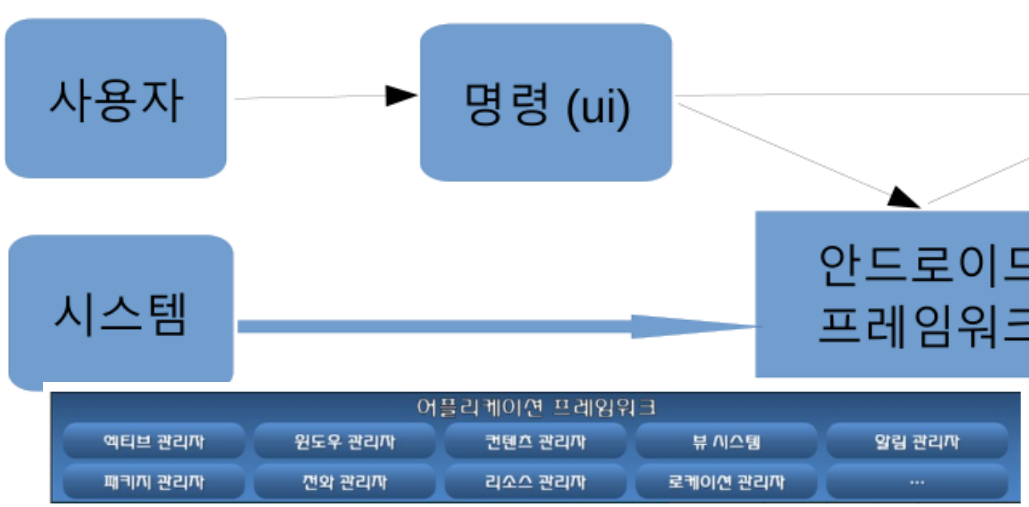
종합설계(aosp) survey/pitching

Aosp조, 201502104 임진현

요약-주제

- 주제 : 안드로이드 프레임워크 개선
- 담당교수 : 조은선 교수님
- 팀원 : 김재현 , 이송무 , 임진현
- 목표 : 안드로이드의 Framework Layer 를 개선하여
- 안드로이드의 속도 , 성능을 높히고
- 가능하다면 안드로이드 소스에 commit

