문제해결 및 실습 : Java

과제 2 보고서

제출일	2024.10.20	전공	소프트웨어학과
과목	문제해결및실습:Java	학번	21011746
담당교수	박상일 교수님	이름	양현석

1. 좌석 예약 프로그램

a) 문제 분석

이 문제는 객체 배열을 사용하는 것이 중요한 문제입니다. 각각의 좌석에 대한 클래스와 예약을 관리하는 프로그램을 만들어서 사용하도록 합니다.

b) 문제 풀이

좌석에 대한 클래스는 좌석에 예약 여부를 확인하는 Boolean 변수와 예약자 이름을 저장하는 String을 저장합니다. 예약자가 없는 경우 "---"를 저장하도록 해서 문제와 동일하게 출력될 수 있도록 했습니다. 더하여 캡슐화를 명확히 하기 위해서 멤버 변수들을 Method를 이용해서 접근해야 합니다.

예약을 관리하는 프로그램은 3*10 크기의 2차원 SeatInfo 배열, 라인 당 총 최대 좌석의 개수(상수), 각 라인 당 예약자 수를 확인하기 위한 배열 pCnt 그리고 Scanner를 지속적으로 사용하기 위해 sc라는 변수에 Scanner을 저장합니다. 이 클래스의 메소드는 main 함수 내의 코드 길이를 줄이기 위한 설계를 기반으로 했습니다. 그래서 프로그램을 반복하여 돌리는 reserveProgram()이 존재합니다.

예약을 관리하는 클래스의 메소드들에 대해 더 알아보겠습니다. 먼저 getMode()입니다. 이 Method는 예약, 조회, 취소, 프로그램 종료 모드를 입력 받아 해당 메소드를 실행시킵니다. Switch 문을 활용해서 핸들링합니다. 그러나 이 때 잘못된 입력이 들어오게 되면 이를 방지하기 위해 반복문을 사용했습니다. 아래는 그 코드입니다.

```
int mode;

while (true) {
    System.out.print("예약 : 1, 조회 : 2, 취소 : 3, 끝내기 : 4 >> ");
    mode = sc.nextInt();

    if (mode < 1 || mode > 4) {
        System.out.println("잘못된 번호입니다. 다시 입력해주세요.");
    } else break;
}
```

다음은 reserve()입니다. 이는 예약 기능을 제공합니다. 먼저 모든 좌석이 가득차게 되면 예약이 불가하도록 return합니다. 만일 좌석이 가득 차지 않았다면, 사용자에게 어느 라인(S, A, B)에 예약을 할 것인지 입력 받습니다. 선택한 좌석이 가득 찼다면 재 입력을 요구하기 위해 반복문을 사용했습니다. 그 후 사용자에게 선택한 라인의 상태를 보여주고(printSeat), 이름과 번호를 입력 받습니다. 이때 입력 받은 번호가 예약이 된 상태라면 재 입력을 받도록 합니다.

다음은 cancelReservation()입니다. 이는 예약 취소를 관리합니다. 만일 전 좌석이 모두 비어 있다면, 알림을 보내고 함수를 종료합니다. 좌석이 비어 있지 않다면, 사용자에게 먼저 라인을 입력

받고, 그 후 이름을 입력 받습니다. 라인의 번호가 잘못 입력된 경우 반복문을 통해 재 입력을 받고, 해당하는 이름이 존재하지 않다면 마찬가지로 반복문을 통한 재 입력을 받도록 합니다.

위의 예약을 취소하는 과정에서 해당하는 이름이 있는지 찾는 기능을 findReservation()으로 수행합니다. 이 함수는 전 좌석을 돌면서 해당하는 이름의 예약자가 존재하는지 판단하고, 존재하면 pCnt[n]의 값을 1만큼 감소시키고 true를 반환, 아니라면 false를 반환합니다.

마지막으로 좌석 라인에 대한 정보를 출력하는 printSeat()입니다. 이 때 함수 재활용을 높이기 위해 좌석 라인의 인덱스를 입력 받고, S, A, B 중 하나를 출력합니다. 이를 구현하기 위해 다음과 같은 배열을 선언했습니다.

```
//  함수 재활용을 위한 배열
char[] lines = {'S', 'A', 'B'};
```

생성자에 대한 얘기를 하고 가자면, 객체 배열을 생성하기 위해 반복문을 사용했고, 배열 pCnt도 모두 0으로 초기화 시켰습니다.

c) 시행 착오

특별한 시행 착오는 없었지만, 재 입력을 받는 부분들을 재귀 함수를 이용해 구현하려 했었지만, 반복문을 활용해서 구현하는 것이 훨씬 효과적이라고 생각했습니다.

2. 추상 클래스 pairMap 구현하기

a) 문제 분석

추상 클래스를 상속 받아, Dictionary 클래스를 만듭니다. 이때 출력 결과를 신경써서 만들어야 합니다.

b) 문제 풀이

먼저 새로 만드는 클래스인 Dictionary의 변수 필드에 추가해야 하는 것은 현재 저장되어 있는 아이템의 개수를 저장하는 items입니다. 구현해야 하는 것 중 현재 저장된 아이템의 수를 출력하는 것이 있기 때문입니다.

가장 먼저 생성자에서 저장할 데이터의 길이를 입력 받아, key와 value의 배열을 생성합니다. 또 items를 0으로 초기화합니다.

Method get()입니다. 이는 입력 받은 key값을 탐색하여 존재한다면 해당하는 value를 반환, 없다면 null을 반환합니다.

