인앱결제 발목까지만 이해하기

이대로만 하면 인앱결제, 할 수는 있다!

zinc0214

STEP 1

1. 우리 모두가 아는 인앱결제에 대해 알아봅시다.

Google play 의 결제시스템 콘텐츠 유형

✓ 일회성 제품

- 반복되지 않는 단일 요금을 지급함으로써 구매할 수 있는 콘텐츠
- Google Play 결제 라이브러리에서 “InApp” 제품에 해당
- 소비성 : 인게임 화폐 등과 같이 앱 내 콘텐츠를 받기 위해 소비하는 제품, 여러번 구매 가능
- 비소비성 : 영구적인 혜택을 제공하는 제품. 프리미엄 업그레이드, 레벨 팩 등

✓ 정기 결제

- 반복적으로 콘텐츠에 대한 액세스를 제공하는 제품. 취소될 때 까지 자동으로 갱신됨

Google Play's billing system overview

1. 우리 모두가 아는 인앱결제에 대해 알아봅시다.

알아두면 좋은 Google play 결제시스템 용어

✓ Product SKU (제품 SKU)

- 각 제품 유형의 ID

✓ Product Token (구매 토큰)

- Google Play 에서 구매자의 자격을 나타내는 문자열.
- 해당 토큰이 생성되었다면 사용자가 어떠한 제품에 대한 비용을 지불했음을 의미한다.

✓ Order Id (주문 ID)

- Google Play 에서 금융 거래를 나타내는 문자열.
- 금융 거래가 발생할 때 마다 생성된다.
- 사용자에게 전송되는 이메일 영수증에 해당 정보가 포함되며 해당 ID 를 통해 환불 처리를 해줄 수도 있다.

Google Play billing-related terminology

STEP 2

1. 기본적인 것 부터 추가하기

app/build.gradle

```
implementation 'com.android.billingclient:billing:4.0.0'
```

AndroidManifest.xml

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

2. 인앱결제 시작하기

```
/**
 *  BillingClient 초기화
 */
private fun initBillingClient() {
    billingClient =
        BillingClient.newBuilder(this@MainActivity).enablePendingPurchases()
            .setListener(object : PurchasesUpdatedListener {
                override fun onPurchasesUpdated(
                    billingResult: BillingResult,
                    purchases: List<Purchase>?
                ) {
                    // TODO
                }
            })
            .build()
}
```

3. Google Play 와 연결하기

```
private fun connectToGooglePlay() {
    billingClient.startConnection(object : BillingClientStateListener {
        override fun onBillingSetupFinished(billingResult: BillingResult) {
            if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                querySkuDetails()
            }
        }

        override fun onBillingServiceDisconnected() {
            Log.e("TAG", "Service Disconnected.")
        }
    })
}
```

4. 구매가능한 아이템 리스트 가져오기

```
/**
 * 구매 가능한 리스트의 아이템을 추가한다
 * Google PlayConsole 의 상품Id 와 동일하게 적어준다.
 *
 * always_item : 중복해서 무제한으로 살 수 있는 아이템
 * once_item : 1번만 살 수 있는 아이템
 */
private val purchasableList = listOf("always_item", “once_item")

private fun querySkuDetails() {
    val params = SkuDetailsParams.newBuilder()
    params.setSkusList(purchasableList).setType(BillingClient.SkuType.INAPP)
    billingClient.querySkuDetailsAsync(params.build()) { result, skuDetails ->
        if (result.responseCode == BillingClient.BillingResponseCode.OK) {
            if (skuDetails.isNullOrEmpty() || skuDetails.size != purchasableList.size) {
                "앗 실패했다 ;ㅁ; 아이템 아이디가 확실한가요??".showToast()
            } else {
                this.skuDetails = skuDetails as ArrayList<SkuDetails>
            }
        } else {
            "앗 실패했다 ;ㅁ; ${result.responseCode}".showToast()
        }
    }
}
```

4. 구매가능한 아이템 리스트 가져오기

SkuType.INAPP
SkuType.SUBS

```
private fun querySkuDetails() {
    val params = SkuDetailsParams.newBuilder()
    params.setSkusList(purchasableList).setType(BillingClient.SkuType.INAPP)
    billingClient.querySkuDetailsAsync(params.build()) { result, skuDetails ->
        if (result.responseCode == BillingClient.BillingResponseCode.OK) {
            if (skuDetails.isNullOrEmpty() || skuDetails.size != purchasableList.size) {
                "앗 실패했다 ;ㅁ; 아이템 아이디가 확실한가요??".showToast()
            } else {
                this.skuDetails = skuDetails as ArrayList<SkuDetails>
            }
        } else {
            "앗 실패했다 ;ㅁ; ${result.responseCode}".showToast()
        }
    }
}
```

4. 구매가능한 아이템 리스트 가져오기

```
private fun querySkuDetails() {
    val params = SkuDetailsParams.newBuilder()
    params.setSkusList(purchasableList).setType(BillingClient.SkuType.INAPP)
    billingClient.querySkuDetailsAsync(params.build()) { result, skuDetails ->
        if (result.res                                    ingResponseCode.OK) {
            if (skuDetails.isNullOrEmpty() || skuDetails.size != purchasableList.size) {
                "앗 실패했다 ;ㅁ; 아이템 아이디가 확실한가요??".showToast()
            } else {
                this.skuDetails = skuDetails as ArrayList<SkuDetails>
            }
        } else {
            "앗 실패했다 ;ㅁ; ${result.responseCode}".showToast()
        }
    }
}
```

Google Play에 인앱 상품 세부정보를 쿼리

4. 구매가능한 아이템 리스트 가져오기

```
private fun querySkuDetails() {
    val params = SkuDetailsParams.newBuilder()
    params.setSkusList(purchasableList).setType(BillingClient.SkuType.INAPP)
    billingClient.querySkuDetailsAsync(params.build()) { result, skuDetails ->
        if (result.responseCode == BillingClient.BillingResponseCode.OK) {
            if (skuDetails.isNullOrEmpty() || skuDetails.size != purchasableList.size) {
                "앗 실패했다 ;ㅁ; 아이템 아이디가 확실한가요??".showToast()
            } else {
                this.skuDetails = skuDetails as ArrayList<SkuDetails>
            }
        } else {
            "앗 실패했다 ;ㅁ; ${result.responseCode}".showToast()
        }
    }
}
```

4. 구매가능한 아이템 리스트 가져오기

```
private fun querySkuDetails() {
    val params = SkuDetailsParams.newBuilder()
    params.setSkusList(purchasableList).setType(BillingClient.SkuType.INAPP)
    billingClient.querySkuDetailsAsync(params.build()) { result, skuDetails ->
        if (result.responseCode == BillingClient.BillingResponseCode.OK) {
            if (skuDetails.isNullOrEmpty() || skuDetails.size != purchasableList.size) {
                "앗 실패했다 ;ㅁ; 아이템 아이디가 확실한가요??".showToast()
            } else {
                this.skuDetails = skuDetails as ArrayList<SkuDetails>
            }
        } else {
            "앗 실패했다 ;ㅁ; ${result.responseCode}".showToast()
        }
    }
}
```

```
SkuDetails: {
    "productId": "always_item",
    "type": "inapp",
    "price": "₩1,100",
    "price_amount_micros": 1100000000,
    "price_currency_code": "KRW",
    "title": "계속 살 수 있어요 (inappbillingTest)",
    "description": "소비성이므로 여러번 구매할 수 있어요 ",
    “skuDetailsToken":
"AEuhp4LL032xiqGfn-40CK7CUtE1eGIkuoBovRl1FEHDjCI_DCrN5JvoVuUel8rdg94o"
}
```

5. 상품 구매하기

```
private fun setUpViews() {
    binding.buyAlwaysButton.setOnClickListener { purchaseItem(0) }
    binding.buyOnceButton.setOnClickListener { purchaseItem(1) }
}

/**
 * 아이템을 구매하기 위해서는 구매가능한 아이템 리스트와 확인이 필요하다.
 * 리스트가 존재할 경우 실제 구매를 할 수 있다.
 */
private fun purchaseItem(itemIndex: Int) {
    val flowParams =
        BillingFlowParams.newBuilder().setSkuDetails(skuDetails[itemIndex]).build()
    val responseCode =
        billingClient.launchBillingFlow(this@MainActivity, flowParams).responseCode
    if (responseCode == BillingClient.BillingResponseCode.OK) {
        // 인앱결제 바텀시트가 노출됨
    } else {
        // 구매요청이 실패한 경우
        "구매요청이 실패했습니다 ;ㅁ; $responseCode".showToast()
    }
}
```

