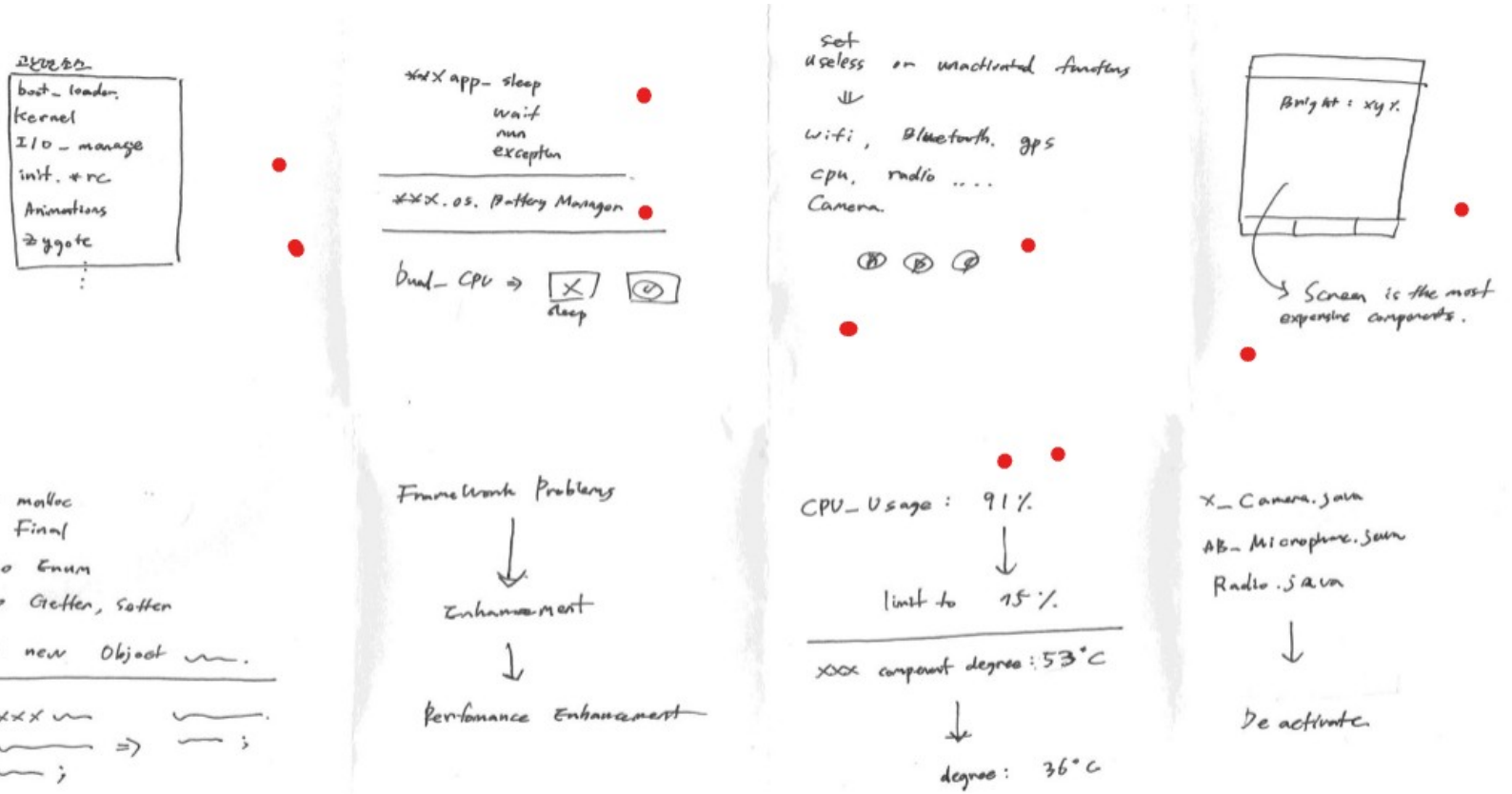
