자료구조 실습 보고서

[제 05 주] 별의 집합

제출일	2017/04/10
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 집합(Set)의 개념 이해
    - ◆ 집합 내의 원소는 정렬되어 있지 않으며 중복이 없어야 한다.
  - 입력
    - ◆ 별의 X 좌표, Y 좌표, 이름
      - i 좌표의 자료형 : int 형
      - ii 이름의 자료형 : String 형
    - ◆ 주어진 별 삭제
      - i 삭제할 별의 이름
    - ◆ 임의의 별 삭제
      - i 집합 내의 임의의 별 삭제
  - 출력
    - ◆ 전체 별의 개수 출력
    - ◆ 이름으로 검색 시
      - i 검색할 별의 이름 입력 후 존재 여부 출력
    - ◆ 좌표로 검색 시
      - i 검색할 좌표값(x, y)를 입력 후 해당 좌표 내의 존재 여부 확인
  - 9 가 입력되면 별의 입력이 종료 된 것으로 간주 후 별이 몇개나 존재하는지 출력

- 자료구조
  - 정렬되지 않으며 중복이 없는 집합(Set) 구조
  - 별을 담는 무작위 1 차원 배열
    - ◆ 배열 내부 삽입 시 별의 이름 혹은 좌표 값으로 존재 여부를 확인하여 삽입
  - 별을 담는 Node
  - Node 가 다른 노드를 연결하며 노드가 추가 될 때마다 크기의 제한없이 계속 증가하는 연결된 체인 형태의 구조(Linked Chain)
    - ◆ 노드 내부 삽입 시 별의 이름 혹은 좌표 값으로 존재 여부를 확인하여 삽입
  - 메뉴, 메뉴 종료, 각 자료구조 실행 시에 실행을 알리고, 잘못된 메뉴 접근 시 경고를 알리게 해주는 enum

2 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	run()	없음	없음	별의 집합 프로그램을 실행시키는 메소드
	showMessage(MessageID aMessageID)	에러 메세지 혹은 알림	없음	에러 메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void inputStar()	없음	없음	집합 내에 별의 존재여부를 체크하여 존재하지 않을 때에 별을 추가
	void searchByCoordinate()	없음	없음	좌표값을 통해 별이 존재하는지 검색을 하여

				존재여부를 출력하게 하는 메소드
	void searchByName()	없음	없음	이름을 통해 해당 이름을 가진 별이 존재하는지 검색하여 존재 여부를 출력하게 하는 메소드
	void remove()	없음	없음	삭제할 별을 입력 받아 해당 별이 존재하는지 확인후, 삭제된 별의 정보를 출력하게 하는 메소드

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	int inputInt()	없음	정수값	입력 받은 정수를 리턴하는 메소드
	String inputString()	없음	문자열값	입력 받은 문자열을 리턴하는 메소드
	void outputMessage(String aMessage)	문자열	없음	전달 받은 문자열을 출력하는 메소드
	void outputStar(Star aStarName, int aX, int aY)	별의 이름, x 좌표값, y 좌표값	없음	전달 받은 별의 이름, 좌표값(x, y)를 출력하는 메소드
	void outputStarExistence(String aStarName, int aX, int aY	별의 이름, x 좌표값, y 좌표값	없음	별이 집합 안에 존재 했을 때 존재한다는 메세지를 이름 혹은 좌표값으로 출력하는 메소드
	void outputNumberOfStars(int aStarCollectorSize)	집합 내부의 별의 총 개수	없음	집합 내부의 별의 총 개수를 전달받아 출력하는 메소드

Class	Star			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Star(int givenX, int givenY)	별의 좌표값(x, y)	없음	주어지는 값으로 별의 좌표를 설정하고 이름은 null 로 초기화하는 설정자 메소드
	Star(String givenStarName)	별의 이름	없음	주어지는 문자열로 별의 이름을 설정하고 좌표를 (0,0)으로 초기화하는 설정자 메소드
	Star(int givenX, int givenY, String givenStarName)	별의 좌표값(x, y), 별의 이름	없음	주어지는 값으로 별의 이름과 좌표를 초기화하는 설정자 메소드
	int xCoordinate()	없음	정수형태의 x 좌표값	별의 x 좌표값 반환
	int yCoordiate()	없음	정수형태의 y 좌표값	별의 y 좌표값 반환
	String starName()	없음	문자열 형태의 별의 이름	별의 이름 반환

	void setXCoordinate(int newX)	정수형태의 x 좌표값	없음	별의 x 좌표값 설정
	void setYCoordinate(int newY)	정수형태의 y 좌표값	없음	별의 y 좌표값 설정
	void setStarName(String newStarName)	문자열 형태의 별의 이름	없음	별의 이름 설정
	boolean equals(Object object)	오브젝트	입력받은 오브젝트의 이름이나 좌표값이 존재하면 true, 아니면 false	입력받은 오브젝트의 좌표값이나 이름을 현재 오브젝트와 비교하여 집합 내부에 동일한 좌표나 이름을 가진 별이 존재하는지 검색하는 메소드

