

10장. 컬렉션 프레임워크



ArrayList, HashMap



제네릭(Generic)

❖ 제네릭 프로그래밍

- 어떤 값이 하나의 자료형이 아닌 여러 자료형을 사용할 수 있도록 프로그래밍하는 것.

Java 5부터 제네릭(Generic) 타입이 새로 추가되었는데, 제네릭 타입을 이용함으로써 잘못된 타입이 사용될 수 있는 문제를 **컴파일 과정에서 제거**할 수 있게 되었다.

또한, 비제네릭 코드는 불필요한 **타입 변환**을 하므로 프로그램 성능에 악영향을 미친다.

- '컬렉션 프레임워크(자료구조)'도 많은 부분이 제네릭으로 구현되어 있다.

```
public class 클래스명 <T> {....}
```



제네릭(Generic)

❖ 제네릭 타입 정의

제네릭 타입은 타입(type)을 파라미터로 가지는 클래스를 말한다.

```
package generic.box;

public class Box<T> {
    //T는 자료형, type은 멤버 변수
    private T type;

    public void set(T type) {
        this.type = type;
    }

    public T get() {
        return type;
    }
}
```

```
public class Car {

    String name;

    Car(String name){
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
}
```



제네릭(Generic)

❖ 제네릭 타입 테스트

```
public class BoxTest {  
    public static void main(String[] args) {  
        //String 형 - 기본 자료형  
        Box<String> box1 = new Box<>();  
        box1.set("행운을 빌어요");  
        String msg = box1.get();  
        System.out.println(msg);  
  
        //Integer 형  
        Box<Integer> box2 = new Box<>();  
        box2.set(10);  
        Integer num = box2.get();  
        System.out.println(num);  
  
        //Car 타입 - 사용자 정의 자료형  
        Box<Car> box3 = new Box<>();  
        box3.set(new Car("아이오닉6"));  
        Car car = box3.get();  
        System.out.println(car);  
    }  
}
```

행운을 빌어요
10
아이오닉6



제네릭(Generic)

❖ 비제네릭 타입 정의

```
package generic.box;

public class Car {

    String name;

    public Car(String name){
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
}
```



제네릭(Generic)

❖ 비제네릭 타입 테스트

```
public class BoxTest {  
    public static void main(String[] args) {  
        // String type  
        Box box1 = new Box();  
        box1.set("Good Luck!!");  
  
        //String이 Object 보다 작으므로 오류 발생(형변환 필요)  
        String msg = (String)box1.get();  
        System.out.println(msg);  
  
        //클래스 형  
        Box box2 = new Box();  
        box2.set(new Car("EV4"));  
  
        //Car 타입이 Object 타입 보다 작으므로 오류 발생(형변환 필요)  
        Car car = (Car)box2.get();  
        System.out.println(car);  
    }  
}
```



제네릭(Generic)

❖ 제네릭 프로그래밍 - 3D 프린터 예제.

```
package generic.printer;
public class GenericPrinter<T> {
    private T material;

    public void setMaterial(T material) {
        this.material = material;
    }

    public T getMaterial() {
        return material;
    }

    @Override
    public String toString() {
        return material.toString();
    }
}
```



제네릭(Generic)

❖ 멀티타입 파라미터 – class<K, V>

```
package generic.product;

//T, M 자료형 정의
public class Product<T, M> {

    private T kind;
    private M model;

    public void setKind(T kind) {
        this.kind = kind;
    }

    public T getKind() {
        return kind;
    }

    public void setModel(M model) {
        this.model = model;
    }

    public M getModel() {
        return model;
    }
}
```



제네릭(Generic)

❖ 멀티타입 파라미터 – class<K, V>

```
public class TV {  
    public String making() {  
        return "회사가 TV를 제조합니다.";  
    }  
}
```

```
public class Car {  
    public String making() {  
        return "회사가 자동차를 제조합니다.";  
    }  
}
```



제네릭(Generic)

❖ 멀티타입 파라미터 – class<K, V>

```
public class GenericProduct {  
  
    public static void main(String[] args) {  
  
        //<클래스, 문자열>  
        Product<TV, String> prod1 = new Product<>();  
        TV tv = new TV();  
  
        prod1.setKind(tv);  
        prod1.setModel("SmartTV");  
        System.out.println("모델: " + prod1.getModel());  
        System.out.println(tv.making());  
  
        //<클래스, 문자열>  
        Product<Car, String> prod2 = new Product<>();  
        Car car = new Car();  
  
        prod2.setKind(car);  
        prod2.setModel("전기차");  
        System.out.println("모델: " + prod2.getModel());  
        System.out.println(car.making());  
  
    }  
}
```

모델: SmartTV
회사가 TV를 제조합니다.
모델: 전기차
회사가 자동차를 제조합니다.



