종합 보고서

AOSP 조

조원 : 김재현, 이송무, 임진현

지도교수: 조은선 교수님

Table of Contents

Introduction

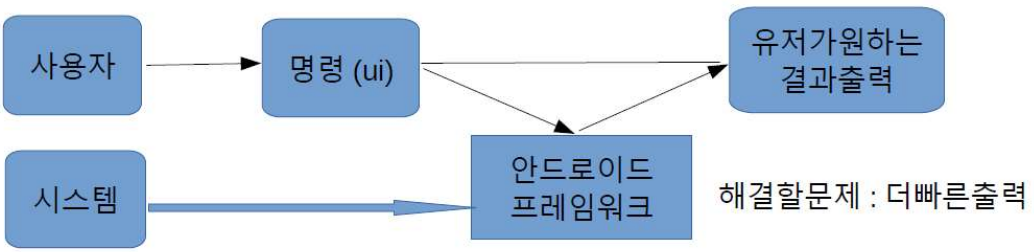
본 보고서는 종합설계 프로젝트에 대한 산출물들을 최종적으로 정리한 문서이다.

보고서에는 테스팅 어플리케이션 프로토타이핑 및 성능 향상에 대하여 어플리케이션을 작성을 하고 이를 확인을 하기 위하여 Log 등을 안드로이드 ADB shell 등을 이용을 하도록 하며 이를 통해서 얻어질 수 있는 설문 및 설문조사 결과를 통하여 프로젝트를 수정하고 보완을 하여 좀 더 나은 프로젝트로 진행이 되도록 하였다.


프로젝트 목표
AOSP 분석
자바 프레임워크단 분석을 하여 개선사항 도출
빌드 및 테스트
안드로이드 8.0//8.1 버전에서 구동 가능한 기기 활용 빌드 후 성능 테스트
AOSP Commit
성능 테스트 확인 후
https://android-review.googlesource.com/q/status:open commit 을 올리는 것이 목표

1. 디자인스프린트

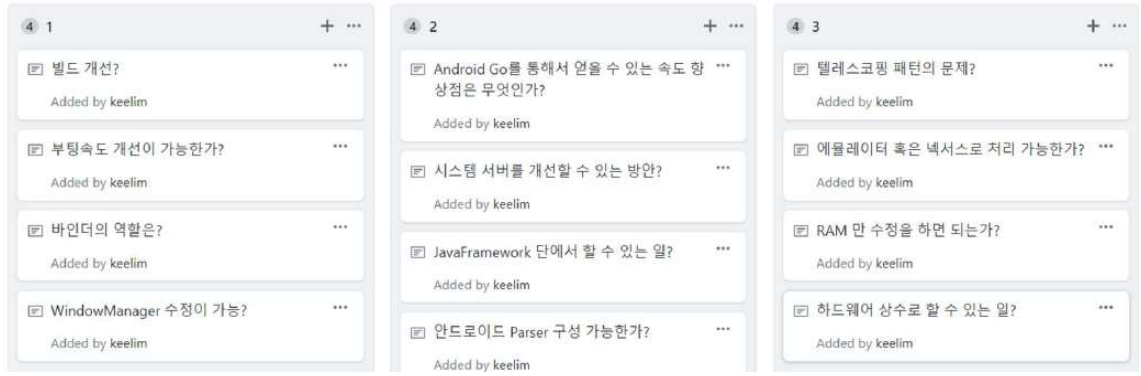
1.1. MAP



1.2. HMW

HMW

어떻게 하면 부팅 시간을 개선할 수 있을까?	어떻게 하면 배터리 사용량을 줄일 수 있을까?	어떻게 하면 필요없는 기능들을 개선해 성능을 향상시킬 수 있을까?
어떻게 하면 RAM을 효율적으로 관리할 수 있을까?	어떻게 하면 전체 프레임워크에서 코드를 개선해 성능을 향상시킬 수 있을까?	어떻게 하면 하드웨어를 조정해서 성능향상을 할 수 있을까?



framework layer 의 역할들 .
이들을 수정해서 개선해야한다 .

어떻게하면 프레임워크의 관리자들을 수정하여 성능향상을 시킬수있을까

어떻게하면 부팅속도를 더 빠르게 만들수있을까

어떻게하면 JNI 코드를 잘짜서 성능을 높힐수있을까

어떻게하면 프레임워크에서 잘 사용되지않는 기능들을 찾고 제거할까

어떻게하면 네이티브 코드를 최대한 사용해서 성능을 높힐까

어떻게하면 윈도우매니저를 수정해 응답성을 높힐수있을까

1.3. 라이트닝데모

1. 어떻게 하면 부팅시간을 개선할 수 있을까?
   기존방안에는 부트로더 최적화, 커널 최적화 방법 등이 있다.
   프레임워크에서 부팅과 관련된 디렉터리나 파일을 분석해 개선여지를 찾는 방법이 있다.

2. 어떻게 하면 배터리 사용량을 줄일 수 있을까?
   기존 방안에는 앱의 상태를 조절하는 방법 및 기기 온도상승을 모니터링해 limit를 주면서 배터리 사용량을 줄이는 방법 등이 있다.
   프레임워크에서 배터리와 관련된 디렉터리나 파일을 분석해 개선여지를 찾는 방법이 있다.

3. 어떻게 하면 RAM을 효율적으로 관리할 수 있을까?
   기존 방안에는 네이티브 코드의 개선방법 및 시스템 메모리 할당 방법을 바꾸는 방법들이 있다.
   프레임워크에서 전체적으로 메모리를 많이 쓰는 파일이나, 램과 관련된 디렉터리나 소스를 분석해 개선여지를 찾는 방법이 있다.

4. 어떻게 하면 프레임워크에서 코드를 개선해 성능을 향상시킬 수 있을까?
   상수, 참조, 인터페이스, 메소드 등 최적화 방안을 찾아서 수정하는 방법이 있다.
   애초에 거의 모든 성능 향상 요건들이 프레임워크 속의 코드 개선으로 귀결된다.

5. 어떻게 하면 하드웨어를 조정해 성능을 향상시킬 수 있을까?
   기존 방안에서는 CPU, GPU 열관리 및 limit를 주면서 열완화를 통해 지속적인 성능을 유지하는 방법이 있다.
   성능에 영향을 주는 하드웨어의 영향을 생각해본 다음, 그와 관련된 프레임워크 소스 파일을 찾아 개선하는 방법이 있다.

