14장. 컬렉션 프레임워크

collection

제네릭(Generic)

제네릭 프로그래밍

- **어떤 값이 하나의 자료형이 아닌 여러 자료형을 사용할 수 있도록 프로그래밍 하는 것.**

  Java 5부터 제네릭(Generic) 타입이 새로 추가되었는데, 제네릭 타입을 이용함으로써 잘못된 타입이 사용될 수 있는 문제를 컴파일 과정에서 제거할 수 있게 되었다.
  또한, 비제네릭 코드는 불필요한 타입 변환을 하므로 프로그램 성능에 악영향을 미친다.

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

```
public class 클래스명<T>{....}
```

제네릭(Generic)

제네릭 프로그래밍

```
public class GenericTest {

    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("grape");
        list.add("egg");
        list.add("coffee");

        String str1 = list.get(0);
        System.out.println(str1);

        //왜 제네릭 프로그래밍을 하는가? 타입을 정해주지 않으면 Object로 됨
        ArrayList cart = new ArrayList<>();
        cart.add("grape");
        cart.add("egg");
        cart.add("coffee");

        String str2 = (String)list.get(0);  //다운캐스팅 - 형변환 해야함
        System.out.println(str2);
    }
}
```

제네릭(Generic)

제네릭 타입

제네릭 타입은 **타입(type)을 파라미터**로 가지는 클래스를 말한다.

class 클래스명<T>{....}

```
package genericex.box;

public class Box<T> {
    private T type;

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

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

제네릭(Generic)

제네릭 타입

```java
public class GenericBoxTest {

    public static void main(String[] args) {
        //String형 사용
        Box<String> box1 = new Box<>();
        box1.set("행운을 빌어요!");
        String msg = box1.get();
        System.out.println(msg);

        //Integer형 사용
        Box<Integer> box2 = new Box<>();
        box2.set(10);
        Integer num = box2.get();
        System.out.println(num);

        //Apple 클래스 사용
        Box<Apple> box3 = new Box<>();
        box3.set(new Apple());
        Apple apple = box3.get();
        System.out.println(apple);
    }
}
```

```java
public class Apple {
    String name;

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

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

제네릭(Generic)

비제네릭 타입

```java
package genericex.box2;

public class Box{
    private Object obj;

    public void set(Object obj) {
        this.obj = obj;
    }

    public Object get() {
        return obj;
    }
}
```

```java
public class GenericBoxTest {

    public static void main(String[] args) {

        //String type
        Box box = new Box();
        box.set("Good Luck!");
        String msg = (String) box.get();
        System.out.println(msg);

        //Class type
        box.set(new Apple("사과"));
        Apple apple = (Apple) box.get();
        System.out.println(apple);
    }
}
```

제네릭(Generic)

제네릭 프로그래밍 – 3D 프린터 예제.

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

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

    public T getMeterial() {
        return material;
    }

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

제네릭(Generic)

제네릭 프로그래밍 – 3D 프린터 예제.

```
public class Plastic {

    @Override
    public String toString() {
        return "재료는 plastic입니다.";
    }
}
```

```
public class Powder {

    @Override
    public String toString() {
        return "재료는 powder입니다.";
    }
}
```

제네릭(Generic)

GenericPrinter Test

```
//Powder 자료형 사용
GenericPrinter<Powder> powderPrinter = new GenericPrinter<>();

powderPrinter.setMaterial(new Powder());
System.out.println(powderPrinter);

//Plastic 자료형 사용
GenericPrinter<Plastic> plasticPrinter = new GenericPrinter<>();
plasticPrinter.setMaterial(new Plastic());
System.out.println(plasticPrinter);
```

```
재료는 powder입니다.
재료는 plastic입니다.
```

제네릭(Generic)

멀티타입 파라미터 – class<K, V>

```
package genericex.product2;

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 Car {

    public void making() {
        System.out.println("회사가 자동차를 제조합니다.");
    }

}
```

```
public class TV {

    public void making() {
        System.out.println("회사가 TV를 제조합니다.");
    }
}
```

