

Mobile Application Development

Produced
by

Eamonn de Leastar (edelestar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>

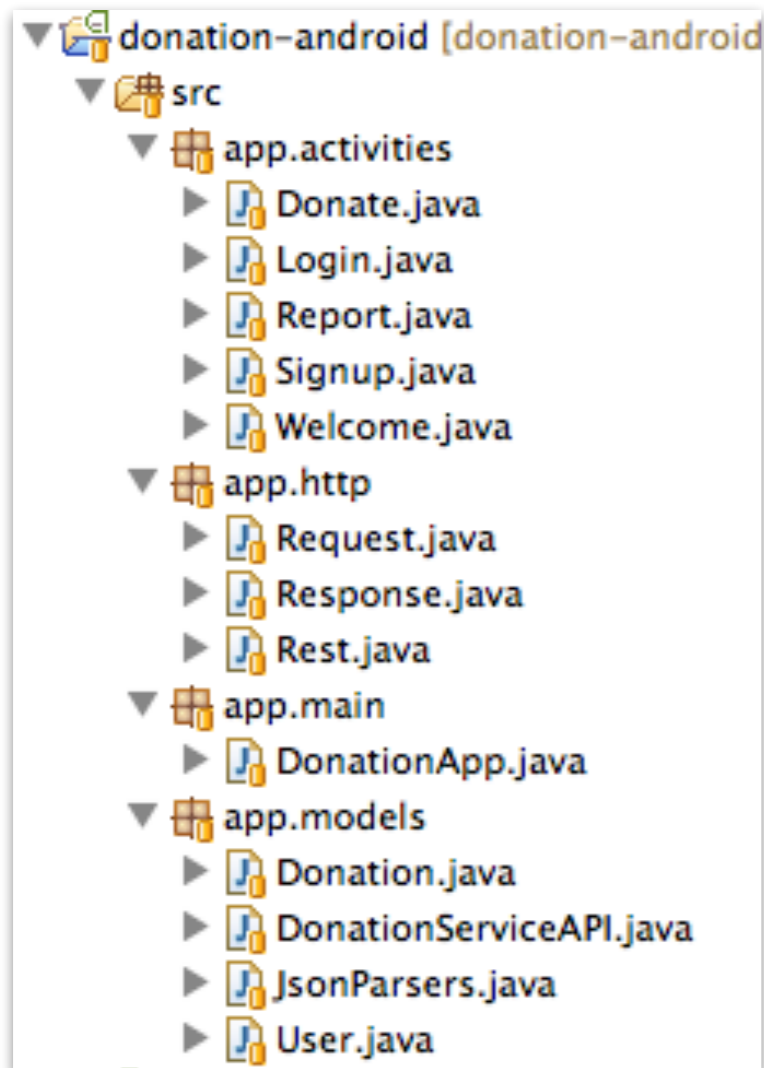


Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



Android <-> Play (3) - Android

donation-android Project v4



- **activities**

- display and handle all UI

- **main**

- retain application wide data structures (users + donations)

- **models**

- Gateway object for accessing donation-service application
- Local copies of core information models for the application (download from donation-play)
- Parsers (transformers) for converting objects into format suitable for upload/download to/from donation-service

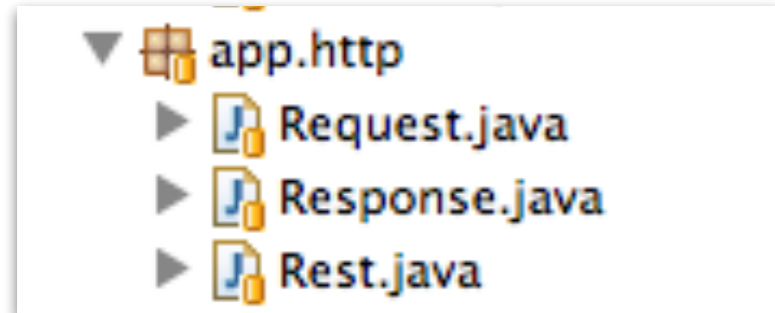
- **http**

- General purpose classes to support asynchronous http request/response to/from donation-service

