**Problem Statement:**

**Code portion A** will display the waveform in android while recording but in this case, we aren’t able to save the recorded file. **Code portion B** will save the wave file in SD card of the phone and also from that we may play the recorded voice but here we aren’t able to display the waveform. The problem I am facing here is integrating these two portions of the codes in a way that the recorded voice should be saved in SD card, alongside the recorded waveform being displayed simultaneously.

**Code Portion A**

**public class** WaveformView **extends** View {  
 **public static final int *MODE\_RECORDING*** = 1;  
 **public static final int *MODE\_PLAYBACK*** = 2;  
  
 **private static final int *HISTORY\_SIZE*** = 6;  
  
 **private** TextPaint **mTextPaint**;  
 **private** Paint **mStrokePaint**, **mFillPaint**, **mMarkerPaint**;  
  
 *// Used in draw* **private int brightness**;  
 **private** Rect **drawRect**;  
  
 **private int width**, **height**;  
 **private float xStep**, **centerY**;  
 **private int mMode**, **mAudioLength**, **mMarkerPosition**, **mSampleRate**, **mChannels**;  
 **private short**[] **mSamples**;  
 **private** LinkedList<**float**[]> **mHistoricalData**;  
 **private** Picture **mCachedWaveform**;  
 **private** Bitmap **mCachedWaveformBitmap**;  
 **private int colorDelta** = 255 / (***HISTORY\_SIZE*** + 1);  
 **private boolean showTextAxis** = **true**;  
  
 **public** WaveformView(Context context) {  
 **super**(context);  
 init(context, **null**, 0);  
 }  
  
 **public** WaveformView(Context context, AttributeSet attrs) {  
 **super**(context, attrs);  
 init(context, attrs, 0);  
 }  
  
 **public** WaveformView(Context context, AttributeSet attrs, **int** defStyle) {  
 **super**(context, attrs, defStyle);  
 init(context, attrs, defStyle);  
 }  
  
 **private void** init(Context context, AttributeSet attrs, **int** defStyle) {  
 *// Load attributes* **final** TypedArray a = getContext().obtainStyledAttributes(  
 attrs, R.styleable.***WaveformView***, defStyle, 0);  
  
 **mMode** = a.getInt(R.styleable.*WaveformView\_mode*, ***MODE\_PLAYBACK***);  
  
 **float** strokeThickness = a.getFloat(R.styleable.*WaveformView\_waveformStrokeThickness*, 1f);  
 **int** mStrokeColor = a.getColor(R.styleable.*WaveformView\_waveformColor*,  
 ContextCompat.*getColor*(context, R.color.*default\_waveform*));  
 **int** mFillColor = a.getColor(R.styleable.*WaveformView\_waveformFillColor*,  
 ContextCompat.*getColor*(context, R.color.*default\_waveformFill*));  
 **int** mMarkerColor = a.getColor(R.styleable.*WaveformView\_playbackIndicatorColor*,  
 ContextCompat.*getColor*(context, R.color.*default\_playback\_indicator*));  
 **int** mTextColor = a.getColor(R.styleable.*WaveformView\_timecodeColor*,  
 ContextCompat.*getColor*(context, R.color.*default\_timecode*));  
  
 a.recycle();  
  
 **mTextPaint** = **new** TextPaint();  
 **mTextPaint**.setFlags(Paint.***ANTI\_ALIAS\_FLAG***);  
 **mTextPaint**.setTextAlign(Paint.Align.***CENTER***);  
 **mTextPaint**.setColor(mTextColor);  
 **mTextPaint**.setTextSize(TextUtils.*getFontSize*(getContext(),  
 android.R.attr.***textAppearanceSmall***));  
  
 **mStrokePaint** = **new** Paint();  
 **mStrokePaint**.setColor(mStrokeColor);  
 **mStrokePaint**.setStyle(Paint.Style.***STROKE***);  
 **mStrokePaint**.setStrokeWidth(strokeThickness);  
 **mStrokePaint**.setAntiAlias(**true**);  
  