요약-map //흐름을파악하는단계



해결할문제 : 더빠른출력

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

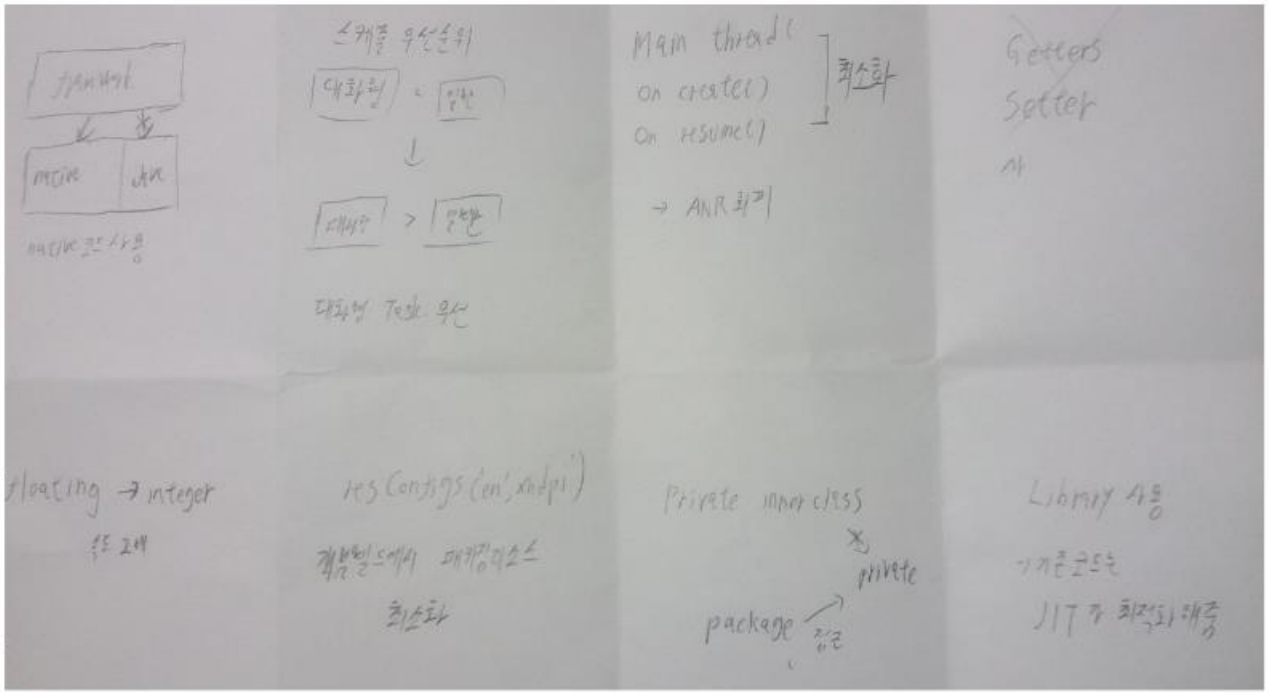
어떻게하면 윈도우매니저를 수정해 응답성을 높힐수있을까

요약-sketch　//해결책을 생각해내는단계

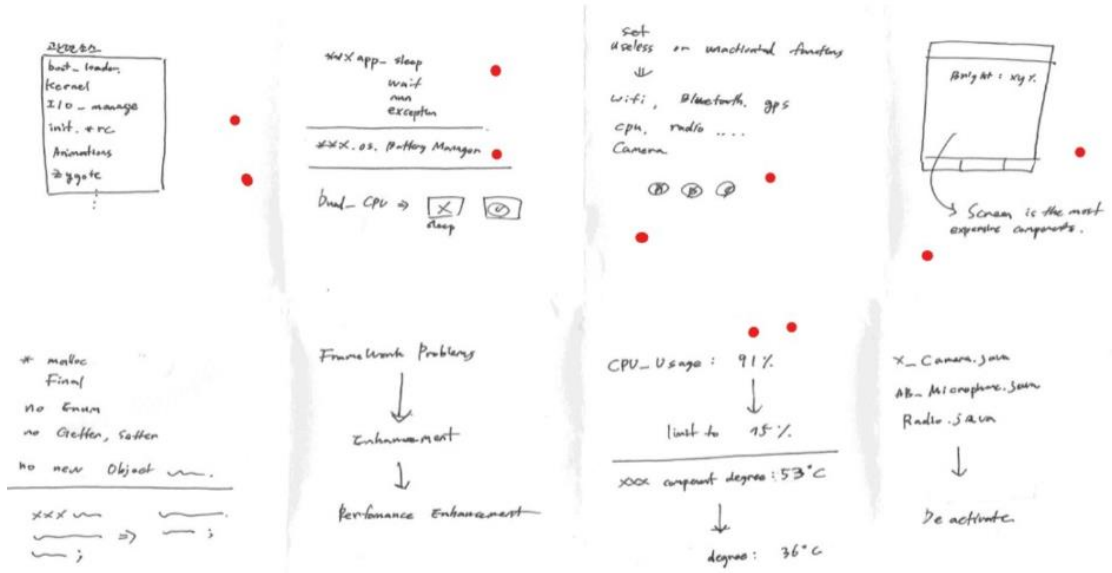
솔루션 +Crazy8

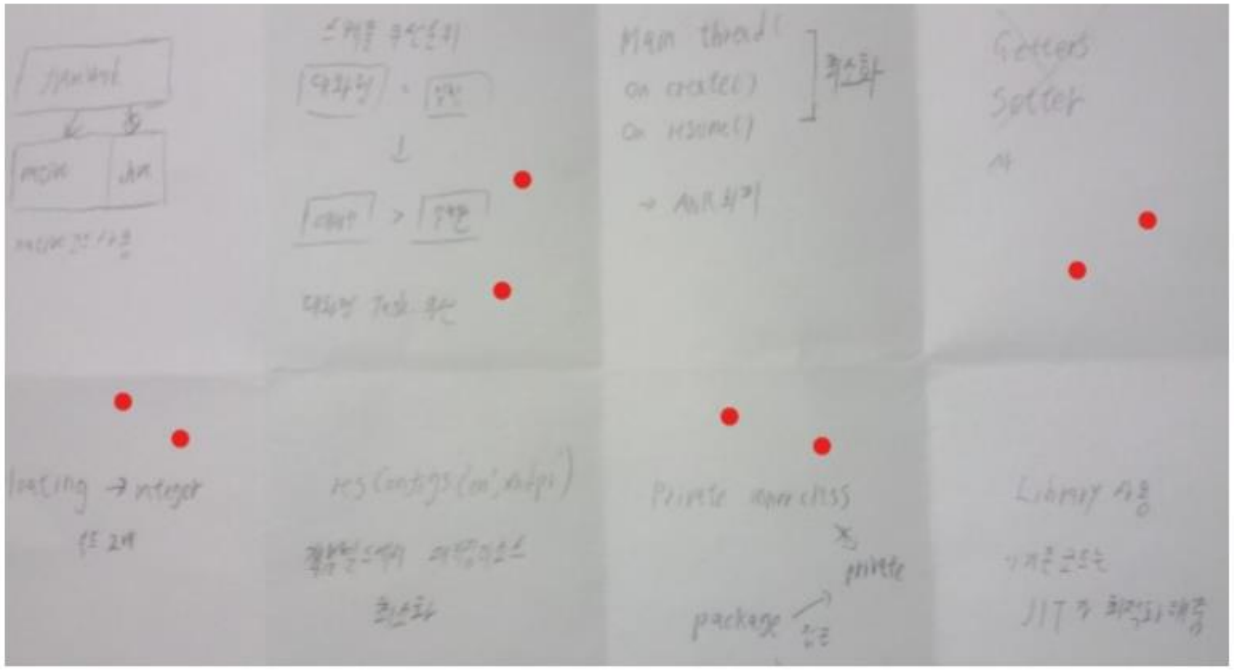