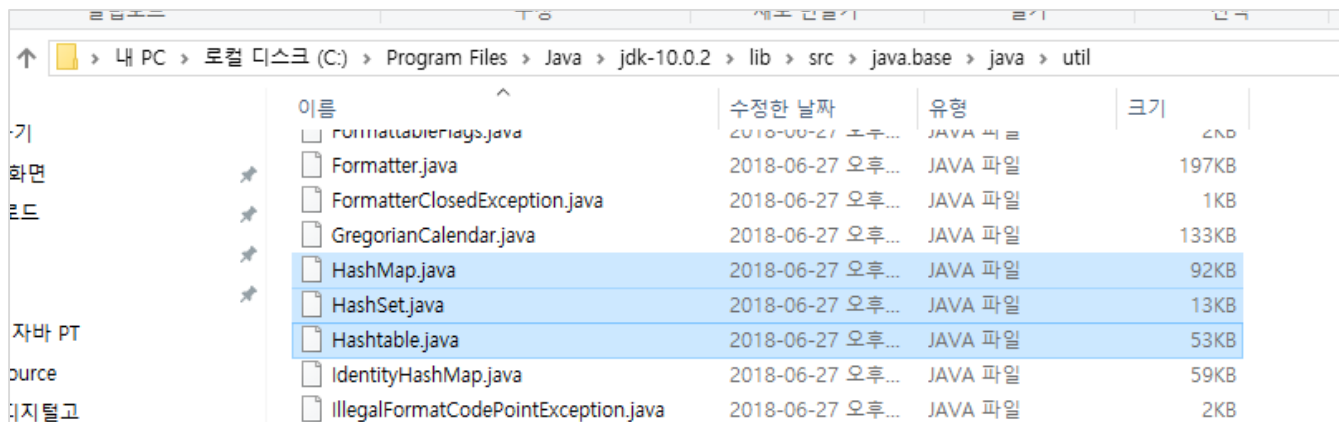
컬렉션 프레임워크

● Collection 프레임워크

- 프로그램 구현에 필요한 자료구조(Data Structure)를 구현해 놓은 라이브러리이다
- 프로그램 실행 중 메모리에 자료를 유지, 관리하기 위해 사용한다.
- java.util 패키지에 구현되어 있음
- 개발에 소요되는 시간을 절약하면서 최적화 된 알고리즘을 사용할 수 있음

● java.util 패키지

- java.util.ArrayList // java.util.HashMap 의 위치하는 곳은 어디일까?



이름	수정된 날짜	유형	크기
Formatter.java	2018-06-27 오후...	JAVA 파일	197KB
FormatterClosedException.java	2018-06-27 오후...	JAVA 파일	1KB
GregorianCalendar.java	2018-06-27 오후...	JAVA 파일	133KB
HashMap.java	2018-06-27 오후...	JAVA 파일	92KB
HashSet.java	2018-06-27 오후...	JAVA 파일	13KB
Hashtable.java	2018-06-27 오후...	JAVA 파일	53KB
IdentityHashMap.java	2018-06-27 오후...	JAVA 파일	59KB
IllegalFormatCodePointException.java	2018-06-27 오후...	JAVA 파일	2KB



컬렉션 프레임워크

● Collection 인터페이스

- 하나의 객체를 관리하기 위한 메서드가 선언된 인터페이스
- 하위에 List와 Set 인터페이스가 있음
- 여러 클래스들이 Collection 인터페이스를 구현함

Module java.base

Package java.util

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, EventSet, List<E>,

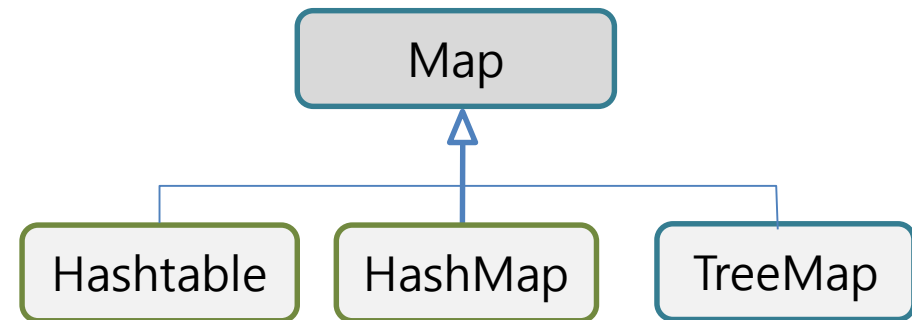
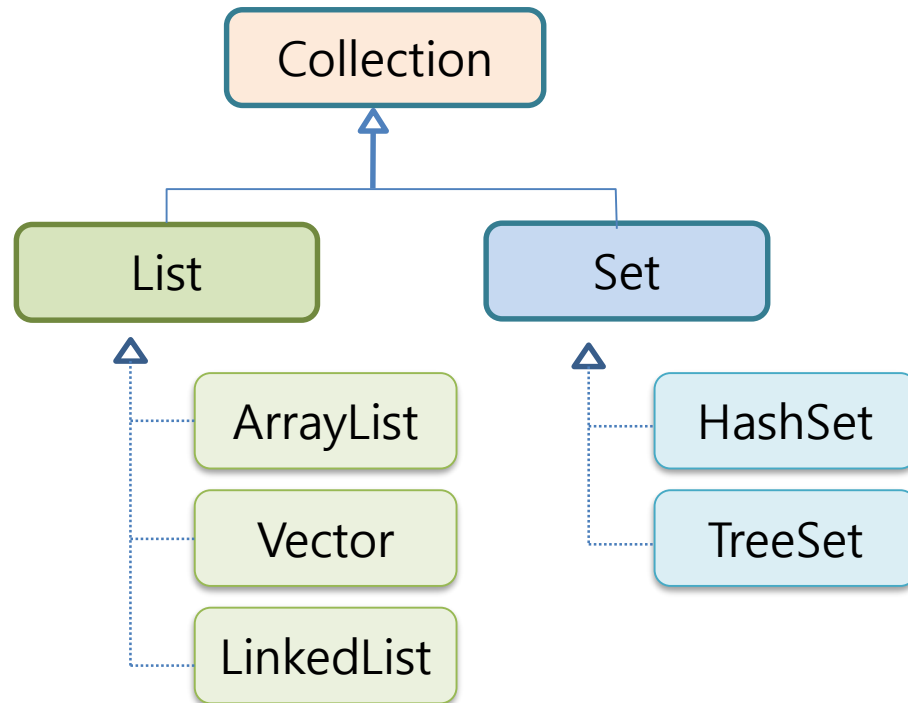
All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, List, Stack, SynchronousQueue, TreeSet, Vector

```
public interface Collection<E>  
    extends Iterable<E>
```