Method put()입니다. 입력 받은 key와 value 쌍을 비어진 공간에 저장하는 역할을 합니다. 만일 key가 존재한다면 value만을 수정한 후 return합니다. 입력된 key가 존재하지 않는다면 “”와 비교

하여 비어있는 곳을 탐색한 후 저장합니다. 이때 items도 1 증가시킵니다.

다음은 delete()입니다. 이는 입력 받은 key값에 해당하는 key와 value를 모두 삭제하고 value를 반환합니다. 만일 존재하지 않는다면 null을 반환하도록 설계하였습니다.

마지막으로 length()입니다. 이는 캡슐화를 위해 items 값을 반환하도록 합니다.

c) 시행 착오

이 문제에서 시행 착오라 할 것은 없지만, null을 출력시키면 null이 문자열의 형태로 출력될 수 있다는 사실을 알게 되었습니다.

3. 도형 클래스와 연결 리스트 구현

a) 문제 분석

이 문제는 class의 업캐스팅과 Overriding을 이용해서 연결리스트를 구현하는 것입니다. SLL(Single Linked List)를 만들 것이고, Dummy Node가 아닌 head만 존재하는 형태로 설계했습니다. 자료 구조의 아이디어를 많이 사용해야 할 것입니다.

b) 문제 풀이

먼저 추상 클래스로 정의된 Shape 클래스 중 구현해야 하는 Method는 draw()입니다. 문제에서 필요한 도형은 Line, Rectangle, Circle로 다음과 같이 구체화합니다.

```
//  Line 인 경우
class Line extends Shape {
    public void draw() {
        System.out.println("Line");
    }
}

//  Rect 인 경우
class Rect extends Shape {
    public void draw() {
        System.out.println("Rect");
    }
}

//  Circle 인 경우
class Circle extends Shape {
    public void draw() {
        System.out.println("Circle");
    }
}
```

➔ 동적 바인딩(Dynamic Binding)에 의해 런타임 중 실행될 Method가 결정될 것입니다.

다음은 전체 프로그램을 관장하는 Graphic 클래스를 만듭니다. 이 클래스에서 연결 리스트를 직접적으로 관리하며, 메소드들을 실행합니다. 이 클래스는 main 함수의 코드 길이를 최대한 줄이기 위해 설계되었습니다.

먼저 startProgram()에서 반복문을 통해 프로그램을 계속 실행시킵니다. 조건이 되면 프로그램을 종료합니다.

getMode()에서는 어떤 작업을 수행할 지 선택하는 함수입니다. 삽입, 삭제, 모두 보기, 종료를 여기서 선택할 수 있습니다. Mode를 입력 받고 이에 따른 메소드를 Switch 문을 통해 구현합니다. 특히 4를 입력할 시 false라는 Boolean 변수를 반환함으로써 프로그램을 종료할 수 있습니다.

다음은 새로운 도형을 삽입하는 insertShape()입니다. 먼저 정수를 입력 받고 이에 맞는 도형을 생성합니다. 그 후 삽입될 위치인 맨 마지막 노드를 찾습니다. head가 Dummy Node가 아니고 레퍼런스이기 때문에, head가 null이라면 새로운 원소를 head에 연결해줍니다. 만일 null이 아니라면, 반복문을 통해 가장 마지막 원소에 도달하여 그 원소의 next에 새로운 원소를 연결해줍니다.

```
//  다음 노드를 찾는다.
if (head == null) { //  head 가 비어있다면,
    head = buff;
} else {
    Shape finder = head;
    while (true) {
        if (finder.getNext() == null) { //  다음 노드의 위치가 비어있는 노드까지
도달
            finder.setNext(buff);
            break;
        }
        finder = finder.getNext();
    }
}
```

다음은 도형을 삭제하는 deleteShape()입니다. 삭제할 위치를 입력 받고 그 위치에 도달하여 삭제하는 기능을 수행합니다. 이때 입력 받은 위치에 찾아가서 삭제하고 연결하기 위해 먼저 이동하는 변수 finder와 finder의 전 상태를 저장하는 buff를 사용했습니다. 만일 위치 정보가 잘못되었다면, 알림을 주고 함수를 종료합니다. 삭제 가능하다면, 크게는 두 가지 상황을 고려해야 합니다.

(1) head가 가리키는 원소가 대상이라면, 그 원소의 다음을 head에 연결한다.
(2) 아니라면, 마지막 원소인지 아닌지 판단해야 합니다. 만일 마지막 원소라면, 그 전 원소의 next에 null을 저장합니다. 아니라면 그 다음 원소를 전 원소에 연결하는 방식으로 구현합니다.

```
//  만일 head 가 대상이라면, buff 의 다음 노드의 주소를 받는다.
if(buff == head){
    head = buff.getNext();
}
else{   //  finder 가 null 이 아닌 경우
    if(finder!=null){
        buff.setNext(finder.getNext());
```

```
        finder.setNext(null);
    }   //  finder 가 null 인 경우(즉 리스트의 끝)
    else buff.setNext(finder);
}
```

마지막으로 drawAll()입니다. 이는 연결 리스트의 원소를 차례로 출력합니다. 이때 Overriding을 통해 구현된 draw()가 사용됩니다. 모든 변수는 업캐스팅 되어있기 때문입니다. 만일 리스트가 비어 있다면 알림을 주고 함수를 종료합니다.

c) 시행 착오

없습니다.

이상입니다.