- 네이티브 코드를 우선사용
- 대화형 task 를 우선순위높게
- Main thread 는 최소화 >anr 방지
- Getters,setter 사용안하기 > 속도
- Floating point 사용안하기
- 개발빌드에서 패키징리소스 최소화
- private 에 접근할때

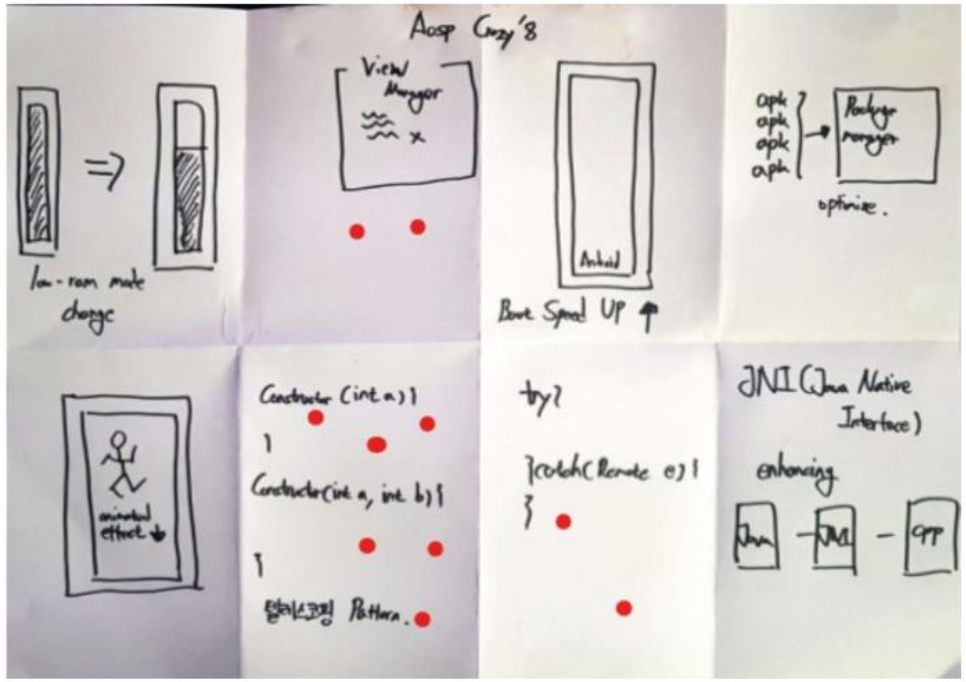
  private inner class 말고 package
- Library 사용 > JIT 최적화 가능