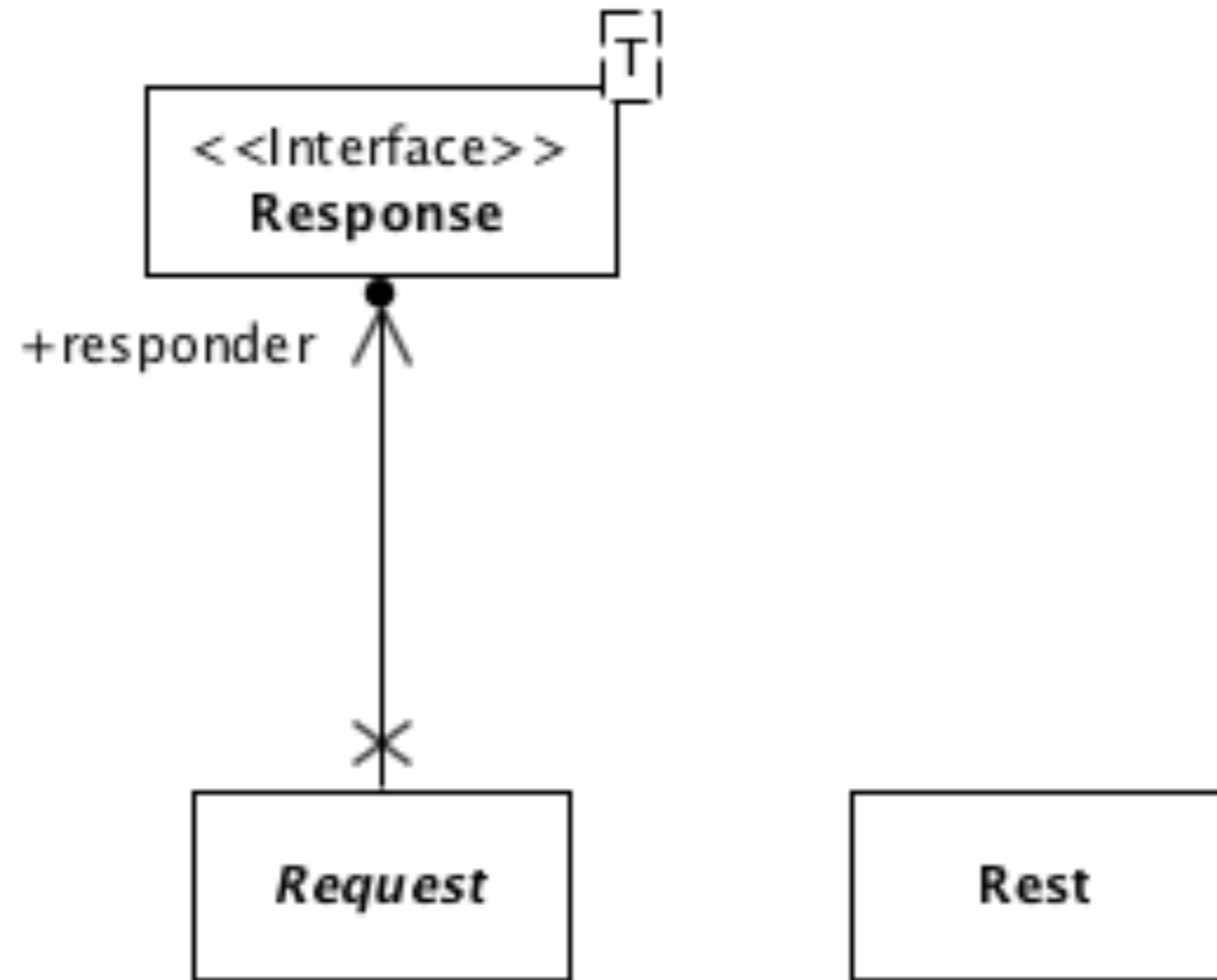
Android AsyncTask Class

- AsyncTask allows you to perform asynchronous work on your user interface.
- It performs the blocking operations in a **worker thread** and then publishes the results on the UI thread.
- You subclass AsyncTask and implement the doInBackground() callback method, which runs in a pool of background threads.
- To update your UI, you implement onPostExecute(), which delivers the result from doInBackground() and runs in the UI thread, so you can safely update your UI.

