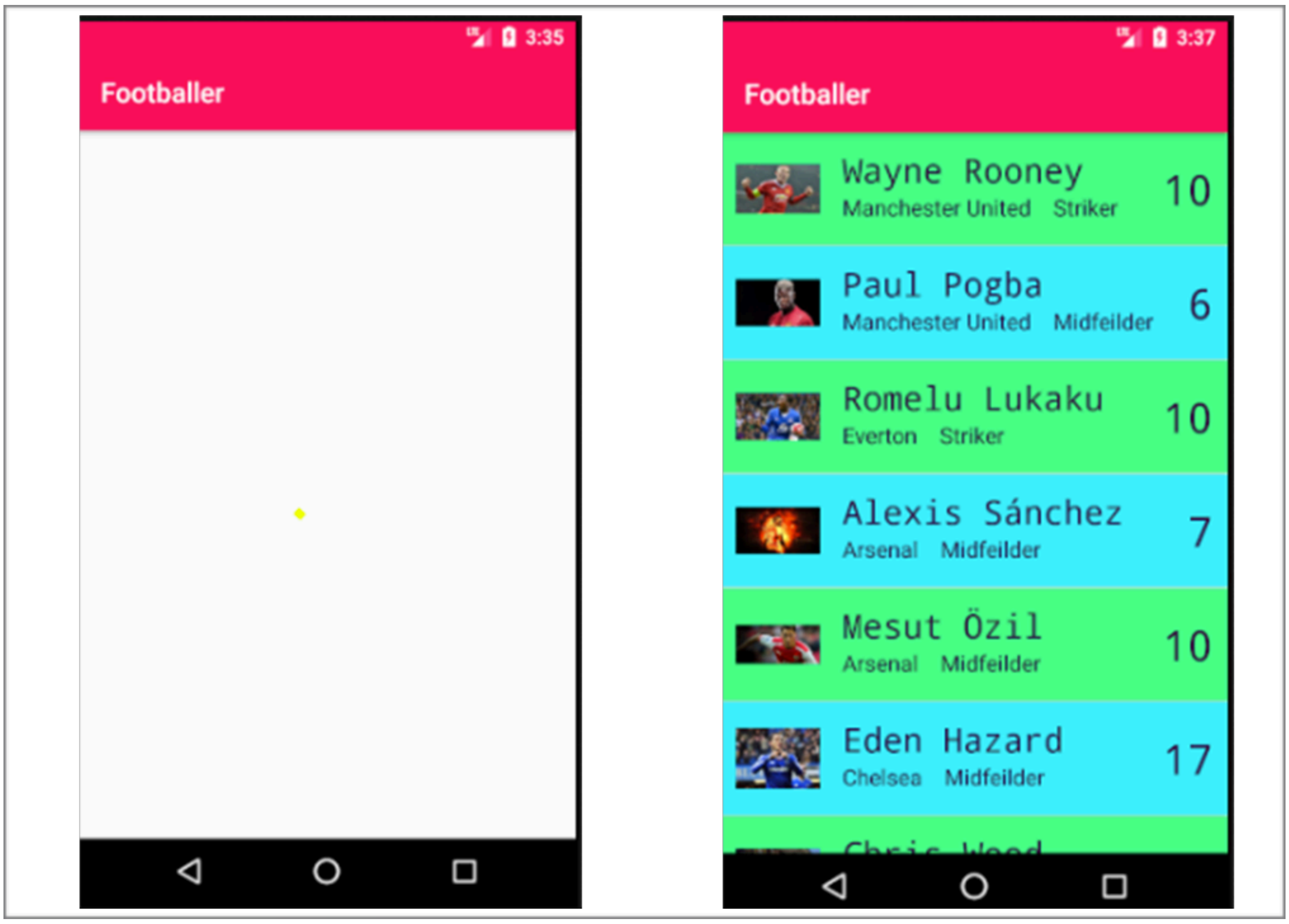
# Android Development

## OOPD - MM3 - Ca2 -Anne Wright

JohnKenny - 21 March 2017



Wayne Rooney 10
Manchester United Striker

Paul Pogba 6
Manchester United Midfeilder

Romelu Lukaku 10
Everton Striker

Alexis Sánchez 7
Arsenal Midfeilder

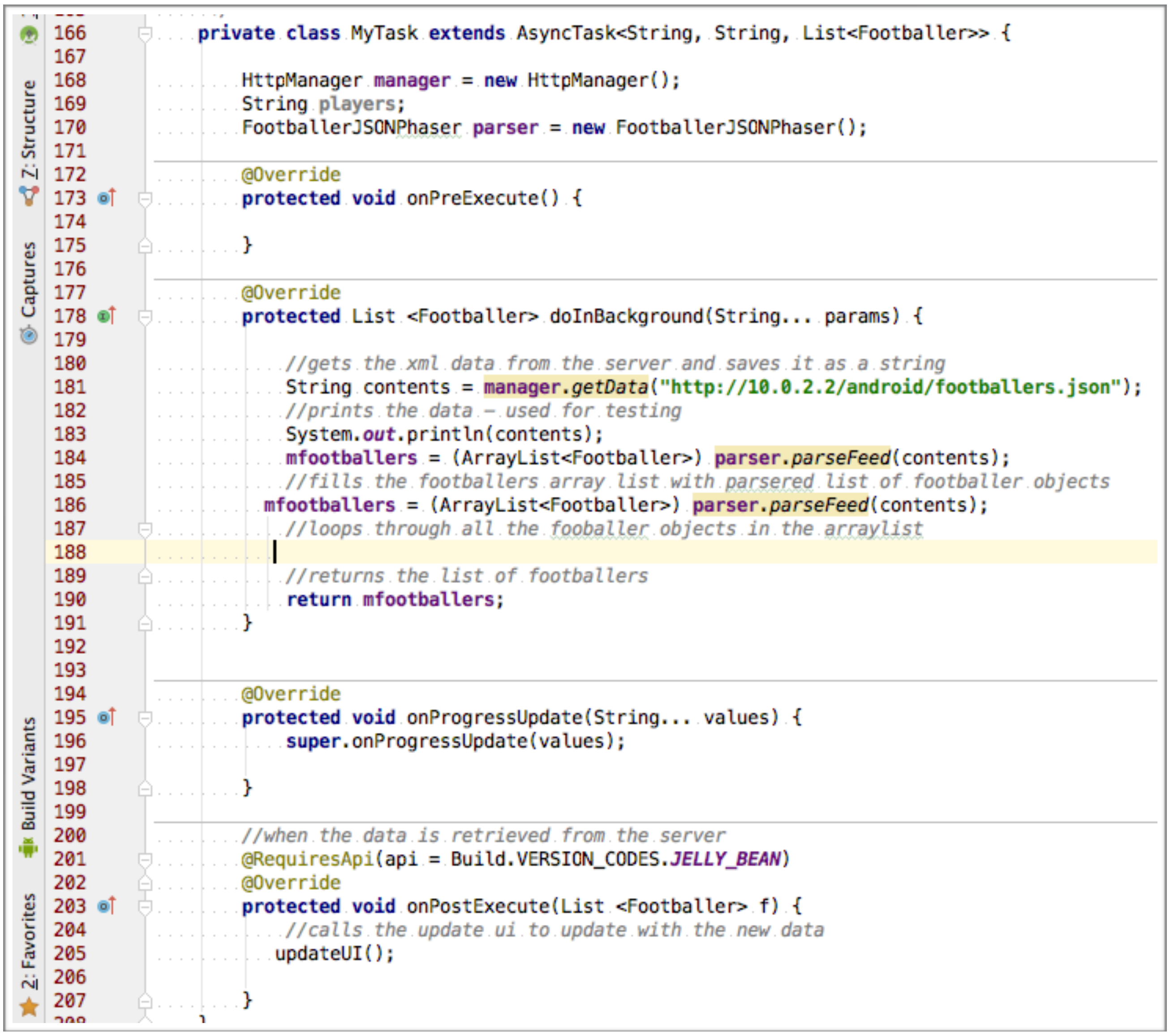
Mesut Özil 10
Arsenal Midfeilder

Eden Hazard 17
Chelsea Midfeilder

## Introduction

The following report is based on my second Android application. The application was created using Android studio and is composed of XML and Java files. The application connects to a remote server (localhost using XAMPP) and downloads a XML or JSON file. The data in the XML and JSON, mirrors the data in the main class of the project(Footballer class). The data is then parsed form the JSON or XML file an an ArrayList of footballer objects is created using the parsed data. The application then downloads all the image assets from the server using the image location from the parsed file and stores the corresponding image with each footballer object. Each footballer is then displayed in a ListView using a fragment ListView. The application is styled using the EPL banding, colours and the application logo is set to the EPL logo.

