Array, Collection

🫧 버블 정렬

▼ Array(배열)

- 특징

생성 시 배열의 크기를 고정적으로 지정하여 확장성이 낮다.

(자바 스크립트에서의 배열은 리스트처럼 자동으로 크기가 커지면서 데이터를 추가할 수 있다)

> 💡 문자열(String)을 담을 수 있는 배열
> 오른쪽 대괄호 [ ] 안에 있는 숫자를 첨자라고 하며
> 해당 배열이 담을 수 있는 값의 개수를 의미한다.

- 배열 선언 및 생성

```
// 값을 넣으면서 배열을 선언
String[] students = {"건희", "태완", "정현"};

// 배열의 크기가 4인 seoyugi 배열 생성
// 배열에 값은 나중에 추가
String[] seoyugi = new String[4];
```

- .length : 배열의 크기 리턴
- .length( ) : 문자열의 길이 리턴

```
// 문자열의 길이
System.out.println(palgye.length());
```

```
// length에 소괄호 없음 = 배열의 크기 리턴
System.out.println(seoyugi.length);
```

- 배열에 값 넣기 : 배열명[값을 넣을 인덱스 번호]

```
// seoyugi = [null, null, null, null]
seoyugi[0] = "삼장";
// seoyugi = ["삼장", null, null, null]
seoyugi[1] = "오공";
// seoyugi = ["삼장","오공", null, null]
seoyugi[2] = "사오정";
seoyugi[3] = "저팔계";
```

- 배열 내 요소 확인 : 반복문 사용

System.out.println에 직접 배열명을 넣어 찍어보면

[Ljava.lang.String;@776ec8df] 같은 16진수로 된 주소값을

콘솔창에 출력함

```
for (int i = 0; i < seoyugi.length; i++) {
  // i 는 0,1,2,3 -> 인덱스로 사용 가능
  System.out.println(seoyugi[i]);
}
```

- 배열의 장단점

장점 : for문을 이용하여 배열의 담긴 모든 요소들을 한 번에 처리 할 수 있다.

단점 : 배열 생성 시 배열의 크기를 지정하기 때문에 배열 사용에 있어 배열의 크기가 고정되어 있어 확장성이 부족하다

- .split(문자열) : 문자열 나누기

괄호 안 문자열을 기준으로 해당 문자열을 나누어서 `문자열 배열`로 리턴

```
String company = "카카오, 네이버, 배민, 요기요";

String[] comArray = company.split(",");
System.out.println(comArray.length);
System.out.println(comArray[1]);
```

- .trim( ) : 좌우 공백 제거

```
String company = "카카오, 네이버, 배민, 요기요";
for (int i = 0; i < comArray.length; i++) {
      comArray[i] = comArray[i].trim();
  }

  System.out.println(comArray[1]);
// 인덱스 1번에 담긴 " 네이버" 에 공백을 제거해서
// "네이버"가 출력된다.
```

- 배열의 복사

```
String[] sinSeoyugi = seoyugi;

// 새로운 문자열 배열 생성과 동시에
// 기존에 있던 배열을 값으로 담기
```

- System.out.println(배열명) :

[Ljava.lang.String;@776ec8df [String 타입 배열, 메모리 주소]

를 출력하는데 @776ec8df 는 메모리 주소의 해쉬코드를 16진수로 나타낸 것

- hashCode( ) : 해쉬코드

```
System.out.println(seoyugi.hashCode());    // 2003749087
System.out.println(sinSeoyugi.hashCode()); // 2003749087
// 10진수 16진수로 변환
System.out.println(Integer.toHexString(2003749087));
```

객체의 메모리 주소값에 해쉬를 적용한 코드

- 해쉬란?

해쉬함수(암호화 알고리즘)을 이용해서 데이터를 암호화하는 기법

임의의 길이의 데이터를 고정된 길이의 데이터로 암호화하는 기법으로 고유한 값을 지닌다.

- .clone( ) : 올바른 배열 복사

기존에 있던 배열의 데이터를 수정,변경,삭제하면 곤란하므로

clone( )를 사용하여 복사본을 만든다.

```
String[] sinSeoyugi = seoyugi;
```

이 방법은 같은 주소지를 바라보고 있는 것이기 때문에 바로가기 아이콘을 만든 것같은 방식이다.