5. 상품 구매하기

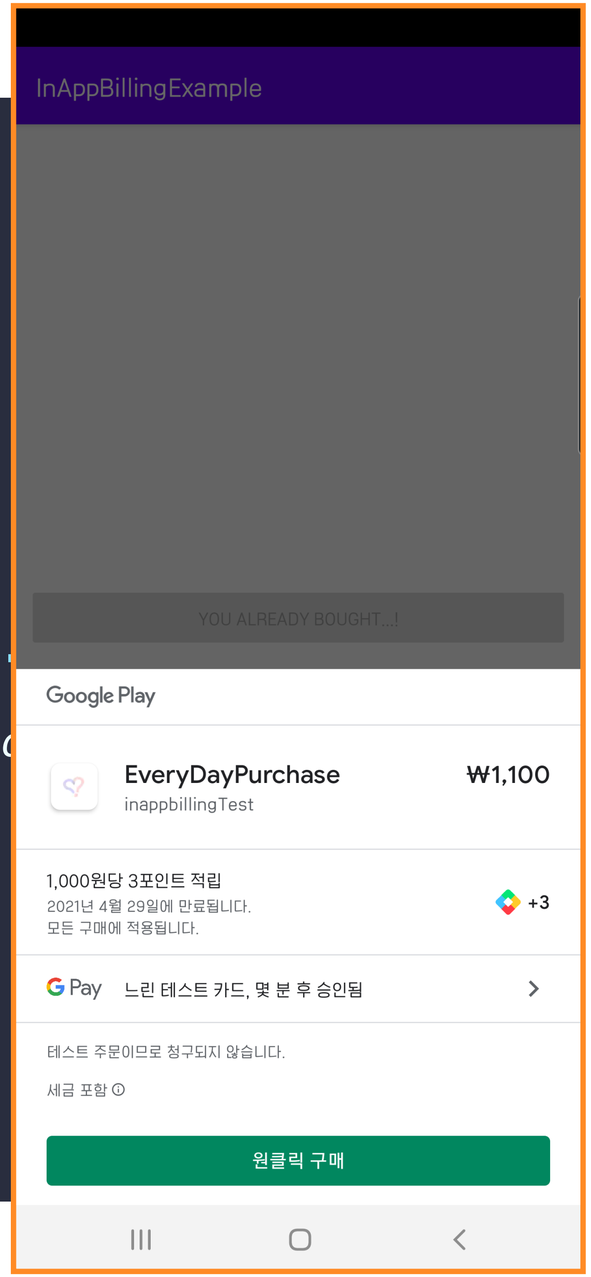
```
private fun setUpViews() {
    binding.buyAlwaysButton.setOnClickListener { purchaseItem(0) }
    binding.buyOnceButton.setOnClickListener { purchaseItem(1) }
}

/**
 * 아이템을 구매하기 위해서는 구매가능한 아이템 리스트와 확인이 필요하다.
 * 리스트가 존재할 경우 실제 구매를 할 수 있다.
 */
private fun purchaseItem(itemIndex: Int) {
    val flowParams =
        BillingFlowParams.newBuilder().setSkuDetails(skuDetails[itemIndex]).build()
    val responseCode =
        billingClient.launchBillingFlow(this@MainActivity, flowParams).responseCode
    if (responseCode == BillingClient.BillingResponseCode.OK) {
        // 인앱결제 바텀시트가 노출됨
    } else {
        // 구매요청이 실패한 경우
        "구매요청이 실패했습니다 ;ㅁ; $responseCode".showToast()
    }
}
```

5. 상품 구매하기

```kotlin
private fun setUpViews() {
    binding.buyAlwaysButton.setOnClickListener { purchaseItem(0) }
    binding.buyOnceButton.setOnClickListener { purchaseItem(1) }
}

/**
 * 아이템을 구매하기 위해서는 구매가능한 아이템 리스트와 확인이 필요하다.
 * 리스트가 존재할 경우 실제 구매를 할 수 있다.
 */
private fun purchaseItem(itemIndex: Int) {
    val flowParams =
        BillingFlowParams.newBuilder().setSkuDetails(skuDetails[itemIndex]).
    val responseCode =
        billingClient.launchBillingFlow(this@MainActivity, flowParams).respo
    if (responseCode == BillingClient.BillingResponseCode.OK) {
        // 인앱결제 바텀시트가 노출됨
    } else {
        // 구매요청이 실패한 경우
        "구매요청이 실패했습니다 ;ㅁ; $responseCode".showToast()
    }
}
```



5. 상품 구매하기

```kotlin
/**
 * 구매 방식에 대해 처리한다. item 의 id 에 따라서
 * 각 처리 방법을 purchaseAlwaysItem 또는 purchaseOnceItem 로 보낸다.
 * (바텀에 결제 화면이 뜬 시점)
 */
override fun onPurchasesUpdated(
    billingResult: BillingResult,
    purchases: List<Purchase>?
) {
    if (billingResult.responseCode == BillingClient.BillingResponseCode.OK && purchases != null) {
        for (purchase in purchases) {
            val isAlwaysItem = purchase.skus.firstOrNull { it == "always_item" }
            if (isAlwaysItem != null) {
                purchaseAlwaysItem(purchase)
            } else {
                purchaseOnceItem(purchase)
            }
        }
    } else if (billingResult.responseCode == BillingClient.BillingResponseCode.USER_CANCELED) {
        "상품 주문이 취소되었습니다".showToast()
    } else {
        "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
    }
}
```

5. 상품 구매하기

```
/**
 * 구매 방식에 대해 처리한다. item 의 id 에 따라서
 * 각 처리 방법을 purchaseAlwaysItem 또는 purchaseOnceItem 로 보낸다.
 * (바텀에 결제 화면이 뜬 시점)
 */
override fun onPurchasesUpdated(
    billingResult: BillingResult,
    purchaseList: MutableList<Purchase>?
) {
    when (billingResult.responseCode) {
        BillingClient.BillingResponseCode.OK -> {
            purchaseList?.forEach {
                if (it.sku == "always_item") {
                    purchaseAlwaysItem(it.purchaseToken)
                } else {
                    purchaseOnceItem(it.purchaseToken)
                }
            }
        }
        …
    }
}
```

6. 소비성 아이템 구매 처리하기

```kotlin
/**
 * 구매 방식에 대해 처리한다. item 의 id 에 따라서
 * 각 처리 방법을 purchaseAlwaysItem 또는 purchaseOnceItem 로 보낸다.
 * (바텀에 결제 화면이 뜬 시점)
 */
override fun onPurchasesUpdated(
    billingResult: BillingResult,
    purchases: List<Purchase>?
) {
    if (billingResult.responseCode == BillingClient.BillingResponseCode.OK && purchases != null) {
        for (purchase in purchases) {
            val isAlwaysItem = purchase.skus.firstOrNull { it == "always_item" }
            if (isAlwaysItem != null) {
                purchaseAlwaysItem(purchase)
            } else {
                purchaseOnceItem(purchase)
            }
        }
    } else if (billingResult.responseCode == BillingClient.BillingResponseCode.USER_CANCELED) {
        "상품 주문이 취소되었습니다".showToast()
    } else {
        "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
    }
}
```

6. 소비성 아이템 구매 처리하기

```
// 소비성 (계속 구매 가능한) 제품 구매시
private fun purchaseAlwaysItem(purchase: Purchase) {

    // 만약에 기존에 같은 토큰값으로 구매한 경우가 있는지 확인한다.
    // 토큰값은 고유한 값이기 때문에 구매한 이력이 있다면 구매자격을 부여하지 않도록 한다.
    // *함정 : 토큰값을 저장할 수 있는 곳은 백엔드이다.  (서버개발자님을 구해보자 아님 파이어베이스라도...?)
    val consumeParams =
        ConsumeParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()

    billingClient.consumeAsync(consumeParams) { billingResult, _ ->
        if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
            "감사합니다! 사,사,사는동안 많이버세요!".showToast()
        } else {
            "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
        }
    }
}
```

6. 소비성 아이템 구매 처리하기

```
// 소비성 (계속 구매 가능한) 제품 구매시
private fun purchaseAlwaysItem(purchase: Purchase) {

    // 만약에 기존에 같은 토큰
    // 토큰값은 고유한 값이기
    // *함정 : 토큰값을 저장
    val consumeParams =
        ConsumeParams.ne

    billingClient.consum
        if (billingResul
            "감사합니다! 사
        } else {
            "구매요청이 실패
        }
    }
}
```

자격을 부여하기 전에 구매 확인

백엔드에서 처리해야 하는 민감한 데이터와 로직의 특별한 사례는 구매 확인입니다. 사용자가 구매한 후에는 다음을 실행해야 합니다.

1. 상응하는 `purchaseToken`을 백엔드로 전송합니다. 즉, 모든 구매의 모든 `purchaseToken` 값을 기록으로 유지해야 합니다.
2. 현재 구매의 `purchaseToken` 값이 이전의 `purchaseToken` 값과 일치하지 않는지 확인합니다. `purchaseToken`은 전역에서 고유하므로 이 값을 데이터베이스에서 기본 키로 안전하게 사용할 수 있습니다.
3. Google Play Developer API의 `Purchases.products:get` 또는 `Purchases.subscriptions:get` 엔드포인트를 사용하여 구매가 합법적인지 Google로 확인합니다.
4. 구매가 합법적이며 과거에 사용되지 않았다면 인앱 상품 또는 정기 결제에 안전하게 자격을 부여할 수 있습니다.
5. 또한, 정기 결제의 경우 `Purchases.subscriptions:get`에 `linkedPurchaseToken`이 설정될 때 데이터베이스에서 `linkedPurchaseToken`을 삭제하고 `linkedPurchaseToken`에 부여된 자격을 취소하여 여러 사용자에게 동일한 구매 자격이 부여되지 않도록 해야 합니다.

