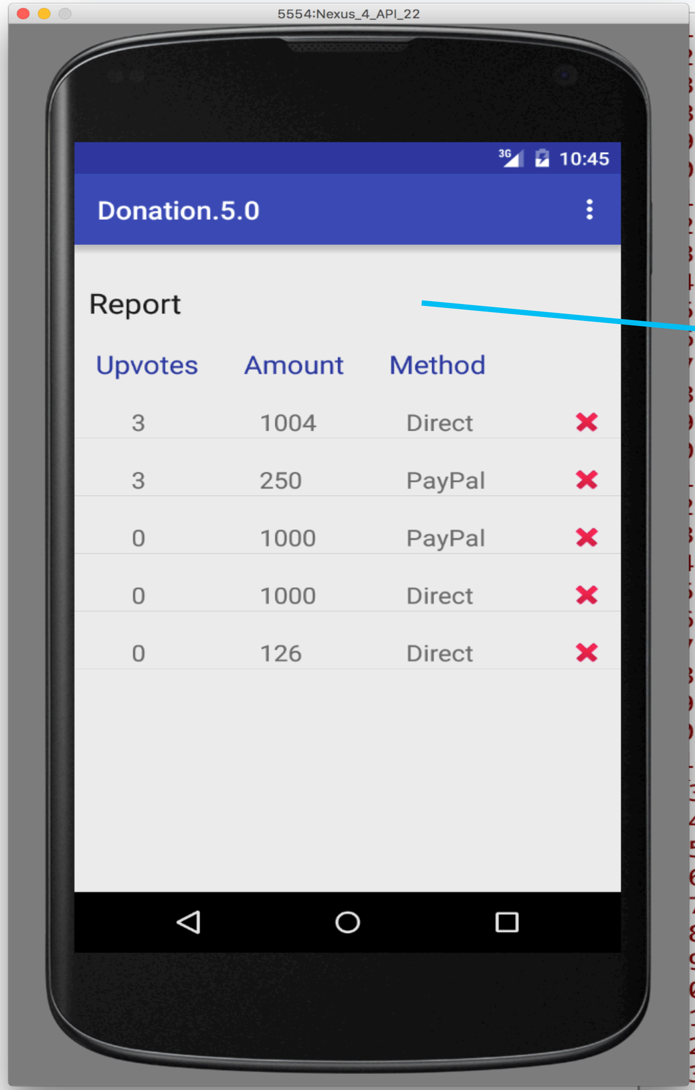
Upvotes	Amount	Method
3	1004	Direct
3	250	PayPal
0	[illegible]	[illegible]
0	1000	Direct
0	126	Direct

Retrieving Donations List



```java
package ie.app.activities;

import ...

public class Report extends Base implements OnItemClickListener, OnClickListener {
    ListView listView;
    DonationAdapter adapter;
    SwipeRefreshLayout mSwipeRefreshLayout;

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

        listView = (ListView) findViewById(R.id.reportList);
        mSwipeRefreshLayout = (SwipeRefreshLayout) findViewById(R.id.report_swipe_refresh_layout);

        new GetAllTask(this).execute("/donations");

        mSwipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
            @Override
            public void onRefresh() {
                new GetAllTask(Report.this).execute("/donations");
            }
        });

    }
    @Override
    public void onItemClick(AdapterView<?> arg0, View row, int pos, long id) {

        new GetTask(this).execute("/donations", row.getTag().toString());
    }

    @Override
    public void onClick(View view) {
        if (view.getTag() instanceof Donation) {
            onDonationDelete((Donation) view.getTag());
        }
    }
```

Donation 5.0 – Report Activity

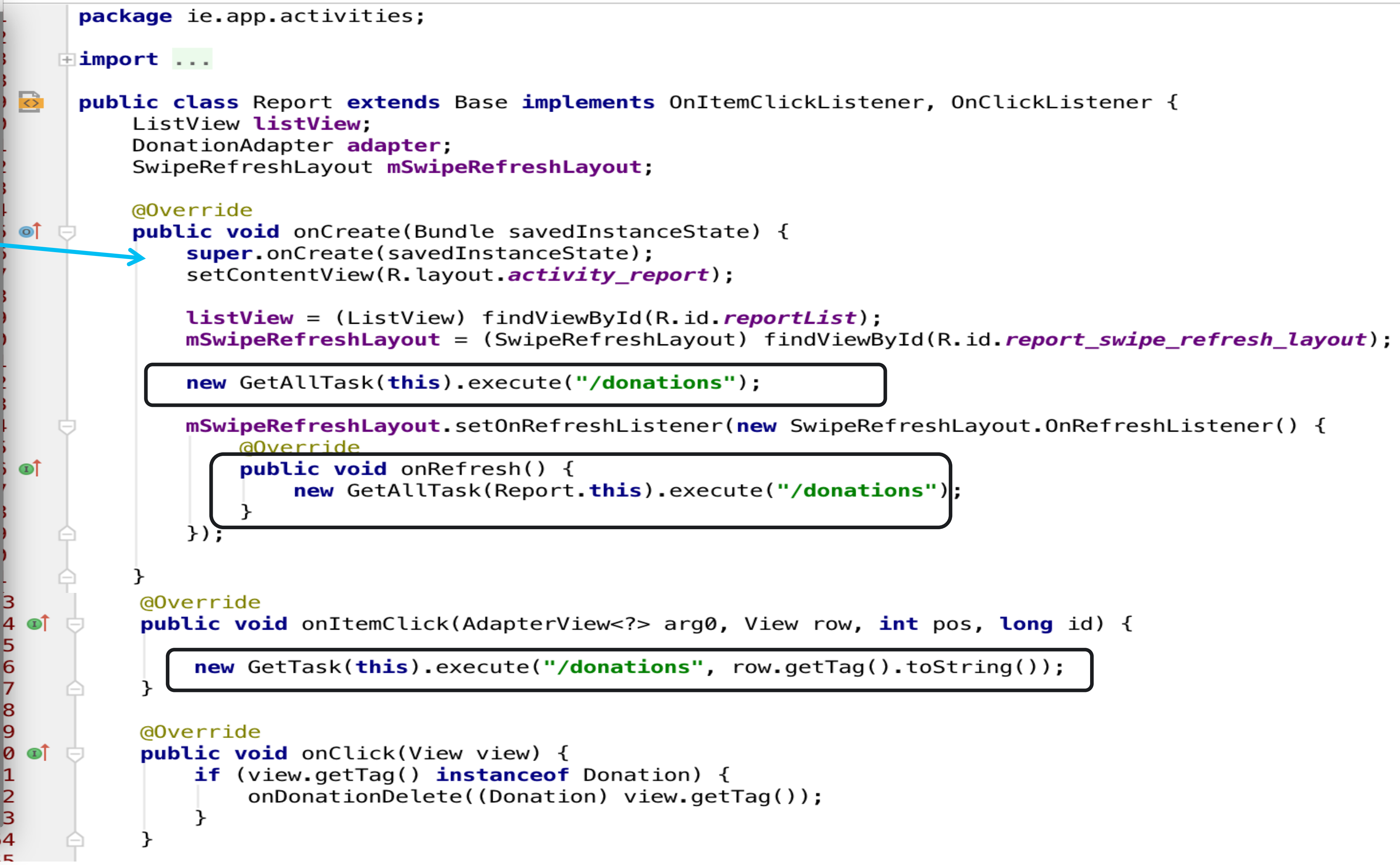


5554:Nexus_4_API_22

Donation.5.0

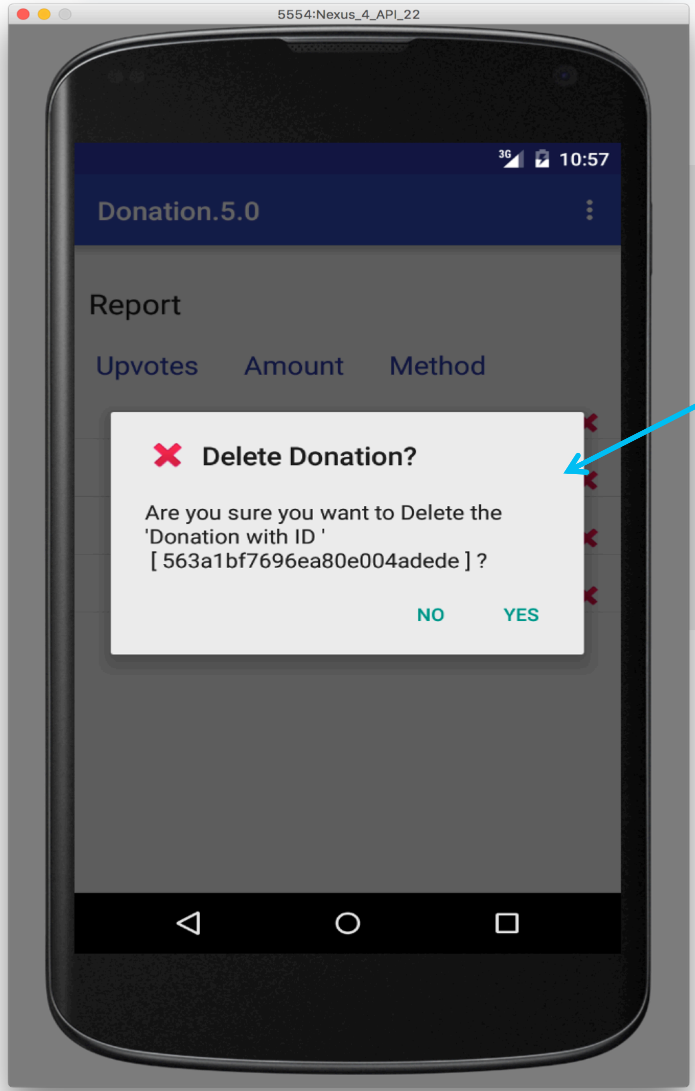
Report

Upvotes	Amount	Method	
3	1004	Direct	✗
3	250	PayPal	✗
0	1000	PayPal	✗
0	1000	Direct	✗
0	126	Direct	✗



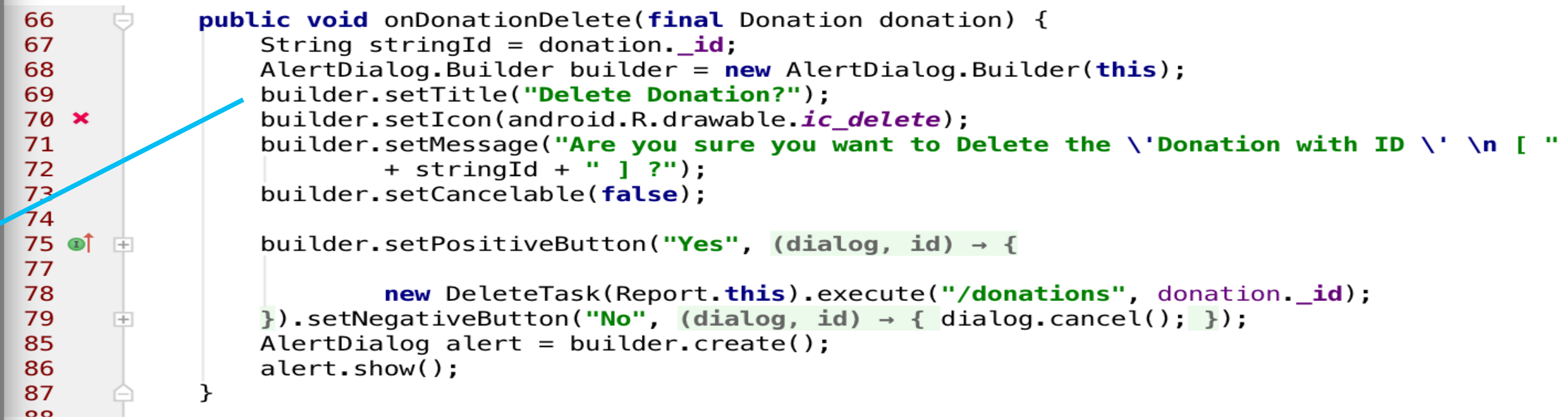
```java
package ie.app.activities;

import ...

public class Report extends Base implements OnItemClickListener, OnClickListener {
    ListView listView;
    DonationAdapter adapter;
    SwipeRefreshLayout mSwipeRefreshLayout;

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

        listView = (ListView) findViewById(R.id.reportList);
        mSwipeRefreshLayout = (SwipeRefreshLayout) findViewById(R.id.report_swipe_refresh_layout);

        new GetAllTask(this).execute("/donations");

        mSwipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
            @Override
            public void onRefresh() {
                new GetAllTask(Report.this).execute("/donations");
            }
        });

    }
     @Override
     public void onItemClick(AdapterView<?> arg0, View row, int pos, long id) {

         new GetTask(this).execute("/donations", row.getTag().toString());
     }

     @Override
     public void onClick(View view) {
         if (view.getTag() instanceof Donation) {
             onDonationDelete((Donation) view.getTag());
         }
     }
```

Donation 5.0 – Report Activity



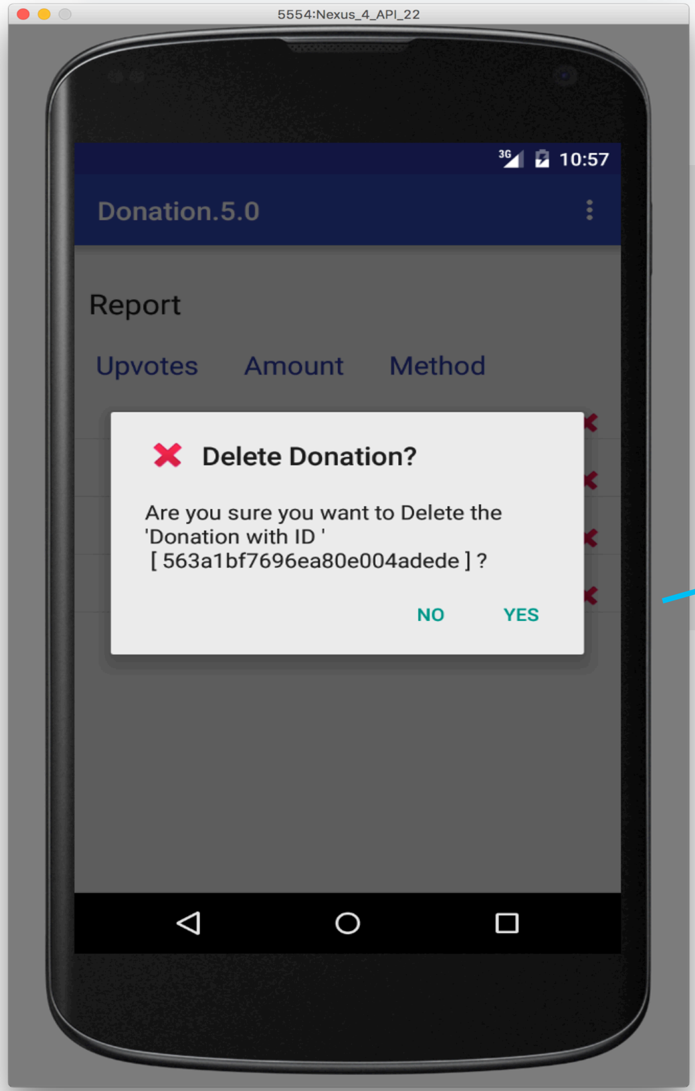
Delete Donation?

