# Lab assignment 8

**Due date**: Saturday, June 25

40 points

**Team of 4 students**

Divide the team into two subgroups with two team members per group.

The group works on the lab assignment and can discuss the code with another group. Each group demonstrates the lab to each other. The team then upload the solution from one of the subgroups.

Write the contribution to the lab assignment, the contribution percentage (maximum of 100%) on the comment box for each team member.

All the team members must agree on the percentage. If the team members have 50% and the lab assignment grade is 40/40, they get only 20 points. The instructor will review the paper summarizing and validate the percentage of each team member. If you have problems with team members, please let me know.

## Problem

Smartphones and tablets include a browser so users can surf the Internet. However, a regular web page displayed on a small screen can be overwhelming. Content apps are apps that display selected content from a website so that the amount of information is manageable and the user experience is optimal. Such content apps exist for many sites, particularly social media websites and news sites. Often, websites offer frequently updated, consistently formatted information available to the outside world in general and app developers in particular. One such formatted information is called a Really Simple Syndication (RSS) feed. A typical RSS feed, not only is formatted in XML, but also often uses generic elements such as item, title, link, description, or pubDate. In order to build a content app for a particular site, we need to understand the formatting of that website's RSS feed, and we need to be able to parse the XML file that is exposed by that website so that we can display its contents.

## Design

### Parsing XML, DOM, and SAX Parsers, Web Content App, Version 0

An XML document is typically accompanied by a Document Type Declaration (DTD) document: it contains the rules that an XML document should adhere to.In Version 0 of the app, we do three things:

There are two types of parsers to parse an XML document:

- DOM parsers
- SAX parsers

A Document Object Model (DOM) parser converts an XML file into a tree. Once the XML has been converted into a tree, we can navigate the tree to modify elements, add elements, delete elements, or search for elements or values. This is similar to what a browser does with an HTML document: it builds a tree, and once that tree data structure is in memory, it can be accessed, searched, or modified using JavaScript.

Using a DOM parser is a good option if we use an XML document as a replacement for a database and we are interested in accessing and modifying that database. That is often not the case for content apps. Content apps typically convert the XML document into a list and allow the user to interact with the list. Often, that list contains links to URLs.

In order to convert an XML document into a list, we use a Simple API for XML (SAX) parser. A SAX parser builds a list as it reads the XML document. The elements of that list are objects. It is generally thought that a SAX parser is faster than a DOM parser.

For example, let's assume that we want to parse the XML document shown below.

```
<?xml version="1.0" encoding="utf-8" ?>
<rss>
  <item>
    <title>Facebook</title>
    <link>http://www.facebook.com</link>
  </item>
  <item>
    <title>Twitter</title>
    <link>http://www.twitter.com</link>
  </item>
  <item>
    <title>Google</title>
    <link>http://www.google.com</link>
  </item>
</rss>
```

**A simple XML document**

A DOM parser would convert that XML document to a tree similar to the tree shown in FIGURE 13.1. There are actually a few more branches in the tree than Figure 1 shows because empty strings between elements are also branches of the tree.

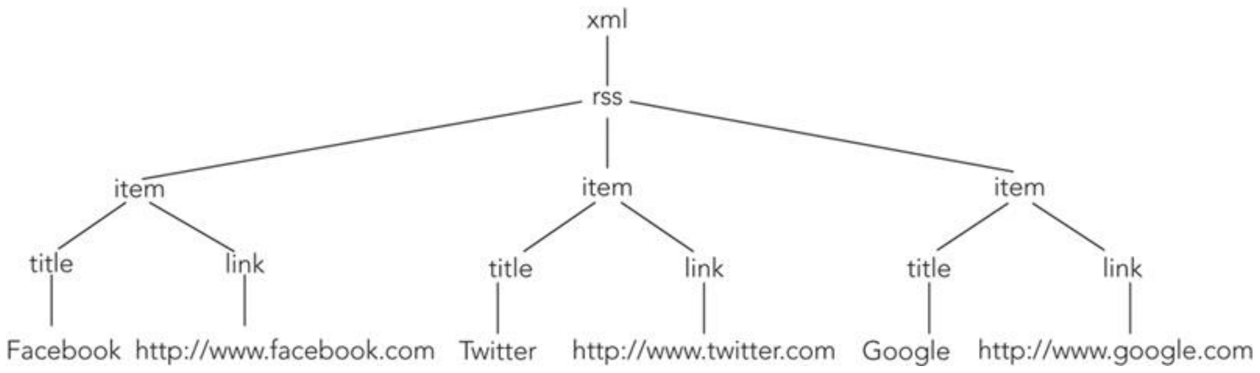


**Figure 1 A tree for a simple XML document**

Instead of looking at that XML document as a tree, we can also look at it as a list of three items. In this document, each item has a title and a link. We can code an Item class with two String instance variables, title and link. A SAX parser would convert that XML document to a list of items containing three elements as shown in figure 2.

```
{ { title: Facebook; link: http://www.facebook.com }, { title:
Twitter; link: http://www.twitter.com }, { title: Google; link:
http://www.google.com } }
```

**Figure 2 A list of items for the simple XML document**

In Version 0 of our app, we concentrate on the process of XML parsing using a SAX parser. We use an existing SAX parser to parse the simple XML document and output feedback on the various steps of the parsing process to Logcat. In order to do this, we build two classes:

‣ The SAXHandler class processes the various elements of the XML document, such as start tags, end tags, and text between tags, as they are encountered by the SAX parser.

