[12/15] Java - Array

1) 배열 https://wikidocs.net/206

1-1) String[ ]

- System.out.print 명령을 통해 배열값을 출력하기위해서는 Arrays.toString( )을 사용하면 된다.
- import java.util.Arrays를 작성 하는 것이 우선이지만 ArrayList를 작성하거나 System.out.println(Arrays.toString(weeks)) 을 Run 하면 자동으로 생성되기도 한다.

```java
import java.util.Arrays;

public class JavaExm12_14_02 {
    public static void main(String[] args){
        int[] odds = {1,3,5,7,9};
        String[] weeks = {"월","화","수","목","금","토","일"};
        String[] week = new String[7];
        week[0] = "월";
        week[1] = "화";
        week[2] = "수";
        week[3] = "목";
        week[4] = "금";
        week[5] = "토";
        week[6] = "일";

        System.out.println(Arrays.toString(weeks)); //[월,화,수,목,금,토,일]
        System.out.println(Arrays.toString(week)); //[월,화,수,목,금,토,일]

        for (int i=0; i< weeks.length; i++){
            System.out.println(weeks[i]);
        }
    }
}
```

```java
for (int i=0; i< weeks.length; i++){
    System.out.println(weeks[i]);
}
```

for 문으로도 출력 가능, 월 ~일 한줄씩

- int[] odds : odds 변수는 정수형의 배열
- String[] weeks : weeks 변수는 문자열의 배열
  - 배열의 형식 : dataType[ ] 변수명 = { }
- weeks : 단순 배열의 문자열
- week : 배열의 오브젝트화
  - String[7] 과 추가되는 week[N] 이 안맞으면 에러뜸

1-2) ArrayList : 작성시 import java.util.Arrays 자동생성

- ArrayList<자료형> 변수명 = new ArrayList<유지/변경할 자료형>();
    - Java8 이후 Generic 도입으로
    - Array 변수명 = new 변수명(); 도 가능해짐

```java
import java.util.ArrayList;

public class JavaExm12_15_01 {
    public static void main(String[] args) {
        ArrayList<String> pitches = new ArrayList<>(); //빈 배열 생성
//      ArrayList pitches = new ArrayList(); //이 형태로도 구동은 가능
        pitches.add("138"); //배열 값 추가
        pitches.add("129"); //배열 값 추가
        pitches.add("145"); //배열 값 추가

        System.out.println(pitches.get(2)); //145
        System.out.println(pitches.size()); //3
        System.out.println(pitches.contains("145")); //true
        System.out.println(pitches.contains("140")); //false
        System.out.println(pitches.remove( index: 0)); //138
        System.out.println(pitches.remove( o: "129")); //true
        System.out.println(pitches); //145
    }
}
```

- get(n) : n 번째의 값을 호출
- size : ArrayList 에서의 length
- contains : 포함여부를 T/F
- remove() : () 안에 숫자 작성시 해당 index 를 자동지정
    - index 로 삭제시 : 삭제 값이 출력
    - 해당 값을 직접 작성해서 삭제시 : T/F

1-2-1) Generic : 클래스 내부에서 지정하는 것이 아닌 외부에서 사용자에 의해 지정되는 것

- 특정(Specific) 타입을 미리 지정해주는 것이 아닌 필요에 의해 지정할 수 있도록 하는 일반(Generic) 타입이라는 것

```java
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<String> list2 = new ArrayList<Integer>();

LinkedList<Double> list3 = new LinkedList<Double>():
LinkedList<Character> list4 = new LinkedList<Character>();
```

원래는 하나씩 데이터타입을 작성해야 한다

- 장점
    - 1. 제네릭을 사용하면 잘못된 타입이 들어올 수 있는 것을 컴파일 단계에서 방지할 수 있다.
    - 2. 클래스 외부에서 타입을 지정해주기 때문에 따로 타입을 체크하고 변환해줄 필요가 없다. 즉, 관리하기가 편하다.
    - 3. 비슷한 기능을 지원하는 경우 코드의 재사용성이 높아진다.