Are you sure you want to Delete the 'Donation with ID ' [563a1bf7696ea80e004adede] ?

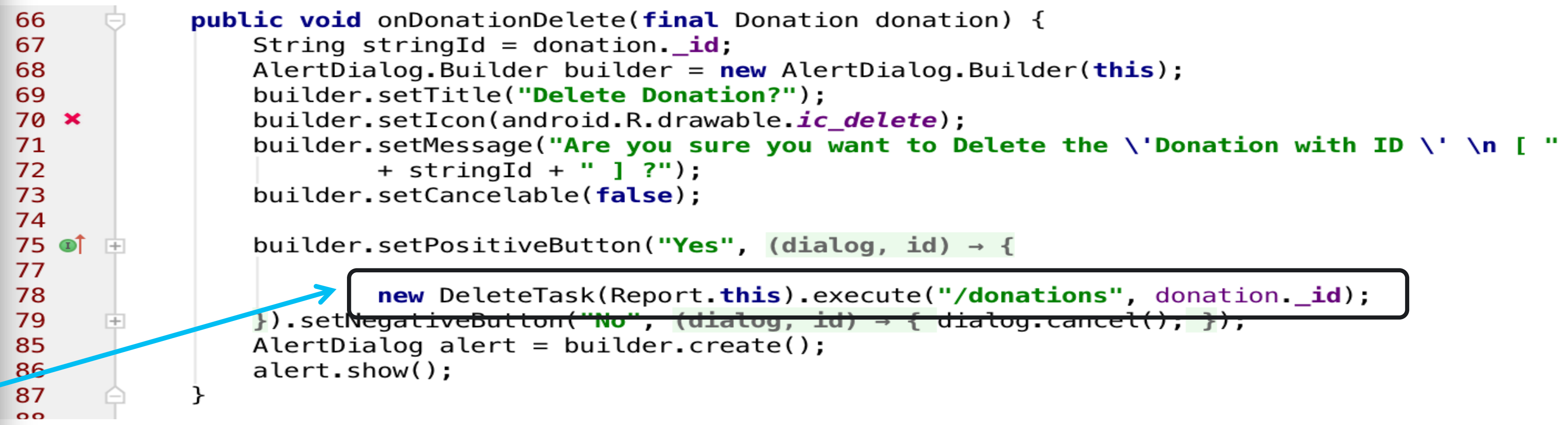


```java
public void onDonationDelete(final Donation donation) {
    String stringId = donation._id;
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Delete Donation?");
    builder.setIcon(android.R.drawable.ic_delete);
    builder.setMessage("Are you sure you want to Delete the \'Donation with ID \' \n [ "
            + stringId + " ] ?");
    builder.setCancelable(false);

    builder.setPositiveButton("Yes", (dialog, id) → {

            new DeleteTask(Report.this).execute("/donations", donation._id);
    }).setNegativeButton("No", (dialog, id) → { dialog.cancel(); });
    AlertDialog alert = builder.create();
    alert.show();
}
```

Donation 5.0 – Report Activity

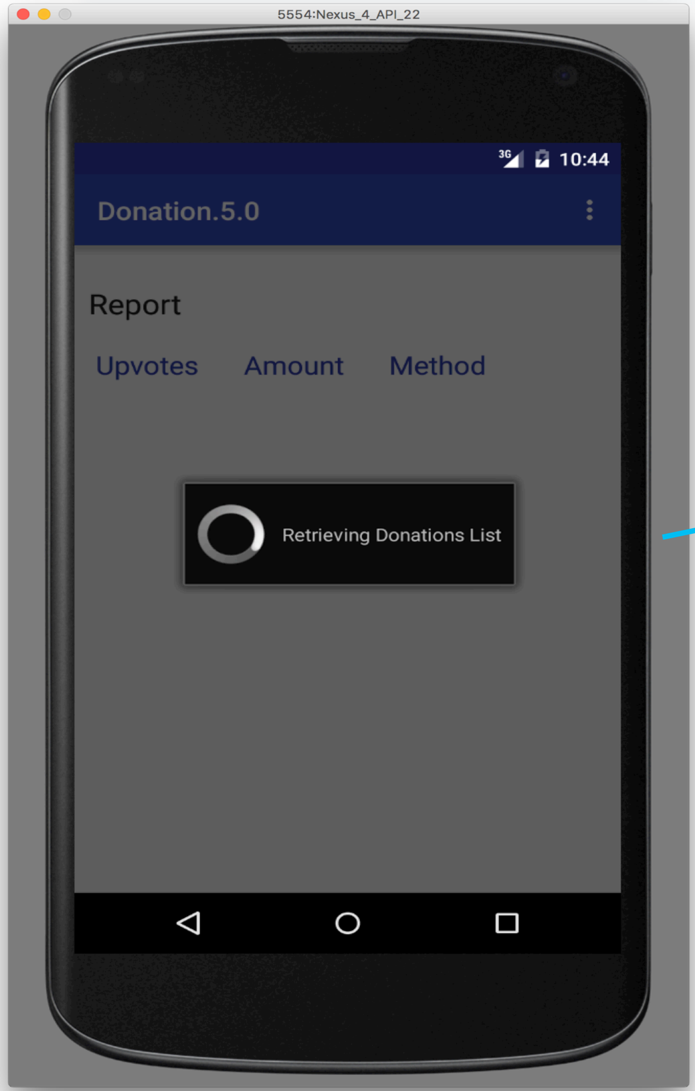


Are you sure you want to Delete the 'Donation with ID ' [563a1bf7696ea80e004adede] ?



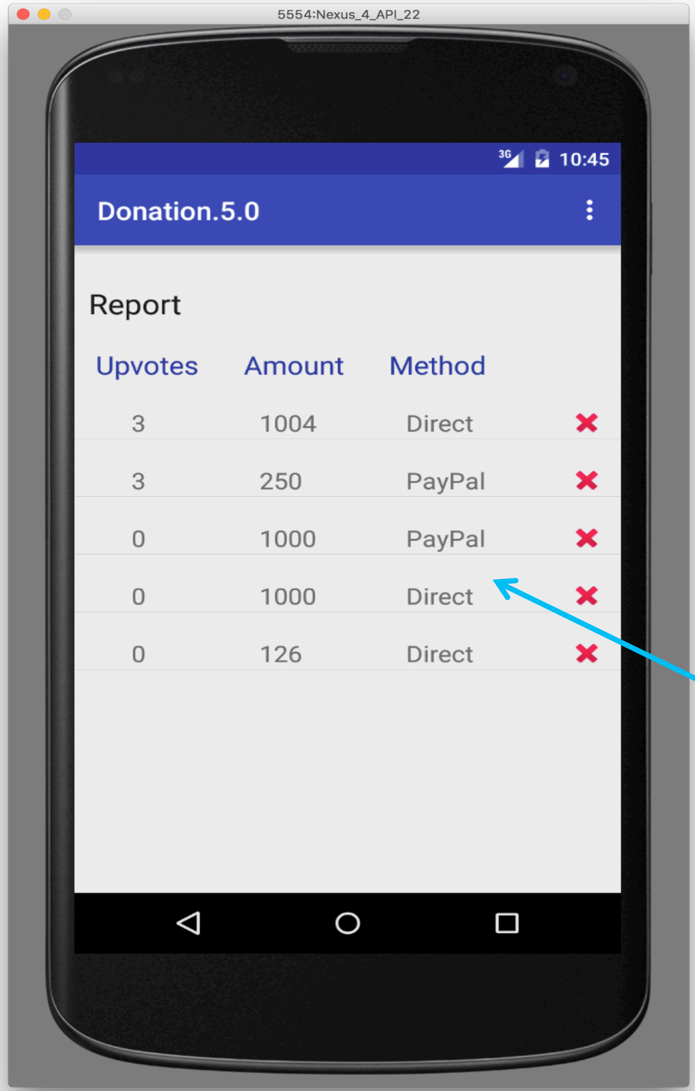
```
public void onDonationDelete(final Donation donation) {
    String stringId = donation._id;
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Delete Donation?");
    builder.setIcon(android.R.drawable.ic_delete);
    builder.setMessage("Are you sure you want to Delete the \'Donation with ID \' \n [ "
            + stringId + " ] ?");
    builder.setCancelable(false);

    builder.setPositiveButton("Yes", (dialog, id) → {

            new DeleteTask(Report.this).execute("/donations", donation._id);
    }).setNegativeButton("No", (dialog, id) → { dialog.cancel(); });
    AlertDialog alert = builder.create();
    alert.show();
}
```

Donation 5.0 – Report : GetAllTask



```java
private class GetAllTask extends AsyncTask<String, Void, List<Donation>> {

    protected ProgressDialog dialog;
    protected Context context;

    public GetAllTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected List<Donation> doInBackground(String... params) {

        try {
            return (List<Donation>) DonationApi.getAll((String) params[0]);
        } catch (Exception e) {
            Log.v("ASYNC", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(List<Donation> result) {
        super.onPostExecute(result);

        app.donations = result;
        adapter = new DonationAdapter(context, app.donations);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(Report.this);
        mSwipeRefreshLayout.setRefreshing(false);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : GetAllTask

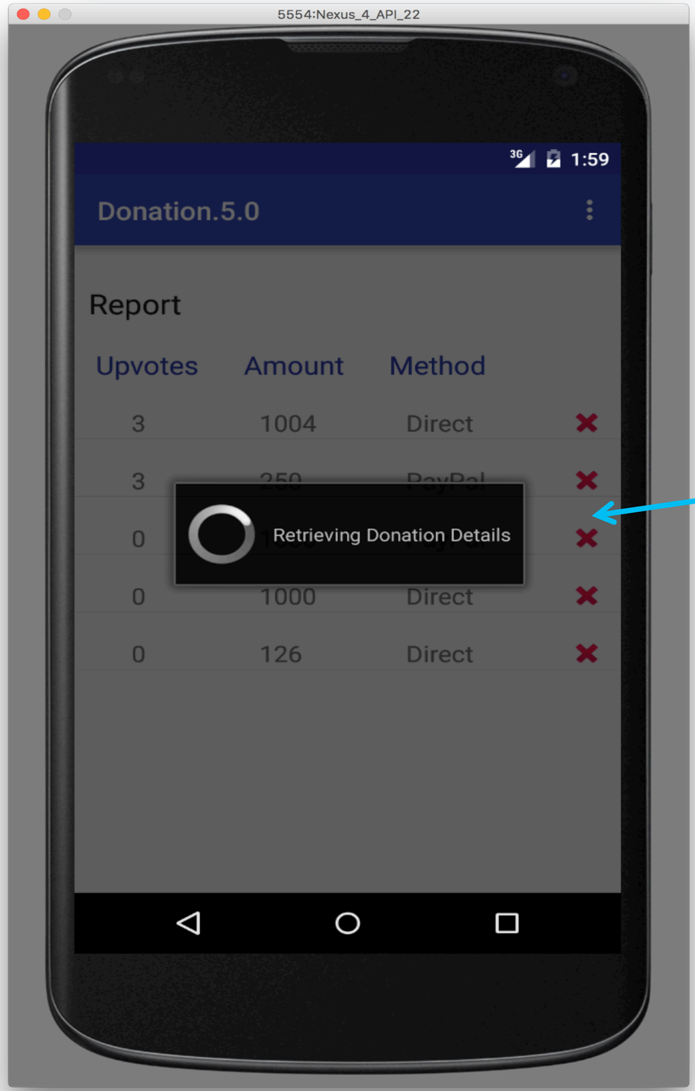


Report

Upvotes	Amount	Method
3	1004	Direct
3	250	PayPal
0	1000	PayPal
0	1000	Direct
0	126	Direct

```
private class GetAllTask extends AsyncTask<String, Void, List<Donation>> {

    protected ProgressDialog dialog;
    protected Context context;

    public GetAllTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected List<Donation> doInBackground(String... params) {

        try {
            return (List<Donation>) DonationApi.getAll((String) params[0]);
        } catch (Exception e) {
            Log.v("ASYNC", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(List<Donation> result) {
        super.onPostExecute(result);

        app.donations = result;
        adapter = new DonationAdapter(context, app.donations);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(Report.this);
        mSwipeRefreshLayout.setRefreshing(false);

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : GetTask

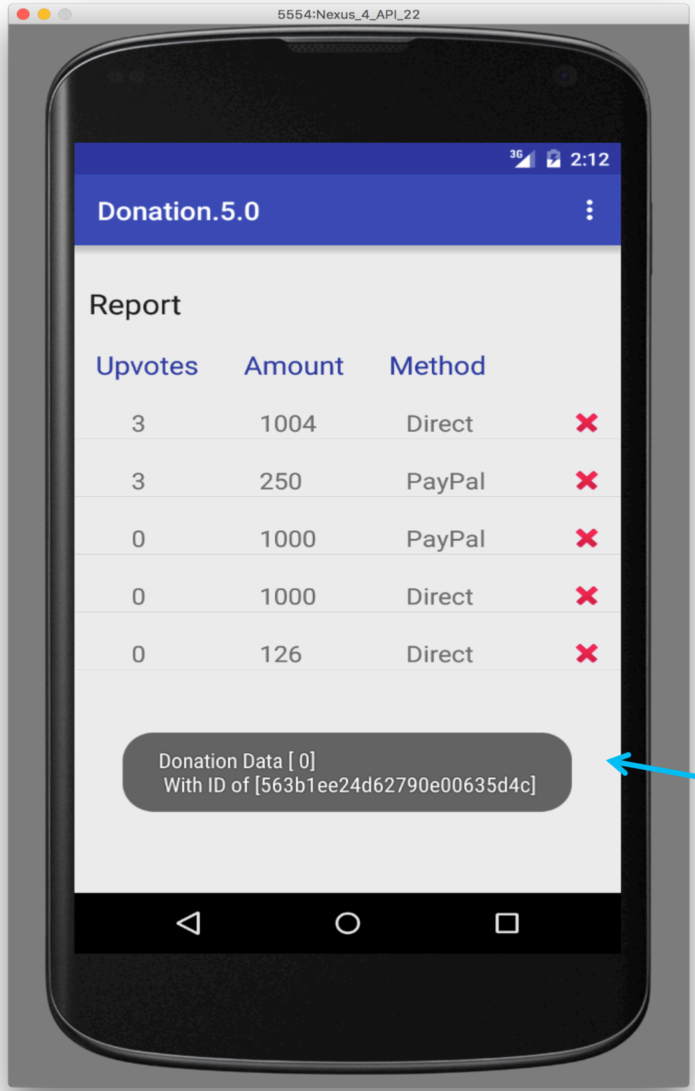


Upvotes	Amount	Method
3	1004	Direct
3	250	PayPal
0	[illegible]	
0	1000	Direct
0	126	Direct

