Unit Testing Report for
Fluxvator: an Elevator Control Simulator

May 13th 2014

Team 3

Team Organization

200913215 기세파

201013275 강태호

201013760 이인구

Context

2

1. Disclaimer & Changes From 2040

본래 implementation시 2040단계에서 도출된 class diagram을 기반으로 구현을 하는 것이 정설이며, 구현당시 이대로 진행하기 위하여 노력하였다. 이는 클래스 다이어그램이 유스케이스와 Sequence Diagram을 기반으로 제작되었기 때문인데, 실제 구현 당시에 분명 유스케이스와 Sequence Diagram의 flow를 그대로 따랐음에도 불구하고 기술적인 이유(e.g. Threading, GUI와의 연동)혹은 구현하고자 한 일부 구조(ex : list나 iterator)가 java 기본 제공 라이브러리에서 이미 제공되어 구현의 편의를 위해 이들을 대신 사용함 등으로 인한 변경점이 있었다. 이처럼 유스케이스나 시퀀스 다이어그램의 **큰 그림에 변화를 주지 않는 수준의 클래스 내 구조 변화**를 적용하고 이전 레포트 클래스 다이어그램에서 메소드들의 return type을 적어놓지 않은 것을 수정하여 클래스 다이어그램을 다시 작성하였다. GUI 관련 객체는 애초에 클래스 다이어그램에 넣지 않기로 하여 변경된 클래스 다이어그램에서도 GUI에 구현에 관련된 클래스나 메소드들은 포함되지 않았다.

2. Unit Testing Environment

Unit Testing 시행은 May 12~13 2014에 진행하였다.
JUnit 4를 사용하였고, 클래스 내의 가장 기본적인 메소드 (getter, setter) 등에 대한 Unit Test는 제외하였다.

3. Unit Testing Specification

Name of Test, Procedure, Projected Result, Actual Result(Pass/Non-pass)

Test Case Name	**1. testEnQueue**
Objective(Procedure)	-QueueAlgorithm의 enqueue기능을 시행하였을때 실제로 요청이 저장되는지 확인 -Queue 내의 searchForNode 기능 작동 확인 -Queue 내의 findClosestNodeFromPosition 기능 작동 확인 1) queueID 1, requestID 1, level 1, load 10 을 넣어 enQueue 2) 앞에 넣은 큐의 selectForNode에 같은 enqueue에 넣어준 값을 넣어 그중 load 값이 처음에 넣은 10과 같은지 확인 3) findClosestNodeFromPosition에서 엘리베이터의 현재 위치를 enqueue에 넣은 요청에서의 목적지와 동일하다고 가정하게 parameter를 주고 역시 리턴된 요청의 load 값 비교
Estimated Result	2) 의 결과 10, 3)의 결과 10으로 assertEquals 통과
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	2. testMakeNode
Objective(Procedure)	-Queue에서의 MakeNode 작동 확인 -makeNode기능을 세번 호출하여 총 세개의 요청을 Queue에 넣은 뒤 Queue의 size가 3과 동일한지 확인 -앞에서 넣은 요청 중 level 5, requestID 3으로 넣은 요청에 대해 searchForNode를 호출하여 결과 일치하는지 확인
Estimated Result	Queue size 3, 검색한 결과값의 level 5, requestID 3
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	3. testSearchNode
Objective(Procedure)	-앞의 테스트에서 이용되긴 하였으나 searchForNode 메소드 정상 작동 확인 -먼저 존재하지 않는 Node인 level 5, reuqestID 4인 Node에 대한 검색 시행, null과 같은지 비교 -다음으로 존재하는 Node인 level 5, requestId 3 인 Node에 대한 검색 시행, 결과노드의 requestId 3 값 맞는지 확인
Estimated Result	assertNotEquals 통과, requestID 비교값 3
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	4. testfindClosestNodeFromPos
Objective(Procedure)	-Queue에서 현재엘레베이터의 위치와 방향을 넣었을 때 가장 가까운 요청 검색하는 기능 -엘리베이터가 3층에서 올라간다고 가정하고 5층으로 내리는 요청과 4층에서 올라가는 요청 정의 -가장 가까운 findClosesetNodeFromPos 호출하고 가장 가까운 요청에 대한 목적지 비교
Estimated Result	가장 가까운 목적지 층 결과값 4
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	5. testOpenNClose
Objective(Procedure)	-Elevator 클래스 내 문을 여는 메소드 OpenDoor와 CloseDoor 두 메소드 작동 확인 -문을 열고 문의 현재상태 비교 -문을 닫고 문의 현재상태 비교

Estimated Result	문을 열었을 때 문의상태 1, 문을 닫았을 때 0
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	**6. testClearResult**
Objective(Procedure)	-Queue 클래스 내에 Queue 내용을 모두 비우는 메소드 ClearNode 메소드 작동 확인 -ClearNode 호출 후 Queue가 비었는지 확인
Estimated Result	isEmpty() true 리턴
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	**7. testCompClosestNode**
Objective(Procedure)	-엘리베이터의 현재위치에 대해 가장 가까운 요청 검색 되는지 확인 -현재 엘리베이터의 위치를 1으로 놓고 5층 상승 요청, 4층 내림 요청 두개 생성 -두 요청에 대해 더 가까운 요청 판별 후 그 요청의 목적지 확인
Estimated Result	결과로 된 요청의 목적지는 4가 되야 할 것이다.
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	**8. testArrivalCalibration**
Objective(Procedure)	-Elevator 내의 arrivalCalibration 적용 되는지 판별
Estimated Result	
Actual Result(Test Pass/Non-Pass)	PASS

