Check It Out - Digital Watch

2019 Software Modeling & Analysis
대응보고서

T6
201711381 김소현
201711401 염혜지
201711420 임수연
201711428 조은지

Contents

1. System Test Response

- Category Partition Testing Result

Timer

Test12 - 1013.2031.3034	Fail	Timer 작동 안 됨.
Test13 - 1013.2032.3033	Fail	Timer 작동 안 됨.
Test14 - 1013.2032.3035	Fail	Timer 작동 안 됨.
Test15 - 1013.2033.3043	Fail	Timer 작동 안 됨.
Test16 - 1013.2033.3032	Fail	Timer 작동 안 됨.
Test17 - 1013.2033.3034	Fail	Timer 작동 안 됨.
Test18 - 1013.2034.3033	Fail	Timer 작동 안 됨.
Test19 - 1013.2034.3035	Fail	Timer 작동 안 됨.
Test20 - 1013.2035.3131	Fail	Timer 작동 안 됨.
Test21 - 1014.2041.3041	Fail	Timer 작동 안 됨.

문제	Timer 실행 불가
대응	확인 결과 초기화면에서의 타이머 실행에는 문제가 없었다. 그러나 타이머 재시작 시 카운트다운이 화면에 보이지 않는 등의 문제로 인해 타이머의 사용이 일회성이 되버리는 문제를 발견하였다. 따라서 타이머의 countdown메소드에서 그 원인을 발견하고 문제를 해결하였다.

Buzzer

Test29 - 1015.2054.3051	Fail	초기 상태에서는 buzzer가 울리나, 한 번 알람을 끄거나 off 후 on을 한 뒤 buzzer가 울리지 않음.

문제	알람을 OnOff한 후에는 버저가 울리지 않음
대응	초기 상태에서 버저가 울린 뒤, 한 번 알람을 끄거나 off 한 후 다시 on 했을 때에도, buzzer가 정상 작동하도록 코드를 수정함.

- Brute Force Testing Result

Test	#	Description	P/F
Action	2-1	Timer/Stopwatch에서 count 해놓고 다른 모드 변경 후 다시 갔을 때 Timer의 동작 하는지 검사	Fail
	2-2	Calculate_calories에서 23:59에서 24:00 넘어간 후 운동을 끝냈을 때 해당 날짜로 저장을 하는지 검사	Pass
	2-3	D+day에서 년도의 임계치를 설정했는지 검사	Pass
	2-4	Fitness에서 Default 값으로 했을 때 작동하지 않는지 검사	Pass
Check	3-1	Timer와 Alarm(주기 설정)의 부저 울리는 시간을 맞춘 후 어떻게 동작하는지 Test	Fail
	3-2	D+day에서 하나 설정 해놓고 setTimeKeeping을 통해 시간 바꿨을 때 D+day 계산이 정상적으로 바뀌는지 검사	Fail
	3-3	Fitness에서 1분 단위 업데이트 직전에 운동 종목을 바꿨을 때 Calories 계산을 제대로 하는지 검사	Fail

문제	Timer/Stopwatch에서 count를 한 후 다른 모드 변경 후 다시 Timer기능으로 돌아왔을 때 Timer의 동작이 되지 않음
원인	Timer 재시작 시 기능 실행 불가
대응	timer의 countdown메소드가 run된 후부터 쓰레드가 종료되기 전까지 loop를 break하지 못하도록 코드를 수정하였다.

문제	Timer의 버저가 울리지 않음
원인	Timer 재시작 시 기능 실행 불가
대응	타이머 재시작 시 버저가 울리지 않는 문제가 있었으나 countdown메소드 문제를 해결하고 난 후 버저가 정상작동하였다.