Retrieving Donation Details

```java
private class GetTask extends AsyncTask<String, Void, Donation> {

    protected ProgressDialog dialog;
    protected Context context;

    public GetTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected Donation doInBackground(String... params) {

        try {
            return (Donation) DonationApi.get((String) params[0], (String) params[1]);
        } catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Donation result) {
        super.onPostExecute(result);

        Donation donation = result;

        Toast.makeText(Report.this, "Donation Data [ " + donation.upvotes + "]\n " +
                "With ID of [" + donation._id + "]", Toast.LENGTH_LONG).show();

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : GetTask



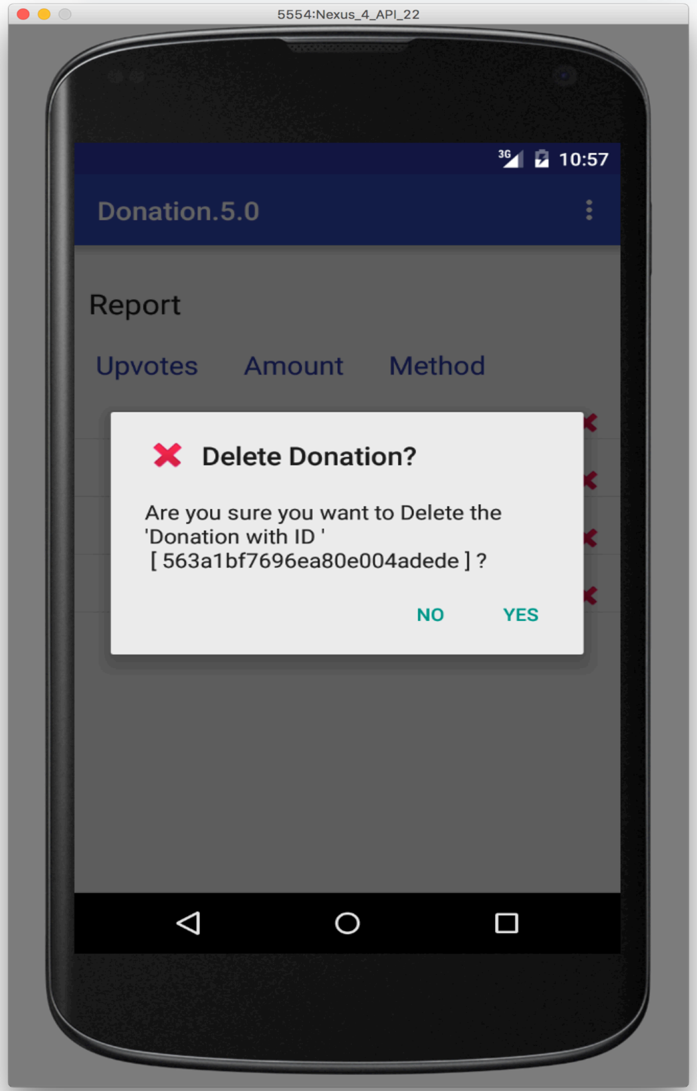
Report

Upvotes	Amount	Method	
3	1004	Direct	✖
3	250	PayPal	✖
0	1000	PayPal	✖
0	1000	Direct	✖
0	126	Direct	✖

```
Donation Data [ 0]
 With ID of [563b1ee24d62790e00635d4c]
```

```java
private class GetTask extends AsyncTask<String, Void, Donation> {

    protected ProgressDialog dialog;
    protected Context context;

    public GetTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected Donation doInBackground(String... params) {

        try {
            return (Donation) DonationApi.get((String) params[0], (String) params[1]);
        } catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Donation result) {
        super.onPostExecute(result);

        Donation donation = result;

        Toast.makeText(Report.this, "Donation Data [ " + donation.upvotes + "]\n " +
                "With ID of [" + donation._id + "]", Toast.LENGTH_LONG).show();

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : DeleteTask


5554:Nexus_4_API_22

10:57

Donation.5.0

Report

Upvotes Amount Method

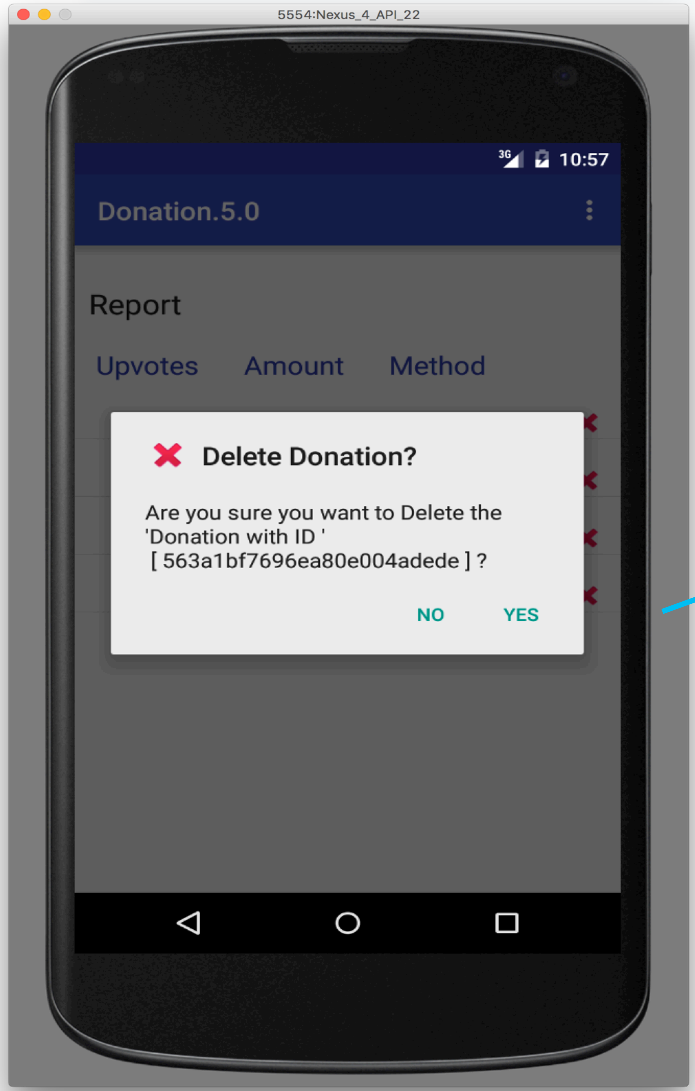
Delete Donation?

Are you sure you want to Delete the 'Donation with ID ' [563a1bf7696ea80e004adede] ?

NO YES

```
private class DeleteTask extends AsyncTask<String, Void, String> {

    protected ProgressDialog dialog;
    protected Context context;

    public DeleteTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected String doInBackground(String... params) {

        try {
            return (String) DonationApi.delete((String) params[0], (String) params[1]);
        } catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        String s = result;
        Log.v("donate", "DELETE REQUEST : " + s);

        new GetAllTask(Report.this).execute("/donations");

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : DeleteTask


5554:Nexus_4_API_22

10:57

Donation.5.0

Report

Upvotes Amount Method

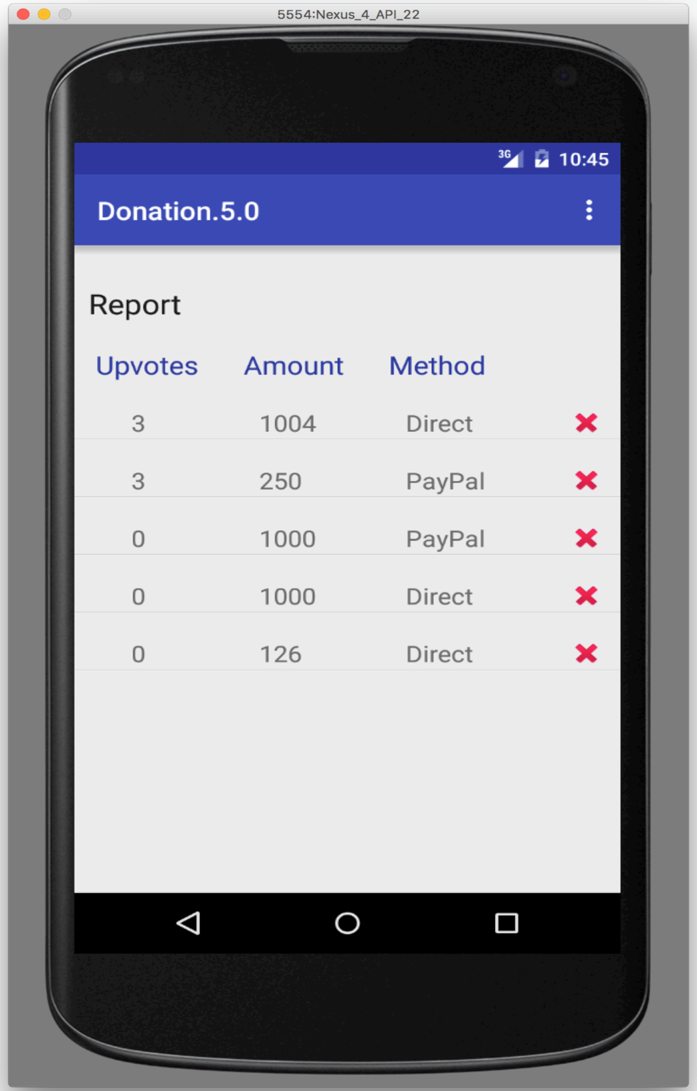
Delete Donation?

Are you sure you want to Delete the 'Donation with ID ' [563a1bf7696ea80e004adede] ?

NO YES

```
private class DeleteTask extends AsyncTask<String, Void, String> {

    protected ProgressDialog dialog;
    protected Context context;

    public DeleteTask(Context context) { this.context = context; }

    @Override
    protected void onPreExecute() {...}

    @Override
    protected String doInBackground(String... params) {

        try {
            return (String) DonationApi.delete((String) params[0], (String) params[1]);
        } catch (Exception e) {
            Log.v("donate", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        String s = result;
        Log.v("donate", "DELETE REQUEST : " + s);

        new GetAllTask(Report.this).execute("/donations");

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Donation 5.0 – Report : DonationAdapter

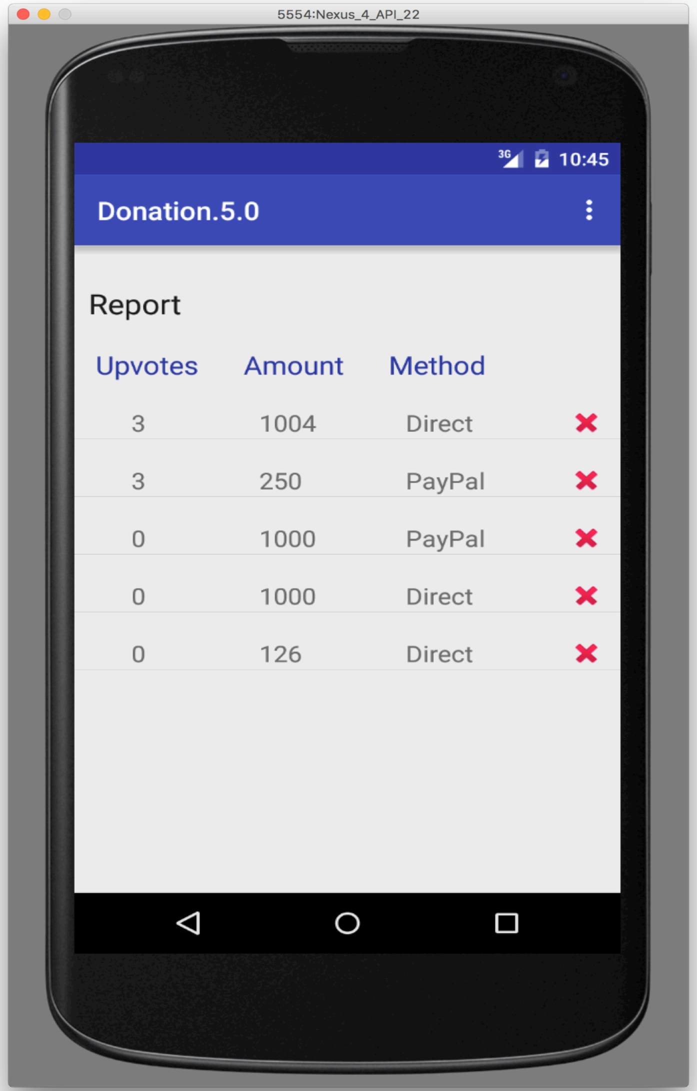


Report

Upvotes	Amount	Method
3	1004	Direct
3	250	PayPal
0	1000	PayPal
0	1000	Direct
0	126	Direct

```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);

        ImageView imgDelete = (ImageView) view.findViewById(R.id.imgDelete);
        imgDelete.setTag(donation);
        imgDelete.setOnClickListener(Report.this);

        TextView amountView = (TextView) view.findViewById(R.id.row_amount);
        TextView methodView = (TextView) view.findViewById(R.id.row_method);
        TextView upvotesView = (TextView) view.findViewById(R.id.row_upvotes);

        amountView.setText("" + donation.amount);
        methodView.setText(donation.paymenttype);
        upvotesView.setText("" + donation.upvotes);

        view.setTag(donation._id); // setting the 'row' id to the id of the donation

        return view;
    }

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

Donation 5.0 – Report : DonationAdapter



Report

Upvotes	Amount	Method	
3	1004	Direct	✖
3	250	PayPal	✖
0	1000	PayPal	✖
0	1000	Direct	✖
0	126	Direct	✖

```
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);

        ImageView imgDelete = (ImageView) view.findViewById(R.id.imgDelete);
        imgDelete.setTag(donation);
        imgDelete.setOnClickListener(Report.this);

        TextView amountView = (TextView) view.findViewById(R.id.row_amount);
        TextView methodView = (TextView) view.findViewById(R.id.row_method);
        TextView upvotesView = (TextView) view.findViewById(R.id.row_upvotes);

        amountView.setText("" + donation.amount);
        methodView.setText(donation.paymenttype);
        upvotesView.setText("" + donation.upvotes);

        view.setTag(donation._id); // setting the 'row' id to the id of the donation

        return view;
    }

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

A bit about JSON & Googles Gson

Question?

- Given a particular set of data, how do you store it permanently?
  - What do you store on disk?
  - What format?
  - Can you easily transmit over the web?
  - Will it be readable by other languages?
  - Can humans read the data?
- Examples:
  - A Square
  - A Dictionary
  - A Donation…

Storage Using Plain Text

- Advantages
  - Human readable (good for debugging / manual editing)
  - Portable to different platforms
  - Easy to transmit using web

- Disadvantages
  - Takes more memory than necessary

- Alternative? - use a standardized system -- JSON
  - Makes the information more portable

JavaScript Object Notation – What is it?

- ❑ Language Independent.
- ❑ Text-based.
- ❑ Light-weight.
- ❑ Easy to parse.

JavaScript Object Notation – What is it?