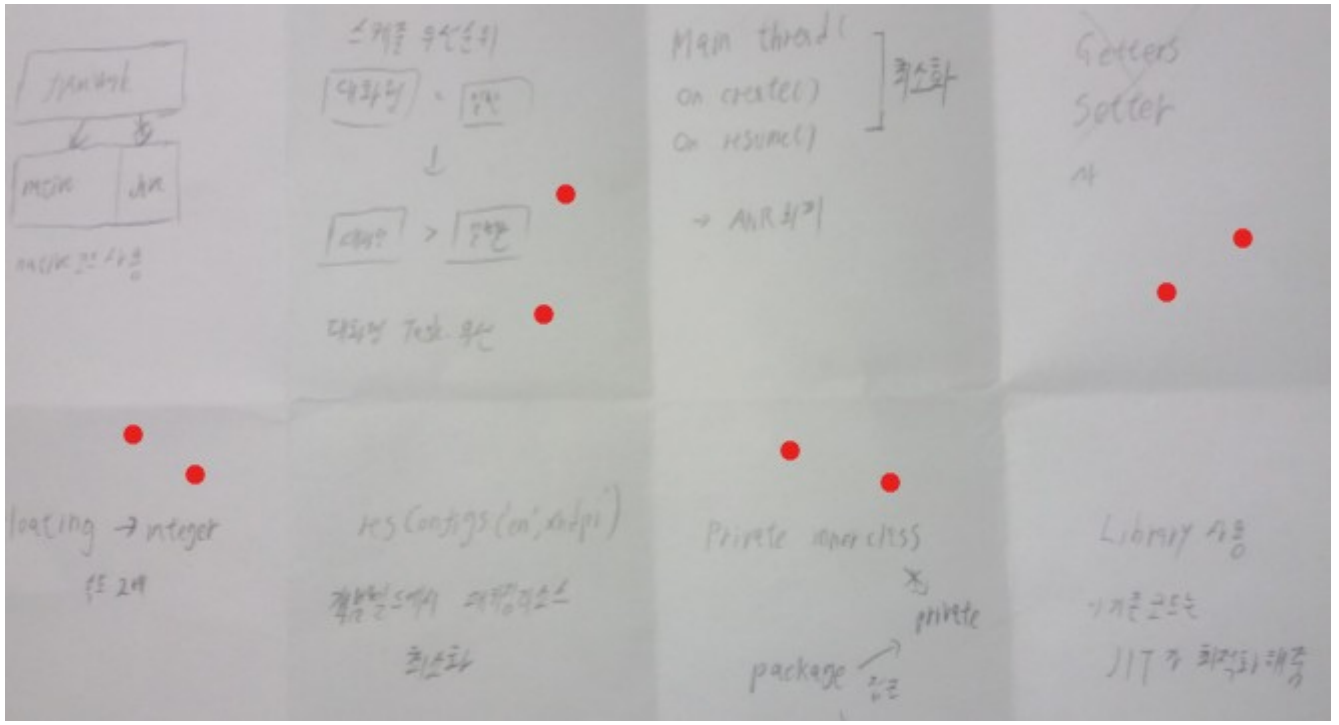
종합설계 졸업과제
solution/prototype