‣ The MainActivity class.

We will create a raw directory within the res directory and place a file named test.xml containing the simple XML document in it. In the version 1, we read an XML document dynamically from a URL.

The DefaultHandler class is the base class for SAX level 2 event handlers. The Default-Handler class implements four interfaces: EntityResolver, DTDHandler, ContentHandler, and ErrorHandler. It implements all the methods that it inherits from these four interfaces as do-nothing methods.

**TABLE 0** The `EntityResolver`, `DTDHandler`, `ContentHandler`, and `ErrorHandler` interfaces

| Interface | Receives Notifications For | Method Names |
|---|---|---|
| EntityResolver | Entity-related events | resolveEntity |
| DTDHandler | DTD-related events | notationDcl, unparsedEntityDcl |
| ContentHandler | Logic-related events | startDocument, endDocument, startElement, endElement, characters, and other methods |
| ErrorHandler | Errors and warnings | error, fatalError, warning |

When parsing an XML file using a SAX parser, there are many events that can happen: for example, whenever the parser encounters a start tag, an end tag, or text, it is an event. The DefaultHandler class includes a number of methods that are automatically called when parsing a document and an event occurs. TABLE 1 shows some of these methods. For example, when the parser encounters a start tag, the startElement method is called. When the parser encounters an end tag, the endElement method is called. When the parser encounters text, the characters method is called. These methods are implemented as do-nothing methods. In order to handle these events, we need to subclass the DefaultHandler class and override the methods that we are interested in.

**TABLE 1** Selected methods of the `DefaultHandler` class

| Method | Description |
|---|---|
| void startElement( String uri, String localName, String startElement, Attributes attributes ) | Called when a start tag is encountered by the parser; uri is the namespace uri and localName its local name; startElement is the name of the start tag; attributes contains a list of (name, value) pairs that are found inside the startElement tag. |
| void endElement( String uri, String localName, String endElement ) | Called when an end tag is encountered by the parser. |
| void characters( char [ ] ch, int start, int length ) | Called when character data is encountered by the parser; ch stores the characters, start is the starting index, and length is the number of characters to read from ch. |

An XML document can contain entity references. An entity reference uses the special syntax &nameOfTheEntity;. For example, &quot; is an entity reference that represents a double quote ("), &lt; represents the less than sign (<), and &amp; represents the ampersand character (&). To keep things simple, we assume that the XML documents we parse in this chapter do not contain entity references.

To parse an XML document, we call one of the parse methods of the SAXParser class. TABLE 2 lists some of them. They specify a DefaultHandler parameter that is used to handle the events encountered during parsing. When we override the DefaultHandler class, we need to override its methods so that we build our list of items inside these methods.

**TABLE 2** Selected `parse` methods of the `SAXParser` class

| Method | Description |
|---|---|
| void parse( InputStream is, DefaultHandler dh ) | Parses the content of the XML input stream is and uses dh to handle events. |
| void parse( File f, DefaultHandler dh ) | Parses the content of the XML file f and uses dh to handle events. |
| void parse( String uri, DefaultHandler dh ) | Parses the content of the XML file located at uri and uses dh to handle events. |

In the class SAXHandler, version 0. We override the startElement, endElement, and characters methods, which all throw a SAXException. Inside them, we output the values of some of their parameters. The SAX related classes are in the org.xml.sax and org.xml.sax.helpers packages.

**SAXHandler.java**

```java
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import android.util.Log;

public class SAXHandler extends DefaultHandler {
    public SAXHandler( ) {
    }

    public void startElement( String uri, String localName,
                              String startElement, Attributes attributes )
            throws SAXException {
        Log.w( "MainActivity", "Inside startElement, startElement = "
                + startElement );
    }

    public void endElement( String uri, String localName,
                            String endElement ) throws SAXException {
        Log.w( "MainActivity", "Inside endElement, endElement = "
                + endElement );
    }

    public void characters( char ch [], int start,
                            int length ) throws SAXException {
        String text = new String( ___, ___, ______ );
        Log.w( "MainActivity", "Inside characters, text = " + text );
    }
}
```

The DefaultHandler class includes other methods, such as error or fatalError that are called when an error in the XML document is encountered. In order to keep things simple, we assume that there is no error of any kind in the XML and the DTD documents, so we do not deal with XML errors.

The SAXParser class is abstract so we cannot instantiate an object from it. The SAXParserFactory class, also abstract, provides methods to create a SAXParser object. TABLE 3 lists two of its methods. The static newInstance method creates a SAXParserFactory object and returns a reference to it, and the newSAXParser method creates a SAXParser object and returns a reference to it.

**TABLE 3** Selected methods of the `SAXParserFactory` class

| Method | Description |
|---|---|
| SAXParserFactory newInstance( ) | Static method that creates and returns a SAXParserFactory. |
| SAXParser newSAXParser( ) | Creates and returns a SAXParser. |

To create a SAXParser object, we can use the following sequence:

SAXParserFactory factory = SAXParserFactory.newInstance( );

SAXParser saxParser = factory.newSAXParser( );

In Version 0, we store a hard coded XML document in a file located in the raw directory of the res directory. Thus, if the name of our file is test.xml, we can access it using the resource R.raw.test. We can convert that resource to an input stream and use the first parse method listed in Table 2.