제네릭(Generic)

```
public class GenericProduct {

    public static void main(String[] args) {
        //<클래스, 문자열> 타입
        Product<TV, String> prod1 = new Product<>();
        TV tv = new TV();
        prod1.setKind(tv);
        prod1.setModel("스마트TV");
        String tvModel = prod1.getModel();
        tv.making();
        System.out.println("모델: " + tvModel);

        //<클래스, 문자열> 타입
        Product<Car, String> prod2 = new Product<>();
        Car car = new Car();
        prod2.setKind(car);
        prod2.setModel("전기차");
        car.making();
        System.out.println("모델: " + prod2.getModel());
    }
}
```

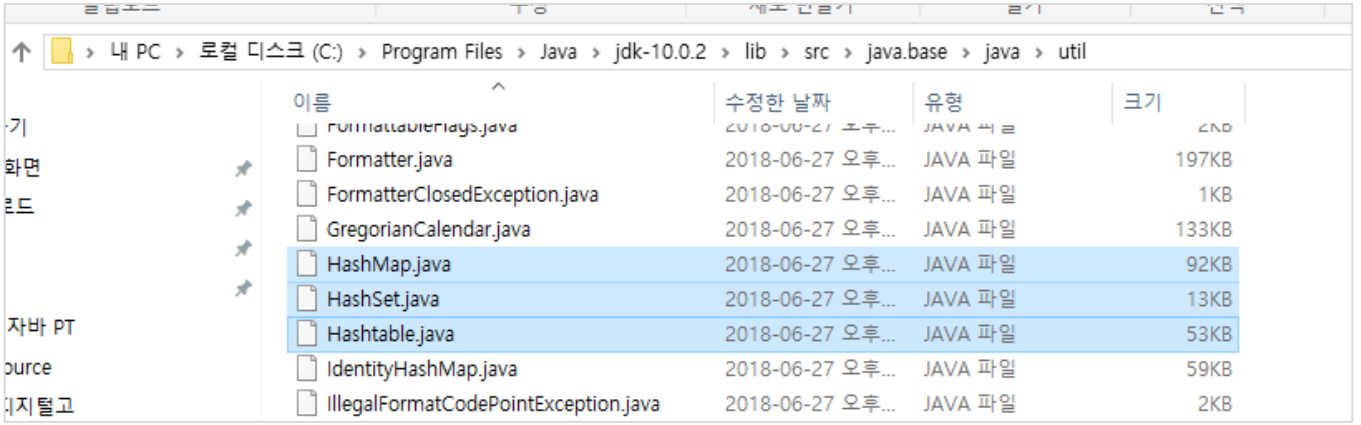
컬렉션 프레임워크

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?



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:

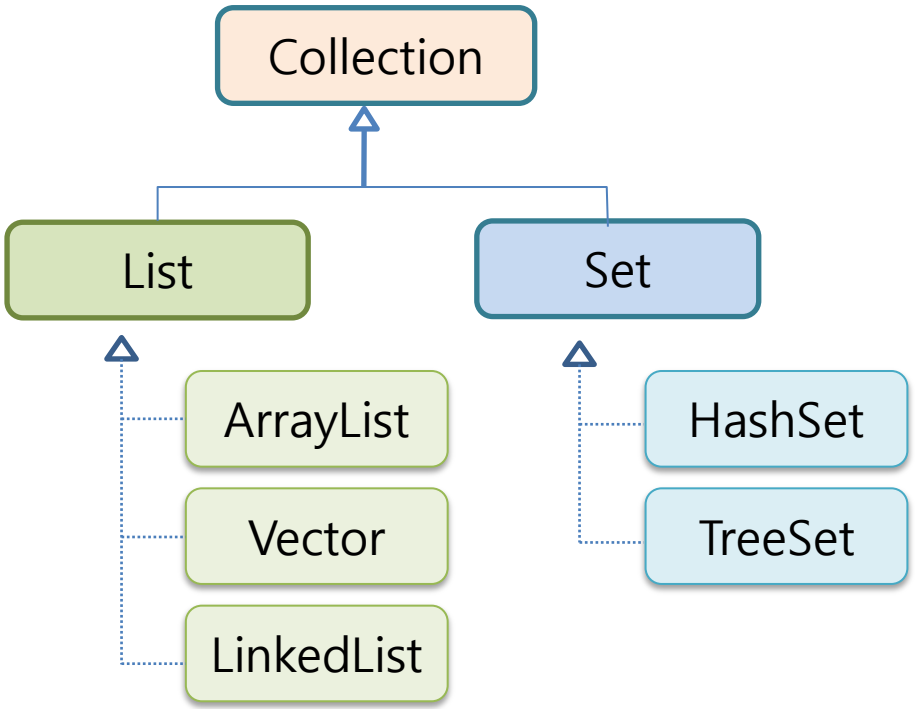
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, EventSet, List<E>,