Class	Node<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Node()	없음	없음	노드를 초기화 시키는 생성자 메소드
	Node(E givenElement)	저장하려는 원소값	없음	주어진 원소를 해당 원소값의 변수에 저장하는 생성자 메소드
	Node(E givenElement, Node<E> givenNext)	저장하려는 원소값, 노드	없음	주어진 원소값을 입력하고, 다음 노드를 알고 있는 노드를 생성하는 생성자 메소드
	E element()	없음	특정 타입	노드의 원소값 리턴
	Node<E> next()	없음	노드	현재 노드의 다음 노드값을 리턴
	void setStar(E anElement)	없음	없음	원소값 새롭게 설정하는 메소드
	void setNext(Node<E> newNext)	없음	없음	다음 노드값을 새롭게 설정

Class	ArraySet<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	ArratSet()	없음	없음	집합을 기본값으로 초기화하는 생성자 메소드
	ArraySet(int givenMaxSize)	원소가 담길 배열의 최대값	없음	주어진 값으로 배열의 크기를 정하는 생성자 메소드
	int size()	없음	집합 내부에 있는 원소의 갯수	집합 내부에 있는 원소의 갯수 리턴
	boolean isEmpty()	없음	집합 내부가 비었으면 true 안 비었으면 false	집합 내부가 비어 있는지 여부 리턴
	boolean isFull()	없음	집합 내부가 꽉 차있으면 true 꽉 차있지 않으면 false	집합 내부가 꽉 차있는지의 여부 리턴
	boolean doesContain(E anElement)	특정 타입의 원소	존재하면 true 없으면 false	집합 내부에 입력받은 특정 타입의 원소가 존재하는지 여부를 검사하여 리턴

	boolean add(E anElement)	특정 타입의 원소	true	집합 내부가 꽉 차 있지 않으면서, 같은 원소가 존재하지 않으면 배열에 추가
	E remove(E anElement)	특정 타입의 원소	삭제된 특정 타입의 원소	입력 받은 특정 타입의 원소가 존재하는지 확인하고, 존재한다면 삭제하고 배열을 한 칸 씩 당기는 메소드. 삭제된 원소의 정보를 전달하기 위해 삭제된 원소를 리턴 한다.
	E removeAny()	없음	삭제된 특정 타입의 원소	배열 내에서 임의의 원소를 삭제한다.
	void clear()	없음	없음	집합 내부를 전부 삭제한다.

Class	LinkedSet<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	SortedArrayBag()	없음	없음	집합을 기본값으로 초기화하는 생성자 메소드
	int size()	없음	집합 안에 들어간 별의 갯수 리턴	집합 내부의 별의 갯수 리턴
	boolean isEmpty()	없음	집합이 비어 있으면 true 비어있지 않으면 false	집합이 비어 있는지 검사
	boolean doesContain(E anElement)	특정 타입의 원소	존재하면 true 없으면 false	집합 내부에 입력받은 특정 타입의 원소가 존재하는지 여부를 검사하여 리턴
	boolean add(E anElement)	특정 타입의 원소	true	집합 내부가 꽉 차있지 않으면서, 같은 원소가 존재하지 않으면 배열에 추가
	E remove(E anElement)	특정 타입의 원소	삭제된 특정 타입의 원소	입력받은 특정 타입의 원소가 존재하는지 확인하고, 존재한다면 삭제하고 배열을 한 칸 씩 당기는 메소드. 삭제된 원소의 정보를 전달하기 위해 삭제된 원소를 리턴한다.
	E removeAny()	없음	삭제된 특정 타입의 원소	배열 내에서 임의의 원소를 삭제한다.
	void clear()	없음	없음	집합 내부를 전부 삭제한다.

3 종합 설명서

- 각 자료구조 별 구현 방법에 따른 차이점을 구현 방법 및 차이를 확인함
  - 정렬되지 않고 중복이 없는 집합(Set)의 개념을 적용한 Array 와 LinkedChain
- 고유한 좌표 값과 이름을 가진 별을 삽입하고 삭제 할 수 있다.
- 별의 이름을 입력하여 삭제할 수 있다.