The getResources method of the Context class, inherited by the Activity class, returns a Resources reference to our application package. With that Resources reference, we can call the openRawResource method, listed in TABLE 4, in order to open and get a reference to an InputStream so that we can read the data in that resource.

**TABLE 4** The `openRawResource` method of the `Resources` class

| Method | Description |
|---|---|
| InputStream openRawResource( int id ) | Opens and returns an InputStream to read the data of the resource whose id is id. |

We can chain method calls to getResources and openRawResource to open an input stream to read data from the file test.xml as follows:

InputStream is = getResources( ).openRawResource( R.raw.test );

In order to parse an InputStream named is containing an XML document with a SAXParser named saxParser that uses a DefaultHandler reference named handler to handle the events, we call the parse method as follows:

saxParser.parse( is, handler );

In the MainActivity class, the SAXParser and SAXParserFactory classes, both part of the javax.xml.parsers package. Many exceptions can be thrown by the various methods that we use:

newSAXParser throws a ParserConfigurationException and a SAXException.

openRawResource throws a Resources.NotFoundException.

parse throws an IllegalArgumentException, an IOException, and a SAXException.

**MainActivity.java**

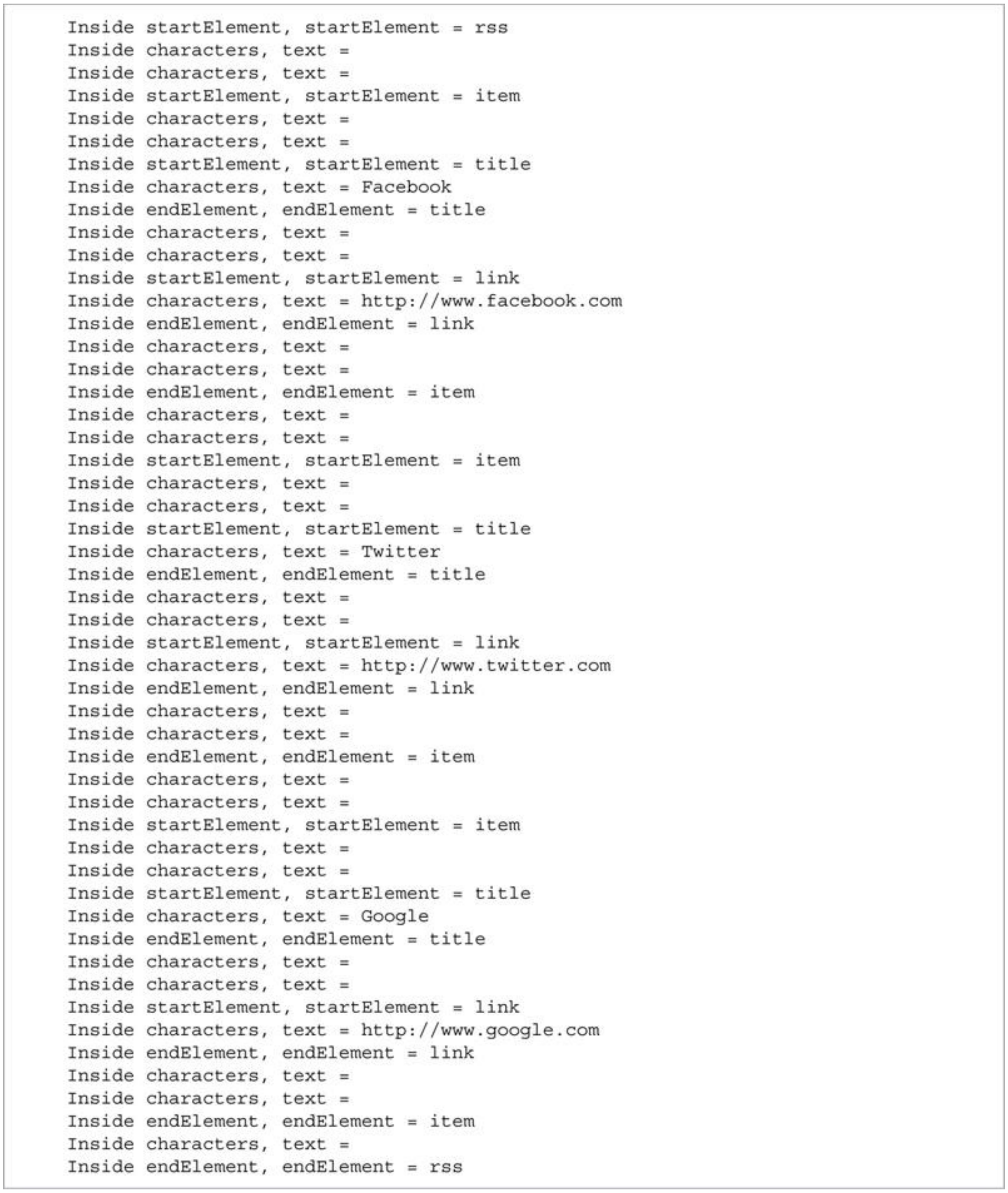
```java
import android.os.Bundle;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;
import java.io.InputStream;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class MainActivity extends AppCompatActivity {
    protected void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_main );
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance( );
            SAXParser saxParser = factory.newSAXParser( );
            SAXHandler handler = new SAXHandler( );
            InputStream is = getResources( ).openRawResource( __________ );
            saxParser.parse( ____, ______ );
        }
        catch ( Exception e ) {
            Log.w( "MainActivity", "e = " + e.toString( ) );
        }
    }
}
```

Some of these exceptions are checked and some others are unchecked. To keep things simple, we use one try block and catch a generic Exception. Inside the try block, we create a SAXParser and declare and instantiate a SAXHandler named handler. We open an input stream to read the data in the test.xml file located in the raw directory of the res directory, and start parsing that input stream with handler. The parsing triggers a number of calls to the various methods inherited from DefaultHandler by SAXHandler, in particular the three methods that we overrode in the SAXHandler class.