- JSON is lightweight text-data interchange format
- JSON is language independent*
  - *JSON uses JavaScript syntax for describing data objects
    - JSON parsers and JSON libraries exists for many different programming languages.
- JSON is "self-describing" and easy to understand
- JSON - Evaluates to JavaScript Objects
  - The JSON text format is syntactically identical to the code for creating JavaScript objects.
  - Because of this similarity, instead of using a parser, a JavaScript program can use the built-in eval()*** function and execute JSON data to produce native JavaScript objects.

When to use JSON?

- SOAP is a protocol specification for exchanging structured information in the implementation of Web Services.
- SOAP internally uses XML to send data back and forth.

- REST is a design concept.
- You are not limited to picking XML to represent data, you could pick anything really (JSON included).

JSON example

```
{
        "firstName": "John",
        "lastName": "Smith",
        "age": 25,
        "address": {
                "streetAddress": "21 2nd Street",
                "city": "New York",
                "state": "NY",
                "postalCode": 10021
        },
        "phoneNumbers": [
                {
                        "type": "home",
                        "number": "212 555-1234"
                },
                {
                        "type": "fax",
                        "number": "646 555-4567"
                }
        ]
}
```

JSON to XML

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
        <person>
                <firstName>John</firstName>
                <lastName>Smith</lastName>
                <age>25</age>
                <address>
                        <streetAddress>21 2nd Street</streetAddress>
                        <city>New York</city>
                        <state>NY</state>
                        <postalCode>10021</postalCode>
                </address>
                <phoneNumbers>
                        <phoneNumber>
                                <number>212 555-1234</number>
                                <type>home</type>
                        </phoneNumber>
                        <phoneNumber>
                                <number>646 555-4567</number>
                                <type>fax</type>
                        </phoneNumber>
                </phoneNumbers>
        </person>
</persons>
```

JSON vs XML size

- XML: 549 characters, 549 bytes
- JSON: 326 characters, 326 bytes
- **XML ~68,4 % larger than JSON!**

- But a large data set is going to be large regardless of the data format you use.
- Most servers gzip or otherwise compress content before sending it out, the difference between gzipped JSON and gzipped XML isn’t nearly as drastic as the difference between standard JSON and XML.

JSON vs XML

Favor XML over JSON when any of these is true:

- ❑ You need message validation
- ❑ You're using XSLT
- ❑ Your messages include a lot of marked-up text
- ❑ You need to interoperate with environments that don't support JSON

Favor JSON over XML when all of these are true:

- ❑ Messages don't need to be validated, or validating their deserialization is simple
- ❑ You're not transforming messages, or transforming their deserialization is simple
- ❑ Your messages are mostly data, not marked-up text
- ❑ The messaging endpoints have good JSON tools

Security problems***

- The eval() function can compile and execute any JavaScript. This represents a potential security problem.
- It is safer to use a JSON parser (like Gson) to convert a JSON text to a JavaScript object. A JSON parser will recognize only JSON text and will not compile scripts.
- In browsers that provide native JSON support, JSON parsers are also faster.
- Native JSON support is included in newer browsers and in the newest ECMAScript (JavaScript) standard.

JSON Schema

- Describes your JSON data format
- http://jsonschemalint.com/
- http://json-schema.org/implementations
- http://en.wikipedia.org/wiki/JSON#Schema_and_Metadata

JSON Values

JSON values can be:

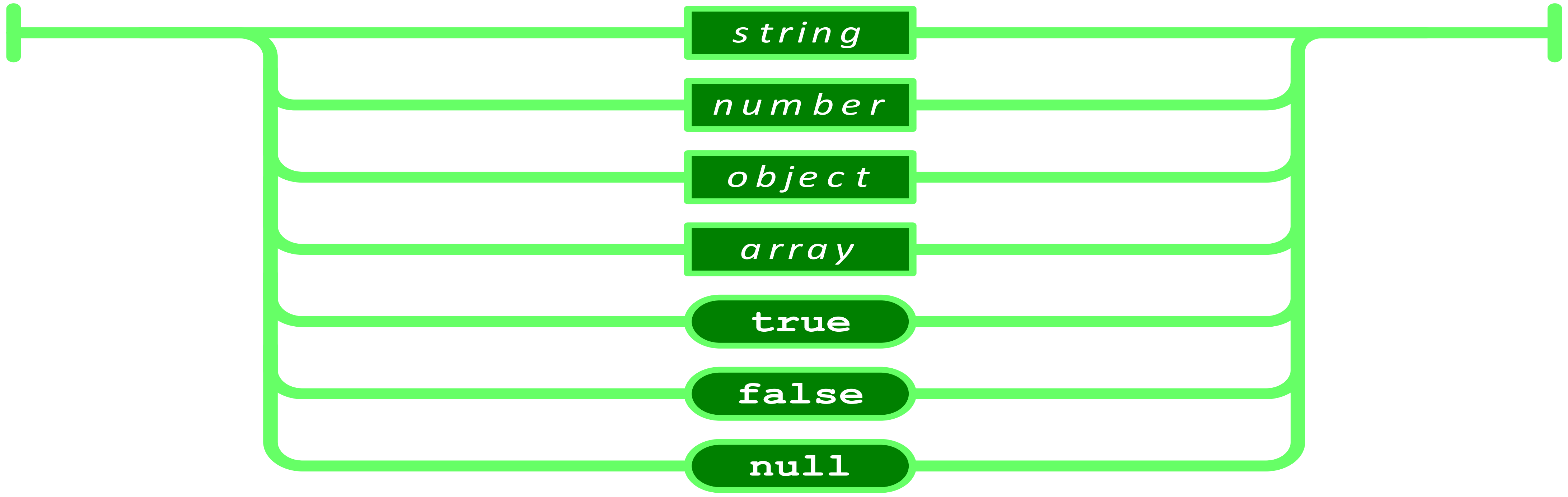
- ❑ A number (integer or floating point)
- ❑ A string (in double quotes)
- ❑ A boolean (true or false)
- ❑ An *object* (in curly brackets)
- ❑ An *array* (in square brackets)
- ❑ null

JSON Values

- Object
  - Unordered set of name-value pairs
  - names must be strings
  - { name1 : value1, name2 : value2, …, nameN : valueN }

- Array
  - Ordered list of values
  - [value1, value2, … valueN]

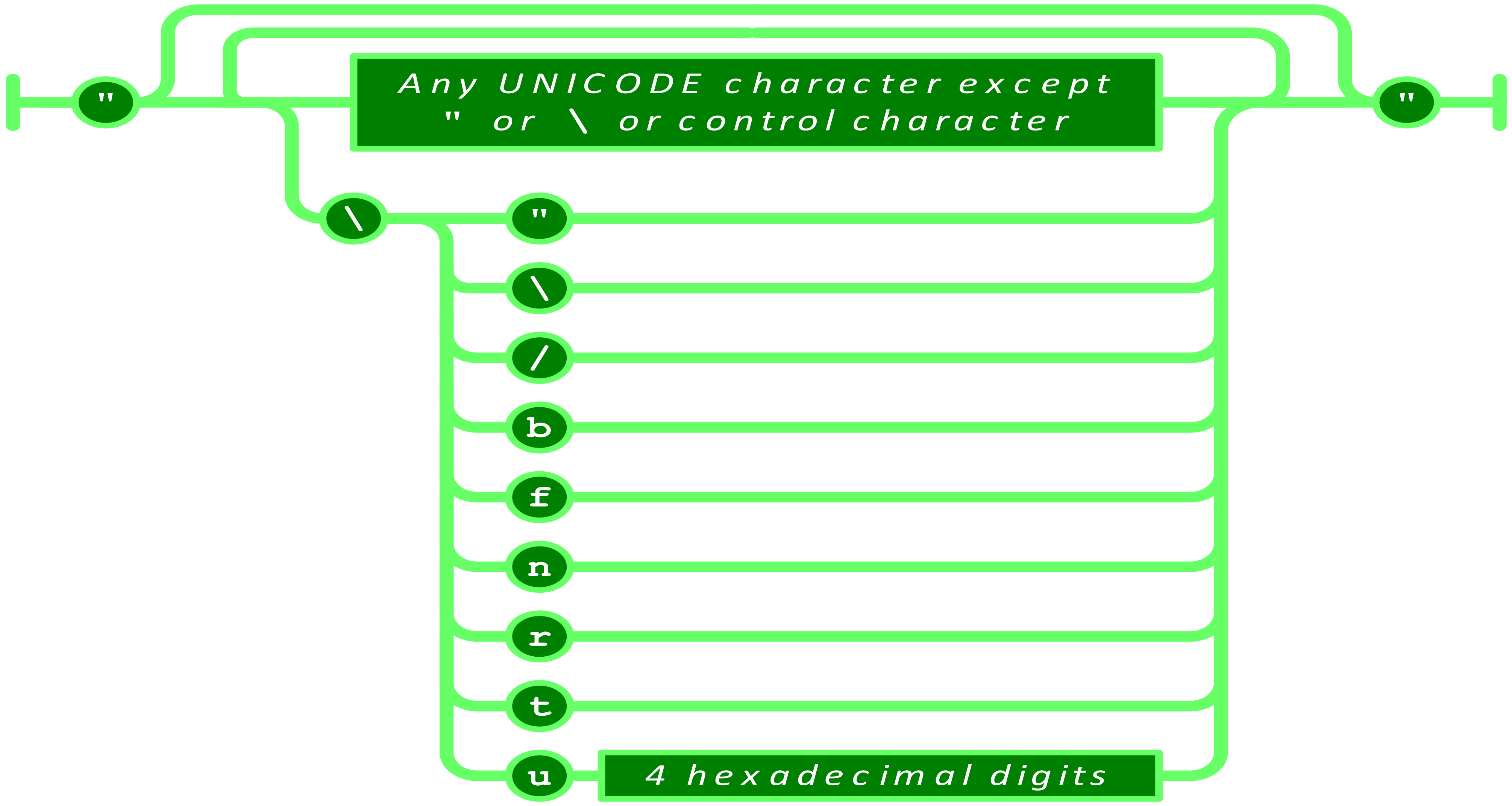
Value



Strings

- Sequence of 0 or more Unicode characters
- No separate character type
  - A character is represented as a string with a length of 1
- Wrapped in "double quotes"
- Backslash escapement

String

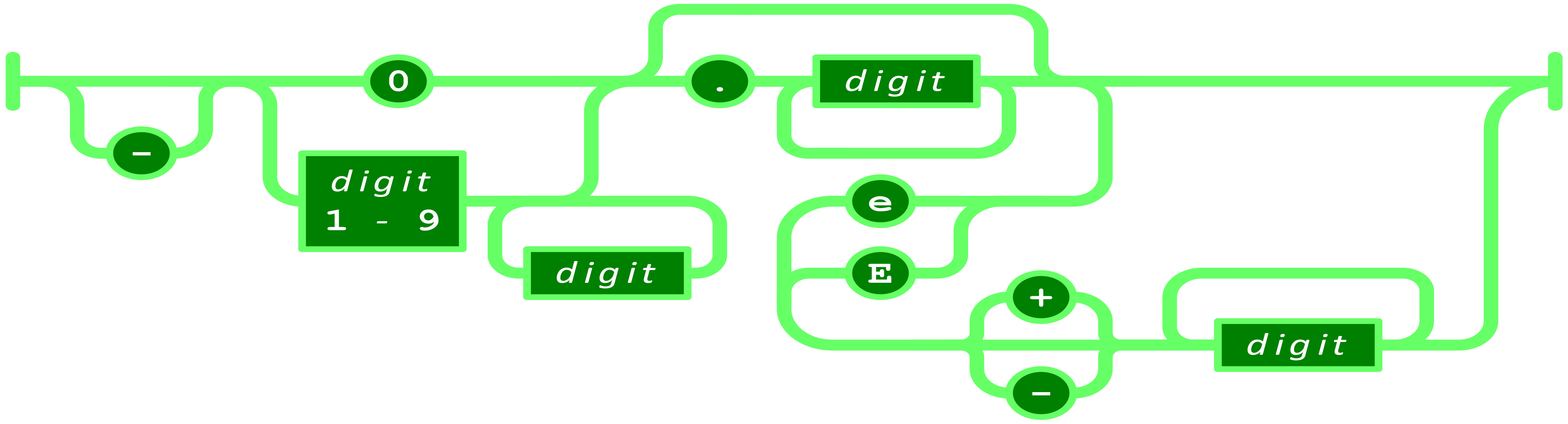

"
Any UNICODE character except " or \ or control character
\
"
\
/
b
f
n
r
t
u
4 hexadecimal digits
"

Numbers

- Integer
- Real
- Scientific

- No octal or hex
- No **`NaN`** or **`Infinity`**
  - Use **`null`** instead

Number



Booleans

- `true`
- `false`

`null`

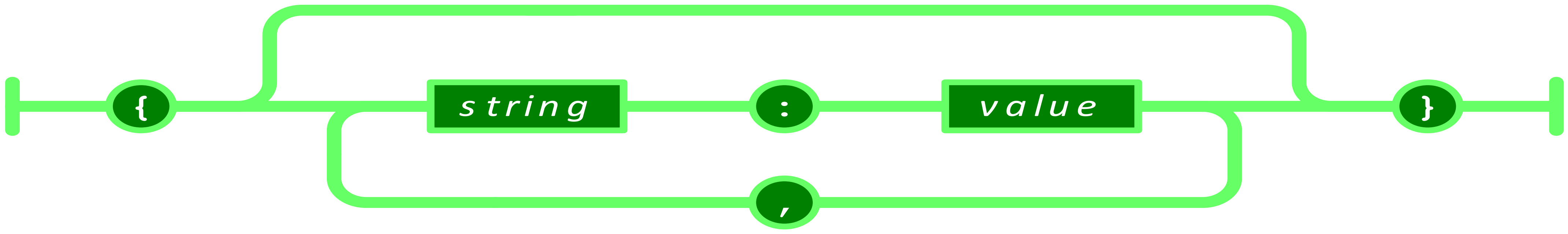
- A value that isn't anything

Object

- Objects are unordered containers of key/value pairs
- Objects are wrapped in `{ }`
- `,` separates key/value pairs
- `:` separates keys and values
- Keys are strings
- Values are JSON values

  - struct, record, hashtable, object

Object

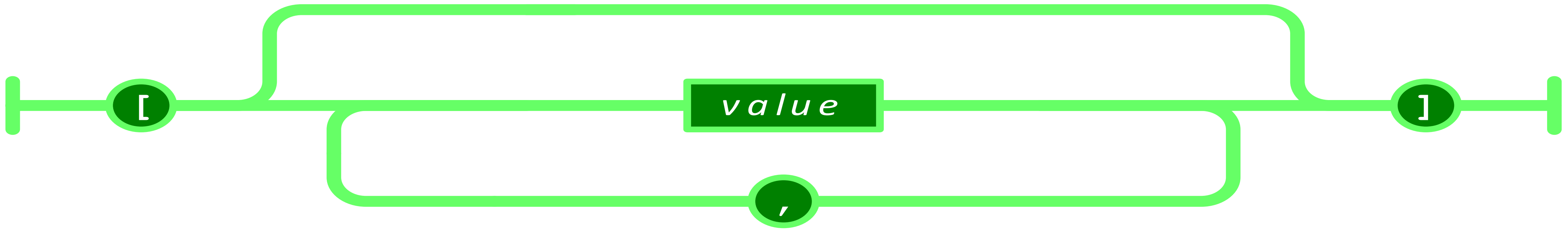