올바른 방식

```
String[] newSeoyugi = seoyugi.clone();
```

clone( ) 없이 배열 복사 (for문 사용)

```
// 1. 복사본 배열과 같은 크기를 갖는 새로운 배열 만들기
String[] newArr = new String[oldArr.length];
```

```
// 2. oldArr의 각 인덱스 값을 newArr에 넣기
for(int i = 0; i < oldArr.length; i++){
    newArr[i] = oldArr[i];
}

System.out.println(newArr);
System.out.println(oldArr);
```

- 배열 내 인덱스 값을 서로 바꿔보기

```
int[] numberArray ={11,2,56,65,89,21};

// 인덱스 0번과 인덱스 1번 위치 바꾸기


// 1번 방법(단순한 버전)



// 2번 방법 (버블정렬식)
```

- Call by Value 와 Call by Reference

Call by Value : 기본타입의 객체(변수)에 대해서 동작

Call by Reference : 참조타입의 객체(변수)에 대해서 동작

- Arrays.sort(배열명) : 오름차순 정렬

```
Arrays.sort(numberArray);
printArray(numberArray);
```

활용편) 내림차순 정렬

1. 반복문을 통해 모든 요소에 접근해 (-1) 를 곱하여 음수로 만들고
2. Arrays.sort로 오름차순 정렬
3. 음수의 상태인 배열을 다시 (-1) 곱하여 양수로 변환

```
int[] numberArray = {2,5,8};

// 1번
for(int i = 0; i < numberArray.length(); i++){
    numberArray[i] *= -1;
}

// 2번
Arrays.sort(numberArray);


// 3번
for(int i = 0; i < numberArray.length(); i++){
    numberArray[i] *= -1;
}

System.out.println(numberArray);
```

- 버블정렬

 버블 정렬

▼ List(리스트)

- 특징

순서를 유지하고 저장하며, 중복 저장이 가능하다.

자바의 일반 배열과는 다르게 제한 없이 객체 추가가 가능하다.

- 리스트 생성

ArrayList<객체타입>리스트명 = new ArrayList<객체타입>( );

```
// new ArrayList의 제너릭안에 타입을 적지 않아도 된다.
ArrayList<String> students = new ArrayList<String>();
ArrayList<String> students2 = new ArrayList<>();

// 다형성을 이용한 선언
List<String> students3 = new ArrayList<>();
```

제너릭 안에는 참조타입 객체만 담을 수 있다.

기본타입을 담으려면 Wrapper Class를 이용해야한다.

```
ArrayList<Integer> intList = new ArrayList<>();
// int 타입(기본타입)은 Integer를 사용해야한다.
```

- .intValue( ) : Wrapper class로 저장된 값을 기본타입으로 꺼내기

(= 언박싱)

```
int num = numObj.intValue();
    System.out.println(num);

// 오토 박싱과 오토 언박싱이 자동으로 이루어지기 때문에 에러 X
num = numObj;
System.out.println(num);
```

- .size( ) : 리스트의 크기 확인

```
ArrayList<Integer> intList = new ArrayList<>();

System.out.println(intList.size());
// 안에 값이 들어가있는 갯수 확인 , 0 출력
```

```
System.out.println(intList);
// 안에 아무런 데이터도 추가되어있지않아 [] 출력
```

- .add( ) : 리스트에 데이터 추가

```
intList.add(10);
System.out.println(intList.size());
// intList.add(10)으로 하나를 담았으므로 1 출력
intList.add(20); // [10, 20] 이 담겨져있음

// 값을 한번에 여러개 넣는것은 불가능하다.
//    intList.add(30,40);
//    System.out.println(intList);
  intList.add(30);
  intList.add(40);

// 제너릭 타입과 일치하지 않는 데이터는 추가할 수 없다.
//     intList.add("받아줘"); // 숫자타입의 배열에 문자열 X
//     intList.add(3.14);     // 숫자타입의 배열에 double X
```

- .get( ) : 리스트의 인덱싱 (= 인덱스 확인)

```
ArrayList<String> stuList = new ArrayList<>();
        stuList.add("건희");
        stuList.add("태완");
        stuList.add("정현");
        stuList.add("승환");
        stuList.add("해성");
        stuList.add("유나");

System.out.println(stuList.get(3));
// stuList.get(3) 이 '문자열' 승환을 출력
```

```
System.out.println(stuList.get(3).substring(0,1));
// 인덱스 3번인 "승환"에 .substring(0,1)을 하면 "승" 반환
```

- .set(인덱스, 변경할 값) : 리스트에 있는 값 변경

```
stuList.set(3, "형구");
// stuList의 인덱스 3번에 담겨있는 값 "승환"을 "형구"로 변경
System.out.println(stuList);
// [건희,태완,정현,형구,해성,유나] 리턴
```