★ **참고:** 모든 구매에서 **`orderId`**를 생성하는 것이 보장되는 것은 아니므로 중복 구매 확인용 또는 데이터베이스 기본 키로 **`orderId`**를 사용하지 않습니다. 특히, 프로모션 코드로 구매하면 **`orderId`**가 생성되지 않습니다.

6. 소비성 아이템 구매 처리하기

```
// 소비성 (계속 구매 가능한) 제품 구매시
private fun purchaseAlwaysItem(purchase: Purchase) {

    // 만약에 기존에 같은 토큰값으로 구매한 경우가 있는지 확인한다.
    // 토큰값은 고유한 값이기 때문에 구매한 이력이 있다면 구매자격을 부여하지 않도록 한다.
    // *함정 : 토큰값을 저장할 수 있는 곳은 백엔드이다.  (서버개발자님을 구해보자 아님 파이어베이스라도...?)
    val consumeParams =
        ConsumeParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()

    billingClient.consumeAsync(consumeParams) { billingResult, _ ->
        if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
            "감사합니다! 사,사,사는동안 많이버세요!".showToast()
        } else {
            "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
        }
    }
}
```

6. 소비성 아이템 구매 처리하기

```
// 소비성 (계속 구매 가능한) 제품 구매시
private fun purchaseAlwaysItem(purchase: Purchase) {

    // 만약에 기존에 같은 토큰값으로 구매한 경우가 있는지 확인한다.
    // 토큰값은 고유한 값이기 때문에 구매한 이력이 있다면 구매자격을 부여하지 않도록 한다.
    // *함정 : 토큰값을 저장할 수 있는 곳은 백엔드이다.  (서버개발자님을 구해보자 아님 파이어베이스라도...?)
    val consumeParams =
        ConsumeParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()

    billingClient.consumeAsync(consumeParams) { billingResult, _ ->
        if (billingRes                                    nt.BillingResponseCode.OK) {
            "감사합니다!                                  ()
        } else {
            "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
        }
    }
}
```

- 사용자에게 구매자격이 부여됐음을 의미
- 소비성 아이템을 또 구매 가능

7. 비소비성 아이템 구매 처리하기

```
// 일회성 제품 구매시
private fun purchaseOnceItem(purchase: Purchase) {
    if (purchase.purchaseState == Purchase.PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {
            val params =
                AcknowledgePurchaseParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()
            billingClient.acknowledgePurchase(params) { billingResult ->
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    "감사합니다! 사,사,사는동안 많이버세요!".showToast()
                    queryPurchaseHistoryAsync()
                } else {
                    "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
                }
            }
        }
    }
}
```

```
public @interface PurchaseState {
    int UNSPECIFIED_STATE = 0;
    int PURCHASED = 1;
    int PENDING = 2;
}
```

- PURCHASED 의 경우에만 구매자격을 부여

7. 비소비성 아이템 구매 처리하기

```
// 일회성 제품 구매시
private fun purchaseOnceItem(purchase: Purchase) {
    if (purchase.purchaseState == Purchase.PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {
            val params =
                AcknowledgePurchaseParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()
            billingClient.acknowledgePurchase(params) { billingResult ->
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    "감사합니다! 사,사,사는동안 많이버세요!".showToast()
                    queryPurchaseHistoryAsync()
                } else {
                    "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
                }
            }
        }
    }
}
```

7. 비소비성 아이템 구매 처리하기

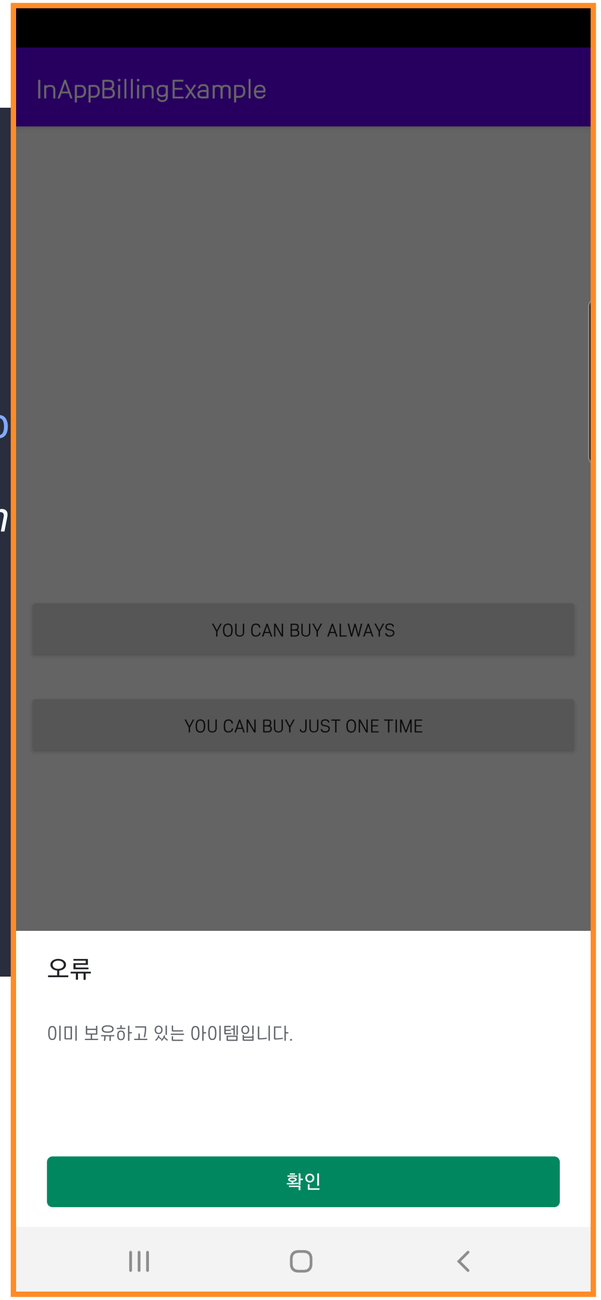
```
// 일회성 제품 구매시
private fun purchaseOnceItem(purchase: Purchase) {
    if (purchase.purchaseState == Purchase.PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {
            val params =
                AcknowledgePurchaseParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()
            billingClient.acknowledgePurchase(params) { billingResult ->
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    "감사합니다! 사,사,사는동안 많이버세요!".showToast()
                    queryPurchaseHistoryAsync()
                } else {
                    "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
                }
            }
        }
    }
}
```

7. 비소비성 아이템 구매 처리하기

```
// 일회성 제품 구매시
private fun purchaseOnceItem(purchase: Purchase) {
    if (purchase.purchaseState == Purchase.PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {
            val params =
                AcknowledgePurchaseParams.newBuilder().setPurchaseToken(purchase.purchaseToken).build()
            billingClient.acknowledgePurchase(params) { billingResult ->
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    "감사합니다! 사,사,사는동안 많이버세요!".showToast()
                    queryPurchaseHistoryAsync()
                } else {
                    "구매요청이 실패했습니다 ;ㅁ; ${billingResult.responseCode}".showToast()
                }
            }
        }
    }
}
```

5. 상품 구매하기 (에서 이미 실패합니다)

```
/**
 * 아이템을 구매하기 위해서는 구매가능한 아이템 리스트와 확인이 필요하다.
 * 리스트가 존재할 경우 실제 구매를 할 수 있다.
 */
private fun purchaseItem(itemIndex: Int) {
    val flowParams =
        BillingFlowParams.newBuilder().setSkuDetails(skuDetails[itemIndex]).b
    val responseCode =
        billingClient.launchBillingFlow(this@MainActivity, flowParams).respon
    if (responseCode == BillingClient.BillingResponseCode.OK) {
        // 인앱결제 바텀시트가 노출됨
    } else {
        // 구매요청이 실패한 경우
        "구매요청이 실패했습니다 ;ㅁ; $responseCode".showToast()
    }
}
```



8. 비소비성 아이템을 이미 구매한 고갱님의 재구매을 막으려면?

```
/**
 * 이미 결제한 적이 있는 아이템을 보여주지 않아야 하는 경우 또는
 * 선택을 비활성화 시켜야 하는 경우 등에 사용할 function
 * pusrchases list 을 통해 구매된 제품만 확인할 수 있다.
 */
fun getAcknowledgePurchasedItem() {
    billingClient.queryPurchasesAsync(
        BillingClient.SkuType.INAPP, object : PurchasesResponseListener {
            override fun onQueryPurchasesResponse(result: BillingResult, pusrchases: MutableList<Purchase>) {
                if (pusrchases.isNullOrEmpty()) {
                    Log.d("TAG", "No existing in app purchases found.")
                } else {
                    pusrchases.forEach { acknowledgePurchase ->
                        purchasableList.forEach { itemId ->
                            val isPurchasedId =
                                acknowledgePurchase.skus.firstOrNull { it == itemId }
                            if (isPurchasedId != null) {
                                binding.buyOnceButton.text = "YOU ALREADY BOUGHT...!"
                                binding.buyOnceButton.isEnabled = false
                            }
                        }
                    }
                }
            }
        })
}
```