요약-solution투표



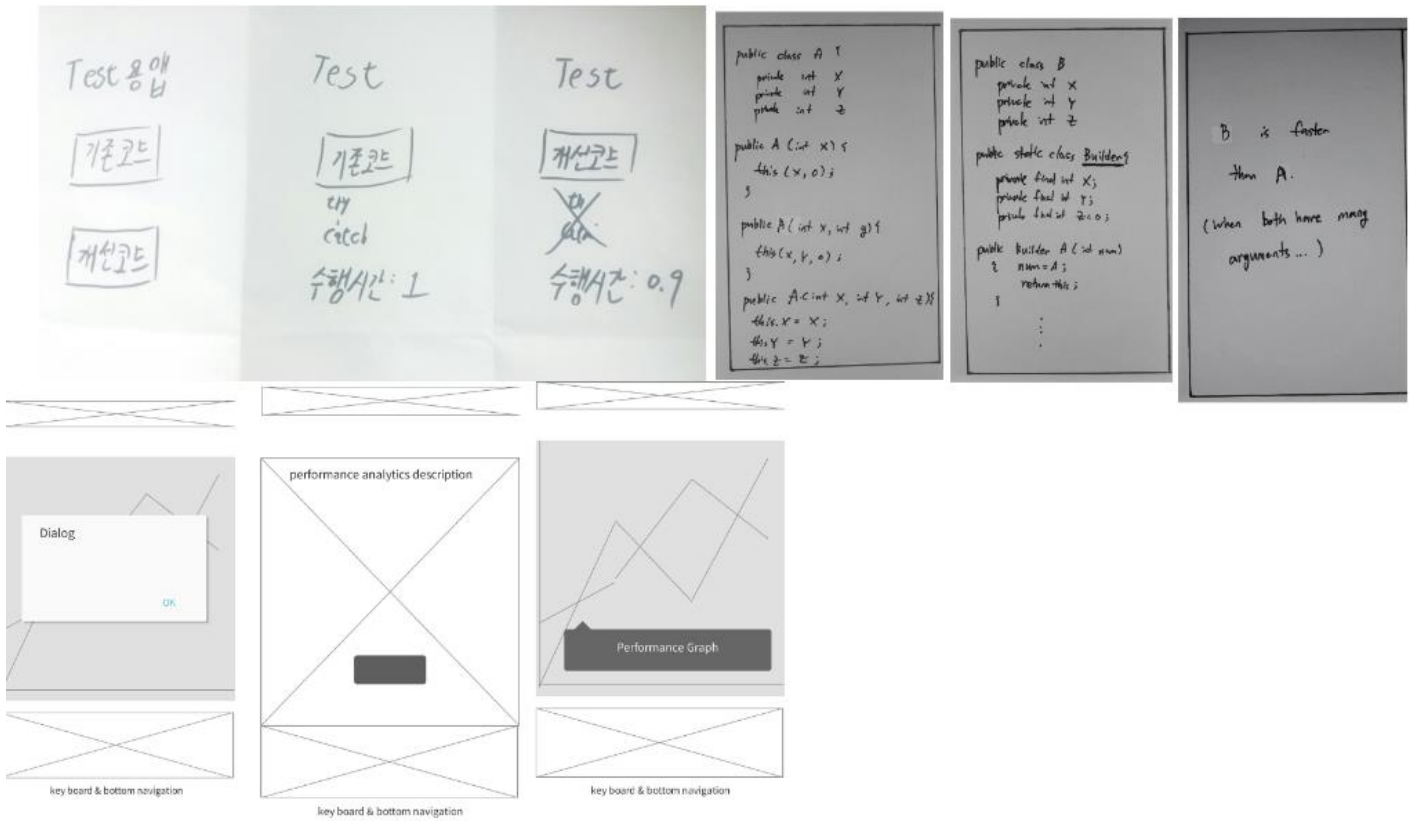




try/catch 블럭
텔레스코핑 / 빌더패턴
테스트앱빌드

이 3가지가 채택됨.