- .isEmpty( ) : 리스트의 데이터 존재 유무 확인

```
System.out.println(stuList.isEmpty());
// stuList = [건희,태완,정현,형구,해성,유나];로
// 크기가 6인 배열이므로 false를 출력한다.
```

- .remove(인덱스) : 해당 인덱스의 데이터 삭제

```
System.out.println(stuList.get(2)); // 정현
  stuList.remove(2); // "정현"을 삭제

System.out.println(stuList); // [건희,태완,형구,해성,유나]
  System.out.println(stuList.get(2)); // "형구"

// remove으로 데이터를 삭제하게 되면
// 삭제된 인덱스 이후의 데이터들이 한 칸씩 앞당겨진다.
```

- .clear( ) : 리스트의 데이터를 모두 삭제

```
stuList.clear();
  System.out.println(stuList);
// [] 아무것도 들어있지 않은 배열을 리턴
```

- .addAll( ) : 리스트 복사 (올바르지 않는 복사법)

```
ArrayList<Integer> copyList = new ArrayList<>();

  // numList 내부 요소들이 전부 copyList에 복사된다.
  copyList.addAll(numList);
    // 빈 리스트명.addAll(복사하고자 하는 원본 리스트)
```

- 올바른 복사법

```
// 리스트 복사 두번째 방법,
// 빈 리스트에 numList를 담는것
ArrayList<Integer> copyList2 = new ArrayList<>(numList);
System.out.println(copyList2);

// 리스트 복사 세번째 방법,
// 새로운 빈 리스트 생성 후 for문으로 데이터 추가
ArrayList<Integer> copyList3 = new ArrayList<>();
for (int i = 0; i < numList.size(); i++) {
    copyList3.add(numList.get(i));
}
System.out.println(copyList3);
```

- 리스트 정렬

- Vector

ArrayList와 동일한 내부 구조를 가지지만 Vector는 동기화된 메소드로 구성 되어 있어 멀티 스레드 환경에서 사용하면 좋다

▼ Hash Map(맵)

- 특징

순서를 유지하지 않고 저장하며, 중복 저장이 불가능하다.

- .put(key, value) : 데이터 추가
- .get(key) : 데이터 조회
- .size( ) : 크기 조회
- **.containsKey( )** : 조건에 맞는 데이터(키 기준) 존재유무 확인
- **.containsValue( )** : 조건에 맞는 데이터(값 기준) 존재유무 확인
- .remove( ) : 조건으로 건 key값을 가지는 내부 요소 삭제

- Map의 내부 데이터 순회
  - .keySet( ) : map 내부 요소들의 key 값을 Set에 담아 확인

```
// System.out.println(stuMap) 은 {첫째=연지, 둘째=미승}

Set<String> keySet = stuMap.keySet();
System.out.println(keySet); // [첫째, 둘째]
```

- .entrySet( ) : map 내부 요소들을 .getkey( ), .getValue( )로 원하는 데이터를 확인

```
Set<Entry<String,String>> entrySet = stuMap.entrySet

for (  Entry<String,String> entry : entrySet ){
    System.out.println(entry);
        // key값 = value값 출력 (첫째=연지 둘째 = 미승 출력
    System.out.println(entry.getKey());
        // key값만 출력 (첫째 둘째 출력)
    System.out.println(entry.getValue());
        // value값만 출력 (연지 미승 출력)
}
```

▼ Set(부모), Hash Set(자식)

- 특징

키와 값이 쌍을 이루며 저장되며, 키의 중복 저장이 불가능하다.

Set에 담겨있는 데이터에는 인덱스가 없으며 순서 또한 존재하지 않는다.

- .add( ) : 데이터 추가
- .contains( ) : 원하는 데이터 존재유무 확인
- .size( ) : Set 내부 요소 개수 리턴
- .addAll( ) : 모든 데이터 복사 붙여넣기
- .clear( ) : Set 내부 비우기

- Set 내부 요소 순회
  - while문 사용

(while문 조건에 .hasNext를 사용하여 존재하지 않을 경우 false를 리턴하므로 자동으로 반복문 종료)

```
//1. while문 사용 (인덱스가 없으므로 for문 사용 힘듦)
  Iterator<String> stuIter = stuSet.iterator();
```

```
  while (stuIter.hasNext()){
    String stu = stuIter.next();
      System.out.println(stu);
  }
```

- 향상된 for문 사용

```
for ( String stu :  stuSet ) {
    System.out.println(stu);
}
```

- forEach문 + 람다식 사용

```
stuSet.forEach(s -> System.out.println(s));
stuSet.forEach(stu -> System.out.println(stu));
```