융합 UI 실습 과제  
키오스크 제작

학번: 2401110252

이름: 박지수

보고서 구성

1. **개요**[1](#_top)
2. 요구 사양[2](#_요구_사양_)
3. 키오스크 진행도[4](#_코드_설명_)
4. **실행결과**[5](#_코드)
5. **코드**[6](#_결과_스크린샷)
6. **고찰**[7](#_고찰)

# [**개요**](#_top)

#### [요구 사양](#_top) 첫번째 창에서는 버튼을 이용해 해당하는 메뉴를 선택하면 ListBox로 물품이 담기고 주문하기 버튼을 누르면 ListBox와 물품들의 금액이 출력된다. 버튼을 이용해 돈을 투입하여 결제까지 마친다면 또 새로운 창이 뜨게 되는데 산 물품과 결제하고 남은 돈이 표시되며 닫기 버튼을 누르면 첫번째 창만 남게 되고 샀던 물품들은 초기화 된다.

#### [키오스크 진행도](#_top)

# **[코드](#_top)**

# **[결과 스크린샷](#_top)**

|  |
| --- |
|  |
| 주문을 마친 경우 |
|  |
| 주문을 안하고 결제하러 갈 경우 |
|  |
| 금액을 부족하게 투입한 경우 |
|  |
| 결제 창에서 다지우고 결제하려는 경우 |
|  |
| 특정 조건을 만족했을 경우 |
|  |
| 결제 완료된 최종 화면 |

# **[고찰](#_top)**

이번 과제를 진행하면서 3가지 정도 막힌 부분이 있었다.

첫번째는 ListBox를 가져오는 것이었다.  
ListBox items를 가져오기 위해 객체를 함수 파라미터에 넣고(ref 사용가능)  
“=” 할당 연산자를 이용해 변수나 배열처럼 가져올 수는 있었지만 얕은 복사라서  
실행해 보니 OrderForm의 창에서 뚝 때서 다음 창으로 가져오는 식이었다.  
  
정말 무식하게 가져오는 식이라 Purchase창에서 역으로 OrderForm 이동할 때  
창이 바뀌어서 OrderForm에서 ListBox가 보이지 않는다.  
위치 좌표를 미리 저장한 다음 새로 지정하고 Controls.Add() 함수를 해야  
다시 나타난다.  
  
메모리 관리면에서는 이런 방식이 좋을 것 같지만 요즘 하드웨어를 생각 한다면  
굳이 이렇게까지 할 필요없이 깊은 복사를 통해 새로운 객체를 만드는 것도  
나름 방법이 될 수도 있다.

그래서 1차원적인 생각으로 OrderForm의 ListBox item들을 다 가져와서  
Purchase, ResultForm에 하나씩 다 넣는 방향으로 진행했다.  
이 때에는 기존 ListBox 함수인 ListBox.items.AddRange 함수를 사용해도 되고  
강의시간에 배운 foreach를 사용해도 된다.  
  
두번째는 Forms의 이해도가 낮아서 나온 LabelSize로 인한 문제였다.  
  
두 문자열을 합치기 위해 + 연산자를 사용하여 합쳤는데 뒷글자가 안나오는  
상황이었다. 여느 때처럼 합칠때를 디버깅하여 보았는데도 문자열이 둘 다 제대로  
들어가 있고 라벨의 텍스트도 제대로 들어가 있었다.  
  
순간 이게 진짜 버그인지라고 생각할 만큼 당혹스러웠지만  
문자열 두개를 합치는 것인 만큼 순서를 바꿔보니 앞의 문자열만 출력되고  
문자열 한 개만 출력하니 제대로 출력되는 것이었다.  
  
아무래도 라벨 속성의 문제인 것 같아 찾아보니 LabelSize를 Auto로 설정하니  
  
잘 출력되었다.  
  
정말 어이없는 문제였지만 당시에는 정말 힘든 순간이었다.  
  
세번째는 Thread 사용이 어려워 따로 공부를 해야 겠다고 느꼈다.  
  
  
그렇게 하기 위해서는 지난번 과제에서 했던 잘못, 실수를 줄이기 위해  
제일 큰 패착이라고 생각했던 ‘코드부터 작성하기’보다는 작성 전 구조를 설계하고  
‘어떻게 작성할 것인지 생각’부터 했다.  
  
그다음 생각을 텍스트와 그림을 곁들여서 기획서를 쓴다는 느낌으로  
정리하고 그 내용을 참고하면서 코드를 작성해 나갔다.  
  
아직 배우는 과정이고 하니 완벽하게 구조를 설계하고 할 수는 없었지만  
생각을 해봤다는 것과 그 생각을 글과 그림으로 그렸다는 것이 구조를 짤 때  
큰 힘이 되었던 것 같다.  
  
자바 수업 시간 때는 멤버 변수와 클래스 내 함수 작성, main으로 가져와서  
실체화만 했을 것이다.

그것 가지고는 이번 과제를 하기에 부족하다고 느낄 찰나 유튜브 알고리즘에  
SOLID 원칙을 소개하는 영상이 있었다. 덕분에 내가 지금 작성한 것이  
원칙에 아주 위배되고 있다는 것을 자각하고 이를 고치기 위해서 주말 동안  
상속과 합성을 알아보았는데 지금 과제에서 적용하기에 너무 좋은 상황이었다.  
  
공부하기 전에는 Player 클래스와 Enemy 클래스, 수치를 입력하는 Value 클래스  
이렇게 나눴는데 Value 클래스를 없애고 Unit이라는 추상 클래스를 만들었다.  
  
설계를 Unit이라면 무조건 가지고 있을 체력, 기본 공격, 이름 정도만 Unit 클래스에  
넣고 SkillUnit에는 Skill과 관련된 MP, Skill 목록, 크리티컬 계수등 Skill과 관련된  
멤버를 넣었다.  
  
그리고 Wizard Interface에는 Wizard에 해당하는 값을 전부 넣었다. Wizard의  
HP, MP, DMG, Skill등 다 세팅할 수 있게 함수도 넣었다.  
이는 Monster 클래스도 마찬가지다.  
  
다음으로 크게 바꾼 것이 공격 관련 내용이다.  
  
기존에는 Combat 클래스 안에 Player가 기본공격, 마법 공격하는 함수 만들고  
Enemy가 공격하는 함수 이렇게 만들었지만,  
  
생각해보니 이러면 Enemy가 새로 생기거나 Wizard가 아니라 Knight 이렇게 새로운 클래스가 생기면 다 하나씩 만들 생각을 하니 이건 매우 잘못되었다고 생각 해  
  
Unit으로 파라미터를 받았다.  
싸움을하는 객체는 유닛임은 틀림없어서 모든 객체가 들어가서  
공격을 할 수 있게 다시 재구성하였다.  
  
그리고 Combat Class는 공격 외에 회피, 도망, 소모품 사용하는 것처럼 추가 될  
사항은 새로운 interface를 만들어서 합치면 Combat을 타고 들어가면 되니  
  
Unit쪽에 새롭게 멤버나 함수 추가하고 관련  
interface를 만들면 되니 크게 수정 할 것이 없다.  
  
기존처럼 공격을 하나씩 만들었으면 추가 될 사항도 하나씩 만들어야 하는데  
작업량이 정말 말도 안 된다.  
  
이번 과제를 하면서 어떨 때 상속을 하고 interface를 만들어 합성해야 할 지  
경험이 생겼고 함수 하나, 변수 하나 만들 때 마다 꼭 필요한지,  
  
재사용성이 높게 작성하였는지 신중히 검토하고 설계해야 한다는  
뜻깊은 교훈을 얻었다.