- 임의의 별을 삭제할 수 있다.
- 이름 혹은 좌표로 검색하여 해당하는 별이 존재하는지 확인 할 수 있다
- 입력 시에 좌표 또는 이름이 같은 별의 존재여부를 확인하고 이미 존재한다는 알림 메세지를 출력할 수 있다.

2 프로그램 장단점 분석

- 장점
  - 중복을 허용하지 않는 자료구조이기 때문에 집합 내부의 원소들의 중복 여부를 체크할 수 있다.
  - 원소의 일부 정보(좌표 값 혹은 이름)으로도 중복 여부를 체크할 수 있다.

- 단점
  - 정렬을 할 수 없다.
  - 원소의 정보가 많아 오류 체크 조건을 많이 생각해야 한다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행 및 별의 입력

```
< 별의 집합을 시작합니다 >


1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 1
-  [입력]  -
-  x좌표를 입력하시오 : 1
-  y좌표를 입력하시오 : 1
-  별의 이름을 입력하시오 : a
```

입력 - 좌표가 중복되는 별의 존재여부 출력

```
1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 1
-  [입력]  -
-  x좌표를 입력하시오 : 1
-  y좌표를 입력하시오 : 2
-  별의 이름을 입력하시오 : b


1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 1
-  [입력]  -
-  x좌표를 입력하시오 : 1
-  y좌표를 입력하시오 : 2
-  별의 이름을 입력하시오 : c
ERROR: 잘못된 입력입니다.
```

입력 - 이름이 중복되는 별의 존재여부 출력

```
1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 1
-  [입력]  -
-  x좌표를 입력하시오 : 1
-  y좌표를 입력하시오 : 3
-  별의 이름을 입력하시오 : a
ERROR: 잘못된 입력입니다.
```

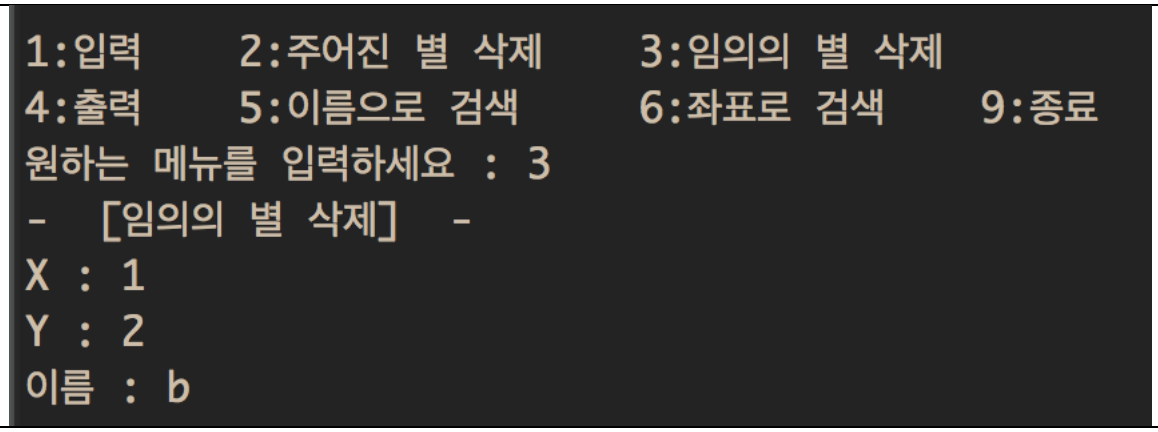
출력 - 정수형 좌표값에 문자형 데이터가 들어갔을 때 오류문

```
1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 1
-  [입력]  -
-  x좌표를 입력하시오 : 3
-  y좌표를 입력하시오 : c
Error Message : For input string: "c"
```

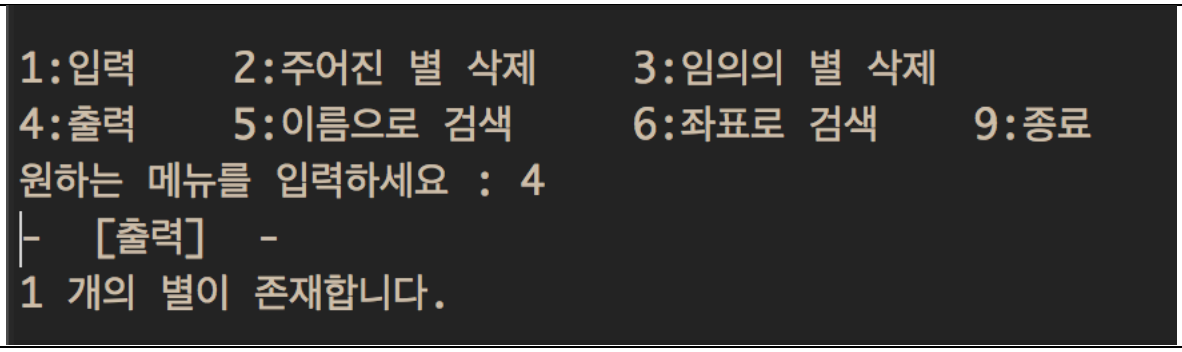
출력 - 원하는 별의 삭제 및 정보 출력

```
1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 2
-  [주어진 별 삭제]  -
-  별의 이름을 입력하시오 : c
X : 1
Y : 3
이름 : c
```