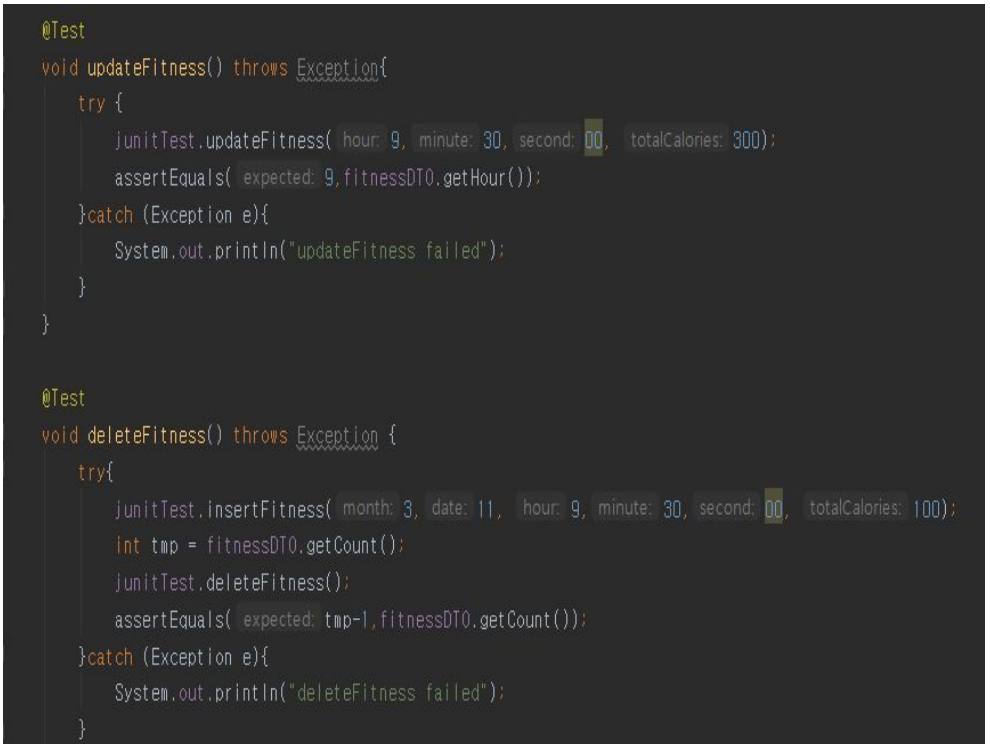
문제	D+Day기능에서 목록을 하나 추가 한 뒤, setTimekeeping을 통해 D+Day보다 이전 날짜로 바꿨을 때 음수 값이 나온다.
대응	D+Day에서 일정을 설정해놓고, set Timekeeping을 통해 시간을 바꾸었을때, D+Day 값을 정상 계산한다. 만약 set Timekeeping을 통해서 현재 날짜를 D+Day 설정 날짜보다 이전으로 변경한 경우(D+Day 설정 날짜가 미래날짜가 된 경우), D-Day 값을 계산하도록 한다.

문제	1분 단위로 운동칼로리양이 update되는 Fitness기능에서 30초, 30초로 총 2번 운동했을 때 총 칼로리는 0으로 바뀌지 않는다.
대응	수정하지 않음 시나리오 상 단순히 운동기록만을 저장하기 때문에 총 운동시간이 갱신이 되었더라도, 칼로리량을 수정해주지 않음

2. Static Analysis Response

- Test Case Refactoring

assert문이 없는 test method에 assert문을 추가하여 code coverage 율을 높였다.

문제	code coverage 퍼센트가 매우 낮다.
대응	setter 또는 void형 메소드의 test method에서 assert문이 없는 것을 확인, 모든 method에 assert문을 추가해주었다. GUI관련 test는 따로 시행하지 않음
	

