ANYTHING BUT JAVA

An Analysis of Kotlin and Xamarin

Paul Sarda

Contents

Introduction

Java is a widely-used programming language developed in 1995, which is the number one language on GitHub.

Kotlin is a programming language, introduced in 2011, which is primarily used as a Java alternative as it has a more updated syntax than Java, allowing programmers to write the same program with less lines and garbage.

Xamarin, which has also been around since 2011, is a cross-development platform that initially operated on iOS and Windows, and more recently on Android. Xamarin is primary used for multiplatform applications.

Research Objective

The primary research objective was to determine how viable alternatives to Java are for writing Android applications. The alternatives I researched were Kotlin and C#/Xamarin.

Methodology

I utilised an investigative research methodology, taking the following approach:

- Initial review of all possible Java alternatives;
- I selected Kotlin, as it was a direct request from my lecturer, and C#/Xamarin as I had C# experience from an earlier class (Object-oriented programming) and wanted to expand the knowledge I already had;
- Created a relatively simple example program. I then took this program and created it three times, first utilising Java, then Kotkin, then Xamarin. This helped me to develop an understanding of each of these Android development platforms, and how they worked, their advantages and disadvantages.

Java Example Program

The following Sections examine how I have written the same Java program in Kotlin and C#. Providing a direct comparison between these three languages has assisted me to establish a baseline. The Java version of the Program can be found under Appendix A.

I found writing the Java version easy to write because there were so many online resources to assist me. Having said this, the actual process of writing Java was frustrating as it was time consuming and seemingly unnecessary so, so half the time I felt like what I was having to do was a waste of my time.

C#/ Xamarin

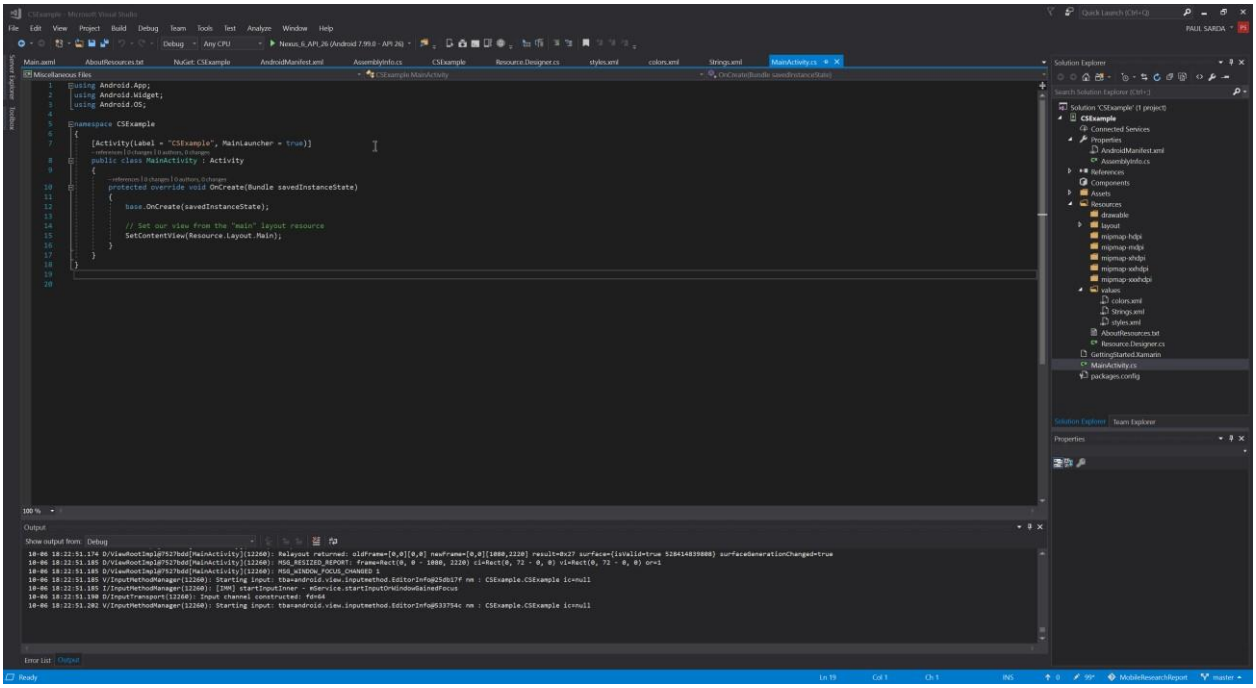
C# is a programming language which is completely separate from Xamarin. Xamarin is a platform which is written in C#.

IDE

In 2016 when Microsoft purchased Xamarin, they halted the development of the independent Xamarin application and moved the Xamarin mobile development into Visual Studio. This made Visual Studio the primary IDE for C# Android development (Microsoftcom, 2016).

Using Visual Studio adding native C# library with your project is much easier (Durrr) due to its ability to just use NuGet.

As for accessing Android-related files, Visual Studio has the same difficulty as Android Studio. The XML files predictive text is simply broken. For example, it won't predict the word 'Android' when typing properties, which resulted in me having to type far more letters of the word than I need to when utilising Android Studio.

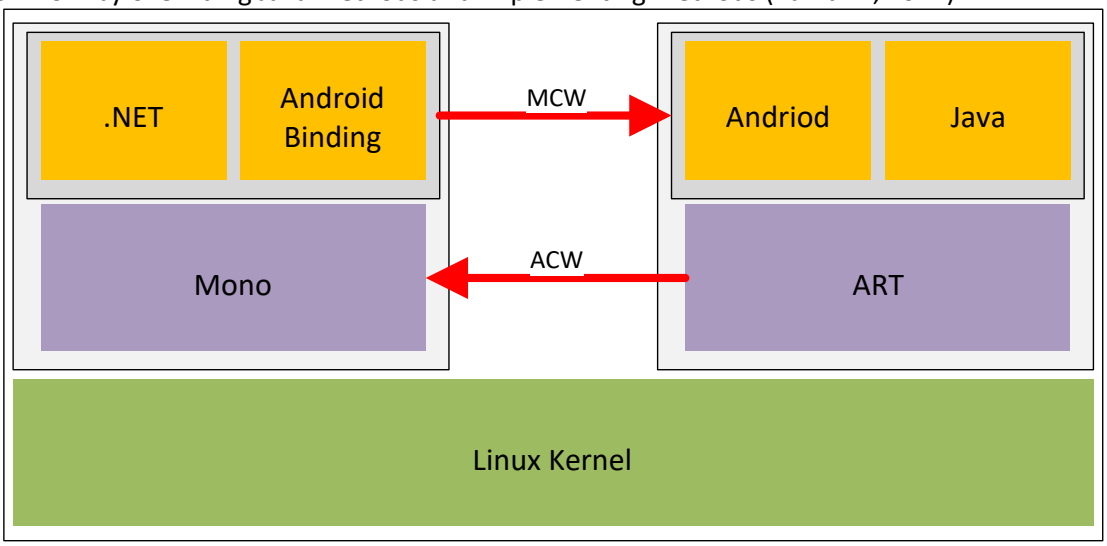


Compilation time in is discussed in the example program section.