 **mFillPaint** = **new** Paint();  
 **mFillPaint**.setStyle(Paint.Style.***FILL***);  
 **mFillPaint**.setAntiAlias(**true**);  
 **mFillPaint**.setColor(mFillColor);  
  
 **mMarkerPaint** = **new** Paint();  
 **mMarkerPaint**.setStyle(Paint.Style.***STROKE***);  
 **mMarkerPaint**.setStrokeWidth(0);  
 **mMarkerPaint**.setAntiAlias(**true**);  
 **mMarkerPaint**.setColor(mMarkerColor);  
 }  
  
 @Override  
 **protected void** onSizeChanged(**int** w, **int** h, **int** oldw, **int** oldh) {  
 **super**.onSizeChanged(w, h, oldw, oldh);  
  
 **width** = getMeasuredWidth();  
 **height** = getMeasuredHeight();  
 **xStep** = **width** / (**mAudioLength** \* 1.0f);  
 **centerY** = **height** / 2f;  
 **drawRect** = **new** Rect(0, 0, **width**, **height**);  
  
 **if** (**mHistoricalData** != **null**) {  
 **mHistoricalData**.clear();  
 }  
 **if** (**mMode** == ***MODE\_PLAYBACK***) {  
 *//createPlaybackWaveform();* }  
 }  
  
 @Override  
 **protected void** onDraw(Canvas canvas) {  
 **super**.onDraw(canvas);  
  
 LinkedList<**float**[]> temp = **mHistoricalData**;  
 **if** (**mMode** == ***MODE\_RECORDING*** && temp != **null**) {  
 **brightness** = **colorDelta**;  
 **for** (**float**[] p : temp) {  
 **mStrokePaint**.setAlpha(**brightness**);  
 canvas.drawLines(p, **mStrokePaint**);  
 **brightness** += **colorDelta**;  
 }  
 } **else if** (**mMode** == ***MODE\_PLAYBACK***) {  
 **if** (**mCachedWaveform** != **null**) {  
 canvas.drawPicture(**mCachedWaveform**);  
 } **else if** (**mCachedWaveformBitmap** != **null**) {  
 canvas.drawBitmap(**mCachedWaveformBitmap**, **null**, **drawRect**, **null**);  
 }  
 **if** (**mMarkerPosition** > -1 && **mMarkerPosition** < **mAudioLength**)  
 canvas.drawLine(**xStep** \* **mMarkerPosition**, 0, **xStep** \* **mMarkerPosition**, **height**, **mMarkerPaint**);  
 }  
 }  
  
 **public int** getMode() {  
 **return mMode**;  
 }  
  
 **public void** setMode(**int** mMode) {  
 mMode = mMode;  
 }  
  
 */\* public short[] getSamples() {  
 return mSamples;  
 }\*/* **public void** setSamples(**short**[] samples) {  
 **mSamples** = samples;  
 calculateAudioLength();  
 onSamplesChanged();  
 }  
  
 */\* public int getMarkerPosition() {  
 return mMarkerPosition;  
 }\*/* **public void** setMarkerPosition(**int** markerPosition) {  
 **mMarkerPosition** = markerPosition;  
 postInvalidate();  
 }  
  
 **public int** getAudioLength() {  
 **return mAudioLength**;  
 }  
  
 */\*public int getSampleRate() {  
 return mSampleRate;  
 }\*/* **public void** setSampleRate(**int** sampleRate) {  
 **mSampleRate** = sampleRate;  
 calculateAudioLength();  
 }  
  
 */\*public int getChannels() {  
 return mChannels;  
 }\*/* **public void** setChannels(**int** channels) {  
 **mChannels** = channels;  
 calculateAudioLength();  
 }  
  
 */\*public boolean showTextAxis() {  
 return showTextAxis;  
 }  
  
 public void setShowTextAxis(boolean showTextAxis) {  
 this.showTextAxis = showTextAxis;  
 }\*/* **private void** calculateAudioLength() {  
 **if** (**mSamples** == **null** || **mSampleRate** == 0 || **mChannels** == 0)  
 **return**;  
  