8. 비소비성 아이템을 이미 구매한 고갱님의 재구매을 막으려면?

```
/**
 * 이미 결제한 적이 있는 아이템을 보여주지 않아야 하는 경우 또는
 * 선택을 비활성화 시켜야 하는 경우 등에 사용할 function
 * pusrchases list 을 통해 구매된 제품만 확인할 수 있다.
 */
fun getAcknowledgePurchasedItem() {
    billingClient.queryPurchasesAsync(
        BillingClient.SkuType.INAPP, object : PurchasesResponseListener {
            override fun onQueryPurchasesResponse(result: BillingResult, pusrchases: MutableList<Purchase>) {
                if (pusrchases.isNullOrEmpty()) {
                    Log.d("TAG", "No existing in app purchases found.")
                } else {
                    pusrchases.forEach { acknowledgePurchase ->
                        purchasableList.forEach { itemId ->
                            val isPurchasedId =
                                acknowledgePurchase.skus.firstOrNull { it == itemId }
                            if (isPurchasedId != null) {
                                binding.buyOnceButton.text = "YOU ALREADY BOUGHT...!"
                                binding.buyOnceButton.isEnabled = false
                            }
                        }
                    }
                }
            }
        })
}
```

8. 비소비성 아이템을 이미 구매한 고갱님의 재구매을 막으려면?

```
/**
 * 이미 결제한 적이 있는 아이템을 보여주지 않아야 하는 경우 또는
 * 선택을 비활성화 시켜야 하는 경우 등에 사용할 function
 * pusrchases list 을 통해 구매된 제품만 확인할 수 있다.
 */
fun getAcknowledgePurchasedItem() {
    billingClient.queryPurchasesAsync(
        BillingClient.SkuType.INAPP, object : PurchasesResponseListener {
            override fun onQueryPurchasesResponse(result: BillingResult, pusrchases: MutableList<Purchase>) {
                if (pusrchases.isNullOrEmpty()) {
                    Log.d("TAG", "No existing in app purchases found.")
                } else {
                    pusrchases.forEach { acknowledgePurchase ->
                        purchasableList.forEach { itemId ->
                            val isPurchasedId =
                                acknowledgePurch
                            if (isPurchasedId !=
                                binding.buyOnceB
                                binding.buyOnceB
                            }
                        }
                    }
                }
            }
        })
}
```

```
{
    "orderId": "GPA.3372-1748-5017-23407",
    "packageName": "com.zinc0214.inappbillingexample",
    "productId": "once_item",
    "purchaseTime": 1604323598343,
    "purchaseState": 0,
    "purchaseToken": "demnhbcgalahgbfppdoa...h6PQs",
    "acknowledged": true
}
```

8. 비소비성 아이템을 이미 구매한 고갱님의 재구매을 막으려면?

```
fun getAcknowledgePurchasedItem() {
    if (!billingClient.isReady) {
        return
    }
    val result = billingClient.queryPurchases(BillingClient.SkuType.INAPP)
    if (result.purchasesList == null) {
        // "No existing in app purchases found."
    } else {
        result.purchasesList?.forEach { acknowledgePurchase ->
            getPurchasableList().forEach { purchasableItem ->
                if (acknowledgePurchase.sku == purchasableItem) {
                    binding.buyOnceButton.text = "YOU ALREADY BOUGHT...!"
                    binding.buyOnceButton.isEnabled = false
                }
            }
        }
    }
}
```

9. 뒤늦게 결제에 성공했다면 알 수 있을까?

```kotlin
/**
 * 최근에 결제한 아이템을 확인하는 목적
 * checkPurchaseHistory function 과 다르게  consumeAsync 를 통해 구매된 상품도 확인할 수 있다.
 * !!주의!! 아이템 Id 로만 가져오기 때문에 여러번 구매하더라도 가장 최근의 제품만 가져온다.
 */
fun queryPurchaseHistoryAsync() {
    billingClient.queryPurchaseHistoryAsync(
        BillingClient.SkuType.INAPP,
        object : PurchaseHistoryResponseListener {
            // 최근 구매한 아이템을 알고자 할 때 사용
            override fun onPurchaseHistoryResponse(
                billingResult: BillingResult,
                purchaseHistoryList: MutableList<PurchaseHistoryRecord>?
            ) {
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    if (!purchaseHistoryList.isNullOrEmpty()) {
                        purchaseHistoryList.forEach {
                            Log.d("hana", "Previous Purchase Item : ${it.originalJson}")
                        }
                    }
                }
            }

        })
}
```

9. 뒤늦게 결제에 성공했다면 알 수 있을까?

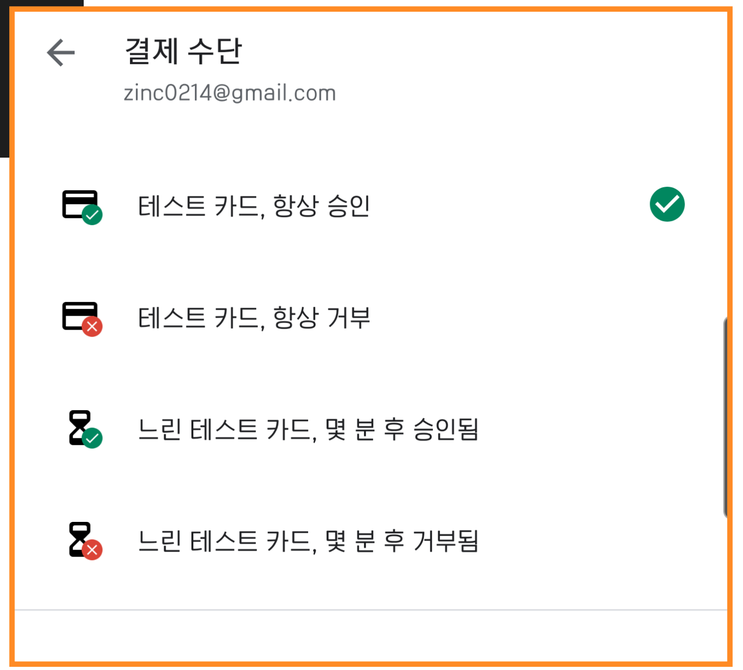
```
/**
 * 최근에 결제한 아이템을 확인하는 목적
 * checkPurchaseHistory function 과 다르게  consumeAsync 를 통해 구매된 상품도 확인할 수 있다.
 * !!주의!! 아이템 Id 로만 가져오기 때문에 여러번 구매하더라도 가장 최근의 제품만 가져온다.
 */
fun queryPurchaseHistoryAsync() {
    billingClient.queryPurchaseHistoryAsync(
        BillingClient.SkuType.INAPP,
        object : PurchaseHistoryResponseListener {
            // 최근 구매한 아이템을 알고자 할 때 사용
            override fun onPurchaseHistoryResponse(
                billingResult: BillingResult,
                purchaseHistoryList: MutableList<PurchaseHistoryRecord>?
            ) {
                if (billingResult.responseCode == BillingClient.BillingResponseCode.OK) {
                    if (!purchaseHistoryList.isNullOrEmpty()) {
                        purchaseHistoryList.forEach {
                            Log.d("hana", "Previous Purchase Item : ${it.originalJson}")
                        }
                    }
                }
            }

        })
}
```

9. 뒤늦게 결제에 성공했다면 알 수 있을까?


```
{
    "productId": "always_item",
    "purchaseToken": "gjfhhalimdgfglnhkjjhghce...Y4iAmAZXakBrbKjb8DURxOE",
    "purchaseTime": 1623243318217,
    "quantity": 1,
    "developerPayload": null
}

{
    "productId": "once_item",
    "purchaseToken": "demnhbcgalahgbfppdoabalk....TduBLh3XDzFTYsh6PQs",
    "purchaseTime": 1604323598343,
    "developerPayload": null
}
```

