# 5. Appendix

## 5.1 – Source Code

### 5.1.1 *Calculator.java*

package com.calculatorproject;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.Toolbar;  
import android.text.Html;  
import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import java.util.EmptyStackException;  
  
public class Calculator extends AppCompatActivity {  
  
 private String mInfix;  
 private TextView mCalculatorDisplay;  
 private TextView mOutputDisplay;  
 private ShuntingYard mShuntingYard;  
 private double mAnswer;  
 private String mPostfix;  
 final String ops = "-+÷×^√";  
 private int mPosition;  
  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView(R.layout.*calc\_main*);  
  
 // Initialise toolbar  
 Toolbar toolbar = findViewById(R.id.*my\_toolbar*);  
 setSupportActionBar(toolbar);  
  
 // reset the display text sizes onCreate().  
 mCalculatorDisplay = findViewById(R.id.*calculatorDisplay*);  
 mCalculatorDisplay.setTextSize(getResources().getDimension(R.dimen.*regular*));  
 mOutputDisplay = findViewById(R.id.*outputDisplay*);  
 mOutputDisplay.setTextSize(getResources().getDimension(R.dimen.*regular*));  
  
 // set HTML markup on buttons so the superscript and appears correctly.  
 Button xyPowerButton = findViewById(R.id.*xyPower*);  
 Button squarePowerButton = findViewById(R.id.*squarePower*);  
 Button xyRootButton = findViewById(R.id.*xyRoot*);  
  
 xyPowerButton.setText(Html.*fromHtml*(getString(R.string.*xyPower*)));  
 squarePowerButton.setText(Html.*fromHtml*(getString(R.string.*squarePower*)));  
 xyRootButton.setText(Html.*fromHtml*(getString(R.string.*xyRoot*)));  
  
 // reset the mInfix onCreate().  
 mInfix = "";  
  
 mShuntingYard = new ShuntingYard();  
  
 }  
  
 public void inputAC(View view) {  
  
 // reset all displays and variables  
 mInfix = "";  
 mPosition = 0;  
 mCalculatorDisplay.setText("");  
 mOutputDisplay.setText("");  
 }  
  
 public void inputDigit(View view) {  
  
 // insert input into correct mPosition  
 String input = (String) view.getTag();  
  
 StringBuilder string = new StringBuilder(mInfix);  
 string.insert(mPosition, input);  
 mInfix = string.toString();  
  
 // increment mPosition variable  
 mPosition++;  
  
 // display to user  
 updateDisplay();  
 }  
  
 public void inputOperator(View view) {  
  
 // get input operator from the view parameter's Tag (XML characteristic)  
 String input = (String) view.getTag();  
  
 // add buffer spaces  
 StringBuilder stringInput = new StringBuilder(input);  
 stringInput.insert(0, " ")  
 .insert(stringInput.length(), " ");  
 input = stringInput.toString();  
  
 // insert input into correct mPosition  
 StringBuilder string = new StringBuilder(mInfix);  
 string.insert(mPosition, input);  
 mInfix = string.toString();  
  
 // adds +3 for the character and 2 whitespaces  
 mPosition = mPosition + 3;  
  
 // display to user  
 updateDisplay();  
 }  
  
  
 public void inputOpShortcut(View view) {  
  
 StringBuilder stringBuilderInfix = new StringBuilder(mInfix);  
  
 // if user taps square power button  
 if (view.getId() == R.id.*squarePower*) {  
  
 // requires a whitespace at the start as it will always immediately follow a digit  
 stringBuilderInfix.insert(mPosition, " ^ 2");  
  
 } else { // if user taps square root button  
  
 // requires a trailing whitespace as a number or bracket will immediately follow  
 stringBuilderInfix.insert(mPosition, "2 √ ");  
 }  
  
 // update position  
 mPosition = mPosition + 4;  
  
 // set member infix variable to stringBuilder  
 mInfix = stringBuilderInfix.toString();  
  
 // update GUI  
 updateDisplay();  
 }  
  
 public void inputBracket(View view) {  
  
 // get input from the view's tag  
 String input = (String) view.getTag();  
  
 // add buffer spaces to input  
 StringBuilder stringBuilderInput = new StringBuilder(input);  
 stringBuilderInput.insert(0, " ")  
 .append(" ");  
 input = stringBuilderInput.toString();  
  
 // insert input into infix  
 StringBuilder stringBuilderInfix = new StringBuilder(mInfix);  
 stringBuilderInfix.insert(mPosition, input);  
 mInfix = stringBuilderInfix.toString();  
  
 // update position  
 mPosition = mPosition + 3;  
  
 validateBracketMultiplication();  
 // update GUI to show new infix  
 updateDisplay();  
  
 }  
  
 public void validateBracketMultiplication() {  
  
 StringBuilder stringBuilderInfix = new StringBuilder(mInfix);  
  
 // validate length to prevent StringIndexOutOfBounds  
 if (mInfix.length() > 4) {  
  
 // check if opening bracket  
 if (mInfix.charAt(mPosition - 2) == '('  
 // check if previous token is a digit  
 && Character.*isDigit*(mInfix.charAt(mPosition - 4))) {  
  
 // insert multiplication token  
 stringBuilderInfix.insert(mPosition - 2, "× ");  
  
 // update position variable  
 mPosition = mPosition + 2;  
 }  
 }  
  
 mInfix = stringBuilderInfix.toString();  
 }  
  
 public void delete(View view) {  
  
 StringBuilder stringBuilderInfix = new StringBuilder(mInfix);  
  
  
 // validate string length to prevent crashes from StringIndexOutOfBounds  
 if (stringBuilderInfix.length() > 2) {  
  
 // delete a single character by itself  
 if (Character.*isDigit*(stringBuilderInfix.charAt(mPosition - 1))  
 || ops.contains(String.*valueOf*(stringBuilderInfix.charAt(mPosition - 1)))) {  
  
