2020년 2학기
객체지향프로그래밍 및 실습
1차 프로그래밍 과제

HW1_Test3

수학과 201921195

2학년 조우연

[1] 서론

본 보고서는 2020학년 2학기 객체지향프로그래밍 및 실습 과목의 1차 프로그래밍 과제에 대한 보고서로서 N x N (4<=N<=10)격자구조의 보드 위에서 로봇을 이동시키는 프로그램을 구현하고, 해당 프로그램에 대한 내용을 담고 있다. 또한 해당 과제는 단계별로 문제가 정의되어 있으며, 현재 작성되는 보고서는 '3단계 문제'에 대한 보고서이다. 1차 프로그래밍 과제는 기존에 해왔던 구조적 프로그래밍에서 벗어나 객체지향적으로 프로그램을 설계하며, 이에 따라 프로그램을 실행하는데 필요한 class와 각 클래스에 필요하다고 판단되는 instance variables, 메소드 등을 생성할 수 있는 것이 핵심이다.

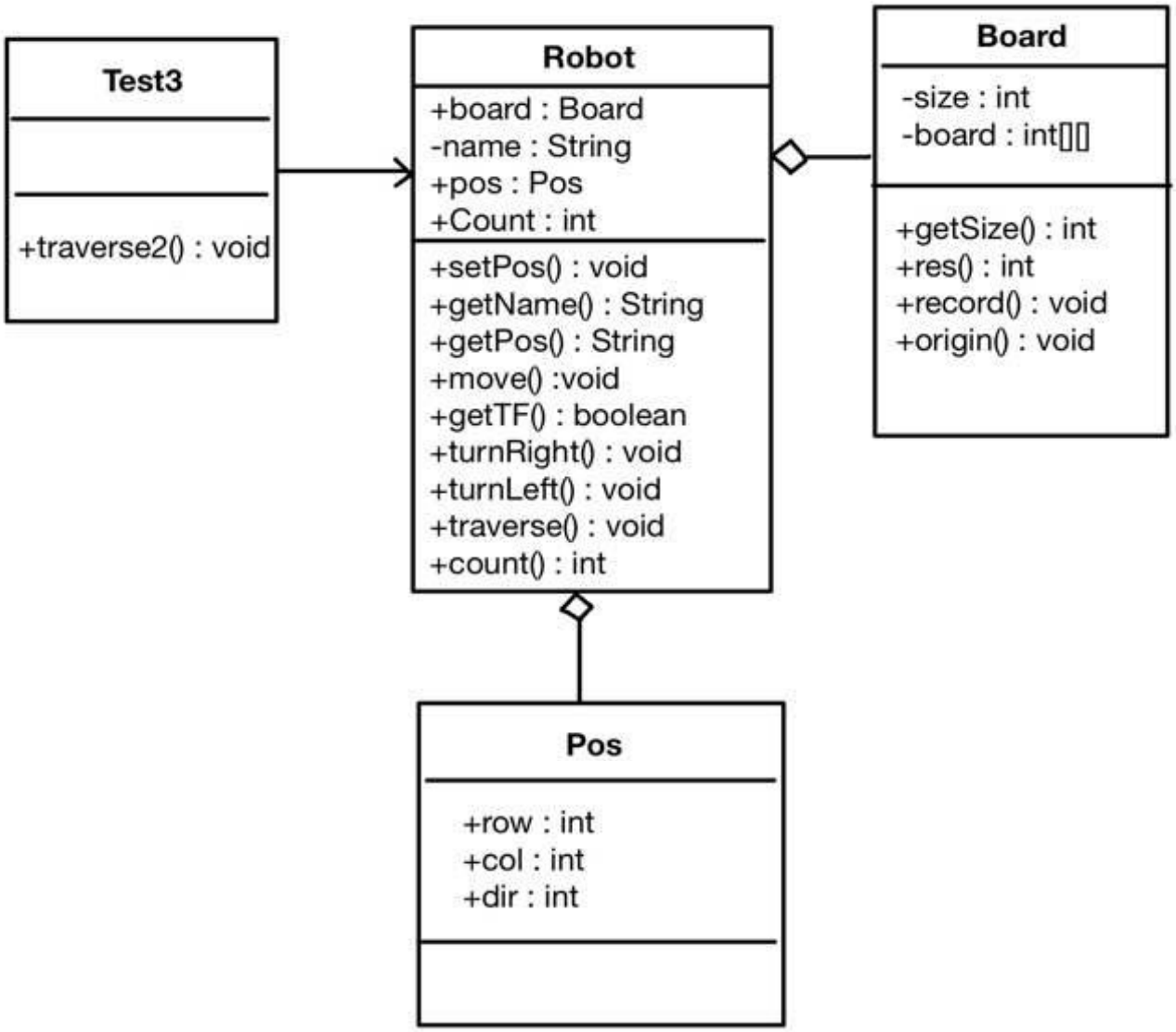
1차 프로그래밍 과제의 '3단계 문제'에 대한 문제요구사항은 다음과 같으며, 문제요구사항에서 언급된 기능은 모두 구현되었음을 명시한다. (단, rb.traverse() 메소드는 '이동하려는 칸에 다른 로봇이 있으면 이동할 수 없다.'라는 조건이 추가되면서 구현하기 어려워 구현하지 못하였다.)