Compatibility with Java

Theory

Xamarin runs on top of the Mono execution environment with the .NET APIs and Android bindings, allowing developers to access the underlying Linux Kernel though Mono. It doesn't allow access to the Android OS, but instead this is completed though the MCW's or managed callable wrappers, which work by overriding Java methods and implementing methods (Xamarin, 2017).



Implementation

There are three approaches to getting Java code running in Xamarin, which are:

1. Create a Java Bindings Library on page 6
2. Java Native Interface on page 8
3. Port the code on page 8

Create a Java Bindings Library

The following provides the technique used by Xamarin for the core Android code.
Xamarin takes an inputted .JAR, which will then generate a `.dll` file that contains; Original `.JAR` files and the MCW.
An example of how this is done can be found in *Figure 1* below (Xamarin, 2017).

In this example I will be binding a small Android library which I used in my custom program. Written by Jake Wharton, it abstracts dealing with the sensor manager making it simple to listen for a single shake. Further details can be found at https://github.com/square/seismic.

The process of creating a binding library is daunting at first, primarily because of the lacking Xamarin page binding libraries, which are in a separate project from the main Android application with the following file structure.

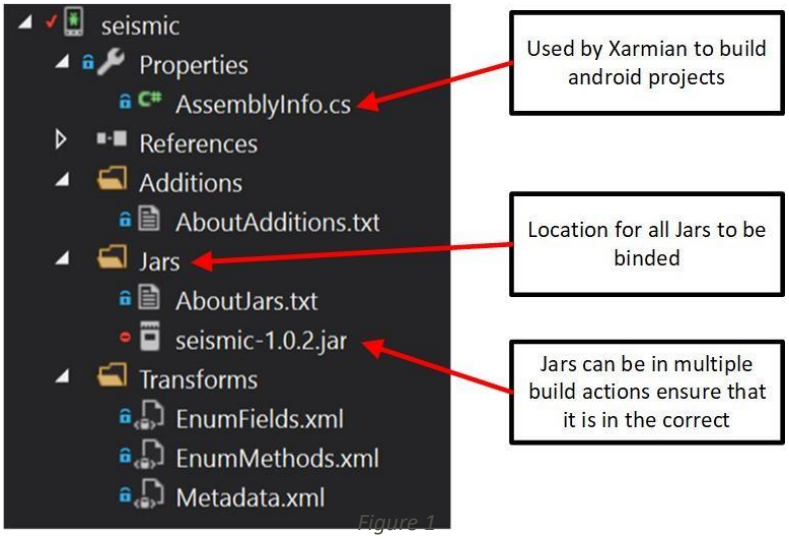


Figure 1

As you can see, the seismic jar has been added, and when the project is built, the binding library is then created.

Adding a reference to the binding library project allows easy access to this library, utilising the same method as in an Android code.

```
...
using Com.Squareup.Seismic;
...

namespace CSExample
{
    ...
    public class MainActivity : Activity, ShakeDetector.IListener
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            ...

            SensorManager sensorManager = (SensorManager)GetSystemService(Context.SensorService);
            ShakeDetector sd = new ShakeDetector(this);
            sd.Start(sensorManager);
        }

        public void HearShake()
        {
            TextView textView = FindViewById<TextView>(Resource.Id.ShakeItUp);

            textView.Visibility = Android.Views.ViewStates.Visible;
        }
    }
}
```

The Binding name is the same as in Java -- except C# naming conventions are applied.

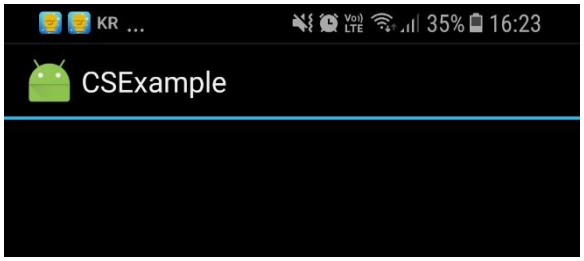
Once again C# naming conventions have been applied to the binding libraries.

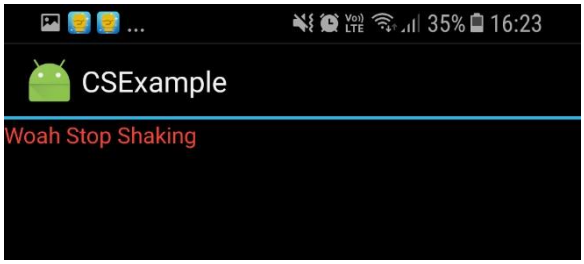
Additionally, they are fully featured and can even be used in an inheritance tree. I found this somewhat bizarre when you think that essentially having a .NET class Inherit from Java code.

This continues the trend of being the same, except C# conventions are applied -- but does not auto-generate properties like Kotlin does.

Does not need the override keyword because it's C# but works all the same.

Below is the application running and the two screens pre-shake and post-shake.





Binding Java libraries offers a quick and easy way of simply using Java Library in a Xamrian project with a few issues that include extra overhead, implementing what is essently Java code in `.NET`, and the fact that the libraries will need to be manual updated.

Java Native Interface

A simpler attentive to creating a bindings library is recommended when needing to call a single method or limited use inside of the main program.

Which is used not for porting but for making C# code interact with Java code.

Where C# classes can inherit from Java classes, which will autogenerate a Android callable wrapper by mandriod.exe which maps to the C# class (Xamarin, 2017).

In the example program I used a Java-native interface because the Android ArrayAdapter required the class to be a Java object, as shown in Figure 2.

```
// Must Inherit from Java.Lang.Object to be used by
// many Andriod methods public class
ListRowHolder : Java.Lang.Object
{     public ListRowHolder(View
view)
    {
        ...
    }
}
```

Figure 2

Simply calling causes the class to inherit from `Java.Lang.Object` or any Java Binding Library will result in an ACW generated for it.

Port the code

As expected, this is just porting the code to C#, which means rewriting it or using an automated converter. Rewriting the code can be slow and monotonous, and requires understanding of Java and C#. it is depended on the amount of code that needs to be translated. This is the method I used for the example program (Xamarin, 2017).

The alternative is by using a Java-to-C# conversion tool. Xarmian recommends the tool Sharpen which can be found at https://github.com/mono/sharpen.

Sharpen, which has not been updated since 2015, only supports up to Java 7. This means that an outdated version of Java needs to be installed and then linked to Maven. I then found out that I needed to use Sharpen through Eclipse, which I don't have installed, so for that reason I didn't try it myself due to time constraints.

Android wear development

Xamarin has full support for Android wear development, but is unable to create a cross-platform single project for Android wear and other platforms. However, two single projects can be created that share common .NET code in a separate library which I dive into more detail in Cross-Platform Development on Page 10 (Xamarin, 2017).

