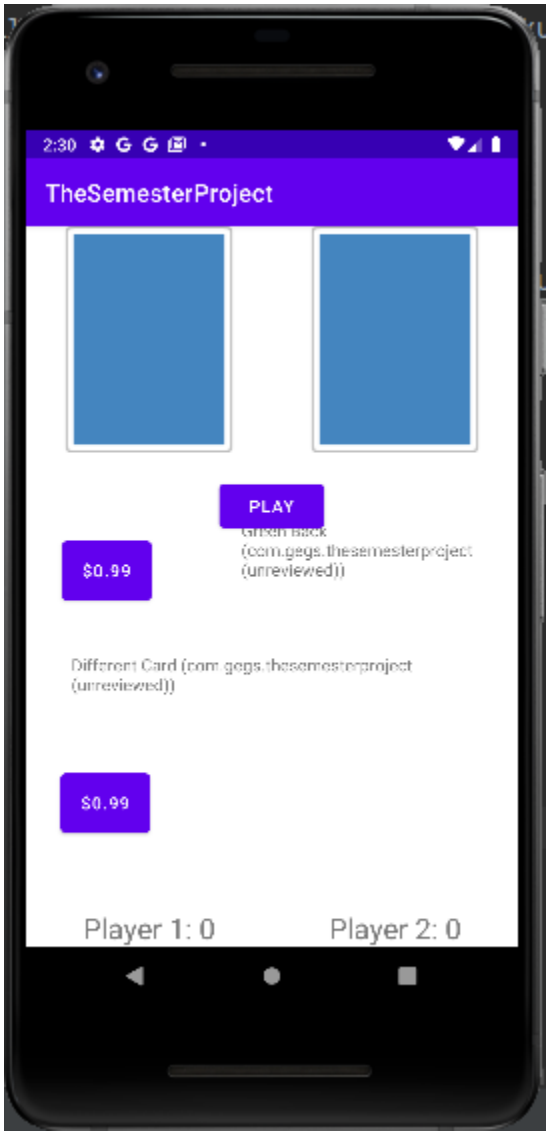
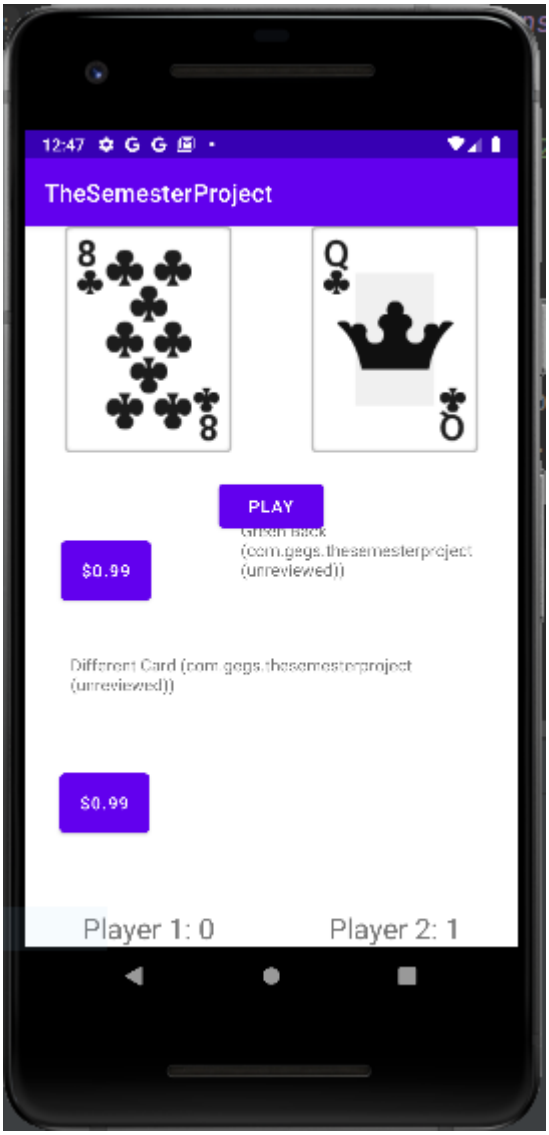