HTTP




- **http**
 - General purpose classes to support **asynchronous** http request/response to/from donation-service
 - These requests are performed in a separate thread of execution



This server is your development machine

Rest

- Issue HTTP requests to a server
 - GET
 - DELETE
 - PUT
 - POST
- (http verbs)



```
public class Rest
{
    private static final String URL = "http://10.0.3.2:9000";

    public static String get(String path) throws Exception
    {
        //...
    }

    public static String delete(String path)
    {
        //...
    }

    public static String put(String path, String json)
    {
        //...
    }

    public static String post(String path, String json)
    {
        //...
    }
}
```

Rest

```
public class Rest
{
    private static final String URL = "http://10.0.2.2:9000";

    public static String get(String path) throws Exception
    {
        //...
    }

    public static String delete(String path) throws Exception
    {
        //...
    }

    public static String put(String path, String json) throws Exception
    {
        //...
    }

    public static String post(String path, String json) throws Exception
    {
        //...
    }
}
```

- Rest class can only send/receive strings (no Model objects like User or Donation)
- Assumes all strings are Json encoded
- Will very likely throw 'exceptions' if server error, network problem or other related issue
- No need to edit/maintain this class as it adheres to HTTP protocol conventions
- Is independent of donation application, and can be used in other apps as is

Response

- An Interface that must be implemented by the Activity that initiated the request.
- Is 'paramaterised' by T, which will typically be some model object we are requesting/updating
 - e.g. User, Donation
- However, interface is application independent, and can be used in other applications not related to Donation app.

```
public interface Response<T>
{
    public void setReponse(List<T> aList);
    public void setReponse(T anObject);
    public void error0ccurred (Exception e);
}
```

Callbacks

```
public interface Response<T>
{
    public void setReponse(List<T> aList);
    public void setReponse(T anObject);
    public void errorOccurred (Exception e);
}
```

- When a request is made by an activity, then the activity will be ‘called back’ when a result becomes available.
- One of these three methods will be called:
 - A single object of type T is returned from the service
 - A list of T objects is returned
 - An error has occurred
- The callback will occur on the UI Thread, so the activity can update its components safely

Request

- Put up a dialog saying 'Processing...'
- Launch a background thread/task
- Report when finished to callback
- Reusable class, can be used in apps unrelated to donation.

```
public abstract class Request extends AsyncTask<Object, Void, Object>
{
    //...

    public Request(Context context, Response responder, String message)
    {
        //...
    }

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

    @Override
    protected Object doInBackground(Object... params)
    {
        //...
    }

    protected abstract Object doRequest(Object... params) throws Exception;

    @Override
    protected void onPostExecute(Object result)
    {
        //...
    }
}
```

Request

```
public abstract class Request extends AsyncTask<Object, Void, Object>
{
    //...

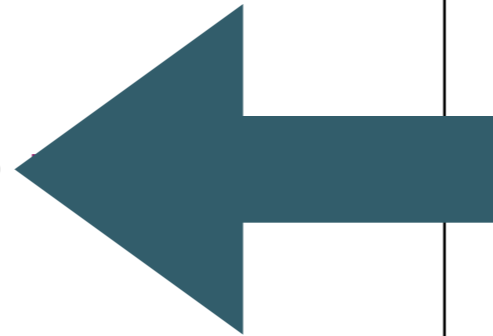
    public Request(Context context, Response responder, String message)
    {
        //...
    }

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

    @Override
    protected Object doInBackground(Object... params)
    {
        //...
    }

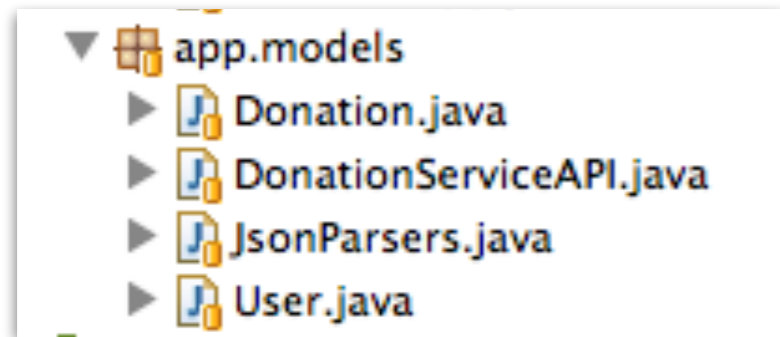
    protected abstract Object doRequest(Object... params)

    @Override
    protected void onPostExecute(Object result)
    {
        //...
    }
}
```



- An 'Abstract' class, so 'abstract' method 'doRequest' must be provided to do the actual background process
- For donation-android app, this will be a call to http.Rest methods to get/set data in android-service

donation-android Project v4



- **models**

- Gateway object for accessing donation-service application
- Local copies of core information models for the application (download from donation-play)
- Parsers (transformers) for converting objects into format suitable for upload/download to/from donation-service