1. 한 칸에는 최대 하나의 로봇이 놓일 수 있다.
2. 로봇은 한번에 한 칸 이상(<N) 이동할 수 있다.
3. 로봇의 이동은 네 방향(오른쪽, 왼쪽, 위쪽, 아래쪽)으로만 가능하다.
4. 로봇은 좌회전 또는 우회전으로 방향전환을 할 수 있다.
5. 로봇은 이동시 경계선을 만나거나, 이동하고자 하는 칸에 다른 로봇이 있으면 이동할 수 없다.
6. 보드상의 좌표 번호는 0~N-1까지이다. 예) N=4인 경우, (0,0),(0,1),(0,2), ...,(3,3)
7. 다음과 같은 메소드는 반드시 구현되어야 한다.
- move() : 이동이 가능하면 위치를 변경시키고, 이동이 가능하지 않으면 위치의 변동이 없으며 오류 메시지를 출력한다.
- getPos() : 현재 위치를 Pos 객체로 return한다.
- setPos() : 현재 위치를 주어진 Pos 객체로 설정한다. 단, 이동방향은 '오른쪽'으로 초기화한다.
- turnRight(), turnLeft() : 각각 로봇의 방향을 오른쪽, 왼쪽으로 전환한다.
- traverse2() : 로봇은 자신의 현재위치에서부터 탐방을 시작한다. 단, 다른 로봇의 방해로 이동이 불가능하면 탐방을 종료한다. 탐방을 시작할 때 로봇의 현재 이동방향과 탐방을 위한 이동방향을 확인하여 이동해야 한다.
- count() : 현재 생성된 robot의 수를 알아낸다.

