MOBILE COMMUNICATIONS AND PROGRAMMING

COURSEWORK 2 - ANDROID

December 3, 2017

Jason Shawn D' Souza
Heriot Watt ID: H00202543
Heriot Watt University
Mobile Communications and Programming

Contents

Introduction 2
Implementation 2
Enhancements 12
Conclusion 20
References 21

INTRODUCTION

For this coursework we were asked to enhance the functionality of the quiz app developed during the lectures. The maths part of the app is functional and we needed to implement the history part of the application. After implementing the history part we need the application to read questions from an XML file. **SAX parser** was used to parse the xml and make the app functional.

IMPLEMENTATION

Quiz Activity

```
public void onClick(View arg0)
{
    if ( arg0 == historyButton ) {
        Log.i("QUIZ", "History selected");
        Toast.makeText(getApplicationContext(), "History button pressed", Toast.LENGTH_LONG).show();
    }
    if ( arg0 == mathsButton ) {
        Log.i("QUIZ", "Maths selected");
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.NumericQuestion.class));
    }
}
```

(a) Initial provided code

```
public void onClick(View arg0)
{
    if ( arg0 == historyButton ) {
        Log.i("QUIZ", "History selected");
        Toast.makeText(getApplicationContext(), "History button pressed", Toast.LENGTH_LONG).show();
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.MultiChoiceQuestionsActivity.class));/*To open history*/
    }
    if ( arg0 == mathsButton ) {
        Log.i("QUIZ", "Maths selected");
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.NumericQuestion.class));
    }
}
```

(b) Edited code

Figure 1: Showing difference between initial and final code

Initially, Figure 1 shows the changes made to the code:
startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.MultiChoiceQuestionsActivity.class));
This line is added so as to bring up the activity for multiple choices (for the history screen).

Android Manifest

After the changes were made in the QuizActivity, for the history part to work a change needs to be made in the AndroidManifest.xml

```
<activity
    android:name=".MultiChoiceQuestionsActivity"
    ></activity>
```

Figure 2: Editing the AndroidManifest.xml

Figure 2 shows that we add the activity to the manifest file.

MultiChoiceQuestions Activity

After this the history part of the application works but the summary isnt functional. To fix the issues the following changes are done :

```
public class MultiChoiceQuestionsActivity extends Activity
implements OnClickListener{
    private QuizApplication application;//Calling QuizApp obj

    private TextView question;
    private TextView summary;
    private Vector<MCQuestion> questions;
    private RadioGroup opts;
    private MCQuestion currentQuestion;
    private int noCorrect;
    private int noAnswered;
    private int noQuestions = 3;
```

(a) Creating object of QuizApplication

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.multichoice);
    Log.i("MCQ", "MultiChoice starts");
    opts = (RadioGroup) findViewById(R.id.opts);
    Button sub = (Button) findViewById(R.id.submit);
    application = (QuizApplication) this.getApplication();//calling this session as the q
    question = (TextView) findViewById(R.id.question_text);
    summary = (TextView) findViewById(R.id.multichoice_summary);
    noCorrect = 0;
    noAnswered = 0;
    sub.setOnClickListener(this);
    initialiseQuestions();
    generateSummary();
    generateQuestion(0);
}
```

(b) Initializing

```
public void onClick(View arg0) {
    noAnswered++;
    application.incrementQuestionsAttempted();
    int id;
    id = opts.getCheckedRadioButtonId();
    if( currentQuestion.isCorrect(id)){
        //this if loop was added so that the summary is proper
        //increment number of correct questions
        noCorrect++;
        application.incrementQuestionsCorrect();