aosp- 임진현

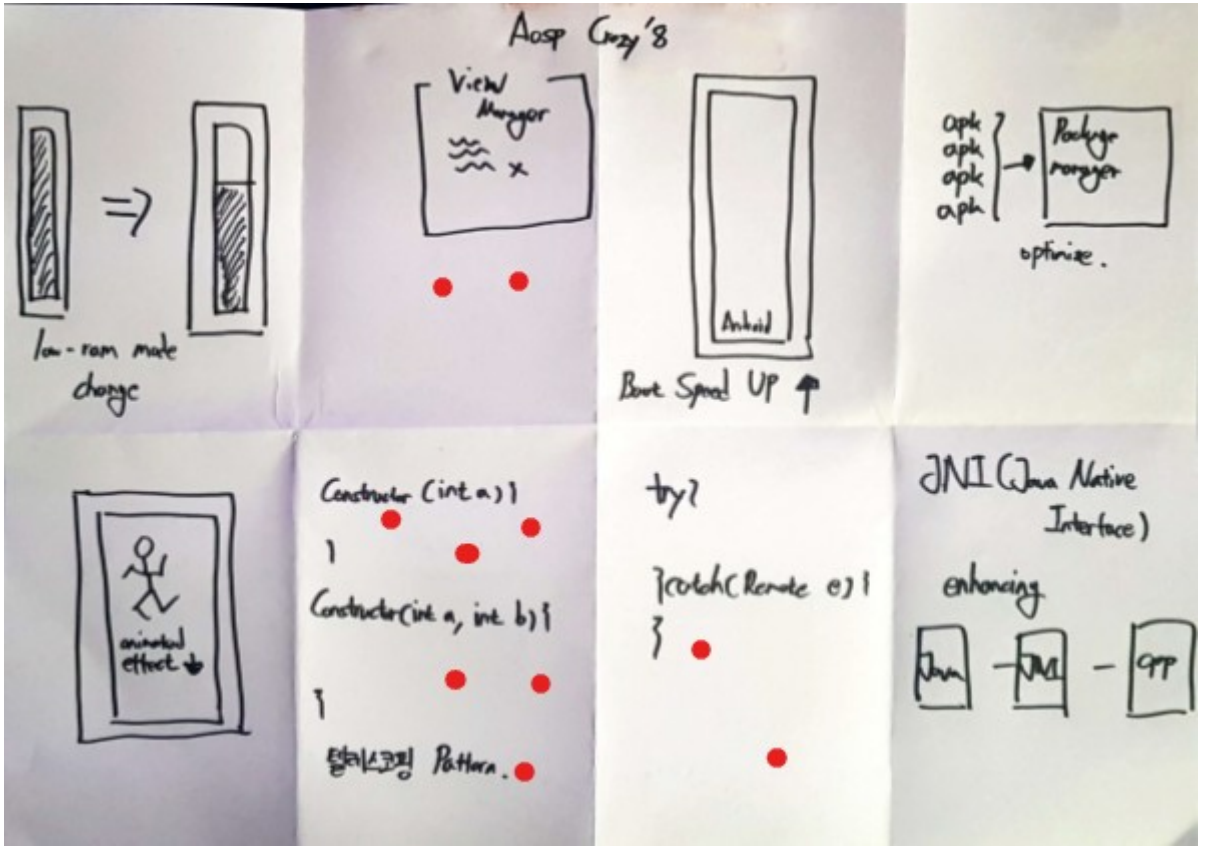
솔루션 투표결과 (구글설문)



솔루션 투표결과


Main thread
on create()
on resume()
최소화
Getters
Setter
floating → integer
Private inner class
private
package
Library

솔루션 투표결과


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

솔루션 투표결과

구글설문지를통한 조내부 설문을통해
try/catch 블럭
텔레스코핑 / 빌더패턴
테스트앱빌드

이 3 가지로 성능향상을 노려보기로함 .

제가 맡은파트는 try/catch 블럭의비용 테스트입니다 .

Stack traces

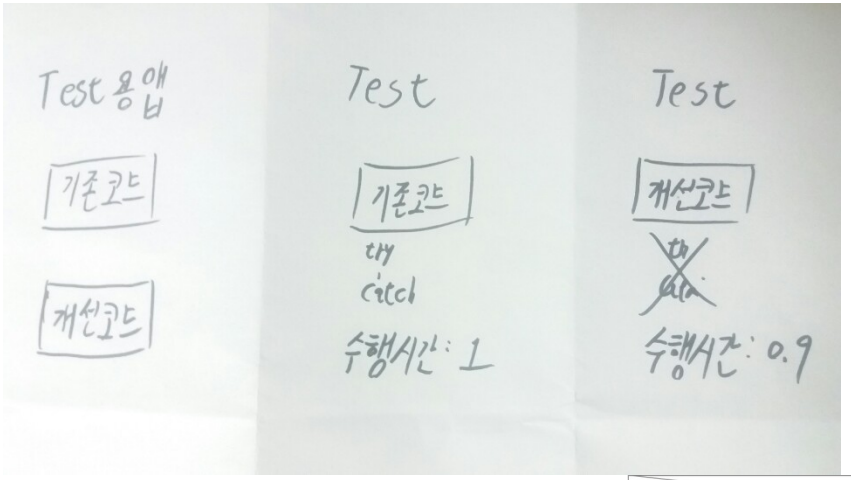
Now that we've seen the cost of calling exceptions rarely, we're going to look at the effect the stack trace depth has on performance. Here are the full, raw results:

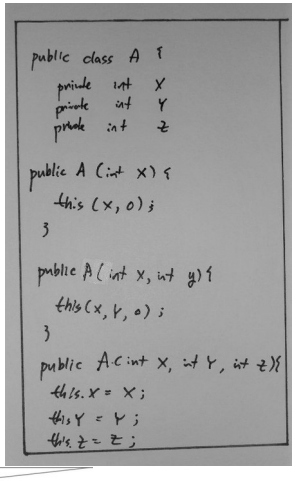
Method	Mean	StdErr	StdDev	Gen 0	Allocated
Exception-Message	9,187.9417 ns	13.4824 ns	48.6117 ns	-	148 B
Exception-TryCatch	9,253.0215 ns	13.2496 ns	51.3154 ns	-	148 B
Exception**Medium**-Message	14,911.7999 ns	20.2448 ns	78.4078 ns	-	916 B
Exception**Medium**-TryCatch	15,158.0940 ns	147.4210 ns	737.1049 ns	-	916 B
Exception**Deep**-Message	19,166.3524 ns	30.0539 ns	116.3984 ns	-	916 B
Exception**Deep**-TryCatch	19,581.6743 ns	208.3895 ns	833.5579 ns	-	916 B
CachedException-StackTrace	29,354.9344 ns	34.8932 ns	135.1407 ns	-	1.82 kB
Exception-StackTrace	30,178.7152 ns	41.0362 ns	158.9327 ns	-	1.93 kB
Exception**Medium**-StackTrace	100,121.7951 ns	129.0631 ns	499.8591 ns	0.1953	15.71 kB
Exception**Deep**-StackTrace	154,569.3454 ns	205.2174 ns	794.8034 ns	3.6133	27.42 kB

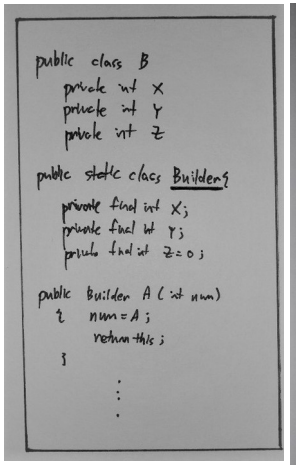
Note: in these tests we are triggering an exception **every-time** a method is called, they aren't the rare cases that we measured previously.

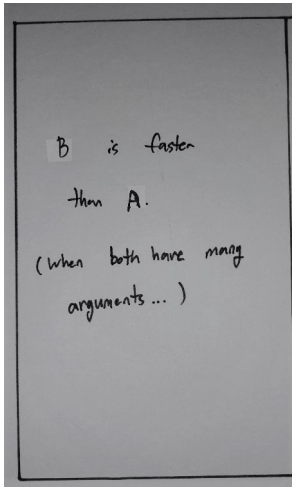
이정도의 비용이 든다고함 .