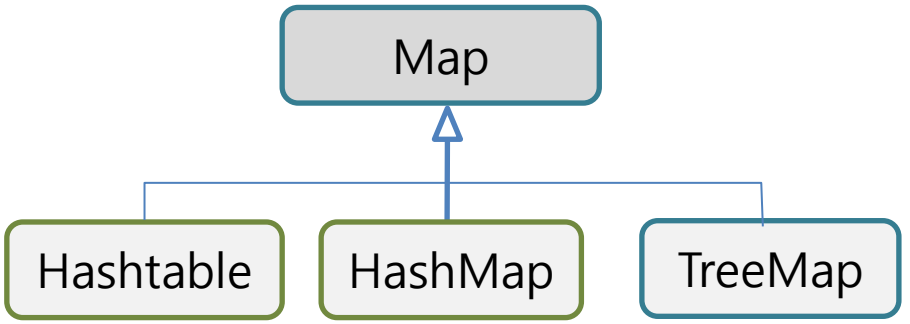
All Known Implementing Classes:

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

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

컬렉션 프레임워크


Collection
List
Set
ArrayList
Vector
LinkedList
HashSet
TreeSet


Map
Hashtable
HashMap
TreeMap

컬렉션 프레임워크

List와 Set 비교

분류	특 징
List 인터페이스	- 순서를 유지하고 저장 - 중복 저장 가능 - 구현클래스 : ArrayList, Vector, LinkedList
Set 인터페이스	- 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

- **List의 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

```java
package collection.list;

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

public class ArrayListTest {

    public static void main(String[] args) {
        List<String> vegeList = new ArrayList<>();

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

        //특정 위치에 객체 추가
        vegeList.add(2, "고추");

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

        //특정 객체 가져오기
        System.out.println(vegeList.get(1));

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

        //객체 수정
        vegeList.set(0, "상추");
        System.out.println();

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

        //객체 삭제
        vegeList.remove(3);
        System.out.println();

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

Vector 클래스

◆ 벡터(Vector) 클래스

- ArrayList와 동일한 자료구조를 가지고 있다.
- 멀티 스레드로 동작하므로 안전하게 객체를 추가, 삭제 할 수 있다.
  (하나의 스레드가 메소드를 실행 완료해야만 다른 스레드가 메서드를 실행할 수 있다.)

```
List<E> list = new Vector<E>();
```

Vector 클래스

◆ 벡터(Vector) 클래스

```java
package collection.list;

import java.util.Vector;

public class VectorTest {

    public static void main(String[] args) {
        Vector<String> vegeList = new Vector<>();

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

        //특정 위치에 객체 추가
        vegeList.add(2, "고추");
```

Vector 클래스

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

        //특정 객체 가져오기
        System.out.println(vegeList.get(1));

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

        //객체 수정
        vegeList.set(0, "상추");

        //객체 삭제
        vegeList.remove(3);
        System.out.println();

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

```
총 객체수 : 4개
마늘
양파 마늘 고추 감자
상추 마늘 고추
```

Vector 클래스

◆ 벡터(Vector) 클래스 예제

```
package collection.vector;