    }
    generateSummary();
    if (noAnswered == noQuestions)
        finish();
    else
        generateQuestion(noAnswered);
}
```

(c) Changes made in the onClick() method

Figure 3: Changes made in MultiChoiceQuestionsActivity.Java

The XML file

The XML file used to test was called testing.xml and was included in the assets folder of the project:

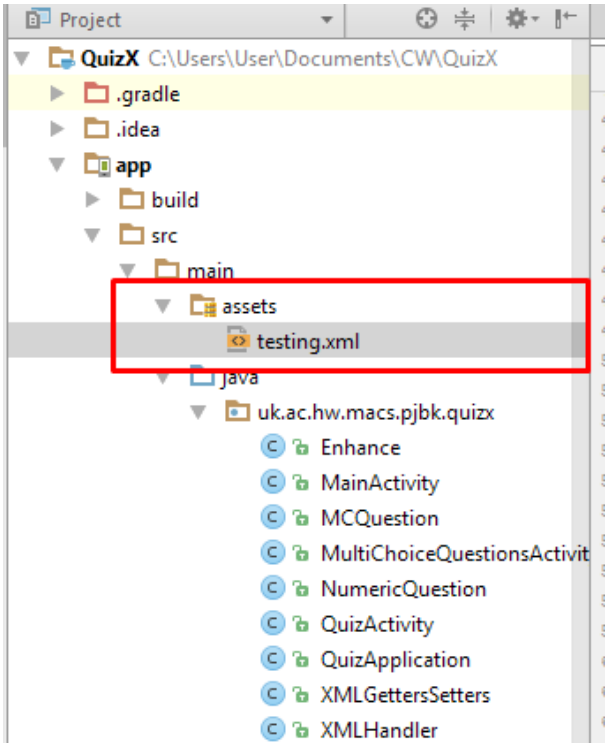

Figure 4: The XML in the assets folder

IT IS NECESSARY TO HAVE THE XML FILE IN THE ASSETS FOLDER.
The contents of the XML file are as follows:

```
<questions title="History">
<question>
<ask>Who won the Battle of Hastings?</ask>
<answer correct="true">William of Normandy</answer>
<answer>Harold Godwinson</answer>
<answer>Edward the Confessor</answer>
<answer>Harald Hadrada</answer>
</question>
<question>
<ask>Who won the Battle of Waterloo?</ask>
<answer correct="true">Wellington</answer>
<answer>Napoleon</answer>
<answer>Blucher</answer>
<answer>Marlborough</answer>
</question>
<question>
<ask>Who was the first Roman emperor?</ask>
<answer correct="true">Augustus</answer>
<answer>Julius Caesar</answer>
```

```
<answer>Nero</answer>
<answer>Trajan</answer>
</question>
<question>
<ask>Which Roman emperor built a wall across Scotland?</ask>
<answer correct="true">Antonius</answer>
<answer>Hadrian</answer>
<answer>Trajan</answer>
<answer>Constantine</answer>
</question>
</questions>
```

Implementing SAX parser

First implementation for this is the XMLSetterGetter class. From the name, it sets and gets relevant data from the XML file: (Similar to MCQuestion class)

Code is as follows :

```
package uk.ac.hw.macs.pjbk.quizx;

import android.util.Log;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by User on 12/1/2017.
 */

public class XMLGettersSetters {


 private int correctAnswer = -1;
 String ask = new String();
 ArrayList < String > answer = new ArrayList < String > ();


 public XMLGettersSetters(String q) {
 \**contains question and arraylist of answers/options */
  ask = q;
  answer = new ArrayList < String > ();
 }
```

```
 public void addAnswer(String s, boolean correct) {
  answer.add(s);
  if (correct)
   correctAnswer = answer.size();
 }

 public String getQuestions() {
  return ask;
 }

 public void setQuestions(String ask) {
  this.ask = ask;
  Log.i("This␣is␣the␣question:", ask);
 }

 public ArrayList < String > getAnswer() {
  return answer;
 }

 public void setAnswers(String answer) {
  this.answer.add(answer);
  Log.i("This␣is␣the␣option:", answer);
 }

 public boolean isCorrect(int i) {
  return i == correctAnswer;
 }

 public int getCorrect() {
  return correctAnswer;
 }

 public void setCorrectAnswer(int correctAnswer) {
  this.correctAnswer = correctAnswer;

 }

}
```

This is self - explanatory. Next comes the XMLHandler class:(Code below this line)

```
package uk.ac.hw.macs.pjbk.quizx;
```

```
/**
 * Created by User on 12/1/2017.
 */

import android.util.Xml;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import static android.R.attr.x;

public class XMLHandler extends DefaultHandler {


    int currentDepth = 0;
    Boolean currentElement = false;
    String currentValue = "";
    XMLGettersSetters xmltest = null;
    boolean correct;
    private ArrayList<XMLGettersSetters> itemsList = new ArrayList
    <XMLGettersSetters>();/*Creating arraylist of xmlgetterssetters*/

    public ArrayList<XMLGettersSetters> getItemsList() {
        return itemsList;
    }/*Return the private list*/