User & Donation

```
public class User
{
    public String firstName;
    public String lastName;
    public String email;
    public String password;

    public User(String firstName, String lastName,
                String email,      String password)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
    }
}
```

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

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

    public String toString()
    {
        return amount + ", " + method;
    }
}
```

- No change from earlier versions

JsonParsers

- Same class as in donation-play!
- Convert Model object to/from Json format

```
public class JsonParsers
{
    static Gson gson = new Gson();

    public static User json2User(String json)
    {
        return gson.fromJson(json, User.class);
    }

    public static List<User> json2Users(String json)
    {
        Type collectionType = new TypeToken<List<User>>().getType();
        return gson.fromJson(json, collectionType);
    }

    public static String user2Json(Object obj)
    {
        return gson.toJson(obj);
    }

    public static Donation json2Donation(String json)
    {
        return gson.fromJson(json, Donation.class);
    }

    public static String donation2Json(Object obj)
    {
        return gson.toJson(obj);
    }

    public static List<Donation> json2Donations(String json)
    {
        Type collectionType = new TypeToken<List<Donation>>().getType();
        return gson.fromJson(json, collectionType);
    }
}
```

DonationServiceAPI

- Enable Activities to 'invoke' services on donation-service app.
- Specifically:
 - GetUsers
 - GetDonations
 - CreateUser
 - CreateDonation
- Each of these requests is 'spun-out' into separate thread

```
public class DonationServiceAPI
{
    public static void getUsers(Context context,
                               Response<User> response,
                               String dialogMessage)
    {
        new GetUsers(context, response, dialogMessage).execute();
    }

    public static void createUser(Context context,
                                  Response<User> response,
                                  String dialogMessage)
    {
        new CreateUser(context, response, dialogMessage).execute(user);
    }

    public static void getDonations(Context context,
                                     Response<User> response,
                                     String dialogMessage)
    {
        new GetDonations(context, response, dialogMessage).execute();
    }

    public static void createDonation(Context context,
                                       Response<User> response,
                                       String dialogMessage)
    {
        new CreateDonation(context, response, dialogMessage).execute(donation);
    }
}
```


GetUsers and CreateUser Requests

- The doRequest() methods will run in a background thread
- ... and will use the Rest class to communicate with the server

```
class GetUsers extends Request
{
    public GetUsers(Context context, Response<User> callback, String message)
    {
        super(context, callback, message);
    }

    @Override
    protected List<User> doRequest(Object... params) throws Exception
    {
        String response = Rest.get("/api/users");
        List<User> userList = JsonParsers.json2Users(response);
        return userList;
    }
}

class CreateUser extends Request
{
    public CreateUser(Context context, Response<User> callback, String message)
    {
        super(context, callback, message);
    }