Cross-Platform Development

Unlike Android Studio, using Xamarin is the only option for C# development on Android devices. Xamarin allows IOS and Android apps (and Windows phone – but who cares about that?) to be developed simultaneously (Xamarincom, 2017), but still requires a Macintosh computer to develop IOS apps., so I am unable to show this feature (Xamarincom, 2017).

Xamarin also supports the following platforms:

- Android
- Android Wear
- IOS Wear
- Tizen (later this year)
- Windows phone
- Windows PC

Allowing shared code between all the above platforms allows for minimal code rewriting, which essentially does the same thing.

Design

Xamarin recommends structuring applications as shown in Figure 3.

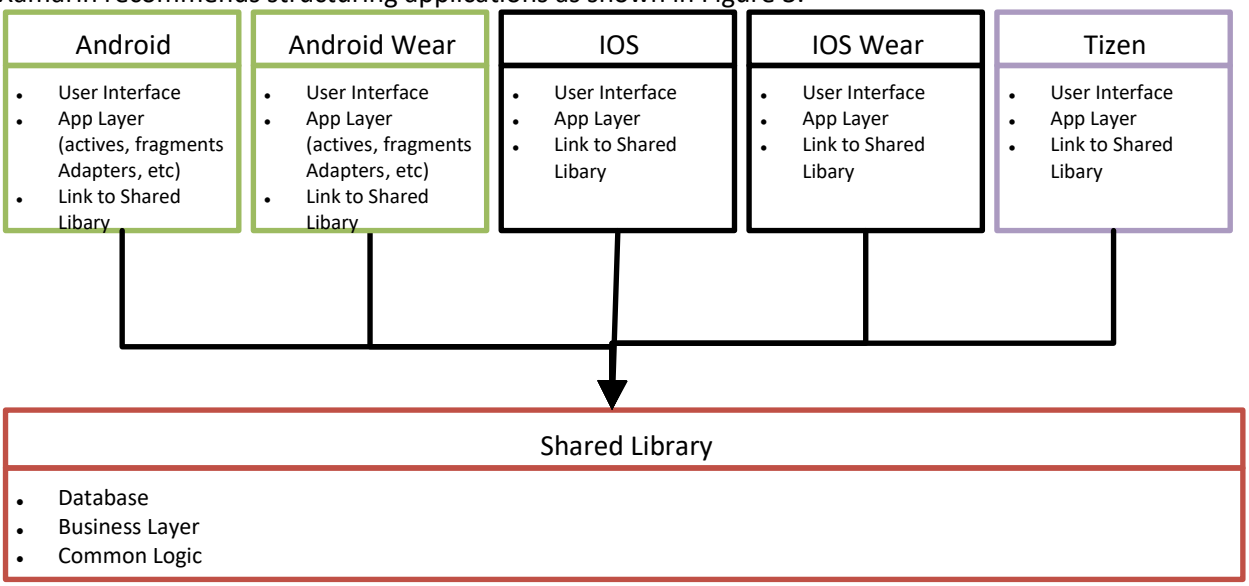


Figure 3

This requires the UI for each platform to follow the UI conventions and patterns, which causes the application to feel like a native IOS, instead of like an Android app running on IOS.

Additionally, because the shared library is used for each platform shared databases are possible allowing users to easily switch between platforms (Xamarin, 2017).

Example Program

When writing C# for Android it is just a skin for Java which shows when it comes to compiling which takes much longer in Xamarin from my not scientific test of cleaning it once and comparing the two. In Android it took 19.42 seconds while in Xamarin it took 55.47 seconds; Now of course this was just my experience, but it was unbearable.

ListRowData Class

I am omitting this Class because it has no Android components, however, if you have your heart set on seeing it can be found in Section B.3 ListRowData of this Report.

ListRowHolder

Very similar to the Kotlin class, ListRowHolder uses the same generic FindViewByID instead of casting, which I prefer.

In C# everything, of course, inherits from `System.Object`, but in order for your C# classes to use any Java binding library classes' methods you will need to make sure that your C# class inherits from `Java.Lang.Object`.

```
namespace ExampleProgram.src
{
    // Must Inherit from Java.Lang.Object to be used by
    // many Andriod methods     public class
ListRowHolder : Java.Lang.Object
    {         public ListRowHolder(View
view)
        {
            Title = view.FindViewById<TextView>(Resource.Id.title);
            SubTitle = view.FindViewById<TextView>(Resource.Id.sub_title);
            Button = view.FindViewById<Button>(Resource.Id.cool_button);
            Input = view.FindViewById<EditText>(Resource.Id.input);
        }

        OMITED GETERS AND SETTERS
    }
}
```

Full Version here: B.2 ListRowHolder

ListExampleAdapter

Creating the ArrayAdapter in C# is done as expected, which includes taking advantage of Getters and Setters. There are some minor differences in how SetTag works, as well as a different Event system. Additionally, the file in the end is 65 lines, but keep in mind that {} have their own lines, so in reality, it is much shorter.

```
public class ListAdapterExmaple : ArrayAdapter<ListRowData> {
    ...SAME AS JAVA...

    // Base is same as super     public ListAdapterExmaple(Context context, int
resource, List<ListRowData> data) :
        base(context, resource, data)
    {
        ...SAME AS JAVA...
        _inflater = LayoutInflater.From(_context);
    }

    // Final is imposible in C# for method      public override View GetView(int
position, View convertView, ViewGroup parent)
    {
        ...SAME AS JAVA...
         if(convertView ==
null)
        {
            ...SAME AS JAVA...
            // Must be a Java Object
            // Tags Must have id's which is a pain for
            // only a single tag
view.SetTag(Resource.Id.TAG_VIEW_HOLDER, vh);
        }         else         {             view = convertView;
vh = view.GetTag(Resource.Id.TAG_VIEW_HOLDER) as ListRowHolder;
        }
         vh.Title.Text = _data[position].Title;
vh.SubTitle.Text = _data[position].SubTitle;
```

```
        // C# style events
vh.Button.Click += (sender, e) => {
            // Similar expect ToastLength is a
            // separate enum instead of being a const in Toast
            Toast.MakeText(
                _context,
                _context.GetString(Resource.String.popup),
                ToastLength.Long
            ).Show();

            _data[position].SubTitle = vh.Input.Text.ToString();
            NotifyDataSetChanged();
        };
         return
view;
    }
}
```

