<ExtendsProblem 복습>

[ExtendsProblem (main) - DiceGame - GameManager - Player - Dice]
(Comparable)

GameManager - Player : 상속 관계
GameManager에 Comparable 인터페이스 구현

!) 재활용한 코드인 문제은행[3] 8번 문제와 다르게
1. GameManager 클래스가 Player에게 상속을 받는다.
2. Comparable 인터페이스가 추가되었다.
   Comparable에서 제공하는 인터페이스는 GameManager 클래스에서 구현해 사용한다.
3. 이전에는 GameManager가 전체 플레이어를 관리했지만, 이번 코드에서
   GameManager 객체는 한 명의 플레이어만을 담당한다.
   그렇기 때문에 플레이어 수만큼의 GameManager 객체가 생겨나게 되고,
   Player클래스를 상속받은 GameManager 클래스는 주사위 굴리기, 주사위 합 구하기,
   특수주사위 발동 조건 확인하기, 특수주사위 굴리기 등의 기능을 처리한다.
   그리고 플레이어는 여러 명이므로 각 GameManager의 주사위값에 대한 처리와
   승부판정 등의 게임의 전반적인 운영을 할 수 있는 DiceGame 클래스가 필요하게 된다.
   - DiceGame: 전반적인 게임 관리
   - GameManager: 담당한 플레이어가 주사위를 굴리도록 하고, 주사위값 관리

- ExtendsProblem (main)

```
DiceGame dg = new DiceGame();

dg.startGame();
// 현재 모든 내용물을 toString()을 통해 간헐적으로 살펴보고 있음
// 더 좋은 방법은 없을까 ???
// 여기에 오늘 학습한 compareTo를 활용해보는 방식은 어떨까 ?
System.out.println(dg);

// 실제로 게임의 승패 판정을 어떻게 할 것인가 ?
// dg.getResult() 형식으로 처리하면 좋을 것이다.
// 여기서 주의해야할 사항은 dg 객체 내에는 GameManager 배열이 들어 있다는 것이다.
// 그러므로 여기서 즉각적인 비교가 가능하다!
dg.printResult();
```

DiceGame 타입의 객체 dg를 생성한다.
객체 dg의 startGame 메소드를 호출한다.
toString으로 맵핑된 dg 객체의 정보를 출력한다.
객체 dg의 printResult 메소드를 호출한다.

- DiceGame 클래스

1. 변수 선언

```
final int PLAYER_NUM = 2;
final int DICE_NUM = 2;

final int DEATH_FLAG = 4000;

boolean[] deathId;

private GameManager[] gmArr;
```

플레이어의 수를 의미하는 PLAYER_NUM = 2
사용할 주사위의 수를 의미하는 DICE_NUM = 2
특수주사위 값이 4가 나올 경우 더해줄 값 4000
특수주사위 값 4로 인해 패배한 플레이어가 있는지 확인하는 배열 deathID
GameManager 타입의 gmArr 배열 선언

2. 생성자

```
public DiceGame () {
    // gmArr를 지역변수로 만들었음
    // 그렇기 때문에 생성자 호출이 끝난 시점에서 변수가 증발함
    // GameManager[] gmArr = new GameManager[PLAYER_NUM]; <<<--- 잘못된 코드
    gmArr = new GameManager[PLAYER_NUM];
    deathId = new boolean[PLAYER_NUM];

    for (int id = 0; id < PLAYER_NUM; id++) {
        gmArr[id] = new GameManager(id, DICE_NUM);
    }
}
```

main에서 이 생성자가 호출되면
PLAYER_NUM의 크기를 가지는 gmArr 배열 생성
PLAYER_NUM의 크기를 가지는 deathId 배열 생성

for문을 통해 PLAYER_NUM만큼 반복 수행
반복문 내부에서 gmArr 배열의 각 인덱스에 플레이어를 관리하는
GameManager 타입의 객체 생성
+ GameManager 객체의 생성자 호출 시 넘겨주는 값
: 반복문을 제어하는 변수인 id, 주사위 개수인 DICE_NUM
// 플레이어가 2명이므로 이들을 관리하는 gameManger의 객체도 2개가 생성된다.
반복문을 제어하는 변수인 id를 주사위 값 비교, 승패 판정 등에 활용한다.