 // delete character  
 stringBuilderInfix.deleteCharAt(mPosition - 1);  
 mPosition --;  
  
 // delete single character including whitespace (e.g. when there's an operator)  
 } else if (stringBuilderInfix.charAt(mPosition -1) == ' ') {  
  
 // remove token  
 stringBuilderInfix.deleteCharAt(mPosition - 1);  
  
 // position must be updated before editing infix to prevent crashes  
 mPosition --;  
  
 // remove whitespace  
 stringBuilderInfix.deleteCharAt(mPosition - 1);  
 mPosition --;  
 }  
  
  
 }  
  
 mInfix = stringBuilderInfix.toString();  
 updateDisplay();  
 }  
  
 public void submitInfix(View view) {  
  
  
 // validation check to prevent StringIndexOutOfBounds  
 if (mInfix.length() > 0) {  
  
 // try-catch to prevent crashes from badly formed input expressions  
 try {  
  
 // get postfix  
 mPostfix = ShuntingYard.*infixToPostfix*(removePositionMarker(mInfix));  
  
 // get answer  
 mAnswer = ShuntingYard.*evaluateRPN*(mPostfix);  
  
 // display answer to user  
 mOutputDisplay.setText(getString(R.string.*answer*, String.*valueOf*(mAnswer)));  
  
 } catch (EmptyStackException e) {  
 e.printStackTrace();  
  
 //*TODO: Add better errors* Toast.*makeText*(this, "ERROR: expression is malformed",  
 Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 }  
  
 public void shiftPosition(View view) {  
  
 // user taps right button  
 if (view.getId() == R.id.*shiftRight*) {  
  
 // validation to prevent StringIndexOutOfBoundsException  
 if (mPosition < mInfix.length() - 1) {  
  
 // validation check to prevent StringIndexOutOfBoundsException  
 if (mInfix.length() - mPosition > 2) {  
  
 // skips whitespace  
 if (mInfix.charAt(mPosition + 2) == ' ') {  
  
 // increments position variable by 2 to cover the whitespace  
 mPosition = mPosition + 2;  
  
 // skips two whitespaces  
 } else if(mInfix.charAt(mPosition + 1) == ' ') {  
  
 // increment position by 3 when cursor is to the right of a digit to cover  
 // two whitespaces  
 mPosition = mPosition + 3;  
  
 } else if (Character.*isDigit*(mInfix.charAt(mPosition + 1))) {  
  
 mPosition = mPosition + 1;  
 }  
  
 } else {  
  
 // increments position variable  
 mPosition++;  
 }  
 }  
  
 } else { // user shifts left  
  
 // validation check to prevent StringIndexOutOfBoundsException  
 if (mPosition > 0) {  
  
 // skips whitespaces  
 if (mInfix.charAt(mPosition - 1) == ' ') {  
  
 // decreases position variable by 2 to cover the whitespace  
 mPosition = mPosition - 3;  
  
 } else {  
  
 // decreases position variable  
 mPosition--;  
 }  
 }  
 }  
  
 // update display  
 updateDisplay();  
  
 }  
  
 public void updateDisplay() {  
  
 // remove the position marker  
 String cleanExpression = removePositionMarker(mInfix);  
  
  
 // add underscore as a position marker to GUI  
 StringBuilder underscoreString = new StringBuilder(cleanExpression);  
 underscoreString.insert(mPosition, "\_");  
 mInfix = underscoreString.toString();  
  
 // update display  
 mCalculatorDisplay.setText(mInfix);  
 }  
  
 public String removePositionMarker(String infix) {  
  
 // replace underscore with empty string to remove it  
 return infix.replaceAll("\_", "");  
 }  
  
 public void onClickEquations(View view) {  
  
 // send to Equations activity  
 startActivity(new Intent(Calculator.this, Equations.class));  
 }  
  
 public void settings(View view) {  
 }  
}

### 5.1.2 *ShuntingYard.java*

package com.calculatorproject;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Stack;  
  
*/\*\*  
 \* This class handles everything relevant to the process of getting a value from the user's  
 \* input expression.  
 \*  
 \* The user's input is converted to a postfix in Reverse Polish Notation using the infixToPostfix  
 \* method. The postfix can then be used to calculate a result that is returned to the user.  
 \*  
 \** ***@author*** *David Denny  
 \* \*/*public class ShuntingYard {  
  
  
 */\*\*  
 \* Method that takes the user's input and iterates through it to create a postfix in the form  
 \* of Reverse Polish Notation and returns the postfix.  
 \*  
 \** ***@param*** *infix user's input string  
 \** ***@return*** *postfix  
 \* \*/* static String infixToPostfix(String infix) {  
  
 // string that represents every operator. Each operator's precedence can be found by  
 // dividing the index of the operator by 2  
 final String mOps = "-+÷×^√";  
  
 // creates the postfix stringBuilder  
 StringBuilder mPostfix = new StringBuilder();  
  
 // Create new stack containing integers  
 Stack<Integer> mStack = new Stack<>();  
  
 // iteras through each token in the user's infix  
 for (String token : infix.split("\\s")) {  
 if (token.isEmpty()) {  
  
 // if there isn't a token, it returns to the for-each loop  
 continue;  
 }  
  
 // Char variable containing current token  
 char character = token.charAt(0);  
  
 // index of where the token exists in the operator member variable  
 int index = mOps.indexOf(character);  
  
 // if the token is an operator (i.e. exists in the mOps string)  
 if (index != -1) {  
  
 if (mStack.isEmpty()) {  
  
 // push index of token onto the stack  
 mStack.push(index);  
  
 } else {  
  
 while (!mStack.isEmpty()) {  
  
 // find precedence value of current and previous operators by dividing  
 // the index by two  
 int previousPrecedence = mStack.peek() / 2;  
 int currentPrecedence = index / 2;  
  