[2] 본론

[가] 분석/설계

- UML class diagram



- 주요 instance variable 및 메소드 설명

Robot 클래스의 instance variable	Robot 클래스의 메소드
board : Board ->로봇이 위치할 보드에 대한 정보를 저장한다. name : String ->로봇의 이름을 저장한다. pos : Pos ->로봇이 있는 좌표를 저장한다. Count : int ->생성된 로봇의 번호를 저장한다.(생성된 순서대로 1,2,3, ...)	setPos() : void ->좌표를 매개변수로 받아, 로봇의 위치를 넘겨받은 좌표로 pos를 변경한다. 이때 이동하기 전의 좌표의 값은 board의 method인 origin()을 통해 0으로 바꾸고, 이동후의 좌표의 값은 board의 record() 메소드를 통해 1로 바꾼다. getName(), getPos() : String ->각각 name, pos를 문자열 형태로 반환한다. move() : void ->정수 n을 매개변수로 넘겨받아, robot이 바라보는 방향을 따라 n칸 이동한다. 단, robot이 이동하고자 하는 곳이 보드를 벗어나는 구역이거나, 다른 로봇이 길을 막고 있으면 에러메세지를 출력하고 이동하지 않는다. setPos()와 마찬가지로 지나온 자리의 좌표의 값은 board의 method인 origin()을 통해 0으로 바꾸고, 이동후의 좌표의 값은 board의 record() 메소드를 통해 1로 바꾼다. turnRight() : void ->로봇의 방향을 로봇이 현재 바라보는 방향의 오른쪽으로 바꾼다. 즉, 남쪽을 바라보고 있을 때에는 서쪽을 바라보게, 서쪽을 바라보고 있을 때에는 북쪽을 바라보게 바꾼다. turnLeft() : void ->로봇의 방향을 로봇이 현재 바라보는 방향의 왼쪽으로 바꾼다. 즉, 남쪽을 바라보고 있을 때에는 동쪽을 바라보게, 서쪽을 바라보고 있을 때에는 남쪽을 바라보게 바꾼다. count() : int ->로봇이 생성될 때마다 증가한 Count의 값(즉, 생성된 로봇의 수)을 반환한다. traverse() : void ->로봇이 보드의 (0,0)지점부터 문제에 주어진 방향대로 보드의 모든 지점을 이동하며 각 지점의 좌표를 출력한다.

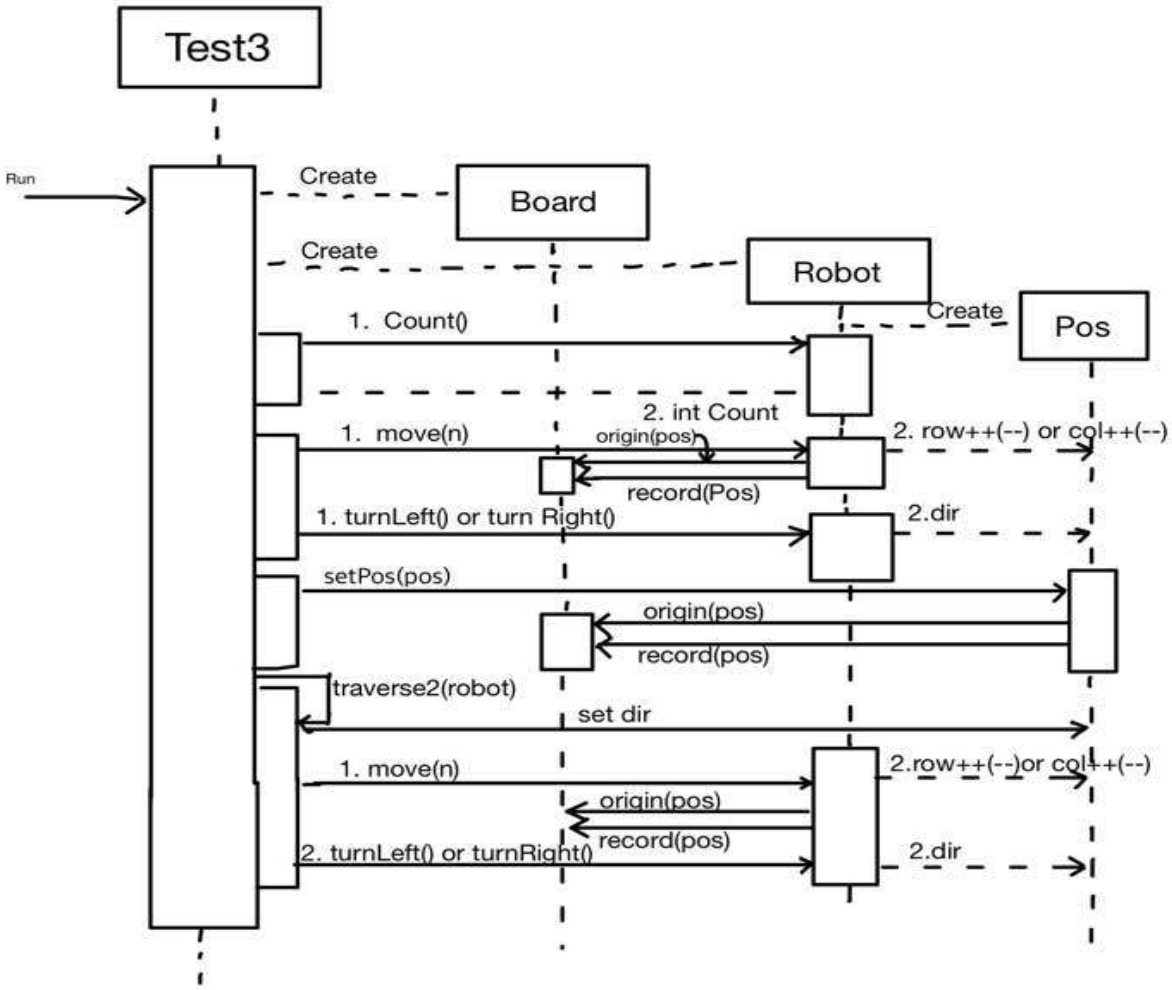
Board 클래스의 instance variable	Board 클래스의 메소드
board : int[][] -> 로봇이 이동할 N x N(4<=N<=10) 격자구조의 보드를 구현할 board. 로봇이 놓인 좌표의 값을 1, 비어있는 좌표의 값을 0으로 설정할 것이다.(따라서 board는 생성당시 모든 값을 0으로 초기화한다.) size : int -> board의 size(N)을 저장하는 변수	getSize() : int -> 보드의 크기(N)을 반환한다. res() : int ->매개변수로 받은 좌표에 있는 값(좌표아님)을 반환한다.(0 or 1) record() : void ->로봇이 있는 좌표의 값을 1로 바꾼다. origin() : void ->로봇이 이동한 자리의 좌표를 다시 0으로 바꾼다.

