61장. Collection

1. List

1) ArrayList

- 배열을 이용하여 만든 리스트
- 인덱스로 조회가 가능하므로, 인덱스만 알면 빠른 탐색이 가능하다.
- 반면 삽입과 삭제 시 나머지 노드의 위치를 조정을 해야 하기 때문에 LinkedList보다 다소 느리다.

```
public class ArrayListTest {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>(Arrays.asList("black", "white"));

        colors.add("pink");

        // advanced For
        for(String color : colors){
            System.out.println(color);
        }

        // Iterator 활용 Loop
        Iterator<String> iterator = colors.iterator();
        while (iterator.hasNext()){
            String color = iterator.next();
            System.out.println(color);
        }
    }
}
```

2) LinkedList

- 연결 리스트를 이용하여 만든 리스트
- 조회 시 첫 노드부터 순회하기 때문에 탐색의 속도가 느리다.
- 반면 삽입과 삭제 시 가리키고 있는 주소값만 변경해주면 되기 때문에 ArrayList에 보다 효율적이다.

```
public class LinkedListTest {
    public static void main(String[] args) {
        LinkedList<String> colors = new LinkedList<>(Arrays.asList("black", "white"));
        colors.add("pink");
```

```
        // advanced For
        for(String color : colors){
            System.out.println(color);
        }
    }
}
```

2. Set

순서가 없는 데이터의 집합(데이터의 중복을 허용하지 않음.)

1) HashSet

- Hash Table을 이용하여 만든 Set
- 중복을 허용하지 않고, 순서를 갖지 않는다.

```
public class HashSetTest {
    public static void main(String[] args) {
        Set<String> fruits = new HashSet<>(Arrays.asList("apple", "banana"));

        fruits.add("grape");
        fruits.add("apple");

        for(String fruit : fruits){
            System.out.println(fruit);
        }
    }
}
```

2) TreeSet

- RBT(Red-Black Tree)를 이용하여 만든 Set
- 중복을 허용하지 않고, 순서를 갖지 않는다.
- 정렬된 상태를 유지하기 때문에 삽입, 삭제 시 HashSet보다 성능이 떨어진다.

```
public class TreeSetTest {
    public static void main(String[] args){

        Set<String> sports = new TreeSet<>(Arrays.asList("baseball","soccer"));

        //add - 새로운 sport 추가
        sports.add("golf");
```

```
        //add - 중복된 sport이므로 저장 X
        sports.add("baseball");

        //for-loop print
        for(String sport : sports){
            System.out.println(sport); //[baseball golf soccer] <- 자동 정렬
        }
    }
}
```

3. Map

<Key,Value> 쌍을 갖는 데이터의 집합(key의 중복을 허용하지 않음.)

1) HashMap

- Hash Table을 이용하여 만든 Map
- Key는 중복이 불가능하고, Value는 중복이 가능하다.

```
public class HashMapTest {
    public static void main(String[] args) {
        Map<String, Integer> store = new HashMap<>();

        //add
        store.put("water", 1);
        store.put("fruit", 5);

        //add - water가 이미 있으므로, update
        store.put("water", 2);

        //keyset을 이용하여 출력
        for(String key: store.keySet()){
            System.out.println(key + " : " + store.get(key));
        }
    }
}
```

2) TreeMap

- RBT(Red-Black Tree)를 이용하여 만든 Map
- Key는 중복이 불가능하고, Value는 중복이 가능하다.
- Key를 기준으로 정렬된 상태를 유지하기 때문에 삽입, 삭제 시 HashMap보다 성능이 떨어진다.

```
public class TreeMapTest {
	public static void main(String[] args){
		Map<String,Integer> clothes = new TreeMap<>();

		//add
		clothes.put("shoes",2);
		clothes.put("jean",1);

		//add - jean이 이미 있으므로 새로운 value로 업데이트
		clothes.put("jean",3);

		//keyset을 이용하여 print
		for(String key : clothes.keySet()){  //[jean:3 shoes:2] <-자동 정렬
			System.out.println(key + ":" + clothes.get(key));
		}
	}
}
```