📘 플랫폼 프로그래밍 작업내역서

— Android Native Plugin 수정 · 모바일 적용 · Unity 연동 전체 과정

본 문서는 다음 세 가지 흐름을 중심으로 작성한 작업 기록입니다.

1. **오래된 Android 프로젝트를 최신 환경으로 마이그레이션하며 플랫폼 구조 학습**
2. **의도적으로 오류가 포함된 Android 플러그인을 직접 수정하며 Native Plugin 구조 이해**
3. **수정된 플러그인을 실제 Unity 게임(IPSimulator)에 적용하여 Toast 메시지를 통한 UI 연동까지 구현**

이 일련의 과정은 플랫폼 프로그래밍에 대한 이해를 높이고,
Unity–Android–네트워크 구조를 실습 기반으로 익히는 데 목적이 있었습니다.

1. 🤖 레거시 Android 프로젝트 최신화

— 플랫폼 구조 이해의 시작

예제 프로젝트는 **Android SDK 26 + Gradle 4.x + Support Library 기반**으로 매우 오래된 환경이었으며,
최신 Android Studio(Iguana) 및 Java 17에서는 빌드가 불가능한 상태였습니다.

🔧 수행 작업

- Gradle 버전 업데이트
- Android Gradle Plugin(AGP) 최신화
- gradle-wrapper.properties 업데이트
- compile/targetSdkVersion 최신 규칙에 맞게 상향
- Support Library → AndroidX 전환
- namespace 추가

📌 학습한 점

- Android 빌드 시스템의 구성 요소와 역할

- 버전 불일치가 빌드 실패로 직결되는 구조
- 레거시 프로젝트를 최신 환경으로 옮길 때 발생하는 문제들
- Unity와 Android 빌드 환경이 서로 요구하는 사항의 차이

2. 🧩 Android Native Plugin 복원

— 의도된 오류 분석 및 구조 재정립

실습용 예제 플러그인은 정상 동작하지 않도록 다양한 오류가 포함되어 있었습니다.

🚧 발견된 문제

- 메서드명 오타: `SetAcivity` → `SetActivity`
- public 접근제한자 누락
- 기본 생성자 없음
- Unity 호출 경로와 패키지명 불일치
- Toast 실행이 UI Thread가 아님
- Android 13+ 권한 미비
- classes.jar(유니티 플레이어) 중복 포함
- Editor 환경에서 JNI 호출 시도

🛠 수정한 내용

- 메서드 오타 및 패키지명 수정
- 모든 메서드를 `public` 으로 정리
- `public Plugin()` 기본 생성자 추가
- `runOnUiThread()` 로 Toast 실행 구조 변경
- UnityPlayer를 JAR에 포함되지 않도록 `compileOnly` 적용
- Plugin.class만 포함한 **Plugin-clean.jar** 생성
- Editor에서는 네이티브 호출 금지(#if UNITY_EDITOR 분기)
- Android 13+ 권한 추가

📌 학습한 점

- Unity–Java JNI 호출 규칙: 패키지/클래스/메서드/시그니처 정확성이 중요
- 플러그인 JAR 구성의 중요성

- Android UI Thread 정책
- Editor 환경과 실제 Device 환경의 차이

3. ⚙ Unity ↔ Android 플러그인 연동

— PluginManager를 통한 구조 구축

✓ 플러그인 기능

기능	설명
`SetActivity(Activity)`	Unity의 currentActivity 전달
`ShowToast(String)`	Android Native Toast 출력

✓ Unity 쪽 호출 구조

```
using (AndroidJavaClass unityPlayer = new AndroidJavaClass("com.unity3d.player.UnityPlayer"))
{
    activity = unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
}

plugin = new AndroidJavaObject("com.sbsgame.android.androidplugin.Plugin");
plugin.Call("SetActivity", activity);
```

게임 로직 어디서든 다음처럼 호출 가능:

```
PluginManager.Instance.ShowToast("로비에 접속했습니다.");
```

4. 📱 IPSimulator Android 빌드 적용

— 모바일 클라이언트로 이식하며 네트워크 검증

PC 서버 기반 게임(IPSimulator)을 Android 클라이언트에서도 구동 가능한 형태로 빌드했다.

🔧 Android 빌드 준비

- Scripting Backend → IL2CPP
- Target Architecture → ARM64
- Manifest 권한 정리
- input handling 재정리
- ADB 중복 실행 문제 해결
- Unity 버전 변경으로 인한 IL2CPP 오류 → Library 삭제 후 재빌드

✓ 네트워크 테스트 결과

- PC 서버 → Android 기기 TCP 연결
- 채팅/상점/페이즈 전환 모두 정상 작동
- UDP ping 정상 수신 (수 ms 수준)

5. 🔥 게임 로직 기반 Toast 메시지 연동

— 단순 테스트를 넘어 실제 게임 이벤트에 적용

Unity 게임 내 상황에 따라 Toast 메시지를 출력하도록 구성했다.

🎉 로그인 성공 시

```
로비에 접속했습니다.
```

🎉 게임 첫 구매 발생 시

```
첫 구매자가 나타났습니다! 30초 후 라운드가 종료됩니다.
```

Android 실제 기기에서 정상적으로 표시되는 것을 확인했다.

6. 📜 최종 작업 결과 요약

항목	결과
플러그인 복구 및 수정	✓ 완료
Toast 네이티브 호출	✓ 정상 작동
IPSimulator 모바일 빌드	✓ 성공
PC 서버 ↔ Android 클라이언트 연결	✓ 성공
TCP/UDP 통신	✓ 정상
게임 이벤트 기반 Toast 메시지	✓ 정상 출력

🎯 최종 결론

이번 작업은 Android 플랫폼과 Unity 네이티브 연동 방식을 이해하기 위한 실습의 성격이 강했습니다.
플러그인과 빌드 환경에서 여러 문제가 발생했지만, 오류를 하나씩 해결해가며 구조를 스스로 확인할 수 있었고,
작은 수정이 전체 흐름에 어떤 영향을 미치는지 실제 사례로 경험할 수 있었습니다.

특히,

- **레거시 프로젝트 최신화 과정**,
- **플러그인의 구조를 직접 수정한 경험**,
- **실제 게임에서 Toast 메시지를 연동해 본 실습**,
- **모바일 환경에서 서버·클라이언트 구조를 검증한 과정**

이 네 가지를 통해 플랫폼 프로그래밍의 기본적인 개념을 차근차근 익힐 수 있었습니다.

아직 부족한 부분도 많지만, 이번 경험을 바탕으로
Android 네이티브 기능과 Unity 간 통합을 더 안정적으로 구현할 수 있도록
지속적으로 학습해나갈 계획입니다.