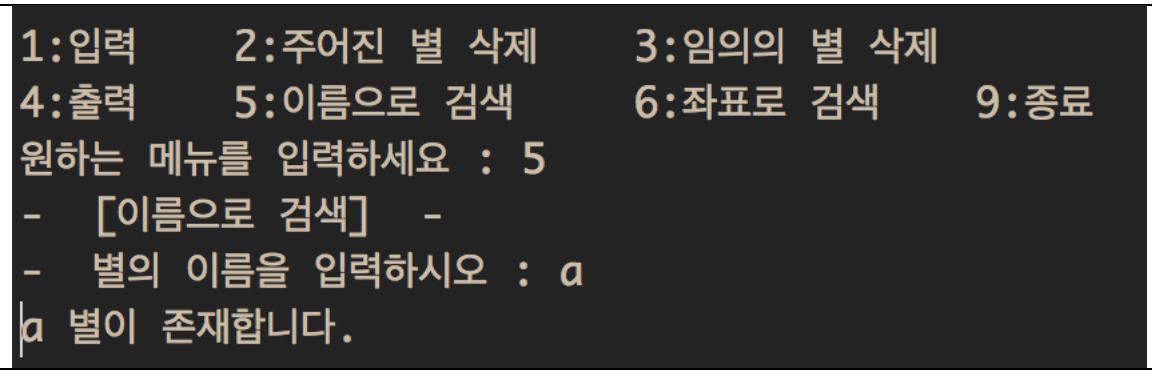
출력 - 임의의 별 삭제 후 정보 출력



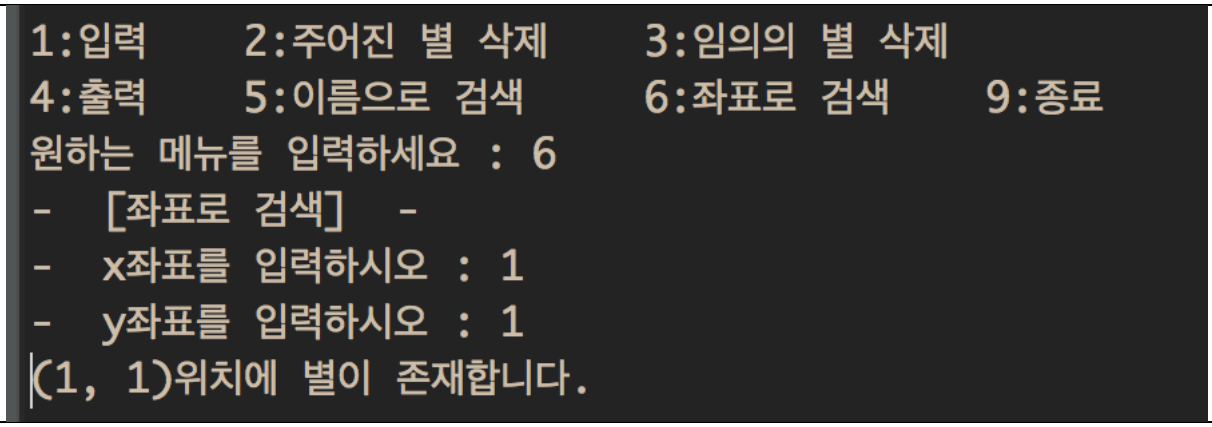
출력 - 집합 내 원소의 총 개수 출력



출력 - 이름으로 원소 검색



출력 - 좌표값으로 원소 검색



출력 - 프로그램 종료 및 집합 내 원소의 총 개수 출력

```
1:입력     2:주어진 별 삭제     3:임의의 별 삭제
4:출력     5:이름으로 검색      6:좌표로 검색     9:종료
원하는 메뉴를 입력하세요 : 9
1 개의 별이 존재합니다.
<별의 집합을 종료합니다.>
```