Test Case Name	**9. testHandleAboard**
Objective(Procedure)	-QueueAlgorithm의 탑승요청 handleAboard() 메소드 기능 확인 -11층에서 상승요청을 의미하는 요청 전달 -탑승요청 큐에 해당 요청 존재하는지 확인
Estimated Result	해당 요청 존재
Actual Result(Test Pass/Non-Pass)	**ERROR**

Test Case Name	10. testEmergencyDestination
Objective(Procedure)	-비상 프로토콜을 위한 자동 비상 목적지 설정, 정차요청 기능 작동 확인
Estimated Result	엘리베이터의 목적지 변경
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	11. testFire
Objective(Procedure)	-화재기능
Estimated Result	큐내 모든요청 삭제됨
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	12. testBlack
Objective(Procedure)	-정전기능
Estimated Result	큐내 모든요청 삭제됨
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	13. testHandleSelect
Objective(Procedure)	-QueueAlgorithm의 하차요청 handleSelect() 메소드 기능 확인 -6층에서 정차를 하도록 하는 요청 전달 -탑승요청 큐에 해당 요청 존재하는지 확인
Estimated Result	해당 요청 존재
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	14. testHandleCancel
Objective(Procedure)	-존재하는 요청 삭제하는 기능인 HandleCancel() 메소드 작동 확인 -한 개의 요청을 넣은 뒤 취소요청 (HandleCancel) 적용 후 해당요청 존재 여부 확인
Estimated Result	해당 요청 없음
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	15. testSetNextDest
Objective(Procedure)	- setNextDestinationByComparison() 메소드 작동 확인 - 엘리베이터 1에대한 메소드 호출 후 엘리베이터의 목적지 확인
Estimated Result	목적지 변화
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	16. testAdd
Objective(Procedure)	-목적지에 도착하여 arrivecalibration 중 해당 하중 내리기/태우기 기능 작동 확인
Estimated Result	
Actual Result(Test Pass/Non-Pass)	ERROR

Test Case Name	17. testDepart
Objective(Procedure)	-엘리베이터 출발 기능 작동 확인
Estimated Result	
Actual Result(Test Pass/Non-Pass)	ERROR

4. Initial Analysis of Unit Testing Results

17개의 Unit Test에 대해 9개의 Test가 Fail이 아닌 Error가 나온데 대하여 해석을 간단하게나마 시도해 보고자 한다. 일차적으로 모든 Error들은 NullPointerException이 호출되어 나타났다.

9. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```
10. NullPointerException : QueueAlgorithm line 330
```
if(targetElev.getDirectionDelta()==1){
```
11. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```
12. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```
13. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```
14. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```

15. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```
16. NullPointerException : Elevator line 127
```
QueueNode selectNode=Fluxvator.queueAlgorithm.elev1DestinationNode;
```
17. NullPointerException : QueueAlgorithm line 145
```
Fluxvator.elev1.stopping();
```

10. 과 16. 을 제외한 모든 Test에서의 오류는 QueueAlgorithm 클래스 line 145의 코드로 인해 나타났다. 코드를 보면 엘리베이터, 알고리즘, 큐, 시뮬레이터들의 인스턴스 와 UI 등을 load 시켜주는 Fluxvator 클래스에서의 elevator를 직접 액세스하게 되는데, 지금 우리가 하고 있는 단위 테스팅에서는 단위 테스트 클라스에서 인스턴스화 시킨 엘리베이터를 불러오게 되어야 테스트가 진행 되기 때문에 없는 elevator를 허상 참조하게 되어 nullpointer가 나타나는 듯 하다. 이는 현재 아직 우리의 구현 버전이 객체지향 규칙을 제대로 따르지 못해 일어나는 것이 아닌지 하는 의문이 든다.

다음으로 10. 에서의 코드는 QueueAlgorithm 클래스 line 330의 코드로 인해 나타났다. 코드를 참조하면 해당 메소드에서 targetElev의 directionDelta를 get 하게 되는데, 이 메소드 초기에서 targetElev를 정의할 때 input으로 들어온 elevID에 따른 targetElev에 엘리베이터 배정을 역시나 위의 케이스 처럼 Fluxvator의 엘리베이터를 직접 액세스 하게 되어 일어난 에러라 보여진다.

마지막으로 16. 에서의 코드는 Elevator 클래스의 line 127 으로 인해 나타났다. 엘리베이터의 목적지 노드를 정할 때 설계에서는 한번에 목적지를 한 개만 정하기로 하였고 한 목적지에서 동시에 진행될 수 있는 요청이 존재할때만 목적지를 두개만 정하기로 하였는데, 구현당시 목적지 노드를 두개를 구현하였다. 헌데 그럴 경우 후자와 같은 동시에 진행될 요청이 존재하지않는 한 한 개의 목적지 노드는 항상 null이 되므로 nullPointer값이 리턴되는 것이 아닌가 하는 의문이 든다.