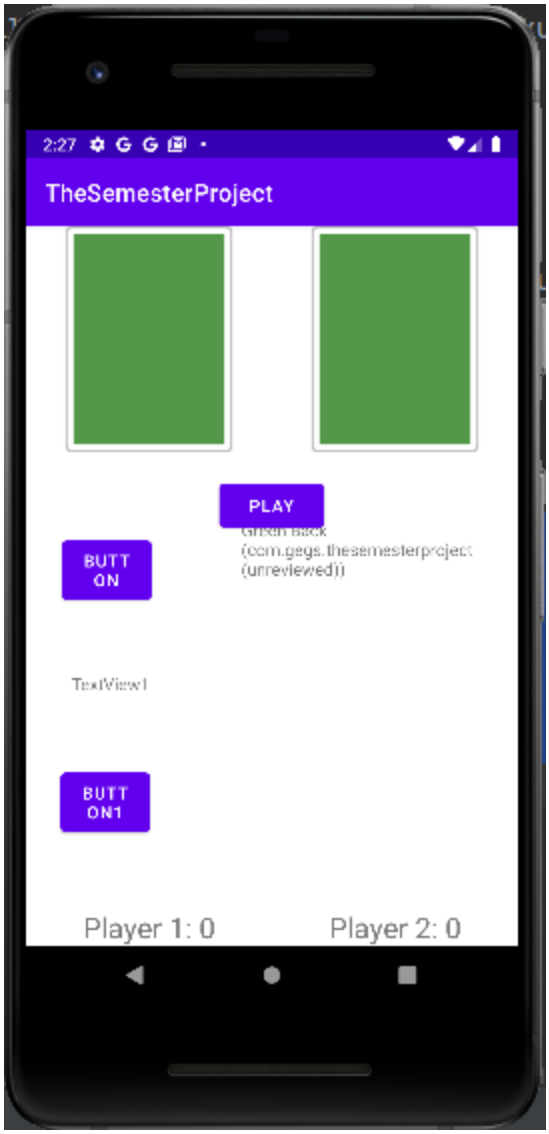
Tutorial for implementing the Google Play Billing Library