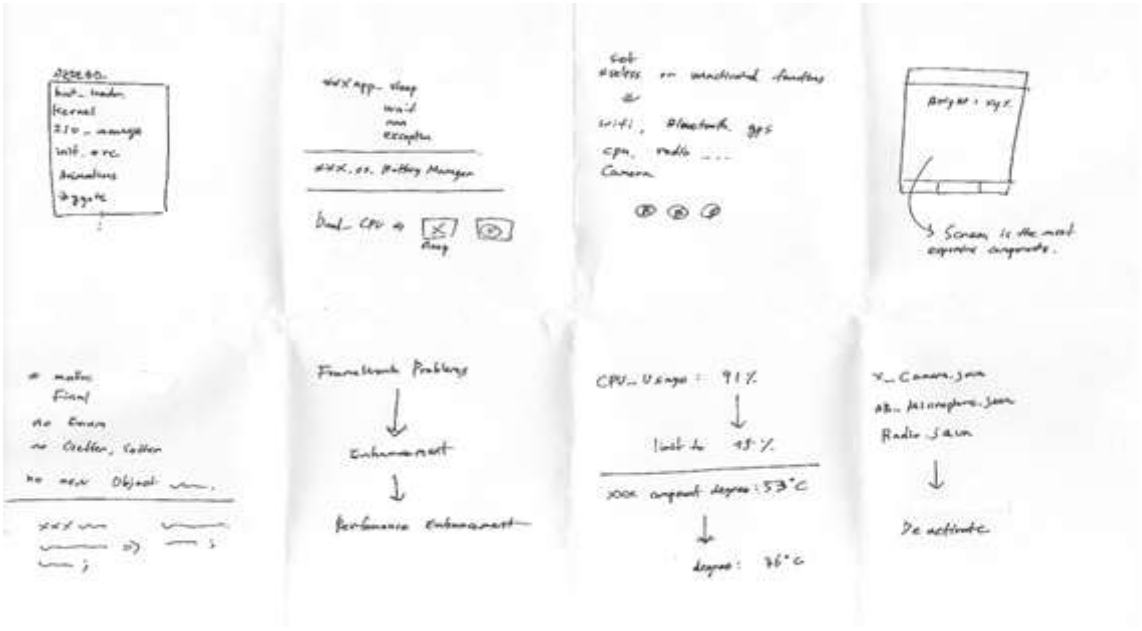
라이트닝 데모

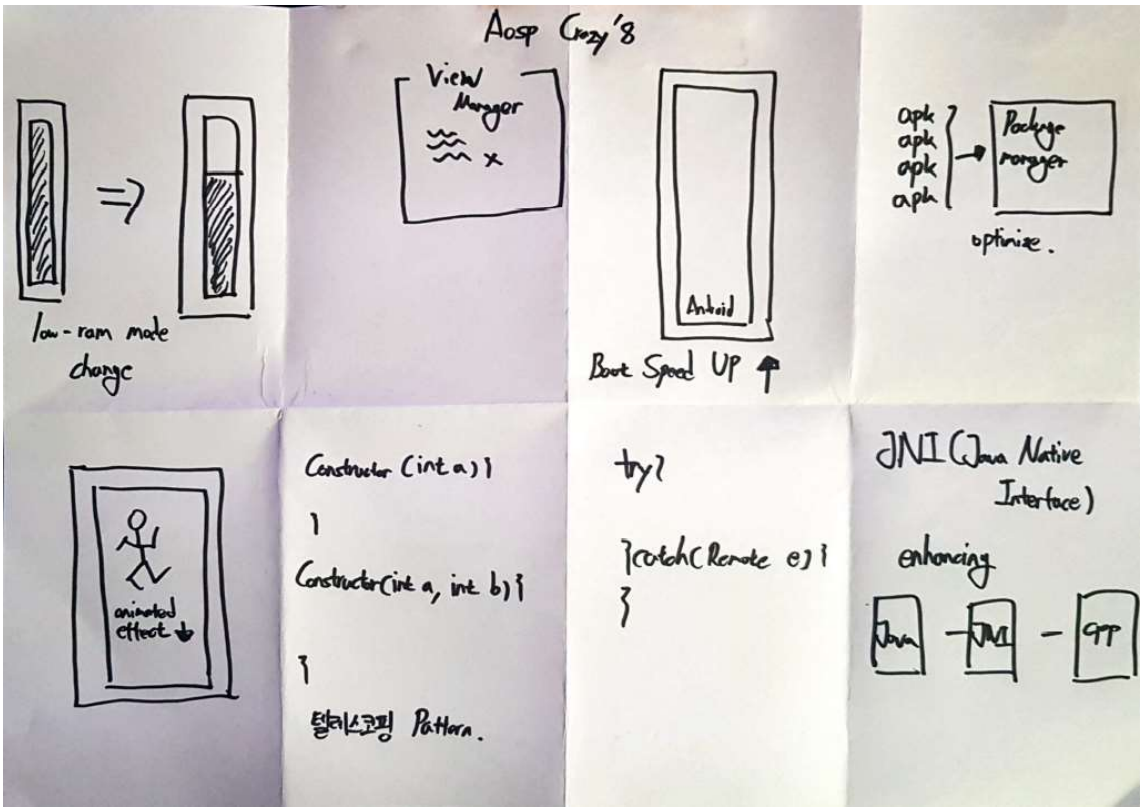
위 프로젝트에 안드로이드 개선을 위해서 다른 오픈소스 프로젝트에서 기여 된 것을 확인을 한다.

Android low row Ram enabler

```
#!/sbin/sh
# Variables
PROPERTY='ro.config.low_ram';
VALUE='true';

# System path
if [ ! "$(getprop 'go.build.system_root_image')" = 'true' ]; then
  system_path='/system';
else
  system_path='/system/system';
```

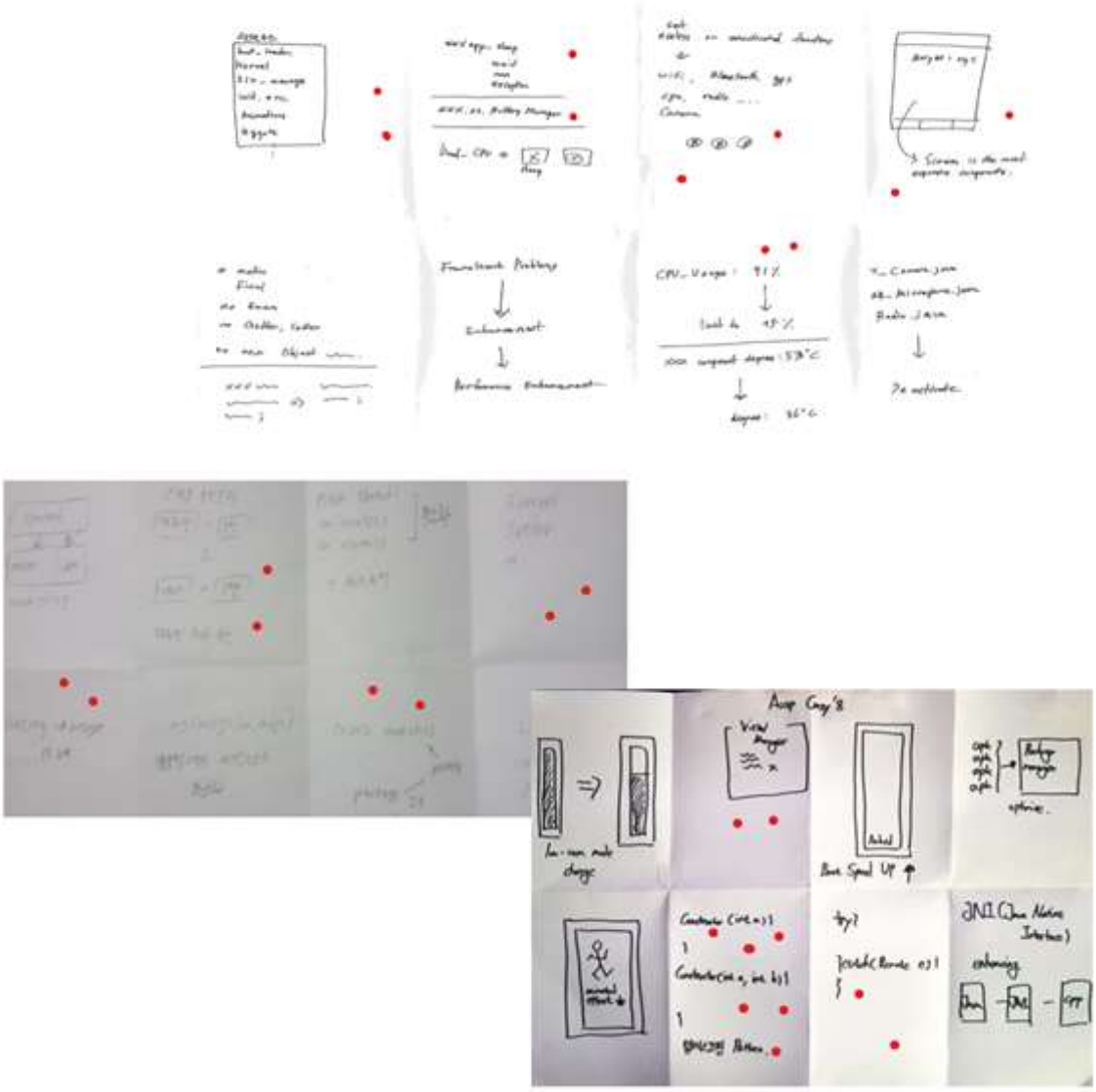
1.4. Crazy 8`s



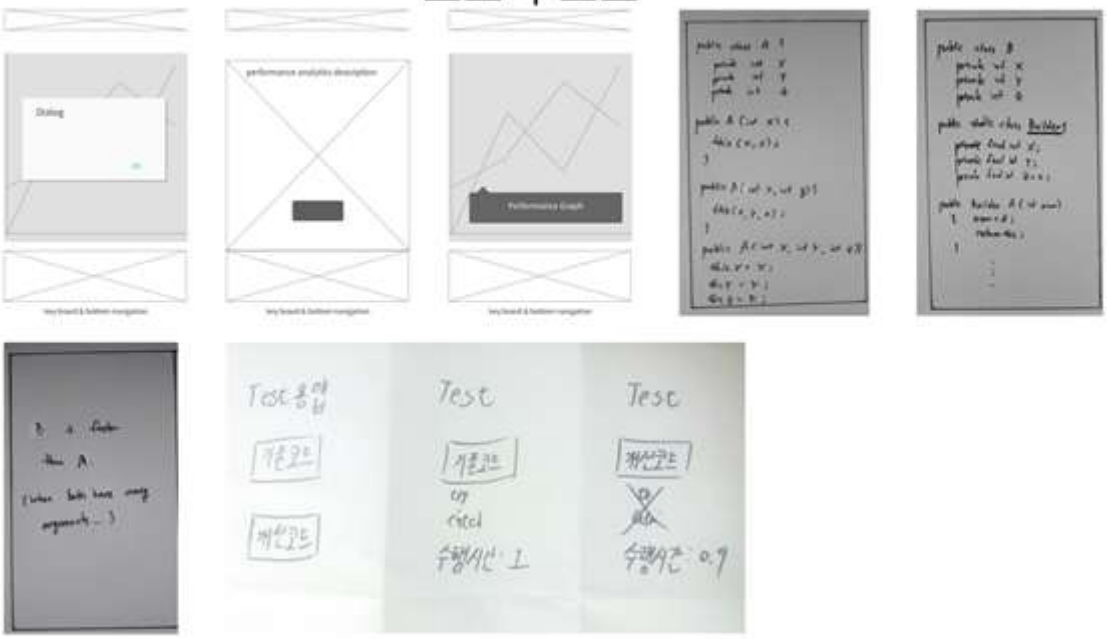

Aosp Crazy 8
View Manager
Android
Boot Speed UP
low-ram mode change
apk apk apk apk
Package manager
optimize.
animated effect
Constructor(int a){
}
Constructor(int a, int b){
}
템플릿코딩 Pattern.
try{
}catch(Remote e){
}
JNI(Java Native Interface)
enhancing
Java
JNI
C++

1.5. 투표결과

• 구글 설문지를 이용해 투표결과를 결산. (직관적으로 표시한 것)

