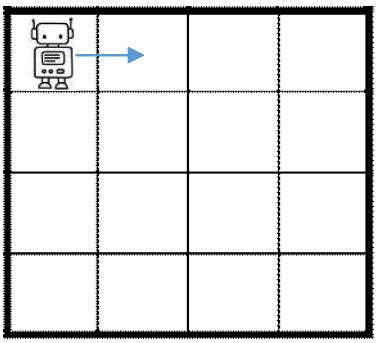
2020년 2학기/객체지향프로그래밍및실습/1차 프로그래밍 과제

1. 문제 정의

1) 개요

다음과 같은 N x N (4<=N<=10) 격자구조의 보드 위에서 로봇을 이동시키는 프로그램을 구현하고자 한다. 보드 위에는 하나 이상의 이동 가능한 로봇들이 놓일 수 있다.



2) 조건

가) 한 칸에는 최대 하나의 로봇이 놓일 수 있다.
나) 로봇은 한 번에 한 칸 이상(<N) 이동할 수 있다.
다) 로봇의 이동은 네 방향(오른쪽, 왼쪽, 위쪽, 아래쪽)으로만 가능하다.
라) 보드 위에 로봇을 처음 만들어 놓을 때, 로봇의 이동 방향은 '오른쪽'이다.
라) 로봇은 좌회전 또는 우회전으로 방향전환을 할 수 있다.
마) 로봇은 이동시 경계선을 만나거나 로봇은 이동하고자 하는 칸에 다른 로봇이 있으면 이동할 수 없다.
사) 보드상의 좌표 번호는 0~N-1까지이다. 예) N=4인 경우, (0,0), (0,1), ..., (3,3)

3) 힌트

이 과제를 여러 개의 작은 단계로 나누어 해결하고자 한다. 그리고 각 단계마다 테스트 코드가 제공된다. 테스트 코드에서 사용되는 클래스 및 메소드는 반드시 구현해야 한다. 하지만 실제 채점에서는 다른 코드를 이용하여 테스트한다는 사실에 주의해야 한다. N값도 범위내에서 변경된다. 뒷 단계를 해결할 수 있으면 앞 단계는 건너뛸 수 있다. 즉, 자신이 해결할 수 있는 마지막 단계에 대한 해답만 제출하면 된다.

2. 단계별 문제

1) 1단계 문제

가) 문제 정의

하나의 로봇을 N x N 보드 위에서 이리저리 이동시키는 프로그램이다. 다른 로봇은 없다고 가정한다. 테스트 코드에 알맞은 Robot, Board, Pos 클래스를 각각 구현해야 한다. 테스트 코드에 사용되는 메소드는 반드시 구현되어야 하며 이 외에도 각 클래스에 필요하다고 판단되는 instance variables과 메소드를 추가할 수 있다.

- move() : 이동이 가능하면 위치를 변경시키고 이동이 가능하지 않으면 위치는 변동이 없으며 오류 메시지를 출력한다.
- getPos() : 현재 위치를 Pos 객체로 return한다.
- setPos() : 현재 위치를 주어진 Pos 객체로 설정한다. 단, 이동방향은 '오른쪽'으로 초기화한다.

나) 테스트 코드

```
// in Test1.java
public static void main(String[] args)
{
    int N=4;
    Board board = new Board(N);          // NxN Board 생성
    Pos p = new Pos(1,2);                // 좌표 (1,2) 객체를 생성
    Robot rb = new Robot(board, "Kiro",  p);   // 좌표 p에 로봇 "Kiro"를 생성
    rb.move(1);                          // 이동방향(여기서는 오른쪽)으로 한 칸 이동
    rb.move(1);                          // 경계선을 지나 이동하므로 오류 메시지 출력
    rb.turnRight();                      // 오른쪽으로 방향전환(여기서는 아래쪽)
    rb.move(2);                          // 이동방향(여기서는 아래쪽) 2칸 이동
    rb.move(1);                          // 경계선을 만나 오류 메시지 출력
    rb.turnLeft();                       // 왼쪽(여기서는 오른쪽)으로 방향전환
    rb.turnLeft();                       // 왼쪽(여기서는 위쪽)으로 방향전환
    rb.move(1);                          // 이동방향(여기서는 위쪽)으로 한칸 이동
    System.out.printf("%s in %s%n", rb.getName(), rb.getPos());   // "Kiro in (2,3)"
    rb.setPos(new Pos(0, 0));            // rb의 위치를 (0, 0)으로 설정
    System.out.printf("%s in %s%n", rb.getName(), rb.getPos());   // "Kiro in (0,0)"
}
```

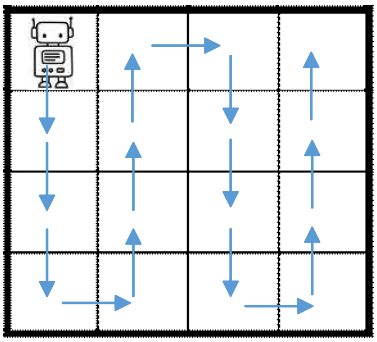
※ 주의사항: Pos 객체를 println의 매개변수로 사용하려면 Pos 클래스에 toString() 메소드를 추가해야 한다.