```
{
"_id":"560515770f76130300c69953",
"usertoken":"11343761234567808125",
"paymenttype":"PayPal",
"__v":0,
"upvotes":0,
"amount":1999
}
```

Array

- ❑ Arrays are ordered sequences of values
- ❑ Arrays are wrapped in `[]`
- ❑ `,` separates values
- ❑ JSON does not talk about indexing.
  - An implementation can start array indexing at 0 or 1.

Array



```
[
{"_id":"560515770f76130300c69953","usertoken":"11343761234567808125","
paymenttype":"PayPal","__v":0,"upvotes":0,"amount":1999},

{"_id":"561252404218920300484O3d","usertoken":"11343761234567808125","
paymenttype":"PayPal","__v":0,"upvotes":5,"amount":1234},

{"_id":"5627620ac9e9e303005b113c","usertoken":"11343761234567808125","
paymenttype":"Direct","__v":0,"upvotes":2,"amount":1001}
]
```

MIME Media Type & Character Encoding

- **`application/json`**
- Strictly UNICODE.
- Default: UTF-8.
- UTF-16 and UTF-32 are allowed.

Versionless

- ❑ JSON has no version number.
- ❑ No revisions to the JSON grammar are anticipated.
- ❑ JSON is very stable.

Rules

- A JSON decoder must accept all well-formed JSON text.
- A JSON decoder may also accept non-JSON text.
- A JSON encoder must only produce well-formed JSON text.
- *Be conservative in what you do, be liberal in what you accept from others.*

Donation 5.0, Googles Gson & REST

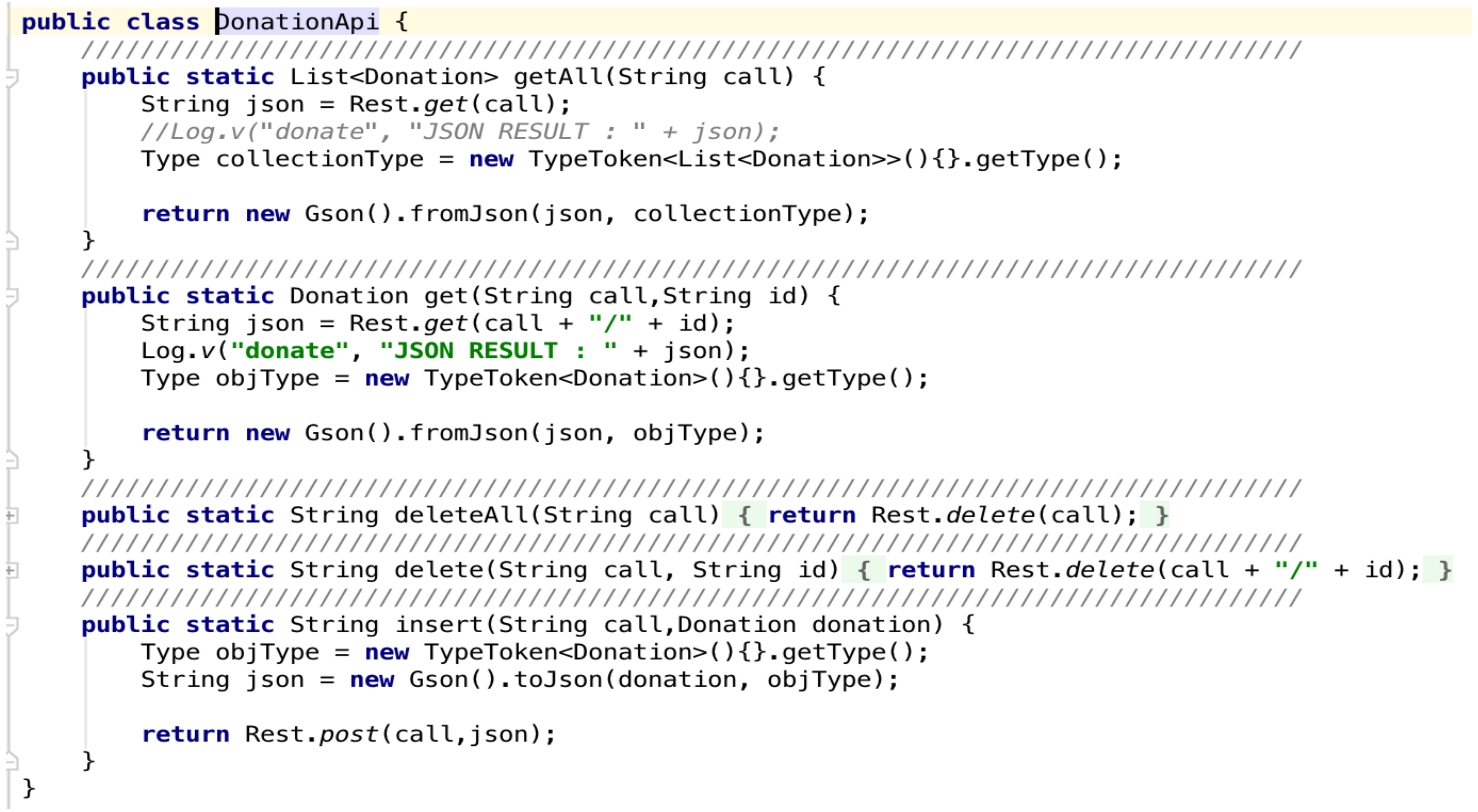
Google’s Gson

https://sites.google.com/site/gson/gson-user-guide

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson is an open-source project hosted at http://code.google.com/p/google-gson.

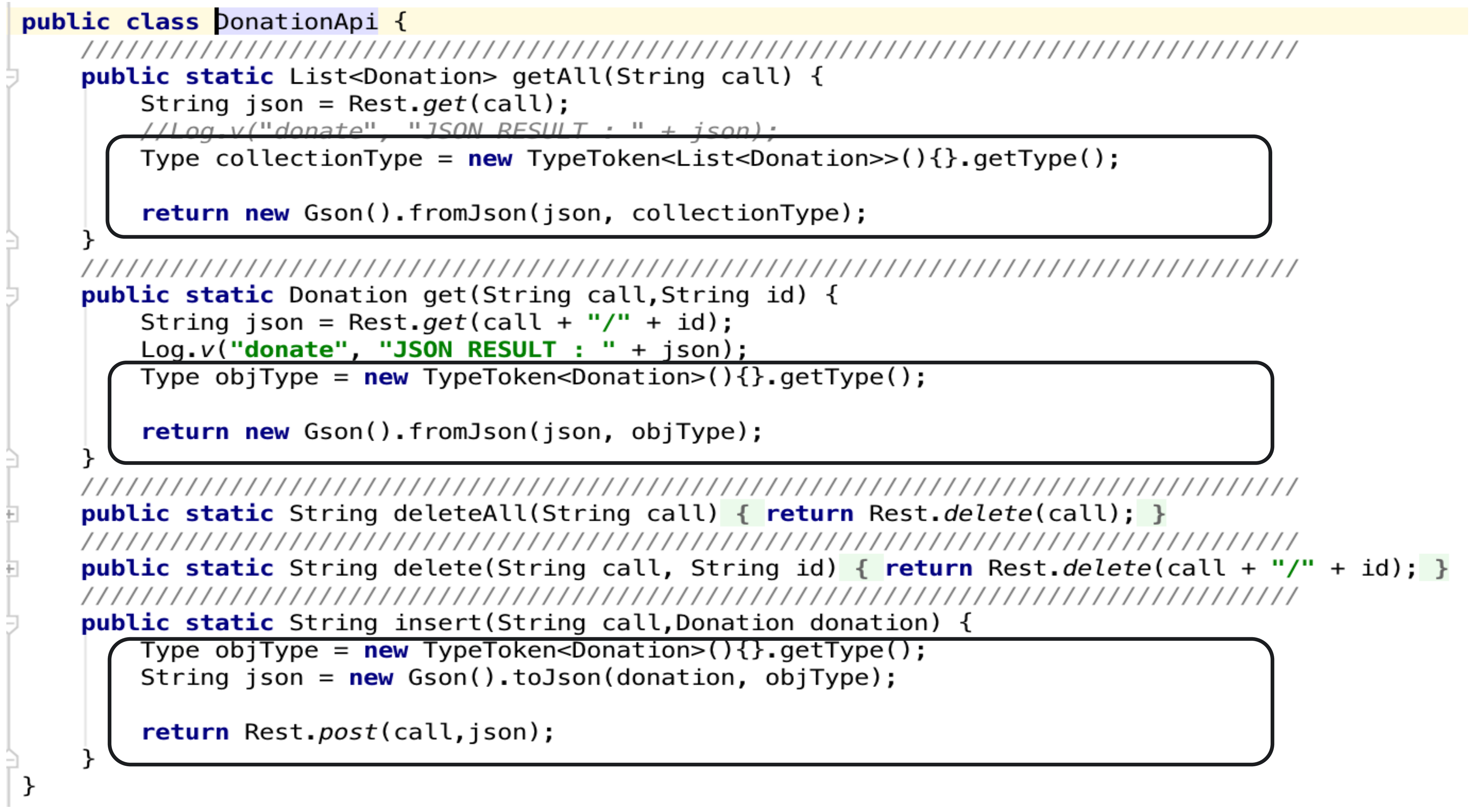
Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

Donation 5.0 & Google's Gson



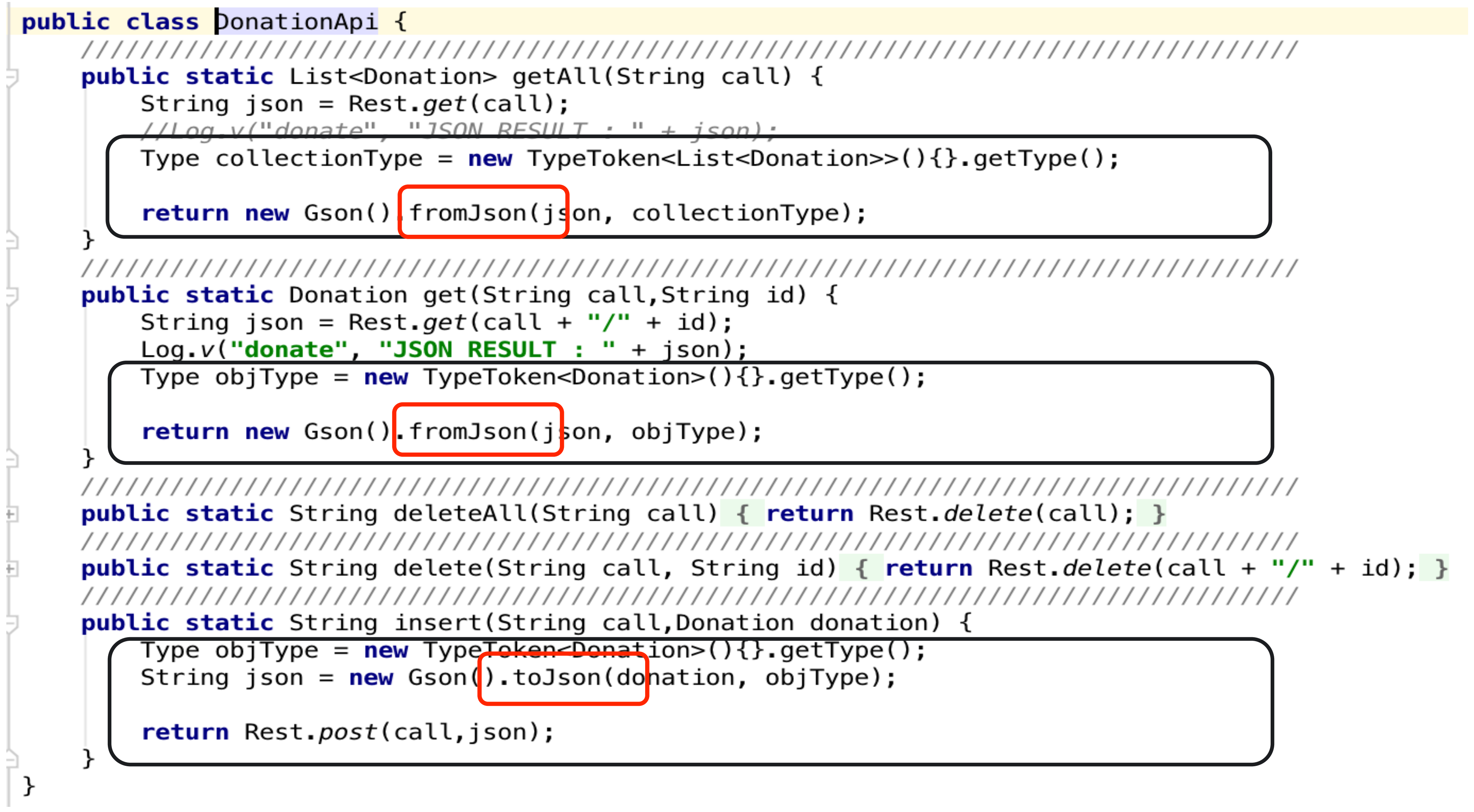
```java
public class DonationApi {
    ///////////////////////////////////////////////////////////////////////////////////
    public static List<Donation> getAll(String call) {
        String json = Rest.get(call);
        //Log.v("donate", "JSON RESULT : " + json);
        Type collectionType = new TypeToken<List<Donation>>(){}.getType();

        return new Gson().fromJson(json, collectionType);
    }
    ///////////////////////////////////////////////////////////////////////////////////
    public static Donation get(String call,String id) {
        String json = Rest.get(call + "/" + id);
        Log.v("donate", "JSON RESULT : " + json);
        Type objType = new TypeToken<Donation>(){}.getType();

        return new Gson().fromJson(json, objType);
    }
    ///////////////////////////////////////////////////////////////////////////////////
    public static String deleteAll(String call) { return Rest.delete(call); }
    ///////////////////////////////////////////////////////////////////////////////////
    public static String delete(String call, String id) { return Rest.delete(call + "/" + id); }
    ///////////////////////////////////////////////////////////////////////////////////
    public static String insert(String call,Donation donation) {
        Type objType = new TypeToken<Donation>(){}.getType();
        String json = new Gson().toJson(donation, objType);

        return Rest.post(call,json);
    }
}
```

Donation 5.0 & Google's Gson



```java
public class DonationApi {
    ////////////////////////////////////////////////////////////////////////////////
    public static List<Donation> getAll(String call) {
        String json = Rest.get(call);
        //Log.v("donate", "JSON RESULT : " + json);
        Type collectionType = new TypeToken<List<Donation>>(){}.getType();

        return new Gson().fromJson(json, collectionType);
    }
    ////////////////////////////////////////////////////////////////////////////////
    public static Donation get(String call,String id) {
        String json = Rest.get(call + "/" + id);
        Log.v("donate", "JSON RESULT : " + json);
        Type objType = new TypeToken<Donation>(){}.getType();

