C++

- The Convenient ATM

Software Modeling & Analysis

OOPT 2nd Cycle

Final Report

T2

201411262 김도현

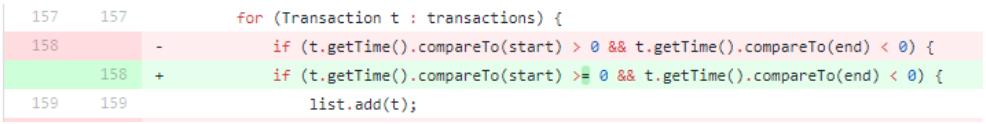
201411271 박상우

201411312 장하나

201411316 정진호

INDEX

1. System Test Response

내용	거래 조회할 때 시작날짜와 끝날짜를 동일하게 입력하였을 때, 그 날의 거래 내용이 조회되지 않는 문제가 발생.
원인	시작하는 날짜를 비교해줄 때, 등호를 생략하여 발생.
대응	해당 조건을 if 문에 추가 

내용	메인 화면/관리창에서 각각 언어 변경을 했을 때 다른 창에 즉시 적용되지 않는다.
원인	유저창과 관리자창을 동시에 사용한다는 상황을 설정하지 않아, 각각의 창에서 언어변경을 했을 때, 다른 창을 동기화하는 코드가 없어서 발생.
대응	Main 클래스에 Singleton 패턴을 적용하여 frame 프로퍼티에 접근이 가능하게 만들어서 어느 한 쪽에서 언어를 변경할 시엔, 다른 창도 초기화면으로 돌아가 인터페이스 언어를 바꾸도록 변경.

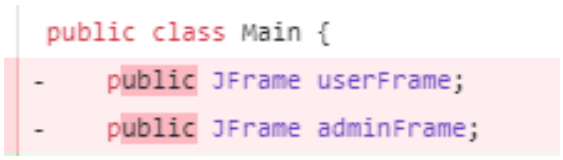
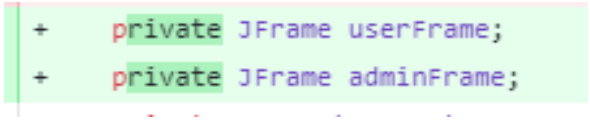
내용	계좌번호를 입력할 때는 invalid 인 경우와 incorrect 인 경우에 따라 뜨는 오류창이 다른데 비밀번호를 입력할 때는 invalid 인 경우와 incorrect 인 경우에 따라 뜨는 오류창이 같다. (invalid: 형식이 맞지 않을 경우, incorrect: 형식은 맞으나 데이터스토어 내 맞는 데이터가 없을 경우)
원인	오류 발생시 뜨는 창을, 원인에 따라 구별하지 않아서 다른 오류임에도 같은 메시지를 출력하는 결과가 발생했다.
대응	오류 상황을 보다 명확히 알 수 있도록 원인에 따라 출력되는 메시지를 변경하였다.

내용	복권 주차에 0 이나 실제 존재하지 않는 주차를 입력해도 오류로 인식하지 않는다. 복권 번호에 0 을 입력하여도 오류로 인식하지 않는다.
원인	주차 및 로또 번호를 판별하는 식에서 등호를 빠뜨려 0 도 올바르게 인식하도록 함.
대응	판별식에 등호를 추가해 올바른 주차와 번호를 입력 받을 수 있게 함. 주차는 로또 시작 주차와의 차이를 계산해서 판별함..

2. Static Analysis Response

내용	카드 분실신고 UI 코드에서 switch 문에서 재발급 확인 버튼 눌렀을 때와 취소 버튼을 눌렀을 때 모두 초기화면으로 돌아가서 코드가 중복되는 문제가 있었음.
	`switch (answer) {`
	`case JOptionPane.YES_OPTION:`
	`try {`
	`system.askRenewCard(true);`
	`} catch (DataStoreError ex) {`
	`}`
	`JOptionPane.showMessageDialog(parentFrame,`
	`setLocalizedString(system, card[i].toString() + " 카드를 재발급 요청하였습니다", card[i].toString() + " is requested to renew"),`
	`"Info",`
	`JOptionPane.INFORMATION_MESSAGE);`
	`case JOptionPane.NO_OPTION:`
	`parentFrame.setContentPane(new SelectFunction(parentFrame, system).getPanel());`
	`parentFrame.invalidate();`
	`parentFrame.validate();`
	`break;`