 **mAudioLength** = AudioUtils.*calculateAudioLength*(**mSamples**.**length**, **mSampleRate**, **mChannels**);  
 }  
  
 **private void** onSamplesChanged() {  
 **if** (**mMode** == ***MODE\_RECORDING***) {  
 **if** (**mHistoricalData** == **null**)  
 **mHistoricalData** = **new** LinkedList<>();  
 LinkedList<**float**[]> temp = **new** LinkedList<>(**mHistoricalData**);  
  
 *// For efficiency, we are reusing the array of points.* **float**[] waveformPoints;  
 **if** (temp.size() == ***HISTORY\_SIZE***) {  
 waveformPoints = temp.removeFirst();  
 } **else** {  
 waveformPoints = **new float**[**width** \* 4];  
 }  
  
 drawRecordingWaveform(**mSamples**, waveformPoints);  
 temp.addLast(waveformPoints);  
 **mHistoricalData** = temp;  
 postInvalidate();  
 } **else if** (**mMode** == ***MODE\_PLAYBACK***) {  
 **mMarkerPosition** = -1;  
 **xStep** = **width** / (**mAudioLength** \* 1.0f);  
 *//createPlaybackWaveform();* }  
 }  
  
 **void** drawRecordingWaveform(**short**[] buffer, **float**[] waveformPoints) {  
 **float** lastX = -1;  
 **float** lastY = -1;  
 **int** pointIndex = 0;  
 **float** max = Short.***MAX\_VALUE***;  
  
 *// For efficiency, we don't draw all of the samples in the buffer, but only the ones  
 // that align with pixel boundaries.* **for** (**int** x = 0; x < **width**; x++) {  
 **int** index = (**int**) (((x \* 1.0f) / **width**) \* buffer.**length**);  
 **short** sample = buffer[index];  
 **float** y = **centerY** - ((sample / max) \* **centerY**);  
  
 **if** (lastX != -1) {  
 waveformPoints[pointIndex++] = lastX;  
 waveformPoints[pointIndex++] = lastY;  
 waveformPoints[pointIndex++] = x;  
 waveformPoints[pointIndex++] = y;  
 }  
  
 lastX = x;  
 lastY = y;  
 }  
 }  
  
 */\*Path drawPlaybackWaveform(int width, int height, short[] buffer) {  
 Path waveformPath = new Path();  
 float centerY = height / 2f;  
 float max = Short.MAX\_VALUE;  
  
 short[][] extremes = SamplingUtils.getExtremes(buffer, width);  
  
  
 waveformPath.moveTo(0, centerY);  
  
 // draw maximums  
 for (int x = 0; x < width; x++) {  
 short sample = extremes[x][0];  
 float y = centerY - ((sample / max) \* centerY);  
 waveformPath.lineTo(x, y);  
 }  
  
 // draw minimums  
 for (int x = width - 1; x >= 0; x--) {  
 short sample = extremes[x][1];  
 float y = centerY - ((sample / max) \* centerY);  
 waveformPath.lineTo(x, y);  
 }  
  
 waveformPath.close();  
  
 return waveformPath;  
 }\*/  
  
 /\*private void createPlaybackWaveform() {  
 if (width <= 0 || height <= 0 || mSamples == null)  
 return;  
  
 Canvas cacheCanvas;  
 if (Build.VERSION.SDK\_INT >= 23 && isHardwareAccelerated()) {  
 mCachedWaveform = new Picture();  
 cacheCanvas = mCachedWaveform.beginRecording(width, height);  
 } else {  
 mCachedWaveformBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB\_8888);  
 cacheCanvas = new Canvas(mCachedWaveformBitmap);  
 }  
  