컬렉션 프레임워크



컬렉션 프레임워크

● List와 Set 비교

분류	특 징
List 인터페이스	<ul style="list-style-type: none">- 순서를 유지하고 저장- 중복 저장 가능- 구현클래스 : ArrayList, Vector, LinkedList
Set 인터페이스	<ul style="list-style-type: none">- 순서를 유지하지 않고 저장- 중복 저장 안됨- 구현클래스 : HashSet, TreeSet



컬렉션 프레임워크

● List 인터페이스

- Collection 하위 인터페이스로 배열의 기능을 구현하기 위한 인터페이스이다.
- 객체를 **순서**에 따라 저장하고 관리하는데 필요한 메서드가 선언된 인터페이스
- 구현 클래스로 **ArrayList, Vector, LinkedList** 등이 많이 사용됨

```
List<E> list = new ArrayList<E>
```

ArrayList

0	1	2	3	4	5	6	7	8	9

E 객체 10개를 저장할 수 있는 초기 용량을 가짐

저장용량(capacity)

- 초기:10개, 초기 용량 지정 가능
- 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어난다.
- 객체가 제거되면 바로 뒤 인덱스부터 앞으로 1씩 당겨진다.



List 인터페이스

● List(ArrayList)의 주요 메서드

기능	메서드	설명
객체 추가	add(element)	주어진 객체를 맨 끝에 추가
	add(index, element)	주어진 인덱스에 객체를 추가
객체 검색	contains(object)	주어진 객체가 저장되어 있는지 여부
	get(index)	주어진 인덱스에 저장된 객체를 리턴
	size()	저장되어 있는 전체 객체 수를 리턴
객체 수정	set(index, element)	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 삭제	clear()	저장된 모든 객체 삭제
	remove(index)	주어진 인덱스에 저장된 객체를 삭제



List 인터페이스

● List(ArrayList) Test

```
import java.util.ArrayList;
import java.util.List;

public class ArrayListTest {
    public static void main(String[] args) {
        //List 타입으로 vegeList(ArrayList) 객체 생성
        List<String> vegeList = new ArrayList<>();

        //요소 추가
        vegeList.add("양파");
        vegeList.add("마늘");
        vegeList.add("감자");

        //객체 출력
        System.out.println(vegeList);

        //객체의 개수
        System.out.printf("총 객체수: %d개\n", vegeList.size());

        //특정 요소 검색(인덱싱)
        System.out.println(vegeList.get(0));
    }
}
```



List 인터페이스

● List(ArrayList) Test

```
//특정 위치에 요소 추가 - 1번 위치에 "고추" 추가
vegeList.add(2, "고추");

//전체 객체 요소 출력
for(int i = 0; i < vegeList.size(); i++) {
    String vegetable = vegeList.get(i);
    System.out.print(vegetable + " ");
}
System.out.println();

//객체 요소 수정 - "감자"를 "고구마"로 변경
vegeList.set(3, "고구마");
System.out.println(vegeList);

//요소 삭제 - "마늘" 삭제
//vegeList.remove(1);
vegeList.remove("마늘");

//항상 for문
for(String vegetable : vegeList)
    System.out.print(vegetable + " ");
}
```