## Asynchronous Threads



```java
    private class MyTask extends AsyncTask<String, String, List<Footballer>> {

        HttpManager manager = new HttpManager();
        String players;
        FootballerJSONPhaser parser = new FootballerJSONPhaser();

        @Override
        protected void onPreExecute() {

        }

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

            //gets the xml data from the server and saves it as a string
            String contents = manager.getData("http://10.0.2.2/android/footballers.json");
            //prints the data - used for testing
            System.out.println(contents);
            mfootballers = (ArrayList<Footballer>) parser.parseFeed(contents);
            //fills the footballers array list with parsered list of footballer objects
           mfootballers = (ArrayList<Footballer>) parser.parseFeed(contents);
            //loops through all the fooballer objects in the arraylist

            //returns the list of footballers
            return mfootballers;
        }


        @Override
        protected void onProgressUpdate(String... values) {
            super.onProgressUpdate(values);

        }

        //when the data is retrieved from the server
        @RequiresApi(api = Build.VERSION_CODES.JELLY_BEAN)
        @Override
        protected void onPostExecute(List <Footballer> f) {
            //calls the update ui to update with the new data
           updateUI();

        }
```

Figure 1

In an Android application, the main thread is utilised for the core components on the UI. If all tasks are running on the main thread it means that no other events can start until the previous one completes. Asynchronous threads are used to run a new task on a separate thread in the background. This allows the application to function normally with no impact on the end user while the new thread is running in the background. In the application created for this project, resources are downloaded from a web server using Asynchronous threads. Figure 1 shows the class used to create the asynchronous thread to download the JSON data from the web server. A HTTP manager object created using the HTTP manager class, is used to connect to the web server and download the contents of the JSON file and store the data as a String.

On line 178 a new background method is created and the contents are parsed and returned as an ArrayList of footballers. This return is not by the application on line 186 the class level ArrayList is filled with the list of footballer objects, just downloaded from the server. The onPostExecute (Line 203) method excuses when the doInBackground method completes and all the data is downloaded from the server. This method calls the updateUI method from the FootballerListFragment, (which the MyTask is an inner class of) and this method updates the UI and populates aa new FragmentList of Footballer objects. The updateUI method was also called in the onResume method and this caused the app to crash so it was removed from the onResume method.

## Footballer Class

The main class used to create content is the Footballer class. This class is very similar to standard Java classes and is used to create Footballer objects following Java's standard Object Orientated Programming principles. The Footballer class has a constructor used to create new Footballers and it has a list of private variables which can be accessed and modified using getter and setter methods. The only thing different about the Footballer class and the class from the first ca and this footballer class is that the image is stored as a bitmap instead of a byte array. The JSON and XML files contain data used to create the footballer objects, for example the JSON contains and array of footballers.

## Class Diagram

The class diagram in figure 2 represents the Android application. Figure 2 illustrates all the properties a Footballer object contains and how each class relates and interacts with each other. The FootballerListFragment is used to display the list of footballer objects. The FootballerListFragment class contains two inner classes, they are MyTask and the footballerAdapter. MyTask uses the HTTPManager class to download data from the web server and the Parser classes to parse the data to create footballer objects. The FootballerAdaper class is used to populate the data from the footballer objects into the to list items.