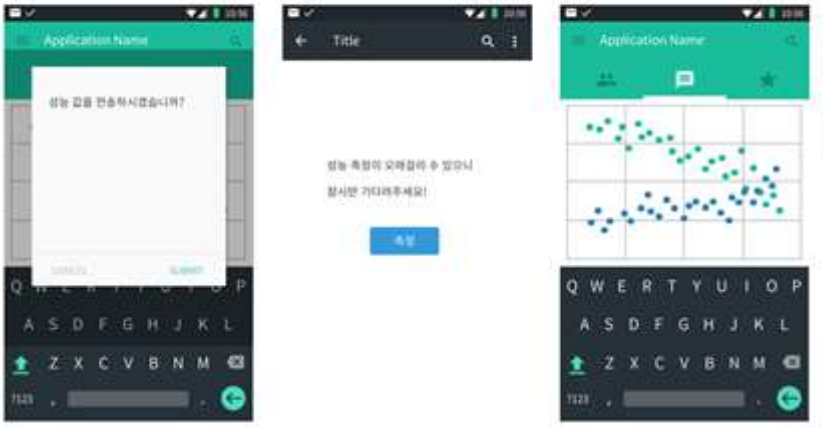


스토리 보드



1.6. 프로토타입

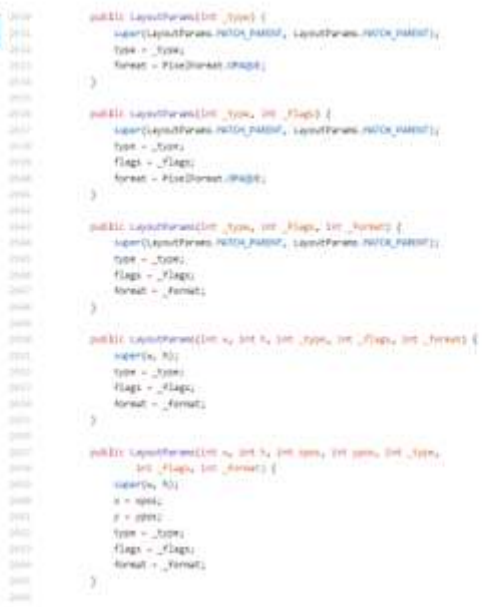
성능측정도구 제작

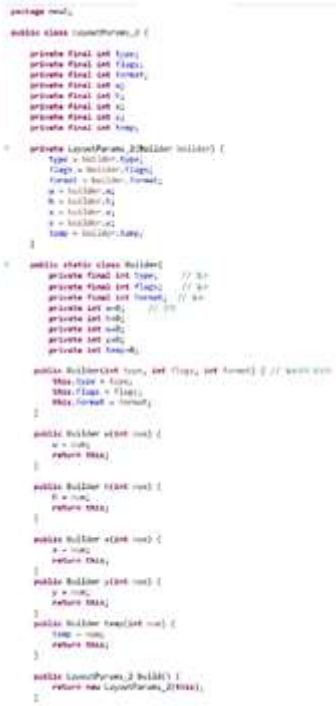


안드로이드의 성능을 개선한 빌드 버전에서 부팅 속도 및 측정할 수 있는 테스트 앱을 설치를 하여 수정된 부분에서 성능 개선이 되었는지를 확인하는 솔루션이다.

텔레스코핑 패턴을 수정하여 성능 개선 방법

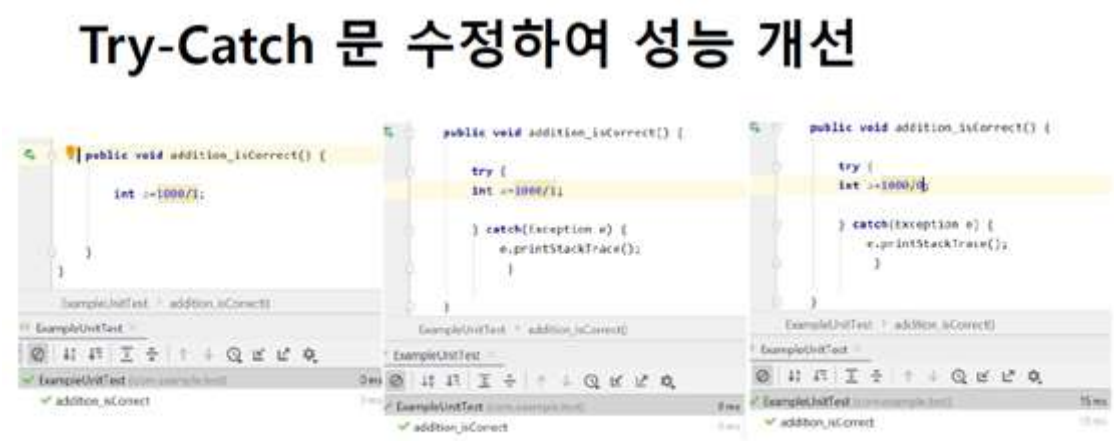
안드로이드 프레임 워크 파일안에 있는 WindowManager.java 파일이다. 실제로 Telescoping Constructor Pattern이 LayoutParams 클래스에서 확인이 가능하다.





(왼쪽 코드와 이어서)

텔레스코핑 패턴으로 임의로 구현했던 코드를 빌더 패턴으로 재 구현한 코드이다.



2. Software Engineering 문서

1. 문제정의서

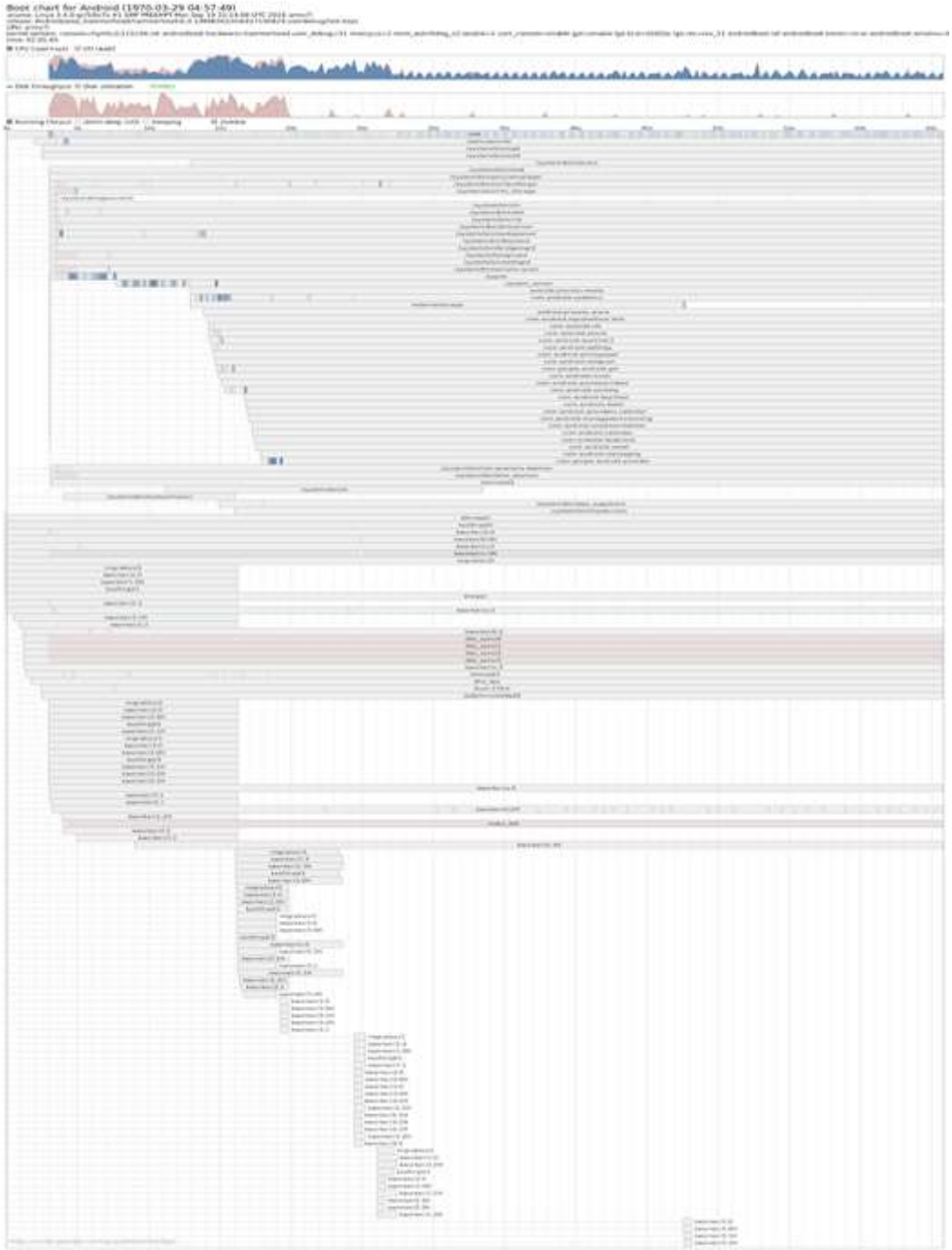
1.1 연구의 필요성

안드로이드 프레임워크레벨 계층은 안드로이드 시스템 서비스 실행, 자원관리, 애플리케이션의 제작, 실행, 같은 안드로이드가 구동하기위한 기본적인 기능들을 제공하는 역할을 하는 계층이다. 프레임워크 계층을 최적화한다면, 안드로이드 전반적인 성능향상을 기대할 수 있다.

1.2 연구의 목표 및 내용

안드로이드 프레임워크의 성능개선을 통해서 궁극적으로 애플리케이션들의 로딩시간을 단축시키는 것 프레임 워크가 제공하는 여러 기능 중에 화면의 출력, 관리와 관련된 View System - 애플리케이션 사용자 인터페이스 생성에 사용되는 확장 가능한 View들의 집합

Window Manager – 화면에 대한 정보, 배치 등을 관리하는 시스템 서비스 이 두가지 소스를 개선하는데 초점을 두기로 했다.



특히, 프로젝트의 목적은 안드로이드 프레임워크 레벨의 성능 향상이기 때문에 성능을 측정을 하는 Boot chart 나 Testing Application 등을 활용한다.

1.3 연구의 추진 전략 및 방법

소스에서 효율을 저하시키는 부분을 찾아내고, 수정하는 식으로 개선 테스트는 프로젝트비로 구입한 공기계로 진행. 구글에서 나온 레퍼런스 폰인 픽셀, 넥서스X5 쓰기로 했다. 버전은 8.1 (api27)

1.4 연구팀의 구성 및 과제 추진 일정

팀 구성: 팀원이 각자 맡은 소스코드를 분석하여 설명하고, 함께 문제점, 최적화방법 토론

1학기: 안드로이드 프레임워크의 기본적인 개념공부 및 기본적인 최적화 익히기

2학기: 본격적인 코드분석과 최적화 기존, 최적화기법의 분석, 개선

2. 요구사항명세서

2.1. 외부 인터페이스 요구사항

사용자는 Google Nexus 5X 스마트폰을 이용해서 안드로이드 8.1의 개선된 소스코드로 인한 성능향상을 측정한다. Android 8.1 (API Level 27)/안드로이드 최초로 최소 커널 버전이 지정됐다.