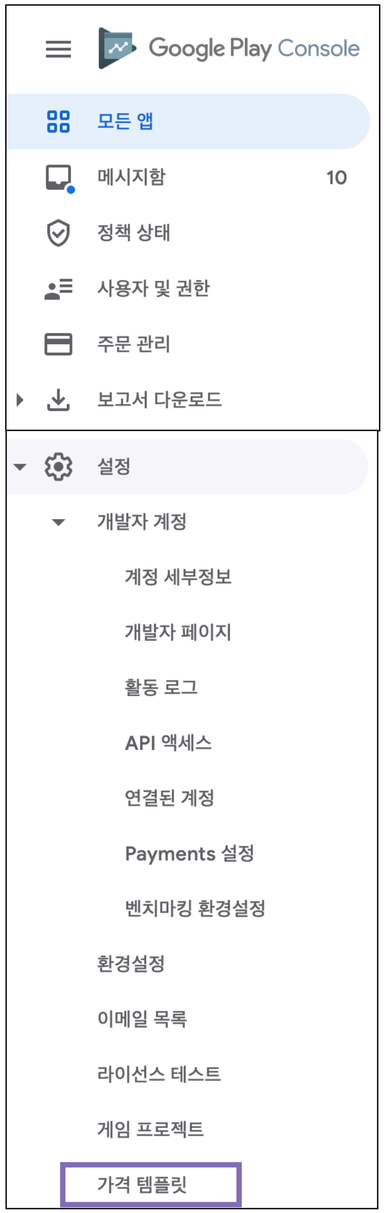



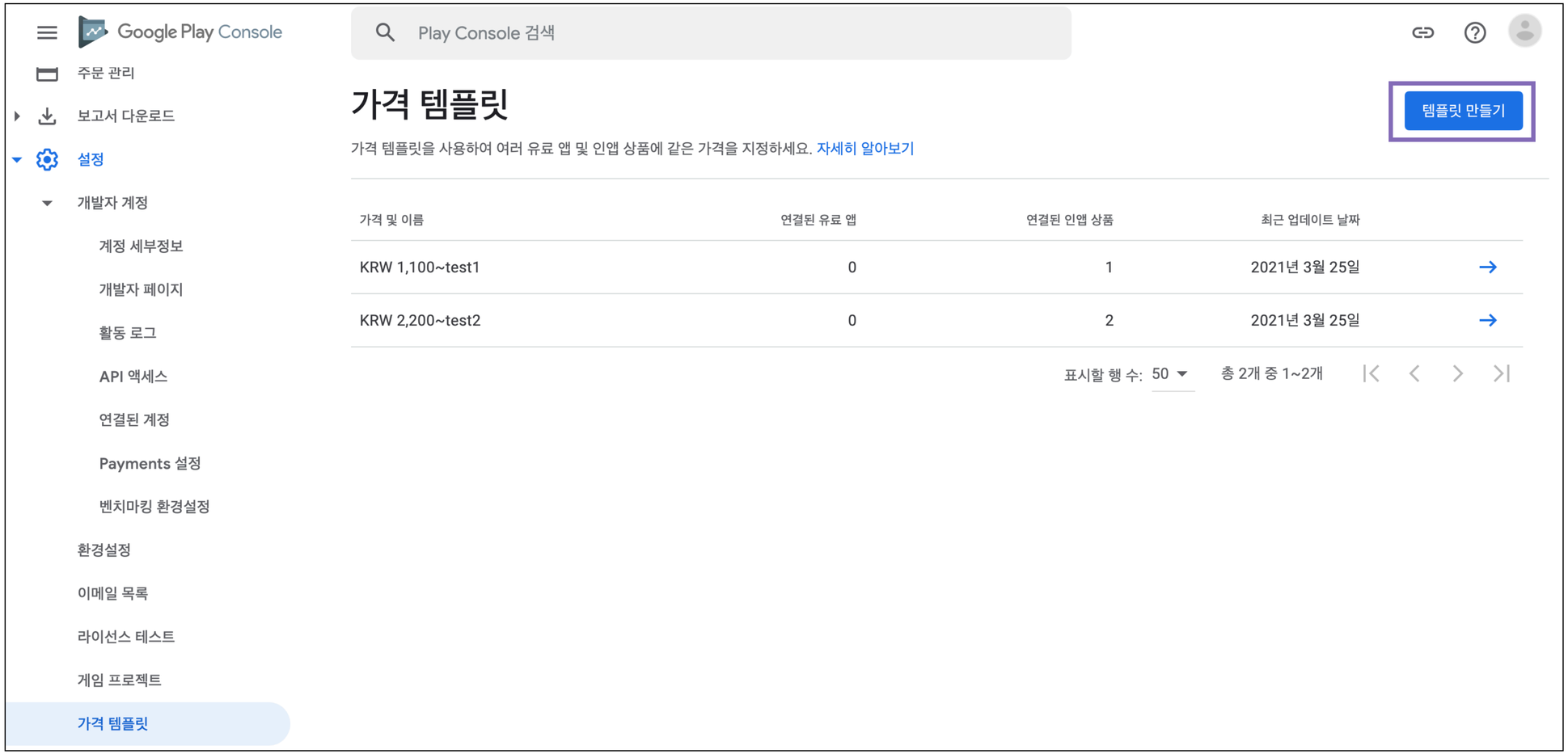
STEP 3

STEP 3

넘어가기 전에 APK 를 하나 말아주세요!

