**Android In-App Billing v3 Library**

This is a simple, straight-forward implementation of the Android v3 In-app billing API.

It supports: In-App Product Purchases (both non-consumable and consumable) and Subscriptions.

**Getting Started**

* You project should build against Android 2.2 SDK at least.
* Add this *Android In-App Billing v3 Library* to your project:
  + If you guys are using Eclipse, download latest jar version from the [releases](https://github.com/anjlab/android-inapp-billing-v3/releases) section of this repository and add it as a dependency
  + If you guys are using Android Studio and Gradle, add this to you build.gradle file:

repositories {

mavenCentral()

}

dependencies {

compile 'com.anjlab.android.iab.v3:library:1.0.+'

}

* Open the *AndroidManifest.xml* of your application and add this permission:

<uses-permission android:name="com.android.vending.BILLING" />

* Create instance of BillingProcessor class and implement callback in your Activity source code. Constructor will take 3 parameters:
  + **Context**
  + **Your License Key from Google Developer console.** This will be used to verify purchase signatures. You can pass NULL if you would like to skip this check (*You can find your key in Google Play Console -> Your App Name -> Services & APIs*)
  + **IBillingHandler Interface implementation to handle purchase results and errors** (see below)

public class SomeActivity extends Activity implements BillingProcessor.IBillingHandler {

BillingProcessor bp;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

bp = new BillingProcessor(this, "YOUR LICENSE KEY FROM GOOGLE PLAY CONSOLE HERE", this);

}

// IBillingHandler implementation

@Override

public void onBillingInitialized() {

/\*

\* Called when BillingProcessor was initialized and it's ready to purchase

\*/

}

@Override

public void onProductPurchased(String productId, TransactionDetails details) {

/\*

\* Called when requested PRODUCT ID was successfully purchased

\*/

}

@Override

public void onBillingError(int errorCode, Throwable error) {

/\*

\* Called when some error occurred. See Constants class for more details

\*

\* Note - this includes handling the case where the user canceled the buy dialog:

\* errorCode = Constants.BILLING\_RESPONSE\_RESULT\_USER\_CANCELED

\*/

}

@Override

public void onPurchaseHistoryRestored() {

/\*

\* Called when purchase history was restored and the list of all owned PRODUCT ID's

\* was loaded from Google Play

\*/

}

}

* override Activity's onActivityResult method:

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (!bp.handleActivityResult(requestCode, resultCode, data))

super.onActivityResult(requestCode, resultCode, data);

}

* Call purchase method for a BillingProcessor instance to initiate purchase or subscribe to initiate a subscription:

bp.purchase(YOUR\_ACTIVITY, "YOUR PRODUCT ID FROM GOOGLE PLAY CONSOLE HERE");

bp.subscribe(YOUR\_ACTIVITY, "YOUR SUBSCRIPTION ID FROM GOOGLE PLAY CONSOLE HERE");

* **That's it! A super small and fast in-app library ever!**
* **And dont forget** to release your BillingProcessor instance!

@Override

public void onDestroy() {

if (bp != null)

bp.release();

super.onDestroy();

}

**Check Play Market services availability**

Before any usage it's good practice to check in-app billing services availability. In some elder devices or chinese ones it may happen that Play Market is unavailable or is deprecated and doesn't support in-app billing.

Simply call static method BillingProcessor.isIabServiceAvailable():

boolean isAvailable = BillingProcessor.isIabServiceAvailable();

if(!isAvailable) {

// continue

}

**Consume Purchased Products**

You can always consume made purchase and allow to buy same product multiple times. To do this you need:

bp.consumePurchase("YOUR PRODUCT ID FROM GOOGLE PLAY CONSOLE HERE");

**Restore Purchases & Subscriptions**

bp.loadOwnedPurchasesFromGoogle();

**Notice On Canceled/Expired Subscriptions**

Since Google's v3 API doesn't provide any callbacks to handle canceled and/or expired subscriptions you have to handle it on your own. The easiest way to do this - call periodically bp.loadOwnedPurchasesFromGoogle() method.

**Getting Listing Details of Your Products**

To query listing price and a description of your product / subscription listed in Google Play use these methods:

bp.getPurchaseListingDetails("YOUR PRODUCT ID FROM GOOGLE PLAY CONSOLE HERE");

bp.getSubscriptionListingDetails("YOUR SUBSCRIPTION ID FROM GOOGLE PLAY CONSOLE HERE");

As a result you will get a SkuDetails object with the following info included:

public final String productId;

public final String title;

public final String description;

public final boolean isSubscription;

public final String currency;

public final Double priceValue;

public final String priceText;

To get info for multiple products / subscriptions on one query, just pass a list of product ids:

bp.getPurchaseListingDetails(arrayListOfProductIds);

bp.getSubscriptionListingDetails(arrayListOfProductIds);

where arrayListOfProductIds is a ArrayList<String> containing either IDs for products or subscriptions.

As a result you will get a List<SkuDetails> which contains objects described above.

**Getting Purchase Transaction Details**

As a part or 1.0.9 changes, TransactionDetails object is passed to onProductPurchased method of a handler class. However, you can always retrieve it later calling these methods:

bp.getPurchaseTransactionDetails("YOUR PRODUCT ID FROM GOOGLE PLAY CONSOLE HERE");

bp.getSubscriptionTransactionDetails("YOUR SUBSCRIPTION ID FROM GOOGLE PLAY CONSOLE HERE");

As a result you will get a TransactionDetails object with the following info included:

public final String productId;

public final String orderId;

public final String purchaseToken;

public final Date purchaseTime;

// containing the raw json string from google play and the signature to

// verify the purchase on your own server

public final PurchaseInfo purchaseInfo;