2 결과 분석

- Bag 은 정렬하지도 않으며 중복된 원소가 있을 수 있는 자료구조이다. 만일 Bag 으로 별의 집합을 구현했다면 중복되는 별이 몇개나 있는지도 검색하는 기능을 만들어야 한다.
- 집합(Set)은 정렬 관계와는 상관없이 없으며 별의 존재 여부를 확인하고 이미 존재하고 있는 똑같은 원소라면 추가하지 않는다. 이것으로 중복되지 않는 원소를 가지는 구조인 것을 알 수 있다.
- LinkedSet 과 ArraySet 은 내부 구성 방법은 다르나, 프로그램 자체의 작동 원리는 Array 혹은 LinkedChain 둘 다 같으므로 자료구조의 구현 방법이 다르더라도 프로그램 실행은 문제 없이 잘 돌아간다.

- 결론
  - 메모리(입력할 데이터의 크기)가 정해져 있는 경우에는 배열
  - 메모리(입력할 데이터의 크기)가 정해져 있지 않은 경우에는 LinkedChain 을 사용해야 한다.
    - ◆ 원소 삽입 시 원소의 중복 여부를 체크해야하는 Set 의 특성 상 자료의 최대 한도가 정해져 있는 경우에는 모든 노드를 검색해야 하는 LinkedChain 보다 배열을 이용하여 빠른 참조가 가능한 Array 방식이 훨씬 좋다.
    - ◆ 자료의 최대 한도가 정해져 있지 않은 경우에는 자료의 메모리 제한없이 할 수 있는 LinkedChain 방식이 훨씬 좋다.

4 소스 코드

Class	AppController

```
public class AppController {
	private AppView _appView;
	private ArraySet<Star> _starCollector;
//	private LinkedSet<Star> _starCollector;

	public AppController() {
		this._appView = new AppView();
	}

	public void run() {
		//this._starCollector = new ArraySet<Star>();
		this._starCollector = new ArraySet<Star>();

		this.showMessage(MessageID.Notice_StartProgram);
		int command = 0;
		while( command != 9) {
			try{
				this.showMessage(MessageID.Notice_Menu);
				command = this._appView.inputInt();
					if (command == 1 ) {
						this.inputStar();
					} else if (command == 2) {
						this.remove();
					} else if (command == 3) {

	this.showMessage(MessageID.Notice_RemoveRandomStar);
						Star removeStar = 
this._starCollector.removeAny();
						if(removeStar != null) {

	this._appView.outputStar(removeStar.starName(), removeStar.xCoordinate(), 
removeStar.yCoordinate());
						} else {

	this.showMessage(MessageID.Error_Remove);
						}
					} else if (command == 4 ) {
						this.showMessage(MessageID.Notice_Show);

	this._appView.outputNumberOfStars(this._starCollector.size());
					} else if (command == 5) {
						this.searchByName();
					} else if (command == 6) {
						this.searchByCoordinate();
					}else if (command == 9) {

	this._appView.outputNumberOfStars(this._starCollector.size());

	this.showMessage(MessageID.Notice_EndProgram);

					} else {
```

```
                this.showMessage(MessageID.Error_WrongMenu);
                                                }
                        } catch (Exception ex) {

                                System.out.println("Error Message : " + ex.getMessage());
                                continue;
                        }
                }




        }
        public void inputStar() {
                this.showMessage(MessageID.Notice_InputStar);

                this.showMessage(MessageID.Notice_InputStarXCoordinate);
                int xCoordinate = this._appView.inputInt();

                this.showMessage(MessageID.Notice_InputStarYCoordinate);
                int yCoordinate = this._appView.inputInt();

                this.showMessage(MessageID.Notice_InputStarName);
                String starName = this._appView.inputString();

                if(! this._starCollector.add(new Star(xCoordinate, yCoordinate, starName))) {
                        this.showMessage(MessageID.Error_Input);
                }

        }


        private void searchByCoordinate() {
                // TODO Auto-generated method stub
                this.showMessage(MessageID.Notice_SearchByCoordinate);
                this.showMessage(MessageID.Notice_InputStarXCoordinate);
                int xCoordinate = this._appView.inputInt();