2017년 제작된 SoC를 사용한 새 제품은 반드시 4.4 이상을 써야 하며 다른 SoC를 달고 출시될 경우 최소한 3.18 버전을 사용해야 한다. 다 만, 커널 버전이 3.18 이하여도 업그레이드 기기에 한해 원하면 쓸 수 있다. 이는 Nexus 5X와 Nexus 6P 등 구형 구글 기기에도 적용된 사항이라 Nexus 5X의 경우 여전히 커널 버전이 3.10.73이다.

본 요구사항을 작성을 하는 데에 있어서 프레임워크 level 특히, Java Framework level 에서 작성을 하였으며 이를 통하여 application level 이 아닌 OS 자체에서의 성능 향상을 기획을 하고 있다.

2.2. 기능 요구사항

요구사항 분류		기능
요구사항 번호		SFR-001
요구사항 명칭		성능 테스팅 애플리케이션 권한 요구사항
요구사항 상세설명	정의	성능 테스팅을 할 수 있는 권한 및 기능 제공
	세부 내용	◦ 권한 및 기능을 제공하고 성능 테스팅 중 문제가 발생할 경우 이를 통제할 수 있는 통합 권한 및 기능 구축 - GUI 기반 관리도구 제공 - 알고리즘 및 매개변수의 편집, 수정, 삭제 등 기능 - 기존 성능 측정 결과의 수정, 삭제 등 기능
산출정보		성능 테스팅 알고리즘 및 소스코드 개선 계획
관련 요구사항		

2.3. 성능 요구사항

요구사항 분류		성능
요구사항 번호		PER-001
요구사항 명칭		Constructor Pattern 차이로 인한 실행 시간
요구사항 상세설명	정의	Telescoping Constructor Pattern 대신 Builder Constructor Pattern 를 이용해 단축된 실행시간 목표
	세부 내용	◦ Telescoping Constructor Pattern 대신 Builder Constructor Pattern 을 이용해 더욱 향상된 처리 소요 시간을 갖는다. • 단, 향상된 처리 소요시간을 명확히 단정할 수는 없다.
산출정보		Builder Constructor Pattern 구성 계획
관련 요구사항		

요구사항 분류		성능
요구사항 번호		PER-002
요구사항 명칭		예외처리로 인한 실행 시간
요구사항 상세설명	정의	무분별한 예외처리를 개선하여 단축된 실행시간 목표
	세부 내용	◦ 무분별한 예외처리 소스코드를 개선해 더욱 향상된 처리 소요 시간을 갖는다. * 단, 향상된 처리 소요시간을 명확히 단정할 수는 없다.
산출정보		예외처리 소스 코드 구성 계획
관련 요구사항		

요구사항 분류		성능
요구사항 번호		PER-003
요구사항 명칭		Testing Application
요구사항 상세설명	정의	요구사항 개선 목록 성능 확인 어플리케이션
	세부 내용	◦ Telescoping Constructor Pattern 대신 Builder Constructor Pattern 를 이용 단축된 실행시간으로 개선하면서 성능시간 측정을 확인하고 개선점을 확인한다. ◦무분별한 예외처리 소스코드를 개선해 더욱 향상된 처리 소요 시간을 측정 확인하고 개선점을 확인한다.
산출정보		예외처리 소스 코드 구성 계획
관련 요구사항		

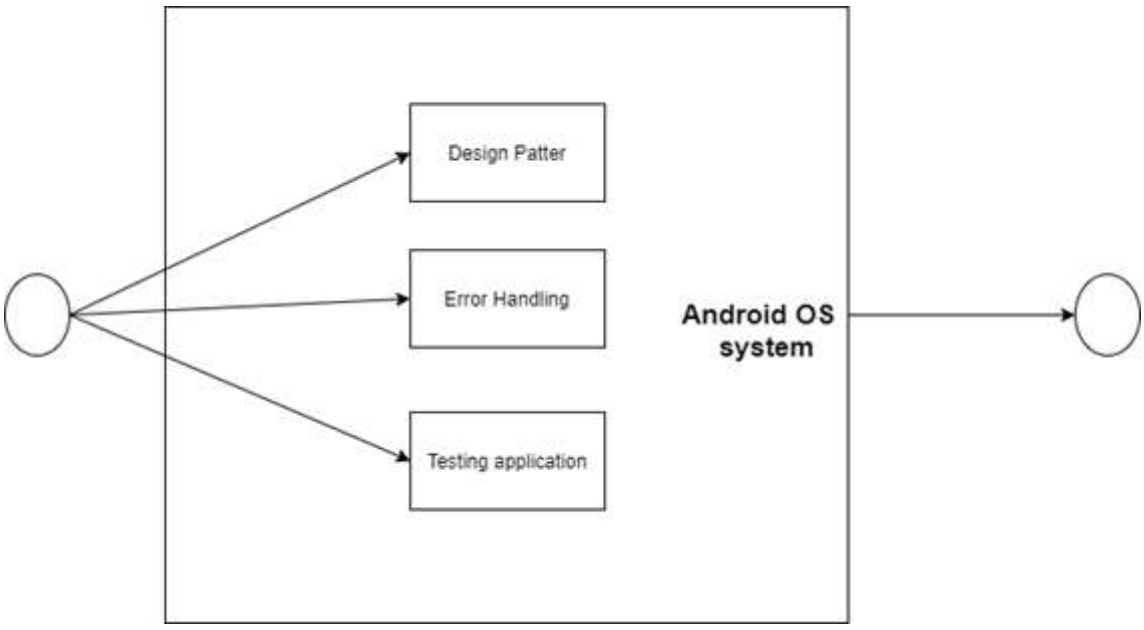
2.4. 소프트웨어 품질속성

요구사항 분류		품질
요구사항 번호		QUR-001
요구사항 명칭		신뢰성(reliability)
요구사항 상세설명	정의	신뢰성 개념 정의
	세부 내용	◦ 성능 측정 중 오류가 발생하는 즉시 사용자에게 관련 메시지를 공지하고 측정을 중단할 것
산출정보		
관련 요구사항		

이러한 최소 요구 사항을 통하여 수정하고자 하는 안드로이드 프레임워크 레벨에서의 성능 향상과 그 측정을 한다. 특히나 주요 요구사항 3가지 Constructor pattern, Error handling, Testing Application 등의 요구사항을 확인하고 검증을 하는 것이 필요하다.

3. 유스케이스

3.1. 유스케이스 다이어그램



3.2. 유스케이스 명세서

Usecase 이름	디자인 패턴
ID	1
간략 설명	관습적으로 구현된 텔레스코핑 패턴을 빌더 패턴으로 변경
Actor	
Pre-Conditions	- 상속 클래스, 인터페이스 상태 확인
Main Flow	1) View System, View Manger에서 텔레스코핑 패턴으로 구현된 메소드, 생성자 패턴 탐색 2) 상속 인터페이스, 클래스 상태 확인 3) 파라미터 적정성 평가 4) 빌더 패턴 변경 5) 연관되어진 클래스 변경
Post-Conditions	- 텔레스코핑 패턴으로 구현된 생성자 → 빌더 패턴으로 구현
Alternative Flow	3-1) 텔레스코핑 패턴으로는 구현이 되어 있으나 파라미터 인자가 단순하거나 혹은 파라미터 인자가 적은 경우 텔레스코핑 패턴으로 유지

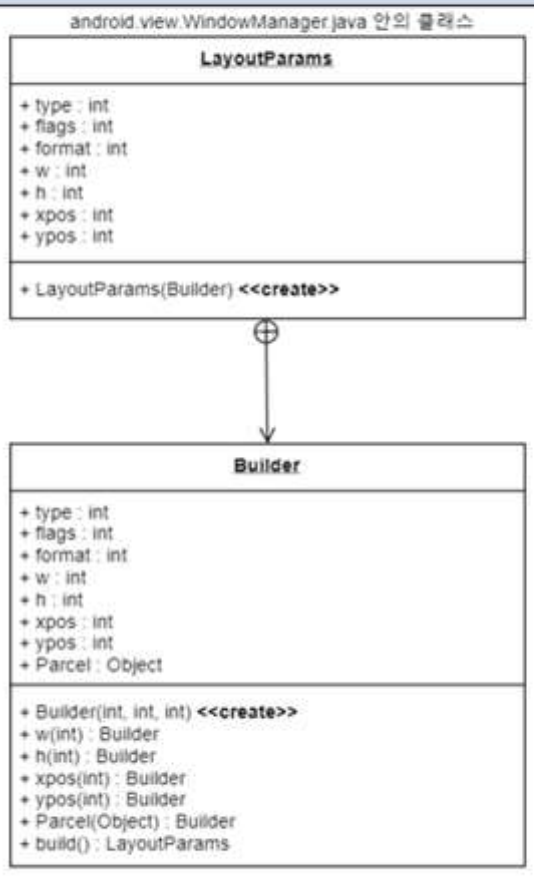
3.2. 유스케이스 명세서

Usecase 이름	디자인 패턴
ID	1
간략 설명	관습적으로 구현된 텔레스코핑 패턴을 빌더 패턴으로 변경
Actor	
Pre-Conditions	- 상속 클래스, 인터페이스 상태 확인
Main Flow	1) View System, View Manger에서 텔레스코핑 패턴으로 구현된 메소드, 생성자 패턴 탐색 2) 상속 인터페이스, 클래스 상태 확인 3) 파라미터 적정성 평가 4) 빌더 패턴 변경 5) 연관되어진 클래스 변경
Post-Conditions	- 텔레스코핑 패턴으로 구현된 생성자 → 빌더 패턴으로 구현
Alternative Flow	3-1) 텔레스코핑 패턴으로는 구현이 되어 있으나 파라미터 인자가 단순하거나 혹은 파라미터 인자가 적은 경우 텔레스코핑 패턴으로 유지