Pos 클래스의 instance variable	Pos 클래스의 메소드
row : int ->좌표의 row를 저장한다. col : int ->좌표의 column을 저장한다. dir : int ->로봇의 방향을 저장한다. (동쪽은 1, 서쪽은 2, 남쪽은 3, 북쪽은 4)	없음

Test3 클래스의 instance variable	Test3 클래스의 메소드
없음	traverse2() : void ->로봇이 자신의 현재위치에서부터 탐방을 시작해 보드 위를 이동한다. 단, 다른 로봇의 방해로 로봇의 이동이 불가능하다면 탐방을 종료한다. 로봇의 탐방방향은 과제문제에 주어진 그림의 방향대로 한다.

[나] 1차 프로그래밍 과제에 대한 sequence diagram

(프로그램을 사용하거나 따온 것이 아니고, 직접 손으로 그린 것임을 명시합니다.)



[다] 실행결과화면, 결과에 대한 설명

Test3.java의 실행결과
Console ⊠ Problems Debug Shell <terminated> Test3 [Java Application] C:₩Program Files₩Java₩openjdk-11.0.2_windows-x64_bin₩jdk-11.0.2₩bin₩java # of robots : 2 다른로봇이 있어 이동이 어렵습니다. Kiro in (1,2) & Miro in (0,2) 보드를 벗어났습니다. Kiro in (1,0) Miro in (2,2) (1,0) (2,0) (3,0) (4,0) (4,1) (3,1) (2,1) (1,1) (0,1) (0,2) (1,2) 다른로봇이 있어 이동이 어렵습니다. Kiro in (1,2)
5x5보드를 생성 후, Robot클래스로 정의된 rb1,rb2 변수를 만든다. 두 로봇은 Robot