[양파, 마늘, 감자]
총 객체수: 3개
양파
양파 마늘 고추 감자
[양파, 마늘, 고추, 고구마]
양파 고추 고구마

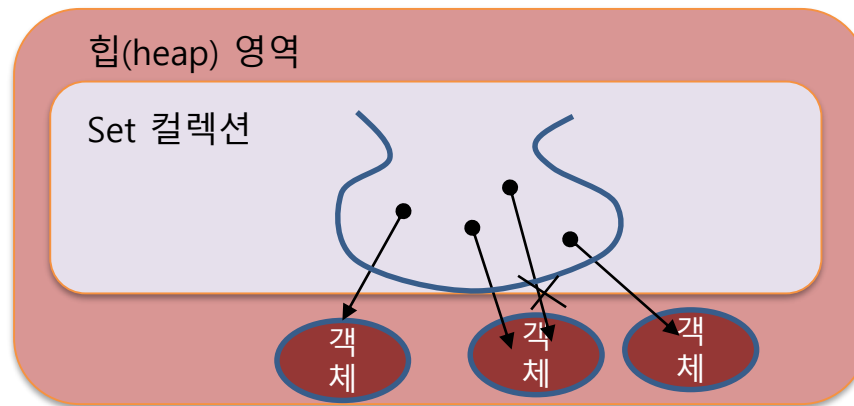


컬렉션 프레임워크

◆ Set 인터페이스

- 특징
 - 수학의 집합에 비유될 수 있다.
 - 저장 순서가 유지되지 않는다.
 - 객체를 중복 저장할 수 없다.
- 구현 클래스

HashSet, LinkedHashSet, TreeSet



컬렉션 프레임워크

◆ Set 인터페이스

● 주요 메소드

기능	메소드	설명
객체 추가	add(element)	주어진 객체를 저장
객체 검색	contains(object)	주어진 객체가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 조사
	iterator()	저장된 객체를 한 번씩 가져오는 반복자 리턴
	size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	clear()	저장된 모든 객체 삭제
	remove(object)	주어진 객체를 삭제



Set 인터페이스

- 객체 추가, 찾기, 삭제

```
Set<String> set =...;  
set.add("홍길동");  
set.add("임꺽정");  
set.remove("홍길동")
```

- Set컬렉션은 인덱스로 객체를 검색해서 가져오는 메소드가 없다.
대신, 전체 객체를 대상으로 한번씩 반복해서 가져오는 **반복자(iterator)**를 제공한다.

```
Iterator<String> iterator = set.iterator();  
while(iterator.hasNext()) {  
    String element = iterator.next();  
}
```



Set 인터페이스

◆ Collection 요소를 순회하는 Iterator

- 순서가 없는 Set 인터페이스를 구현한 경우에는 **get(i)** 메서드를 사용할 수 없다.
이때 **Iterator** 클래스의 **iterator()** 메서드를 호출하여 참조한다.

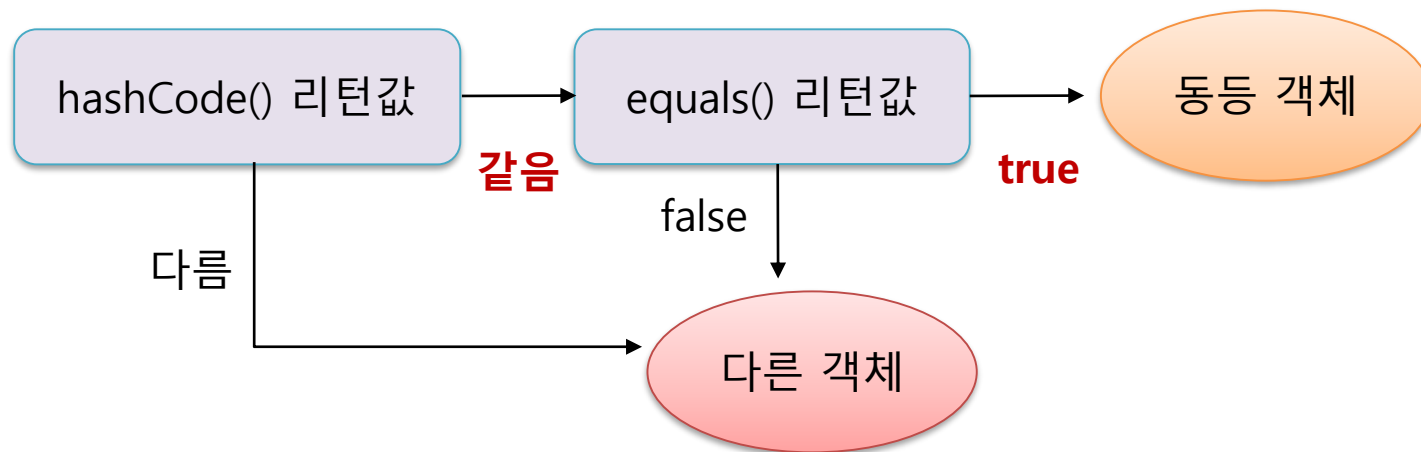
리턴 타입	메소드명	설명
Boolean	hasNext()	가져올 객체가 있으면 true, 없으면 false 리턴
E	next()	컬렉션에서 하나의 객체를 가져온다.
void	remove()	Set 컬렉션에서 객체를 제거한다.