public class Board {

    String subject;    //제목
    String content;    //내용
    String writer;     //글쓴이

    public Board(String subject, String content, String writer) {
        this.subject = subject;
        this.content = content;
        this.writer = writer;
    }
}
```

Vector 클래스

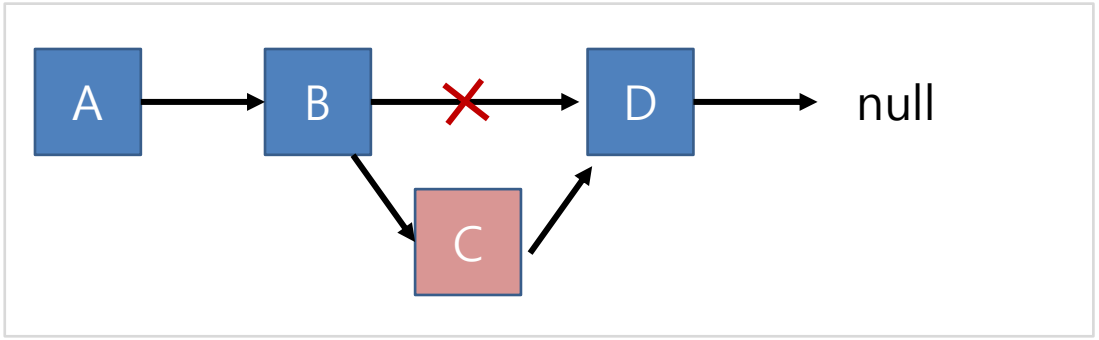
```
public class VectorSample {

    public static void main(String[] args) {

        List<Board> list = new Vector<>();

        list.add(new Board("제목1", "내용1", "글쓴이1"));
        list.add(new Board("제목2", "내용2", "글쓴이2"));
        list.add(new Board("제목3", "내용3", "글쓴이3"));
        list.add(new Board("제목4", "내용4", "글쓴이4"));
        list.add(new Board("제목5", "내용5", "글쓴이5"));

        list.remove(2);  //2번 인덱스 삭제
        list.remove(3);

        for(int i=0; i<list.size(); i++) {
            Board board = list.get(i);
            System.out.println(board.subject +
                    "\t" +  board.content + "\t" + board.writer);
        }
    }
}
```

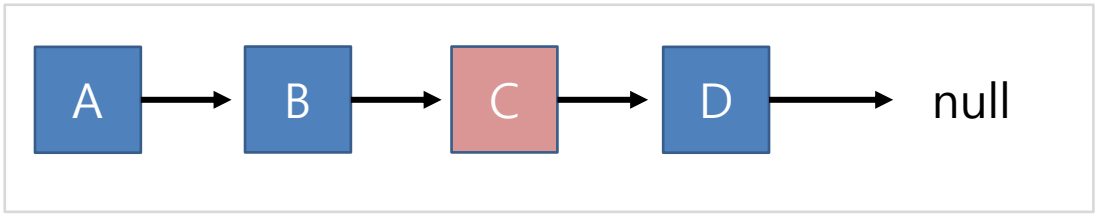
```
제목1    내용1    글쓴이1
제목2    내용2    글쓴이2
제목4    내용4    글쓴이4
```

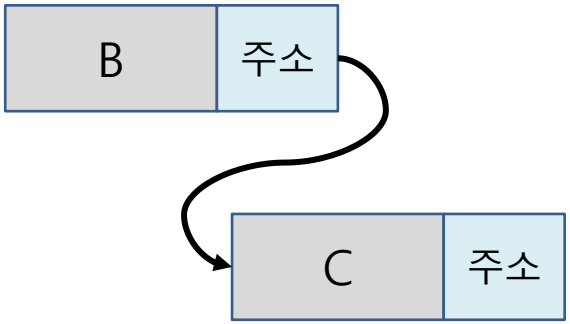
LinkedList 클래스

◆ 링크드 리스트(LinkedList) 클래스

- 링크드 리스트의 각 요소는 다음 요소를 가리키는 주소값을 가지고 물리적인 메모리는 떨어져 있더라도 논리적으로는 앞뒤 순서가 있다.
- 배열은 자료를 삽입 삭제시 공간을 비워서 뒤 요소를 밀고 그 자리에 놓지만, 링크드 리스트는 주소값만 변경하여 자료이동이 발생하지 않음
- ArrayList 보다 중간에 자료를 넣고 제거하는 데 시간이 적게 걸리고 크기를 동적으로 증가시킬수 있다.







LinkedList 클래스