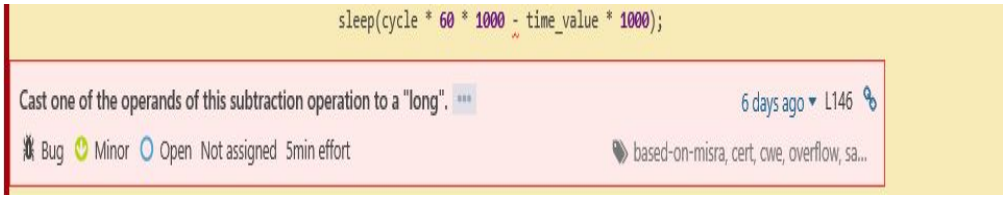
```
@Test
void updateFitness() throws Exception{
    try {
        junitTest.updateFitness( hour: 9, minute: 30, second: 00,  totalCalories: 300);
        assertEquals( expected: 9,fitnessDTO.getHour());
    }catch (Exception e){
        System.out.println("updateFitness failed");
    }
}

@Test
void deleteFitness() throws Exception {
    try{
        junitTest.insertFitness( month: 3, date: 11,  hour: 9, minute: 30, second: 00,  totalCalories: 100);
        int tmp = fitnessDTO.getCount();
        junitTest.deleteFitness();
        assertEquals( expected: tmp-1,fitnessDTO.getCount());
    }catch (Exception e){
        System.out.println("deleteFitness failed");
    }
```

- 주요 Bug

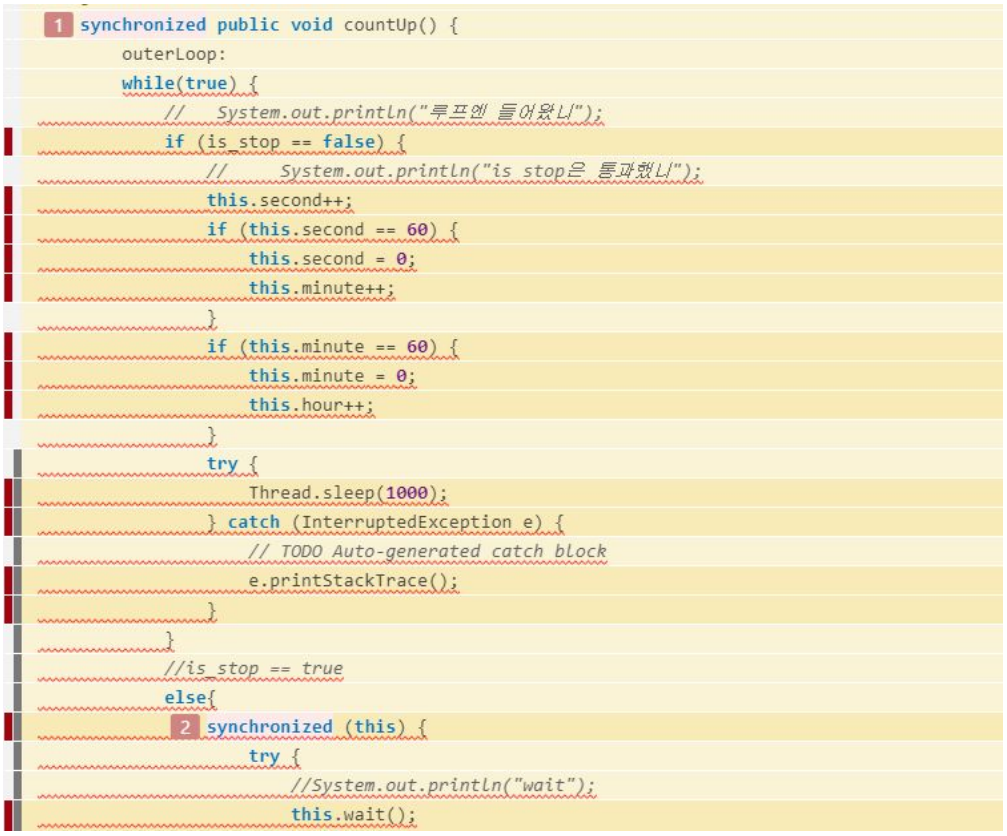
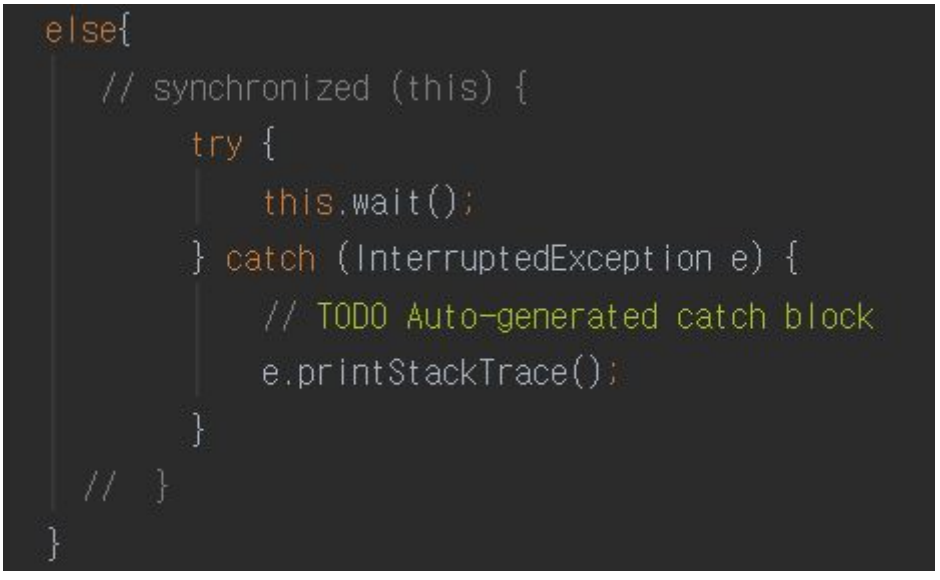
공통적인 Bug