- 제네릭 사용이전에는 명시적 형변환을 했었다.

```java
public class JavaExm12_15_02 {
    public static void main(String[] args) {
        ArrayList play = new ArrayList<>();
        play.add("129");
        play.add("135");
        play.add("129");

        String one = (String) play.get(0);
        String two = (String) play.get(1);
        String sam = (String) play.get(2);
    }
}
```

1-3) 동일한 결과 코드

```java
import java.util.ArrayList;
import java.util.Arrays;

public class JavaExm12_15_03 {
    public static void main(String[] args) {
        String[] data = {"129", "135", "129"};
        ArrayList<String> play = new ArrayList<>(Arrays.asList(data));
        System.out.println(play); //[129, 135, 129]
    }
}
```

1-4) 정렬

- 오름차순 : play.sort(Comparator.naturalOrder());
- 내림차순 : play.sort(Comparator.reverseOrder());

```java
ArrayList<Integer> play = new ArrayList<>(Arrays.asList(1,7,2,4,5,3,6));
play.sort(Comparator.reverseOrder());
System.out.println(play); //[7, 6, 5, 4, 3, 2, 1]
```

  - 복붙하면 잘안될 수 도 있으니 직접 작성

1-6) join : 배열에서도 사용 가능

```java
public class JavaExm12_15_05 {
    public static void main(String[] args) {
        ArrayList<String> pitches = new ArrayList<>(Arrays.asList("138", "129", "142"));
        String result = String.join( delimiter: ",", pitches);
        System.out.println(result);  // 138,129,142 출력
    }
}
```

2) Map

- object{key:value} 구조와 동일
    - Key값은 중복이 불가능하고 Value는 중복이 가능, 순서없는 자료 저장
- 종류 : HashMap, LinkedHashMap, TreeMap, Hashtable
- 데이터 읽기 성능 : HashMap > LinkedHashMap > HashTable > TreeMap

- 특징 : Map과 Multimap, TreeMap, HashMap, LinkedHashMap 차이
    - Multimap : 키 중복의 경우, 같은 키의 여러 value의 경우
    - TreeMap : 정렬이 필요한 경우 (검색에 있어서 가장 느리다.)
    - HashMap : 메모리 보다 성능이 우선일 경우
        - 검색에 관해 대부분 O(1) 성능인 HashMap 사용
    - LinkedHashMap : 일정한 수행시간과 삽입 순서를 유지, 입력순서가 중요할 때

2-1) HashMap : 맵 자료형 중 가장 기본

- HashMap 역시 제네릭스를 이용한다.
- HashMap의 제네릭스는 key, value 모두 String 자료형이다.
- null key null value를 모두 허용

```java
import java.util.HashMap;
public class JavaExm12_15_06 {
    public static void main(String[] args) {
        HashMap<String, String> map = new HashMap<>();
        map.put("people", "사람");
        map.put("baseball", "야구");
        System.out.println(map.get("people")); //사람
        System.out.println(map.getOrDefault( key: "java",  defaultValue: "자바")); //자바
        System.out.println(map.containsKey("people")); //true
        System.out.println(map.remove( key: "people"));  // "사람" 출력
        System.out.println(map.size()); //앞서 remove를 했으므로 1 이 출력
        System.out.println(map.keySet()); // remove를 하지 않았을 경우 [baseball, people]
    }
}
```

    - import java.util.HashMap; 을 써야 HashMap -> HashMap이 됨
    - put : put method 를 통해 자료형 추가
    - get : get method 를 통해 key 를 지정해 해당 value 를 호출
    - getOrDefault : key 가 있으면 그대로 해당 value를 호출, 없으면 defaultValue 를 호출
    - remove : 삭제
    - size : = length
    - keyset : 모든 key 호출

2-2) LinkedHashMap : 입력된 순서대로 데이터를 저장

- 검색과 삽입에 O(1) 시간이 소요된다.
- 구현은 양방향 연결 버킷(double-linked bucket)으로 구현되어 있다.

