자바를 이용한 온라인 멀티플레이 Reversi 구현

2017116186

임정민

소개

Reversi, 오델로 게임이라고도 불리는 게임을 자바로 구현하였습니다. 게임의 규칙은 아래와 같습니다.

- 처음에 판 가운데에 사각형으로 엇갈리게 배치된 돌 4개를 놓고 시작한다.
- 돌은 반드시 상대방 돌을 양쪽에서 포위하여 뒤집을 수 있는 곳에 놓아야 한다.
- 돌을 뒤집을 곳이 없는 경우에는 차례가 자동적으로 상대방에게 넘어가게 된다.
- 아래와 같은 조건에 의해 양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게 된다.
  - 64개의 돌 모두가 판에 가득 찬 경우 (가장 일반적)
  - 어느 한 쪽이 돌을 모두 뒤집은 경우
  - 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우
- 게임이 끝났을 때 돌이 많이 있는 플레이어가 승자가 된다. 만일 돌의 개수가 같을 경우는 무승부가 된다.

Wikipedia (https://ko.wikipedia.org/wiki/오델로)

게임의 구현을 위해서 우선 게임의 알고리즘을 짠 후 오프라인 플레이를 만들어 게임 알고리즘이 정상적으로 작동하는 지 확인하였습니다. 그 후, 온라인 멀티플레이 기능을 위해 자바의 소켓 프로그래밍을 이용하여 TCP 서버-클라이언트 모델을 구현하였습니다.

코드

Reference.

```
public class Player1 extends Player implements Comparable<Player2>{
    protected Player2 opponent;

    public void setOpponent(Player2 opponent) {
        this.opponent = opponent;
    }
    ....
```

```
public class Player2 extends Player implements Comparable<Player1>{
    protected Player1 opponent;

    public void setOpponent(Player1 opponent) {
        this.opponent = opponent;
    }
    ....
```

Player1 class와 Player2 class에서 reference를 이용하였습니다.

각각의 Player class에서는 게임을 진행하면서 상대의 정보를 계속 수정하여야 합니다. 그때마다 상대의 정보를 받아오는 것보다 reference를 이용해 상대의 정보를 Player의 변수로 저장해놓는 것이 나을 것 같아서 사용하였습니다.

상속

```
public abstract class Player {
    protected String username;
    protected int score = 2;

    public void setName(String name){
        this.username = name;
    }

    public String getName(){
        return this.username;
    }
    ....

public class Player1 extends Player implements Comparable<Player2>{
    ....

public class Player2 extends Player implements Comparable<Player1>{
    ....
```

Player1 class와 Player2 class는 Player class를 상속받은 class입니다.

Player1 class와 Player2 class는 기본적으로는 같은 역할을 합니다. 하지만, 기본적인 setName, getName, getScore와 같은 메소드 외에 Player1과 Player2의 행동을 평가하는 메소드는 서로 조금씩 달라야 합니다. 그래서 중복되는 변수나 함수들은 Player 클래스에 넣어서 상속 받고 차이가 필요한 메소드들은 각각의 class에 따로 선언해주었습니다.

Method Overloading

```
public class Sys {

    public void run(){
        Offline o = new Offline();
        o.playGame();
    }

    public void run(int port){
        Server s = new Server();
        try{
            s.playGame(port);
        }catch(IOException e){
            System.err.println("I/O error");
        }
    }

    public void run(String adr, int port){
        Client c = new Client();
        try{
            c.playGame(adr, port);
        }catch(IOException e){
            System.err.println("I/O error");
        }
    }
}
```

method overloading은 메소드의 리턴타입과 매개변수를 다르게 하여 같은 이름의 method를 여러 개 사용할 수 있게 해주는 기능입니다.

Sys 클래스는 사용자의 입력에 맞는 모드를 실행시켜주는 method를 가지고 있는데 서로 다른 이름으로 method를 선언하는 것 보다 같은 이름으로 method overloading을 사용하는 편이 나을 것 같아서 사용하였습니다.

Method Overriding

```
public abstract class Player {
    ....
    public abstract boolean isPlayable();
    public abstract boolean isPlacable(int y, int x);
    public abstract boolean checkVec(int y, int x, int i);
    public abstract void placePoint(int y, int x);
}
```

```
@Override
public boolean isPlayable(){
    ....
}

@Override
public boolean isPlacable(int y, int x){
    ....
```

method overriding은 상위 클래스에 있는 메소드를 재정의 하여 사용할 수 있는 기술입니다.

Player class를 상속받는 Player1 class와 Player2 class는 모두 player의 행동을 평가하고 실행하는 메소드가 필요합니다. 하지만 두 메소드가 조금씩 다르기 때문에 Player class에서 method를 정의하지 않고 abstract method를 선언하여 자식 클래스인 Player1 class와 Player2 class에서 메소드를 재정의 하도록 하였습니다.