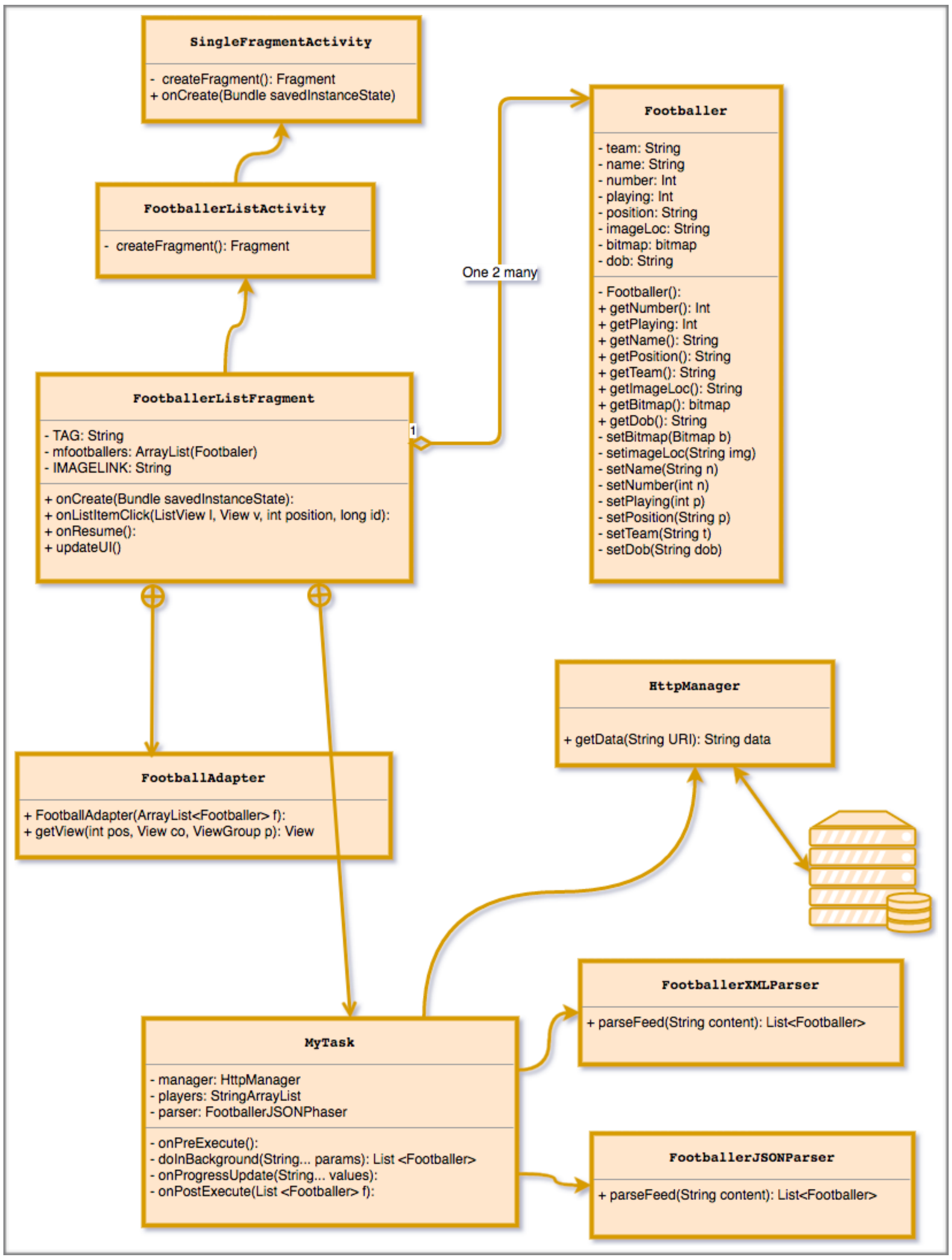
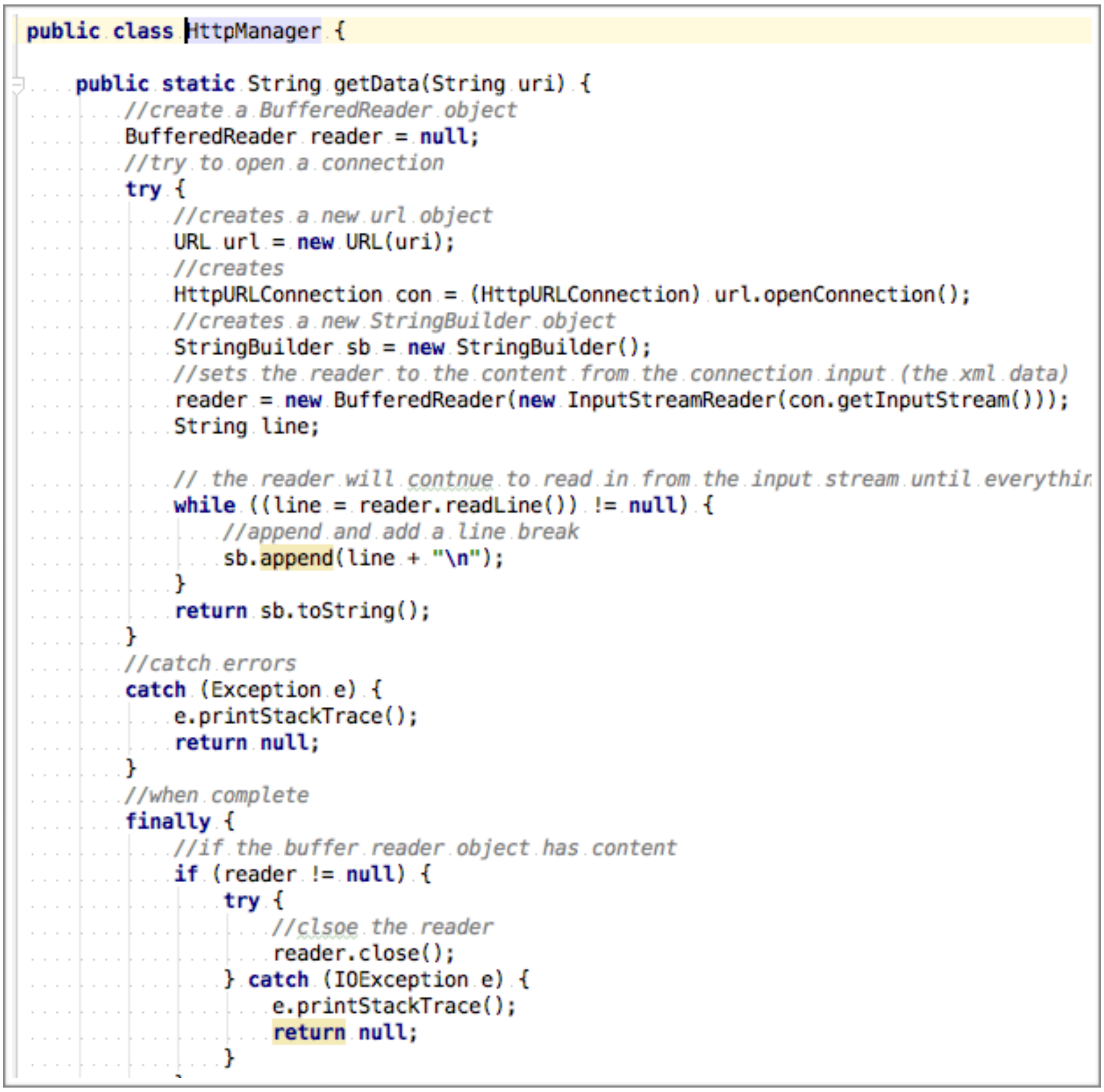