```
public class LinkedListExample2 {

    public static void main(String[] args) {

        List<String> list1 = new ArrayList<>();
        List<String> list2 = new LinkedList<>();

        long startTime, endTime;

        startTime = System.nanoTime();
        for(int i=0; i<10000; i++) {
            list1.add(0, String.valueOf(i));
        }
        endTime = System.nanoTime();
        System.out.println("ArrayList 걸린시간: " + (endTime-startTime) + " ns");

        startTime = System.nanoTime();
        for(int i=0; i<10000; i++) {
            list2.add(0, String.valueOf(i));
        }
        endTime = System.nanoTime();
        System.out.println("LinkedList 걸린시간: " + (endTime-startTime) + " ns");
    }
}
```

```
ArrayList 걸린시간: 6436000 ns
LinkedList 걸린시간: 2300400 ns
```

LinkedList 클래스

```java
public class LinkedListExample1 {

    public static void main(String[] args) {
        //List 타입으로 LinkedList 객체 생성
        List<String> myList = new LinkedList<>();

        //객체 추가
        myList.add("A");
        myList.add("B");
        myList.add("C");

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

        //특정 위치에 객체 추가
        myList.add(2, "D");
        System.out.println(myList);

        //요소 삭제
        myList.remove(1);
        System.out.println(myList);

        //요소 전체 출력
        for(int i=0; i<myList.size(); i++) {
            String list = myList.get(i);
            System.out.print(list + " ");
        }
    }
```

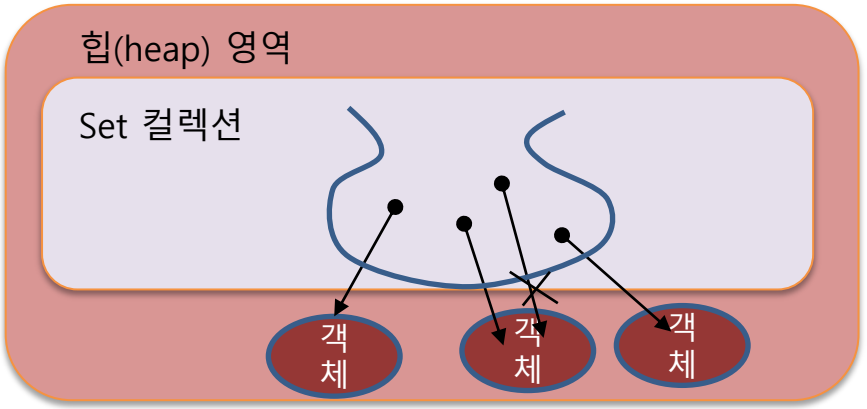
```
[A, B, C]
[A, B, D, C]
[A, D, C]
A D C
```

컬렉션 프레임워크

◆ Set 인터페이스

- 특징
  - 수학의 집합에 비유될 수 있다.
  - 저장 순서가 유지되지 않는다.
  - 객체를 중복 저장할수 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.

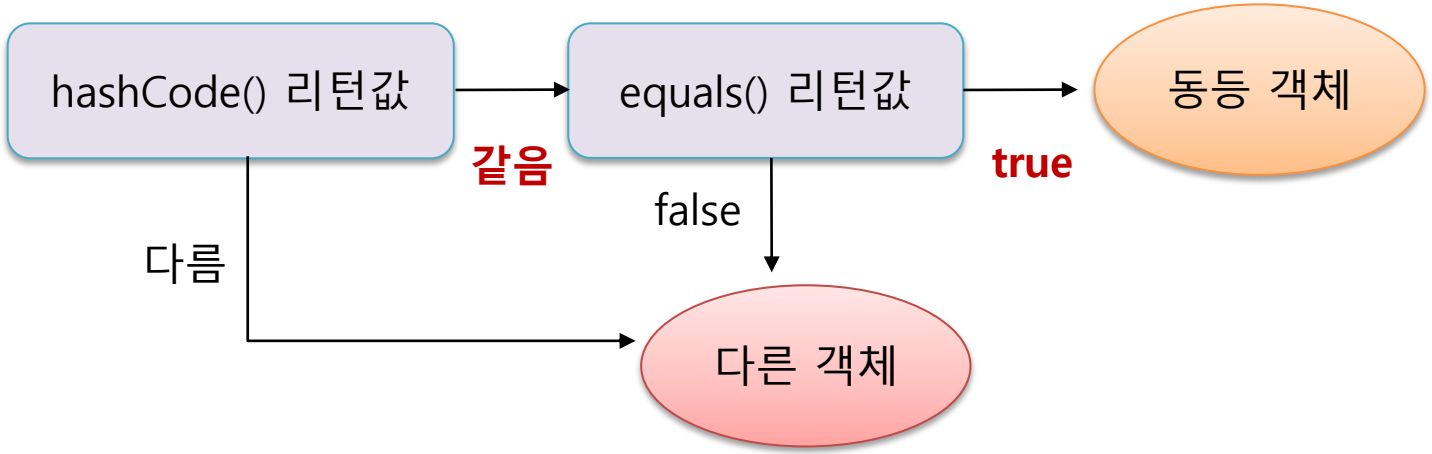
  이때 **Iterator 클래스**의 **iterator() 메서드**를 호출하여 참조한다.

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