 //if the previous operator is greater than the current operator or is the  
 // same as long as the current isn't a power  
 if (previousPrecedence > currentPrecedence ||  
 (previousPrecedence == currentPrecedence && character != '^')) {  
  
 // index of previous operator is popped off the stack and appends its  
 // corresponding character in the mOps string to the mPostfix string  
 mPostfix.append(mOps.charAt(mStack.pop())).append(' ');  
  
 } else {  
  
 // break out of loop if prevPrecedence is not greater  
 break;  
 }  
 }  
  
 // at the end of the stack, push the current token's index to the stack  
 mStack.push(index);  
 }  
  
 } else if (character == '(') {  
  
 // push "-2" onto the stack to represent the starting bracket  
 mStack.push(-2);  
  
 } else if (character == ')') {  
  
 // loops over the tokens inside the brackets  
 while (mStack.peek() != -2) {  
  
 // appends all tokens inside the brackets to postfix stringBuilder to ensure  
 // that calculations inside the brackets are done first  
 mPostfix.append(mOps.charAt(mStack.pop())).append(' ');  
 }  
  
 // pops the bracket off the stack  
 mStack.pop();  
  
 } else {  
  
 // if the token is a digit (not bracket or operator), append it to the mPostfix  
 // stringBuilder  
 mPostfix.append(token).append(' ');  
 }  
 }  
  
 while (!mStack.isEmpty()) {  
  
 // pops off and appends the remaining tokens to the mPostfix stringBuilder  
 mPostfix.append(mOps.charAt(mStack.pop())).append(' ');  
 }  
  
 // returns the created postfix string which is now in Reverse Polish Notation  
 return mPostfix.toString();  
 }  
  
 */\*\*  
 \* Method to take the postfix string and convert it into a numerical result to be returned the  
 \* user.  
 \*  
 \** ***@param*** *postfix user's postfix that is used to calculate a result  
 \** ***@return*** *numerical value that is the result of the user's input expression  
 \*\*/* static Double evaluateRPN(String postfix) {  
  
 // make a stack containing Doubles  
 Stack<Double> tokens = new Stack<>();  
  
 // for-each loop iterating over every token in the postfix (removing whitespaces to split  
 // each token up)  
 for (String token : postfix.split(" ")) {  
  
 // finds the corresponding enum to the token  
 Sign sign = Sign.*find*(token);  
  
 // if the .find() function succesfully finds a corresponding enum  
 // (i.e. the token is an operator)  
 if (sign != null) {  
  
 // calls calcSign() function to apply the current operator to the first two  
 // doubles popped off the stack  
 *calcSign*(tokens, sign);  
  
 } else { // i.e. if the token is a number  
  
 // the token is casted to a Double to ensure data types are compatible  
 Double doubleToken = new Double(token);  
  
 // push double onto the stack  
 tokens.push(doubleToken);  
  
 }  
 }  
  
 // pops off the only number remaining in the stack after the loop. This will be the user's  
 // answer and returns it.  
 return tokens.pop();  
 }  
  
  
 protected static Stack<Double> calcSign(Stack<Double> tokens, Sign sign) {  
  
 // pushes the result of the sign parameter with the first two tokens popped off the stack  
 tokens.push(sign.apply(tokens.pop(), tokens.pop()));  
  
 // returns the stack where an operator has been used on two numbers  
 return tokens;  
 }  
  
 public enum Sign {  
  
 *ADD*("+") {  
 public Double apply(Double num1, Double num2) {  
  
 // adds the two input numbers  
 return num2 + num1;  
 }  
 },  
 *SUBTRACT*("-") {  
 public Double apply(Double num1, Double num2) {  
  
 // subtracts the first number from the second number  
 return num2 - num1;  
 }  
 },  
 *MULTIPLY*("×") {  
 public Double apply(Double num1, Double num2) {  
  
 // multiplies the input numbers  
 return num2 \* num1;  
 }  
 },  
 *DIVIDE*("÷") {  
 public Double apply(Double num1, Double num2) {  
  
 // returns the second number divided by the first  
 return num2 / num1;  
 }  
 },  
 *POWER*("^") {  
 public Double apply(Double num1, Double num2) {  
  
 // returns the second number raised by the first  
 return Math.*pow*(num2, num1);  
 }  
 },  
 *ROOT*("√") {  
 public Double apply(Double num1, Double num2) {  
  
 // returns the second number rooted by the first  
 return Math.*pow*(num1, 1.0 / num2);  
 }  
 };  
  
 // operator text constructor  
 private final String mOperatorText;  
  
 // sets the corresponding string denoting the operator to the constructor  
 Sign(String operatorText) {  
 this.mOperatorText = operatorText;  
 }  
  
 // abstract class to make the Sign's corresponding subclasses  
 public abstract Double apply(Double num1, Double num2);  
  
 // create Map member variable made up of a string and enum  
 private static final Map<String, Sign> *mMap*;  
  
 static {  
  
 // initialise mMap variable as a HashMap  
 *mMap* = new HashMap<>();  
  
 // use a for-each loop to populate HashMap with the enum and it's corresponding string  
 for (Sign sign : Sign.*values*()) {  
 *mMap*.put(sign.mOperatorText, sign);  
 }  
 }  
  
  
 public static Sign find(String sign) {  
  
 // returns the Sign enum if it exists in the HashMap  
 return *mMap*.get(sign);  
 }  
  
 }  
  
}

### 5.1.3 *Equations.java*

package com.calculatorproject;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.LinearLayoutManager;  
import android.support.v7.widget.RecyclerView;  
import android.support.v7.widget.Toolbar;  
import android.view.View;  
  
public class Equations extends AppCompatActivity {  
 Toolbar mToolbar;  
 RecyclerView mRecyclerView;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*maths\_equations*);  
  