1. 가격 템플릿 추가하기

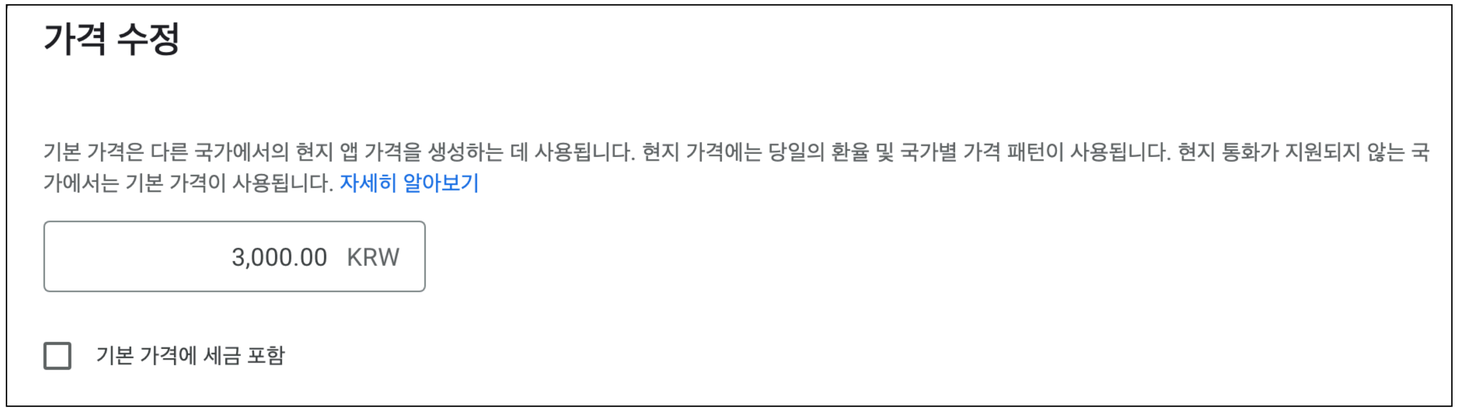




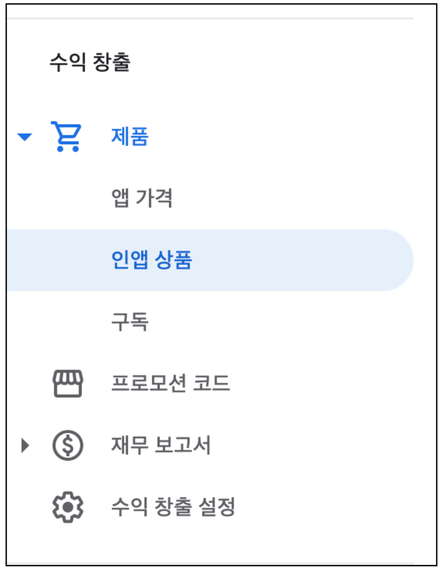
Google Play Console

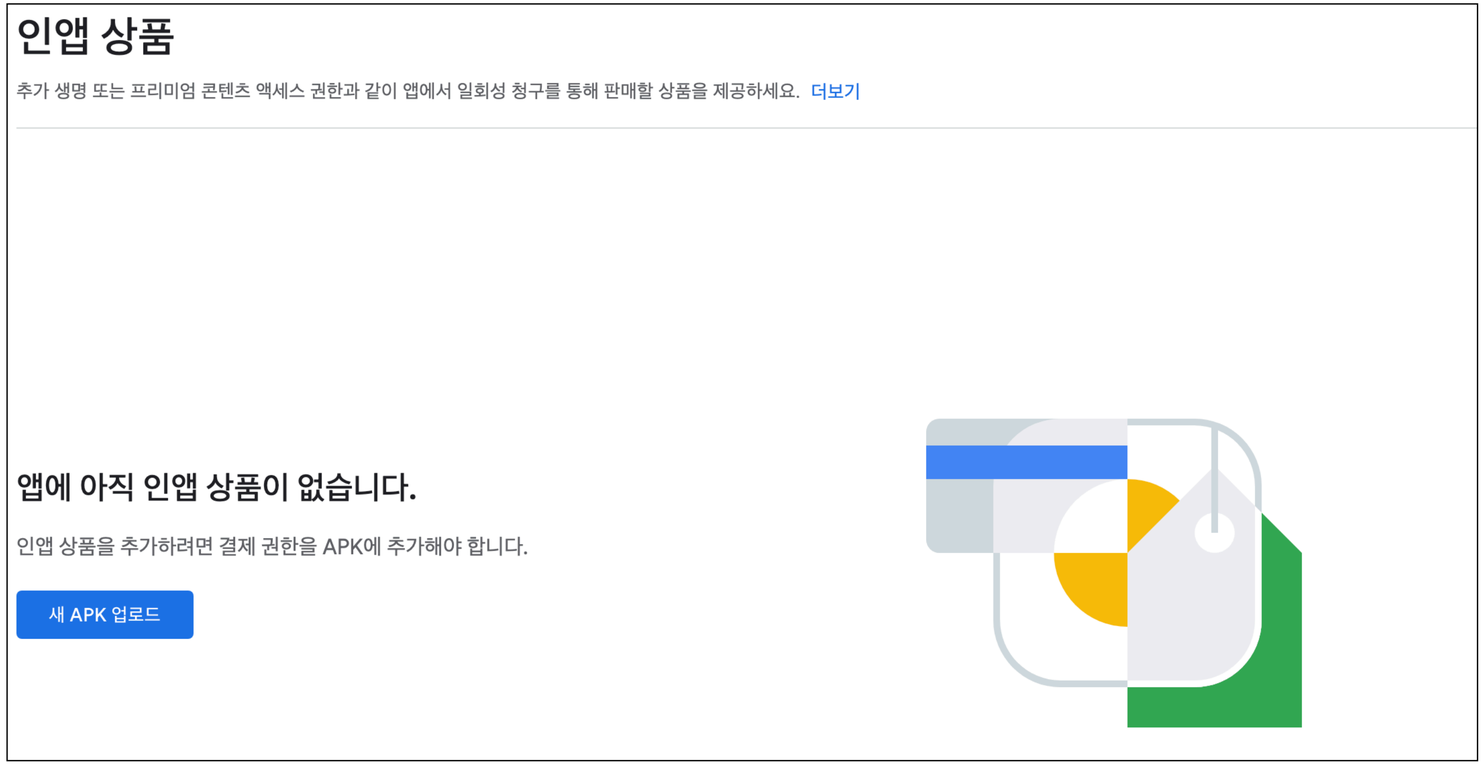
1. 가격 템플릿 추가하기


← 가격 템플릿
가격 템플릿 만들기
가격 템플릿을 사용하여 여러 유료 앱 및 인앱 상품에 동일한 가격을 설정하거나 관리할 수 있습니다. 자세히 알아보기
세부정보
연결된 제품
* – 필수 입력란
세부정보
이름
new_price
기본 가격이 이 이름 앞에 추가됨
9/30
가격 *
가격 설정

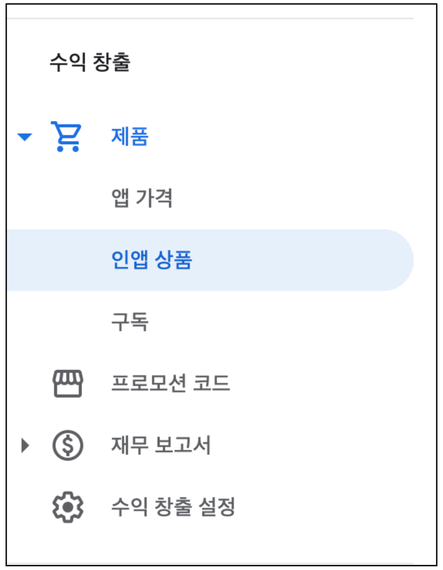

가격 수정
기본 가격은 다른 국가에서의 현지 앱 가격을 생성하는 데 사용됩니다. 현지 가격에는 당일의 환율 및 국가별 가격 패턴이 사용됩니다. 현지 통화가 지원되지 않는 국가에서는 기본 가격이 사용됩니다. 자세히 알아보기
3,000.00 KRW
기본 가격에 세금 포함

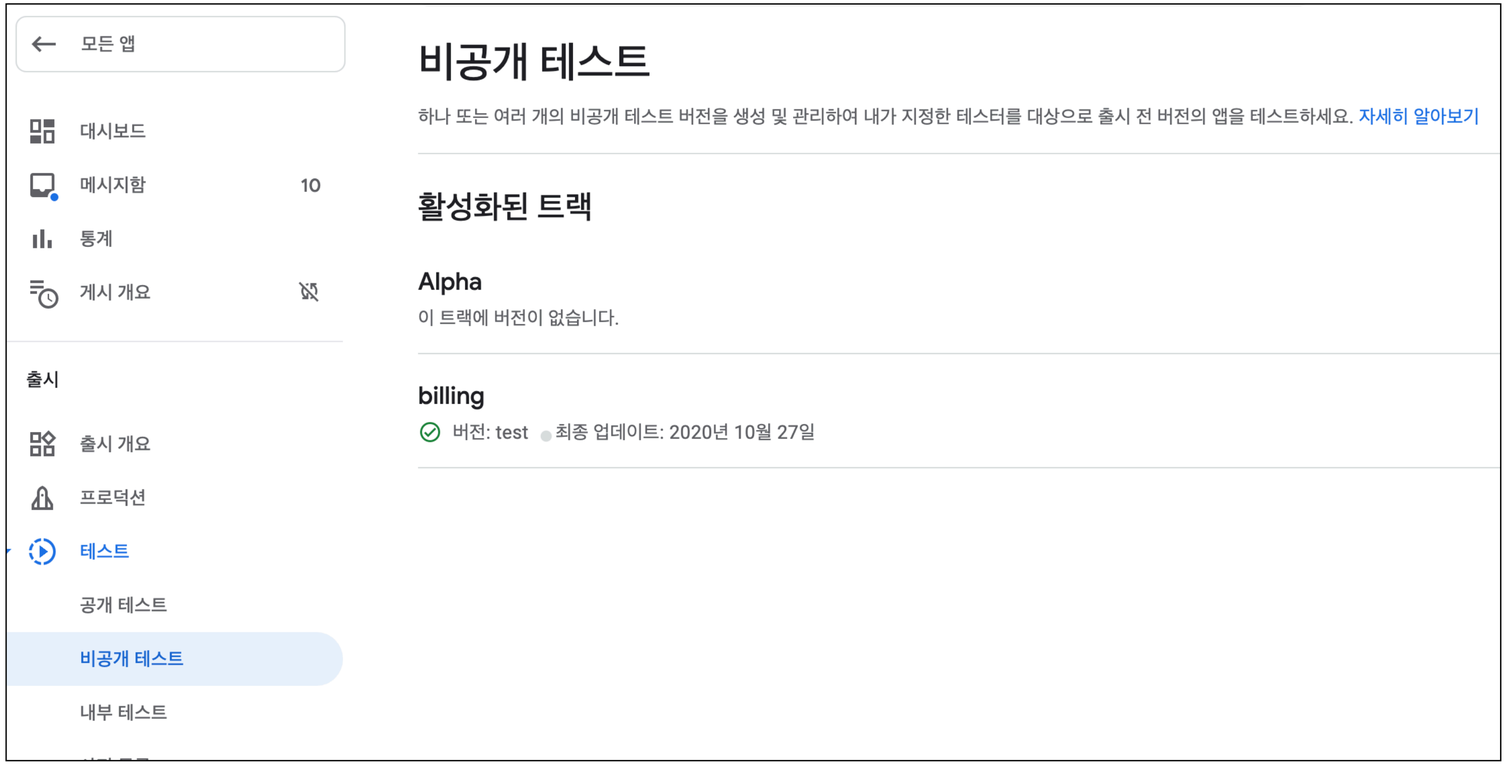
2. 인앱 상품 추가하기


수익 창출
제품
앱 가격
인앱 상품
구독
프로모션 코드
재무 보고서
수익 창출 설정


인앱 상품
추가 생명 또는 프리미엄 콘텐츠 액세스 권한과 같이 앱에서 일회성 청구를 통해 판매할 상품을 제공하세요. 더보기
앱에 아직 인앱 상품이 없습니다.
인앱 상품을 추가하려면 결제 권한을 APK에 추가해야 합니다.
새 APK 업로드