 Path mWaveform = drawPlaybackWaveform(width, height, mSamples);  
 cacheCanvas.drawPath(mWaveform, mFillPaint);  
 cacheCanvas.drawPath(mWaveform, mStrokePaint);  
 drawAxis(cacheCanvas, width);  
  
 if (mCachedWaveform != null)  
 mCachedWaveform.endRecording();  
 }\*/  
  
 /\* private void drawAxis(Canvas canvas, int width) {  
 if (!showTextAxis) return;  
 int seconds = mAudioLength / 1000;  
 float xStep = width / (mAudioLength / 1000f);  
 float textHeight = mTextPaint.getTextSize();  
 float textWidth = mTextPaint.measureText("10.00");  
 int secondStep = (int)(textWidth \* seconds \* 2) / width;  
 secondStep = Math.max(secondStep, 1);  
 for (float i = 0; i <= seconds; i += secondStep) {  
 canvas.drawText(String.format("%.2f", i), i \* xStep, textHeight, mTextPaint);  
 }  
 }\*/*}

**Code Portion B**

**public class** conduct\_word\_test **extends** AppCompatActivity {  
  
 Button **summary**, **stop**, **record**, **btn\_play\_audio**;  
  
 String **AudioSavePathInDevice** = **null**;  
 MediaRecorder **mediaRecorder** ;  
 Random **random** ;  
 String **RandomAudioFileName** = **"ABCDEFGHIJKLMNOP"**;  
 **public static final int *RequestPermissionCode*** = 1;  
 MediaPlayer **mediaPlayer** ;  
  
  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  
 getWindow().setFlags(WindowManager.LayoutParams.***FLAG\_FULLSCREEN***, WindowManager.LayoutParams.***FLAG\_FULLSCREEN***);  
 Configuration config = getResources().getConfiguration();  
 **if** (config.**smallestScreenWidthDp** > 480) {  
 setContentView(R.layout.***conduct\_wordtest\_tablet***);  
 } **else** {  
 setContentView(R.layout.***conduct\_wordtest***);  
 }  
  
 Resources res = getResources();  
 Drawable drawable = res.getDrawable(R.drawable.***play\_img***);  
 drawable = DrawableCompat.*wrap*(drawable);  
 DrawableCompat.*setTint*(drawable, getResources().getColor(R.color.***colorPrimary***));  
 Resources res1 = getResources();  
 Drawable drawable1 = res1.getDrawable(R.drawable.***stop\_img***);  
 drawable = DrawableCompat.*wrap*(drawable1);  
 DrawableCompat.*setTint*(drawable, getResources().getColor(R.color.***colorPrimary***));  
 Resources res2 = getResources();  
 Drawable drawable2 = res2.getDrawable(R.drawable.***record\_img***);  
 drawable = DrawableCompat.*wrap*(drawable2);  
 DrawableCompat.*setTint*(drawable, getResources().getColor(R.color.***colorPrimary***));  
  
  
 **summary** = (Button) findViewById(R.id.***summary\_btn***);  
 **stop** = (Button) findViewById(R.id.***stop\_btn***);  
 **record** = (Button) findViewById(R.id.***record\_btn***);  
 **btn\_play\_audio** = (Button)findViewById(R.id.***btn\_play\_audio***);  
  
  
 **summary**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if** (v.getId() == R.id.***summary\_btn***) {  
 Intent k = **new** Intent(conduct\_word\_test.**this**, summary.**class**);  
 startActivity(k);  
 }  
 }  
 });  
  
 **stop**.setEnabled(**false**);  
 **btn\_play\_audio**.setEnabled(**false**);  
 **random** = **new** Random();  
 **record**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
  
 **if**(checkPermission()) {  
  
 **AudioSavePathInDevice** = Environment.*getExternalStorageDirectory*().getAbsolutePath() + **"/"** + CreateRandomAudioFileName(5) + **"AudioRecording.3gp"**;  
  
 MediaRecorderReady();  
  
 **try** {  
 **mediaRecorder**.prepare();  
 **mediaRecorder**.start();  
 } **catch** (IllegalStateException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 } **catch** (IOException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
  
