

종합설계 2주차

AOSP - 201502088 이송무
(Storyboard + Prototype)

투표 결과

- 구글 설문지를 이용해 투표결과를 결산. (직관적으로 표시한 것)

고전적 OS
boot-loader
kernel
I/O - manage
init. + rc
Animations
@ ygotc

* malloc
Final
no Enum
no Giften, Sotter
no new Object ~.
xxx ~ ~ ~
~ ~ ~ => ~ ~ ~
~ ~ ~ ;

xxx app- sleep
wait
nan
exception

xxx. os. Battery Manager
Dual-CPU => ☒ ☐
deep

Framework Problems
↓
Enhancement
↓
Performance Enhancement

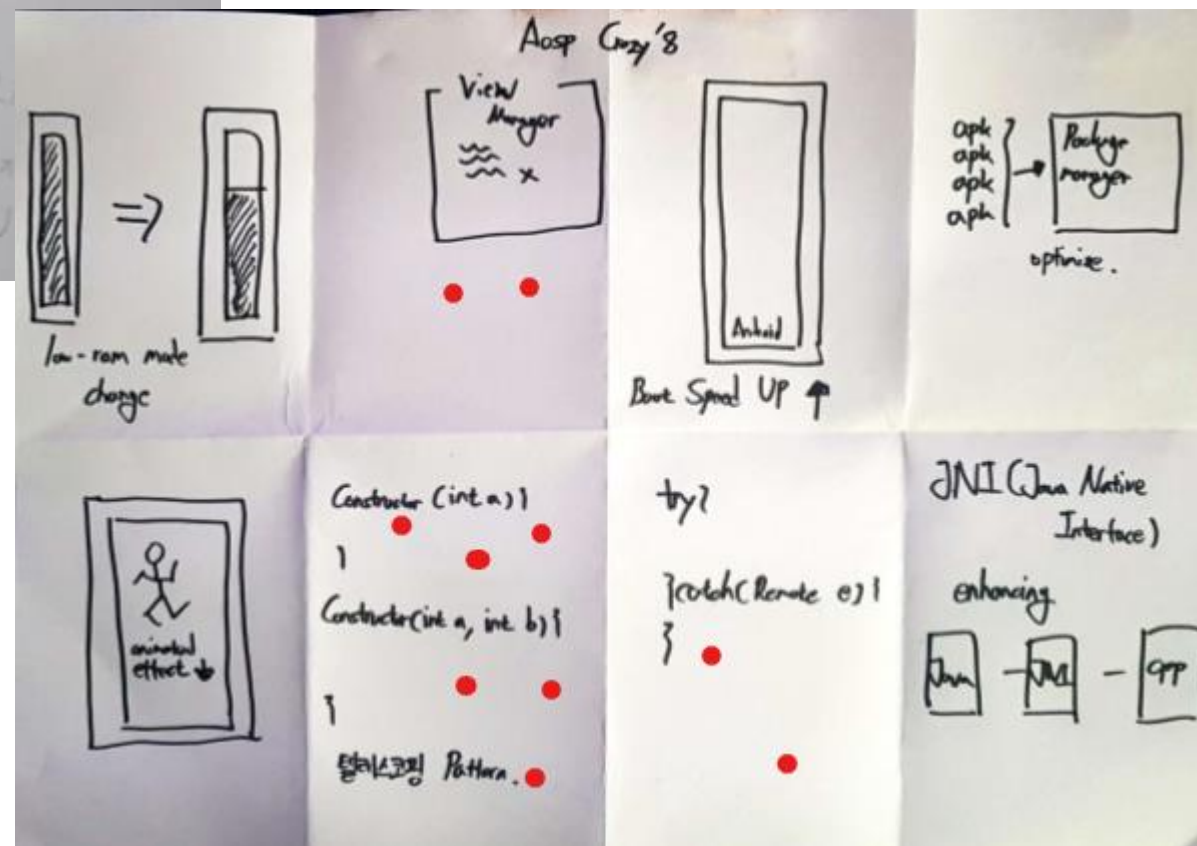
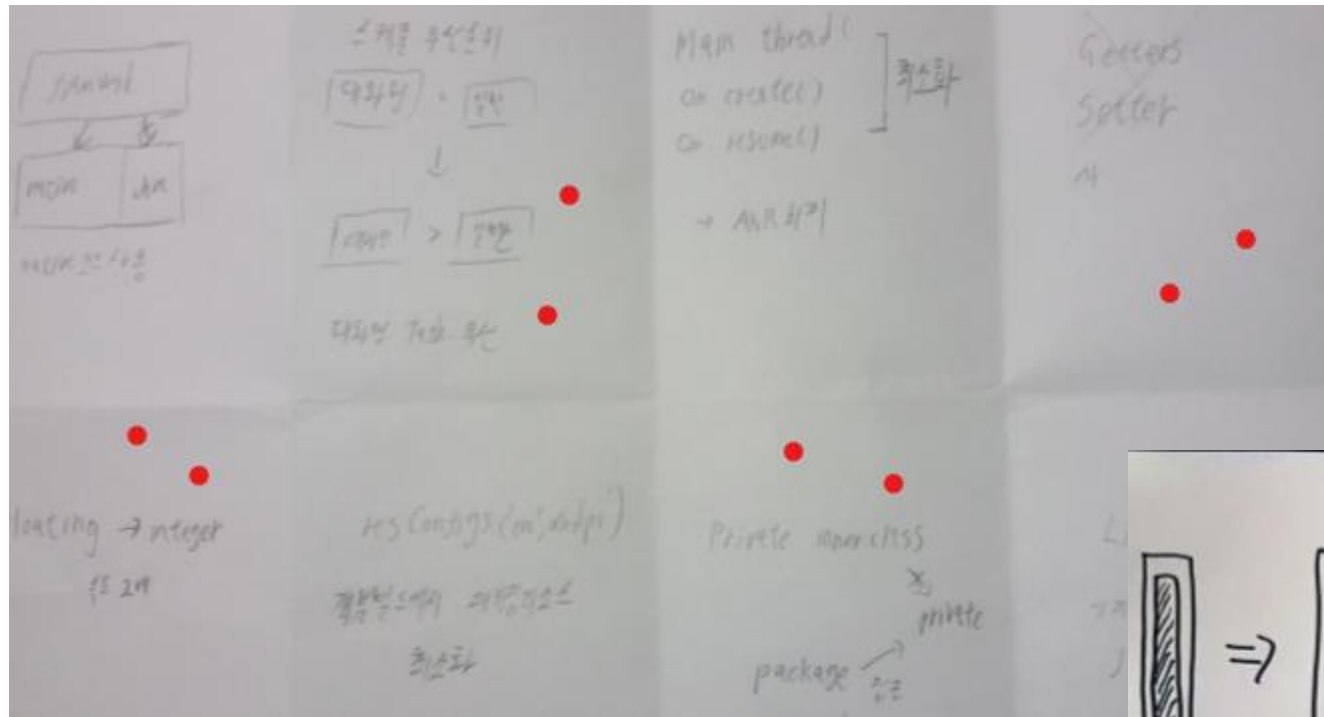
set
useless on unactivated functions
↓
wifi, Bluetooth, gps
cpu, radio ...
Camera
Ⓟ Ⓟ Ⓟ