Figure 2

## HTTPManager

Figure 3 illustrates the HTTPManager class used to connect to the server and get the data. The same HTTPManager is used by both the XML and JSON version of the application. Similar to the file input in Java a BufferReader object is created and used to read each line of the file. A URL is passed into this method allowing the came class to be reusable with various URLs allowing for code reuse. A String is returned from the main betted of this class representing the data downloaded from the server.

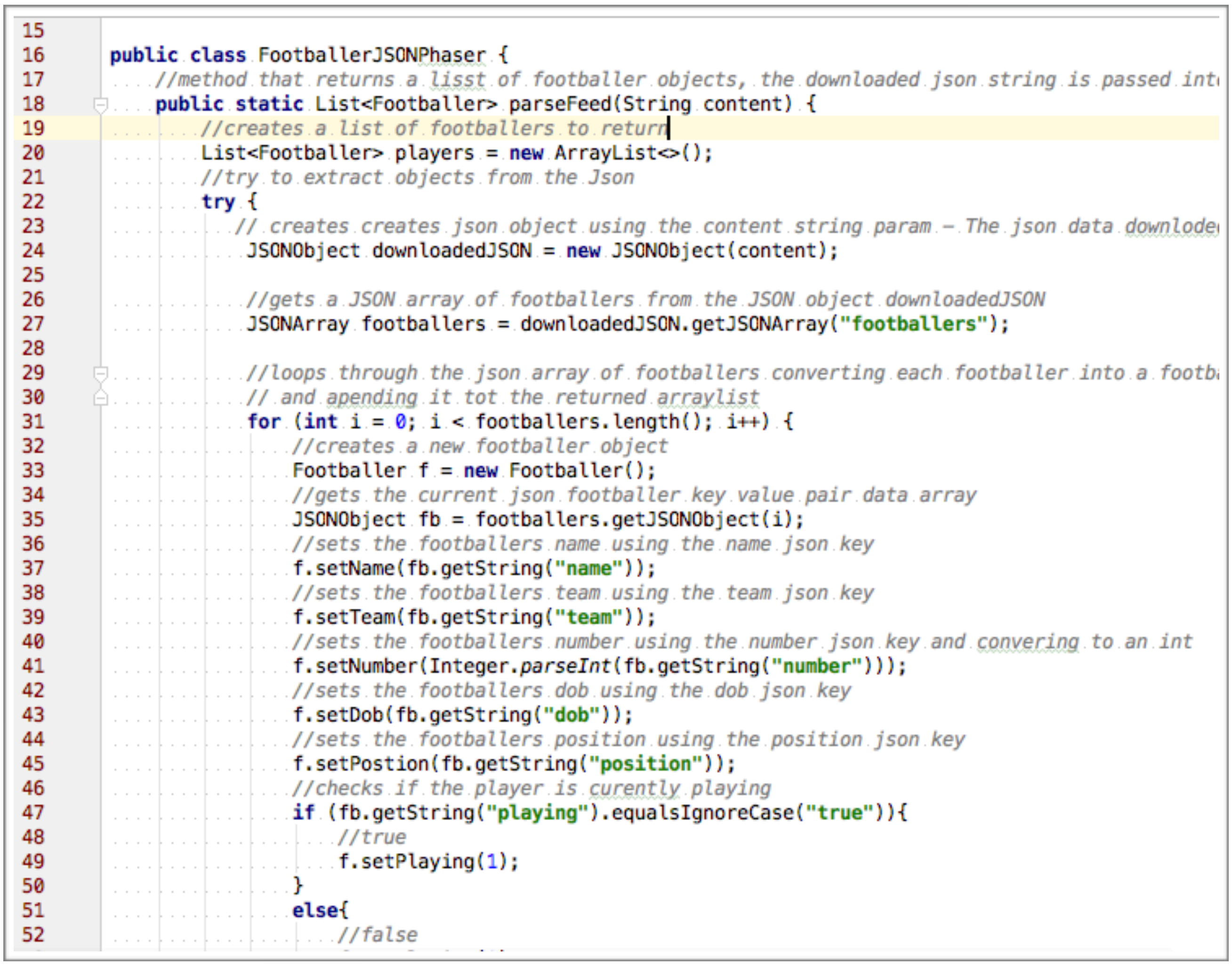


```
public class HttpManager {

    public static String getData(String uri) {
        //create a BufferedReader object
        BufferedReader reader = null;
        //try to open a connection
        try {
            //creates a new url object
            URL url = new URL(uri);
            //creates
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            //creates a new StringBuilder object
            StringBuilder sb = new StringBuilder();
            //sets the reader to the content from the connection input (the xml data)
            reader = new BufferedReader(new InputStreamReader(con.getInputStream()));
            String line;

            // the reader will contnue to read in from the input stream until everythin
            while ((line = reader.readLine()) != null) {
                //append and add a line break
                sb.append(line + "\n");
            }
            return sb.toString();
        }
        //catch errors
        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        //when complete
        finally {
            //if the buffer reader object has content
            if (reader != null) {
                try {
                    //clsoe the reader
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                    return null;
                }
```