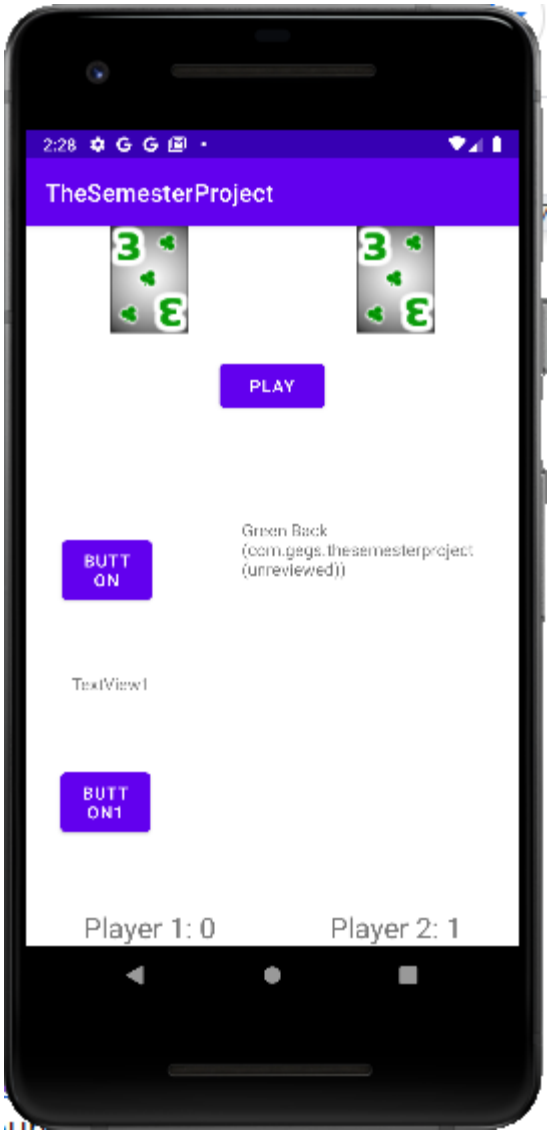
Overview

The Google Play Billing Library allows creators to sell products inside of their apps. This includes things like one-time products that can only be purchased once and will persist forever after. Things like premium versions or cosmetic upgrades/changes. There are also consumable products, like in-game currency or boosters, that a user can purchase multiple times. Finally, the google play billing library allows users to set up and sell subscriptions in their apps as well, including billing periods, free trials, and even grace periods. This tutorial will focus on one-time products that will alter the appearance of the app. The app in question is a simple version of the card game war. Since this tutorial is about the billing library, it will focus more on those aspects rather than the game creation.









Users can play a simple game of war where when a player's points on the bottom will increase each time they win a round. The two price buttons will open up a billing window and allow the user to purchase a product based on the button they pressed. The top button changes the color of the backs of the cards, while the second button changes the appearance of the face of the cards.

Getting Started

To fully follow this tutorial you must have Android Studio Arctic Fox for your development environment. To use the Google Play Billing library for in-app purchases, you will need to have a Google Play development account and use the Google Play Console. To sell paid apps and in-app purchases on Google Play, you must also set up a profile in the Google Payments Center and then link that profile to your Google Play developer account. A link for how to

link those is provided here, Link a Google Play developer account to your payments profile.

For any additional help you may need, you can access the android developers page for the billing library. There is lots of helpful information there for what is being covered in this tutorial.

To integrate the billing system you will need to add this dependency to your Android Studio project in your build grade file:

```
dependencies {
    val billing_version = "4.0.0"

    implementation("com.android.billingclient:billing:$billing_version")
}
```

Once that is added you will then need to upload your app using the Google Play Console. For this step, create your app and then publish it to any track, including the internal test track. After uploading your app you will need to create the products for the app using the Google Play Console. To create and configure the one-time products, see Create a managed product.

Step by Step Instructions

Before we begin it would be helpful to understand the basic life of a purchase which looks like this:

- Show the user what they can buy.
- Launch the purchase flow for the user to accept the purchase.
- Verify the purchase on your server. (if you want one)
- Give content to the user, and acknowledge delivery of the content. Optionally, mark the item as consumed so that the user can buy the item again.

For this tutorial, we will start off with the game itself. You should create a new project in Android Studio starting with an empty activity. All it requires is a button to play the cards, images that display the cards, textviews that show the players and their scores, and the buttons and textviews for the products that can be purchased. You can add them using the design view, or you can simply put them in by coding through your activity_main.xml file. If you choose to code it in the activity_main.xml file you can use the GitHub file from the repo found at the bottom of this tutorial as a reference when

creating all of those elements. You can constrain them on the screen however you like, but the image from above is how I implemented the design. You can choose whatever images you would like for the card backs and card fronts, just make sure to save the image files in the drawable file of the project directory.

The remainder of this tutorial will be handling code in the Main_Activity.kt file of the project.

Firstly, we will set up the billingClient for the app as well as the variables that will reference the elements we have added to the layout of the app.