    // Called when tag starts
    @Override
    public void startElement(String uri, String localName, String qName,
                             Attributes attributes) throws SAXException {
/*Elements at the start of each tag
*For the history file it is question*/
        currentElement = true;
        currentValue = "";
```

```
        currentDepth = 0;

        if (localName.equals("question")) {
            xmltest = new XMLGettersSetters("");
        }
        int length = attributes.getLength();
        // process each attribute
        for (int i=0; i<length; i++) {
// get qualified (prefixed) name by index




// get attribute's value by index.
            String value = attributes.getValue(i);//getting the value as "true"

            if (value.equals("true")){
                xmltest.isCorrect(currentDepth);
            }




        }
    }
    // Called when tag closing

    public void endElement(String uri, String localName, String qName)
            throws SAXException {

        currentElement = false;

        /** set value */

        if (localName.equalsIgnoreCase("ask"))
            xmltest.setQuestions(currentValue);
        else if (localName.equalsIgnoreCase("answer"))
            xmltest.setAnswers(currentValue);
        else if (localName.equalsIgnoreCase("question"))
        //checks for ending tag for next question or end of questions
```

```
            itemsList.add(xmltest);
    }

    // Called to get tag characters
    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {

        if (currentElement) {
            currentValue = currentValue + new String(ch, start, length);
        }

    }


}
```

Changes to MultiChoiceQuestions

Now after setting up the classes to get and set XML values and handling XML, changes had to be made in the MultiChoiceQuestions class:

```
public class MultiChoiceQuestionsActivity extends Activity
implements OnClickListener {
    private QuizApplication application;//Caling QuizApp obj

    private TextView question;
    private TextView summary;
    private Vector<MCQuestion> questions;
    private Vector<XMLGettersSetters> questions2;
    private Button sub;
    private Button hint;
    private RadioGroup opts;
    private MCQuestion currentQuestion;
    private XMLGettersSetters currentq;
    private int noCorrect;
    private int noAnswered;
    private int noQuestions = 4;
    private ClassLoader context;
```

Figure 5: Setting up a vector of XMLGettersSetters and an XMLGetterSetter

After creating a vector as shown in Figure 5, we then initialise this vector and make the

necessary changes to the initialisequestions() method:
(Code below this line)

```
private void initialiseQuestions() {

        questions2 = new Vector<XMLGettersSetters>();
        String parsedData = "";

        try {

            Log.w("AndroidParseXMLActivity", "Start");
            /** Handling XML */

            SAXParserFactory spf = SAXParserFactory.newInstance();
            SAXParser sp = spf.newSAXParser();
            XMLReader xr = sp.getXMLReader();

            XMLHandler myXMLHandler = new XMLHandler();
            xr.setContentHandler(myXMLHandler);
            InputSource inStream = new InputSource();

            Log.w("AndroidParseXMLActivity", "Parse1");
            InputStreamReader xmlFile = new InputStreamReader(getAssets().
            open("testing.xml"));
            inStream.setCharacterStream(new StringReader(xmlFile.toString()));
            Log.w("AndroidParseXMLActivity", "Parse2");

            xr.parse(new InputSource(getAssets().open("testing.xml")));
            Log.w("AndroidParseXMLActivity", "Parse3");

            ArrayList<XMLGettersSetters> itemsList = myXMLHandler.getItemsList();

            for (int i = 0; i < itemsList.size(); i++) {
                XMLGettersSetters item = itemsList.get(i);

                parsedData = parsedData + "----->\n";

                parsedData = parsedData + "Question: "+ item.getQuestions()+"\n";
                parsedData = parsedData + "Option: "+item.getAnswer()+ "\n";
                //questions2.add(itemsList);


                XMLGettersSetters q2 = new XMLGettersSetters(item.getQuestions());
                q2.addAnswer(item.getAnswer().get(0).toString()
```