클래스의 constructor인 public Robot(Board b, String n, Pos p)을 통해 각각의 board, name, pos(좌표), count가 (board,"Kiro",(1,2),1), (board,"Miro",(0,0),2)로 저장된다. 두 로봇이 모두 같은 board로 할당되었기 때문에 같은 격자보드 위를 이동할 것이다. Robot 클래스의 static메소드인 count()를 실행해 현재 생성된 로봇의 수를 정수형 변수인 count에 저장하고 로봇의 수를 출력하였다. rb2.move(2)가 실행되며 rb2의 좌표는 (0,2)로 바뀌었고, rb2.turnRight()을 실행하며 rb2의 방향이 아래로 바뀌었다. rb2.move(1)을 실행하여 rb2가 (1,2)로 이동하려고 하지만, (1,2)에는 rb1이 존재하므로 오류메세지를 출력하고 이동하지 않는다. 그 후, rb1과 rb2의 이름과 좌표를 함께 출력하여 “Kiro in (1,2) & Miro in (0,2)”를 출력되었다. rb1.turnLeft()를 연달아 실행해 rb1의 방향이 북쪽으로 먼저 바뀌고, 이어서 서쪽으로 바뀌었다. rb1.move(3)를 실행하였지만 서쪽으로 세칸을 이동하면 보드를 벗어나므로 에러메세지를 출력하고 이동하지 않는다. rb1.move(2)를 실행하여 rb1이 서쪽으로 두칸 이동해 rb1의 좌표가 (1,0)으로 바뀌었다. 이때의 rb1의 이름과 위치를 출력하여 "Kiro in (1,0)"이 출력되어 이어서 rb2.move(2)를 실행해 rb2가 남쪽으로 2칸 이동하고, rb2의 좌표는 (2,2)로 바뀌었다. 이때의 rb2의 이름과 좌표를 출력하여 “Miro in (2,2)"가 출력되고 rb1.setPos(new Pos(1,0))을 실행해 rb1의 좌표가 (1,0)으로 바뀌고, rb1의 방향은 동쪽으로 바뀌었다. 이 상태에서 traverse(rb1)을 실행하여 rb1이 (1,0)부터 (2,0),(3,0),(4,0), ...(1,2)까지 이동하고 (2,2)에는 rb2가 존재하여 다른로봇이 있음을 알리는 메시지를 출력하고 탐방을 중단하였다. 마지막으로 rb1의 이름과 좌표를 출력하면 "Kiro in (1,2)"를 출력한다.

변형한 Test3.java코드 및 실행결과(Test3.java의 변경사항(추가사항)은 아래 코드의 주석으로 달아놓음)

```
public class Test3 {
    public static void main(String[] args)
    {
        int N=6; //보드 크기 6으로 변경
        Board board = new Board(N);
        Robot rb1 = new Robot(board, "Kiro", new Pos(1,2));
        Robot rb2 = new Robot(board, "Miro");
        Robot rb3 = new Robot(board, "Tiro", new Pos(3,3)); //새로운 로봇3 생성
        int count = Robot.count();
        System.out.println("# of robots : "+count);

        rb3.move(1); //rb3 우측으로 2이동, (3,4)
        rb3.move(2); //보드를 벗어남, 에러출력
        System.out.printf("%s in %s%n", rb3.getName(), rb3.getPos());//rb3의 위치출력
        rb2.move(2);
        rb2.turnRight();
        rb2.move(1);
        System.out.printf("%s in %s & %s in %s & %s in %s%n",
                rb1.getName(), rb1.getPos(), rb2.getName(), rb2.getPos(), rb3.getName(),rb3.getPos());//rb3의 위치출력추가
        rb1.turnLeft();
        rb1.turnLeft();
        rb1.move(3);
        rb1.move(2);
        System.out.printf("%s in %s%n", rb1.getName(), rb1.getPos());
        rb2.move(2);
        rb2.turnLeft();//rb2방향 동쪽으로 변경
        rb2.move(3); //rb2가 우측으로 2이동, (2,5)
        System.out.printf("%s in %s%n", rb2.getName(), rb2.getPos());
        rb1.setPos(new Pos(1,0));
        traverse2(rb1);
        System.out.printf("%s in %s%n", rb1.getName(), rb1.getPos());
    }
```