대응	확인/취소를 if 문으로 확인 버튼을 눌렀을 때만 판별한 후, 초기화면으로 돌아가는 루틴을 하나로 통합하여 코드 가독성 향상.
	`if (answer == 0) {`
	`try {`
	`system.askRenewCard(true);`
	`} catch (DataStoreError ex) {`
	`}`
	`}`
	`JOptionPane.showMessageDialog(parentFrame, setLocalizedString(system, card[i].toString() + " 카드를 재발급 요청하였습니다", card[i].toString() + " is requested to renew"),`
	`"Info",`
	`JOptionPane.INFORMATION_MESSAGE);`
	`parentFrame.setContentPane(new SelectFunction(parentFrame, system).getPanel());`
	`parentFrame.invalidate();`
	`parentFrame.validate();`
	`

내용	Main 클래스를 Singleton 패턴으로 만들어서 인스턴스를 받을 때 userFrame, adminFrame 속성이 public 으로 선언되어 있어서 보안 측면에서 문제가 됨.  public class Main { - public JFrame userFrame; - public JFrame adminFrame;
대응	Public 지시자를 private 으로 수정.  + private JFrame userFrame; + private JFrame adminFrame;

내용	IntelliJ 자동완성을 사용하면서 사용하지 않는 라이브러리에 대한 import 문이 생성됨.
대응	IntelliJ 내의 optimize import 기능을 사용하여 사용하지 않는 import 문 정리.

내용	카드 분실 신고 및 재발급 UI 폼에서 card 번호를 text 로 입력받았음에도, toString()으로 변환하여 중복되게 처리하였다.

대응	toString() 메소드를 호출하는 부분을 제거하였다.
	 
	setLocalizedString(system, card[i].toString() + " 카드를 중지처리하였습니다", card[i].toString() + " is now closed"),
	setLocalizedString(system, card[i] + " 카드를 중지처리하였습니다", card[i] + " is now closed"),
	"Info",
	JOptionPane.INFORMATION_MESSAGE);
	lic class ReportLostCard extends JFrame {
	} catch (DataStoreError ex) {
	}
	}
	JOptionPane.showMessageDialog(parentFrame, setLocalizedString(system, card[i].toString() + " 카드를 재발급 요청하였습니다", card[i].toString() + " is requested to renew"),
	JOptionPane.showMessageDialog(parentFrame, setLocalizedString(system, card[i] + " 카드를 재발급 요청하였습니다", card[i] + " is requested to renew"),

내용	사용자가 입력한 지폐를 각 지폐 종류별 몇 장인지 계산하는 루틴에서 지폐 종류가 존재하지 않을 경우 null 값을 반환하여, 예기치 못한 에러가 발생하였을 시 지폐 수량을 제대로 계산하지 못하였다.

대응	지폐 종류를 인식하지 못할 경우 지폐 수량은 없다고 처리하였다.
	`default:`
	`return null;`
	`return new int[]{0,0,0,0,0,0,0,0,0,0};`

내용	`public boolean checkAdminAccount(String adminID, String adminPW) {`
	`if (this.id.equals(adminID) && this.password.equals(adminPW)) {`
	`return true;`
	`}`
	`return false;`
	`return this.id.equals(adminID) && this.password.equals(adminPW);`
	`}`

대응	불필요한 if 문 정리.

내용	BufferReader 를 사용한 뒤 close 를 하지 않아 메모리 누수가 우려되는 상황이었다. 또, 파일 인코딩이 지정되지 않아 예기치 못한 오류가 발생할 수도 있었다.
	`try {`
	`BufferedReader br = new BufferedReader(new FileReader("data/user/" + userId + ".json"));`
	`try (BufferedReader br = Files.newBufferedReader(path, StandardCharsets.UTF_8)) {`

대응	Try-with-resources 문을 사용하여 BufferReader 의 수명을 한정 지어줬고, 파일의 인코딩을 명시하였다.