Set 인터페이스

◆ HashSet 클래스

- HashSet은 Set 인터페이스의 구현 클래스이다.
- 특징
 - 동일 객체 및 동등 객체는 중복 저장하지 않는다.
 - 순서없이 저장
 - 동등 객체 판단 방법



Set 인터페이스

◆ HashSet을 이용한 자료 관리

```
public class HashSetTest {  
    public static void main(String[] args) {  
  
        //Set 타입으로 HashSet 객체 생성  
        Set<String> set = new HashSet<>();  
  
        //요소 추가  
        set.add("Java");  
        set.add("C++");  
        set.add("Python");  
        set.add("Java");  
        set.add("JDBC");  
  
        //객체 출력 - 순서가 없고, 중복 불가  
        System.out.println(set);  
  
        //객체의 크기  
        int size = set.size();  
        System.out.println("총 객체수: " + size);  
    }  
}
```



Set 인터페이스

◆ HashSet을 이용한 자료 관리

```
//특정 요소 검색
if(set.contains("JDBC")) {
    System.out.println("JDBC");
}

//전체 요소 출력
Iterator<String> ir = set.iterator(); //반복자 객체 생성
while(ir.hasNext()) { //요소를 순회하면서
    String element = ir.next(); //요소 1개 가져오기
    System.out.println("\t" + element);
}
System.out.println("=====");

//요소 삭제
if(set.contains("C++")) {
    set.remove("C++");
}

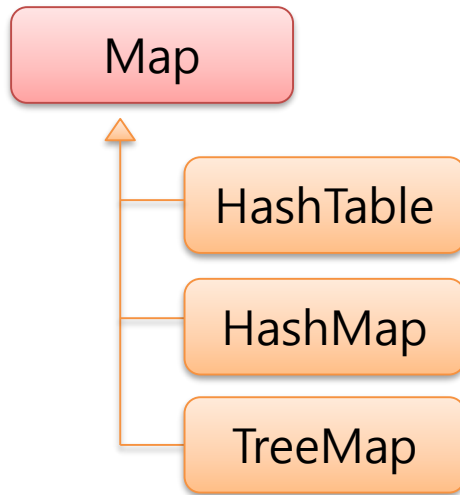
//항상 for
for(String element : set)
    System.out.println("\t" + element);
}
```

```
[Java, C++, JDBC, Python]
총 객체수: 4
JDBC
        Java
        C++
        JDBC
        Python
=====
        Java
        JDBC
        Python
```



Map 인터페이스

◆ Map 인터페이스



Module java.base

Package java.util

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Known Subinterfaces:

Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>,

All Known Implementing Classes:

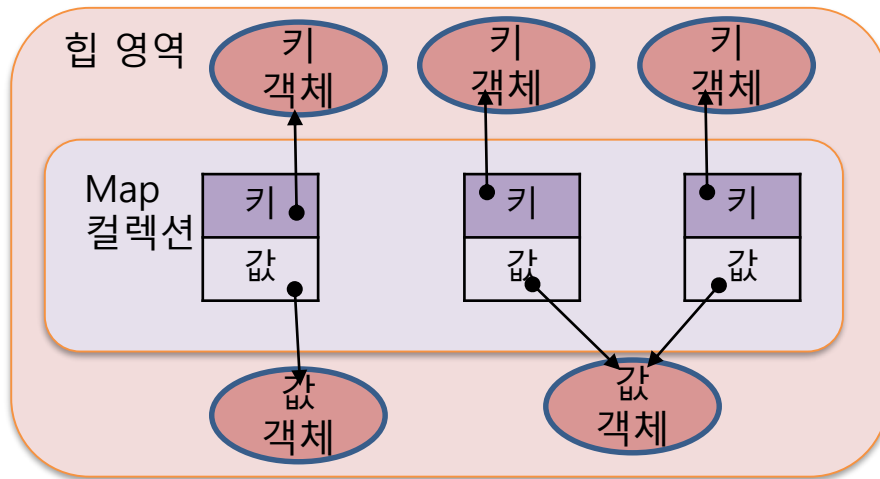
AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, C
Provider, RenderingHints, SimpleBindings, TabularDataSuppo

```
public interface Map<K,V>
```



Map 인터페이스

◆ Map 인터페이스



분류	특 징
Map 인터페이스	<ul style="list-style-type: none">- 키(key)와 값(Value)의 쌍으로 저장- 키는 중복 저장 안되고, 값은 가능- 구현 클래스 : HashMap, Hashtable, TreeMap



Map 인터페이스

◆ Map 인터페이스

- Key-value pair의 객체를 관리하는데 필요한 메서드가 정의 됨
- Key를 이용하여 값을 저장하거나 검색, 삭제 할때 사용하면 편리함

```
Map<K, V> map = new HashMap<K, V>();
```

```
Map<String, Integer> map = new HashMap<>();
```