thread(-wait,notify의 사용), while문의 사용에 대한 bug가 많이 발생하였다.

문제	사용하지 않는 field의 존재
	
대응	사용하지 않은 field 모두 삭제 , class Diagram에도 반영하였다.

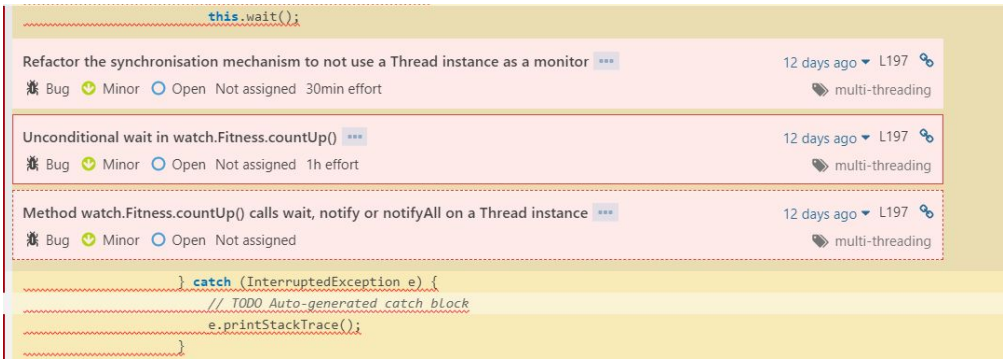
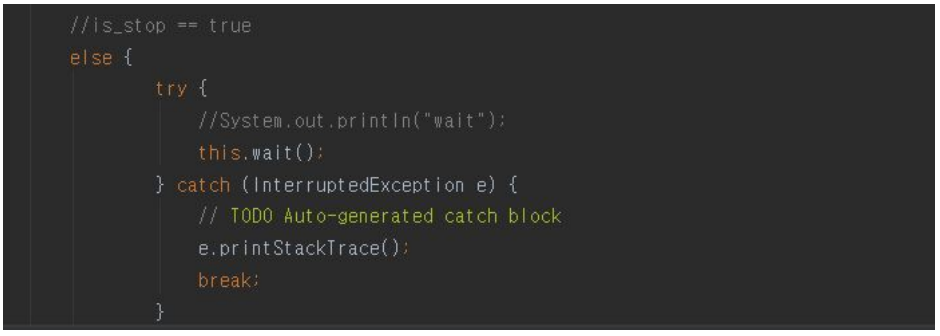
문제	올바르지 않은 형이 operand에 사용되었다.
	