```
                , item.isCorrect(-1));\\correct= true xml
                q2.addAnswer(item.getAnswer().get(1).toString()
                , item.isCorrect(0));
                q2.addAnswer(item.getAnswer().get(2).toString()
                , item.isCorrect(0));
                q2.addAnswer(item.getAnswer().get(3).toString()
                , item.isCorrect(0));
                questions2.add(q2);
            }
            Log.w("AndroidParseXMLActivity", "Done");
        } catch (Exception e) {
            Log.w("AndroidParseXMLActivity", e);
        }


    }
```

The generateQuestions method is then modified to allow for the xml components to have radio buttons and functionality like the previous implementation :(Code below this line)

```
private void generateQuestion(int noQ) {
        opts.removeAllViews();
        currentq = questions2.get(noQ);
        question.setText(currentq.getQuestions());
        int i = 1;
        for (String ans : currentq.getAnswer()) {
            RadioButton rb = new RadioButton(this);
            rb.setText(ans);
            rb.setId(i++);
            opts.addView(rb);
        }

    }
```

ENHANCEMENTS

Two enhancements were implemented:

- An implementation of multiple choice Science questions (Based on the history multiplechoice without xml implementation)
- An Hint feature (hints to answers for the user)

Science Questions

```
private QuizApplication application;
private Button historyButton;
private Button mathsButton;
private TextView summary;
private Button scienceButton;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    application = (QuizApplication) this.getApplication();

    historyButton = (Button) findViewById(R.id.history_button);
    historyButton.setOnClickListener(this);
    mathsButton = (Button) findViewById(R.id.maths_button);
    mathsButton.setOnClickListener(this);
    scienceButton = (Button) findViewById(R.id.science_button);
    scienceButton.setOnClickListener(this);
    summary = (TextView) findViewById(R.id.overall_summary);
    generateSummary();
}
```

(a) Initializing the button

```
public void onClick(View arg0)
{
    if ( arg0 == historyButton ) {
        Log.i("QUIZ", "History selected");
        Toast.makeText(getApplicationContext(), "History button pressed", Toast.LENGTH_LONG).show();
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.MultiChoiceQuestionsActivity.class));/*To open history*/
    }
    if ( arg0 == mathsButton ) {
        Log.i("QUIZ", "Maths selected");
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.NumericQuestion.class));
    }
    if ( arg0 == scienceButton ) {
        Log.i("QUIZ", "Science selected");
        Toast.makeText(getApplicationContext(), "Science button pressed", Toast.LENGTH_LONG).show();
        startActivity(new Intent(this, uk.ac.hw.macs.pjbk.quizx.Enhance.class));
    }
}
```

(b) Setting up on click event

Figure 6: Changes made to implement science questions

Figure 6 shows the steps taken to implement the button on the main screen.

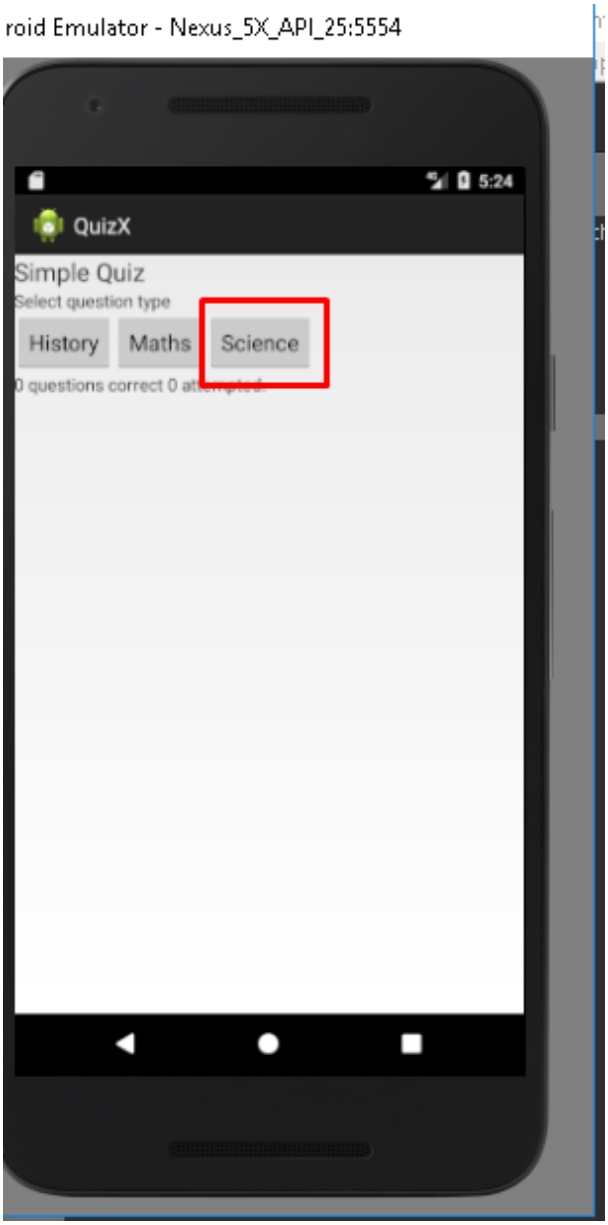

Figure 7: The main screen with the science button

The implementation of the science questions is done in the same way as the history questions was done before:(Code below this line)