 // initialise toolbar  
 mToolbar = findViewById(R.id.*quadratic\_equation\_toolbar*);  
 setSupportActionBar(mToolbar);  
  
 // initialise RecyclerView  
 mRecyclerView = findViewById(R.id.*maths\_equation\_recycler*);  
 EquationsRAdapter adapter = new EquationsRAdapter(this);  
  
 mRecyclerView.setAdapter(adapter);  
 mRecyclerView.setLayoutManager(new LinearLayoutManager(this));  
  
 }  
  
 public void handleBackButton(View view) {  
  
 startActivity(new Intent(Equations.this, Calculator.class));  
 }  
}

### 5.1.4 *EquationsRAdapter.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
import android.support.annotation.NonNull;  
import android.support.v7.widget.RecyclerView;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.RelativeLayout;  
import android.widget.TextView;  
  
import java.util.ArrayList;  
  
public class EquationsRAdapter extends RecyclerView.Adapter<EquationsRAdapter.ViewHolder> {  
  
 private ArrayList<String> mContentArray;  
 private RecyclerView mRecyclerView;  
 private View.OnClickListener mOnClickListener = new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 int position = mRecyclerView.getChildLayoutPosition(v);  
  
 switch(position) {  
  
 //Quadratic Equation  
 case 0:  
 v.getContext().startActivity(  
 new Intent(v.getContext(), QuadraticEquation.class));  
 break;  
  
 // Pythagoras' Theorem  
 case 1:  
  
 v.getContext().startActivity(  
 new Intent(v.getContext(), PythagorasTheorem.class));  
 break;  
  
 // Cosine Rule  
 case 2:  
  
 v.getContext().startActivity(new Intent(v.getContext(), CosineRule.class));  
 break;  
  
 // Sine Rule  
 case 3:  
 v.getContext().startActivity(new Intent(v.getContext(), SineRule.class));  
 break;  
  
 // Area of a Triangle  
 case 4:  
 v.getContext().startActivity(new Intent(v.getContext(), AreaTriangle.class));  
 break;  
 }  
  
 }  
 };  
  
 public static class ViewHolder extends RecyclerView.ViewHolder {  
  
 public RelativeLayout mRow;  
 public TextView mContent;  
  
 public ViewHolder(View itemView) {  
 super(itemView);  
  
 mRow = itemView.findViewById(R.id.*default\_recycler\_row*);  
 mContent = itemView.findViewById(R.id.*default\_recycler\_content*);  
 }  
 }  
  
 public EquationsRAdapter(Context context) {  
  
 mContentArray = new ArrayList<>();  
  
 mContentArray.add("Quadratic Equation");  
 mContentArray.add("Pythagoras' Theorem");  
 mContentArray.add("Cosine Rule");  
 mContentArray.add("Sine Rule");  
 mContentArray.add("Area of a Triangle");  
 }  
  
 @NonNull  
 @Override  
 public EquationsRAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
  
 LayoutInflater inflater = LayoutInflater.*from*(parent.getContext());  
 View view = inflater.inflate(R.layout.*recycler\_default\_row*, parent, false);  
  
 view.setOnClickListener(mOnClickListener);  
 return new ViewHolder(view);  
 }  
  
 @Override  
 public void onBindViewHolder(@NonNull EquationsRAdapter.ViewHolder holder, int position) {  
  
 TextView contentTextView = holder.mContent;  
 contentTextView.setText(mContentArray.get(position));  
 }  
  
 @Override  
 public int getItemCount() {  
 return mContentArray.size();  
 }  
  
 @Override  
 public void onAttachedToRecyclerView(@NonNull RecyclerView recyclerView) {  
 super.onAttachedToRecyclerView(recyclerView);  
  
 mRecyclerView = recyclerView;  
 }  
}

### 5.1.5 *QuadraticEquation.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.text.Editable;  
import android.text.Html;  
import android.text.TextWatcher;  
import android.view.View;  
import android.view.inputmethod.InputMethodManager;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.github.kexanie.library.MathView;  
  
public class QuadraticEquation extends AppCompatActivity {  
 MathView quadraticDisplay;  
 String aString;  
 String bString;  
 String cString;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*quadratic\_equation*);  
  
 // set default quadratic equation  
 aString = "a";  
 bString = "b";  
 cString = "c";  
  
 // initialise display TextView  
 quadraticDisplay = findViewById(R.id.*quadratic\_display*);  
  
 // Use string substitution to get text to be displayed and set it to the textview  
 updateDisplay();  
  
 // get EditText  
 EditText aInput = findViewById(R.id.*a\_input*);  
 EditText bInput = findViewById(R.id.*b\_input*);  
 EditText cInput = findViewById(R.id.*c\_input*);  
  
 // Set TextWatcher to update display when user inputs into "a" EditText  
 TextWatcher aTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 aString = s.toString();  
 updateDisplay();  
  
 }  
 };  
  
 // Set TextWatcher to update display when user inputs into "b" EditText  
 TextWatcher bTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 bString = s.toString();  
 updateDisplay();  
 }  
 };  
  
 // Set TextWatcher to update display when user inputs into "c" EditText  
 TextWatcher cTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 cString = s.toString();  
 updateDisplay();  
 }  
 };  
  
 // add TextWatchers to the EditTexts  
 aInput.addTextChangedListener(aTextWatcher);  
 bInput.addTextChangedListener(bTextWatcher);  
 cInput.addTextChangedListener(cTextWatcher);  
  
 }  
  
 public void calculateQuadratic(View view) {  
  
  
 // hide keyboard on button click  
 InputMethodManager inputManager = (InputMethodManager)  
 getSystemService(Context.*INPUT\_METHOD\_SERVICE*);  
 assert inputManager != null;  
 inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),  
 InputMethodManager.*HIDE\_NOT\_ALWAYS*);  
  