대응	알맞은 형으로 casting 해주었다.

문제	의도된 무한루프라 하더라도 while문에 종료 조건을 넣어야 한다.
	
대응	catch블락에 인터럽트가 생기면 while문을 빠져나가도록 추가해주었다.
	

문제	multiple lock이 발생하지 않도록 해야한다.
	
대응	synchronized public void countUp()내부의 synchronized(this){ } 블록을 지워주었다.
	

```
synchronized public void countUp() {
        outerLoop:
        while(true) {
            //   System.out.println("루프엔 들어왔니");
            if (is_stop == false) {
                //     System.out.println("is_stop은 통과했니");
                this.second++;
                if (this.second == 60) {
                    this.second = 0;
                    this.minute++;
                }
                if (this.minute == 60) {
                    this.minute = 0;
                    this.hour++;
                }
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            //is_stop == true
            else{
                synchronized (this) {
                    try {
                        //System.out.println("wait");
                        this.wait();
```

```
else{
    // synchronized (this) {
        try {
            this.wait();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
//  }
}
```

문제	의도치 않은 wait()이 호출 될 수 있다.
	
대응	is_stop, is_pause라는 condition variable을 통하여 wait()을 호출하도록 했기 때문에 문제가 발생하지 않을 것이다. 각 기능마다 하나의 스레드만 실행이 되기 때문에 또 다른 스레드가 깨어나는 문제는 발생하지 않을 것이다.
	

DBManager

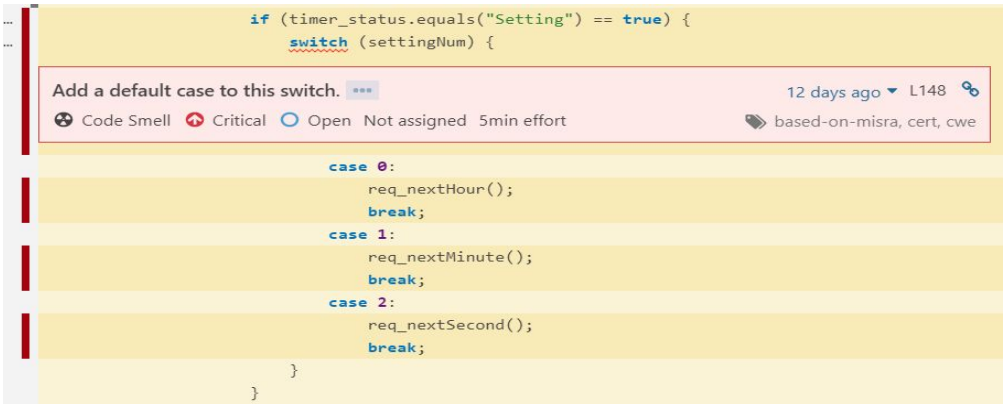
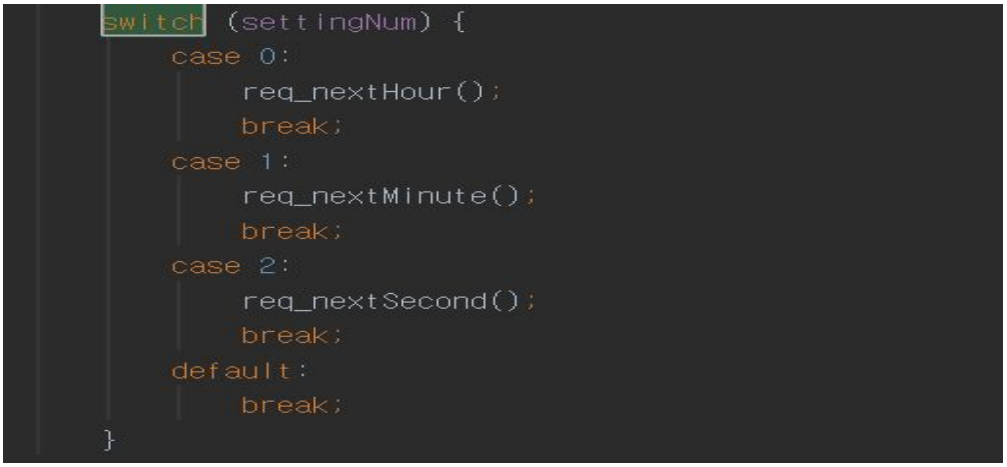
문제	statement를 수행한 후 close를 해주어야 한다.
	