```
package uk.ac.hw.macs.pjbk.quizx;

/**
 * Created by User on 12/2/2017.
 */
import java.util.Vector;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

public class Enhance extends  Activity
        implements OnClickListener{
    //Previous implementation of History just modified to
    //include science questions
    private TextView question;
    private TextView summary;
    private Vector<MCQuestion> questions;
    private RadioGroup opts;
    private MCQuestion currentQuestion;
    private int noCorrect;
    private int noAnswered;
    private int noQuestions = 3;
    private QuizApplication application;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.multichoice);
        Log.i("MCQ", "MultiChoice starts");
        application = (QuizApplication) this.getApplication();
        opts = (RadioGroup) findViewById(R.id.opts);
        Button sub = (Button) findViewById(R.id.submit);
        question = (TextView) findViewById(R.id.question_text);
        summary = (TextView) findViewById(R.id.multichoice_summary);
        noCorrect = 0;
        noAnswered = 0;
        sub.setOnClickListener(this);
        initialiseQuestions();
        generateSummary();
        generateQuestion(0);
    }
```

```
private void generateQuestion(int noQ) {
    opts.removeAllViews();
    currentQuestion = questions.get(noQ);
    question.setText(currentQuestion.getQuestion());
    int i = 1;
    for( String ans: currentQuestion.getAnswers()) {
        RadioButton rb = new RadioButton(this);
        rb.setText(ans);
        rb.setId(i++);
        opts.addView(rb);
    }
}

private void initialiseQuestions() {
    questions = new Vector<MCQuestion>();
    MCQuestion q1 = new MCQuestion("Brass gets discoloured in air because
    of the presence of which of the following gases in air?");
    q1.addAnswer("Oxygen", false);
    q1.addAnswer("Hydrogen Sulphide", true);
    q1.addAnswer("Carbon dioxide", false);
    q1.addAnswer("Nitrogen", false);
    questions.add(q1);
    q1 = new MCQuestion("\t\n" +
            "Which of the following is used in pencils?");
    q1.addAnswer("Silicon", false);
    q1.addAnswer("Graphite", true);
    q1.addAnswer("Lead",false);
    q1.addAnswer("Charcoal", false);
    questions.add(q1);
    q1 = new MCQuestion("Which of the following metals forms an
    amalgam with other metals?");
    q1.addAnswer("Tin", false);
    q1.addAnswer("Mercury", true);
    q1.addAnswer("Lead", false);
    q1.addAnswer("Zinc", false);
    questions.add(q1);
    // more questions could be added
}

private void generateSummary()
{
    summary.setText( noCorrect + " answers out of " + noAnswered
            + " attempted of " +noQuestions + " questions");
```

```
    }

    public void onClick(View arg0) {
        noAnswered++;
        application.incrementQuestionsAttempted();
        int id;
        id = opts.getCheckedRadioButtonId();
        if( currentQuestion.isCorrect(id)){
            System.out.println(noCorrect);
            noCorrect++;
            application.incrementQuestionsCorrect();
        }
        generateSummary();
        if (noAnswered == noQuestions)
            finish();
        else
            generateQuestion(noAnswered);
    }

}
```

The only changes made are fixing the summary and the contents of the questions to suit the topic of Science.

Hint

This implementation was easy and was done in the following way :