                this.showMessage(MessageID.Notice_InputStarYCoordinate);
                int yCoordinate = this._appView.inputInt();

                Star searchStar = new Star(xCoordinate, yCoordinate);
                if(this._starCollector.doesContain(searchStar)){
                        this._appView.outputStarExistence(null, xCoordinate, yCoordinate);
                } else
                        this.showMessage(MessageID.Error_Remove);



        }
        private void searchByName() {
                // TODO Auto-generated method stub
                this.showMessage(MessageID.Notice_SearchByName);
                this.showMessage(MessageID.Notice_InputStarName);
                Star searchStar = new Star(this._appView.inputString());
                if(this._starCollector.doesContain(searchStar)) {
                        this._appView.outputStarExistence(searchStar.starName(), 0, 0);
                } else
                        this.showMessage(MessageID.Error_Remove);
```

```
	}
	private void remove() {
		// TODO Auto-generated method stub
		this.showMessage(MessageID.Notice_RemoveStar);
		this.showMessage(MessageID.Notice_InputStarName);
		Star searchStar = new Star(this._appView.inputString());
		if(this._starCollector.doesContain(searchStar)) {
			Star removedStar = this._starCollector.remove(searchStar);
			this._appView.outputStar(removedStar.starName(),
removedStar.xCoordinate(), removedStar.yCoordinate());
		} else {
			this.showMessage(MessageID.Error_Remove);
		}


	}
	private void showMessage(MessageID aMessageID) {
		switch (aMessageID) {
		case Notice_StartProgram :
			this._appView.outputMessage("< 별의 집합을 시작합니다 >₩n");
			break;
		case Notice_Menu :
			this._appView.outputMessage("₩n₩n1:입력      2:주어진 별 삭제      3:임의의
별 삭제₩n" +
		"4:출력      5:이름으로 검색      6:좌표로 검색      9:종료₩n");
			this._appView.outputMessage("원하는 메뉴를 입력하세요 : ");
			break;
		case Notice_EndProgram :
			this._appView.outputMessage("<별의 집합을 종료합니다.>₩n");

			break;
		case Notice_InputStar :
			this._appView.outputMessage("-  [입력]  -₩n");
			break;
		case Notice_InputStarXCoordinate:
			this._appView.outputMessage("-  x 좌표를 입력하시오 : ");
			break;
		case Notice_InputStarYCoordinate:
			this._appView.outputMessage("-  y 좌표를 입력하시오 : ");
			break;
		case Notice_InputStarName:
			this._appView.outputMessage("-  별의 이름을 입력하시오 : ");
			break;
		case Notice_Show:
			this._appView.outputMessage("-  [출력]  -₩n");
			break;
		case Notice_RemoveRandomStar:
			this._appView.outputMessage("-  [임의의 별 삭제]  -₩n");
			break;
		case Notice_RemoveStar:
			this._appView.outputMessage("-  [주어진 별 삭제]  -₩n");
			break;
		case Notice_SearchByName:
			this._appView.outputMessage("-  [이름으로 검색]  -₩n");

			 break;
```

```
                case Notice_SearchByCoordinate:
                        this._appView.outputMessage("-  [좌표로 검색]  -₩n");
                        break;



                case Error_Remove :
                        this._appView.outputMessage("별이 존재하지 않습니다.");
                        break;
                case Error_WrongMenu :
                        this._appView.outputMessage("잘못된 메뉴의 입력입니다.");
                        break;
                case Error_Input :
                        this._appView.outputMessage("ERROR: 잘못된 입력입니다.");
                        break;
                default:
                        break;

                }
        }
}
```

