1. 개요

사용자의 입력에 따라 회원 및 비회원으로 영화를 예매하는 프로그램인 Ticketer를 검사한 결과임.

1. 단위 검사

# main(Ticketer?) 클래스

* + 1. main 함수

input 은 global 변수, MenuManager 클래스는 MM으로 호출한 상태.

1. 부분 설계

* 사용자의 입력을 무한 루프를 탈출할 때까지 받아가며 다음과 같은 진행을 한다.

1. “EXIT”가 입력된 경우 : 루프를 탈출하고 “프로그램이 종료됩니다.”를 출력하고 프로그램을 종료한다.
2. “RESTART”가 입력된 경우 :

MM.getMI().getwhere == 0 일 때 – 아무 것도 진행하지 않고 다음 입력을 받기 위해 input을 비우고 다시 입력을 대기한다.

MM.getMI().getwhere != 0 일 때

MM.getMI().setMI()함수를 호출해 self.\_\_menuNum = 4200, self.\_\_isMember = False, self.\_\_where = 1로 변경한다.

MM의 변수 username과 password를 None으로 변경한다.

출력 화면을 clear한다.

MM.print\_login\_menu() 함수를 호출해 로그인 메뉴를 출력한다.

다음 입력을 받기 위해 변수 input 을 비운다.

1. “BACK”이 입력된 경우 :

MM.getMI().getwhere != 2 일 때 – 아무 것도 진행하지 않고 다음 입력을 받기 위해 input을 비우고 다시 입력을 대기한다.

MM.getMI().getwhere == 2 일 때

MM.getMI().setMI()함수를 호출해 self.\_\_menuNum = 4300 으로 변경한다.(나머지는 그대로)

출력 화면을 clear한다.

MM.print\_main\_menu() 함수를 호출해 로그인 메뉴를 출력한다.

다음 입력을 받기 위해 변수 input 을 비운다.

1. 그 외 :

MM.manageMenu() 함수를 input을 인자로 호출한다.

다음 입력을 받기 위해 변수 input 을 비운다.

1. 검사 방법
2. Test Cases 및 결과 정리

# menuInfo 클래스

* + 1. setMI 함수

변수 mN은 int형, 변수 iM은 bool형, 변수 w는 int형으로 assert 조건을 걸어준 상태.

1. 부분 설계

* 인자로 받은 mN, iM, w 를 클래스 변수인 self.\_\_menuNum, self.\_\_isMember, self.\_\_where에 각각 넣어줘야함.
* mN, iM, w 의 자료형이 assert 조건에 위배된다면, AssertionError 메시지를 출력해야함.
* 정확히 세 개의 인자를 받아야함. 인자의 개수가 틀릴 경우 TypeError 메시지를 출력해야함.

1. 검사 방법
2. Test Cases 및 결과 정리

# MenuManager 클래스

# Menu 클래스

# TypeChecker 클래스

* + 1. ID\_check 함수

1. 부분 설계

* ID를 입력받아 검증

1. 검사 방법
2. Test Cases 및 결과 정리

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + 1. date\_check 함수

1. 부분 설계
2. 검사 방법
3. Test Cases 및 결과 정리

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + 1. time\_check 함수
    2. pw\_check 함수
    3. movieTitle 함수
    4. cardNum 함수
    5. checkyoursheet 함수(오타지만 이미 만들었으므로)
    6. checkReservationCode 함수

input 은 global 변수, MenuManager 클래스는 MM으로 호출한 상태.