```
private TextView question;
private TextView summary;
private Vector<MCQuestion> questions;
private Vector<XMLGettersSetters> questions2;
private Button sub;
private Button hint;
private RadioGroup opts;
private MCQuestion currentQuestion;
private XMLGettersSetters currentq;
private int noCorrect;
private int noAnswered;
private int noQuestions = 4;
private ClassLoader context;


public MultiChoiceQuestionsActivity() throws XmlPullParserExcept
}

/**
 * Called when the activity is first created.
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.multichoice);
    Log.i("MCQ", "MultiChoice starts");
    opts = (RadioGroup) findViewById(R.id.opts);
    sub = (Button) findViewById(R.id.submit);
    hint = (Button) findViewById(R.id.hint);
    hint.setOnClickListener(this);
    application = (QuizApplication) this.getApplication();//call
    question = (TextView) findViewById(R.id.question_text);
    summary = (TextView) findViewById(R.id.multichoice_summary);
```

(a) Initializing the button

```
if (arg0 == sub) {
    noAnswered++;
    application.incrementQuestionsAttempted();
    int id;
    id = opts.getCheckedRadioButtonId();
    if (currentq.isCorrect(id)) {
        //this if loop was added so that the summary is proper
        //increment number of correct questions
        noCorrect++;
        application.incrementQuestionsCorrect();

    }
    generateSummary();
    if (noAnswered == noQuestions)
        finish();
    else
        generateQuestion(noAnswered);

}

if (arg0 == hint) {
    Log.i("QUIZ", "History selected");
    Toast.makeText(getApplicationContext(), "Battle of Hastins was won by Princ
}
```

(b) Setting up on click event

Figure 8: Implementing the Hint feature

Figure 8 shows how the hint button was implemented and the Figure 9 shows its working on running the application :

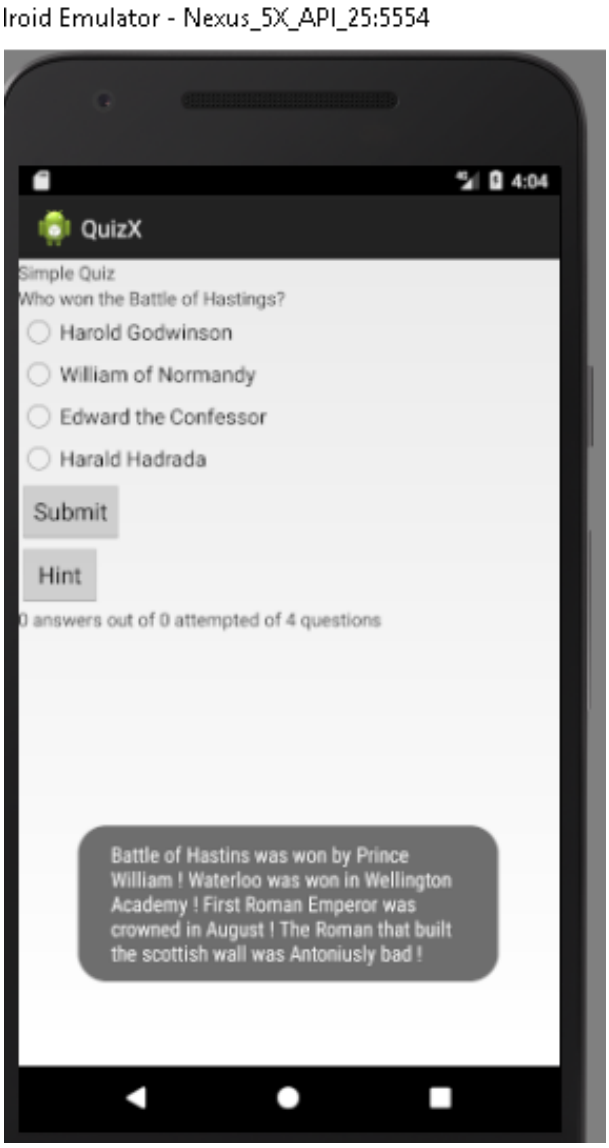

Figure 9: Setting up a vector of XMLGettersSetters

CONCLUSION

The application runs with no problems, the summary works and also the questions for the history portion of the quiz are initiated using a provided xml file. Two enhancements were added which included the multiple choice science questions and an hint feature for the history portion of the application.

REFERENCES

http://www.mysamplecode.com/2011/11/android-parse-xml-file-example-using.html