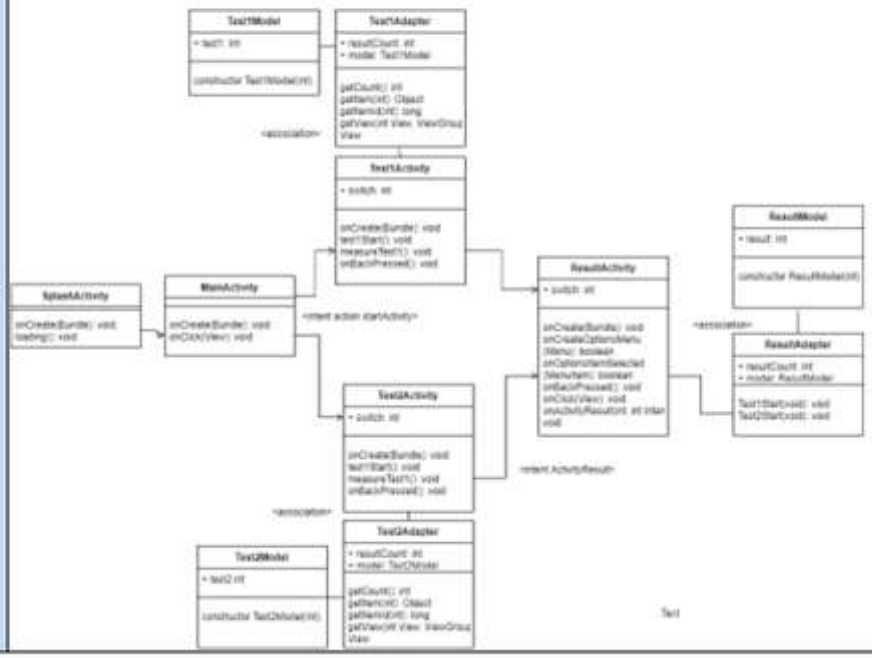
Usecase 이름	Error Handling(오류처리)
ID	2
간략 설명	잘못된 try-catch 같은 오류 핸들링 구문의 경우 성능 저하를 초래 할 수 있기 때문에 개선이 필요하다.
Actor	
Pre-Conditions	- 상속 클래스, 인터페이스 상태 확인
Main Flow	1) View System, View Manger에서 RemoteException을 catch 하는 비어있는 구문 탐색 2) 상속하는 클래스 및 인터페이스 상태 확인 3) Throw 경우를 제외한 일반적인 관습 핸들링 제거
Post-Conditions	
Alternative Flow	2-1) 상속하는 클래스가 Exception 을 Throw 를 하는 경우는 안드로이드 OS 에서의 DeadObjectException 을 처리를 해야 하는 구문으로 이를 제거 시 어플리케이션 작동에 있어서 문제점 발생 함.

Usecase 이름	Testing application (테스트 어플리케이션)
ID	3
간략 설명	수정된 안드로이드의 성능을 측정을 하기 위하여 안드로이드 플랫폼에서 제공을 해주는 것이 아닌 직접 만들어서 제공
Actor	
Pre-Conditions	- 이미지, 비디오 등의 View 를 표현 할 수 있는 application
Main Flow	1) Image View 를 표시를 할 수 있는 View 2) Video View 를 표시를 할 수 있는 View 3) 성능 측정을 하여 저장을 하는 Activity 4) 측정된 데이터를 전송을 하는 Activity
Post-Conditions	
Alternative Flow	3-1) 성능 측정을 함에 있어서 어플리케이션 레벨 뿐만 아니라 하드웨어 레벨 또한 성능 측정을 하여야 하기 때문에 JNI, NDK 를 활용을 하여서 어플리케이션을 구성을 해야 한다.

4. 클래스 다이어그램

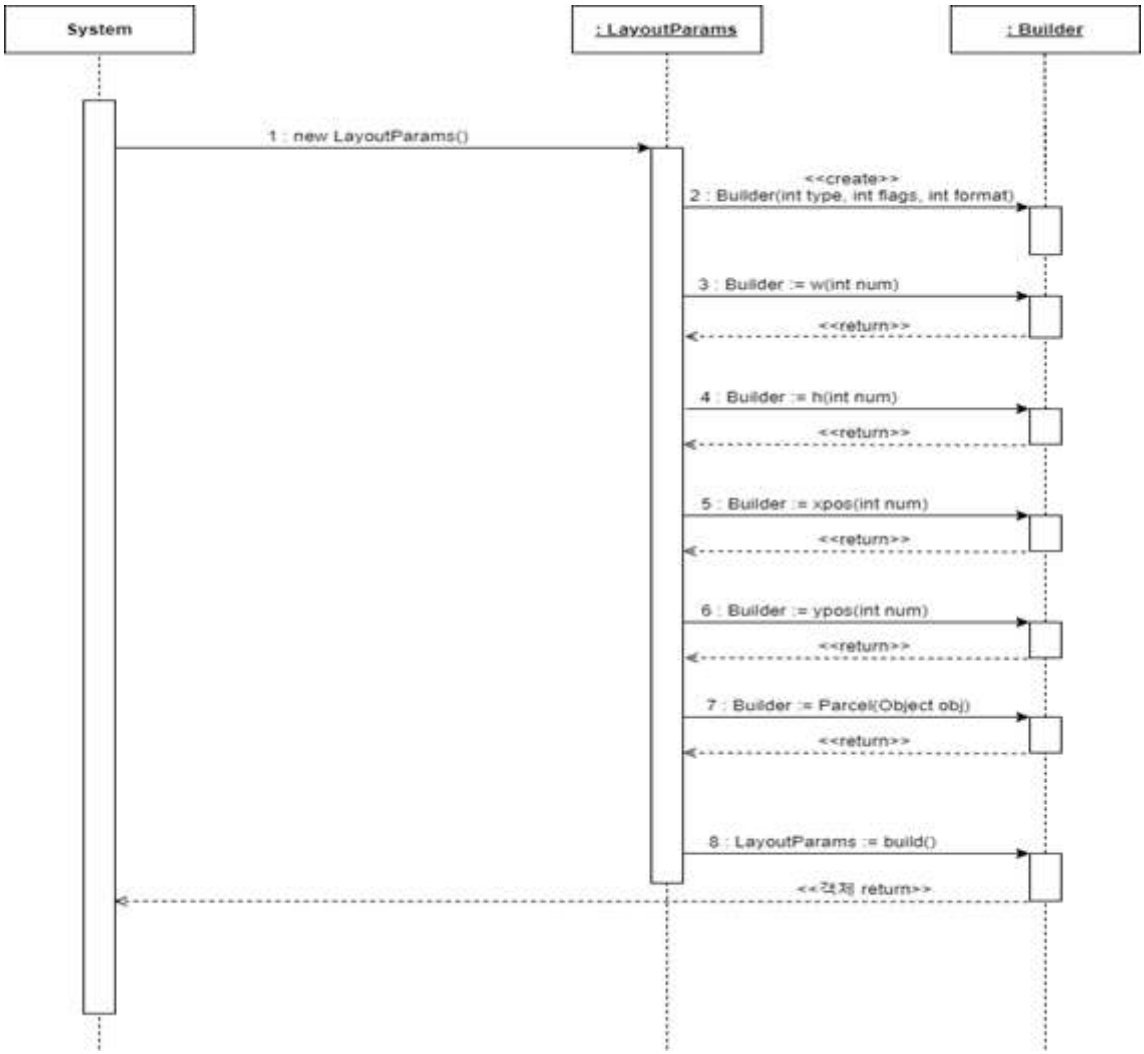
Use Case 이름	디자인 패턴
실현 클래스 목록	

Use Case 이름	예외처리
실현 클래스 목록	block_check +log :int optimize(file) : void check():void remove() : void rollback() :void

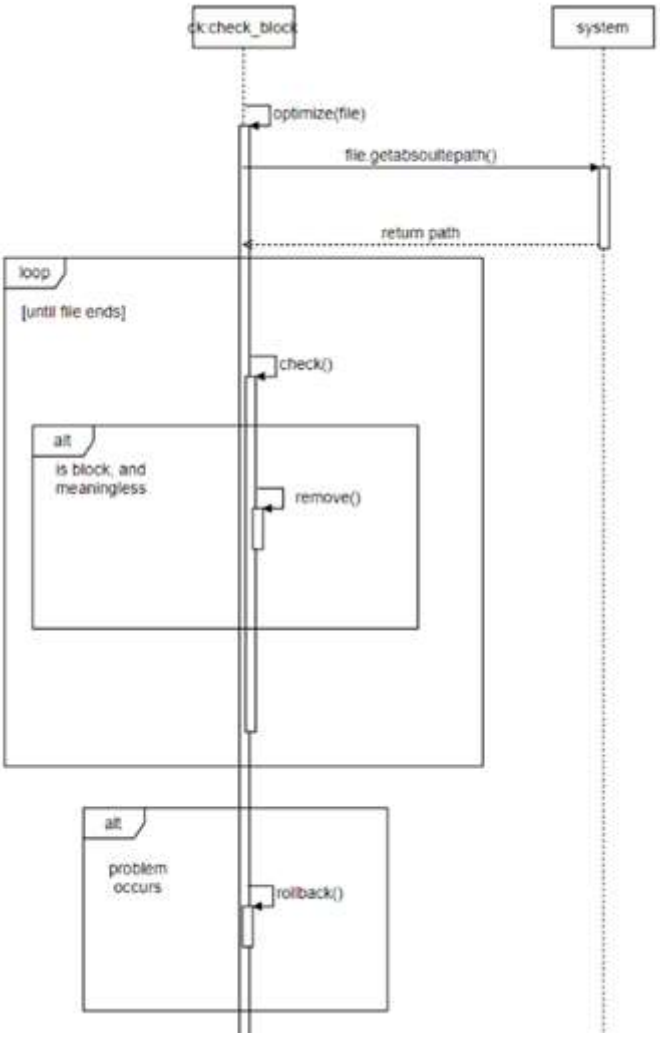
Use Case 이름	Testing Application
실현 클래스 목록	