        return new Gson().fromJson(json, objType);
    }
    ////////////////////////////////////////////////////////////////////////////////
    public static String deleteAll(String call) { return Rest.delete(call); }
    ////////////////////////////////////////////////////////////////////////////////
    public static String delete(String call, String id) { return Rest.delete(call + "/" + id); }
    ////////////////////////////////////////////////////////////////////////////////
    public static String insert(String call,Donation donation) {
        Type objType = new TypeToken<Donation>(){}.getType();
        String json = new Gson().toJson(donation, objType);

        return Rest.post(call,json);
    }
}
```

Donation 5.0 & Google's Gson



```java
public class DonationApi {
    //////////////////////////////////////////////////////////////////////////////////
    public static List<Donation> getAll(String call) {
        String json = Rest.get(call);
        //Log.v("donate", "JSON RESULT : " + json);
        Type collectionType = new TypeToken<List<Donation>>(){}.getType();

        return new Gson().fromJson(json, collectionType);
    }
    //////////////////////////////////////////////////////////////////////////////////
    public static Donation get(String call,String id) {
        String json = Rest.get(call + "/" + id);
        Log.v("donate", "JSON RESULT : " + json);
        Type objType = new TypeToken<Donation>(){}.getType();

        return new Gson().fromJson(json, objType);
    }
    //////////////////////////////////////////////////////////////////////////////////
    public static String deleteAll(String call) { return Rest.delete(call); }
    //////////////////////////////////////////////////////////////////////////////////
    public static String delete(String call, String id) { return Rest.delete(call + "/" + id); }
    //////////////////////////////////////////////////////////////////////////////////
    public static String insert(String call,Donation donation) {
        Type objType = new TypeToken<Donation>(){}.getType();
        String json = new Gson().toJson(donation, objType);

        return Rest.post(call,json);
    }
}
```

Summary

- JSON is a standard way to exchange data
  - Easily parsed by machines
  - Human readable form

- JSON uses dictionaries and lists
  - Dictionaries are unordered
  - Lists are ordered

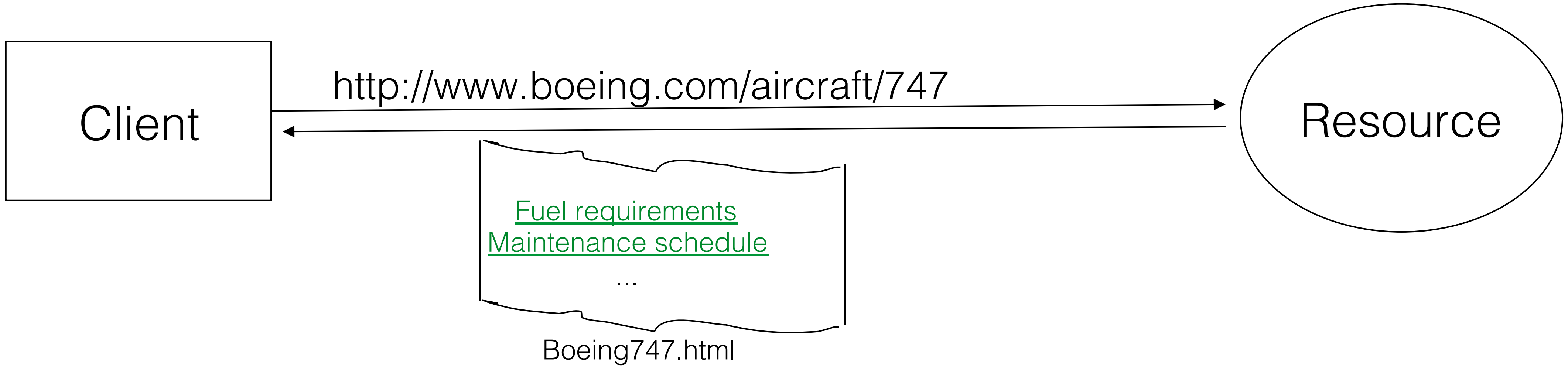
- GSON is Googles JSON parser
  - Very simple to use

What is REST?

"*REST* " was coined by Roy Fielding in his Ph.D. dissertation [1] to describe a *design pattern* for implementing networked systems.

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Why is it called "**Re**presentational **S**tate **T**ransfer"?



- The Client references a Web resource using a URL.
- A **representation** of the resource is returned (in this case as an HTML document).
- The representation (*e.g.*, Boeing747.html) places the client in a new **state**.
- When the client selects a hyperlink in Boeing747.html, it accesses another resource.
- The new representation places the client application into yet another state.
- Thus, the client application **transfers** state with each resource representation.

REST Characteristics

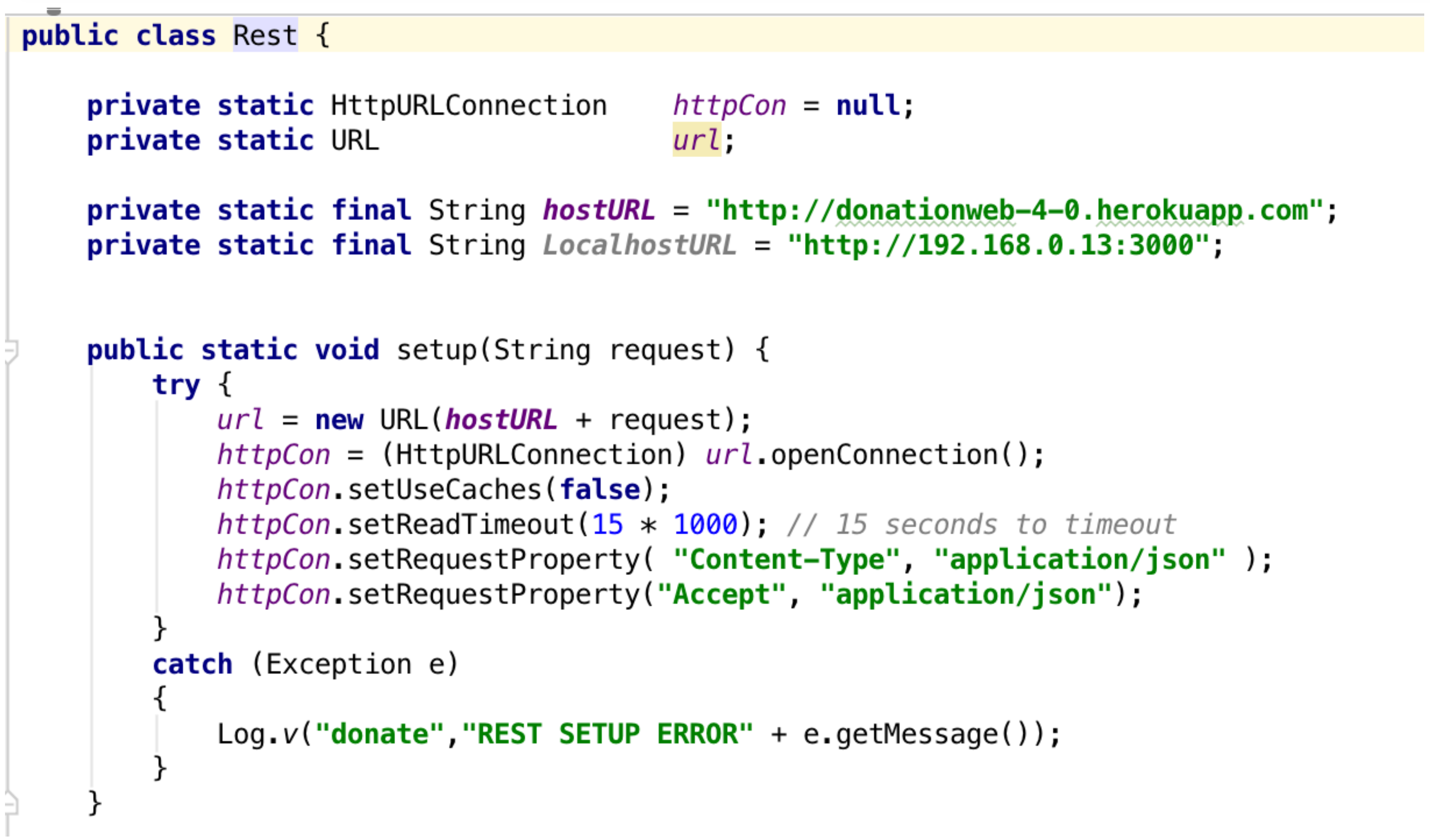
- REST is not a standard (unlike SOAP)
  - You will not see the W3C putting out a REST specification.
  - You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.
- REST is just a **design pattern**
  - You can't bottle up a pattern.
  - You can only understand it and design your Web services to it.
- REST does prescribe the **use** of standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/*etc.* **(Resource Representations)**
  - text/xml, text/html, image/gif, image/jpeg, *etc.* (Resource Types, MIME Types)

REST Principles

- ❑ Everything is a resource
- ❑ Every resource is identified by a unique identifier
- ❑ Use simple and uniform interfaces
- ❑ Communication is done by representation
- ❑ Be Stateless

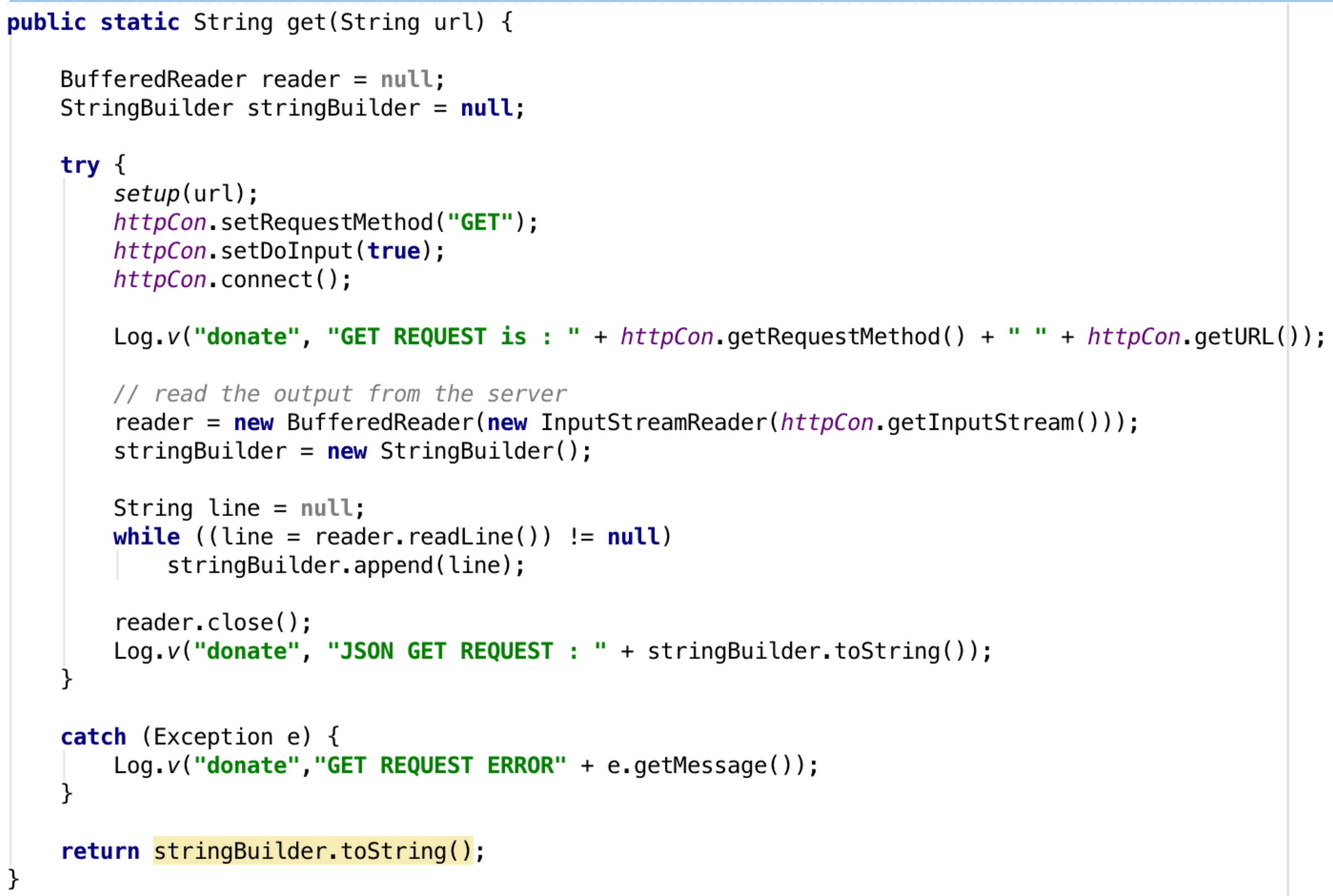
- ❑ *We'll look at these, and more, next year ☺.*

Donation 5.0 & Rest.java



```
public class Rest {

    private static HttpURLConnection    httpCon = null;
    private static URL                  url;

    private static final String hostURL = "http://donationweb-4-0.herokuapp.com";
    private static final String LocalhostURL = "http://192.168.0.13:3000";


    public static void setup(String request) {
        try {
            url = new URL(hostURL + request);
            httpCon = (HttpURLConnection) url.openConnection();
            httpCon.setUseCaches(false);
            httpCon.setReadTimeout(15 * 1000); // 15 seconds to timeout
            httpCon.setRequestProperty( "Content-Type", "application/json" );
            httpCon.setRequestProperty("Accept", "application/json");
        }
        catch (Exception e)
        {
            Log.v("donate","REST SETUP ERROR" + e.getMessage());
        }
    }
```

Donation 5.0 & Rest.java – GET Request

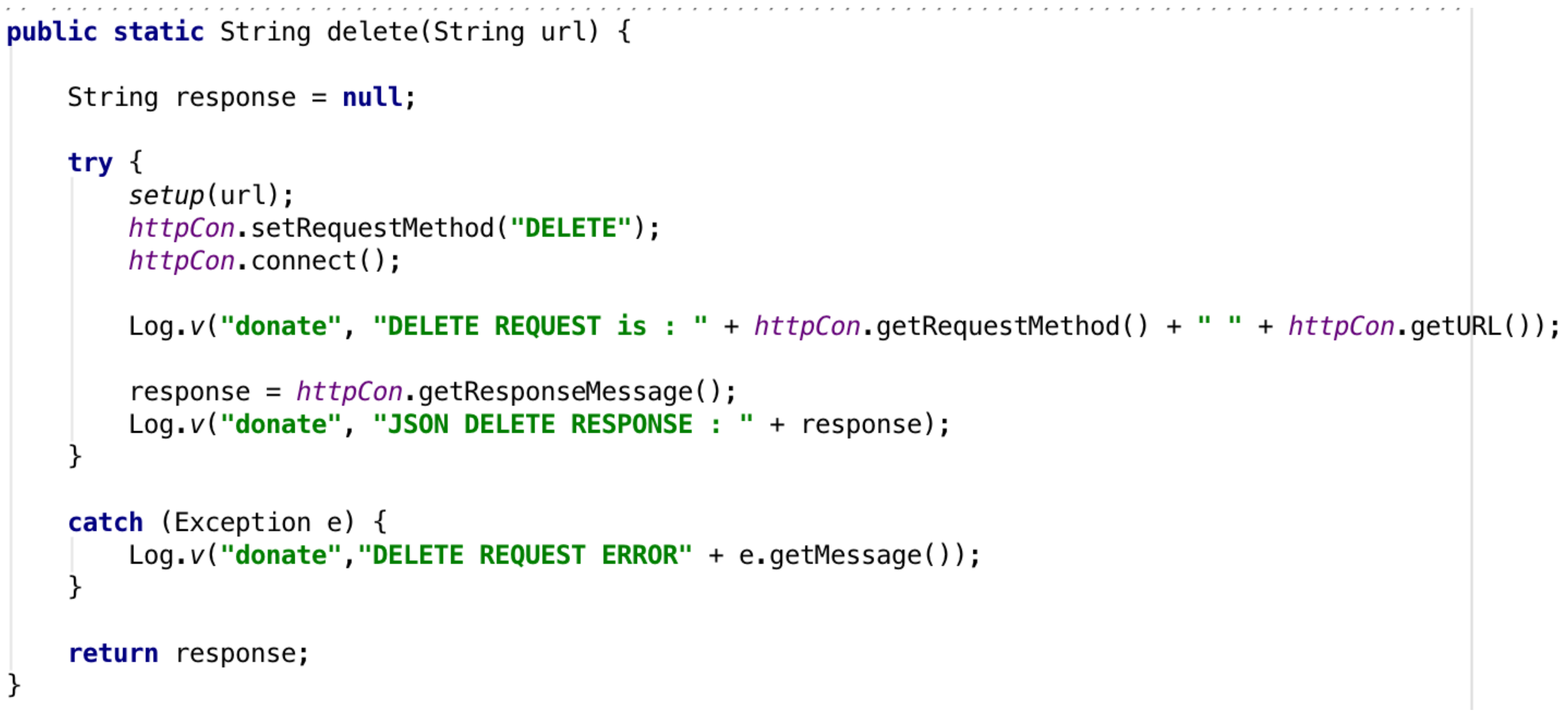