기능	메소드	설명
객체 추가	put(key, value)	주어진 키로 값을 저장
객체 검색	contains(object key)	주어진 키가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 조사
	size()	저장되어 있는 키의 총 수를 리턴
객체 삭제	clear()	저장된 모든 키와 값을 삭제
	remove(Object key)	주어진 키와 일치하는 객체를 삭제하고 값을 리턴



Map 인터페이스

◆ Map을 이용한 자료 관리

```
public class HashMapTest {  
  
    public static void main(String[] args) {  
        //이름과 점수를 저장할 Map 객체 생성  
        Map<String, Integer> map = new HashMap<>();  
  
        //요소 저장  
        map.put("강감찬", 95);  
        map.put("홍길동", 75);  
        map.put("이순신", 80);  
  
        //요소의 개수  
        System.out.println("요소 수 - " + map.size() + "개");  
  
        //요소 검색  
        System.out.println("홍길동의 점수: " + map.get("홍길동"));  
  
        //요소 수정 - 키는 중복되지 않으므로 기존 값을 덮어 쓴다.  
        map.put("이순신", 85);  
    }  
}
```



Map 인터페이스

◆ Map을 이용한 자료 관리

```
//반복자 객체로 출력(순서 없이 출력됨)
Iterator<String> iterator = map.keySet().iterator();
while(iterator.hasNext()) {
    String key = iterator.next();
    Integer value = map.get(key);
    System.out.println(key + " : " + value);
}

//요소 삭제
if(map.containsKey("홍길동")) {
    map.remove("홍길동");
}

//요소의 개수
System.out.println("요소 수 - " + map.size() + "개");
}
```

요소 수 - 3개
홍길동의 점수: 75
홍길동 : 75
강감찬 : 95
이순신 : 85
요소 수 - 2개



Map 인터페이스

◆ 컴퓨터 용어 사전 만들기

- ✓ 간단한 영어 단어 사전 프로그램을 구현함
- ✓ HashMap을 사용하여 단어와 그 정의를 저장함
- ✓ 사용자 입력을 통해 단어를 검색할 수 있는 기능을 제공함

```
=====
프로그램을 종료하려면 q 또는 Q를 입력하세요
=====
검색할 단어를 입력하세요: 비트
정보 기술에서 데이터의 가장 작은 단위로, 0 또는 1의 값을 가진다.
검색할 단어를 입력하세요: 버그
프로그램이 적절하게 동작하는데 실패하거나 오류가 발생하는 코드
검색할 단어를 입력하세요: q
프로그램을 종료합니다.
```



Map 인터페이스

◆ 컴퓨터 용어 사전 만들기

```
public class Dictionary {  
  
    public static void main(String[] args) {  
  
        //Map 자료구조 객체 생성  
        Map<String, String> dic = new HashMap<>();  
  
        //단어와 정의 저장  
        dic.put("이진수", "컴퓨터가 사용하는 0과 1만으로 이루어진 수");  
        dic.put("비트", "정보 기술에서 데이터의 가장 작은 단위로, 0 또는 1의 값을 가진다.");  
        dic.put("버그", "프로그램이 적절하게 동작하는데 실패하거나 오류가 발생하는 코드");  
        dic.put("알고리즘", "어떤 문제를 해결하기 위해 정해진 절차");  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("=====");  
        System.out.println("프로그램을 종료하려면 q 또는 Q를 입력하세요");  
        System.out.println("=====");  
    }  
}
```



Map 인터페이스

◆ 컴퓨터 용어 사전 만들기

```
while(true) {
    System.out.print("검색할 단어를 입력하세요: ");
    String word = sc.nextLine(); //단어 입력

    //검색 종료
    if(word.toLowerCase().equals("q")) {
        System.out.println("프로그램을 종료합니다.");
        break;
    }

    //정의된 단어 가져오기
    String definition = dic.get(word);

    if(definition != null) {
        System.out.println(definition);
    }else {
        System.out.println(word + "는 없는 단어입니다.");
    }
}
sc.close();
}
```



회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

```
package collection.member;

public class Member {
    private int memberId;      //회원 아이디
    private String memberName; //회원 이름

    public Member(int memberId, String memberName) {
        this.memberId = memberId;
        this.memberName = memberName;
    }

    public int getMemberId() {
        return memberId;
    }

    public void setMemberId(int memberId) {
        this.memberId = memberId;
    }
}
```



회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

```
public String getMemberName() {  
    return memberName;  
}  
  
public void setMemberName(String memberName) {  
    this.memberName = memberName;  
}  
  
@Override  
public String toString() {  
    return memberName + " 회원님의 아이디는 " + memberId + "입니다.";  
}
```



회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

```
public class MemberArrayList {  
    private ArrayList<Member> arrayList;  
  
    public MemberArrayList() { //클래스 사용시 arralist 객체 생성  
        arrayList = new ArrayList<Member>();  
    }  
  
    //회원 추가 메서드 정의  
    public void addMember(Member member) {  
        arrayList.add(member);  
    }  
  
    //회원 조회  
    public void showAllMember(){  
        for(int i=0; i<arrayList.size(); i++) {  
            Member member = arrayList.get(i);  
            System.out.println(member);  
        }  
    }  
}
```