대응	finally block을 추가하여 statement.close()를 해줌
	

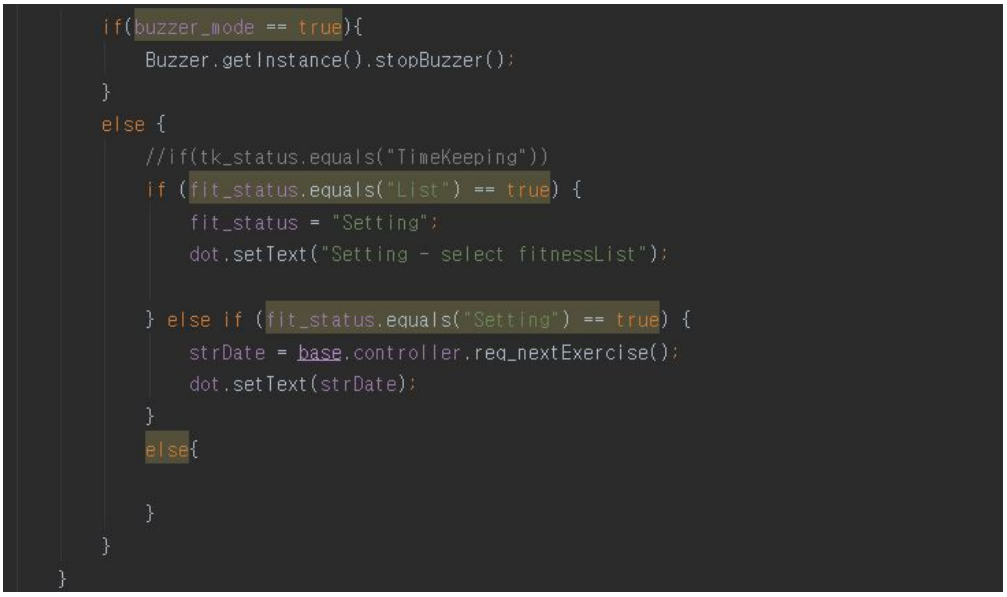
- **CodeSmell (Critical, Blocked)**

Critical

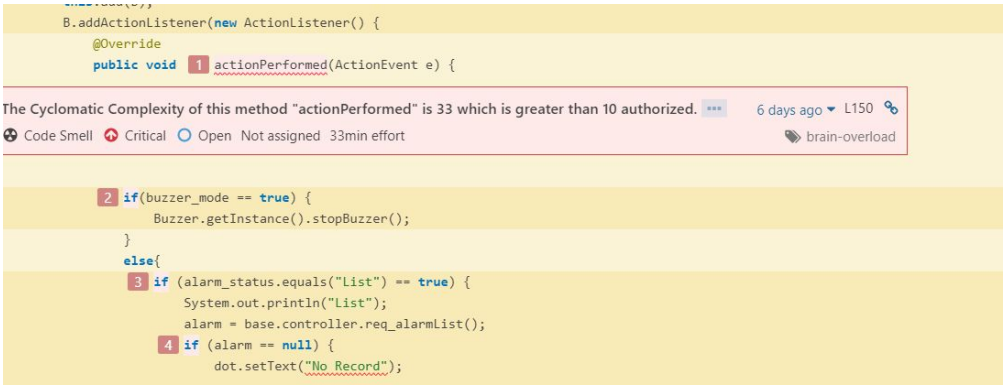
조건문과 try/catch문이 중첩되어있는 복잡한 구조에 의한 code smell문제가 많이 발생하였다.