요약-스토리보드



각자 담당한 솔루션에대한
스토리보드 완성후 병합

try/catch >> 예상가능한 예외에 작성된 try/catch 제거
패턴 >> 매개변수가 많다면 빌더패턴을 고려 else, 텔레스코핑패턴
테스트앱 >> 우리의 목적에 부합하는 성능 test app 작성

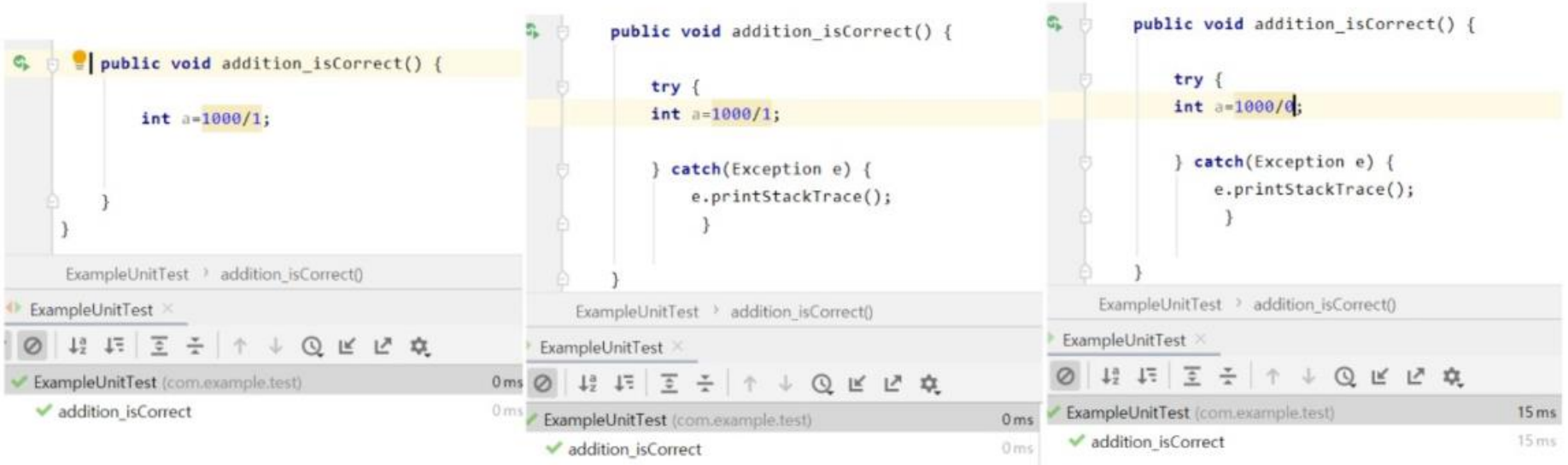
요약-prototype

성능연구 주제를 최대한 사용자 관점에서 접근하기위해,
기술개발의 결과가 어떤 제품에 사용됬을때
사용자의 user experience를 최대한 예측
//
어떤 실질적인 성능특히 사용자경험에서 제일 중요한 속도측면에있어
솔루션들이 조금이라도 지연을 단축시키는지를 확인해
제품에 적용시 사용자경험개선을 확실히 보장함을 확인하는 방향으로진행.

환경 = 안드로이드 api 26 oreo(8 버전)

- try-catch 문 비용
- 텔레스코프 , 빌더 패턴비교
- 테스트앱빌드

요약-prototype



try문에서 예외 발생시 추가연산발생

요약-prototype

인자갯수에따른
텔레스코핑패턴
빌더패턴
의 효율

우선, 텔레스코핑 패턴을 이용한 결과 화면이다.