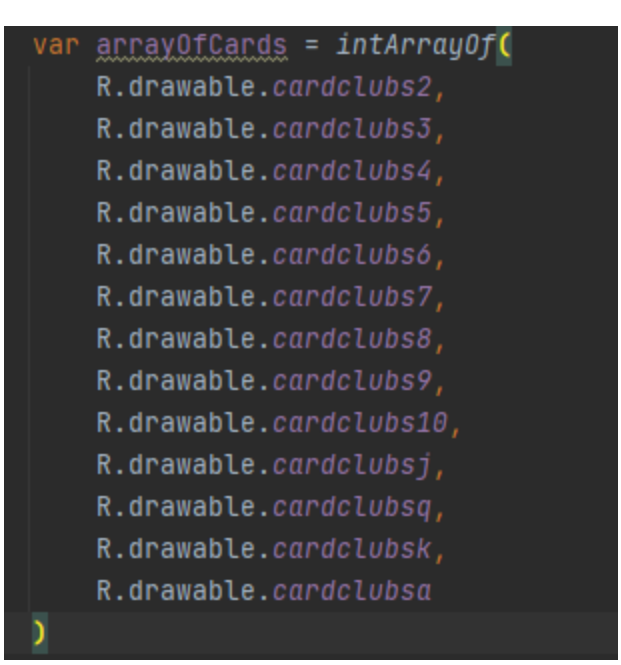
```
class MainActivity : AppCompatActivity() {
    lateinit var billingClient: BillingClient
    lateinit var iv_card1: ImageView
    lateinit var iv_card2: ImageView

    lateinit var tv_player1: TextView
    lateinit var tv_player2: TextView
    lateinit var b_deal: Button
```

Next, we will create a random variable that will be used to draw cards, and then we will create the numbers that will represent each of the cards.

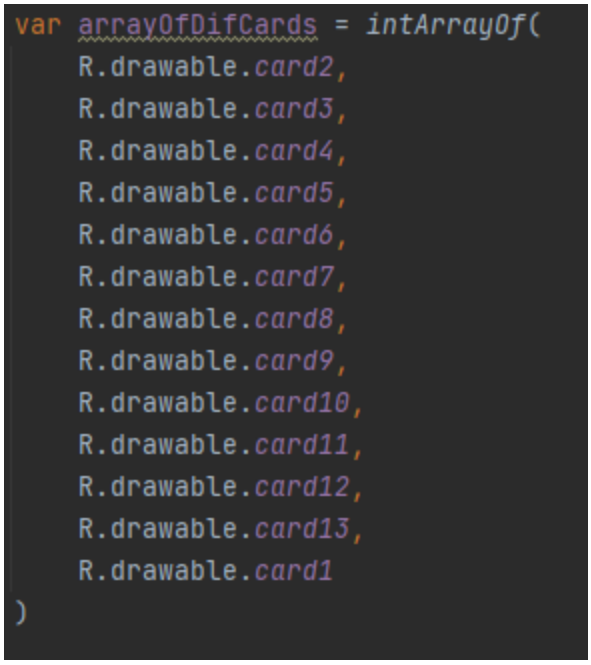
```
lateinit var random: Random

var card1: Int = 0
var card2: Int = 0

var player1 = 0
var player2 = 0

var cardColor: String = "reg"
```

The cardColor string will be used for altering the color of the cards when a purchase is confirmed later.

Next, we need to initialize two arrays that will hold the images of the cards that we will use for the apps.



```
var arrayOfCards = intArrayOf(
    R.drawable.cardclubs2,
    R.drawable.cardclubs3,
    R.drawable.cardclubs4,
    R.drawable.cardclubs5,
    R.drawable.cardclubs6,
    R.drawable.cardclubs7,
    R.drawable.cardclubs8,
    R.drawable.cardclubs9,
    R.drawable.cardclubs10,
    R.drawable.cardclubsj,
    R.drawable.cardclubsq,
    R.drawable.cardclubsk,
    R.drawable.cardclubsa
)
```



```
var arrayOfDifCards = intArrayOf(
    R.drawable.card2,
    R.drawable.card3,
    R.drawable.card4,
    R.drawable.card5,
    R.drawable.card6,
    R.drawable.card7,
    R.drawable.card8,
    R.drawable.card9,
    R.drawable.card10,
    R.drawable.card11,
    R.drawable.card12,
    R.drawable.card13,
    R.drawable.card1
)
```