Full version here: B.4 ListAdatperExample

Main Activity

Main Activity is very similar to the Java code, with some pre-processor labels to be used by the Java binding library.

```
[Activity(Label = "ExampleProgram", MainLauncher = true)] public
class MainActivity : Activity
{     private ViewHolder
_viewHolder;
     protected override void OnCreate(Bundle savedInstanceState)

{         base.OnCreate(savedInstanceState);

        // uses Resouces instend of R
        SetContentView(Resource.Layout.Main);

        // Same as Java just no get
        _viewHolder = new ViewHolder(Window.DecorView.RootView);
        InisliseListView();
    }      private void
InisliseListView()
    {
        List<ListRowData> listData = new List<ListRowData>
        {             new ListRowData("One Cool beans", "subtitle beans
one"),             new ListRowData("Two Cool beans", "subtitle beans
three"),              new ListRowData("Three Cool beans", "subtitle beans
nighty"),              new ListRowData("Four Cool beans", "subtitle beans
beanos")         };

        // C# naming convesions
        IListAdapter listAdapter = new ListAdapterExmaple(this, Resource.Layout.list_row, listData);
        _viewHolder.ListView.Adapter = listAdapter;
    }
}
```

Full version here: B.1 Main Activity

APK Size

Xamarin APKs are generally larger because of the extra bloat from the wrappers, etc, causing the final application to be 15.88MB. This is extremely large considering what the application is, and is 10.84MB or 146% larger than the Java version APK. This means the footprint of the storage on the phone is larger, and therefore will more likely be deleted when the phone is low in storage.

Launch time

Time in ms	Average diffrence
852	149.5
676	-26.5
673	-29.5
674	-28.5
671	-31.5
669	-33.5

The launch time is, on an average, 702.5ms – almost three times slower than the Java launch time, at 273.5ms. When I first launched my application, as always, it took a fair bit longer to launch, but as I continued to hard-close the app, then relaunch, the start-up time was more consistent.

Conclusion

C# has consistently shorter and fewer lines with utilising Getters and Setters. The limited online resources available (due to the smaller community) is an issue, making it difficult to start-off if you don't already understand what an array adapter should look like. It also has an amazingly larger APK size, which means that it is a bigger program that takes longer to download, leaves a larger footprint on the phone, which means it will probably be deleted when the phone is running out of memory.

Kotlin

Introduction

Kotlin, which first appeared in 2011, is a programming language designed by JetBrains (Jetbrainscom, 2017). An open source under the Apache license, Kotlin is 100% compatible with Java code, to the point of being able to call Java functions (Kotlinlangorg, 2017).

Kotlin's full source code can be found at https://github.com/jetbrains/kotlin

Key Differences

Kotlin, like C++ and unlike Java and C#, can work with both Object-oriented programming and functional programming styles. It also supports (Kotlinlangorg, 2017). After writing my example program in Kotlin I found the transition was harder going from C# to Java than I was expecting, but the conscience of the code without the loss of what the code is doing was very appealing.

Example Program

I recreated the previously mentioned Example program in Kotlin, which I found to be a very pleasant experience. Having said this if I had not had any Android development experience I would have found it much more difficult as there was a lack of online examples available.

ListRowData Class

Thr ListRow Data Class is much shorter at 20 lines including whitespace, compared to the Java equivalent at 32 lines. Although its only 12 lines, I found the minutiae of typing soul crushing.

```
public void SetTitle(String value)
```

makes it worth learning a new language.

```
//Constructor arguments in class delegation public class
ListRowData(title : String, subTitle : String){
    public var Title : String =
title         get() = field
        // Values type does not have to be declared but value must be a string
set(value) {              if(value.length > 4){                field = value
            }
        }
    public var SubTitle : String =
subTitle         get() = field
set(value) {             if(value !=
Title){                 field = value
            }
        }
}
```

Full Version found here: C.3 List Row Data

ListRowHolder

Once again, the ListRow Holder class is about half the length of the Java version at 15 lines, while Kotlin is 6 lines. The Kotlin version is also null safe compared to the Java version which is missing null checks.

```
// row is null safe public class ListRowHolder(row: View) {     public val
title : TextView = row.findViewById<TextView>(R.id.title)     public val
subTitle: TextView = row.findViewById<TextView>(R.id.sub_title)     public val
button : Button = row.findViewById<Button>(R.id.cool_button)     public val
input : EditText = row.findViewById<EditText>(R.id.input)
}
```

Full Version found here: C.2 List Row Holder

MainActivity class

This class is the most like the Java version except for a few details commented in the code

```
class MainActivity : AppCompatActivity() {

     private class ViewHolder(view : View){         var listView :

ListView = view.findViewById(R.id.list_view)

    }


    // ? means that it can be null     private

var _viewHolder : ViewHolder? = null

     override fun onCreate(savedInstanceState: Bundle?)

{         super.onCreate(savedInstanceState)

setContentView(R.layout.activity_main)

        _viewHolder = ViewHolder(window.decorView.rootView)

        InitliseListView()

    }      private fun InitliseListView(){         // No new operator in

Kotlin         val listData : MutableList<ListRowData> =

mutableListOf<ListRowData>()

         listData.add(ListRowData("One Cool beans", "subtitle beans one"))

listData.add(ListRowData("Two Cool beans", "subtitle beans three"))

listData.add(ListRowData("Three Cool beans", "subtitle beans nighty"))

listData.add(ListRowData("Four Cool beans", "subtitle beans beanos"))

         val listAdapter : BaseAdapter

=

            ListExampleAdapter(this.applicationContext, R.layout.list_row, listData)

        // Because Viewholder can be null ?. must be made to make a safe call

        // if _viewHolder is null ViewHolder? will be thrown

        _viewHolder?.listView?.adapter = listAdapter

    }

}
```

Full Version found here: C.1 Main Activity

ListExampleAdapter

Unlike the other two classes, the ListExampleAdapter class is only slightly shorter due to the constructor and super call located in the same line as the declaration. The getters and setters also make it easier accessing and setting data

```
public class ListExampleAdapter(context: Context, resourceId: Int, data :
MutableList<ListRowData>) :

    // subclass constuntor is called here

    ArrayAdapter<ListRowData>(context, resourceId, data) {


    // No constuntor is needed here     private val

_inflator = LayoutInflater.from(context)     private val

_data = data     private val _contex = context

     override fun getView(position: Int, convertView: View?, parent: ViewGroup): View?

{         val view: View?

        val vh: ListRowHolder

         if (convertView == null) {             view =

this._inflator.inflate(R.layout.list_row, parent, false)             vh =

ListRowHolder(view)             view.tag = vh

        } else {             view =

convertView             vh = view.tag as

ListRowHolder

        }


        // Text for text view is a simple property

vh.title.text = _data[position].Title         //

Access to List's can be done though []

vh.subTitle.text = _data[position].SubTitle


        // Much shorter than in Java

vh.button.setOnClickListener{


            // Line is exact same as in Java

            Toast.makeText(

                _contex,
```

```
                _contex.getString(R.string.popup),
                Toast.LENGTH_SHORT).show()


            _data[position].SubTitle = vh.input.text.toString()
            // Line is exact same as in Java
notifyDataSetChanged()
        }
         return
view
    }
}
```

Full version found here: C.4 List Example Adapter

APK size

As expected the APK size is almost the same as Java and close enough to 5.93MB. The extra size (93KB) is almost definitely not due to Kotlin, but much more likely due to me putting extra garbage somewhere.