Figure 3

## Parser

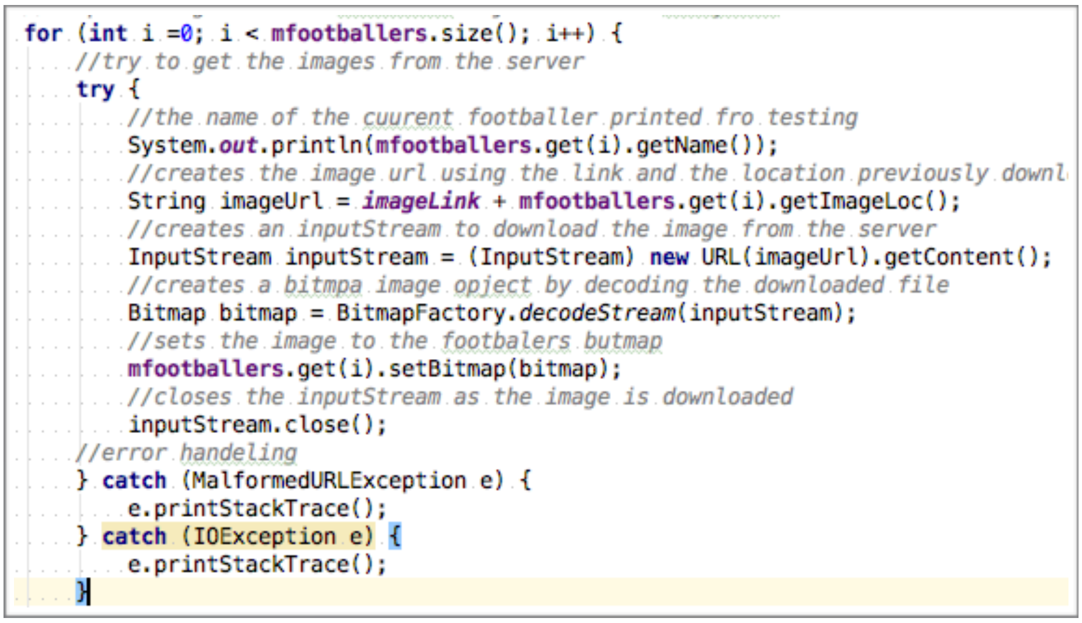
This application contains two parsers one for the XML content and one for the JSON content. Figure 4 demonstrates the JSON parser class. Both parser classes follow the same structure however the XML parser uses a while loop to loop through all the objects XML footballer objects and a switch statement to extract the data from the tags to create footballer objects. The JSON parser class shown in figure 4, contains a method that returns a list of footballer objects using a String passed into the method (this String would contain the JSON data downloaded from the server using the HTTP manager class). The String is then used to create a JSON Java object, containing a JSON array of footballer objects. A for loop is then used to loop through each JSON footballer and a footballer object is created using the footballer class.



```
public class FootballerJSONPhaser {
    //method that returns a lisst of footballer objects, the downloaded json string is passed int
    public static List<Footballer> parseFeed(String content) {
        //creates a list of footballers to return
        List<Footballer> players = new ArrayList<>();
        //try to extract objects from the Json
        try {
           // creates creates json object using the content string param - The json data downlode
            JSONObject downloadedJSON = new JSONObject(content);

            //gets a JSON array of footballers from the JSON object downloadedJSON
            JSONArray footballers = downloadedJSON.getJSONArray("footballers");

            //loops through the json array of footballers converting each footballer into a footb
            // and apending it tot the returned arraylist
            for (int i = 0; i < footballers.length(); i++) {
                //creates a new footballer object
                Footballer f = new Footballer();
                //gets the current json footballer key value pair data array
                JSONObject fb = footballers.getJSONObject(i);
                //sets the footballers name using the name json key
                f.setName(fb.getString("name"));
                //sets the footballers team using the team json key
                f.setTeam(fb.getString("team"));
                //sets the footballers number using the number json key and convering to an int
                f.setNumber(Integer.parseInt(fb.getString("number")));
                //sets the footballers dob using the dob json key
                f.setDob(fb.getString("dob"));
                //sets the footballers position using the position json key
                f.setPostion(fb.getString("position"));
                //checks if the player is curently playing
                if (fb.getString("playing").equalsIgnoreCase("true")){
                    //true
                    f.setPlaying(1);
                }
                else{
                    //false
```