FIGURE 3 shows the output inside Logcat when we run the app.



```
Inside startElement, startElement = rss
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = item
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = title
Inside characters, text = Facebook
Inside endElement, endElement = title
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = link
Inside characters, text = http://www.facebook.com
Inside endElement, endElement = link
Inside characters, text =
Inside characters, text =
Inside endElement, endElement = item
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = item
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = title
Inside characters, text = Twitter
Inside endElement, endElement = title
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = link
Inside characters, text = http://www.twitter.com
Inside endElement, endElement = link
Inside characters, text =
Inside characters, text =
Inside endElement, endElement = item
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = item
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = title
Inside characters, text = Google
Inside endElement, endElement = title
Inside characters, text =
Inside characters, text =
Inside startElement, startElement = link
Inside characters, text = http://www.google.com
Inside endElement, endElement = link
Inside characters, text =
Inside characters, text =
Inside endElement, endElement = item
Inside characters, text =
Inside endElement, endElement = rss
```

Figure 3 Logcat output of the Web Content app, version 0

We observe the following:

- Whenever a start tag is encountered, we execute inside the `startElement` method, and the value of the `startElement` parameter is the name of the tag.
- Whenever an end tag is encountered, we execute inside the `endElement` method, and the value of the `endElement` parameter is the name of the tag.
- Between the start and end tags of the same element, we execute inside the `characters` method and the array `ch` stores the characters between the start and end tags (also called the element contents).
- Between the end tag of an element and the start tag of another element, we execute inside the `characters` method and the text only contains white space characters.

## Parsing XML into a List, Web Content App, Version 1

In Version 1, we parse the test.xml file and convert it to a list. We code the Item class as part of our Model. The Item class encapsulates an item in the XML document: it has two String instance variables, title and link. In the SAXHandler class, we build an ArrayList of Item objects that reflect what we find in the XML document.

### Item.java

```java
public class Item {
    private String title;
    private String link;

    public Item( String newTitle, String newLink ) {
        setTitle( newTitle );
        setLink( newLink );
    }

    public void setTitle( String newTitle ) { title = newTitle; }
    public void setLink( String newLink ) { link = newLink; }
    public String getTitle( ) {
        return title;
    }
    public String getLink( ) {
        return link;
    }
    public String toString( ) {
        return title + "; " + link;
    }
}
```

In the SAXHandler class, Version 1, we build an ArrayList of Item objects.

**SAXHandler.java**

```
import java.util.ArrayList;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXHandler extends DefaultHandler {
    private boolean validText;
    private String element = "";
    private Item currentItem;
    private ArrayList<Item> items;

    public SAXHandler( ) {
        validText = false;
        items = new ArrayList<Item>( );
    }

    public ArrayList<Item> getItems( ) { return items; }

    public void startElement( String uri, String localName,
                              String startElement, Attributes attributes )
            throws SAXException {
        validText = true;
        element = startElement;
        if( startElement.equals( ____________ ) ) // start current item
            currentItem = new Item( "", "" );
    }

    public void endElement( String uri, String localName,
                            String endElement ) throws SAXException {
        validText = false;
        if( endElement.equals( __________ ) ) // add current item to items
            items.add( _____________ );
    }

    public void characters( char ch [], int start,
                            int length ) throws SAXException {
        if( currentItem != null && element.equals( ______" ) && validText )
            currentItem.setTitle( new String( ch, start, length ) );
        else if( currentItem != null && element.equals( _______ ) && validText )
            currentItem.setLink( new String( _____, _____, length ) );
    }
}
```

When the startElement method is called, if the tag is item, we need to instantiate a new Item object. If the tag is title or link, the text that we will read inside the characters method is the value for the title or link instance variable of the current Item object. When the endElement method is called, if the tag is item, we are done processing the current Item object and we need to add it to the list.

We need to be careful that inside the characters method, we process the correct data. There are white spaces that are typically present between tags, for example between an end tag and a start tag, in an XML document. We must be careful not to process them. The character data that we want to process is found between a start tag and the corresponding end tag. Thus, we use a state variable, validText, to assess if the character data that we process inside characters is the data that we want to process or not. When we encounter a start tag, we set validText to true. When we encounter an end tag, we set validText to false. We only process the character data in the characters method if validText is true.

We declare four instance variables:

- `validText`, a `boolean` state variable. If its value is `true`, we process character data inside the `characters` method, otherwise we do not.
- `element`, a `String`, stores the current element (`xml`, `rss`, `item`, `title`, or `link` in this example).
- `currentItem`, an `Item`, stores the current `Item` object.
- `items`, an `ArrayList`, stores the list of `Item` objects that we build.