8개의 인자를 주고, 8개의 인자를
호출하는 함수의 실행시간을 측정하였다
.

실행시간은 506 * 10 ^ -6 초인것을
확인 가능하다.

```
package new1;

public class LayoutParams {

    private final int type;
    private final int flags;
    private final int format;
    private final int w;
    private final int h;
    private final int x;
    private final int y;
    private final int temp;

    public LayoutParams(int type, int flags, int format) {
        this(type, flags, format, 0);
    }

    public LayoutParams(int type, int flags, int format, int w) {
        this(type, flags, format, w, 0);
    }

    public LayoutParams(int type, int flags, int format, int w, int h) {
        this(type, flags, format, w, h, 0);
    }

    public LayoutParams(int type, int flags, int format, int w, int h,int x) {
        this(type, flags, format, w, h, x, 0);
    }

    public LayoutParams(int type, int flags, int format, int w, int h,int x, int y) {
        this(type, flags, format, w, h, x, y, 0);
    }

    public LayoutParams(int type, int flags, int format, int w, int h, int x, int y, int temp) {
        this.type = type;
        this.flags = flags;
        this.format = format;
        this.w = w;
        this.h = h;
        this.x = x;
        this.y = y;
        this.temp = temp;
    }
    public void print() {
        System.out.println(type+" "+flags+" "+format+" "+w+" "+h+" "+x+" "+y+" "+" "+temp);
    }
    public static void main(String[] args) {

        LayoutParams t2 = new LayoutParams(1,3,2,100,200,300, 400, 77);
        long time1 = System.nanoTime();
        t2.print();
        long time2 = System.nanoTime();
        System.out.println((time2- time1)/1000);
    }
```

우선, 빌더 패턴을 활용한 결과 화면이다

똑같이 8개의 인자를 주고, 8개의 인자를
호출하는 함수의 실행시간을 측정하였
다.
실행시간은 223 * 10^-6 초로
텔레스코핑 패턴보다 우수하다.
(물론 여러 조건이 있다. 인수가 많아야
하는 것도 조건 중 일부다)

(왼쪽 코드와 이어서)

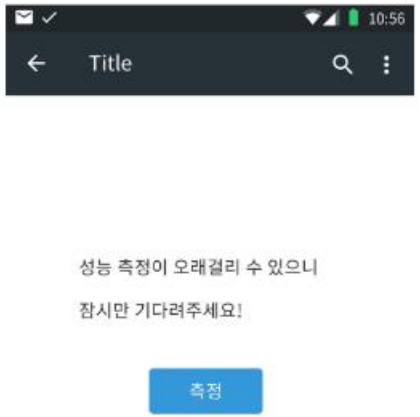
```
    public void print() {
        System.out.println(type+" "+flags+" "+format+" "+w+" "+h+" "+x+" "+y+" "+temp);
    }

    public static void main(String[] args) {

        LayoutParams_2 t2 = new LayoutParams_2.Builder(1, 3, 2).w(100).h(200).x(300).y(400).temp(77).build();
        long time1 = System.nanoTime();
        t2.print();
        long time2 = System.nanoTime();
        System.out.println((time2-time1)/1000);
    }
}
```

텔레스코핑 패턴으로 임의로 구
코드를 빌더 패턴으로 재 구현한
코드이다.

요약-prototype



변경한코드의 성능개선을 측정하기위한
빠르고 가벼운 도구

이번주-survey/pitching

질문목록:

1.개선된 안드로이드에서 마음에 들었던 점이 있다면 어느 점에서 장점을 느끼셨나여?

2.안드로이드에서 신뢰할 수 없는 기능이나 단점이 있다면 어떤 것이며 그 이유는 무엇인가요?

3.개선된 안드로이드에서 추가된 기능이 필요없다 생각하는 부분이 있나요?

4.안드로이드에서 개선의 여지가 있는 부분이 있다면 어떤 것이며 그 이유는 무엇인가요?

5.어느 방면에서 성능이 향상되었다고 생각하시나요?