 // initialise output TextViews  
 TextView textOutput = findViewById(R.id.*quadratic\_output\_text*);  
 TextView rootOutput = findViewById(R.id.*quadratic\_output\_roots*);  
 TextView discriminantOutput = findViewById(R.id.*quadratic\_output\_discriminant*);  
  
 // validation to prevent crashes from NumberFormatException caused by the user not  
 // inputting all the required values  
 try {  
  
 // initialise variable values  
 double a = Double.*parseDouble*(aString);  
 double b = Double.*parseDouble*(bString);  
 double c = Double.*parseDouble*(cString);  
  
 // calculate the discriminant  
 double discriminant = b \* b - 4 \* a \* c;  
  
 // two real roots  
 if (discriminant > 0) {  
  
 // calculate roots  
 double firstRoot = (-b + Math.*sqrt*(discriminant)) / (2 \* a);  
 double secondRoot = (-b - Math.*sqrt*(discriminant)) / (2 \* a);  
  
 // display roots  
 textOutput.setText(R.string.*quadraticTextMultipleRoots*);  
 rootOutput.setText(Html.*fromHtml*(getString(R.string.*quadraticOutputMultipleRoots*,  
 String.*valueOf*(firstRoot), String.*valueOf*(secondRoot)))  
 );  
  
 // repeated real roots  
 } else if (discriminant == 0) {  
  
 // calculate root  
 double root = (-b + Math.*sqrt*(discriminant)) / (2 \* a);  
  
 // display root  
 textOutput.setText(R.string.*quadraticRepeatedRootsText*);  
 rootOutput.setText(getString(R.string.*quadraticOutputRepeatedRoots*, String.*valueOf*(root)));  
  
 // no real roots  
 } else {  
 // display lack of roots to the user.  
 textOutput.setText(R.string.*quadraticOutputNoRealRoots*);  
 }  
  
 // display the discriminant  
 discriminantOutput.setText(  
 getString(R.string.*quadraticOutputDiscriminant*, String.*valueOf*(discriminant)));  
 } catch (NumberFormatException e) {  
   
 // inform user of error  
 Toast.*makeText*(this, "ERROR: input all required values", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 public void updateDisplay() {  
  
 // create TeX code  
 String quadraticEquation = String.*format*(  
 "$$\\color{white}{x = \\frac{- %2$s \\pm \\sqrt{%2$s^2 - 4 \\times %1$s \\times %3$s}}{2 \\times %1$s}}$$",  
 aString, bString, cString);  
  
 // render and display TeX code  
 quadraticDisplay.setText(quadraticEquation);  
  
 }  
  
 // send user to Equations class onclick  
 public void handleBackButton(View view) {  
 startActivity(new Intent(QuadraticEquation.this, Equations.class));  
 }  
  
}

### 5.1.6 *PythagorasThereom.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.text.Editable;  
import android.text.TextWatcher;  
import android.view.View;  
import android.view.inputmethod.InputMethodManager;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.github.kexanie.library.MathView;  
  
public class PythagorasTheorem extends AppCompatActivity {  
  
 MathView pythagorasDisplay;  
 String aString;  
 String bString;  
 String cString;  
 EditText aInput;  
 EditText bInput;  
 EditText cInput;  
 TextView pythagorasTextOutput;  
 TextView pythagorasAnswerOutput;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.pythagoras\_theorem);  
  
 // set default quadratic equation  
 aString = "a";  
 bString = "b";  
 cString = "c";  
  
 // initialise display TextView  
 pythagorasDisplay = findViewById(R.id.pythagoras\_display);  
  
 // find and set display text  
 updateDisplay();  
  
 // initialise output TextViews  
 pythagorasTextOutput = findViewById(R.id.pythagoras\_output\_text);  
 pythagorasAnswerOutput = findViewById(R.id.pythagoras\_output\_answer);  
  
 // initialise input EditTexts  
 aInput = findViewById(R.id.pythagorasAInput);  
 bInput = findViewById(R.id.pythagorasBInput);  
 cInput = findViewById(R.id.pythagorasCInput);  
  
 // set TextWatcher to update display after user input  
 TextWatcher aTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 aString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
  
 }  
 };  
  
 TextWatcher bTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 bString = s.toString();  
  
 // apply new input  
 updateDisplay();  
  
 }  
 };  
  
 TextWatcher cTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {  
  
 }  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {  
  
 }  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 //get input  
 cString = s.toString();  
  
 // apply input  
 updateDisplay();  
  
 }  
 };  
  
 // add listeners to each EditText  
 aInput.addTextChangedListener(aTextWatcher);  
 bInput.addTextChangedListener(bTextWatcher);  
 cInput.addTextChangedListener(cTextWatcher);  
 }  
  
 public void calculatePythagoras(View view) {  
  
 // hide keyboard on button click  
 InputMethodManager inputManager = (InputMethodManager)  
 getSystemService(Context.INPUT\_METHOD\_SERVICE);  
 assert inputManager != null;  
 inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),  
 InputMethodManager.HIDE\_NOT\_ALWAYS);  
  
 // prevents crashes from NumberFormatException when the user hasn't input the required  
 // values  
 try {  
  
 // "c" is left empty, so calculate hypotenuse  
 if (cInput.getText().toString().equals("")) {  
  
 // get values of triangle sides  
 double a = Double.parseDouble(aString);  
 double b = Double.parseDouble(bString);  
  
 // calculate the hypotenuse  
 double c = Math.sqrt((a \* a) + (b \* b));  
  
 // display answer  
 pythagorasTextOutput.setText(R.string.pythagoras\_text\_hypotenuse);  
 pythagorasAnswerOutput.setText(String.valueOf(c));  
  