대부분의 실행결과 설명은 위(원래 Test3.java)실행결과 설명과 동일하며, 변경된 코드와 그에 대한 실행결과에 대한 설명은 주석으로 대신하겠습니다.

(위 코드는 Test3.java를 기반으로 만들어진 새로운 Test3.java의 코드로, 로봇의 수가 하나 증가하였고, 보드의 크기가 6으로 변경되었으며, rb2.move(), rb2.turnLeft()메소드를 조금 더 추가하여, traverse2(rb1)이 보드 위를 조금 더 많이 탐방할 수 있게끔 변경되었음.

[3] 결론

1차 프로그래밍 과제를 마치며 객체지향적 프로그래밍을 통해 구현한 클래스, 메소드의 의미와 쓰임새 등을 이해할 수 있었다. 특히 이론만으로는 이해가 가지 않았던 private변수와 public변수, static변수의 (scope에 따른) 쓰임이나 용도를 확실히 알게 되었다. C언어의 구조체와 자바의 class가 거의 동일한 개념이라고 생각했었지만, 다양한 타입의 변수를 한 객체로 정의할 수 있다는 공통점 외에도 클래스는 instance variable을 이용해 메소드를 구현할 수 있다는 것을 알 수 있었고, 이에 따른 encapsulation에 대한 개념도 조금 더 체계적으로 이해할 수 있게 되었다.

객체지향적 프로그래밍의 관점으로 프로그래밍을 하면서 구조적 프로그래밍과의 차이점, 객체지향적 프로그래밍의 장점 등을 확실히 깨닫고 느낄 수 있었지만, 프로그래밍 자체는 아직은 구조적으로 프로그래밍을 하는데 익숙해서 private변수를 사용하는 데 어려움이 있었다. 특히 이 프로그램을 처음 짜기 시작했을 때에는 private변수의 쓰임을 명확히 이해하지 못한 상태에서 시작했기 때문에, 클래스의 constructor를 private변수로 선언하지 못하고 public으로 선언하여 사용하는 등의 미숙함이 있었다. 결과적으로 public변수로 선언해 만든 프로그램이 잘 돌아가고, 뒤늦게 처음부터 다시 수정하기는 어려워서 프로그램을 수정하지는 못했지만, 앞으로의 실습과 과제에 있어서는 private변수로 constructor를 구성하고, 이에 따른 메소드를 적절히 구현하며 프로그램을 설계할 것이다.

객체지향프로그래밍은 encapsulation의 성질을 가지고 있는만큼 책임분배를 통하여 어떤 클래스가 각 태스크를 담당할 것인지를 결정해야하고, 각 클래스들간의 관계를 명확히 해야하는데, 이것이 잘 이루어지지 못했다. 프로그램을 짜기 전에 책임분배, 플로우 차트 등을 통해 클래스들간의 관계를 정하고, 이에 맞게 클래스를 선택해 메소드를 구현하는 등의 설계를 우선적으로 했었어야 했는데, 이를 간과하여 프로그래밍 시 코드가 깔끔하지 못했던 것이 아쉽고, 이러한 이유로 UML class diagram, Sequence diagram으로 표현하는 데에도 큰 어려움을 겪었다. 1차 프로그래밍 과제는 본인의 부족함과 미숙한 부분을 확실히 볼 수 있게 해주었고, 앞으로의 실습과 과제에 있어서 프로그램 설계에 더욱 힘써야함을 깨달았다.