5. 시퀀스 다이어그램

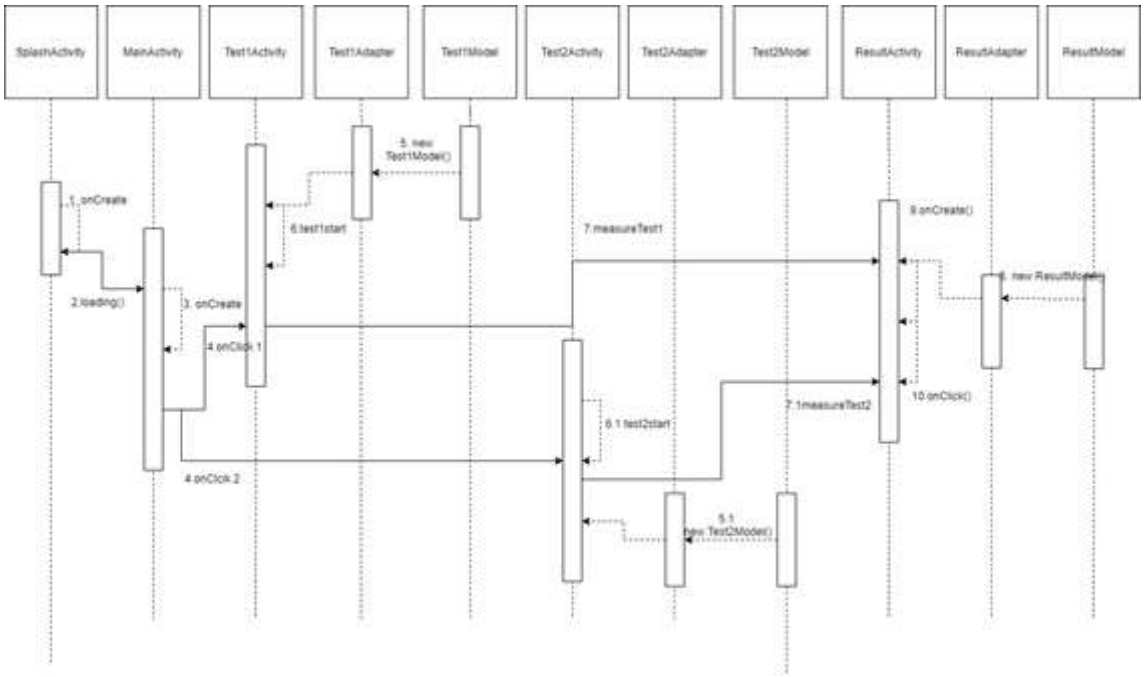
5.1 WindowManger



5.2 Error Handling



5.3 Testing Application



특히, Testing Application의 경우는 WindowMangerService 를 직접적으로 사용을 하는 AlertDialog.Builder 의 부분과 커스텀 서비스를 구성을 하고 이를 WindowManager 를 사용하여 추가적인 뷰가 화면에 표시가 되도록 하는 방법으로 성능 향상을 측정을 한다.

이를 위해 5개의 주요 Activity 를 사용을 하고 어플리케이션의 주요 로직을 구성을 한다.

3. 멘토링

멘토링을 진행을 하면서 지도교수님 이신 조은선 교수님과, LG 전자 이원영 연구원님의 도움으로 진행을 할 수 있었다. 주 1회 미팅을 가지면서 안드로이드 프레임워크 레벨에서의 운영되는 원리와 IPC → Binder 를 통하여 네이티브 레벨과 자바레벨의 구성, 안드로이드 OS 아키텍처의 구성 원리를 공부를 하였다.

김재현 : https://keelim.github.io/AOSP/

이송무 : https://songmoolee.github.io

임진현 : https://nivkhdif.github.io

<안드로이드 프레임워크 레벨 정리>

당장 WindowManagerGlobal에서 thread를 사용하는부분에서 전체 객제체로 lock을 걸고 접근하고 있어서 개선의 여지가 있지 않을까 생각됩니다.

LG전자 멘토님과 함께 학생들이 계획에 적어준 1~4에 대해 논의했습니다.

1. 커스텀 롬, 커널 사례 연구=> 사례 연구에서 끝나지않고, 커스텀 빌드 후 코드를 수정하여 개선한다면 괜찮을 것 같습니다.

2 Treble에 대한 멘토님의 의견입니다.
=> 아직 완벽한 기능이 아니라고 생각됩니다. 구글에서 처음에 시작할때는 이론적으로는 탄탄했으나 실제로 100% 적용되어 있진 않다고 알고 있습니다. 그래서 Treble이 어떤 목표로 만들어졌고 어떤식으로 hardware dependency를 끊어 낼수 있는가? 와 같은 이론을 연구해보는것까지는 괜찮을텐데 뭔가 눈에 보이는 결과물을 만들어서 실행해본다거나 하는건 어려울것 같습니다.

3. 안드로이드 OS를 적용함에 있어서 Pixel과 같은 AOSP가 적용되는 기기와는 별도로 각 제조사별 AOSP 커스텀에 관한 방법 연구에 대한 멘토님 의견
=> 첫번째 주제에 연결되는 내용일텐데, 순수 AOSP에 제조사별 커스텀이 어떤식으로 들어가느냐를 비교하는건 어렵습니다. 제조사에서 만드는 코드들은 Open source license외에는 모두 대외비 이기 때문에 이를 외부에서 확인하기에는 어려움이 있습니다. 다만 연구 자체에 의미를 둔다면, 동작에 대한 dump나 로그 등으로 어느정도의 구조 파악은 해볼 수 있습니다. 타겟에 실제 올려보고 동작하기에는 제한이 많습니다.

4. Android Go 대한 멘토님 의견
=> 일단 Framework에서 어느정도 저사양 단말을 위해 기능 제한이나 feature off 등을 하는데, 이를 외부에서 따로 커스텀 빌드 한다거나 하는건 조금 제약이 있을것 같은데요, 따라서 만약 이쪽을 연구해보고 싶다 하면, Android Go를 위한 적용 기술을 일반 AOSP에 적용해본다거나 하는 방법은 해볼만 할 것 같습니다.

뭔가 잘 진행되고 있는것 같고 잘 하고 있으신것 같네요!
내용에 대해 조금 커맨트 하자면,

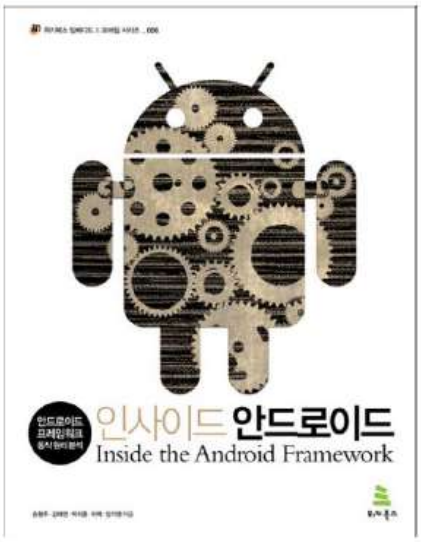
1번 아이디어의 경우 알고계신것 처럼 작년 조에서 했던 내용으로, 어느정도의 결과물을 얻었던 것으로 알고 있습니다.
2번의 경우 동기화 라는게 생각보다 중요한 문제인데, Android framework이 지금 보고 있는 WindowManager는 여러개의 framework service들 중 하나기 때문에
자칫 다른 서비스와의 deadlock state를 만들어 낼 수 있어 조심히 다뤄야 하는 부분입니다. 코틀린을 포함시키는것도 아마 빌드 문제부터 이것저것 고려해야 할게 많아질것 같긴 합니다.

일단 현재 진행중인 아이디어로 잘 활용해보시면 간단한것 같으면서도 괜찮은 결과를 얻어볼 수 있을것 같네요.
그리고 말씀하신대로 디테일 하게 WindowManager 중점으로 보는것도 좋은 방향입니다.

SystemServer로 부터 올라오는 서비스가 정말 다양한데, 모든 서비스들이 다 얽혀있기 때문에 명확한 방향을 가지고 하나만 디테일 하게 보는게 더 좋을것 같다는데 동의합니다.

잘 하고 계신것 같으니 조금 더 힘 내시고 궁금한점 있으면 메일 주세요.

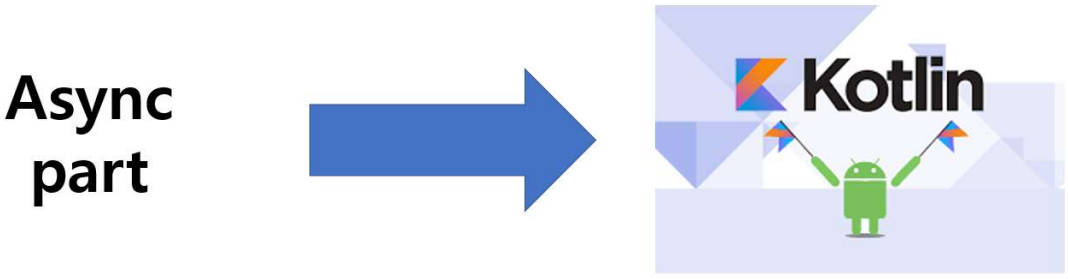
감사합니다.