 // "a" XOR "b" is empty so calculate the side. Exclusive Or is used here to make sure  
 // at least one side is inputted  
 } else if (aInput.getText().toString().equals("") ^ bInput.getText().toString().equals("")) {  
  
 // calculate a  
 if (aInput.getText().toString().equals("")) {  
  
 // get side and hypotenuse values  
 double b = Double.*parseDouble*(bString);  
 double c = Double.*parseDouble*(cString);  
  
 // only continue if hypotenuse is greater than the side  
 if (c > b) {  
  
 // calculate side a  
 double a = Math.*sqrt*((c \* c) - (b \* b));  
  
 // display answer  
 pythagorasTextOutput.setText(R.string.*pythagoras\_text\_a*);  
 pythagorasAnswerOutput.setText(String.*valueOf*(a));  
 } else {  
  
 Toast.*makeText*(this,  
 "ERROR: a side cannot be larger than the hypotenuse.",  
 Toast.*LENGTH\_SHORT*).show();  
 }  
  
 // calculate b  
 } else {  
  
 // get side and hypotenuse values  
 double a = Double.*parseDouble*(aString);  
 double c = Double.*parseDouble*(cString);  
  
 // only continue if hypotenuse is greater than the side  
 if (c > a) {  
  
 // calculate side b  
 double b = Math.*sqrt*((c \* c) - (a \* a));  
  
 // display answer  
 pythagorasTextOutput.setText(R.string.*pythagoras\_text\_b*);  
 pythagorasAnswerOutput.setText(String.*valueOf*(b));  
 } else {  
  
 Toast.*makeText*(this,  
 "ERROR: a side cannot be larger than the hypotenuse.",  
 Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 } else {  
 Toast.makeText(this,  
  
 "ERROR: make sure you leave one variable blank", Toast.LENGTH\_SHORT).show();  
 }  
  
 } catch (NumberFormatException e) {  
  
 // inform user of error  
 Toast.makeText(this,  
 "ERROR: input all required values", Toast.LENGTH\_SHORT).show();  
 }  
 }  
  
 // update user's input in display expression  
 public void updateDisplay() {  
  
 // create Spanned containing new input  
 String pythagorasExpression = String.format(  
 "$$\\color{white}{%1$s^2 + %2$s^2 = %3$s^2}$$",  
 aString, bString, cString);  
  
 // display new input  
 pythagorasDisplay.setText(pythagorasExpression);  
 }  
  
 // send user back to MathsEquation  
 public void handleBackButton(View view) {  
  
 startActivity(new Intent(PythagorasTheorem.this, Equations.class));  
 }  
}

### 5.1.7 *CosineRule.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.text.Editable;  
import android.text.TextWatcher;  
import android.view.View;  
import android.view.inputmethod.InputMethodManager;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.github.kexanie.library.MathView;  
  
public class CosineRule extends AppCompatActivity {  
  
 MathView cosineDisplay;  
 String aString;  
 String bString;  
 String cString;  
 String angleString;  
 EditText aInput;  
 EditText bInput;  
 EditText cInput;  
 EditText angleInput;  
 TextView cosineTextOutput;  
 TextView cosineAnswerOutput;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*cosine\_rule*);  
  
 // set generic cosine equation values  
 aString = "a";  
 bString = "b";  
 cString = "c";  
 angleString = "A";  
  
 // initialise display MathView  
 cosineDisplay = findViewById(R.id.*cosine\_display*);  
  
 // find and set display text  
 setDisplayExpression();  
  
 // initialise output TextViews  
 cosineTextOutput = findViewById(R.id.*cosine\_output\_text*);  
 cosineAnswerOutput = findViewById(R.id.*cosine\_output\_answer*);  
  
 // initialise input EditTexts  
 aInput = findViewById(R.id.*cosine\_a\_input*);  
 bInput = findViewById(R.id.*cosine\_b\_input*);  
 cInput = findViewById(R.id.*cosine\_c\_input*);  
 angleInput = findViewById(R.id.*cosine\_angle\_input*);  
  
 // set TextWatchers to update display after user input  
 TextWatcher aTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 aString = s.toString();  
  
 // display new input to user  
 setDisplayExpression();  
 }  
 };  
  
 TextWatcher bTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 bString = s.toString();  
  
 // display new input to user  
 setDisplayExpression();  
 }  
 };  
  
 TextWatcher cTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 cString = s.toString();  
  
 // display new input to user  
 setDisplayExpression();  
 }  
 };  
  
 TextWatcher angleTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 angleString = s.toString();  
  
 // display new input to user  
 setDisplayExpression();  
 }  
 };  
  
 aInput.addTextChangedListener(aTextWatcher);  
 bInput.addTextChangedListener(bTextWatcher);  
 cInput.addTextChangedListener(cTextWatcher);  
 angleInput.addTextChangedListener(angleTextWatcher);  
 }  
  
 public void setDisplayExpression() {  
  
 // get the String to be used with string substitution string substitution  
 String cosineRule = String.*format*(  
 "$$\\color{white}{%1$s^2 = %2$s^2 + %3$s^2 \\times %2$s \\times %3$s \\times cos(%4$s)}$$",  
 aString, bString, cString, angleString);  
  
  
 // set TextView to user's new expression  
 cosineDisplay.setText(cosineRule);  
 }  
  
 public void calculateCosineRule(View view) {  
  
 // hide keyboard on button click  
 InputMethodManager inputManager = (InputMethodManager)  
 getSystemService(Context.*INPUT\_METHOD\_SERVICE*);  
 assert inputManager != null;  
 inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),  
 InputMethodManager.*HIDE\_NOT\_ALWAYS*);  
   
 try {  
  
  
 // user wants to calculate "a"  
 if (aInput.getText().toString().equals("")) {  
  
 // get sides and angle values  
 double b = Double.*parseDouble*(bString);  
 double c = Double.*parseDouble*(cString);  
 double angle = Double.*parseDouble*(angleString);  
  