Next, we will override the onCreate function in order to create the content view, bind the elements in the view to the variables we have created, and initialize the billingClient that will be used by the app.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    random = Random

    iv_card1 = findViewById(R.id.iv_card1)
    iv_card2 = findViewById(R.id.iv_card2)

    iv_card1.setImageResource(R.drawable.cardbackblue1)
    iv_card2.setImageResource(R.drawable.cardbackblue1)

    tv_player1 = findViewById(R.id.tv_player1)
    tv_player2 = findViewById(R.id.tv_player2)

    b_deal = findViewById(R.id.b_deal)
    b_deal.setOnClickListener {
      card1 = random.nextInt(arrayOfCards.size)
```

```
            card2 = random.nextInt(arrayOfCards.size)

            setCardImage(card1, iv_card1)
            setCardImage(card2, iv_card2)

            if (card1 > card2) {
                player1++
                tv_player1.text = "Player 1: $player1"
            } else {
                player2++
                tv_player2.text = "Player 2: $player2"
            }
        }

        val purchaseUpdateListener =
            PurchasesUpdatedListener { billingResult, purchases ->
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK &&
purchases != null) {
                    for (purchase in purchases) {
                        handlePurchase(purchase)
                        //testConsumePurchase(purchase)
                    }
                } else if (billingResult.responseCode ==
BillingClient.BillingResponseCode.USER_CANCELED) {
                    // Handle an error caused by a user cancelling the purchase flow.
                } else {
                    // Handle any other error codes.
                }
            }

        billingClient = BillingClient.newBuilder(this)
            .enablePendingPurchases().setListener(purchaseUpdateListener)
            .enablePendingPurchases().build()

        connectToGooglePlayBilling()
    }
```

Next, we will set up the connectToGooglePlayBilling function in order to connect to google play billing, there is also a function called queryPurchasesAsync which is from

the google play billing library that checks the purchases a user has made in the app. We then use the list that that function returns to change the variables that are responsible for the appearances of the cards.

```
fun connectToGooglePlayBilling(){
        billingClient.startConnection(object : BillingClientStateListener {
            override fun onBillingSetupFinished(billingResult: BillingResult) {
                if (billingResult.responseCode ==  BillingClient.BillingResponseCode.OK) {
                    // The BillingClient is ready. You can query purchases here.
                    getProductDetails()//Retrieve product details that were made in googleplayconsole

billingClient.queryPurchasesAsync(BillingClient.SkuType.INAPP){billingResult,
purchases->//This function is what checks what the user has purchased if anything
                        if(billingResult.responseCode == BillingClient.BillingResponseCode.OK){
                            if(purchases != null){
                                for(item in purchases){
                                    if(item.skus.contains("green_back")){//If they have purchased this
                                        Log.d("contains green","green Contained")
                                        changeBacks()
                                    }
                                    if(item.skus.contains("differe_card")){//If they have purchased this
                                        Log.d("contains diff","diff Contained")
                                        cardColor = "diff"
                                        changeCards()
                                    }
                                }

                            }
                        }else{
                            Log.d("Non-ok: ", billingResult.responseCode.toString())
                            }
                    }
                }
            }
            override fun onBillingServiceDisconnected() {
                // Try to restart the connection on the next request to
                // Google Play by calling the startConnection() method.
                connectToGooglePlayBilling()
            }
```

```
        })
    }
```

Next, we will create the getProductDetails function which is what checks the products from the google play console that are available for the app and then assigns the appropriate information from those products to the buttons and textviews of the products.