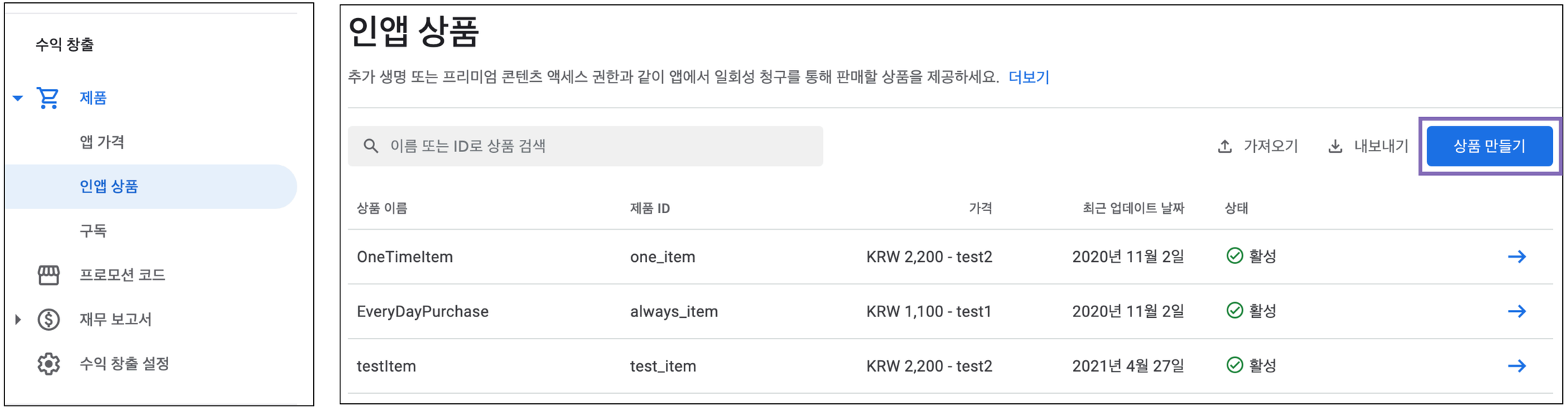
2. 인앱 상품 추가하기





활성화 되기 까지 하루 넘게 걸릴수도…

2. 인앱 상품 추가하기

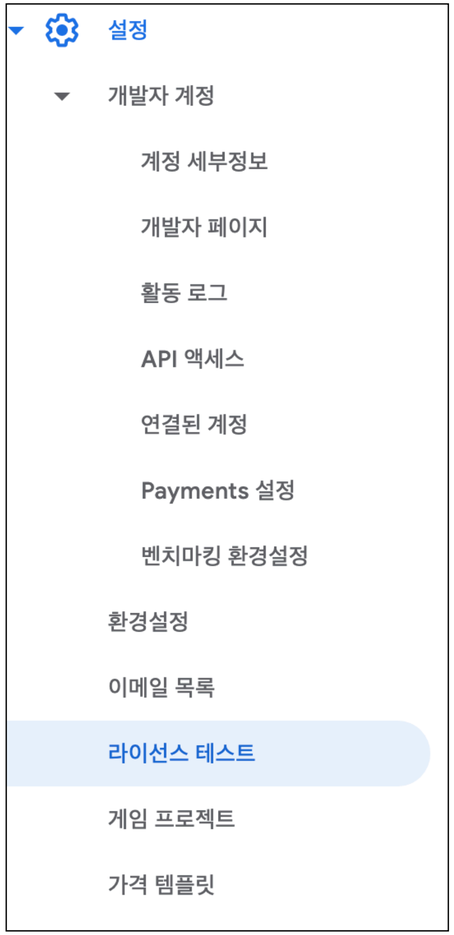

수익 창출
제품
앱 가격
인앱 상품
구독
프로모션 코드
재무 보고서
수익 창출 설정
인앱 상품
추가 생명 또는 프리미엄 콘텐츠 액세스 권한과 같이 앱에서 일회성 청구를 통해 판매할 상품을 제공하세요. 더보기
이름 또는 ID로 상품 검색
가져오기
내보내기
상품 만들기
상품 이름
제품 ID
가격
최근 업데이트 날짜
상태
OneTimeItem
one_item
KRW 2,200 - test2
2020년 11월 2일
활성
EveryDayPurchase
always_item
KRW 1,100 - test1
2020년 11월 2일
활성
testItem
test_item
KRW 2,200 - test2
2021년 4월 27일
활성

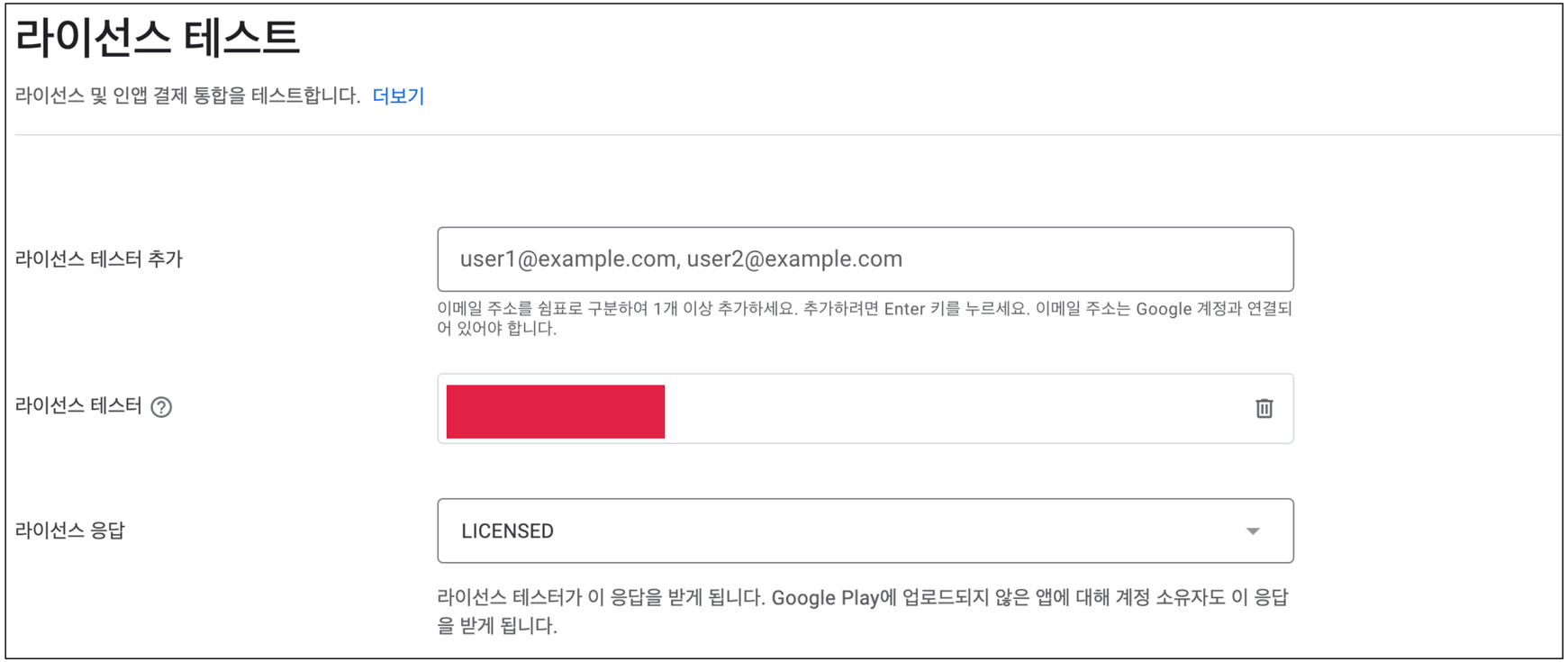
2. 인앱 상품 추가하기


← 인앱 상품
인앱 상품 만들기
* – 필수 입력란입니다. 한국어의 모든 입력란을 작성해 주세요.
제품 ID *
0/136
숫자 또는 소문자로 시작해야 하며 숫자(0~9), 소문자(a~z), 밑줄(_), 마침표(.)를 포함할 수 있습니다.
상품을 만든 다음에는 상품 ID를 변경하거나 재사용할 수 없습니다. 자세히 알아보기
기본값 – 한국어 – ko-KR 번역 관리
상품 세부정보
이름 *
0/55
설명 *
0/200
가격
기본 가격 *
가격 템플릿 선택
가격 설정


← 인앱 상품
test
삭제
비활성 test
* – 필수 입력란입니다. 한국어의 모든 입력란을 작성해 주세요.
기본값 – 한국어 – ko-KR 번역 관리
상품 세부정보
이름 *
test
4/55
설명 *
test
4/200
변경사항 저장
활성화

3. 라이선스 테스터 추가


설정
개발자 계정
계정 세부정보
개발자 페이지
활동 로그
API 액세스
연결된 계정
Payments 설정
벤치마킹 환경설정
환경설정
이메일 목록
라이선스 테스트
게임 프로젝트
가격 템플릿


라이선스 테스트
라이선스 및 인앱 결제 통합을 테스트합니다. 더보기
라이선스 테스터 추가
user1@example.com, user2@example.com
이메일 주소를 쉼표로 구분하여 1개 이상 추가하세요. 추가하려면 Enter 키를 누르세요. 이메일 주소는 Google 계정과 연결되어 있어야 합니다.
라이선스 테스터
라이선스 응답
LICENSED
라이선스 테스터가 이 응답을 받게 됩니다. Google Play에 업로드되지 않은 앱에 대해 계정 소유자도 이 응답을 받게 됩니다.

STEP 4

1. 저는 이런것을 겪었습니다

- 간헐적으로 PurchasesUpdatedListener 가 여러번 호출되는 문제

  -> 현상 : 해당 리스너에서 다음으로 호출되는 메소드에 의해 구매완료 또는 구매취소 토스트가 여러번 뜨게된다

  -> 해결방법 : 같은 값의 token 이 해당 리스너를 탔을 때, 이전의 responseCode 가 성공이었다면 다음 메소드로 넘어가지 않도록 해결

- 돈을 받기 위해서(?)는 신분증 제출이 필요하다

1. 저는 이런것을 겪었습니다

- 간헐적으로 PurchasesUpdatedListener 가 여러번 호출되는 문제

  -> 현상 : 해당 리스너에서 다음으로 호출되는 메소

  -> 해결방법 : 같은 값의 token 이 해당 리스너를

  드로 넘어가지 않도록 해결

- 돈을 받기 위해서(?)는 신분증 제출이 필요하다

Google

이 이메일은 아직 **Google** 계정에서 인증 절차를 완료하지 않으셨기 때문에 발송되었습니다. 대한민국 법률에 따라 특정 거래 기준액에 도달하면 **Google**과 같은 결제 시스템 공급자가 고객의 신원을 인증하는 단계를 거쳐야 합니다.