Class	AppView

```
import java.util.Scanner;

public class AppView {
        private Scanner _scanner;

        public AppView() {
                this._scanner = new Scanner(System.in);
        }

        public int inputInt() {
                return Integer.parseInt(this._scanner.next());
        }
        public String inputString() {
                return this._scanner.next();
        }
        public void outputMessage(String aMessage) {
                System.out.print(aMessage);
        }
        public void outputStar(String aStarName, int aX, int aY) {
                System.out.println("X : " + aX +"₩n" + "Y : " + aY + "₩n"+ "이름 : " + aStarName);
        }
        public void outputStarExistence(String aStarName, int aX, int aY){
                if(aX == 0 && aY == 0) {

                System.out.println(aStarName + " 별이 존재합니다.");
                } else if (aStarName == null) {
                        System.out.println("("+ aX+", "+aY+")" + "위치에 별이 존재합니다.");
                }
        }
        public void outputNumberOfStars(int aStarCollectorSize) {
                System.out.println(aStarCollectorSize + " 개의 별이 존재합니다.");
```

```
	}



}
```

Class	ArraySet

```
public class ArraySet<E> {

	private static final int DEFAULT_MAX_SIZE = 100;
	private int _maxSize;
	private int _size;
	private E[] _elements;


	@SuppressWarnings("unchecked")
	public ArraySet() { //집합의 기본 생성자.
		this._maxSize = DEFAULT_MAX_SIZE;
		this._elements = (E[]) new Object[DEFAULT_MAX_SIZE];
		this._size = 0;

	}

	@SuppressWarnings("unchecked")
	public ArraySet(int givenMaxSize) { //집합의 생성자
		this._maxSize = givenMaxSize; //주어진 값으로 최대크기를 정함
		this._elements = (E[]) new Object[givenMaxSize];
		this._size = 0;

	}
	public int size() {
		return this._size; //집합 내부의 원소 사이즈 리턴
	}
	public boolean isEmpty() {
		return (this._size == 0); // 집합이 비었는지 안비었는지
	}
	public boolean isFull() {
		return (this._size == this._maxSize); //집합이 꽉 찼는지 안찼는지
	}
	public boolean doesContain(E anElement) { //집합에 중복되는 원소가 있는지 없는지
		boolean found = false;

		for (int i = 0; i < this._size && ! found; i++) { //집합의 원소 갯수만큼이나 특정 원소를
찾을때까지
			if(this._elements[i].equals(anElement)) { // i 번째의 원소가 특정 원소와 같다면
				found = true;
			}
		}

		return found;
	}
```

```
	public boolean add(E anElement) { // 집합에 원소넣기
		if(this.isFull()) { //만약 꽉 찼으면
			return false; //넣지 말기
		} else if (this.doesContain(anElement)){
			return false;
		} else{//꽉안찼으면
			this._elements[this._size] = anElement; //size 번째 공간에 원소넣기
			this._size++; //사이즈 증가
			return true;
		}
	}

	public E remove(E anElement) { //집합의 원소 삭제하기
		int foundIndex = 0;
		boolean found = false;

		for (int i = 0; i < this._size && !found; i++) { //집합에 들어있는 원소의 갯수나 특정 원소를
찾을 때까지
			if(this._elements[i].equals(anElement)) { //i 번째 원소가 특정 원소랑 같다면
				foundIndex = i; //원소의 번호저장
				found = true; //빙고

			}
		}
		E removedStar = this._elements[foundIndex];
		//삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다.
		if (!found) { //찾지않았으면
			return null; //통과
		} else { //찾았으면
			for (int i = foundIndex; i < this._size-1; i++) { //찾은 원소의 순번부터 원소갯수의 -
1 만큼 진행
				this._elements[i] = this._elements[i+1]; //찾은 원소의 순번에 그 다음
순번의 원소를 집어넣음
				/*한칸씩 땡김*/
			}
			this._elements[this._size-1] = null; //한칸씩땡기면 맨 뒷칸은 비어있게되므로
null 처리
			this._size--; //원소가 들어있는 총 갯수 줄임
			return removedStar;
		}
	}

	public E removeAny() {
		E removedElements = this._elements[this._size-1];
		this._elements[this._size-1] = null;
		this._size--;
		return removedElements; //아무거나 지우는 것은 배열의 맨 뒤에 있는 녀석을 지움.
		 //맨 앞에 있는 것을 지우면 한칸씩 앞으로 당겨야하는 번거로움 발생


	}
	public void clear() { //원소 비우기

		for(int i = 0; i < this._size; i++) {
			this._elements[i] = null; //집합의 모든 내용물을 비우는 작업
		}
		this._size = 0; //들어있는 갯수를 0 개로
	}
```

```
}
```