3. startGame

```
public void startGame () {
    for (int id = 0; id < PLAYER_NUM; id++) {
        gmArr[id].rollEveryDice();
        gmArr[id].checkSpecialDice();
        gmArr[id].rollSpecialDice();
        // gmArr[id].applySkillEffect(id); <<<--- 여기 넣는 방식이면 이렇게 처리하면됨
    }

    // 현재는 재활용 관점에서 접근해서 아래와 같이 활용하였다.
    // 마음에 들지 않는 부분이라면
    // 현재 구조에서는 for 문 밖으로 빠져 나왔는데 applySkillEffect에
    // 여러가지 기능들이 결합되어 있어 아래와 같이 밑으로 빼야하는 상황이다.
    // 이 구조를 조금 더 분리하여 만들었으면 좀 더 예쁜 코드를 만들 수 있었을 것이다.
    applySkillEffect();
}
```

for문을 통해 PLAYER_NUM만큼 반복을 수행한다.
반복문 내부에서
1. gmArr 배열에 생성된 각 객체의 rollEveryDice 메소드 호출
2. gmArr 배열에 생성된 각 객체의 checkSpecialDice 메소드 호출
3. gmArr 배열에 생성된 각 객체의 rollSpecialDice 메소드 호출
DiceGame 클래스의 applySkillEffect 메소드를 호출한다.

GameManager 클래스는 Player 클래스로부터 상속을 받기 때문에
Player 클래스에 있는 메소드들을 GameManager 객체에서 사용할 수 있다.

4. applySkillEffect

!) 기존 코드와 다르게 DiceGame 클래스에서 선언된 gmArr 배열의 각 객체들의 주사위값을 처리하기 위해 applySkillEffect 메소드가 DiceGame 클래스에 위치해있다.

```
public void applySkillEffect () {
    int tmp;
    for (int i = 0; i < PLAYER_NUM; i++) {
        if (gmArr[i].isGetSpecial()) {
            switch (tmp = gmArr[i].getSpecialDiceNum()) {
                case 1:
                    // 1번의 경우 상대방의 주사위 눈금을 2 떨군다.
                    System.out.println("1번 - 상대 눈금을 2떨굼");
                    for (int j = 0; j < PLAYER_NUM; j++) {
                        if (i == j) {
                            continue;
                        }

                        gmArr[j].operateDice(-2);
                    }
                    break;
                case 3:
                    // 3번의 경우 다 같이 -6을 적용한다.
                    System.out.println("3번 - 다같이 6떨굼");
                    for (int j = 0; j < PLAYER_NUM; j++) {
                        gmArr[j].operateDice(-6);
                    }
                    break;
                case 4:
                    // 4번의 경우 그냥 패배
                    System.out.println("4번 - 잘가");
                    gmArr[i].operateDice(4444);
                    break;
                case 6:
                    // 6번의 경우 모든 상대방에게 3을 뺏어서 내거에 3을 더한다.
                    System.out.println("6번 - 상대편 3을 뺏어서 내쪽으로 3을 땡겨옴");
                    for (int j = 0; j < PLAYER_NUM; j++) {
                        if (i == j) {
                            continue;
                        }

                        gmArr[j].operateDice(-3);
                        gmArr[i].operateDice(3);
                    }
                    break;
                default:
                    System.out.println("디폴트 2, 5!!!");
                    gmArr[i].operateDice(tmp);
                    break;
            }
        }
    }
```

5. printResult

```
public void printResult() {
    // deathId 부분에서 누가 죽었는지를 알고 있으므로
    // 이를 기반으로 검사를 진행하면 된다.
    checkDeath();
    settleResult();
}
```

DiceGame 클래스의 checkDeath 메소드를 호출한다.
DiceGame 클래스의 settleResult 메소드를 호출한다.

6. checkDeath

```
public void checkDeath () {
    for (int id = 0; id < PLAYER_NUM; id++) {
        if (gmArr[id].getSum() > DEATH_FLAG) {
            deathId[id] = true;
        }
    }
}
```

for문을 통해 PLAYER_NUM만큼 반복을 수행한다.
이 for문에서도 제어변수로 생성자의 for문에서 객체 생성 시 사용했던 제어변수인 id를 사용해 gmArr의 변수와 순서를 일치시킨다.
반복문 내부에서 gmArr의 배열에 할당된 각 객체의 getSum 메소드를 호출하고
메소드의 리턴값과 DEATH_FLAG와의 크기를 비교한다.
객체의 getSum 리턴값이 DEATH_FLAG보다 크다면 특수주사위 4가 발동된 것이므로
현재 객체를 의미하는 death[id] 의 값을 true로 바꾼다.

7. settleResult

```java
public void settleResult () {
    boolean death = false;

    for (int id = 0; id < PLAYER_NUM; id++) {
        if (deathId[id] == true) {
            System.out.printf("플레이어 %d가 패배하였습니다!\n", id);
            death = true;
        }
    }

    if (!death) {
        int res = gmArr[0].compareTo(gmArr[1]);

        if (res > 0) {
            System.out.println("플레이어 0 승리!");
        } else if (res < 0) {
            System.out.println("플레이어 1 승리");
        } else {
            System.out.println("무승부");
        }
    }
}
```

주사위합을 비교하기 전 특수주사위 4로 인해 패배한 플레이어가 있는지 확인이 필요하다.
확인을 위한 값으로 boolean형 변수 death를 선언하고 false로 초기화한다.
// 주사위 합 비교절차를 수행할지 결정하는 플래그

for문을 통해 PLAYER_NUM만큼 반복을 수행한다.
반복문 내부에서는 deathId 배열에 true값이 있는지 검사를 수행한다.
true 값이 있다면 특수주사위 4가 발동된 것이므로 death 값을 true로 바꿔
메소드를 종료시킨다.