The constructor initializes `validText` to `false` and instantiates `items`. We code an accessor to `items` so we can access it from an outside class.

In the `startElement` method, we set `validText` to `true` and assign `startElement`, the text inside the start tag, to `element`. If the start tag is equal to `item` , we instantiate a new `Item` object and assign its reference to `currentItem`.

If there is an end tag following the start tag, the `characters` method is called next. If `currentItem` is not `null` and `validText` is `true` and the value of `element` is either `title` or `link`, we are in the middle of building the current `Item` object. If `element` is equal to `title` or `link`, we set the value of the `title` or `link` instance variable of the current `Item` object to the characters read.

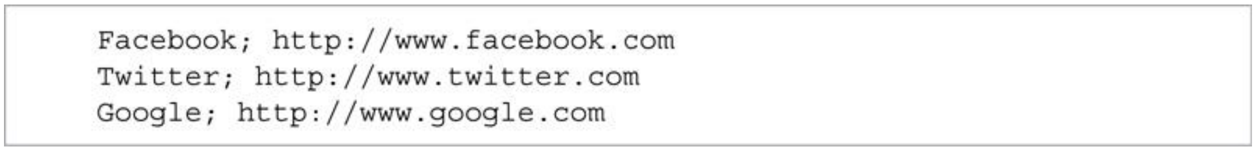
After the `startElement` and `characters` methods are called, the `endElement` method is called. We assign `false` to `validText`. In this way, any text found after an end tag and before a start tag is not processed by the `characters` method. If the value of element is `item` , we are done processing the current `Item` object, and we add it to `items`.

In the main class, Version 1. We retrieve the list of items that we parsed and assign it to the ArrayList items. We loop through it and output it to Logcat in order to check the correctness of the parsing.

**MainActivity.java**

```
saxParser.parse( is, handler );
    ArrayList<Item> items = handler.____________________________;
    for( Item ____________ )
        Log.w( "MainActivity", ________________ );
}
catch ( Exception e ) {
    Log.w( "MainActivity", "e = " + e.toString( ) );
}
```

Figure 4 shows the output inside Logcat when we run the app, Version 1. We can check that the ArrayList items contains the three Item objects that are listed in the test.xml file.



```
Facebook; http://www.facebook.com
Twitter; http://www.twitter.com
Google; http://www.google.com
```

Figure 4 Logcat output of the Web Content app, Version 1

## Parsing a Remote XML Document, Web Content App, Version 2

In Version 2, we parse an XML document from a remote website. Instead of reading data from the test.xml file, we pull data from the rss feed for the NASA blog.

The URL is https://www.nasa.gov/rss/dyn/educationnews.rss. We access the Internet, open that URL, and read the data at that URL. Starting with Version 3.0, Android does not allow an app to open a URL in its main thread. A thread is a sequence of code that executes inside an existing process. The existing process is often called the main thread. For an Android app, it is also called the user-interface thread. There can be several threads executing inside the same process. Threads can share resources, such as memory.

When we start the app, our code executes in the main thread, so we need to open that URL in a different thread. Furthermore, we need the XML parsing in the secondary thread to finish before trying to fill the list with data in the main thread because we use the data read in the secondary thread for the list.

A Looper manages tasks that a Thread will run. It puts them in a queue and then the Thread takes the next task in line. A Looper is tied to a specific Thread.

An Executor encapsulates managing and distributing tasks to different Threads. If you have a fixed threadpool size of 1 then it would be similar in design to a Looper because it will just queue up the work for that one Thread. If you have a threadpool with size > 1 then it will manage giving the task to the next Thread available to do the work, or in other words it will distribute tasks among all threads.

Executors are more flexible. For Android, the only time you really use Looper is when trying to make a Handler to communicate with the main thread from a background thread (which could even be in an ExecutorService).

Handler mainThreadHandler = new Handler(Looper.getMainLooper());

mainThreadHandler.post(new Runnable...); //runs on main thread

Since our app accesses the Internet, we need to add the appropriate permission code inside the manifest element of the AndroidManifest.xml file as follows:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.example.xml">
    <uses-permission android:name="android.permission.INTERNET"/>

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

    <application
        android:allowBackup="true"
```

**MainActivity.java**

```
private final String URL = "https://www.nasa.gov/rss/dyn/educationnews.rss";
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

ExecutorService executor = Executors.newSingleThreadExecutor();
Handler handler = new Handler(Looper.getMainLooper());
executor.execute(new Runnable() {
    @Override
    public void run() {
        //Do background here
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            SAXHandler handler = new SAXHandler();
            saxParser.parse(____,_______);
            Items = _______.getItems();
        } catch (Exception e) {
            Log.w("MainActivity", e.toString());

        }
        handler.post(new Runnable() {
            @Override
            public void run() {
               //Call displayList
               __________________________
            }
        });
    }
});

}//end onCreate

public void displayList (ArrayList < Item > items)
{    if (items != null) {
    for (Item _______________)
```

```
        Log.w("MainActivity", item.toString());
}
```

Run the app. You will see the following partial output inside the Logcat.