```java
public static String get(String url) {

    BufferedReader reader = null;
    StringBuilder stringBuilder = null;

    try {
        setup(url);
        httpCon.setRequestMethod("GET");
        httpCon.setDoInput(true);
        httpCon.connect();

        Log.v("donate", "GET REQUEST is : " + httpCon.getRequestMethod() + " " + httpCon.getURL());

        // read the output from the server
        reader = new BufferedReader(new InputStreamReader(httpCon.getInputStream()));
        stringBuilder = new StringBuilder();

        String line = null;
        while ((line = reader.readLine()) != null)
            stringBuilder.append(line);

        reader.close();
        Log.v("donate", "JSON GET REQUEST : " + stringBuilder.toString());
    }

    catch (Exception e) {
        Log.v("donate","GET REQUEST ERROR" + e.getMessage());
    }

    return stringBuilder.toString();
}
```

Donation 5.0 & Rest.java – POST Request

```java
public static String post(String url, String json) {

    OutputStreamWriter writer = null;
    StringBuilder stringBuilder = null;

    try {
        setup(url);
        httpCon.setRequestMethod("POST");
        httpCon.setDoOutput(true);
        httpCon.setDoInput(true);
        httpCon.connect();

        Log.v("donate", "POST REQUEST is : " + httpCon.getRequestMethod() + " " + httpCon.getURL());

        // read the output from the server
        writer = new OutputStreamWriter(httpCon.getOutputStream());
        writer.write(json);
        writer.close();

        stringBuilder = new StringBuilder();
        int HttpResult = httpCon.getResponseCode();
        if(HttpResult == HttpURLConnection.HTTP_OK) {
            BufferedReader br = new BufferedReader(new InputStreamReader(httpCon.getInputStream(), "utf-8"));
            String line = null;
            while ((line = br.readLine()) != null)
                stringBuilder.append(line + "\n");

            Log.v("donate", "JSON POST RESPONSE : " + stringBuilder.toString());
        }
    }

    catch (Exception e) {
        Log.v("donate","POST REQUEST ERROR" + e.getMessage());
    }

    return stringBuilder.toString();
}
```

Donation 5.0 & Rest.java – DELETE Request



```java
public static String delete(String url) {

    String response = null;

    try {
        setup(url);
        httpCon.setRequestMethod("DELETE");
        httpCon.connect();

        Log.v("donate", "DELETE REQUEST is : " + httpCon.getRequestMethod() + " " + httpCon.getURL());

        response = httpCon.getResponseMessage();
        Log.v("donate", "JSON DELETE RESPONSE : " + response);
    }

    catch (Exception e) {
        Log.v("donate","DELETE REQUEST ERROR" + e.getMessage());
    }

    return response;
}
```

Android Networking (Using Volley)

Volley is an HTTP library developed by Google that makes networking for Android apps easier and most importantly, faster. Volley is available through the open AOSP repository.

Introduced during Google I/O 2013, it was developed because of the absence, in the Android SDK, of a networking class capable of working without interfering with the user experience.

Volley

- Volley offers the following benefits:
  - Automatic scheduling of network requests.
  - Multiple concurrent network connections.
  - Transparent disk and memory response caching with standard HTTP cache coherence.
  - Support for request prioritization.
  - Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
  - Ease of customization, for example, for retry and backoff.
  - Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
  - Debugging and tracing tools.

Why Volley?

- Avoid `HttpUrlConnection` and `HttpClient`
  - On lower API levels (mostly on Gingerbread and Froyo), HttpUrlConnection and HttpClient are far from being perfect. There are some known issues and bugs that were never fixed.
  - Moreover, HttpClient was deprecated in the last API update (API 22), which means that it will no longer be maintained and may be removed in a future release.
  - These are sufficient reasons for deciding to switch to a more reliable way of handling your network requests.

Why Volley?

- Avoid **`AsyncTask`**
  - Since the introduction of Honeycomb (API 11), it's been mandatory to perform network operations on a separate thread, different from the main thread. This substantial change led the way to massive use of the **`AsyncTask<Params, Progress, Result>`** specification.
  - The class is pretty straightforward, way easier than the implementation of a **`service`**, and comes with a ton of examples and documentation.
  - The main problem (next slide), however, is the serialization of the calls. Using the **`AsyncTask`** class, you can't decide which request goes first and which one has to wait. Everything happens FIFO, first in, first out.

Problem Solved…

- ❑ The problems arise, for example, when you have to load a list of items that have attached a thumbnail. When the user scrolls down and expects new results, you can't tell your activity to first load the JSON of the next page and only then the images of the previous one. This can become a serious user experience problem in applications such as Facebook or Twitter, where the list of new items is more important than the thumbnail associated with it.
- ❑ Volley aims to solve this problem by including a powerful cancellation API. You no longer need to check in **`onPostExecute`** whether the activity was destroyed while performing the call. This helps avoiding an unwanted **`NullPointerException`**.

Why Volley?

- It's Much Faster
  - Some time ago, the Google+ team did a series of performance tests on each of the different methods you can use to make network requests on Android. Volley got a score up to ten times better than the other alternatives when used in RESTful applications.
- Small Metadata Operations
  - Volley is perfect for small calls, such as JSON objects, portions of lists, details of a selected item, and so on. It has been devised for RESTful applications and in this particular case it gives its very best.

Why Volley?

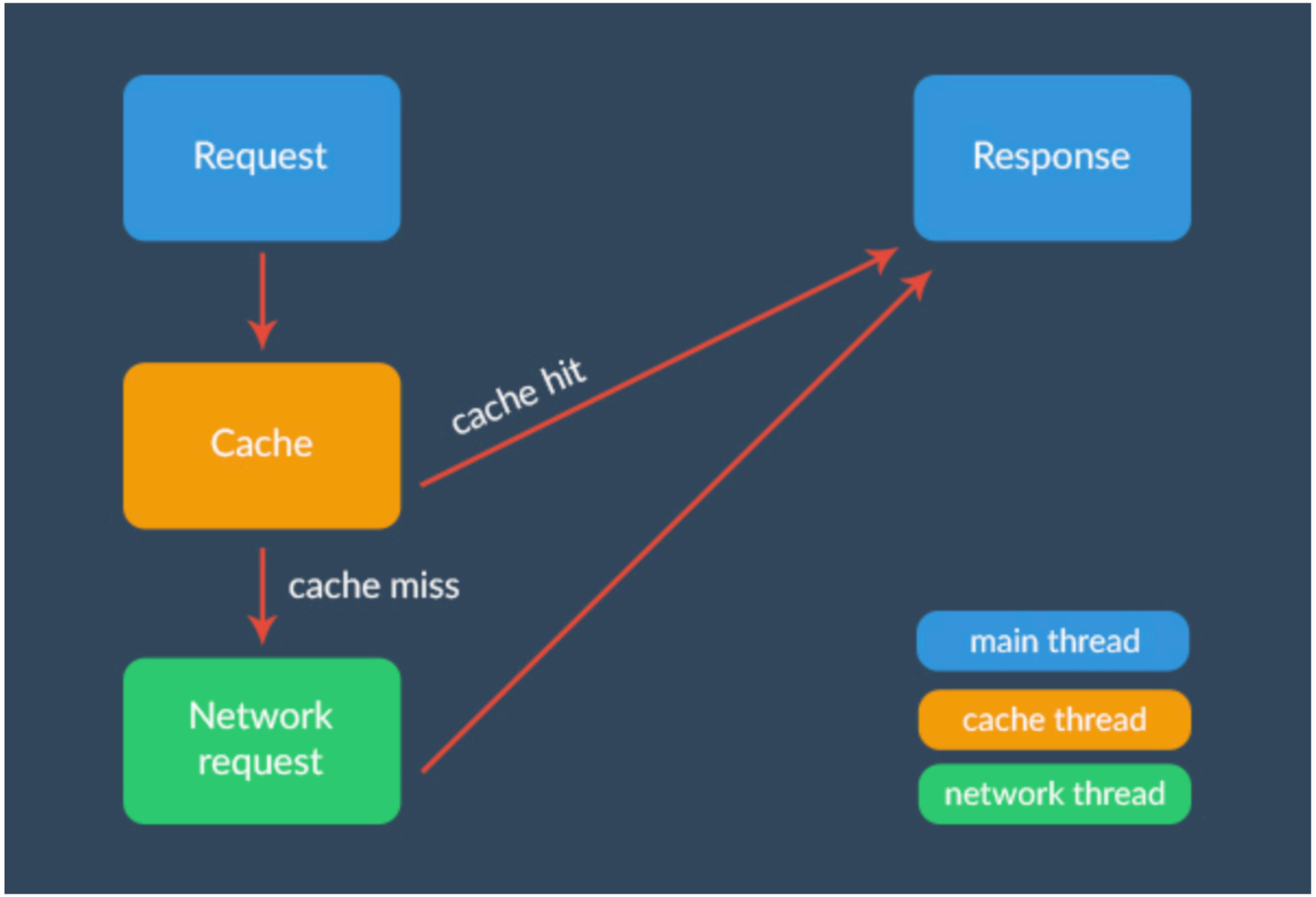
- It Caches Everything
  - Volley automatically caches requests and this is something truly life-saving. Let’s return for a moment to the example given earlier. You have a list of items—a JSON array let’s say—and each item has a description and a thumbnail associated with it. Now think about what happens if the user rotates the screen: the activity is destroyed, the list is downloaded again, and so are the images. Long story short, a significant waste of resources and a poor user experience.
  - Volley proves to be extremely useful for overcoming this issue. It *remembers* the previous calls it did and handles the activity destruction and reconstruction. It caches everything without you having to worry about it.

Why Not Volley?

- It is not so good, however, when employed for streaming operations and large downloads. Contrary to common belief, Volley's name doesn't come from the sport dictionary. It’s rather intended as repeated bursts of calls, grouped together. It's somehow intuitive why this library doesn't come in handy when, instead of a volley of arrows, you want to fire a cannon ball.

Under the Hood

- Volley works on three different levels with each level operating on its own thread.



Under the Hood

- Main Thread
  - On the main thread, consistently with what you already do in the AsyncTask specification, you are only allowed to fire the request and handle its response. Nothing more, nothing less.
  - The main consequence is that you can actually ignore everything that was going on in the doInBackground method. Volley automatically manages the HTTP transactions and the catching network errors that you needed to care about before.

Under the Hood

- Cache and Network Threads
  - When you add a request to the queue, several things happens under the hood. First, Volley checks if the request can be serviced from cache. If it can, the cached response is read, parsed, and delivered. Otherwise it is passed to the network thread.
  - On the network thread, a round-robin with a series of threads is constantly working. The first available network thread dequeues the request, makes the HTTP request, parses the response, and writes it to cache. To finish, it dispatches the parsed response back to the main thread where your listeners are waiting to handle the result.

Getting Started With Volley

- Download the Volley Source
  - `git clone https://android.googlesource.com/platform/frameworks/volley`
- Import Source as **`Module`**
  - File -> New Module, choose Import Existing Project
  - Add dependency `compile project(':volley')`
- Alternative – *unofficial* mirror site so beware
  - `compile 'com.mcxiaoke.volley:library-aar:1.0.15'`

Using Volley

- Volley mostly works with just two classes, **`RequestQueue`** and **`Request`**. You first create a **`RequestQueue`**, which manages worker threads and delivers the parsed results back to the main thread. You then pass it one or more **`Request`** objects.

- The **`Request`** constructor always takes as parameters the method type (GET, POST, etc.), the URL of the resource, and event listeners. Then, depending on the type of request, it may ask for some more variables.

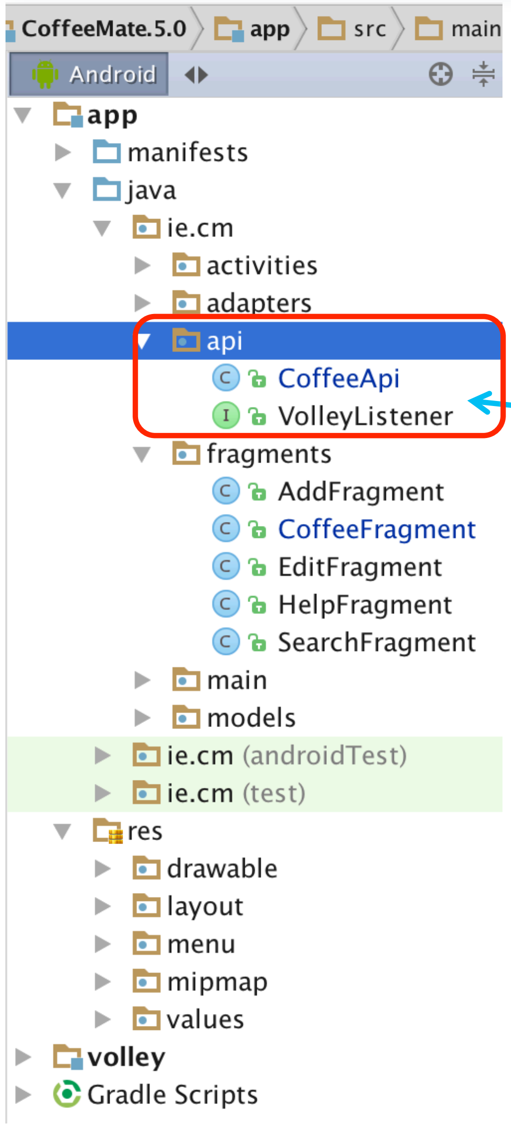
Using Volley *

- Here we create a **`RequestQueue`** object by invoking one of Volley's convenience methods, **`Volley.newRequestQueue`**. This sets up a **`RequestQueue`** object, using default values defined by Volley.
- As you can see, it's incredibly straightforward. You create the request and add it to the request queue. And you're done.
- If you have to fire multiple requests in several activities, you should avoid using this approach - better to instantiate one shared request queue and use it across your project (CoffeeMate 5.0)