```
fun getProductDetails(){
        val productIds = arrayListOf<String>()
        productIds.add("green_back")
        productIds.add("differe_card")

        val skuList = ArrayList<String>()
        skuList.add("green_back")
        skuList.add("differe_card")
        val params = SkuDetailsParams.newBuilder()//Create a list that the billingClient can use
        params.setSkusList(skuList).setType(BillingClient.SkuType.INAPP)


        var activity: Activity = this
        billingClient?.querySkuDetailsAsync(params.build()) { billingResult, skuDetailsList ->
            if( billingResult.responseCode == BillingClient.BillingResponseCode.OK &&
skuDetailsList != null){//This is where the product's buttons and text is created
                var greenTextView: TextView = findViewById(R.id.greenText)
                var greenButton: Button = findViewById(R.id.greenButton)
                var greenItemInfo: SkuDetails = skuDetailsList[1]
                var changeTheTextView: TextView = findViewById(R.id.changeText)
                var changeTheButton: Button = findViewById(R.id.changeButton)
                var changeItemInfo: SkuDetails = skuDetailsList[0]
                greenTextView.text = greenItemInfo.title
                greenButton.text = greenItemInfo.price
                changeTheTextView.text = changeItemInfo.title
                changeTheButton.text = changeItemInfo.price

                greenButton.setOnClickListener {//Listener for the purchase buttons
                    billingClient.launchBillingFlow(//Function that will start the purchase popup window
                        activity,
                        BillingFlowParams.newBuilder().setSkuDetails(greenItemInfo).build()
```

```
                )
            }
            changeTheButton.setOnClickListener {
                billingClient.launchBillingFlow(                    activity,
                    BillingFlowParams.newBuilder().setSkuDetails(changeItemInfo).build()
                )
            }
        }
    }

}
```

In order for our users to have access to the products, we must acknowledge the purchases that the user has made, we do that with the handlePurchases function as shown below. If a purchase is not acknowledged it will be automatically refunded.

```
fun handlePurchase(purchase: Purchase) {
    if (purchase.purchaseState == Purchase.PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {//If it is purchased but has not been acknowledged
            val acknowledgePurchaseParams =
AcknowledgePurchaseParams.newBuilder()
                .setPurchaseToken(purchase.purchaseToken).build()
            billingClient?.acknowledgePurchase(acknowledgePurchaseParams) {
billingResult ->//Acknowledge the purchase
                val billingResponseCode = billingResult.responseCode
                val billingDebugMessage = billingResult.debugMessage
            }
        }
    }
}
```

We will use two functions that are called when the purchases have been made, and these functions will change the appearance of the cards. The two functions are changeCards, and changeBacks as seen below:

```
fun changeCards() {
    var x: Int = 0
    if (cardColor != "reg") {
        while (x < arrayOfCards.size) {
            arrayOfCards[x] = arrayOfDifCards[x]
            x += 1
```

```
        }
    }
}

fun changeBacks(){
    iv_card1.setImageResource(R.drawable.cardbackgreen1)
    iv_card2.setImageResource(R.drawable.cardbackgreen1)
}
```

In order to actually change the front of the cards we must create the function below that is used when the start button is pressed:

```
private fun setCardImage(number: Int, image: ImageView){
    if(cardColor == "reg") {
        image.setImageResource(arrayOfCards[number])
    }else{
        image.setImageResource(arrayOfDifCards[number])
    }
}
```

Your app should now be functioning and should handle the purchases that a user may make when using your app.

Conclusion

The google play billing library is a great tool that can be utilized for mobile application development. If you are trying to make an app that will have in-app purchases or subscriptions(more info. on subscriptions), it is very beneficial to set up the google play console and use this library as well. Although it takes a lot to set it up and can be tricky to get the hang of, the google play billing library has all the tools you need to sell your digital products inside of your apps.

Overall I believe that this project was very relevant for mobile application development as any developer who is making any app with hopes of generating revenue would find this library helpful if they develop for android. As aforementioned, the library may start off tricky to understand, but once you have a grasp on how to use it, it is very useful and can be utilized to great effect for adding some monetization to a mobile application.

A link to the full GitHub project can be found here.