문제	클래스 내에서 한 변수에 같은 String을 중복해서 사용함 만약 status설정에서 setting을 다른 단어로 변경해야할 시 수정에 번거로운 문제가 발생한다.
	
대응	추가적인 final변수를 선언할 필요는 없다고 생각하여 수정하지 않음

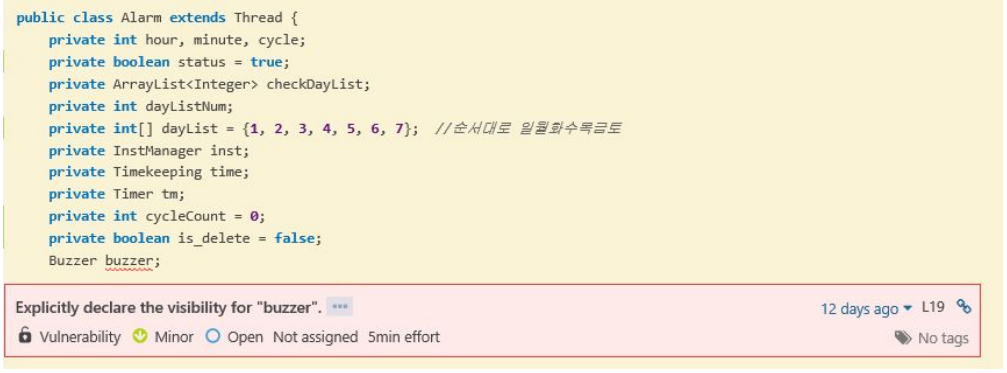
문제	switch문에 default case가 없다.
	
대응	default case를 추가해주었다.
	

문제	if/else 문에 else 가 존재하지 않음
	
대응	대부분의 조건문에 else문이 빠져있었다. 모든 if문에 대해 else문을 추가해주었다.
	

View 공통

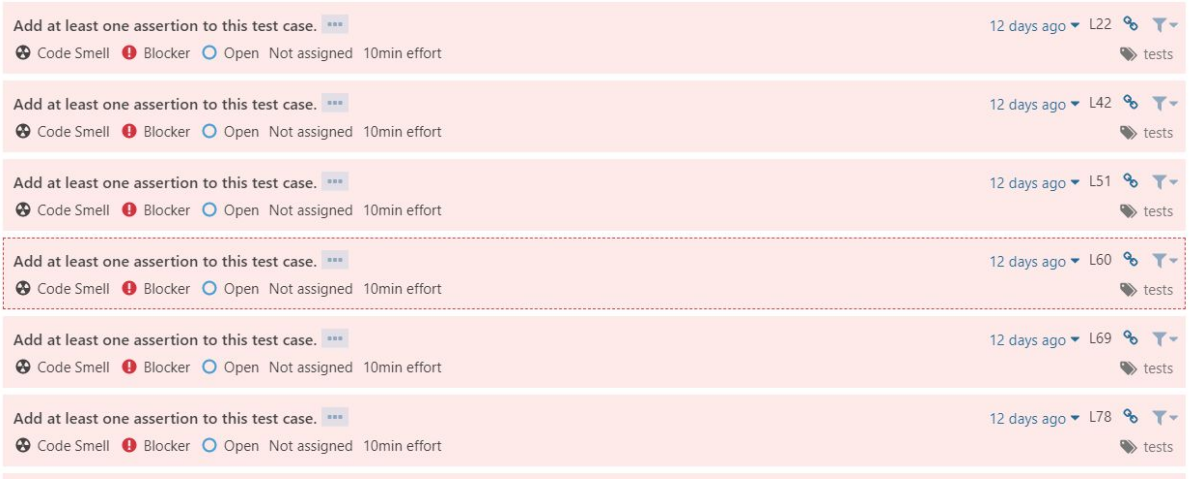
문제	메소드/생성자의 내부 구조가 복잡하다.
	