CPU-Usage : 91%
↓
limit to 15%

xxx component degree : 53°C
↓
degree : 36°C

Brightness : xy %
↓
Screen is the most
expensive components.

X-Camera.join
AB-Microphone.join
Radio.join
↓
De activate.

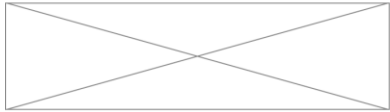


투표 결과

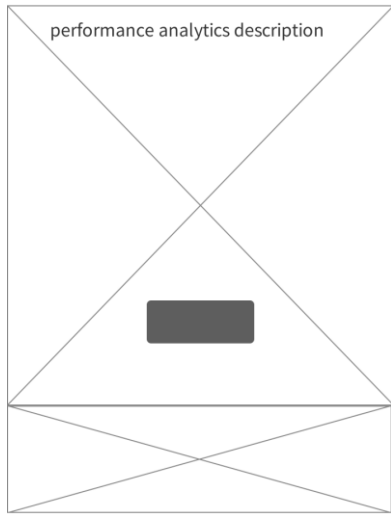
- 텔레스코핑 패턴을 수정하여 성능을 개선하는 솔루션
- Try catch 문을 수정하여 성능을 개선하는 솔루션
- 성능 측정 프로파일 도구를 직접 만드는 솔루션

들이 채택되었다.