내용	예외 발생시 Stack Trace 를 그대로 노출시켜 보안 문제가 있었다.
대응	Stack Trace 를 출력하는 부분을 제거하였다.

내용	Break 를 case 문 마지막에 넣지 않은 switch 문이 존재하였다.
	
	`switch (answer) {`
	`  case JOptionPane.YES_OPTION:`
	`  try {`
	`  system.askRenewCard(true);`
	`  } catch (DataStoreError ex) {`
	`  }`
	
	`  JOptionPane.showMessageDialog(parentFrame,`
	`  setLocalizedString(system, card[i].toString() + " 카드를 재발급`
	`  "Info",`
	`  JOptionPane.INFORMATION_MESSAGE);`
	`  case JOptionPane.NO_OPTION:`
	`  parentFrame.setContentPane(new SelectFunction(parentFrame, system)`
	`  parentFrame.invalidate();`
	`  parentFrame.validate();`
	`  break;`
	`if (answer == 0) {`
	`  try {`
	`  system.askRenewCard(true);`
	`  } catch (DataStoreError ex) {`
	`  }`
	`}`
	`JOptionPane.showMessageDialog(parentFrame, setLocalizedString(system, card`
	`  "Info",`
	`  JOptionPane.INFORMATION_MESSAGE);`
	`parentFrame.setContentPane(new SelectFunction(parentFrame, system).getPane`
	`parentFrame.invalidate();`
	`parentFrame.validate();`

대응	해당 switch 문을 if 문으로 바꾸었다.

내용	수정되지 않아야 변수들이 일반 변수로 선언되어 있다.
	
	`public static int wonSize = 4;`
	`public static int dollarSize = 7;`
	`public final static int wonSize = 4;`
	`public final static int dollarSize = 7;`

대응	상수 변수로 선언한다.

3. OOPT Review

소프트웨어 모델링 및 연습 수업을 들으면서 느낀 점
실습을 진행하면서 소프트웨어 모델링 작업이 소프트웨어 개발에서 매우 중요하다는 사실을 깨달았다. 실제 구현단계 전까지 점점 더 세세하게 설계 및 수정하는 작업을 반복하는 것이 쉽지 않았지만 이러한 방법을 통해 실제 소프트웨어의 구현이 좀 더 용이했다는 사실을 알았다. 또한 유닛 테스트를 비롯한 다양한 소프트웨어 검증 단계를 거치며 구현을 하니 수정이 훨씬 쉬워졌으며 정적 분석을 통하여 평소 코딩 스타일에서 미흡한 부분을 고칠 수 있었다.

우리 조가 생각하는 Object Oriented Analysis & Design 의 장점과 단점
우리 조가 생각하는 OOAD 의 장점은 요구사항의 정리가 매우 용이했다는 것이다. Waterfall 과 달리 꾸준한 revise 를 수행하므로 소프트웨어 아키텍처 및 문서에 대한 수정이 없어 큰 일이 일어나지 않는다는 점이 있다. 요구사항의 수정이 있더라도 반영이 매우 쉽다는 장점도 있다. OOAD 를 사용하면 클라이언트가 결과물을 최대한 빠르게 볼 수 있어서 개발자의 성취감과 클라이언트의 만족을 동시에 얻을 수 있다. 또, 실제 구현 단계에서도 OOAD 의 장점을 느꼈는데, 우리 조가 반복적으로 수정하고 설계한 것을 바탕으로 코드를 만드니까 클래스를 만들고 내부 코드를 채워가는 과정이 굉장히 순조로웠다. 그러나 단점으로는 큰 수정이 없는 대신 작은 수정이 많이 필요할 수 있어서 사실상 큰 수정 한번 하는 것과 큰 차이가 없는 상황이 발생할 수 있다는 것과 잦은 수정으로 인한 피로감이 든다는 점이다.

소프트웨어 검증팀과의 협업 후기
소프트웨어 검증팀이 선택한 환경이 다른 팀들의 환경과는 달리 생소하여서 처음에는 익숙해지는 데 시간이 필요하였다. 그러나 서로 간의 소통이 원활하여서 CTIP 환경의 사용에 무리가 없었고 팀 간의 협업 또한 수월하게 진행되었다.