Survey　//장점

개선된 안드로이드에서 마음에 들었던 점이 있다면 어느 점에서 장점을 느끼셨나여?

빠른 속도입니다.
좀더 빠른로딩속도
좀 더 빠른 속도를 경험할 수 있을 것 같음
속도가 향상됨
코드 가독성 향상

Survey // 불편한점

안드로이드에서 신뢰할 수 없는 기능이나 단점이 있다면 어떤 것이며 그 이유는 무엇인가요?

어플 몇개만 실행하면 가용램이 너무 적다. / 램이 적으면 실행속도가 느려지기 때문이다.
몇몇 안쓰는코드 레거시코드들의 존재나 요즘 하드웨어들의 성능을믿고 통으로짠 코드들이단점으로보임
저사양 기기의 경우 어플리케이션을 몇개만 실행시켜도 버벅 거리는 현상
안드로이드 운영체제 자체가 오픈되어있다는 느낌;
성능 향상 과제라고 설명 들어 단점을 없을 것 같으나 옳은 방향인지 확인이 필요

Survey

개선된 안드로이드에서 추가된 기능이 필요없다 생각하는 부분이 있나요?

없다.
속도관련 개선은 다다익선이라고 생각함 다필요
오류 확인 부분에 있어 관습적인 코드 일 수 도 있을 거라는 것이 생각
없다.
try-catch 문은 확인을 할 필요

Survey //개선의여지

안드로이드에서 개선의 여지가 있는 부분이 있다면 어떤 것이며 그 이유는 무엇인가요?

배터리를 줄일 수 있는 방법이 있을까? / 배터리 문제 또한 중요하기 때문
자바가 코드짜기 편하긴하지만 속도가 느린거같으면 c를 써보는것도 좋아보임
안드로이드 JNI 를 사용하기 때문에 관습적인 코드가 있을 수 밖에 없다고 생각
c 프레임워크 단에서 레거시 코드들이 있을 것으로 생각
텔레스코핑 패턴 말고도 다른 레거시 패턴들이 있을 것 같다.

Survey

어느 방면에서 성능이 향상되었다고 생각하시나요?

실행의 속도
압도적으로 느껴지는 정도는 아니지만 로딩속도향상
속도면에서 향상 시켰다 생각
속도 및 안정화라고 생각한다.
코드 가독성, 앞으로 코드 개발에 있어서 유의미한 점

평가

1 -대체적으로 사용자들은 속도의 향상에 관심을 가지고있음.
보통 사용자환경에서 속도,지연시간이 절대적인 체감요소임을 다시한번확인함.

2. -몇사람들은 속도가아닌 가독성에 집중을함
아마 특수한업종의 특수한환경의 직업을 가진사람들은 속도를 포기하더라도
/다양한기능/유지,보수의용이/연결의안정성/생산중단,개발중단된것들에 대한 지원/
이런것들을 원할수도있지만 일반인들은 속도가 절대적인요소.
하지만 속도에 집중하더라도, 수정한코드가 작동하더라도
생산적,구조적인 틀을 깨면 안된다는걸 확실히 각인함.

3. -속도향상은 크게
-CPU의 불필요한 연산,메소드을 줄이고
-메모리,하드를 차지하는 불필요한 코드들을 제거하는것
-가독성의java vs 속도의c 사이의 적절한 조율
에서 이루어질것같음.

후기

디자인스프린트를 통해 매번 주제에대해 생각하며
주제를 더 깊이, 단계별로, 구조적으로 이해할수있었으며
조원들과의 소통도 점점더 원활해졌다.

앞으로 안드로이드 프레임워크 성능개선을 진행하며
이런 디자인스프린트 기법을 유용하게 사용해야겠다.
물론 더깊은 연구,개발이나 코드분석들을 계속진행해야하지만
디자인스프린트를 통해 그린 청사진을 바탕으로
계속 나아간다면 무리없이 과제를 완수할수있을것같다.

프로젝트 주소

- https://github.com/keelim/project_aosp
- https://www.youtube.com/channel/UC05ce0W79PQWj6H86cN4_LQ