Launch Times

The table below demonstrates the times the Kotlin example program took to launch.

Time in ms	Average diffrence
353	66.5
346	59.5
258	-28.5
286	-0.5
228	-58.5
248	-38.5

The average launch time was 286.5ms. Once again, the trend continued of the first launch and sometimes second launch taking longer than in subsequent launches, but the difference is so minute it is hardly noticeable.

Conclusion

Writing in Kotlin was extremely pleasant because I simply viewed Java examples online, and copy and paste those examples into Android Studio, which has an automatic Java to Kotlin conversion tool. You are not sacrificing anything by using Kotlin as the launch time and APK size are similar to Java's it makes it much more appealing so there are seemingly no trade-offs from Java. Additionally, as lines convey the same amount of information with fewer statements, which makes it faster to write and easier to read, thus less soul crushing.

IDE

Android Studio 3.0 has full Kotlin support and even offers a Java-to-Kotlin conversion tool. (Kotlinlangorg, 2017) making Android Studio the primary IDE for Kotlin Android development.

Android studio also offers a conversion tool for Android-to-Java. (Androidcom, 2017)

Original Java

153: lines, 3905: characters

Original Java can be found at:

https://github.com/sardap/DynamicInstantCircumEngine/blob/master/Project/app/src/main/Java/club/psarda/dynamicinstantcircumengine/database/DataHolders/StatsDataJav.jav

Autogenerated Kotlin

Lines: 113, 3039: characters

```
class StatsData : Parcelable { //can implement the parceable with no issues

     var _name: String? = null     var _type: ChartType? = null     var

_sides: Long = 0     var _numberOfDice: Long = 0     var _rollData:

ArrayList<RollData>? = ArrayList()         private set //can only be set

privately but can be gotten publicly      var _timeCreation: Long = 0

private set



    constructor(name: String, type: ChartType, sides: Long, numberOfDice: Long, rollData:
ArrayList<RollData>) {

        …Nothing Special…

    }      constructor() {} //empty

constructor



    …Nothing Special…

     fun GetAverageRoll(): Double { //this was generated with two errors but they were easy to fix

         var sum: Double? = 0.0 //unneeded ? caused an error

var length: Double? = 0.0

         for (rolldata in _rollData!!)

            sum += UtilsClass.GetSum(rolldata._rolls).toDouble()

length += rolldata._totalRolls.toDouble()         }

         return sum!! /

length!!

    }
```

```
    protected constructor(`in`: Parcel)
{
        _name = `in`.readString()
        _type = `in`.readValue(ChartType::class.Java.classLoader) as ChartType
        _sides = `in`.readInt().toLong()
_numberOfDice = `in`.readInt().toLong()         if
(`in`.readByte().toInt() == 0x01) {
            _rollData = ArrayList()
            `in`.readList(_rollData, RollData::class.Java.classLoader)
        } else {
            _rollData = null
        }
    }      override fun writeToParcel(dest: Parcel, flags:
Int) {         dest.writeString(_name)
dest.writeValue(_type)         dest.writeLong(_sides)
dest.writeLong(_numberOfDice)         if (_rollData ==
null) {             dest.writeByte(0x00.toByte())
        } else
{             dest.writeByte(0x01.toByte())
dest.writeList(_rollData)
        }
    }      companion object { //Parcelable
creator code
         val CREATOR: Parcelable.Creator<StatsData> = object : Parcelable.Creator<StatsData>
{             override fun createFromParcel(`in`: Parcel): StatsData {                 return
StatsData(`in`)
            }
             override fun newArray(size: Int): Array<StatsData> { //Should be <StatsData?>
return arrayOfNulls(size)
            }
        }
    }
}
```

The full version can be found at:
https://github.com/sardap/DynamicInstantCircumEngine/blob/master/Project/app/src/main/Java/club/psarda/dynamicinstantcircumengine/database/DataHolders/StatsData.kt

The autogenerated version, with my comments, was 900 characters shorter and 40 less lines, and there was little clean-up related to the null. This is very impressive, considering it was autogenerated and meant I only needed to fix a couple of typing issues, which wasn't a huge issue.

Compatibility with Java

Kotlin will compile into Java Bytecode and is also 100% compatible with any pre-existing Java classes for Android development (Kotlinlangorg, 2017).

Out of interest and sickness of using Java in my custom program *Dynamic Instant Circum Engine* (DICE) I wrote any new classes in Kotlin to be used with my existing Java codebase.

The full Kotlin class can be found at:
https://github.com/sardap/DynamicInstantCircumEngine/blob/master/Project/app/src/main/Java/club/psarda/dynamicinstantcircumengine/database/DataHolders/SettingsData.kt

Abridged version below

```
public class SettingsData() {

     public var DefualtSides : Long =

6         get() = field

set(value){             if(value > 0)

field = value

        }

     public var DefualtReroll : Long =

0         get() = field

set(value){

            // Use of The amazing range operator

if(value in

0..DefualtSides){                 field = value

            }

        }

    ...      public var TargetsAsStrings : ArrayList<String> =

ArrayList()         get(){             val result =

ArrayList<String>()
```

```
            for(target in
Targets){                result.add(target.GetString())
            }
             return
result
        }


    ...
}
```

The previous advantages of using Kotlin over Java hold true with shorter, easier-to-write code.

Below is a Java code snippet where the SettingsData Kotlin class is used with methods which haven't been declared. If the Java class has a Getter and Setter for a member, it will automatically allow you to use that member as a property. For example, if I had get length and set length in my Java class and I was accessing that Java class in Kotlin, I could just type in `(object).length` = for set and `(object).length` which is easier and saves time.

```
private Bundle RollDefaultBundle(){
    DataStorage dataStorage = DataStorage.Companion.getINSTANCE();
    SettingsData settingsData = dataStorage.getSettings();
    Bundle bundle = new Bundle();     bundle.putString("sides",
Long.toString(settingsData.getDefualtSides())); // Autogenerated Method
bundle.putString("NumberOfDice", Long.toString(settingsData.getDefualtNumberOfDice()));
bundle.putStringArrayList("targets", settingsData.getTargetsAsStrings());     return bundle;
}
```