확인을 완료하려면 다음 서류의 사본을 제출해 주셔야 합니다.

- 주민등록증 또는 운전면허증

서류를 제출할 때 앞면과 뒷면 사본을 제출해 주셔야 합니다. 서류를 제출하는 방법은 다음 도움말을 참조하세요.

고객님께서는 서류를 제출함에 따라 아래와 같이 개인정보를 수집, 이용 및 보관하는데 동의합니다.

개인정보 수집의 목적 : 관련 법률 및 규정에 따른 본인 확인 절차의 준수

개인정보 보유 기간 : **Google Payment** 계정 해지 후 **5**년간

서류의 제출을 거부하실 경우 저희와의 어떠한 금융 거래도 불가능함을 양지하여 주시기 바랍니다.

Google은 고객의 정보를 안전하게 보관하기 위해 최선을 다하고 있으며 개인정보처리방침에 명시된 극히 일부의 경우를 제외하고는 누구와도 고객의 세부정보를 공유하지 않으므로 안심하시기 바랍니다.

더 궁금한 점이 있으면 언제든지 본 이메일에 답장하시거나 고객센터에 문의해 주세요.

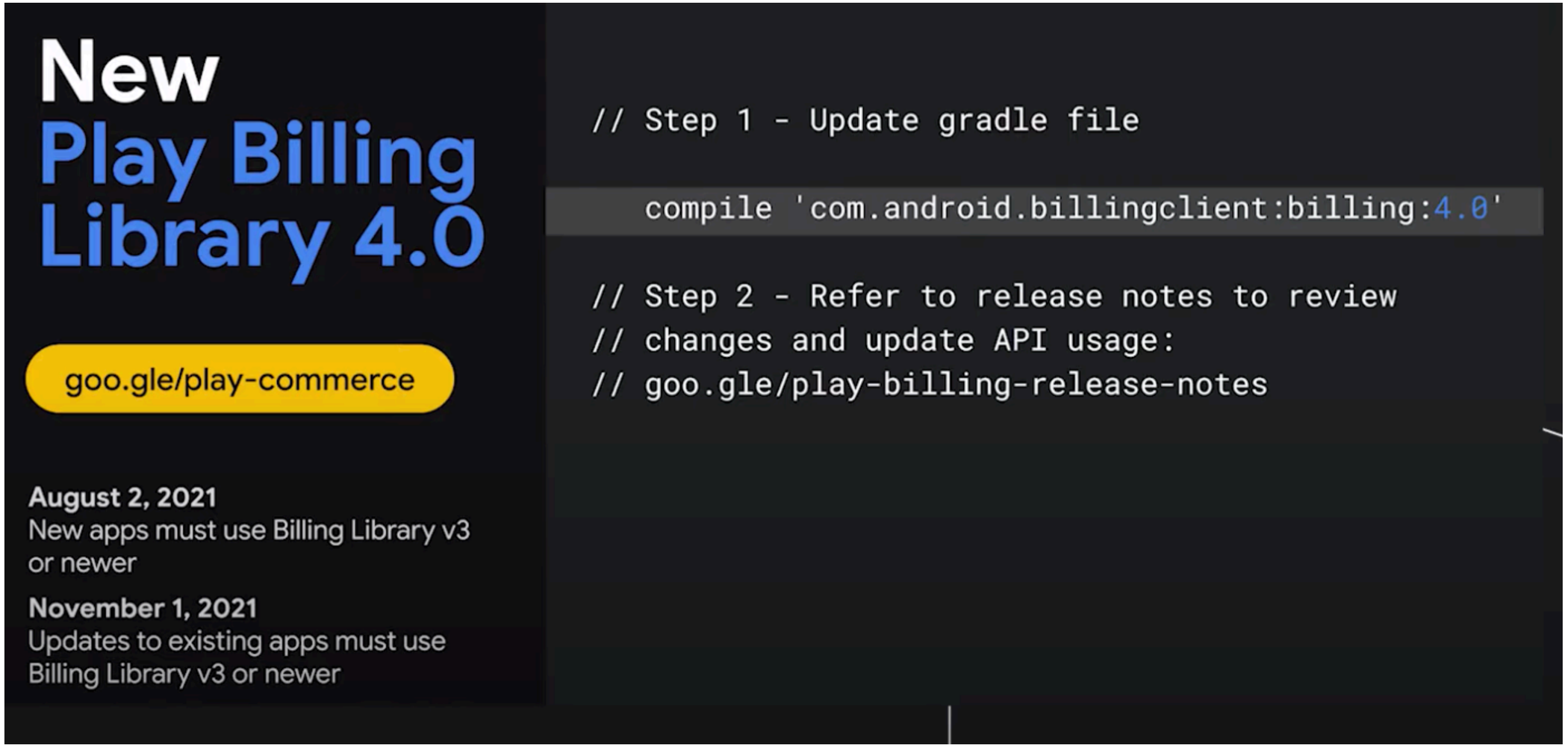
협조해 주셔서 감사합니다.

2. 이런 점에 유의해야 될 수도 있습니다

- 사용자에게 환불을 해줄 것인지
  -> 환불을 하더라도 최근 구매 내역에는 남아있기 때문에 인앱결제 API 상으로 알 수 있는 방법이 없다
  -> 따라서 서버 등에서 따로 관리가 필요하다 ~~(아니면 그냥 해주지 말자!)~~

- 뒤늦게 구매에 성공한 경우, 추후에 어떻게 알 수 있을까
  -> 구매 토큰과 성공 여부를 서버에 저장한다
  -> 어플 진입 시 구매했던 내역 중 실패했던 토큰과 최근 구매한 토큰의 내역을 비교
  -> 비교해서 구매한 토큰이 있다면 성공한 것으로 간주하여 재저장

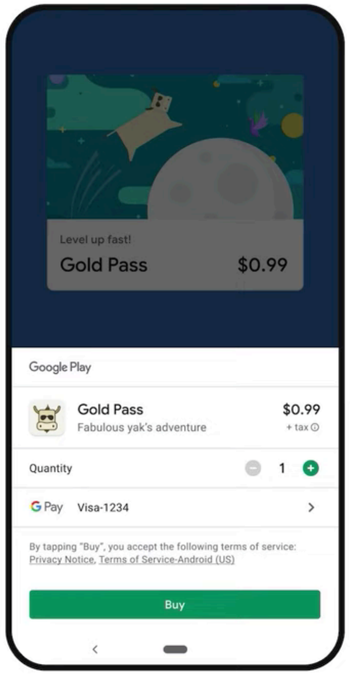
3. 곧 지원이 중단됩니다



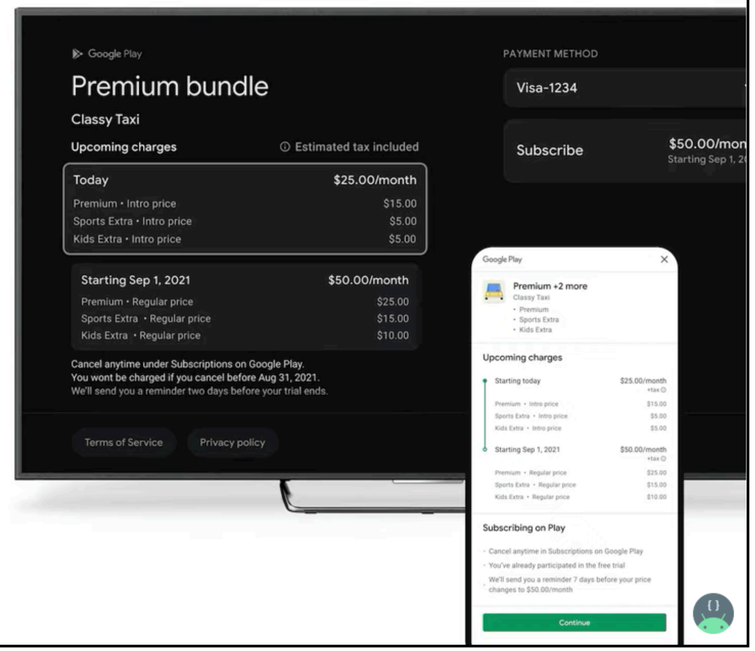
https://android-developers.googleblog.com/2020/06/meet-google-play-billing-library.html

4. 이런 것을 사용할 수 있습니다

New
Multi-quantity purchases



New
Multi-line subscriptions



https://android-developers.googleblog.com/2021/05/whats-new-in-google-play-2021.html

감사합니다 :)