```java
package collection.set;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class HashSetExample {

    public static void main(String[] args) {

        Set<String> set = new HashSet<>();

        //요소 저장
        set.add("Java");
        set.add("JDBC");
        set.add("JSP/Servlet");
        set.add("Java");
        set.add("MyBatis");

        int size = set.size();
        System.out.println("총 객체수: " + size);
```

```
총 객체수: 4
        Java
        JDBC
        MyBatis
        JSP/Servlet
총 객체수: 3
        Java
        MyBatis
        JSP/Servlet
비어 있음
```

Set 인터페이스

```
        Iterator<String> iterator = set.iterator(); //반복자 얻기
        while(iterator.hasNext()) {                 //객체 수만큼 반복
            String element = iterator.next();  //1개의 객체 가져옴
            System.out.println("\t" + element);
        }

        //요소 삭제
        set.remove("JDBC");
        System.out.println("총 객체수: " + set.size());

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

        //모든 요소 삭제
        set.clear();
        if(set.isEmpty()) {
            System.out.println("비어 있음");
        }
}
```

Set 인터페이스

```
public class Student {
    String name;
    int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public int hashCode() {              //name과 age 값이 같으면
        return name.hashCode() + age;    //동일한 hashCode를 리턴
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Student) {           //name과 age 값이 같으면
            Student student = (Student)obj;    //true 리턴
            if(student.name.equals(name) && (student.age==age)) {
                return true;
            }
        }
        return false;
    }
}
```

Set 인터페이스

```
public class HashSetExample2 {

    public static void main(String[] args) {

        Set<Student> set = new HashSet<>();

        set.add(new Student("오지능", 30));  //인스턴스는 다르지만 내용이 같으므로
        set.add(new Student("오지능", 30));  //객체 1개만 저장

        System.out.println("총 객체수: " + set.size());

    }
}
```

```
총 객체수: 1
```

HashSet 응용 프로그램

- 실습 예제

StudentTest의 출력 결과가 다음처럼 나오도록 Student 클래스를 구현해 보세요.

```
public class StudentTest {
    public static void main(String[] args) {
        HashSet<Student> set = new HashSet<>();

        set.add(new Student("100", "홍길동"));
        set.add(new Student("200", "강감찬"));
        set.add(new Student("300", "이순신"));
        set.add(new Student("400", "정약용"));
        set.add(new Student("100", "송중기"));

        System.out.println(set);
    }
}
```

<출력 결과>

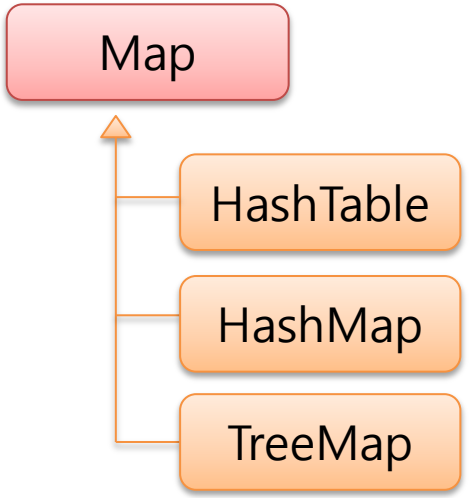
400:정약용 100:홍길동, 200:강감찬, 300:이순신

HashSet 응용 프로그램

```
public class Student {
    private String number;
    private String name;

    public Student(String number, String name) {
        this.number = number;
        this.name = name;
    }