```java
String url = "http://httpbin.org/html";

// Request a string response
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {

        // Result handling
        System.out.println(response.substring(0,100));

    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {

        // Error handling
        System.out.println("Something went wrong!");
        error.printStackTrace();

    }
});

// Add the request to the queue
Volley.newRequestQueue(this).add(stringRequest);
```

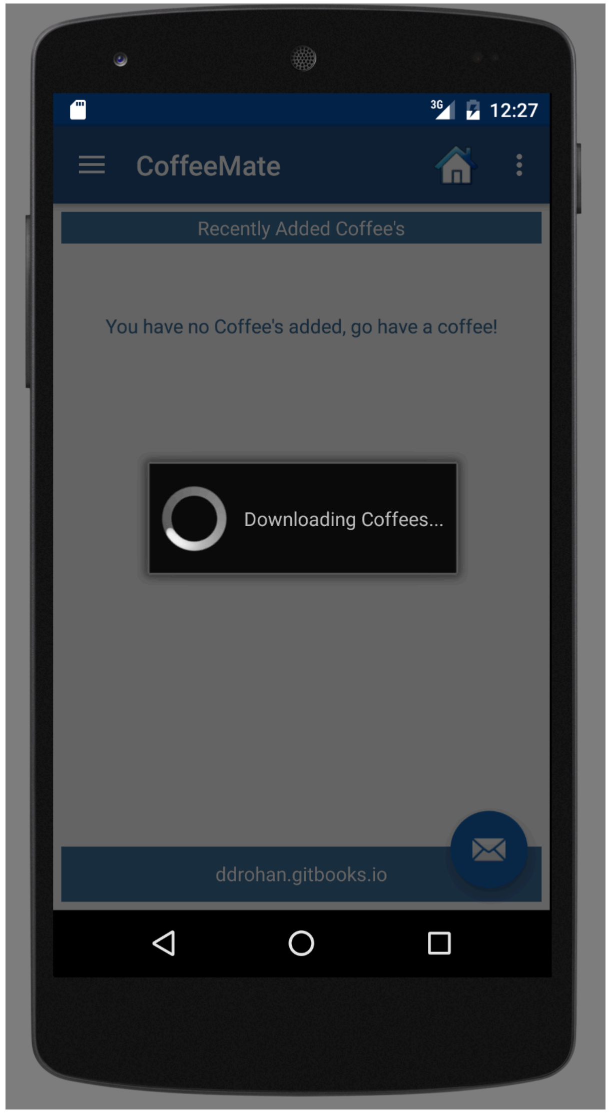
CoffeeMate Example (Using Volley)

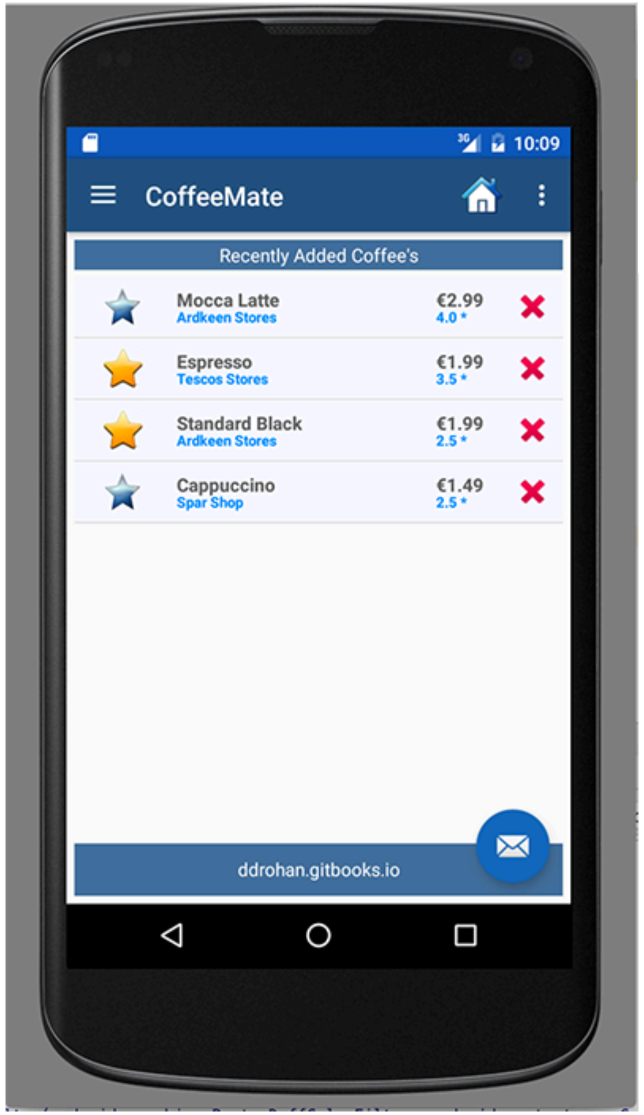
CoffeeMate 5.0 API & Callback Interface



- api class for calling REST service
- callback mechanism to update UI

CoffeeMate 5.0 & Volley


12:27
CoffeeMate
Recently Added Coffee's
You have no Coffee's added, go have a coffee!
Downloading Coffees...
ddrohan.gitbooks.io


10:09
CoffeeMate
Recently Added Coffee's

Mocca Latte	Ardkeen Stores	€2.99	4.0 *
Espresso	Tescos Stores	€1.99	3.5 *
Standard Black	Ardkeen Stores	€1.99	2.5 *
Cappuccino	Spar Shop	€1.49	2.5 *

ddrohan.gitbooks.io

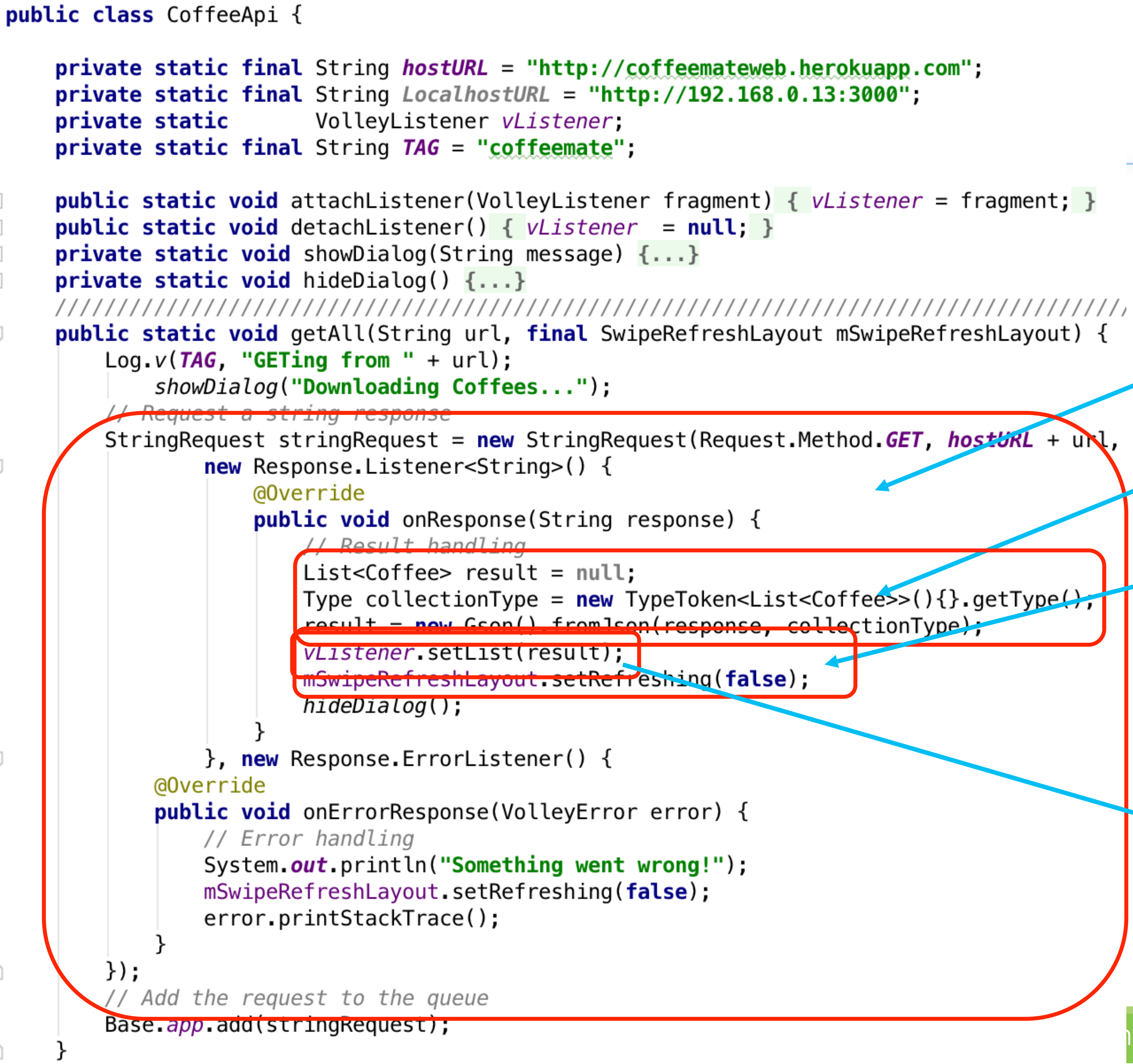
```java
@Override
public void onResume() {
  super.onResume();
  //updateUI(this);
  CoffeeApi.attachListener(this);
  CoffeeApi.getAll("/coffees/" + Base.googleToken, mSwipeRefreshLayout);
}
```

- Here we 'attach' our **`VolleyListener`** to the Fragment (**`CoffeeFragment`**) and then *getAll()* of the current users coffees.
- This method triggers a call to *setList()* via the callback interface, which in turn updates the UI ONLY when our API call completes.

```java
@Override
public void setList(List list) {
  Base.app.coffeeList = list;
  updateUI(this);
}
```

- We use a similar approach for Updating, Deleting etc.

CoffeeApi – refactored with Volley *



```java
public class CoffeeApi {

    private static final String hostURL = "http://coffeemateweb.herokuapp.com";
    private static final String LocalhostURL = "http://192.168.0.13:3000";
    private static       VolleyListener vListener;
    private static final String TAG = "coffeemate";

    public static void attachListener(VolleyListener fragment) { vListener = fragment; }
    public static void detachListener() { vListener  = null; }
    private static void showDialog(String message) {...}
    private static void hideDialog() {...}
    /////////////////////////////////////////////////////////////////////////////////////
    public static void getAll(String url, final SwipeRefreshLayout mSwipeRefreshLayout) {
        Log.v(TAG, "GETing from " + url);
            showDialog("Downloading Coffees...");
        // Request a string response
        StringRequest stringRequest = new StringRequest(Request.Method.GET, hostURL + url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        // Result handling
                        List<Coffee> result = null;
                        Type collectionType = new TypeToken<List<Coffee>>(){}.getType();
                        result = new Gson().fromJson(response, collectionType);
                        vListener.setList(result);
                        mSwipeRefreshLayout.setRefreshing(false);
                        hideDialog();
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Error handling
                System.out.println("Something went wrong!");
                mSwipeRefreshLayout.setRefreshing(false);
                error.printStackTrace();
            }
        });
        // Add the request to the queue
        Base.app.add(stringRequest);
    }
```

- Here we create a **`StringRequest`** **`GET`** request.
- On a successful **`RESPONSE`** we convert the result into a List of coffees and
- Trigger the callback to set the list in the fragment (and cancel the refresh spinner)

```java
@Override
public void setList(List list) {
    Base.app.coffeeList = list;
    updateUI(this);
}
```

CoffeeFragment (Extracts) *

```java
public class CoffeeFragment  extends Fragment implements AdapterView.OnItemClickListener,
                                                         View.OnClickListener,
                                                         VolleyListener
```

```java
@Override
public void onResume() {
  super.onResume();
  //updateUI(this);
  CoffeeApi.attachListener(this);
  CoffeeApi.getAll("/coffees/" + Base.googleToken, mSwipeRefreshLayout);
}

@Override
public void onPause() {
  super.onPause();
  CoffeeApi.detachListener();
}
```

```java
@Override
public void setList(List list) {
  Base.app.coffeeList = list;
  updateUI(this);
}
```

```java
@Override
public void updateUI(Fragment fragment) {
  //System.out.println("CALLING updateUI in CoffeeFragment");
  titleBar = (TextView)getActivity().findViewById(R.id.recentAddedBarTextView);
  titleBar.setText(R.string.recentlyViewedLbl);
  ((TextView)getActivity().findViewById(R.id.empty_list_view)).setText(R.string.r

  listAdapter = new CoffeeListAdapter(getActivity(), this, Base.app.coffeeList);
  listView.setAdapter (listAdapter);
  coffeeFilter = new CoffeeFilter(Base.app.coffeeList,"all",listAdapter);
```

Overriding the necessary methods from the interface

CoffeeMate 5.0 – Using AsyncTasks Vs Volley

- Using AsyncTasks
  - CoffeeApi
  - CallBackListener
  - Rest
  - TaskManager
  - CRUD Tasks x 6
- **Total = 10 Classes**

- Using Volley
  - CoffeeApi
  - VolleyListener
- **Total = 2 Classes**

Summary

- ❑ We looked at data persistence and multithreading in Android Development and how to use an SQLite database
- ❑ We covered a brief overview of JSON & Googles Gson
- ❑ We covered in detail the use of **`AsyncTasks`** and **`Volley`** to execute background tasks and make API calls
- ❑ We Compared the two in our CoffeeMate Case Study

References

- Victor Matos Notes – Lesson 13 (Cleveland State University)
- Android Developers
  http://developer.android.com/index.html

Sources

- http://en.wikipedia.org/wiki/JSON
- http://www.w3schools.com/json/
- http://www.json.org/
- http://json-schema.org
- http://www.nczonline.net/blog/2008/01/09/is-json-better-than-xml/
- http://en.wikipedia.org/wiki/SOAP_(protocol)
- http://en.wikipedia.org/wiki/REST
- http://stackoverflow.com/questions/16626021/json-rest-soap-wsdl-and-soa-how-do-they-all-link-together

Questions?