 **record**.setEnabled(**false**);  
 **stop**.setEnabled(**true**);  
  
  
 Toast.*makeText*(conduct\_word\_test.**this**, **"Recording started"**, Toast.***LENGTH\_LONG***).show();  
 } **else** {  
 requestPermission();  
 }  
  
 }  
 });  
 **stop**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **mediaRecorder**.stop();  
 **stop**.setEnabled(**false**);  
 **btn\_play\_audio**.setEnabled(**true**);  
  
 Toast.*makeText*(conduct\_word\_test.**this**, **"Recording Completed"**, Toast.***LENGTH\_LONG***).show();  
 }  
 });  
 **btn\_play\_audio**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) **throws** IllegalArgumentException,  
 SecurityException, IllegalStateException {  
  
 **stop**.setEnabled(**false**);  
 **record**.setEnabled(**true**);  
  
 **mediaPlayer** = **new** MediaPlayer();  
 **try** {  
 **mediaPlayer**.setDataSource(**AudioSavePathInDevice**);  
 **mediaPlayer**.prepare();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 **mediaPlayer**.start();  
 Toast.*makeText*(conduct\_word\_test.**this**, **"Recording Playing"**, Toast.***LENGTH\_LONG***).show();  
 }  
 });  
  
 }  
  
 **public void** MediaRecorderReady(){  
 **mediaRecorder**=**new** MediaRecorder();  
 **mediaRecorder**.setAudioSource(MediaRecorder.AudioSource.***MIC***);  
 **mediaRecorder**.setOutputFormat(MediaRecorder.OutputFormat.***THREE\_GPP***);  
 **mediaRecorder**.setAudioEncoder(MediaRecorder.OutputFormat.***AMR\_NB***);  
 **mediaRecorder**.setOutputFile(**AudioSavePathInDevice**);  
 }  
  
 **public** String CreateRandomAudioFileName(**int** string){  
 StringBuilder stringBuilder = **new** StringBuilder( string );  
 **int** i = 0 ;  
 **while**(i < string ) {  
 stringBuilder.append(**RandomAudioFileName**.  
 charAt(**random**.nextInt(**RandomAudioFileName**.length())));  
  
 i++ ;  
 }  
 **return** stringBuilder.toString();  
 }  
  
 **private void** requestPermission() {  
 ActivityCompat.*requestPermissions*(conduct\_word\_test.**this**, **new** String[]{***WRITE\_EXTERNAL\_STORAGE***, ***RECORD\_AUDIO***}, ***RequestPermissionCode***);  
 }  
  
 @Override  
 **public void** onRequestPermissionsResult(**int** requestCode, String permissions[], **int**[] grantResults) {  
 **switch** (requestCode) {  
 **case *RequestPermissionCode***:  
 **if** (grantResults.**length**> 0) {  
 **boolean** StoragePermission = grantResults[0] == PackageManager.***PERMISSION\_GRANTED***;  
 **boolean** RecordPermission = grantResults[1] == PackageManager.***PERMISSION\_GRANTED***;  
  
 **if** (StoragePermission && RecordPermission) {  
 Toast.*makeText*(conduct\_word\_test.**this**, **"Permission Granted"**, Toast.***LENGTH\_LONG***).show();  
 } **else** {  
 Toast.*makeText*(conduct\_word\_test.**this**,**"Permission Denied"**,Toast.***LENGTH\_LONG***).show();  
 }  
 }  
 **break**;  
 }  
 }  
  
 **public boolean** checkPermission() {  
 **int** result = ContextCompat.*checkSelfPermission*(getApplicationContext(), ***WRITE\_EXTERNAL\_STORAGE***);  
 **int** result1 = ContextCompat.*checkSelfPermission*(getApplicationContext(), ***RECORD\_AUDIO***);  
 **return** result == PackageManager.***PERMISSION\_GRANTED*** && result1 == PackageManager.***PERMISSION\_GRANTED***;  
 }  
  
  
}