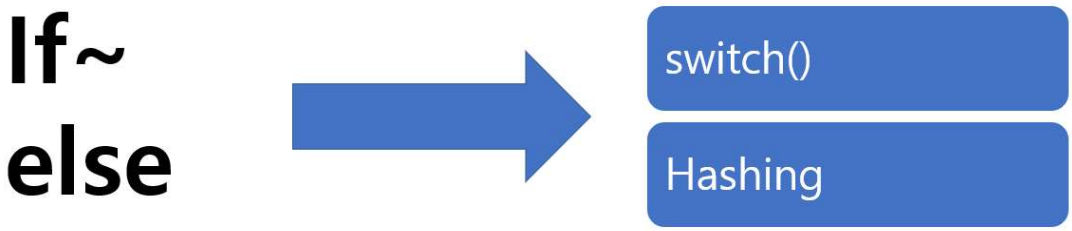
프레임워크 단의 내용을 확인 하기 위하여 SDK (Software Development Kit) 수준인 PDK(Platform Development Kit) 수준으로 진행을 하였다.

이러한 스터디 과정을 거치면서 프레임워크를 일정 부분 수정을 함에 있어서 오류나 기타 논리적으로 맞지 않는 부분이 생길 경우 수정을 할 수 있도록 조치 하였다.

또한, 추가적으로 멘토링 과정에서 성능 향상을 위하여 나온 아이디어를 정리를 하자면



WindowManger 를 처리를 함에 있어서 중간의 windowGlobalLock 을 걸어두는 부분을 발견을 할 수 있다. 이를 Kotlin 의 Coroutine 이라는 비동기적인 처리를 활용을 하여 구성을 할 수 있도록 하는 방안



WindowManager 의 경우 addVIew() 메소드 안에서 error 처리를 하는 부분에서 파라미터 마다 if else 를 반복을 하는 패턴을 발견을 할 수 있었다. 이를 switch 나 hashing 통하여 좀더 간단한 바이트 코드를 통하여 진행이 될 수 있는지에 대한 방안

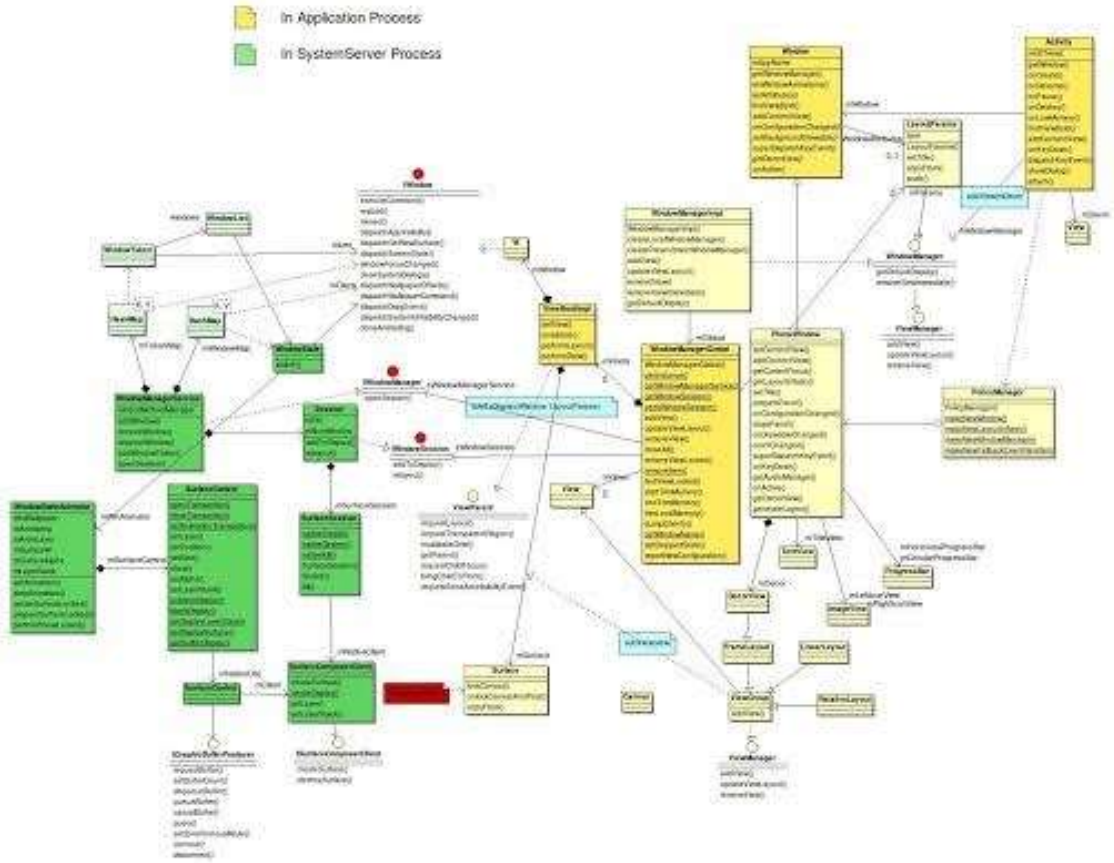
등의 아이디어가 나왔다.

4. 설문

등의 설문조사를 통하여 프로젝트 진행 방향을 수정 할 수 있었다.

5. 프로토타입 데모

프로토타이핑을 작성을 “WindowManager” 을 wm.addView() 를 통하여 “WindowManager” 의 성능 측정을 확인을 한다.

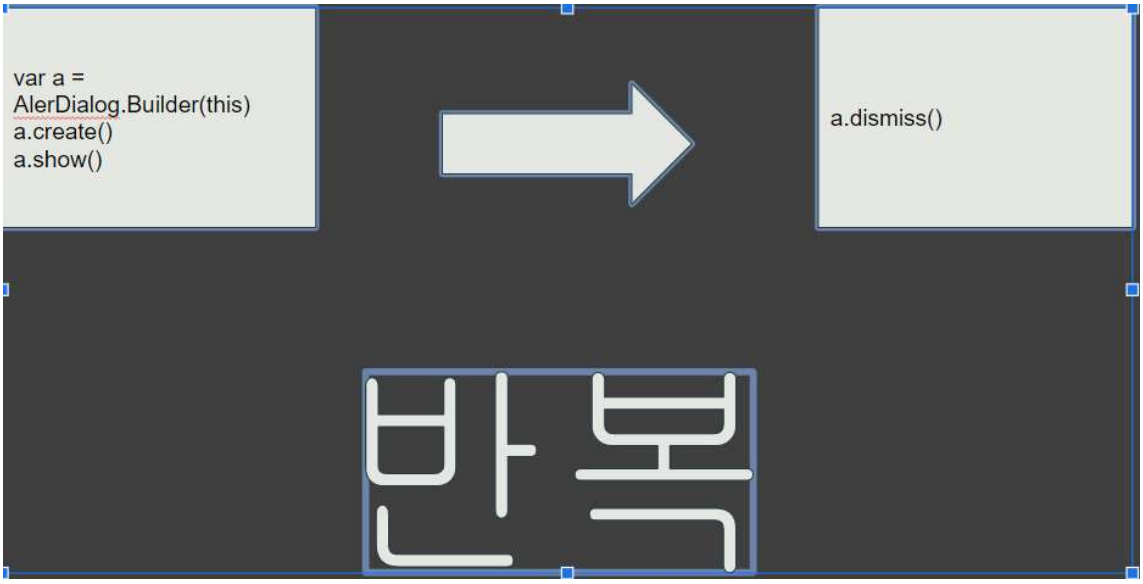


“WindowManager” 에서 연관된 클래스들을 나타낸다. 이 안에서 의 WindowManger 와 직접적으로 관련된 섹션을 수정을 하여 “WindowManager” 의 성능을 확인을 하는 Testing Application 을 작성을 한다.

5.1 Android Testing Application Specification

1. **compileSdkVersion: 29**
2. **buildToolsVersion: “29.0.3”**
3. **minSdkVersion: 28**
4. **targetSdkVersion: 29**
5. **compileOptions: sourceCompatibility = 1.8**
   targetCompatibility = 1.8
6. **kotlinOptions: jvmTarget = ‘1.8’**

5.2 Testing Application Logic



<수정 해야 한다.>

Testing Application Measurement Index

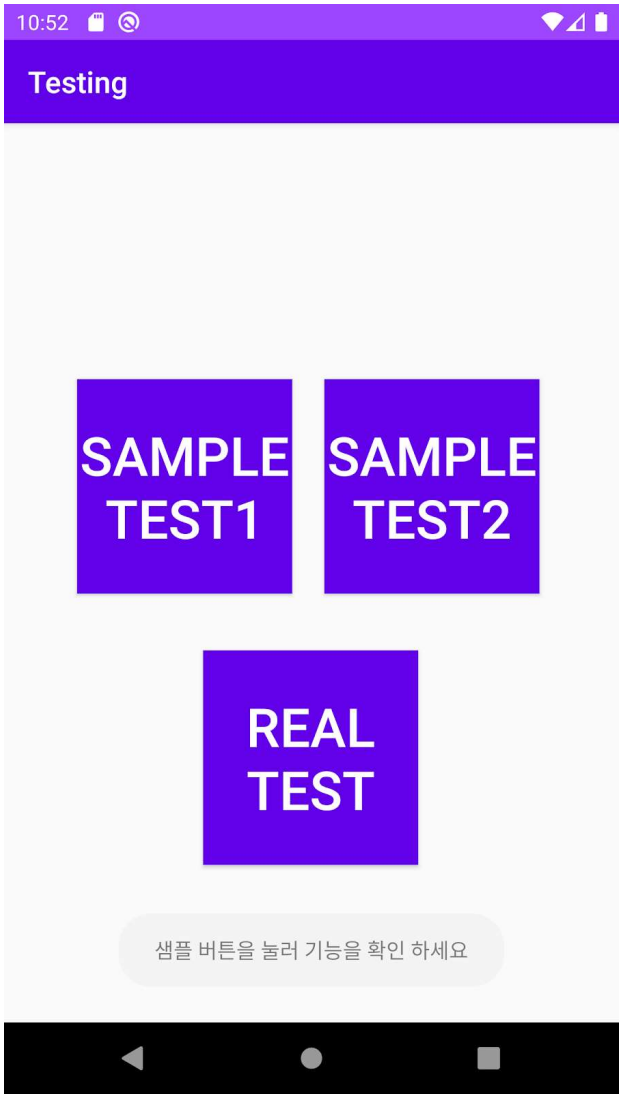
```
public AlertDialog create() {
    // Context has already been wrapped with the appropriate theme.
    final AlertDialog dialog = new AlertDialog(P.mContext, 0, false);
    P.apply(dialog.mAlert);
    dialog.setCancelable(P.mCancelable);
    if (P.mCancelable) {
        dialog.setCanceledOnTouchOutside(true);
    }
    dialog.setOnCancelListener(P.mOnCancelListener);
    dialog.setOnDismissListener(P.mOnDismissListener);
    if (P.mOnKeyListener != null) {
        dialog.setOnKeyListener(P.mOnKeyListener);
    }
    return dialog;
}
```