    @Override
    protected User doRequest(Object... params) throws Exception
    {
        String response = Rest.post ("/api/users", JsonParsers.user2Json(params[0]));
        return JsonParsers.json2User(response);
    }
}
```

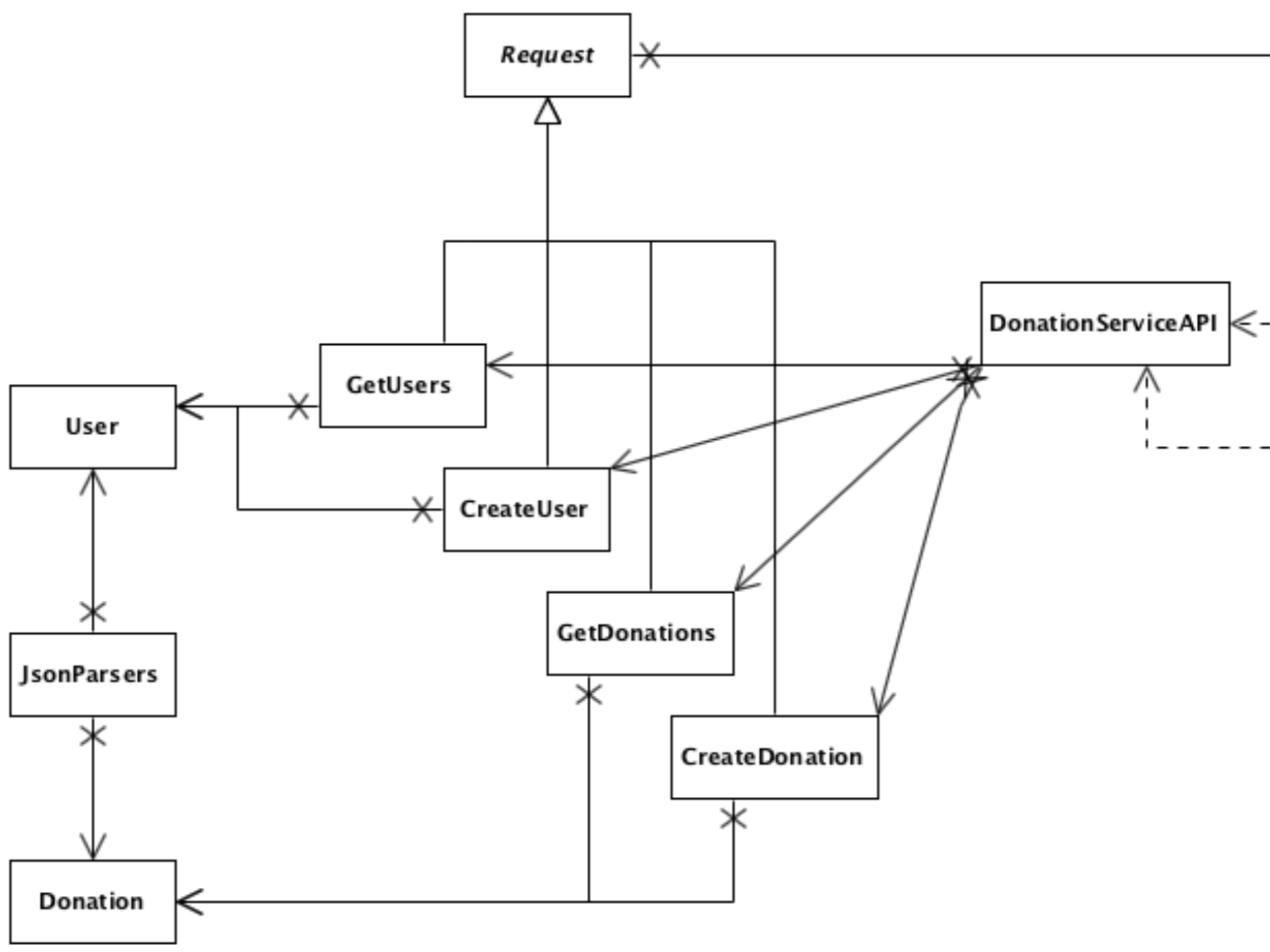
GetDonations and CreateDonation Requests

```
class GetDonations extends Request
{
    public GetDonations(Context context, Response<Donation> callback, String message)
    {
        super(context, callback, message);
    }

    @Override
    protected List<Donation> doRequest(Object... params) throws Exception
    {
        String response = Rest.get("/api/donations");
        List<Donation> donationList = JsonParsers.json2Donations(response);
        return donationList;
    }
}

class CreateDonation extends Request
{
    public CreateDonation(Context context, Response<Donation> callback, String message)
    {
        super(context, callback, message);
    }

    @Override
    protected Donation doRequest(Object... params) throws Exception
    {
        String response = Rest.post ("/api/donations", JsonParsers.donation2Json(params[0]));
        return JsonParsers.json2Donation(response);
    }
}
```



Activities

- If an Activity needs to make a request of the donation-service, it must do two things:
 1. Initiate a request by calling one of the methods in DonationServiceAPI
 2. Implement the Response interface, through which the response (or error) will be delivered.

```
public class DonationServiceAPI
{
    public static void getUsers(..)
        //..
    public static void createUser(..)
        //..
    public static void getDonations(..)
        //..
    public static void createDonation(..)
}

```

```
public interface Response<T>
{
    public void setReponse(List<T> aList);
    public void setReponse(T anObject);
    public void errorOccurred (Exception e);
}

```

Login

- When the Login activity starts we can:
 - Request the list of users from the donation-service
 - ... and when those requests arrive, we store them in the application object
- Then, when Login button pressed, we authenticate against this list as usual.

Login Activity

(no features
hidden)