```
gov/press-release/la-nasa-desaf-a-a-estudiantes-a-dise-ar-robots-para-excavar-en-la-
luna, NASA Challenges Students to Design Moon-Digging Robots;
http://www.nasa.gov/press-release/nasa-challenges-students-to-design-moon-digging-
robots, NASA Announces Winners of Deep Space Food Challenge;
http://www.nasa.gov/press-release/nasa-announces-winners-of-deep-space-food-
challenge, NASA Administrator to Meet with Florida STEM Students;
http://www.nasa.gov/press-release/nasa-administrator-to-meet-with-florida-stem-
students, Nebraska Youth to Hear from NASA Astronauts Aboard Space Station;
http://www.nasa.gov/press-release/nebraska-youth-to-hear-from-nasa-astronauts-aboard-
space-station, NASA Invites Students, Educators to Join Artemis I Mission;
http://www.nasa.gov/press-release/nasa-invites-students-educators-to-join-artemis-i-
mission, NASA Highlights Minority Serving Institution Capabilities to Increase
Diversity in STEM Workforce; http://www.nasa.gov/feature/langley/nasa-highlights-
minority-serving-institution-capabilities-to-increase-diversity-in-stem, Katherine
Johnson's STEM Contributions Marked on her 103rd Birthday;
http://www.nasa.gov/feature/langley/katherine-johnson-s-stem-contributions-marked-on-
her-103rd-birthday]
```

## Displaying the Results in a ListView, Web Content App, Version 3

In Versions 0, 1, and 2, we do not have any user interface. In Version 3, we display a list of all the titles retrieved from the XML document on the screen.

In order to keep this example simple, we only allow our app to run in vertical position. Thus, we add the following to the AndroidManifest.xml file inside the activity element:

**android:screenOrientation="portrait"**

To display the titles as a list on the screen, we use a ListView element in the activity_main.xml file. A ListView contains an arbitrary number of Strings. We do not have to specify how many elements when we define the ListView. Thus, we can build an ArrayList of Item objects dynamically and place the corresponding list of titles inside the ListView programmatically.

In the activity_main.xml file, Version 3. We replace the TextView that displays Hello World! in the skeleton file with a ListView element. We give it an id so that we can access it by code using the findViewById method and populate it with data. We use the android:divider and the android:dividerHeight attributes to add a red, 2 pixels thick dividing line between the items displayed.

```
<?xml version="1.0" encoding="UTF-8"?>

    <RelativeLayout tools:context="com.android.example.xml.MainActivity"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
```

```
android:layout_height="match_parent" android:layout_width="match_parent"
xmlns:tools="http://schemas.android.com/tools"
xmlns:android="http://schemas.android.com/apk/res/android">

    <ListView android:layout_height="match_parent"
android:layout_width="match_parent" android:dividerHeight="2dp"
android:divider="#FF00" android:id="@+id/list_view"/>

</RelativeLayout>
```

```
MainActivity.java

private ___________________;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    listView = (ListView) findViewById( _________________ );
…..
}

public void displayList (ArrayList < Item > items)
{
    if (items != null) {
        // Build ArrayList of titles to display
        ArrayList<String> titles = new ArrayList<String>( );
        for( Item __________________ )
            titles.add( item.________________ );

        ArrayAdapter<String> adapter = new ArrayAdapter<String>( this,
                android.R.layout.simple_list_item_1, _________________ );
        listView._______________( ___________ );

    } else
        Toast.makeText( this, "Sorry - No data found",
                Toast.LENGTH_LONG ).show( );
}
```

In the MainActivity class, Version 3. We include a ListView instance variable, listView.Inside the displayList method, we populate listView with all the titles from the ArrayList items. We generate titles, an ArrayList of Strings containing the titles in items. We create an ArrayAdapter containing titles. We use the constructor listed in TABLE 5, passing three arguments.

**TABLE 5** Selected constructor of the `ArrayAdapter` class

| Constructor | Description |
|---|---|
| ArrayAdapter( Context context, int textViewResourceId, List<T> objects ) | Context is the current context; textViewResourceId is the id of a layout resource containing a TextView; objects is a list of objects of type T. |

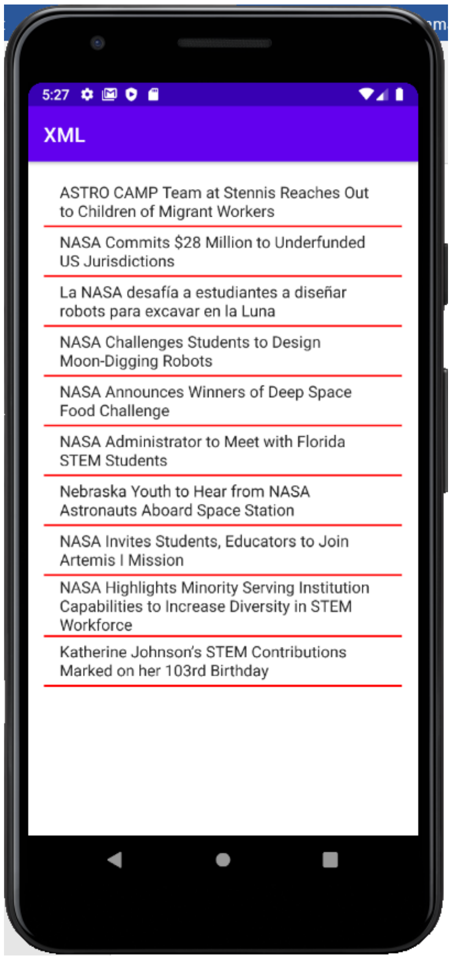
The first argument is this, a reference to this activity (Activity inherits from Context). The second argument is android.R.layout.simple_list_item_1, a constant of the R.layout class. It is a simple layout that we can use to display some text. The third argument is titles, an ArrayList and therefore a List.

We retrieve the ListView defined in activity_main.xml using its id and assign it to listView. We assign the list of titles as the list to display inside listView, calling the setAdapter method of the ListView class, shown in TABLE 6. The adapter reference is actually an ArrayAdapter and therefore a ListAdapter because ArrayAdapter inherits from the ListAdapter interface. ListAdapter is a bridge between a ListView and its list of data.

**TABLE 6** The `setAdapter` method of the `ListView` class

| Method | Description |
|---|---|
| void setAdapter( ListAdapter adapter ) | Sets the ListAdapter of this ListView to adapter; adapter stores the list of data backing this ListView and produces a View for each item in the list of data. |

**Run the app.**



ASTRO CAMP Team at Stennis Reaches Out to Children of Migrant Workers
NASA Commits $28 Million to Underfunded US Jurisdictions
La NASA desafía a estudiantes a diseñar robots para excavar en la Luna
NASA Challenges Students to Design Moon-Digging Robots
NASA Announces Winners of Deep Space Food Challenge
NASA Administrator to Meet with Florida STEM Students
Nebraska Youth to Hear from NASA Astronauts Aboard Space Station
NASA Invites Students, Educators to Join Artemis I Mission
NASA Highlights Minority Serving Institution Capabilities to Increase Diversity in STEM Workforce
Katherine Johnson's STEM Contributions Marked on her 103rd Birthday

## Opening a Web Browser Inside the App, Web Content App, Version 4

In Version 4, we open the browser at the URL chosen by the user from the list. In order to do that, we do the following:

- Implement event handling when the user selects an item from the list.
- Retrieve the link associated with that item.
- Open the browser at the URL contained in the link.

**MainActivity.java**