<AlerDialog.Builder().create()>

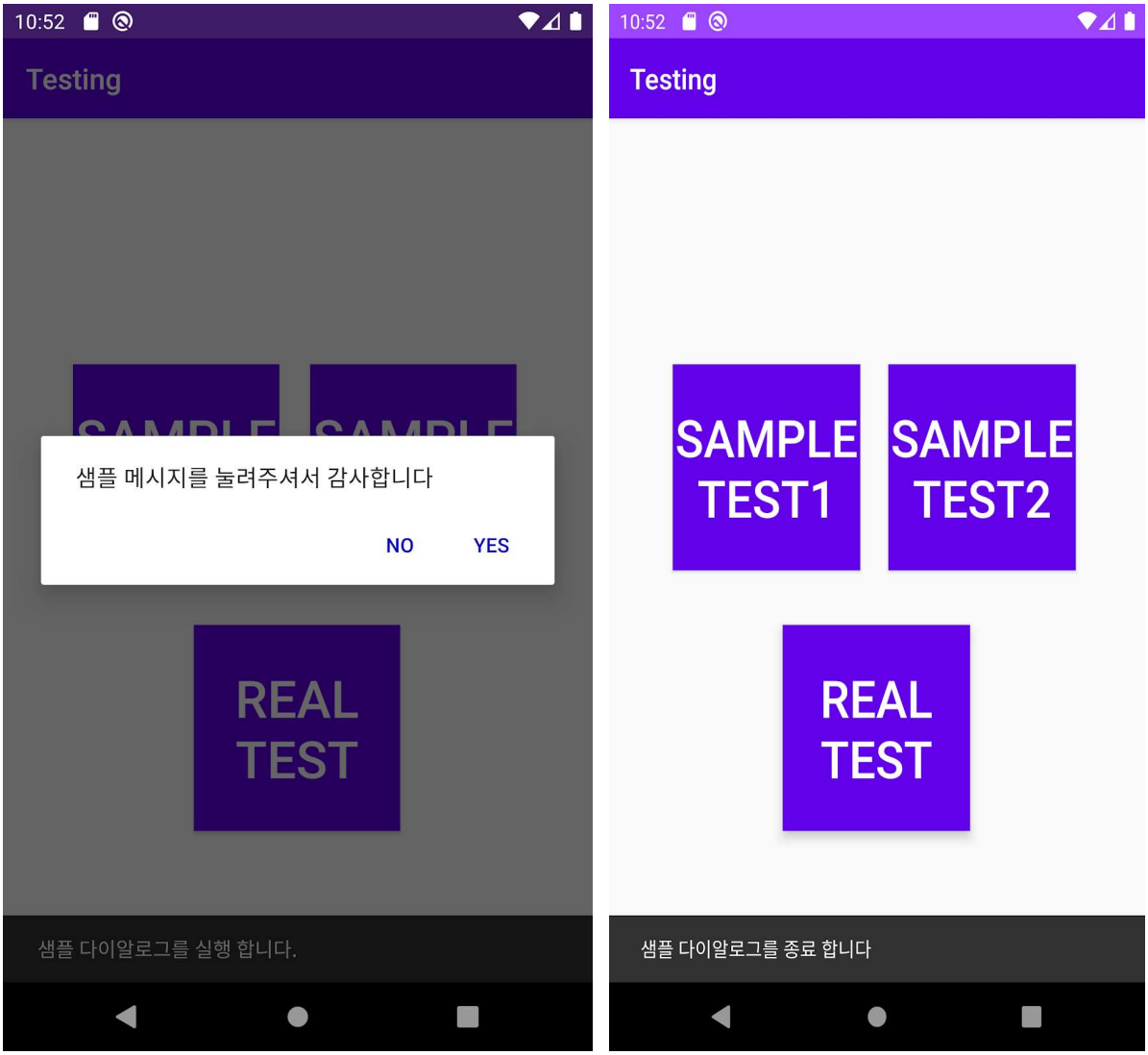
AlertDialog.Builder().create() 를 한후 AlertDialog.show() 를 호출을 한 경우 이 상태에서 화면에서의 표시해야 할 내용을 등록을 하고

```
public void show() {
    if (mShowing) {
        if (mDecor != null) {
            if (mWindow.hasFeature(Window.FEATURE_ACTION_BAR)) {
                mWindow.invalidatePanelMenu(Window.FEATURE_ACTION_BAR);
            }
            mDecor.setVisibility(View.VISIBLE);
        }
        return;
    }

    mCanceled = false;

    if (!mCreated) {
        dispatchOnCreate(null);
    } else {
        // Fill the DecorView in on any configuration changes that
        // may have occured while it was removed from the WindowManager.
        final Configuration config = mContext.getResources().getConfiguration();
        mWindow.getDecorView().dispatchConfigurationChanged(config);
    }

    onStart();
    mDecor = mWindow.getDecorView();

    if (mActionBar == null && mWindow.hasFeature(Window.FEATURE_ACTION_BAR)) {
        final ApplicationInfo info = mContext.getApplicationInfo();
        mWindow.setDefaultIcon(info.icon);
        mWindow.setDefaultLogo(info.logo);
        mActionBar = new WindowDecorActionBar(this);
    }

    WindowManager.LayoutParams l = mWindow.getAttributes();
    boolean restoreSoftInputMode = false;
    if ((l.softInputMode
            & WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION) == 0) {
        l.softInputMode |=
                WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION;
        restoreSoftInputMode = true;
    }

    mWindowManager.addView(mDecor, l);
    if (restoreSoftInputMode) {
```

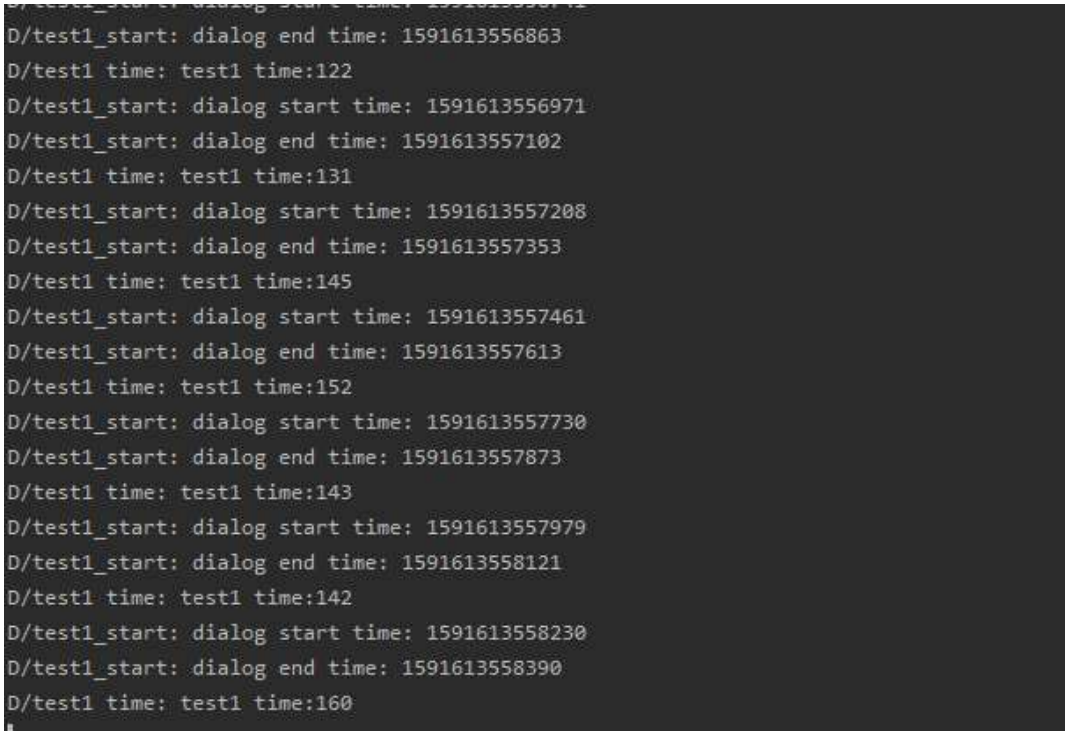
show() 메소드는 WindowManger에서의 LayoutParams 를 설정을 하고 WindowManger.addView(mDecor, l); 를 붙여주어 window 를 붙여 주면서 성능 상태를 확인을 할 수 있다.



<Sample → Test1Activity (AlertDialog Performance)>



<Sample → Dialog Test>



<Dialog Performance Measurement>

Dialog 계속 create() → show() → dismiss() 를 반복을 하면서 계속 성능 측정을 시작한다.