 // convert angle to radians as Math.cos() interprets the input as radians  
 angle = Math.*toRadians*(angle);  
  
 // calculate the side a  
 double a = Math.*sqrt*((b \* b) + (c \* c) \* 2.0 \* b \* c \* Math.*cos*(angle));  
  
 // display answer to user  
 cosineTextOutput.setText(R.string.*cosine\_text\_side*);  
 cosineAnswerOutput.setText(String.*valueOf*(a));  
  
 // user wants to calculate the angle  
 } else if (angleInput.getText().toString().equals("")) {  
  
 double a = Double.*parseDouble*(aString);  
 double b = Double.*parseDouble*(bString);  
 double c = Double.*parseDouble*(cString);  
  
 double angle = Math.*toDegrees*(Math.*acos*(((a \* a) - (b \* b) - (c \* c)) / (2 \* b \* c)));  
  
 cosineTextOutput.setText(R.string.*cosine\_text\_angle*);  
 cosineAnswerOutput.setText(getString(R.string.*cosine\_answer\_angle*, String.*valueOf*(angle)));  
  
 // user has made an error inputting their values  
 } else {  
 Toast.*makeText*(this,  
 "ERROR: your inputs are not valid", Toast.*LENGTH\_SHORT*).show();  
 }  
 } catch (NumberFormatException e) {  
  
 Toast.*makeText*(this,  
 "ERROR: input all required values", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 public void handleBackButton(View view) {  
  
 startActivity(new Intent(CosineRule.this, Equations.class));  
 }  
}

### 5.1.8 *SineRule.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.text.Editable;  
import android.text.TextWatcher;  
import android.view.View;  
import android.view.inputmethod.InputMethodManager;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.github.kexanie.library.MathView;  
  
  
public class SineRule extends AppCompatActivity {  
 MathView sineDisplay;  
 String aString;  
 String bString;  
 String aAngleString;  
 String bAngleString;  
 EditText aInput;  
 EditText aAngleInput;  
 EditText bInput;  
 EditText bAngleInput;  
 TextView sineTextOutput;  
 TextView sineAnswerOutput;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*sine\_rule*);  
  
 // initialise variables  
 aString = "a";  
 bString = "b";  
 aAngleString = "A";  
 bAngleString = "B";  
  
 // get MathView display  
 sineDisplay = findViewById(R.id.*sine\_display*);  
  
 // update display  
 updateDisplay();  
  
 // initialise EditTexts  
 aInput = findViewById(R.id.*a\_sine\_input*);  
 aAngleInput = findViewById(R.id.*a\_capital\_sine\_input*);  
 bInput = findViewById(R.id.*b\_sine\_input*);  
 bAngleInput = findViewById(R.id.*b\_capital\_sine\_input*);  
  
 // initialise output TextViews  
 sineTextOutput = findViewById(R.id.*sine\_output\_text*);  
 sineAnswerOutput = findViewById(R.id.*sine\_output\_answer*);  
  
 // set TextWatchers to update display after user input  
 TextWatcher aTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 aString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 TextWatcher aCapitalTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 aAngleString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 TextWatcher bTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 bString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 TextWatcher bCapitalTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 bAngleString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 // add TextWatchers  
 aInput.addTextChangedListener(aTextWatcher);  
 aAngleInput.addTextChangedListener(aCapitalTextWatcher);  
 bInput.addTextChangedListener(bTextWatcher);  
 bAngleInput.addTextChangedListener(bCapitalTextWatcher);  
  
 }  
  
 public void updateDisplay() {  
  
 // create new TeX code string  
 String sineEquation = String.*format*(  
 "$$\\color{white}{\\frac{%1$s}{sin%2$s} = \\frac{%3$s}{sin%4$s}}$$",  
 aString, aAngleString, bString, bAngleString);  
  
 // display new equation  
 sineDisplay.setText(sineEquation);  
 }  
  
 public void calculateSineRule(View view) {  
  
 // hide keyboard on button click  
 InputMethodManager inputManager = (InputMethodManager)  
 getSystemService(Context.*INPUT\_METHOD\_SERVICE*);  
 assert inputManager != null;  
 inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),  
 InputMethodManager.*HIDE\_NOT\_ALWAYS*);  
  
 // prevent crashes from NumberFormatExceptions (when user hasn't entered input)  
 try {  
  
 // if the user wants to calculate one of the sides  
 if (aInput.getText().toString().equals("") || bInput.getText().toString().equals("")) {  
  
 // get angle values and convert to radians  
 double angleA = Double.*parseDouble*(aAngleString);  
 double angleB = Double.*parseDouble*(bAngleString);  
 angleA = Math.*toRadians*(angleA);  
 angleB = Math.*toRadians*(angleB);  
  
 // if user wants to calculate side a  
 if (aInput.getText().toString().equals("")) {  
  
 // get side b value  
 double b = Double.*parseDouble*(bString);  
  
 // calculate side a using the Sine Rule  
 double a = Math.*sin*(angleA) \* (b / Math.*sin*(angleB));  
  
 // display the answer  
 sineTextOutput.setText(R.string.*sine\_text\_a*);  
 sineAnswerOutput.setText(String.*valueOf*(a));  
  
 // if the user wants to calculate side b  
 } else {  
  
 // get value of side a  
 double a = Double.*parseDouble*(aString);  
  
 // calculate side a using the Sine Rule  
 double b = Math.*sin*(angleB) \* (a / Math.*sin*(angleA));  
  
 // display the answer  
 sineTextOutput.setText(R.string.*sine\_text\_b*);  
 sineAnswerOutput.setText(String.*valueOf*(b));  
 }  
  