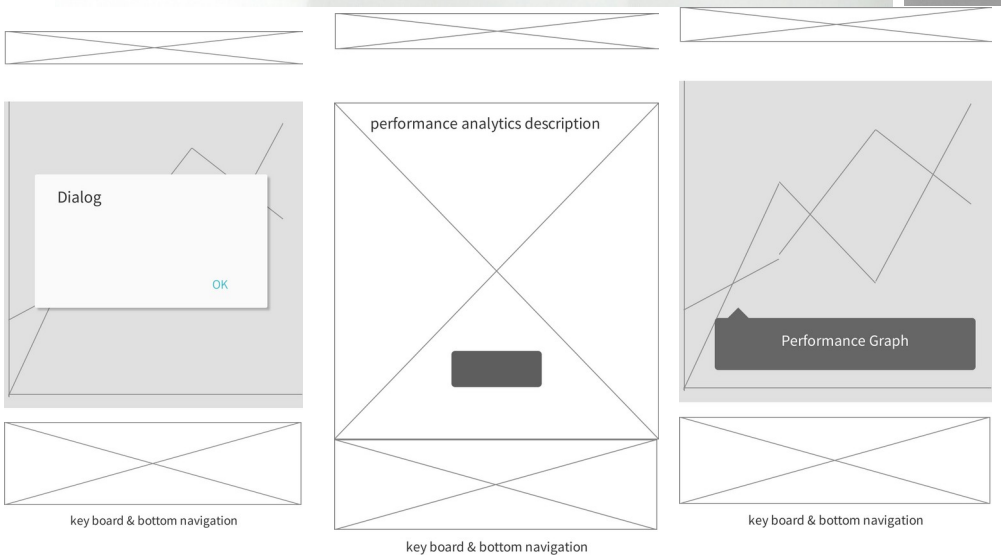
스토리보드











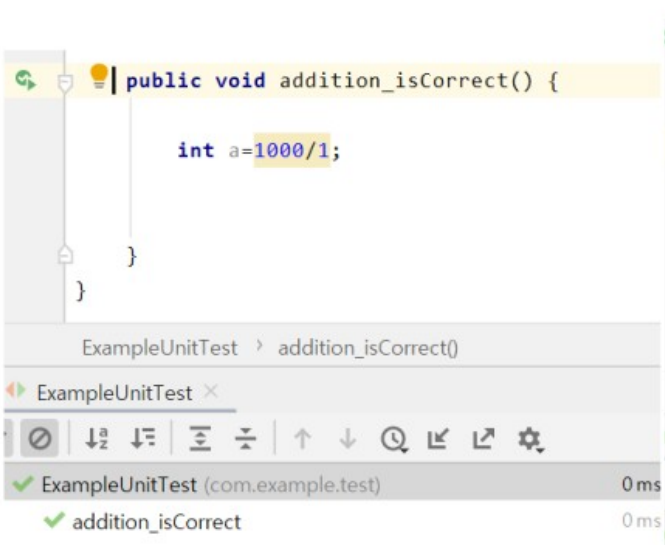
try/catch >> 예상가능한 예외에 작성된 try/catch 제거
패턴 >> 매개변수가 많다면 빌더패턴을 고려 else, 텔레스코핑패턴
테스트앱 >> 우리의 목적에 부합하는 성능 test app 작성

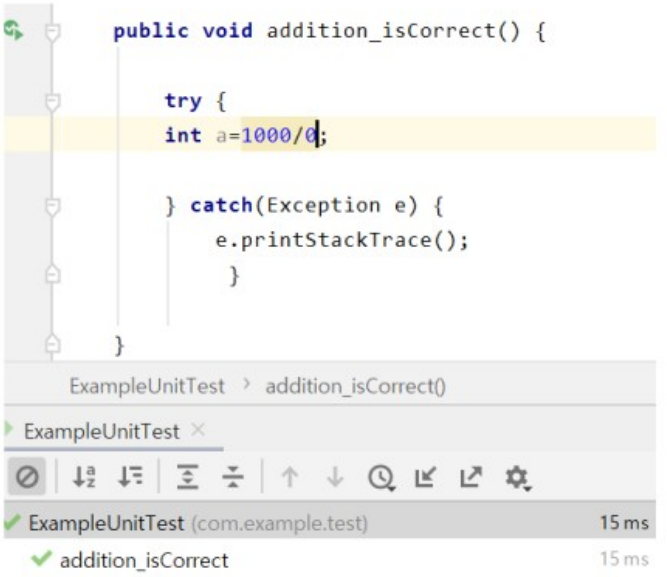
프로토타입

환경 = 안드로이드 api 26 oreo(8 버전)