```
public class Login extends Activity implements Response<User>
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        DonationServiceAPI.getUsers(this, this, "Retrieving list of users");
    }

    public void signinPressed (View view)
    {
        DonationApp app = (DonationApp) getApplication();
        TextView email    = (TextView) findViewById(R.id.loginEmail);
        TextView password = (TextView) findViewById(R.id.loginPassword);
        if (app.validateUser(email.getText().toString(), password.getText().toString()))
        {
            startActivity (new Intent(this, Donate.class));
        }
        else
        {
            Toast toast = Toast.makeText(this, "Invalid Credentials", Toast.LENGTH_SHORT);
            toast.show();
        }
    }

    @Override
    public void setResponse(List<User> aList)
    {
        DonationApp app = (DonationApp) getApplication();
        app.users = aList;
    }

    @Override
    public void errorOccurred(Exception e)
    {
        Toast toast = Toast.makeText(this, "Donation Service Unavailable. Try again later",
                                     Toast.LENGTH_LONG);

        toast.show();
        startActivity (new Intent(this, Welcome.class));
    }

    @Override
    public void setResponse(User anObject)
    {}
}
}
```

Login Activity

```
public class Login extends Activity implements Response<User>
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        DonationServiceAPI.getUsers(this, this, "Retrieving list of users");
    }

    public void signinPressed (View view)
    {
        DonationApp app = (DonationApp) getApplication();

        TextView email = (TextView) findViewById(R.id.loginEmail);
        TextView password = (TextView) findViewById(R.id.loginPassword);

        if (app.validateUser(email.getText().toString(), password.getText().toString()))
        {
            startActivity (new Intent(this, Donate.class));
        }
        else
        {
            Toast toast = Toast.makeText(this, "Invalid Credentials", Toast.LENGTH_SHORT);
            toast.show();
        }
    }

    //...
}
```

Login Activity

```
public class Login extends Activity implements Response<User>
{
    //..
    @Override
    public void setReponse(List<User> aList)
    {
        DonationApp app = (DonationApp) getApplication();
        app.users = aList;
    }

    @Override
    public void errorOccurred(Exception e)
    {
        Toast toast = Toast.makeText(this, "Donation Service Unavailable. Try again later",
                                     Toast.LENGTH_LONG);

        toast.show();
        startActivity (new Intent(this, Welcome.class));
    }

    @Override
    public void setReponse(User anObject)
    {}
}
```


Report Activity

```
public class Report extends Activity implements Response <Donation>
{
    private ListView      listView;
    private DonationApp   app;
    private DonationAdapter adapter;

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    //.. no change

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    //.. no change

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

        app = (DonationApp) getApplication();

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

        listView.setAdapter(adapter);

        DonationServiceAPI.getDonations(this, this, "Downloading Donations List..");
    }

    @Override
    public void setReponse(List<Donation> aList)
    {
        app.donations = aList;
        adapter.donations = aList;
        adapter.notifyDataSetChanged();
    }

    @Override
    public void setReponse(Donation anObject)
    {
    }

    @Override
    public void errorOccurred(Exception e)
    {
        Toast toast = Toast.makeText(this, "Donation Service Unavailable!", Toast.LENGTH_LONG);
        toast.show();
        startActivity (new Intent(this, Welcome.class));
    }
}
```

Report Activity

```
public class Report extends Activity implements Response <Donation>
{
    private ListView      listView;
    private DonationApp   app;
    private DonationAdapter adapter;

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

        app = (DonationApp) getApplication();

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

        listView.setAdapter(adapter);