Figure 4

## Image Download

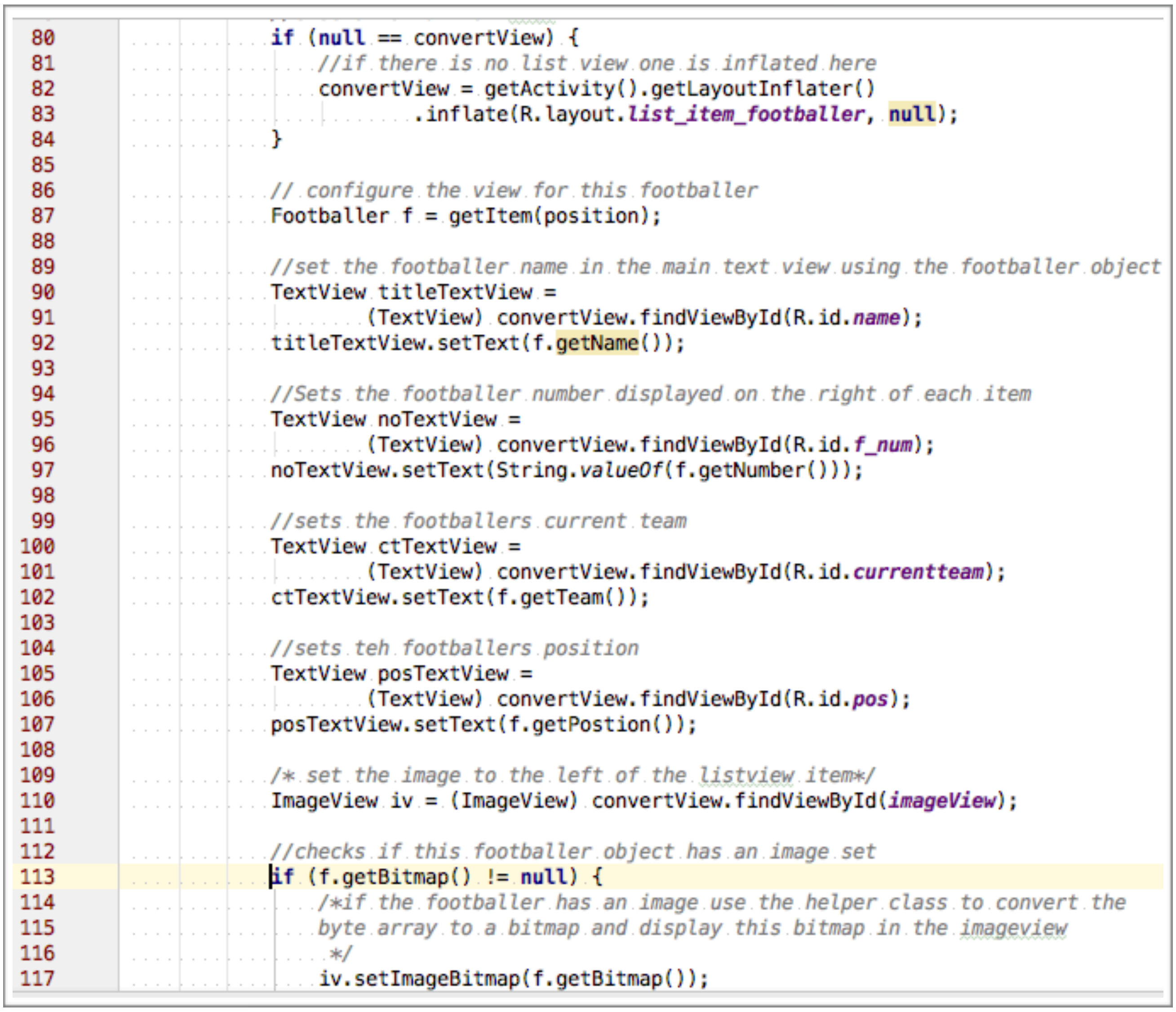


```
for (int i =0; i < mfootballers.size(); i++) {
    //try to get the images from the server
    try {
        //the name of the cuurent footballer printed fro testing
        System.out.println(mfootballers.get(i).getName());
        //creates the image url using the link and the location previously downl
        String imageUrl = imageLink + mfootballers.get(i).getImageLoc();
        //creates an inputStream to download the image from the server
        InputStream inputStream = (InputStream) new URL(imageUrl).getContent();
        //creates a bitmpa image opject by decoding the downloaded file
        Bitmap bitmap = BitmapFactory.decodeStream(inputStream);
        //sets the image to the footbalers butmap
        mfootballers.get(i).setBitmap(bitmap);
        //closes the inputStream as the image is downloaded
        inputStream.close();
    //error handeling
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 5

In the async thread, code was added to use the file location (contained in the JSON & XML files) to go back to the web server and download the image for each footballer object. Figure 5 illustrates the code to download the images for each footballer and set the bitmap variable of each footballer to the image.

## Fill List With Footballer Objects