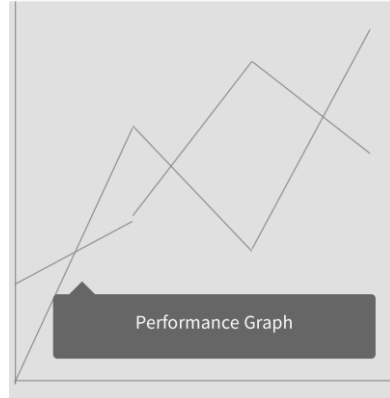
스토리 보드



key board & bottom navigation



key board & bottom navigation



key board & bottom navigation

```
public class A {
    private int x;
    private int y;
    private int z;

    public A (int x) {
        this (x, 0);
    }

    public A (int x, int y) {
        this (x, y, 0);
    }

    public A (int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
```

```
public class B
    private int x;
    private int y;
    private int z;

    public static class Builders {
        private final int x;
        private final int y;
        private final int z = 0;

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

        :
    }
}
```

B is faster
than A.

(when both have many
arguments...)

Test 용 애

기존 코드

개선 코드

Test

기존 코드

타
c/c++

수행 시간: 1

Test

개선 코드

~~타~~
~~c/c++~~

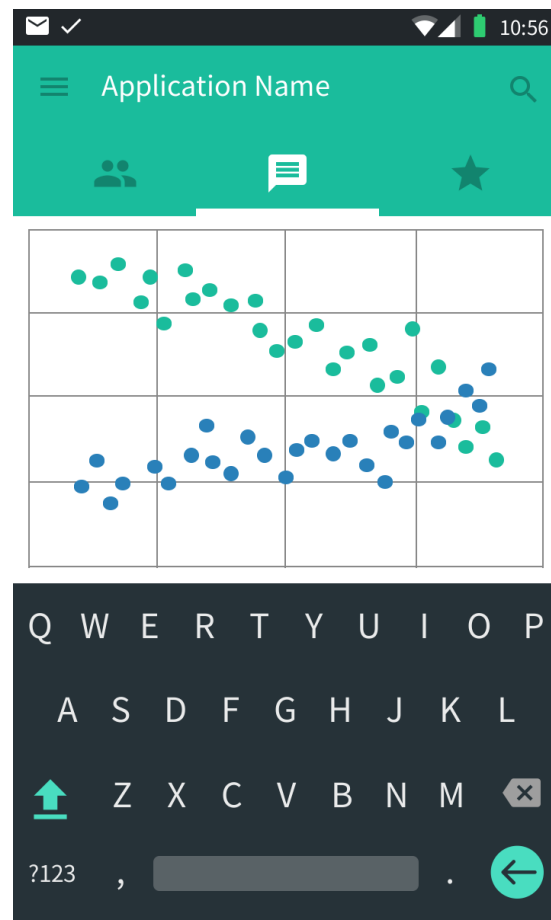
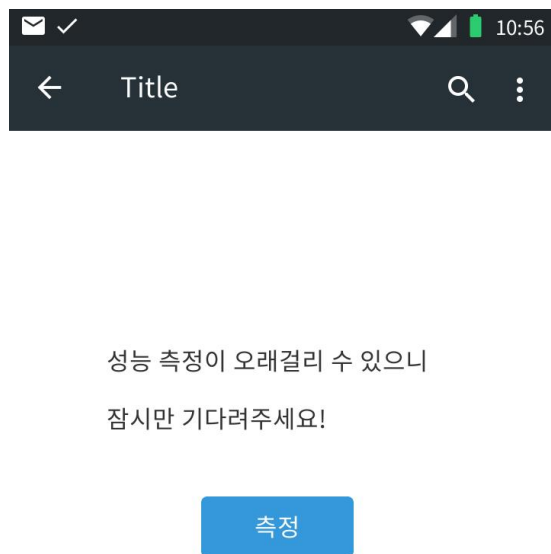
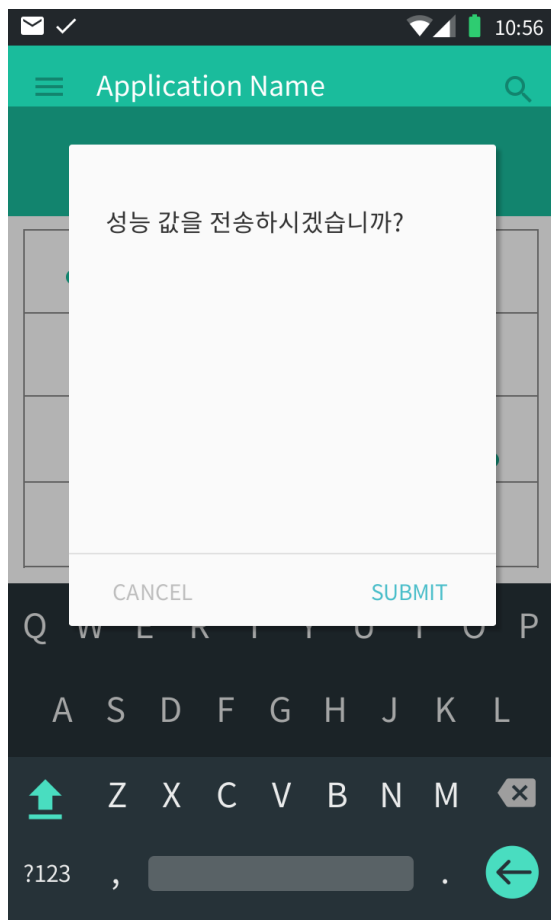
수행 시간: 0.9