회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

```
//회원 삭제
public boolean removeMember(int memberId) {
    for(int i=0; i<arrayList.size(); i++) {
        //이미 등록된 memberId를 dbId에 저장함
        int dbId = arrayList.get(i).getMemberId();
        if(dbId == memberId) { //dbId가 외부 입력한 memberId와 일치하면
            arrayList.remove(i); //해당 객체 삭제
            return true;
        }
    }
    System.out.println(memberId + "가 존재하지 않습니다.");
    return false;
}
```



회원 관리 프로그램

MemberArrayListTest.java

추신수 회원님의 아이디는 1001입니다.
손흥민 회원님의 아이디는 1002입니다.
박인비 회원님의 아이디는 1003입니다.
김연아 회원님의 아이디는 1004입니다.

1005가 존재하지 않습니다.

추신수 회원님의 아이디는 1001입니다.
손흥민 회원님의 아이디는 1002입니다.
김연아 회원님의 아이디는 1004입니다.

```
MemberArrayList memberArrayList = new MemberArrayList();

Member chu = new Member(1001, "추신수");
Member son = new Member(1002, "손흥민");
Member park = new Member(1003, "박인비");
Member kim = new Member(1004, "김연아");

//회원 추가
memberArrayList.addMember(chu);
memberArrayList.addMember(son);
memberArrayList.addMember(park);
memberArrayList.addMember(kim);

//회원 전체 목록
memberArrayList.showAllMember();

System.out.println("-----");

//회원 삭제
memberArrayList.removeMember(1003);
memberArrayList.removeMember(1005);    //존재하지 않는 아이디

memberArrayList.showAllMember();
```



Map 인터페이스

◆ HashMap을 활용한 회원관리 프로그램

```
package collection.member;

import java.util.HashMap;
import java.util.Iterator;

public class MemberHashMap {
    HashMap<Integer, Member> hashMap;

    public MemberHashMap() {
        hashMap = new HashMap<>();
    }

    public void addMember(Member member) {
        //key:memberId, value:member
        hashMap.put(member.getMemberId(), member);
    }
}
```



Map 인터페이스

◆ HashMap을 활용한 회원관리 프로그램

```
//회원 목록
public void showAllMember() {
    Iterator<Integer> ir = hashMap.keySet().iterator();
    while(ir.hasNext()) {
        int key = ir.next(); //key값을 가져와서
        Member member = hashMap.get(key); //키로부터 value 가져오기
        System.out.println(member);
    }
    System.out.println();
}

//회원 삭제
public boolean removeMember(int memberId) {
    if(hashMap.containsKey(memberId)) { //입력받은 회원아이디가 존재한다면
        hashMap.remove(memberId); //해당 회원 삭제
        return true;
    }
    System.out.println(memberId + "가 존재하지 않습니다.");
    return false;
}
```



Map 인터페이스

```
public class MemberHashMapTest {  
  
    public static void main(String[] args) {  
        //mHashMap 객체 생성  
        MemberHashMap mHashMap = new MemberHashMap();  
  
        //회원 추가  
        mHashMap.addMember(new Member(1001, "삼성전자"));  
        mHashMap.addMember(new Member(1002, "LG전자"));  
        mHashMap.addMember(new Member(1003, "네이버"));  
        mHashMap.addMember(new Member(1004, "카카오"));  
        mHashMap.addMember(new Member(1002, "현대자동차"));  
  
        //회원 목록 조회  
        mHashMap.showAllMember();  
        System.out.println("=====");  
  
        //회원 삭제  
        mHashMap.removeMember(1001);  
        mHashMap.removeMember(1005);  
  
        //조회  
        mHashMap.showAllMember();  
    }  
}
```

삼성전자 회원님의 아이디는 1001입니다.
현대자동차 회원님의 아이디는 1002입니다.
네이버 회원님의 아이디는 1003입니다.
카카오 회원님의 아이디는 1004입니다.

=====

회원 아이디 1005가 존재하지 않습니다.
현대자동차 회원님의 아이디는 1002입니다.
네이버 회원님의 아이디는 1003입니다.
카카오 회원님의 아이디는 1004입니다.

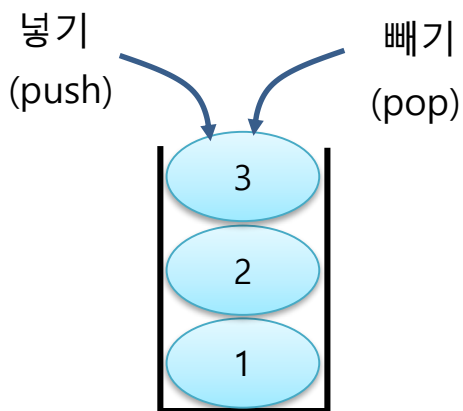


스택(Stack)과 큐(Queue)