1. 부분 설계

* ㅁㅁ

1. ㅇㅇ
2. ㄷㄷ

# TypeChecker 클래스 2차 설계

* + 1. ID\_check 함수

1. 부분 설계

* FileManager 클래스가 추가되어 csv 파일은 FileManager클래스 내의 함수로 처리하므로 입력 받은 아이디를 UserList.csv 파일 내의 내용과 대조하는 구문 삭제. assert문 추가

1. 검사 방법
2. Test Cases 결과 및 정리
   * 1. date\_check 함수
3. 부분 설계
4. 검사 방법
5. Test Cases
   * 1. time\_check 함수
6. 부분 설계

* date\_check 함수의 변경사항과 동일

1. ㅇ
   * 1. checkMovieTitle 함수
2. 부분 설계

* FileManager 클래스가 추가되어 csv 파일은 FileManager클래스 내의 함수로 처리하므로 입력 받은 영화 제목을 MovieList.csv 파일 내의 내용과 대조하는 구문 삭제   
  출력 뿐만 아니라 입력도 숫자+영화제목의 5.movie와 같은 형식으로 이루어져야 하므로 영화 제목 앞에 숫자,온점(‘.’)이 포함되었는지도 검증하도록 부분 설계 변경. assert문 추가

1. ㅇ
2. ㅇ
   * 1. cardNum 함수
3. 부분 설계

* 카드번호가 CardList 내에 존재하는지 확인하는 구문 삭제, assert문 추가

1. 검사 방법

* TypeChecker 클래스 상에 테스트를 위한 코드를 작성 :
* Test1 : int, char, str 등 다양한 형식의 값을 인자로 cardNum함수를 호출. assert 오류가 제대로 출력되는지 확인  
  Test2 : 반복문에 조건문을 넣어서 입력값을 인자로 cardNum 함수를 호출. 이 과정을 반복. 조건문은 반환값에 따라 출력 메시지가 달라지게 하는 조건문인데, 카드번호가 올바른 형식이면 “올바른 형식” 출력, 틀린 형식이면 “틀린 형식 출력.

1. Test Cases 결과 및 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(호출) | 예상 결과 | 실제 결과 | 비고 |
| 1 | AssertionError | AssertionError | Test1 |
| 2 | AssertionError | AssertionError |
| 1.5 | AssertionError | AssertionError |
| ‘a’ | AssertionError | True 반환 | Test1, char형에서 오류가 없었음 |
| 1234123412ee | 잘못된 형식 | 올바른 형식 | Test2, 올바르지 않은 형식이나 True가 반환됨 |
| 1234123 | 잘못된 형식 | 잘못된 형식 | Test2 |
| 12341234 | 올바른 형식 | 잘못된 형식 | Test2, 올바른 형식임에도 False 반환 |

* + 1. checkyoursheet 함수
    2. checkReservationCode 함수

1. 부분 설계

* 파이썬에서는 배열의 명시된 범위 바로 앞의 요소까지만 인자로 받는다는 것을 확인.  
  code[0:7] 🡪 code[0:8]  
  code[10:13] 🡪 code[10:15]  
  code[8:9] 🡪 code[8:10]으로 수정

1. 검사 방법

* 1차 설계와 동일

1. Test Cases 결과 및 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(호출) | 예상 결과 | 실제 결과 | 비고 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# TypeChecker 클래스 3차 설계

checkyourseat함수만 단독으로 실행할 땐 올바르게 동작함을 확인하였으나, 프로그램의 일부로 동작할 땐 제대로 동작하지 않음. 원인은 입력 문자열 스트링을 FileManager 클래스의 seats\_to\_list 함수로 한 좌석씩 나눠서 좌석들이 담긴 리스트를 반환한 후 checkyourseat함수로 검증하였는데, 형식 검증보다 문자열을 나누는 과정이 먼저 이루어져 발생한 오류였다. 좌석 문자열의 형식이 먼저 검증된 후에 배열로 반환되어야 한다고 판단, 이 클래스에서 좌석 문자열 형식을 검증하고 배열로 반환하는 기능까지 수행하도록 설계 변경하였다. 이 과정에서 FileManager의 seats\_to\_list는 사용할 필요가 없어졌고, 코드의 효율성을 위해 한 좌석씩 검증하는 checkOneSeat 함수를 추가했다. checkMovieTitle 함수의 or조건문 세 개를 수정했고, 테스트를 진행했다. cardNum의 조건문을 수정했다.