2-3) TreeMap : 입력된 key의 오름차순으로 데이터를 저장

- 검색과 삽입에 O(log N) 시간이 소요된다.
- 즉, 키는 반드시 Comparable 인터페이스를 구현하고 있어야 한다.
- 구현은 레드-블랙 트리로 구현되어 있다.

2-4) Hashtable : key 값에 index를 부여해 빠른 검색 속도를 확보

- 검색과 삽입에 O(1) 시간이 소요된다.
- 키의 순서는 무작위로 섞여 있다.
- 빠른 검색속도를 제공하는 이유는 내부적으로 배열(버킷)을 사용하여 데이터를 저장
- 각각의 Key값에 해시함수를 적용해 배열의 고유한 index를 생성하고, 이 index를 활용해 값을 저장하거나 검색하게 된다. 여기서 실제 값이 저장되는 장소를 버킷 또는 슬롯이라고 한다.

2-5) HashMap(해시맵)과 HashTable(해시테이블) 차이 : 동기화 지원 여부

- 병렬 처리를 하면서 자원의 동기화를 고려해야 하는 상황이라면 해시테이블(HashTable)을 사용
- 병렬 처리를 하지 않거나 자원의 동기화를 고려하지 않는 상황이라면 해시맵(HashMap)을 사용

3) 집합(set) 자료형 : 집합과 관련된 것을 쉽게 처리하기 위해 만든 것

- 중복값 자동 제거, 순서없음(인덱싱 불가, map과 유사)
- 종류 : HashSet, TreeSet, LinkedHashSet
- 활용 : 중복을 제거하기 위한 필터 역할

3-1) HashSet : 오름차순 정렬
자바 깊이 알기 / HashSet의 원리

```java
public class JavaExm12_15_07 {
    public static void main(String[] args) {
        HashSet<Integer> s1 = new HashSet<>(Arrays.asList(1,2,3,4,5));
        HashSet<Integer> s2 = new HashSet<>(Arrays.asList(2,4,6,8,10));

        HashSet<Integer> intersection = new HashSet<>(s1); //s1으로 intersection 생성
        intersection.retainAll(s2); //교집합
        System.out.println( intersection); //[2,4]

        HashSet<Integer> union = new HashSet<>(s1);  //s1으로 union 생성
        union.addAll(s2); //합집합
        System.out.println(union);  //[1, 2, 3, 4, 5, 6, 7, 8, 9]

        HashSet<Integer> substract = new HashSet<>(s1);  //s1으로 substract 생성
        substract.removeAll(s2); //차집합
        System.out.println(substract);  //[1, 2, 3]
    }
}
```

4) enum(상수 집합) : 서로 연관 있는 여러 개의 상수 집합을 정의할 때 사용

-

```java
public class JavaExm12_15_08 {
    no usages
    enum CoffeeType {
        no usages
        AMERICANO,
        no usages
        ICE_AMERICANO,
        no usages
        CAFE_LATTE
    };
    public static void main(String[] args) {
        System.out.println(CoffeeType.AMERICANO);  // AMERICANO
        System.out.println(CoffeeType.ICE_AMERICANO);  // ICE_AMERICANO
        System.out.println(CoffeeType.CAFE_LATTE);  // CAFE_LATTE
    }
}
```

- Enum을 사용한 매직넘버 제거

```java
public class JavaExm12_15_10 {
    6 usages
    enum CoffeeType {
        2 usages
        AMERICANO,
        1 usage
        ICE_AMERICANO,
        1 usage
        CAFE_LATTE
    };
    2 usages
    static void printCoffeePrice(CoffeeType type){
        HashMap<CoffeeType, Integer> priceMap = new HashMap<>();
        priceMap.put(CoffeeType.AMERICANO, 3000);
        priceMap.put(CoffeeType.ICE_AMERICANO, 4000);
        priceMap.put(CoffeeType.CAFE_LATTE, 5000);
        int price = priceMap.get(type);
        System.out.println(String.format("가격은 %d 원입니다.", price));
    }

    public static void main(String[] args) {
        printCoffeePrice(CoffeeType.AMERICANO);
        printCoffeePrice( type: 99);
    }
}
```