2) 2단계 문제

가) 문제 정의

1단계 문제를 확장해서 로봇이 아래 그림처럼 (0,0)에서 출발하여 보드의 모든 칸을 한 번씩 방문하는 일을 하는 메소드를 2가지 방식으로 추가한다. 필요하면 추가적인 메소드를 정의하여 사용할 수 있다.

1) Robot내에 travese() method 정의
2) Test2 클래스에 static traverse() method 정의



단, Robot의 move()와 turnRight(), turnLeft() 메소드를 이용하여 로봇을 이동시킨다.

나) 테스트 코드

```
// in Test2.java
public static void main(String[] args)
{
    int N=4;
    Board board = new Board(N);          // NxN Board 생성
    Pos p = new Pos(1,2);                // 좌표 (1,2) 객체를 생성
    Robot rb = new Robot(board, "Kiro", p);     // 좌표 p에 로봇 "Kiro"을 생성
    rb.move(1);                          // 이동방향(여기서는 오른쪽)으로 한 칸 이동
    rb.move(1);                          // 경계선을 지나 이동하므로 오류 메시지 출력
    rb.turnRight();                      // 오른쪽으로 방향전환(여기서는 아래쪽)
    rb.move(2);                          // 이동방향(여기서는 아래쪽) 2칸 이동
    rb.move(1);                          // 경계선을 만나 오류 메시지 출력
    rb.turnLeft();                       // 왼쪽(여기서는 오른쪽)으로 방향전환
    rb.turnLeft();                       // 왼쪽(여기서는 위쪽)으로 방향전환
    rb.move(1);                          // 이동방향(여기서는 위쪽)으로 한칸 이동
    System.out.printf("%s in %s%n", rb.getName(), rb.getPos());   // "Kiro in (2,3)"
    traverse(rb);       // 보드를 위 그림처럼 이동하면서 좌표들을 출력
                        // (0,0 (1,0) (2,0) (3,0) (3,1) (2,1) (1,1) ... (3,2) (3,3) (2,3) (1,3) (0,3)
    System.out.printf("%s in %s%n", rb.getName(), rb.getPos());   // "Kiro in (0,3)"
    rb.traverse();      // 위 traverse(rb)와 동일한 결과
```

```
    System.out.printf("%s in %s%n", rb.getName(), rb.getPos());    // "Kiro in (0,3)"
}

public static void traverse(Robot rb)
{

}
```

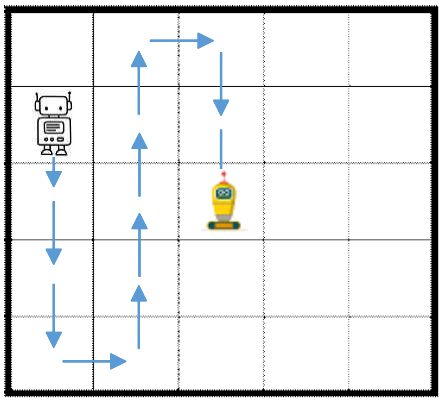
※ 주의사항: traverse()의 실행 시작 시 로봇의 위치를 (0,0)으로 설정한다.

3) 3단계 문제

가) 문제 정의

2개 이상의 로봇이 보드 위에 배치될 수 있도록 2단계 문제를 확장한다. 단, 2개 이상의 로봇이 동일한 위치에는 생성되지 않는다고 가정한다. 테스트 코드에 주어지는 모든 클래스와 메소드를 적절하게 구현해야 한다. move()의 경우 로봇의 이동 경로상 에 다른 로봇이 있으면 이동할 수 없고, 오류메시지만 출력한다. 또한 traverse()를 확장하여 traverse2() 메소드를 추가한다. 이러한 코드들을 구현하기 위해 추가적인 메소드가 필요하다면 추가해도 좋다.