 // if the user wants to calculate an angle  
 } else if (aAngleInput.getText().toString().equals("")  
 || bAngleInput.getText().toString().equals("")) {  
  
 // get side variable values  
 double a = Double.*parseDouble*(aString);  
 double b = Double.*parseDouble*(bString);  
  
 // if user wants to calculate angle A  
 if (aAngleInput.getText().toString().equals("")) {  
  
 // get angle B value and convert to radians  
 double bAngle = Double.*parseDouble*(bAngleString);  
 bAngle = Math.*toRadians*(bAngle);  
  
 // calculate angle A using the Sine Rule and convert to degrees  
 double aAngle = Math.*toDegrees*(Math.*asin*(a \* Math.*sin*(bAngle) / b));  
  
 // display result to user  
 sineTextOutput.setText(R.string.*sine\_text\_aAngle*);  
 sineAnswerOutput.setText(String.*valueOf*(aAngle));  
  
 // if the user wants to calculate side B  
 } else {  
  
 // get value of angle A and convert to radians  
 double aAngle = Double.*parseDouble*(aAngleString);  
 aAngle = Math.*toRadians*(aAngle);  
  
 // calculate angle B and convert it to degrees  
 double bAngle = Math.*toDegrees*(Math.*asin*(b \* (Math.*sin*(aAngle) / a)));  
  
 // display result to user  
 sineTextOutput.setText(R.string.*sine\_text\_bAngle*);  
 sineAnswerOutput.setText(String.*valueOf*(bAngle));  
 }  
  
 // user has not entered all the required inputs  
 } else {  
 Toast.*makeText*(this,  
 "ERROR: input all required values", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 // prevent crash and inform user of error  
 } catch (NumberFormatException e) {  
 Toast.*makeText*(this,  
 "ERROR: input all required values", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 }  
  
 // send user back to Equations  
 public void handleBackButton(View view) {  
 startActivity(new Intent(SineRule.this, Equations.class));  
 }  
}

### 5.1.9 *AreaTriangle.java*

package com.calculatorproject;  
  
import android.content.Context;  
import android.content.Intent;  
  
import android.os.Bundle;  
import android.support.annotation.NonNull;  
import android.support.v7.app.AppCompatActivity;  
import android.text.Editable;  
import android.text.TextWatcher;  
import android.view.View;  
import android.view.inputmethod.InputMethodManager;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import io.github.kexanie.library.MathView;  
  
public class AreaTriangle extends AppCompatActivity {  
  
 MathView areaTriangleDisplay;  
 String aString;  
 String bString;  
 String cAngleString;  
 EditText aInput;  
 EditText bInput;  
 EditText cAngleInput;  
 TextView areaTriangleAnswer;  
 TextView areaTriangleText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*area\_triangle*);  
  
 // initialise variables  
 aString = "a";  
 bString = "b";  
 cAngleString = "C";  
  
 // get MathView display  
 areaTriangleDisplay = findViewById(R.id.*area\_triangle\_display*);  
  
 // update display  
 updateDisplay();  
  
 // initialise EditTexts  
 aInput = findViewById(R.id.*area\_triangle\_a\_input*);  
 bInput = findViewById(R.id.*area\_triangle\_b\_input*);  
 cAngleInput = findViewById(R.id.*area\_triangle\_c\_angle\_input*);  
  
 // initialise output TextViews  
 areaTriangleText = findViewById(R.id.*area\_triangle\_output\_text*);  
 areaTriangleAnswer = findViewById(R.id.*area\_triangle\_output\_answer*);  
  
 // set TextWatchers to update display after user input  
 TextWatcher aTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 aString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 TextWatcher bTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 bString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 TextWatcher cAngleTextWatcher = new TextWatcher() {  
 @Override  
 public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
  
 @Override  
 public void onTextChanged(CharSequence s, int start, int before, int count) {}  
  
 @Override  
 public void afterTextChanged(Editable s) {  
 // get input  
 cAngleString = s.toString();  
  
 // display new input to user  
 updateDisplay();  
 }  
 };  
  
 // add TextWatchers  
 aInput.addTextChangedListener(aTextWatcher);  
 bInput.addTextChangedListener(bTextWatcher);  
 cAngleInput.addTextChangedListener(cAngleTextWatcher);  
 }  
  
 public void updateDisplay() {  
  
 // create new TeX code string  
 String areaTriangle = String.*format*(  
 "$$\\color{white}{\\frac{1}{2} \\times %1$s \\times %2$s \\times sin(%3$s)}$$",  
 aString, bString, cAngleString);  
  
 // render new TeX equation  
 areaTriangleDisplay.setText(areaTriangle);  
 }  
  
 public void calculateAreaOfATriangle(View view) {  
  
 // hide keyboard on button click  
 InputMethodManager inputManager = (InputMethodManager)  
 getSystemService(Context.*INPUT\_METHOD\_SERVICE*);  
 assert inputManager != null;  
 inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),  
 InputMethodManager.*HIDE\_NOT\_ALWAYS*);  
  
 // prevent crashes from NumberFormatException (when user hasn't entered required input)  
 try {  
  
 double a = Double.*parseDouble*(aString);  
 double b = Double.*parseDouble*(bString);  
 double cAngle = Double.*parseDouble*(cAngleString);  
 cAngle = Math.*toRadians*(cAngle);  
  
 double area = 0.5 \* a \* b \* Math.*sin*(cAngle);  
  
  
 areaTriangleText.setText(R.string.*area\_triangle\_text*);  
 areaTriangleAnswer.setText(String.*valueOf*(area));  
  
 } catch (NumberFormatException e) {  
  
 Toast.*makeText*(this,  
 "ERROR: input all required values", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 // send user back to Equations  
 public void handleBackButton(View view) {  
 startActivity(new Intent(AreaTriangle.this, Equations.class));  
 }  
}