```
80          if (null == convertView) {
81              //if there is no list view one is inflated here
82              convertView = getActivity().getLayoutInflater()
83                      .inflate(R.layout.list_item_footballer, null);
84          }
85
86          // configure the view for this footballer
87          Footballer f = getItem(position);
88
89          //set the footballer name in the main text view using the footballer object
90          TextView titleTextView =
91                  (TextView) convertView.findViewById(R.id.name);
92          titleTextView.setText(f.getName());
93
94          //Sets the footballer number displayed on the right of each item
95          TextView noTextView =
96                  (TextView) convertView.findViewById(R.id.f_num);
97          noTextView.setText(String.valueOf(f.getNumber()));
98
99          //sets the footballers current team
100         TextView ctTextView =
101                 (TextView) convertView.findViewById(R.id.currentteam);
102         ctTextView.setText(f.getTeam());
103
104         //sets teh footballers position
105         TextView posTextView =
106                 (TextView) convertView.findViewById(R.id.pos);
107         posTextView.setText(f.getPostion());
108
109         /* set the image to the left of the listview item*/
110         ImageView iv = (ImageView) convertView.findViewById(imageView);
111
112         //checks if this footballer object has an image set
113         if (f.getBitmap() != null) {
114             /*if the footballer has an image use the helper class to convert the
115             byte array to a bitmap and display this bitmap in the imageview
116              */
117             iv.setImageBitmap(f.getBitmap());
```

Figure 6

Figure 6 illustrates the code used to create the UI list elements using the ArrayList of footballer objects and the array adapter. Line 90 to 105 uses the text variables from the footballer object using the get methods and sets the connects of the TextView boxes on each list item. The image view displayed on the left of the list elements displays the bitmap image of the footballer downloaded from the server. The image is retrieved from each footballer object using the getBitmap method. Before the image is added to the list an if statement ensures that the objects Bitmap is not null, displayed on line 113.

## Conclusion

The Android application is fully functional and tested to ensure everything works as it should. There are two versions of the application one to parse XML data and one to parse JSON data. After completing this project I feel I now have a good grasp on Android programming. I now feel comfortable with the styling and creating new fragments. This was my first project using in android using asynchronous threads and downloading data and resources from a server. Ideally would more time I would have liked to add a feature to edit the download footballer objects and save the updates to the XML or JSON file.