Static 변수

```
public class Map {

    public static int bSize = 8;
    public static int[][] board = new int[Map.bSize+2][Map.bSize+2];
```

```
public final static int[][] vec = {
    {-1, -1}, {-1, 0}, {-1, 1},
    {0, -1},           {0, 1},
    {1, -1},  {1, 0},  {1, 1},
};
```

static 변수는 non-static 변수와 달리 프로그램이 실행될 때 메모리에 할당되어 프로그램이 종료되어야 해제되는 변수입니다.

게임의 보드와 관련된 변수들은 게임에서 여러 개가 필요하지 않기 때문에 Map 클래스의 객체를 생성할 때마다 만들어질 필요 없이 게임이 실행될 때 한번만 할당되면 됩니다. 그래서 관련 변수들을 static으로 선언하였습니다.

Abstract Class

```
public abstract class Player {
    ....
```

```
    public abstract boolean isPlayable();
    public abstract boolean isPlacable(int y, int x);
    public abstract boolean checkVec(int y, int x, int i);
    public abstract void placePoint(int y, int x);
}
```

class에 abstract method가 하나라도 있을 경우, class를 abstract class로 선언해 주어야 합니다. Player class에서 player의 행동을 체크하는 메소드들은 Player마다 조금씩 다르므로 각각의 클래스에서 정의 해주었습니다. 그리고 Player class에서 해당 메소드들을 추상 메소드로 정의해서 하위 클래스에서의 통일성을 유지하도록 하였습니다.

Interface

```
public class Player2 extends Player implements Comparable<Player1>{
 ....
public class Player1 extends Player implements Comparable<Player2>{
 ....
```

Player1 class와 Player2 class에 comparable 인터페이스를 implement 해서 서로 다른 클래스인 Player1과 Player2 사이에 점수를 비교하여 정렬할 수 있도록 하였습니다.