프로토타입

- 순서 :

1. 성능측정 프로파일 도구를 만드는 솔루션
2. 텔레스코핑패턴을 수정하여 성능을 개선하는 솔루션
3. Try-catch 문을 개선하여 성능을 개선하는 솔루션

성능측정도구 제작



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

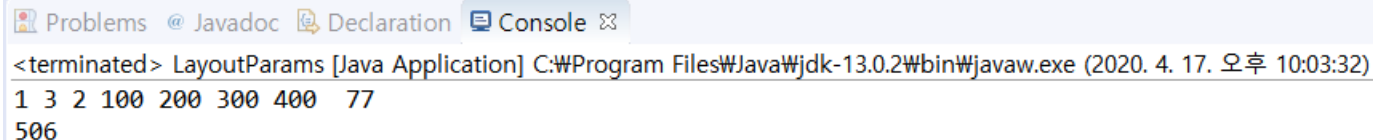
텔레스코핑 패턴을 수정하여 성능 개선 방법

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

```
2630 public LayoutParams(int _type) {
2631     super(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
2632     type = _type;
2633     format = PixelFormat.OPAQUE;
2634 }
2635
2636 public LayoutParams(int _type, int _flags) {
2637     super(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
2638     type = _type;
2639     flags = _flags;
2640     format = PixelFormat.OPAQUE;
2641 }
2642
2643 public LayoutParams(int _type, int _flags, int _format) {
2644     super(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
2645     type = _type;
2646     flags = _flags;
2647     format = _format;
2648 }
2649
2650 public LayoutParams(int w, int h, int _type, int _flags, int _format) {
2651     super(w, h);
2652     type = _type;
2653     flags = _flags;
2654     format = _format;
2655 }
2656
2657 public LayoutParams(int w, int h, int xpos, int ypos, int _type,
2658     int _flags, int _format) {
2659     super(w, h);
2660     x = xpos;
2661     y = ypos;
2662     type = _type;
2663     flags = _flags;
2664     format = _format;
2665 }
2666
```


Telescoping Constructor Pattern

우선, 텔레스코핑 패턴을 이용한 결과 화면이다.



```
<terminated> LayoutParams [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (2020. 4. 17. 오후 10:03:32)
1 3 2 100 200 300 400 77
506
```

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

실행시간은 $506 * 10^{-6}$ 초 인것을 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);
    }
}

```

WindowManager.java 속에 있던
LayoutParams 클래스 파일과 비슷
하게 텔레스코핑 패턴으로 임의로
구현한 코드이다.

Builder Constructor Pattern

우선, 빌더 패턴을 활용한 결과 화면이다.

Problems @ Javadoc Declaration Console

<terminated> LayoutParams_2 [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (2020. 4. 17. 오후 10:12:27)

```
1 3 2 100 200 300 400 77  
223
```

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 생성자
            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 구현했던 코드를 빌더 패턴으로 재 구현한 코드이다.

Try-Catch 문 수정하여 성능 개선

The image displays three sequential screenshots of an IDE, illustrating the modification of a try-catch block to improve performance.

Left Screenshot: Shows the initial code snippet:

```
public void addition_isCorrect() {  
    int a=1000/1;  
}
```

The test runner shows the test `addition_isCorrect` passed in 0 ms.

Middle Screenshot: Shows the code snippet with a try-catch block added:

```
public void addition_isCorrect() {  
    try {  
        int a=1000/1;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

The test runner shows the test `addition_isCorrect` passed in 0 ms.

Right Screenshot: Shows the code snippet with the try-catch block modified to handle a division by zero error:

```
public void addition_isCorrect() {  
    try {  
        int a=1000/0;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

The test runner shows the test `addition_isCorrect` passed in 15 ms.

try문에서 예외 발생시 추가연산발생

링크들

- 깃허브 : https://github.com/keelim/project_aosp
- 유튜브 : https://www.youtube.com/channel/UC05ce0W79PQWj6H86cN4_LQ
- 이송무 조원이 분담한 스토리보드, 프로토타입 : 텔레스코핑 패턴 개선