```
private _______________ listItems;

public void displayList (ArrayList < Item > items)
{
     _______________=items;
    if (items != null) {
      ………………….
```

```
        listView.______( ______ );
        ListItemHandler lih = new ListItemHandler( );
        listView._______ClickListener( ______ );
    } else
        Toast.makeText( this, "Sorry - No data found",
                Toast.LENGTH_LONG ).show( );
}

private class ListItemHandler
        implements AdapterView.OnItemClickListener {
    public void _______(AdapterView<?> parent, View view,
                        int position, long id ) {
        Item selectedItem = listItems.get( ________ );
       Uri uri = Uri.parse( selectedItem._________________ );
       Intent browserIntent = new Intent( _________________, ________________ );
        startActivity( __________________ );

    }
}
```

In Version 3, our list shows the titles of the ArrayList of Item objects that we generated parsing the XML document. In Version 4, we actually need to access the link values. In order to retrieve the link for a given Item object, we add an instance variable to the MainActivity class that represents the list of Item objects. When the user selects an item from the list, we retrieve its index, retrieve the Item object from the ArrayList at that index, and retrieve the value of the link instance variable of that Item.

In the MainActivity class, Version 4, we declare the listItems instance variable, an ArrayList of Item objects. Inside displayList, we assign its items parameter to listItems.

OnItemClickListener is a public inner interface of the AdapterView class. It includes one abstract callback method, onItemClick. That method is automatically called when the user selects an item from a list associated with an AdapterView, provided that an OnItemClickListener object is listening for events on that list. TABLE 7 shows the onItemClick method.

**TABLE 7** The `OnItemClickListener` interface

| Method | Description |
|---|---|
| void onItemClick( AdapterView<?> parent, View view, int position, int id ) | This method is called when an item is selected in a list: parent is the AdapterView; view is the view within the AdapterView that was selected; position is the position of view within the AdapterView; id is the row index of the item selected. |

In order to implement list event handling, we do the following:

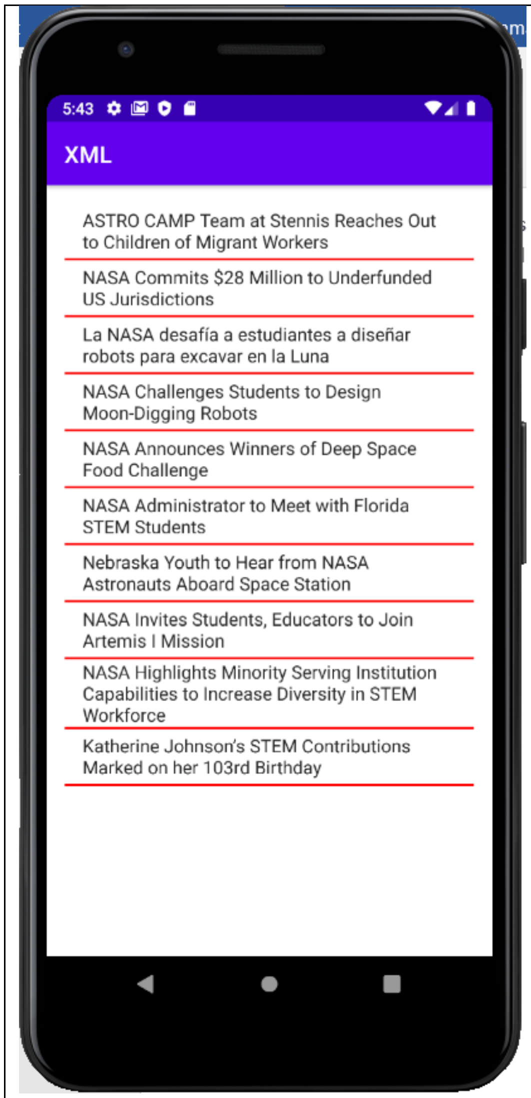
- We code a private handler class that extends OnItemClickListener, and override its only abstract method, onItemClick.
- We declare and instantiate an object of that class .
- We register that handler on the ListView so that it listens to list events.