Cross-Platform Development

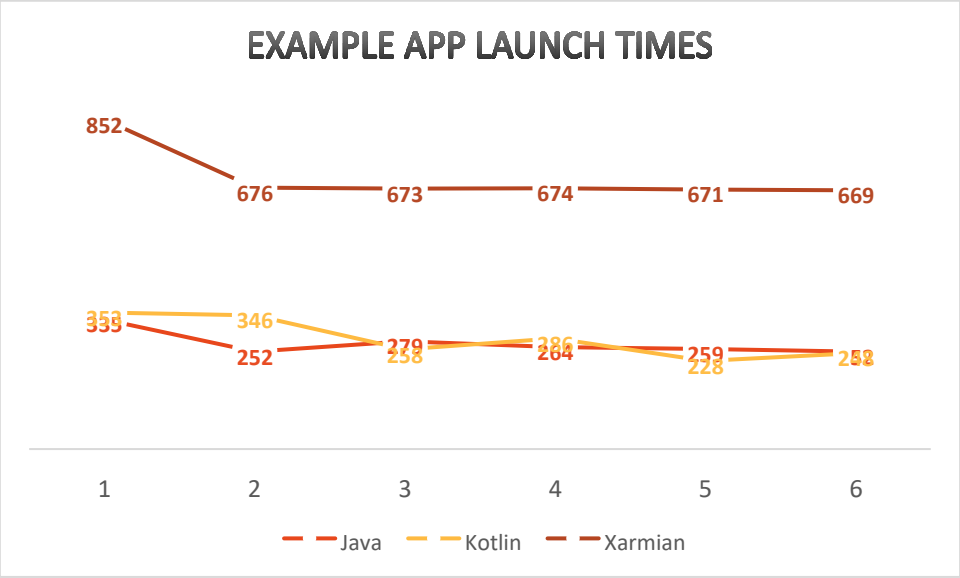
Cross-platform development for Kotlin, like almost everything about Kotlin, is the same with Java and has no cross compilers or options for any other mobile platforms.

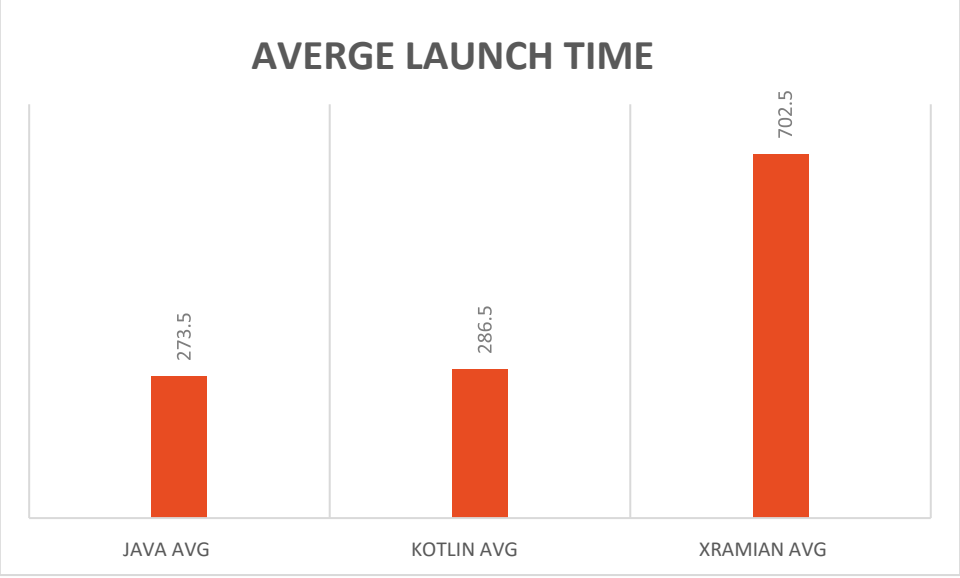
The method for cross-platform development is to split the application into multiple parts with a common library, which is Android or IOS specific.

Conclusion

The researched I conducted to create this Report allowed me to reach my primary objective of determining how viable alternatives to Java are for writing Android applications.

Prior to this research, the knowledge I had regarding Java alternatives for developing Android applications was zero. Through this process I now understand how Android applications run on Android, as well as how cross-platform development works. I also have discovered the advantages and disadvantages of Kotlin and C#/Xamarin, which will help me in future Programming. What really stood out for me was the launch time between Java and Kotlin, which were minute so not making any difference, with Xamarin taking much longer to launch, which is clearly a concern. The following tables demonstrate my findings





Summary Table

Platform	IDE	Average launch time	APK Size
Java	Android studio	273.5	5.04MB
C#/Xarmian	Visual Studio	702.5	15.78MB
Kotlin	Android studio	286.5	5.95MB

My other findings included:

- Kotlin:
    - Writing in Kotlin was extremely pleasant due to all the online resources including an automatic Java-to-Kotlin conversion tool;
    - There is no sacrifice using the platform, as the launch time and APK sizes are similar to Java's; ○ Lines convey the same amount of information with fewer statements, which makes it faster to write and easier to read, thus less soul crushing.
- C#/Xamarin
    - Has consistently shorter and fewer lines with utilising Getters and Setters; ○ The limited online resources available (due to the smaller community) is an issue, making it difficult to start-off if you don't already understand what an array adapter should look like;
    - It has an amazingly larger APK size, which means that it is a bigger program that takes longer to download, leaves a larger footprint on the phone, which means it will probably be deleted when the phone is running out of memory.

In the future when having to develop an Android application I will most likely utilise Kotlin over Xamarin or Java, or even extend my knowledge and look at other platforms that may prove to be a better programming solution.

Reference List

Xamarincom. 2017. *Xamarin.* [Online]. [20 September 2017]. Available from: https://www.xamarin.com/getting-started/android

Xamarincom. 2017. *Xamarin.* [Online]. [20 September 2017]. Available from: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1__understanding_the_xamarin_mobile_platform/

Xamarin. 2017. *Java Integration Overview.* [Online]. [8 October 2017]. Available from: https://developer.xamarin.com/guides/android/advanced_topics/Java_integration_overview/

Xamarin. 2017. *Binding a Java Library.* [Online]. [8 October 2017]. Available from: https://developer.xamarin.com/guides/android/advanced_topics/binding-a-Java-library/

Xamarin. 2017. *Architecture.* [Online]. [6 October 2017]. Available from: https://developer.xamarin.com/guides/android/under_the_hood/architecture/

Xamarin. 2017. *Porting Java to C#.* [Online]. [8 October 2017]. Available from: https://developer.xamarin.com/guides/android/advanced_topics/Java_integration_overview/porting_Java_to_csharp/

Xamarin. 2017. *Working With JNI.* [Online]. [15 October 2017]. Available from: https://developer.xamarin.com/guides/android/advanced_topics/Java_integration_overview/working_with_jni/

Xamarin. 2017. *Introduction to Android Wear.* [Online]. [13 October 2017]. Available from: https://developer.xamarin.com/guides/android/wear/intro-to-wear/

Xamarin. 2017. *Building Cross Platform Applications Overview.* [Online]. [13 October 2017]. Available from: https://developer.xamarin.com/guides/crossplatform/application_fundamentals/building_cross_platform_applications/part_0_-_overview/

Microsoftcom. 2016. *The Official Microsoft Blog.* [Online]. [20 September 2017]. Available from: https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-moredevelopers-to-build-apps-on-any-device/

Jetbrainscom. 2017. *Jetbrainscom.* [Online]. [23 September 2017]. Available from: https://blog.jetbrains.com/kotlin/2011/07/