대응	ActionListner메소드에서 많은 중첩된 if문과 try/catch, switch문을 사용하였다. 구조적으로 복잡하지만 인터랙션 다이어그램에 최대한 맞춰 구현을 하였고, 따로 메소드를 만들어 사용할 필요는 없다고 생각하여 수정을 하지 않았다. 제한된 4개의 버튼에서 여러 상태에 대한 역할을 다르게 구현을 해줘야 하는 상황이였기 때문에 많은 if문의 사용은 불가피했다.

Minor Code Smell문제

문제	field를 불필요하게 public으로 선언했다.
	
대응	field를 private으로 선언하고, 다른 클래스에서 해당 field에 접근할 수 있도록 getter, setter를 추가해주었다.

Blocked

Test 클래스의 일부 메소드에서 assert를 사용하지 않아서 생긴 문제가 많이 발생했다. 앞에서 언급한 code coverage와도 관련된 문제였음.



문제	일부 test method에 assert문이 존재하지 않음
대응	109개의 test method에 대해 모두 assert문을 추가해주었다.

```
@Test
void updateFitness() throws Exception{
    try {
        junitTest.updateFitness( hour: 9, minute: 30, second: 00,  totalCalories: 300);
        assertEquals( expected: 9,fitnessDTO.getHour());
    }catch (Exception e){
        System.out.println("updateFitness failed");
    }
}

@Test
void deleteFitness() throws Exception {
    try{
        junitTest.insertFitness( month: 3, date: 11,  hour: 9, minute: 30, second: 00,  totalCalories: 100);
        int tmp = fitnessDTO.getCount();
        junitTest.deleteFitness();
        assertEquals( expected: tmp-1,fitnessDTO.getCount());
    }catch (Exception e){
        System.out.println("deleteFitness failed");
    }
```

3. OOPT Review

- 지난 3개월 동안 설계, 구현, 통합테스트, 검증 단계를 거치며 소프트웨어 공학에서 중요한 것은 구현만이 아니라는 것을 알게 되었으며 팀원 간의 커뮤니케이션의 중요성에 대해 여실히 느꼈습니다.

- OOAD 방식으로 Digital Watch를 완성하면서, 객체지향으로 Analysis하고 Design하는 것이 무엇인지 경험할 수 있어서 정말 좋았습니다.

- 팀원간의 의사소통의 중요성에 대해 알게되었습니다. 요구사항분석을 통하여 프로젝트를 진행할 때 회의를 할 때 혼동이 적었고 정해진 시나리오대로 구현을 할 수 있었습니다. OOPT단계에서 interaction다이어그램을 통해 클래스들 간의 상호작용에 대해 고민해볼 수 있는 좋은 시간이였습니다. 또한 static analysis를 통하여 코드의 안정성에 대해 검토해 볼 수 있었고, 검토를 하면서 중첩된 if문의 사용과 같이 복잡한 메소드를 구현하게 되어 매우 아쉬웠습니다. OOAD를 통해 구현의 어려움을 많이 줄였지만 개인적으로 구현능력이 부족함을 뼈저리게 느꼈습니다... 이번 수업을 통해서 이상적인 객체지향 프로그래밍에 대해 고민을 하게 된 것 같습니다.

- 구현 이외의 많은 것들을 배우게 되어 정말 유익했고 스스로 부족함을 많이 느낀 시간이었습니다. 초반부터 의사소통을 위해 trello를 사용했는데 편안함을 느껴 효율적인 협업을 위한 도구도 알게 되었고, 소검팀과 slack과 trello를 통해 소통하며 초반에는 툴을 사용하는 데에 서툴렀으나 static anlaysis를 하면서 소통을 많이 하게 된 것 같습니다. 큰 도움을 받아 감사하고 기뻤습니다.