Class	LinkedSet

```
public class LinkedSet<E> {

	private int _size; //여기서 사이즈는 총 노드의 갯수
	private Node<E> _head;

	public LinkedSet() {
		this._head = null; //초기화실행
		this._size = 0;

	}
	public int size() {
		return this._size; //노드갯수 리턴.
	}
	public boolean isEmpty() {
		return (this._size == 0); //비었는지 안비었는지 체크
	}
	public boolean doesContain(E anElement) {
		boolean found = false;

		Node<E> searchNode = this._head; //제일 앞부터 검색해나가기
		while(searchNode != null && !found) { //검색할 노드가 없고, 못찾을때까지 검색
			if(searchNode.element().equals(anElement)){ //노드의 원소값이 입력된 원소값과
같다면
				found = true; //빙고
			}
			searchNode = searchNode.next(); //아니라면 다음 노드로 이동
		}
		return found;
	}

	public boolean add(E anElement) { //원소 저장하기

		if(! this.doesContain(anElement)) {
			Node<E> newNode = new Node<E>(anElement);
			//주어진 원소값의 새로운 노드를 생성
			newNode.setNext(this._head); // 현재 노드의 다음값을 현재 헤드로 지정. 지금
헤드노드는 뒤로 밀려남.
			this._head = newNode; //현재헤드를 지금 헤드로 새로 지정. 노드를 추가할때는
맨 앞에서 추가하므로 원래 있던 헤드는 뒤로 밀려나가는 형식
			this._size++;
			return true;
		} else {
			return false;
		}
```

```
	}
	public E remove(E anElement) {
		if(this.isEmpty()){
			return null;
		} else {
			Node<E> previousNode = null; //이전노드
			Node<E> currentNode = this._head; //현재노드는 머리부터
			boolean found = false;

			/*입력된 코인값을 가진 노드를 찾기*/
			while(currentNode != null && !found) { //지우고싶은 코인값의 노드 찾기 현재노드가 존재하고있으면서 못찾을때까지 계속
				if(currentNode.element().equals(anElement)) { //입력한 값의 원소값과 현재 원소값이 같다면
					found = true; //빙고
				} else {
					previousNode = currentNode; //아니라면 현재값을 이전값으로 하고
					currentNode = currentNode.next(); //현재값을 현재의 다음 값으로하여 다음 노드로 진행
				}
			}
			E removedElement = currentNode.element();

			/*해당 노드를 찾아 반복문을 탈출했다면 노드를 삭제하는 방법*/
			if(!found) {
				return null; //패스
			} else {
				if(currentNode == this._head) { //찾았을 때, 현재 노드가 헤드라면
					this._head = this._head.next(); //헤드를 헤드노드의 다음 노드로 설정
				} else {
					previousNode.setNext(currentNode.next()); //아니라면 이전노드의 다음 노드를 현재 노드의 다음 노드로 설정
					//이전노드와 다음노드의 사이에 있는 현재 노드를 삭제
				}
				this._size--; //여기서 없어진 노드는 garbage collection 으로 처리.
				return removedElement;
			}
		}

	}
	public E removeAny () {
		if (this.isEmpty()) {
			return null ;
		} else {
			E removedCoin = this._head.element(); //삭제될 원소값을 저장. 삭제되는 원소값이 무엇인지 확인해야하므로
			this._head = this._head.next();
			this._size--;
			return removedCoin; //아무거나 없애는 것이면 맨 앞에 있는 노드를 없애는 것.
		}
	}
	public void clear() {
		this._size = 0;
		this._head = null; //초기화. 헤드도 없앰.

	}
```

```
}
```

Class	Star

```
public class Star {
	private int _xCoordinate; //x 좌표
	private int _yCoordinate;
	private String _starName;

	public Star(int givenX, int givenY) { //좌표값만 있는 별
		this._xCoordinate = givenX;
		this._yCoordinate = givenY;
		this._starName = null;

	}
	public Star(String givenStarName) { //이름만 있는 별
		this._starName = givenStarName;
		this._xCoordinate= 0;
		this._yCoordinate=0;
	}
	public Star(int givenX, int givenY, String givenStarName) { //이름과 좌표가 둘 다 있는 별
		this._xCoordinate = givenX;
		this._yCoordinate = givenY;
		this._starName = givenStarName;


	}
	public int xCoordinate() { //X 좌표
		return this._xCoordinate;
	}
	public int yCoordinate() { //Y 좌표
		return this._yCoordinate;

	}
	public String starName() { //별 이름 반환
		return this._starName;
	}

	public void setXCoordinate(int newX) { //x 좌표 설정
		this._xCoordinate = newX;
	}
	public void setYCoordinate(int newY) { //Y 좌표 설정

		this._yCoordinate = newY;
	}
	public void setStarName(String newStarName) {
		this._starName = newStarName; //별 이름 설정
	}

	public boolean equals (Object object) { //입력받은 별이 현재별과 같은지 중복여부 확인

		Star aStar = (Star) object;
		if (this._xCoordinate== aStar.xCoordinate() && this._yCoordinate == aStar.yCoordinate())
{
```

```
                        return true;
                }
                if(aStar.starName() != null && this._starName.equals(aStar.starName())) {
                        return true;

                } else {
                        return false;
                }
        }

}
```

Class	Node

```
public class Node<E> {

        private E _element; //노드에 담기는 코인
        private Node<E> _next;



        public Node(){ //노드 초기화
                this._element = null;
                this._next = null;
        }

        public Node(E givenElement) { //노드의 원소값을 설정
                this._element = givenElement;
                this._next = null;
        }

        public Node(E givenElement,   Node<E> givenNext) { //노드의 원소와 다음노드값을 설정
                this._element = givenElement;
                this._next = givenNext;
        }

        public E element() {


                return this._element; //원소값 리턴
        }

        public Node<E> next() {
                return this._next; //다음노드값 리턴

        }
        public void setStar(E anElement) {
                this._element = anElement; //원소값 새로 설정

        }
        public void setNext (Node<E> newNext) {
                this._next = newNext; //다음 노드값 새로 설정
        }
```

```
}
```