Kotlinlangorg. 2017. *Kotlin.* [Online]. [23 September 2017]. Available from: https://kotlinlang.org/docs/reference/faq.html

Kotlinlangorg. 2017. *Kotlin.* [Online]. [23 September 2017]. Available from: https://kotlinlang.org/docs/tutorials/kotlin-android.html

Androidcom. 2017. *Androidcom.* [Online]. [26 September 2017]. Available from: https://developer.android.com/kotlin/get-started.html

Appendix

Appendix A Java Example Program

Appendix A.1 Main activity

Example Program Java: Main activity

```
package paulsarda.Javaexmaple;
 import
android.support.v7.app.AppCompatActivity; import
android.os.Bundle; import android.view.View;
import android.widget.ListAdapter; import
android.widget.ListView;
 import
Java.util.ArrayList; import
Java.util.List;
 public class MainActivity extends AppCompatActivity
{
         private class ViewHolder{         public
ListView listView; // Not Null Safe
                 public ViewHolder(View view){             listView
= (ListView) view.findViewById(R.id.list_view);
        }
    }          private ViewHolder
_viewHolder;


    @Override     protected void onCreate(Bundle
savedInstanceState)
{         super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);


        _viewHolder = new ViewHolder(getWindow().getDecorView().getRootView());
        InitliseListView();
    }          private void
InitliseListView(){
        List<ListRowData> listData = new ArrayList<>();
                 listData.add(new ListRowData("One Cool beans", "subtitle beans
one"));         listData.add(new ListRowData("Two Cool beans", "subtitle beans
three"));          listData.add(new ListRowData("Three Cool beans", "subtitle beans
nighty"));          listData.add(new ListRowData("Four Cool beans", "subtitle beans
beanos"));
```

```
        ListAdapter listAdapter = new ListAdapterExmaple(this, R.layout.list_row, listData);
        _viewHolder.listView.setAdapter(listAdapter);
    }
}
```

Appendix A.2 ListRowHolder

Example Program Java: ListRowHolder

```
package paulsarda.Javaexmaple;

 import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

 import

org.w3c.dom.Text;


/**

 * Created by pfsar on 24/09/2017.

 */  public class

ListRowHolder {

     public TextView title;

public TextView subTitle;

public Button button;

public EditText input;

         public ListRowHolder(View row){         title =

(TextView) row.findViewById(R.id.title);         subTitle =

(TextView) row.findViewById(R.id.sub_title);          button =

(Button)row.findViewById(R.id.cool_button);         input =

(EditText)row.findViewById(R.id.input);

    }


}
```

A.3 ListRowData

Example Program Java: ListRowData

```
package paulsarda.Javaexmaple; public class

ListRowData {

         private String

_title;     private String

_subtitle;

         public ListRowData(String title, String

subTitle){
```

```
        SetTitle(title);
        SetSubTitle(subTitle);
    }          public String
GetTitle(){         return
_title;
    }          public String
GetSubtitle(){         return
_subtitle;
    }          public void
SetTitle(String
value){         if(value.length() > 4){
            _title = value;
        }
    }          public void
SetSubTitle(String
value){         if(value != _title){
            _subtitle = value;
        }
    }
}
```

A.4 ListAdapterExample

Example Program Java: ListAdatperExample

```
package paulsarda.Javaexmaple;
 import android.content.Context;
import android.view.LayoutInflater;
import android.view.View; import
android.view.ViewGroup; import
android.widget.ArrayAdapter; import
android.widget.BaseAdapter; import
android.widget.Toast;
 import
Java.util.ArrayList; import
Java.util.List;


/**
```

```
 * Created by pfsar on 24/09/2017.
 */  public class ListAdapterExmaple extends
ArrayAdapter<ListRowData> {
         private LayoutInflater
_inflater;     private
List<ListRowData> _data;     private
Context _contex;
         public ListAdapterExmaple(Context mContext, int resource, List<ListRowData>
data) {
                 super(mContext,
resource, data);


        _data = data;
        _contex = mContext;
        _inflater = LayoutInflater.from(_contex);
    }


    @Override     public View getView(final int position, View convertView,
ViewGroup parent) {
        View view;
final ListRowHolder vh;
         if (convertView == null) {             view =
LayoutInflater.from(getContext()).inflate(R.layout.list_row, parent, false);             vh =
new ListRowHolder(view);             view.setTag(vh);
        }else{             view = convertView;
vh = (ListRowHolder)view.getTag();
        }
vh.title.setText(_data.get(position).GetTitle());
vh.subTitle.setText(_data.get(position).GetSubtitle());
                 vh.button.setOnClickListener(new
View.OnClickListener() {
            @Override             public
void onClick(View view) {
                Toast.makeText(_contex, _contex.getString(R.string.popup), Toast.LENGTH_SHORT).show();
// This Line is the exact same
```

```
                _data.get(position).SetSubTitle(vh.input.getText().toString());
notifyDataSetChanged();
            }
        });
return view;
    }
}
```

Appendix B Example Program C#

B.1 Main Activity

```
using Android.App; using
Android.Widget; using Android.OS;
using Android.Views; using
System.Collections.Generic;
 namespace ExampleProgram.src
{
    [Activity(Label = "ExampleProgram", MainLauncher = true)]
public class MainActivity : Activity
    {         private class
ViewHolder
        {             public
ViewHolder(View view)
            {
                ListView = view.FindViewById<ListView>(Resource.Id.list_view);
            }
             public ListView ListView


{                 get;
set;
            }


        }
         private ViewHolder _viewHolder;
```

```
        protected override void OnCreate(Bundle savedInstanceState)

{             base.OnCreate(savedInstanceState);

            // uses Resouces instend of R
            SetContentView(Resource.Layout.Main);

            // Same as Java just no get
            _viewHolder = new ViewHolder(Window.DecorView.RootView);
            InisliseListView();
        }
         private void InisliseListView()
        {
            List<ListRowData> listData = new List<ListRowData>
            {                 new ListRowData("One Cool beans", "subtitle beans one"),
new ListRowData("Two Cool beans", "subtitle beans three"),                 new
ListRowData("Three Cool beans", "subtitle beans nighty"),                 new
ListRowData("Four Cool beans", "subtitle beans beanos")             };

            // C# naming convesions
            IListAdapter listAdapter = new ListAdapterExmaple(this, Resource.Layout.list_row, listData);

            _viewHolder.ListView.Adapter = listAdapter;
        }
    }
}
```