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

프로토타입 (성능측정)

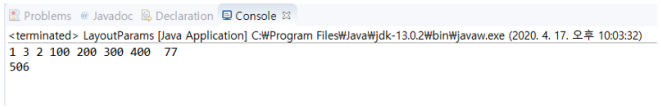




try문에서 예외 발생시 추가연산발생

프로토타입 (성능측정)

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);
    }

}
```

프로토타입 (성능측정)

우선, 빌더 패턴을 활용한 결과 화면이다
.

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

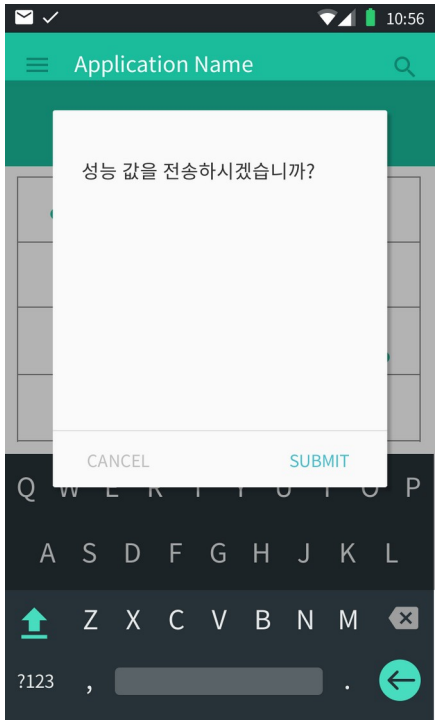
```
package new1;

public class LayoutParams_2 {

    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;

    private LayoutParams_2(Builder builder) {
        type = builder.type;
        flags = builder.flags;
        format = builder.format;
        w = builder.w;
        h = builder.h;
        x = builder.x;
        y = builder.y;
        temp = builder.temp;
    }

    public static class Builder{
        private final int type;     // 필수
        private final int flags;    // 필수
        private final int format;  // 필수
        private int w=0;     // 선택
        private int h=0;
        private int x=0;
        private int y=0;
        private int temp=0;

     public Builder(int type, int flags, int format) { // 필수인자 설정자
         this.type = type;
         this.flags = flags;
         this.format = format;
     }

     public Builder w(int num) {
         w = num;
         return this;
     }

     public Builder h(int num) {
         h = num;
         return this;
     }

     public Builder x(int num) {
         x = num;
         return this;
     }
     public Builder y(int num) {
         y = num;
         return this;
     }
     public Builder temp(int num) {
         temp = num;
         return this;
     }

     public LayoutParams_2 build() {
         return new LayoutParams_2(this);
     }
```

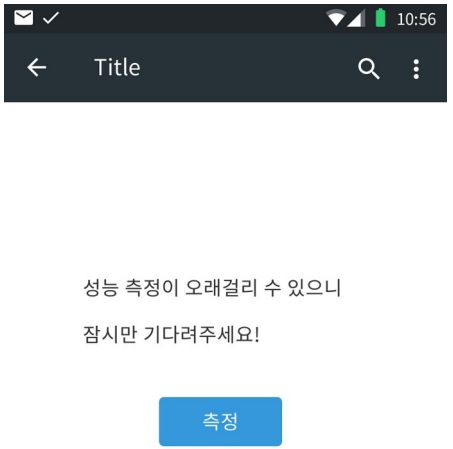
(왼쪽 코드와 이어서)

```
    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.

프로토타입 (테스트앱)


10:56
Application Name
성능 값을 전송하시겠습니까?
CANCEL
SUBMIT


10:56
Title
성능 측정이 오래걸리 수 있으니
잠시만 기다려주세요!
측정


10:56
Application Name

프로젝트 주소

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