특수주사위 4가 발동되지 않았을 때
gmArr 배열의 각 객체들간의 값 비교를 위해 GameManger 클래스에
구현된 인터페이스를 사용한다.
gmArr 배열의 각 객체들간의 값 비교 결과를 res에 대입한다.
if문을 통해 res값을 0과 비교한다.
res가 0보다 크다면 플레이어 0이 승리한다.
res가 0보다 작다면 플레이어 1이 승리한다.
그 외의 경우 무승부

- GameManager 클래스

1. 클래스와 변수 선언

```
public class GameManager extends Player implements Comparable {
private int playerId;
```

GameManager 클래스에서 Player의 정보를 상속 받고,
Comparable 인터페이스를 구현한다.
플레이어 식별을 위한 변수 playerId

2. 생성자

```
public GameManager (final int playerId, final int diceCnt) {
    super(diceCnt);

    System.out.printf("GameManager(): playerId - %d, diceCnt - %d\n", playerId, diceCnt);

    this.playerId = playerId;
}
```

new 연산자로 GameManager 객체를 생성하면, GameManager 클래스의 객체가 메모리에 올라갈 때 부모인 Player 클래스도 함께 메모리에 올라간다.
super는 부모를 가리키는 키워드, super()는 부모의 생성자를 의미한다.
super 키워드는 생성자뿐만 아니라 부모의 메소드나 필드를 사용할 때도
-> diceCnt값을 Player 클래스에 넘겨주며 Player의 생성자를 호출한다.
DiceGame 클래스에서 넘어오는 PLAYER_NUM을 GameManager 클래스의 playerId에 대입

3. compareTo

```
@Override
public int compareTo(Object otherObject) {
    GameManager other = (GameManager) otherObject;

    if (this.getSum() < other.getSum()) {
        // getPlayerId() 같은것을 만들면
        // 사용자가 여러명이여도 아래의 루틴을 처리할 수 있게 된다.
        // playerId <<<--- 이 사용자가 누구인지 판별할 수 있도록 id값 부여함
        System.out.println("id(0) 보다 id(1)이 크다.");
        return -1;
    } else if (this.getSum() > other.getSum()) {
        System.out.println("id(0) 보다 id(1)이 작다.");
        return 1;
    } else {
        System.out.println("id(0) 과 id(1)이 같다.");
        return 0;
    }
}
```

comparable 인터페이스를 가져와 구현하기 위해 인터페이스에 작성되어있는 프로토타입을 가져와 기능을 작성한다.

+ 매서드(함수)의 프로토타입 : 함수의 리턴 타입, 이름, 파라메터(입력 인자)만 있는 경우

- **Comparable 인터페이스**

```
public interface Comparable {
    // Object는 자바에 존재하는 집합체중 가장 거대함
    // 그러므로 모든 원소를 포함 관계에 놓을 수 있다.
    // 즉 어떤 타입으로든 타입 캐스팅을 할 수 있다는 뜻
    // (int) Math.random() <<<< 여기서 (int)가 타입 캐스팅
    int compareTo(Object other);
```

- Comparable 인터페이스에서 작성한 매서드 프로토타입

int compareTo(Object other);

Object는 자바에 존재하는 집합체중 가장 거대하므로 모든 원소를 포함 관계에 놓을 수 있다 (어떤 타입으로든 타입 캐스팅을 할 수 있다)

- GamaManager 클래스에서 인터페이스 구현

```
public int compareTo(Object otherObject) {
    GameManager other = (GameManager) otherObject;

    if (this.getSum() < other.getSum()) {
        // getPlayerId() 같은것을 만들면
        // 사용자가 여러명이여도 아래의 루틴을 처리할 수 있게 된다.
        // playerId <<<--- 이 사용자가 누구인지 판별할 수 있도록 id값 부여함
        System.out.println("id(0) 보다 id(1)이 크다.");
        return -1;
    } else if (this.getSum() > other.getSum()) {
        System.out.println("id(0) 보다 id(1)이 작다.");
        return 1;
    } else {
        System.out.println("id(0) 과 id(1)이 같다.");
        return 0;
    }
}
```

gmArr 배열의 인덱스는 0, 1 이고 각 인덱스에 GameManager 타입의 객체가 생성되어있다.
인덱스는 id를 의미한다.
생성된 두 객체를 this와 other로 구분한다.
현재 객체에서 getSum 메소드를 호출했을 때 리턴값과, other 객체에서 getSum 메소드를 호출했을 때의 리턴값을 비교한다.