❖ Stack 클래스

- 후입선출(LIFO : Last in First Out) 구조 – (응용 예: JVM 스택 메모리, 접시닦이, 게임 무르기)
- 주요 메소드

메소드명	설명
push	주어진 객체를 스택에 넣는다.
pop()	스택의 맨 위 객체를 가져온다. 객체를 스택에서 제거한다.
isEmpty()	스택의 객체가 비어있는지 여부



```
Module java.base
Package java.util

Class Stack<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>
        java.util.Stack<E>

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>,

public class Stack<E>
  extends Vector<E>
```



스택(Stack)과 큐(Queue)

❖ Stack 클래스로 동전 넣고 빼기 구현

```
package collection.list;

import java.util.Stack;

class Coin{
    private int value;

    public Coin(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

꺼내온 동전	:	10원
꺼내온 동전	:	50원
꺼내온 동전	:	100원
꺼내온 동전	:	500원



스택(Stack)과 큐(Queue)

```
public class StackCoinTest {
    public static void main(String[] args) {
        Stack<Coin> coinBox = new Stack<>();

        //동전 객체 생성
        Coin coin500 = new Coin(500);
        Coin coin100 = new Coin(100);
        Coin coin50 = new Coin(50);
        Coin coin10 = new Coin(10);

        //스택에서 동전 넣기(순서 : 500 - 100 - 50 - 10)
        coinBox.push(coin500);
        coinBox.push(coin100);
        coinBox.push(coin50);
        coinBox.push(coin10);

        //스택에서 동전 빼기
        //System.out.println(coinBox.pop().getValue());

        while(!coinBox.isEmpty()) { //순서 : 10 - 50 - 100 - 500
            Coin coin = coinBox.pop();
            System.out.println("꺼내온 동전 : " + coin.getValue() + "원");
        }
    }
}
```



스택(Stack)

❖ ArrayList를 활용하여 Stack 구현

```
class MyStack{
    private ArrayList<String> arrayStack;

    public MyStack() {
        arrayStack = new ArrayList<>();
    }

    //자료 추가(넣기)
    public void push(String data) {
        arrayStack.add(data);
    }

    //자료 삭제(빼기)
    public String pop() {
        int len = arrayStack.size();
        if(len==0) {
            System.out.println("스택이 비었습니다.");
            return null;
        }
        return arrayStack.remove(len-1);
    }
}
```



스택(Stack)

❖ ArrayList를 활용하여 Stack 구현

```
public class ArrayStackTest {  
  
    public static void main(String[] args) {  
        MyStack stack = new MyStack();  
  
        //객체 넣기  
        stack.push("돼지");  
        stack.push("닭");  
        stack.push("소");  
  
        //객체 빼기  
        System.out.println(stack.pop());  
        System.out.println(stack.pop());  
        System.out.println(stack.pop());  
        System.out.println(stack.pop());  
    }  
}
```

소
닭
돼지
스택이 비었습니다.
null

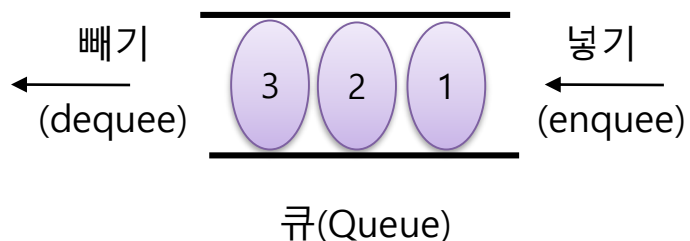


스택(Stack)과 큐(Queue)

❖ Queue 인터페이스

- 선입선출(FIFO : First in First Out) 구조 – (응용 예: 버스정류장 줄서기, 운영체제 메시지큐)
- 주요 메소드

메소드명	설명
offer()	주어진 객체를 넣는다.
poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.
isEmpty()	스택의 객체가 비어있는지 여부



Module java.base

Package java.util

Interface Queue<E>

Type Parameters:

E - the type of elements held in this queue

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Subinterfaces:

BlockingDeque<E>, BlockingQueue<E>, Deque<E>,



큐(Queue)

❖ ArrayList활용하여 Queue(큐) 구현

```
class MyQueue{
    private ArrayList<String> arrayQueue = new ArrayList<>();

    //큐의 맨 뒤에 추가
    public void enqueue(String data) {
        arrayQueue.add(data);
    }

    //큐의 맨 앞에서 꺼냄
    public String dequeue() {
        int len = arrayQueue.size();
        if(len==0) {
            System.out.println("큐가 비었습니다.");
            return null;
        }
        return arrayQueue.remove(0);
    }
}
```



큐(Queue)

❖ ArrayList활용하여 Queue(큐) 구현

```
public class ArrayQueueTest {  
    public static void main(String[] args) {  
        MyQueue queue = new MyQueue();  
        queue.enqueue("A");  
        queue.enqueue("B");  
        queue.enqueue("C");  
  
        System.out.println(queue.dequeue());  
        System.out.println(queue.dequeue());  
        System.out.println(queue.dequeue());  
        System.out.println(queue.dequeue());  
    }  
}
```

A
B
C
큐가 비었습니다.
null