B.2 ListRowHolder

```
using System; using
System.Collections.Generic; using
System.Linq; using System.Text;
 using Android.App;
using Android.Content;
using Android.OS; using
Android.Runtime; using
Android.Views; using
Android.Widget;
 namespace ExampleProgram.src
{
    // Must Inherit from Java.Lang.Object to be used by
    // many Andriod methods     public class
ListRowHolder : Java.Lang.Object
    {         public ListRowHolder(View
view)
        {
            Title = view.FindViewById<TextView>(Resource.Id.title);
            SubTitle = view.FindViewById<TextView>(Resource.Id.sub_title);
            Button = view.FindViewById<Button>(Resource.Id.cool_button);
            Input = view.FindViewById<EditText>(Resource.Id.input);
        }
         public TextView Title

{             get;
set;
        }
         public TextView SubTitle

{             get;
set;
        }
         public Button Button
```

```
{             get;
set;
        }
         public EditText Input

{             get;
set;
        }
    }
}
```

B.3 ListRowData

```
namespace ExampleProgram.src
{     public class
ListRowData
    {         private string
_title;         private string
_subTitle;
         public ListRowData(string title, string
subTitle)
        {
            _title = Title;
            _subTitle = subTitle;
        }
         public string
Title
        {             get
{                 return
_title;
            }             set
{                 if(value.Length >
4)
                {
                    _title = value;
                }
            }
        }
         public string
SubTitle
        {             get
{                 return
_subTitle;
            }             set
{                 if(value !=
Title)
                {
                    _subTitle = value;
                }
```

```
            }
        }


    }
}
```

B.4 ListAdatperExample

```
using System; using
System.Collections.Generic; using
System.Linq; using System.Text;
 using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
 namespace
ExampleProgram.src
{     public class ListAdapterExmaple :
ArrayAdapter<ListRowData>     {          private LayoutInflater
_inflater;         private List<ListRowData> _data;
private Context _context;


        // Base is same as super
        public ListAdapterExmaple(Context context, int resource, List<ListRowData> data) :
base(context, resource, data)
        {
            _data = data;
            _context = context;
            _inflater = LayoutInflater.From(_context);
        }


        // Final is imposible in C# for method          public override View
GetView(int position, View convertView, ViewGroup parent)
        {
            View view;
            ListRowHolder vh;
             if(convertView ==
null)
            {                 view = _inflater.Inflate(Resource.Layout.list_row,
parent, false);
```

```
                vh = new ListRowHolder(view);
                // Must be a Java Object
                // Tags Must have id's which is a pain for
                // only a single tag
view.SetTag(Resource.Id.TAG_VIEW_HOLDER, vh);
            }             else             {                 view =
convertView;                vh = view.GetTag(Resource.Id.TAG_VIEW_HOLDER) as
ListRowHolder;             }
             vh.Title.Text = _data[position].Title;
vh.SubTitle.Text = _data[position].SubTitle;
            // C# style events
vh.Button.Click += (sender, e) => {
                // Simlar expect ToastLength is a seprate enum insted of being a const in Toast
                Toast.MakeText(_context, _context.GetString(Resource.String.popup),
ToastLength.Long).Show();
                _data[position].SubTitle = vh.Input.Text.ToString();
                NotifyDataSetChanged();
            };
             return
view;
        }
    }
}
```

Appendix C Kotlin Example Program

C.1 Main Activity

```
package paulsarda.kotlinexample
 import
android.support.v7.app.AppCompatActivity
import android.os.Bundle import
android.view.View import
android.widget.BaseAdapter import
android.widget.ListAdapter import
android.widget.ListView
 class MainActivity : AppCompatActivity()
{
```

```
     private class ViewHolder(view : View){         var listView :
ListView = view.findViewById(R.id.list_view)     }


    // ? means that it can be null     private
var _viewHolder : ViewHolder? = null
     override fun onCreate(savedInstanceState: Bundle?)
{         super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
        _viewHolder = ViewHolder(window.decorView.rootView)
        InitliseListView()
    }      private fun InitliseListView(){         // No new opreator in
kotlin         val listData : MutableList<ListRowData> =
mutableListOf<ListRowData>()
         listData.add(ListRowData("One Cool beans", "subtitle beans one"))
listData.add(ListRowData("Two Cool beans", "subtitle beans three"))
listData.add(ListRowData("Three Cool beans", "subtitle beans nighty"))
listData.add(ListRowData("Four Cool beans", "subtitle beans beanos"))
         val listAdapter : BaseAdapter
=
            ListExampleAdapter(this.applicationContext, R.layout.list_row, listData)
        // Becuase Viewholder can be null ?. must be made to make a safe call
        // if _viewHolder is null ViewHolder? will be thorwn
        _viewHolder?.listView?.adapter = listAdapter
    }
}
```

C.2 List Row Holder

```
package paulsarda.kotlinexample
 import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView



/**
```

```
 * Created by pfsar on 23/09/2017.
 */


// row is null safe public class ListRowHolder(row: View) {     public val title : TextView =
row.findViewById<TextView>(R.id.title)     public val subTitle: TextView =
row.findViewById<TextView>(R.id.sub_title)     public val button : Button =
row.findViewById<Button>(R.id.cool_button)     public val input : EditText =
row.findViewById<EditText>(R.id.input)
}
```

C.3 List Row Data

```
package paulsarda.kotlinexample


/**
 * Created by pfsar on 24/09/2017.
 */


//Constutor agumentes in class delecation public class
ListRowData(title : String, subTitle : String){
     public var Title : String =
title         get() = field
        // Values type does not have to be declared but value must be a
string         set(value) {              if(value.length >
4){                 field = value
            }
        }
     public var SubTitle : String =
subTitle         get() = field
set(value) {             if(value !=
Title){                 field = value
            }
        }
}
```

C.4 List Example Adapter

```
package paulsarda.kotlinexample

 import android.content.Context

import android.view.LayoutInflater

import android.view.View import

android.view.ViewGroup import

android.widget.ArrayAdapter import

android.widget.Toast


/**

 * Created by pfsar on 23/09/2017.

 */  public class ListExampleAdapter(context: Context, resourceId: Int, data :

MutableList<ListRowData>) :

    // subclass constuntor is called here

    ArrayAdapter<ListRowData>(context, resourceId, data) {


    // No constuntor is needed here     private val

_inflator = LayoutInflater.from(context)     private val

_data = data     private val _contex = context

     override fun getView(position: Int, convertView: View?, parent: ViewGroup): View?

{         val view: View?

        val vh: ListRowHolder

         if (convertView == null) {             view =

this._inflator.inflate(R.layout.list_row, parent, false)             vh =

ListRowHolder(view)             view.tag = vh

        } else {             view =

convertView             vh = view.tag as

ListRowHolder

        }


        // Text for text view is a simple property

vh.title.text = _data[position].Title         //

Access to List's can be done though []

vh.subTitle.text = _data[position].SubTitle
```

```
        // Much shorter than in Java
vh.button.setOnClickListener{


            // Line is exact same as in Java
            Toast.makeText(
                _contex,
                _contex.getString(R.string.popup),
                Toast.LENGTH_SHORT).show()


            _data[position].SubTitle = vh.input.text.toString()
            // Line is exact same as in Java
notifyDataSetChanged()
        }
         return view
    }
}
```