* + 1. cardNum 함수
    2. checkMovieTitle 함수
    3. checkOneSeat 함수

1. 입력된 문자열의 길이가 2이고, 첫번째 글자는 대문자이고 두번째 글자는 숫자인지 검증.  
   세 조건이 모두 맞다면 True 반환, 하나라도 틀리면 False 반환.
2. 검사 방법  
   TypeChecker 클래스 상에 테스트를 위한 코드를 작성 : 반환값에 따라 출력 메시지가 달라지는 조건문을 추가. 올바른 형식이면 “올바른 형식” 출력, 틀린 형식이면 “틀린 형식 출력. 반복문에 조건문을 넣고 checkOneSeat 함수를 반복해서 호출하고 반복해서 입력.
3. Test Cases 결과 및 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(호출) | 예상 결과 | 실제 결과 | 비고 |
| 11 | 틀린 형식 | 틀린 형식 |  |
| AA | 틀린 형식 | 틀린 형식 |  |
| (공백 하나) | 틀린 형식 | 틀린 형식 |  |
| (공백 둘) | 틀린 형식 | 틀린 형식 |  |
| (공백 셋) | 틀린 형식 | 틀린 형식 |  |
| A1 | 올바른 형식 | 올바른 형식 |  |
| A2 | 올바른 형식 | 올바른 형식 |  |
| a1 | 틀린 형식 | 틀린 형식 |  |
| a2 | 틀린 형식 | 틀린 형식 |  |
| E# | 틀린 형식 | 틀린 형식 |  |
| V% | 틀린 형식 | 틀린 형식 |  |
| V5 | 올바른 형식 | 올바른 형식 |  |
| VV3 | 틀린 형식 | 틀린 형식 |  |
| V33 | 틀린 형식 | 틀린 형식 |  |
| $% | 틀린 형식 | 틀린 형식 |  |
| M9\ | 틀린 형식 | 틀린 형식 |  |
| B7 | 올바른 형식 | 올바른 형식 |  |
| g8 | 틀린 형식 | 틀린 형식 |  |
| ㅎㅎㅎ | 틀린 형식 | 틀린 형식 |  |
| ㅎ2 | 틀린 형식 | 틀린 형식 |  |

* + 1. checkSeatsList 함수

1. 부분 설계

* 기존의 checkyoursheet함수 대신 형식 검증, 배열 반환까지 수행하기 위해 만든 함수로 문자열을 입력받아 길이가 2 미만인지 우선 확인. 좌석 형식 상 문자열의 길이 2미만일 수는 없으므로 길이부터 틀리면 빈 배열을 반환.  
  다음으로 길이가 3 미만인지 확인. 맞으면 checkOneSeat함수를 호출해 형식 검증하고 True를 반환하면 좌석 한 개가 올바르게 입력된 문자열이므로 해당 문자열을 list라는 배열에 저장해서 반환, checkOneSeat함수가 False를 반환하면 빈 배열을 반환  
  문자열의 길이가 3 이상이면 ‘~’가 1개 포함되어 있는지 확인. 있으면 ~를 기준으로 문자열을 자르고, 잘린 문자열에 대해 checkOneSeat 함수로 형식 검증. 맞으면 firstseat에 저장된 앞 문자열의 영문+숫자, … , lastseat에 저장된 뒤 문자열 영문+숫자인 배열을 반환. checkOneSeat함수가 False를 반환하면 빈 배열을 반환.  
  문자열의 길이가 3 이사이고 ‘~’가 1개 포함하지 않고 ‘,’는 포함한다면(~~나 ~, 반드시 검증) ‘,’를 기준으로 문자열을 잘라서 list라는 배열에 담는다. 배열의 모든 인자를 checkOneSeat함수로 검증하고 false를 반환하는 인자가 있으면 list 배열을 비워서 반환하고 함수 종료. 한 개도 없다면 ‘,’를 기준으로 잘린 문자열 저장된 list배열 반환

1. 검사 방법

* 입력된 값을 인자로 checkSeatsList함수를 호출하고 반환된 배열의 내용을 출력하게 한다. 올바른 형식이면 좌석번호를 반환하고, 형식에 오류가 있으면 빈 배열을 반환한다.