패키지

src

Week8.java

- player

  Player.java

  Player1.java

  Player2.java

- sys

  Sys.java

  Start.java

  Offline.java

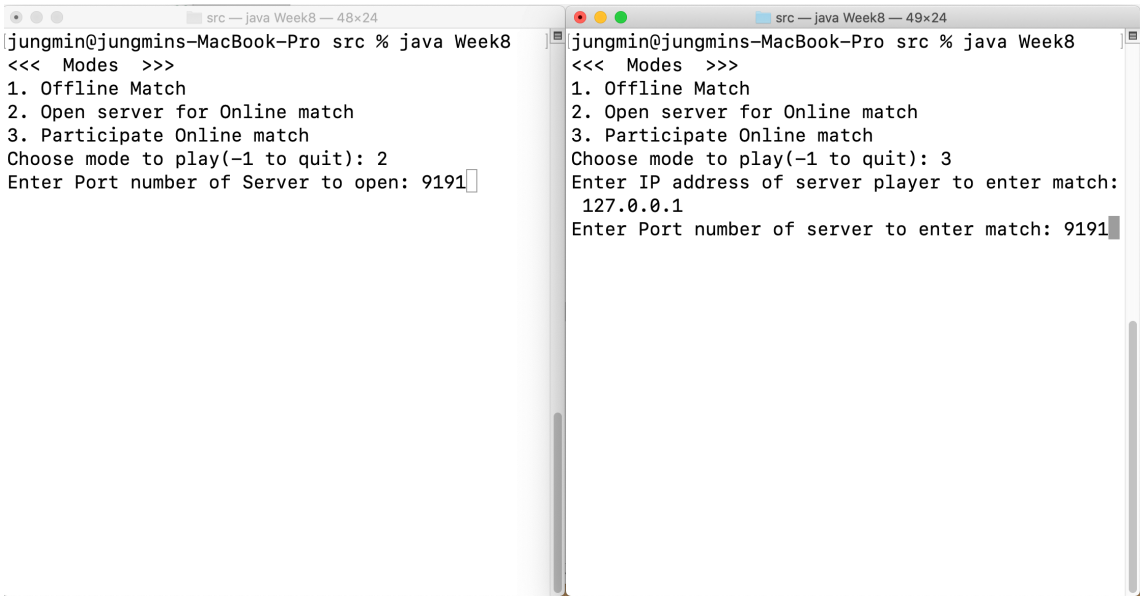
  Server.java

  Client.java

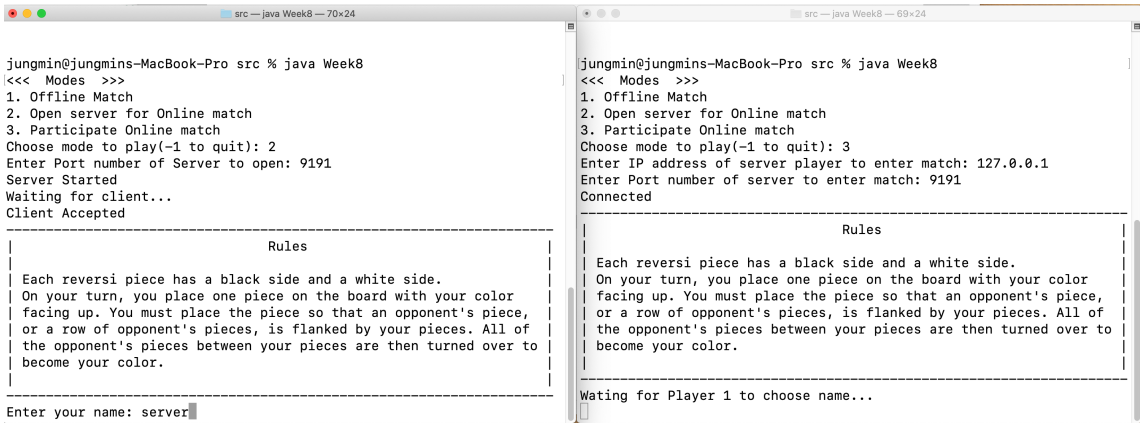
  Map.java

player 패키지와 sys 패키지를 만들어서 플레이어의 행동을 체크하는 클래스들은 player 패키지로, 게임 시스템과 관련된 클래스 들은 sys 패키지로 분류하여 정리할 수 있었습니다.

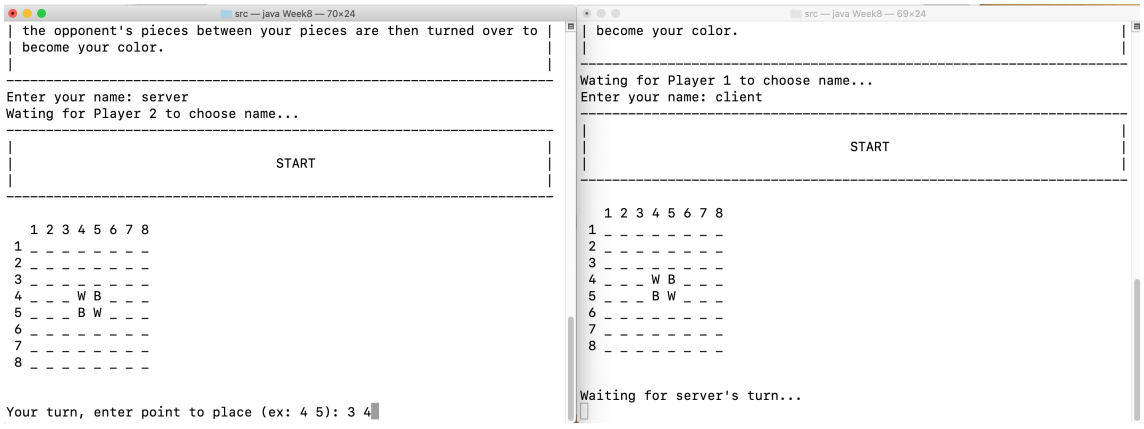
입출력 예시



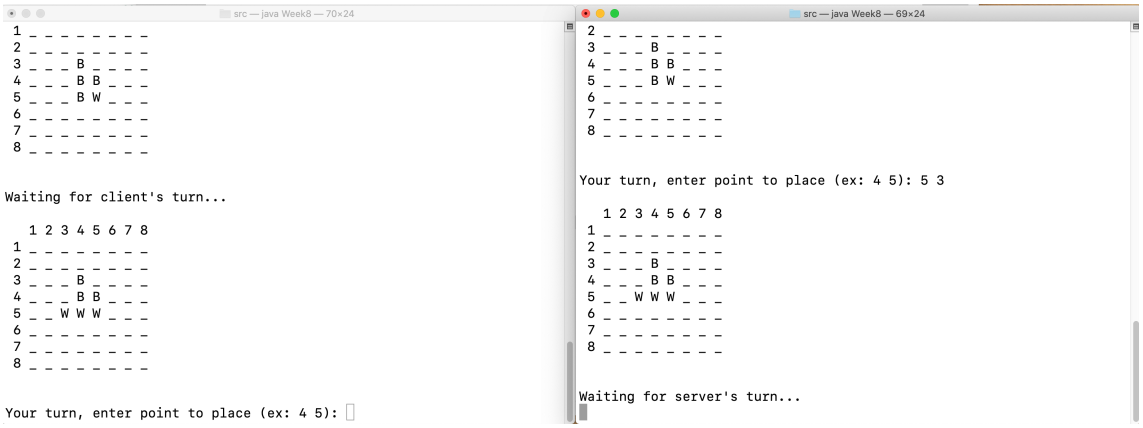
플레이하려고 하는 모드를 선택하고 접속하거나 열 서버 정보를 입력합니다



접속이 성공하면 플레이 할 이름을 입력합니다



게임이 시작되면 서버측에서 먼저 차례를 시작합니다.



자세한 플레이 영상은 sample 폴더에 있습니다