        DonationServiceAPI.getDonations(this, this, "Downloading Donations List..");
    }
}
```

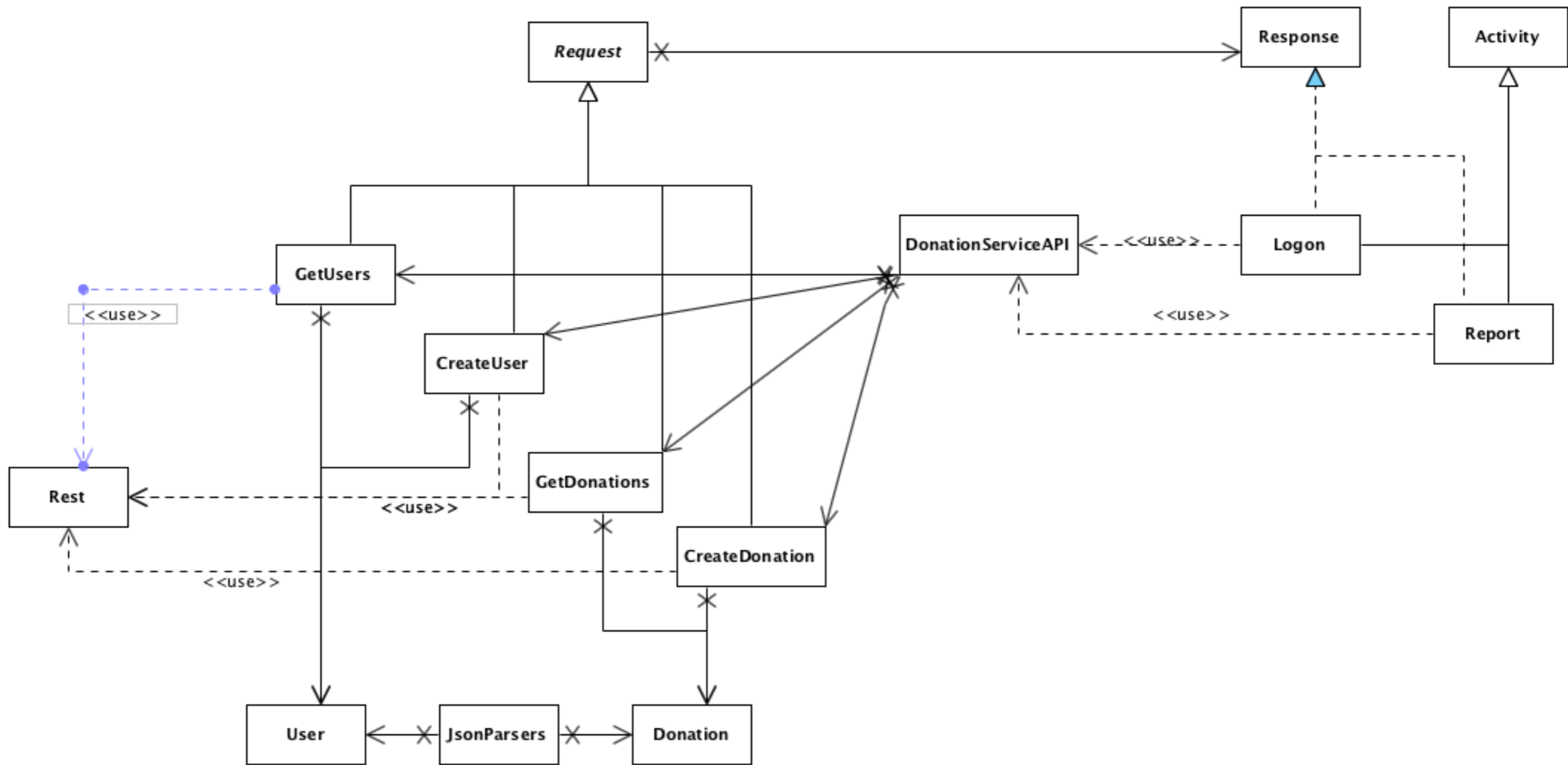
Report Activity

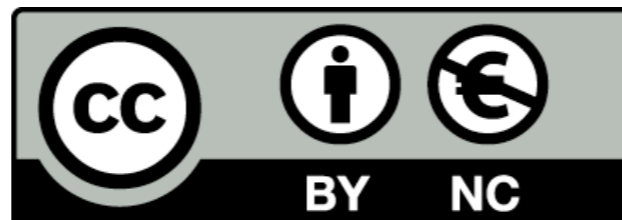
```
public class Report extends Activity implements Response <Donation>
{
    private ListView      listView;
    private DonationApp   app;
    private DonationAdapter adapter;

    @Override
    public void setReponse(List<Donation> aList)
    {
        app.donations      = aList;
        adapter.donations = aList;
        adapter.notifyDataSetChanged();
    }

    @Override
    public void setReponse(Donation anObject)
    {
    }

    @Override
    public void error0ccurred(Exception e)
    {
        Toast toast = Toast.makeText(this, "Donation Service Unavailable!", Toast.LENGTH_LONG);
        toast.show();
        startActivity (new Intent(this, Welcome.class));
    }
}
```





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

Some of this material is adapted from <http://www.vogella.com/articles/JavaConcurrency/article.html>
An excellent source for well structured tutorials and explanations of all thing related ot Java, Eclipse and Android development