- traverse2() : 로봇은 자신의 현재 위치에서부터 탐방을 시작한다. 단, 다른 로봇의 방해로 이동이 불가능하면 탐방을 종료한다. traverse()와는 다르게 로봇의 현재 위치에서의 이동 방향에 어떠한 가정도 없다. 즉, 탐방을 시작할 때 로봇의 현재 이동 방향과 탐방을 위한 이동 방향을 확인하여 이동해야 한다. 필요하면 각 클래스에 필요한 메소드를 추가할 수 있다. 힌트) 로봇의 현재 이동 방향 상태값이나 N값을 얻어내는 메소드 등.



나) 테스트 코드

```
// in Test3.java
public static void main(String[] args)
{
    int N=5;
```

```
    Board board = new Board(N);          // NxN Board 생성
    Robot rb1 = new Robot("board, "Kiro", new Pos(1,2));  // 좌표 (1,2)에 로봇 생성
    Robot rb2 = new Robot(board, "Miro"); // default 좌표인 (0,0)에 두 번째 로봇 생성
    int count = Robot.count();           // 현재 생성된 robot의 수를 알아냄
    System.out.println("# of robots: "+count); // robot 수 출력

    rb2.move(2);                         // 이동방향(여기서는 오른쪽)으로 한 칸 이동
    rb2.turnRight();                     // 이동방향 전환(여기서는 아래쪽)
    rb2.move(1);                         // rb1이 있어 이동 불가. 오류 메시지 출력
    System.out.printf("%s in %s & %s in %s%n",rb1.getName(),rb1.getPos(),
        rb2.getName(), rb2.getPos());    // Kiro in (1,2) & Miro in (0,2)
    rb1.turnLeft();
    rb1.trunLeft();
    rb1.move(3);                         // 왼쪽 경계선 만나 오류메시지 출력
    rb1.move(2);                         // 왼쪽으로 두 칸 이동 (1,0)
    System.out.printf("%s in %s%n", rb1.getName(), rb1.getPos());   // Kiro in (1,0)
    rb2.move(2);                         // 아래쪽으로 두 칸 이동 (2,2)
    System.out.printf("%s in %s%n", rb2.getName(), rb2.getPos());    // Miro in (2,2)
    rb1.traverse();                      // 2단계 문제와 동일한 방식으로 진행
    System.out.printf("%s in %s%n", rb1.getName(), rb1.getPos());   // Kiro in (4,4)
    rb1.setPos(new Pos(1,0));
    traverse2(rb1);                      // 보드를 위 그림처럼 이동하면서 좌표들을 출력
                        //(1,0) (2,0) (3,0) (4,0) (4,1) (3,1) (2,1) (1,1) (0,1) (0,2) (1,2)
    System.out.printf("%s in %s%n", rb1.getName(), rb1.getPos()); //Kiro in (1,2)
}
public static void traverse2(Robot rb)
{

}
```

※ **주의사항:** traverse2()의 구현시 2단계와 마찬가지로 move()와 turnRight(), turnLeft() 메소드를 이용하여 로봇을 이동시킨다.

3. 제출물

1) Project File (자신이 제출하고 싶은 마지막 단계)
   - project name은 HW1_TestN (N은 제출하는 문제의 단계 번호)
2) 보고서 파일 (pdf)
   가) 표지(과제명, 과제번호, 이름, 학번, 학과, 학년 포함)
   나) 소개 (구현한 부분과 구현하지 못한 부분을 명확하게 명시해야 한다.)
   다) 분석/설계 (UML class diagram과 주요 instance variable 및 메소드에 대한 설명)
   라) 주요 알고리즘을 sequence diagram으로 표현 (메시지 호출의 관점)

마) 실행 결과 화면과 결과에 대한 설명 (다양한 경우를 다루는 테스트 결과)
바) 결론 (이 프로그램에서 배운 점 등과 어려웠던 부분 등에 대해 기술한다. 평서체로 기술보고서 형식으로 작성한다. 반성문과 같은 형식을 지양할 것.)

4. 제출방법

1) 하나의 디렉토리(OOP-HW1)를 만들어 eclipse project를 export한 파일과 보고서 파일을 이 디렉토리에 넣는다.
예) .../OOP-HW1/source_학번 (exported file)
.../OOP-HW1/report_학번.pdf

2) 위 디렉토리를 zip하여 Bb 과제게시판에 올린다.

5. 제출일

- 2020년 10월 10일(토) 23:59PM

★주의사항★ 과제제출 지연은 최대 3일간 주어지며 1일 지연 때마다 취득한 점수 5%씩 감점됨