The parameter position of the onItemClick method stores the index of the item in the list that the user selected. We use it to retrieve the Item object in listItems at that index at line 51. At line 52, we use the static parse method of the Uri class, shown in TABLE 8, to create a Uri object with the String stored in the link instance variable of that Item object. We create an Intent to open the web browser at that Uri. We pass the ACTION_VIEW constant of the Intent class and the Uri object to the Intent constructor. We start a new activity with that Intent. Since the activity is to open the browser, there is no layout file and class associated with that activity.

**TABLE 8** The `parse` method of the `Uri` class

| Method | Description |
|---|---|
| static Uri parse( String uriString ) | Creates a Uri that parses uriString and returns a reference to it. |

Run the app. After the user selects an item from the list. If we click on the back button of the emulator, we go back to the list of links, which then becomes the activity at the top of the activity stack.



5:43

XML

ASTRO CAMP Team at Stennis Reaches Out to Children of Migrant Workers

NASA Commits $28 Million to Underfunded US Jurisdictions

La NASA desafía a estudiantes a diseñar robots para excavar en la Luna

NASA Challenges Students to Design Moon-Digging Robots

NASA Announces Winners of Deep Space Food Challenge

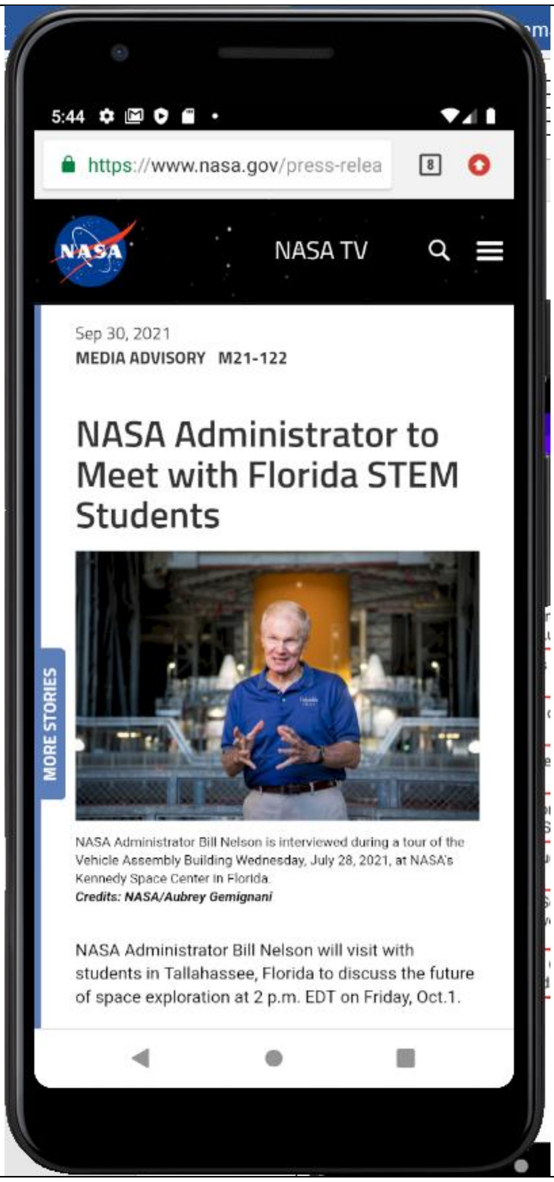
NASA Administrator to Meet with Florida STEM Students

Nebraska Youth to Hear from NASA Astronauts Aboard Space Station

NASA Invites Students, Educators to Join Artemis I Mission

NASA Highlights Minority Serving Institution Capabilities to Increase Diversity in STEM Workforce

Katherine Johnson's STEM Contributions Marked on her 103rd Birthday



5:44

https://www.nasa.gov/press-relea

NASA TV

MORE STORIES

Sep 30, 2021
MEDIA ADVISORY M21-122

NASA Administrator to Meet with Florida STEM Students

NASA Administrator Bill Nelson is interviewed during a tour of the Vehicle Assembly Building Wednesday, July 28, 2021, at NASA's Kennedy Space Center in Florida.
*Credits: NASA/Aubrey Gemignani*

NASA Administrator Bill Nelson will visit with students in Tallahassee, Florida to discuss the future of space exploration at 2 p.m. EDT on Friday, Oct.1.

**Grading**:

1. Submit the project in zip file (extension zip)
2. A pdf document that contains the content of the following files: MainActivity class, activity_main.xml, and AndroidManifest.xml.
3. A zoom link or YouTube that demonstrate the app (less than 5 minutes). Write the link on the comment section in the dropbox.
   Your video should demonstrate the features of the app.

<u>Note</u>:
App is running correctly and met all the requirements is 40 points, but
Incomplete  item 1(-40 points)
Incomplete item 2 (-10 points)
Incomplete item 3 (-20 points)