1. Test Cases 결과 및 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(호출) | 예상 결과 | 실제 결과 | 비고 |
| (공백 한 칸) | [] | [] |  |
| a | [] | [] |  |
| a2 | [] | [] |  |
| A2 | [‘A2’] | [‘A2’] |  |
| A1~A3 | [‘A1’, ‘A2’, ‘A3’] | index out of range | 배열 범위 오류 |
| A1,A1 | [] | [‘A1’, ’A1’] | 첫 문자열에 대해 제대로 검증하지 못함 |
| A3,A3 | [] | [‘A3’, A3’] |
| A1,A2,A1 | [] | [] |  |
| A1,A3,A2,A2 | [] | [] |  |

오류 분석 : split()을 사용한 부분에서 오류가 나는데, split은 아무 조건이 주어지지 않았을 경우 공백을 기준으로 split함. 이 문자열에는 공백이 없어서 기존 코드대로 진행하면 lastseat[1]에 아무 것도 들어가지 않아서 lastseat[1]에 접근하려고 하면 오류가 발생한다.  
같은 좌석을 구분하는 기능이 동작하긴 하나 첫 좌석과 두번째 좌석이 같은 경우는 제대로 동작하지 않음

# TypeChecker 클래스 4차 설계

* + 1. checkSeatsList 함수

1. 부분 설계 수정사항

* 굳이 문자열을 나누지 않아도 이후 코드를 진행하는데 문제가 없을 거라 판단, 아래의 코드에 split을 사용하지 않도록 수정하였다.  
  firstseat = list[0].split() -> firstseat = list[0]  
  lastseat = list[1].split() -> lastseat = list[1]  
  또한 첫번째 문자열과 두번째 좌석이 같은 경우를 제대로 구분하지 못해서 find대신 count를 사용하여 테스트하였다.  
  strseat[3:].find(i)>1-> strseat.count(i)>0

1. 검사 방법

* 3차 설계와 동일하고, 각 좌석에 대한 형식 검증은 checkOneSeat함수가 처리하므로 좌석 입력 자체가 틀린 경우에 관한 검사를 여러 번 진행하지는 않았다

1. Test Cases 결과 및 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(호출) | 예상 결과 | 실제 결과 | 비고 |
| A1 | [‘A1’] | [‘A1’] |  |
| A2 | [‘A2’] | [‘A2’] |  |
| A3 | [‘A3’] | [‘A3’] |  |
| A1~A5 | [‘A1’, ‘A2’, ‘A3’, ‘A4’, ‘A5’] | [‘A1’, ‘A2’, ‘A3’, ‘A4’, ‘A5’] |  |
| A5~A2 | [] | [] |  |
| A9~A7 | [] | [] |  |
| A7,B9 | [‘A7’, ‘B9’] | [‘A7’, ‘B9’] |  |
| A3,A3,B5 | [] | [] |  |
| A3,A3,A5, | [] | [] | 문자열 뒤의 ‘,’를 구분하는지 |
| A3~A4~A5 | [] | [] |  |
| A4~A5,B5 | [] | [] |  |
| A1,A1 | [] | [] |  |
| N5,N5 | [] | [] |  |
| N9~N10 | [] | [] |  |
| N1~N9 | [‘N1’, ‘N9’] | [‘N1’, ‘N9’] |  |
| e3 | [] | [] |  |
| e1~e3 | [] | [] |  |
| ,,, | [] | [] |  |
| , E6 | [] | [] |  |
| ,E9 | [] | [] |  |
| E7, | [] | [] |  |
| E7,E | [] | [] |  |
| E9,E9 | [] | [] |  |