현재 객체의 getSum 리턴값이 더 작다면 -1을 리턴해 현재 객체값이 other 객체값보다 더 작다고 판별한다.
현재 객체의 getSum 리턴값이 더 작다면 1을 리턴해 현재 객체값이 other 객체값보다 더 크다고 판별한다.
그 외의 경우 두 객체의 값이 같은 경우이므로 0을 리턴해 무승부임을 판별한다.

4. toString

```
@Override
public String toString() {
    return "GameManager{" + '\n' +
            "player=" + super.toString() + '\n' +
            "playerId=" + playerId +
            '}' + '\n';
}
```

GameManager 클래스와 부모 클래스인 Player 클래스 사용자를 식별할 수 있는 playerId를 맵핑시킨다.

- Player 클래스

: GameManager 클래스의 부모 클래스로 여기에 선언된 변수와 메소드들을 GameManager 객체에서 사용할 수 있다.

1. 변수 선언

```
private int diceCnt;
private Dice[] diceArr;
private Dice special;

private int sum;
private boolean getSpecial;
```

2. 생성자

```java
public Player(final int diceCnt) {
    System.out.printf("Player(): diceCnt - %d\n", diceCnt);

    this.diceCnt = diceCnt;
    diceArr = new Dice[diceCnt];

    for (int i = 0; i < diceCnt; i++) {
        diceArr[i] = new Dice();
    }

    special = new Dice();
}
```

3. rollEveryDice

```java
public void rollEveryDice () {
    sum = 0;

    System.out.println("Player::rollEveryDice()");

    for (int i = 0; i < diceCnt; i++) {
        diceArr[i].rollDice();
        sum += diceArr[i].getDiceNum();
    }
}
```

4. checkSpecialDice

```java
public boolean checkSpecialDice () {
    if (sum % 2 == 0) {
        getSpecial = true;
        return getSpecial;  // <<<--- 이거 정말 필요했던건가요 ?
    } else {
        getSpecial = false;
        return getSpecial;
    }
}
```

5. rollSpecialDice

```
public void rollSpecialDice () {
    if (getSpecial) {
        special.rollDice();
        //special.setDiceNum(4);  // 다 같이 죽는 경우 테스트를 위해서
    }
}
```

6. isGetSpecial

```
public boolean isGetSpecial() { return getSpecial; }
```

7. getSpecialDiceNum

```
public int getSpecialDiceNum () { return special.getDiceNum(); }
```

8. operateDice

```
public void operateDice (int num) {
    sum += num;

    if (sum < 0) {
        sum = 0;
    }
}
```

9. getSum

```
public int getSum () { return sum; }
```

10. toString

```
@Override
public String toString() {
    return "Player{" +
            "diceCnt=" + diceCnt +
            ", diceArr=" + Arrays.toString(diceArr) +
            ", special=" + special +
            ", sum=" + sum +
            ", getSpecial=" + getSpecial +
            '}';
}
```

- Dice 클래스

1. 변수 선언

```
final int MAX = 6;
final int MIN = 1;

int range;
int diceNum;
```

주사위 값의 범위 (1 ~ 6)을 지정하기 위한 MAX, MIN, range
주사위 값을 의미하는 diceNum

2. 생성자

```
public Dice () {
    //System.out.println("나는 Dice 클래스의 기본 생성자!");
    range = MAX - MIN + 1;
}
```

Dice 타입의 객체가 생성되면 해당 객체의 주사위 값의 범위를 계산한다.

3. rollDice

```
public void rollDice () { diceNum = (int) (Math.random() * range + MIN); }
```

1~6 사이의 난수를 발생시켜 주사위 값인 diceNum에 대입한다.

4. getDIceNum

```
public int getDiceNum () { return diceNum; }
```

rollDice 메소드를 통해 구해진 diceNum을 Player 클래스에 반환한다.

```
Player(): diceCnt - 2
GameManager(): playerId - 0, diceCnt - 2
Player(): diceCnt - 2
GameManager(): playerId - 1, diceCnt - 2
Player::rollEveryDice()
Player::rollEveryDice()
1번 - 상대 눈금을 2떨굼
DiceGame{PLAYER_NUM=2, DICE_NUM=2
, gmArr=[GameManager{
player=Player{diceCnt=2, diceArr=[Dice{diceNum=3}, Dice{diceNum=6}], special=Dice{diceNum=0}, sum=7, getSpecial=false}
playerId=0}
, GameManager{
player=Player{diceCnt=2, diceArr=[Dice{diceNum=1}, Dice{diceNum=5}], special=Dice{diceNum=1}, sum=6, getSpecial=true}
playerId=1}
]}
id(0) 보다 id(1)이 작다.
플레이어 0 승리!
```