    @Override
    public String toString() {
        return number + ":" + name;
    }

    @Override
    public int hashCode() {
        return Integer.parseInt(number);
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Student) {
            Student student = (Student)obj;
            if(this.number==student.number)
                return true;
        }
        return false;
    }
```

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>,

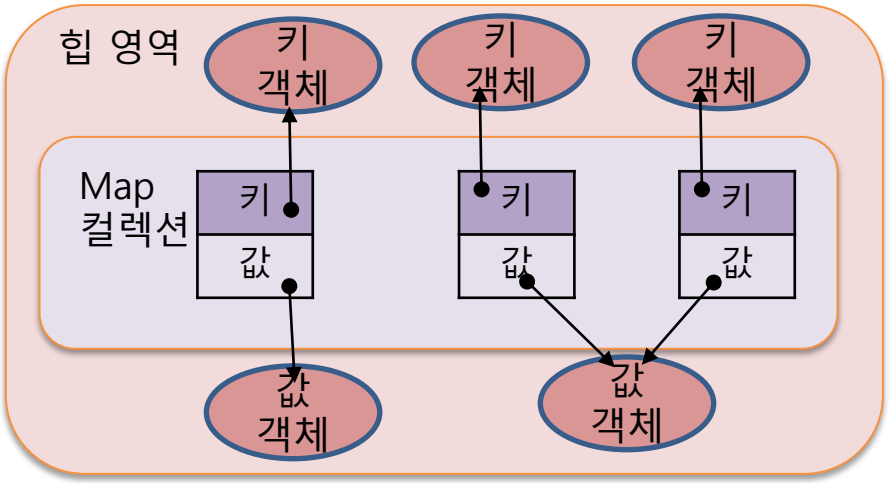
All Known Implementing Classes:

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

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

Map 인터페이스

◆ Map 인터페이스



분류	특 징
Map 인터페이스	- 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
- 내부적으로 hash 방식으로 구현됨 . **Index = hash(key)** ->해시함수가 위치 계산
- Key가 되는 객체는 유일성 여부를 알기 위해 equals()와 hashCode() 메서드를 재정의 함

키타입 값타입

```
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

```java
package collection.map;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapSample {

    public static void main(String[] args) {
        //Map 컬렉션 생성
        Map<String, Integer> map = new HashMap<>();

        //객체 저장
        map.put("이순신", 85);
        map.put("홍길동", 90);
        map.put("강감찬", 80);
        map.put("홍길동", 75);
        System.out.println("총 객체 수: " + map.size());
```

```
총 객체 수: 3
        홍길동: 75

        홍길동 : 75
        강감찬 : 80
        이순신 : 85

총 객체 수: 2
```

Map 인터페이스

```
        //객체 찾기
        System.out.println("\t홍길동: " + map.get("홍길동"));
        System.out.println();

        //객체를 하나씩 처리
        Set<String> keySet = map.keySet();
        Iterator<String> iterator = keySet.iterator();
        while(iterator.hasNext()) {
            String key = iterator.next();
            Integer value = map.get(key);
            System.out.println("\t" + key + " : " + value);
        }
        System.out.println();

        //객체 삭제
        map.remove("홍길동");  //키로 제거
        System.out.println("총 객체 수: " + map.size());
    }
```

Map 인터페이스

◆ HashMap 예제

```
var map = new HashMap<String, Integer>();
int idx = 0;
map.put("Java",       ++idx);
map.put("C",          ++idx);
map.put("C++",        ++idx);
map.put("JavaScript", ++idx);
map.put("Python",     ++idx);
map.put("PHP",        ++idx);

Set<String> keys = map.keySet();

System.out.printf("총 %d개의 Map Entry가 있습니다.\n", keys.size());

//요소 삭제 후 추가하기
if(map.containsKey("PHP")) map.remove("PHP");
map.put("Delphi",     idx);
```

Map 인터페이스

◆ HashMap 예제

```
//전체 요소 조회
for(String key : keys) {
    System.out.println(key + " : " + map.get(key));
}

System.out.println("==Lambda 식====");
keys.forEach(key -> System.out.println(key));

System.out.println("====Method reference====");
map.keySet().forEach(System.out::println);

//자료 존재 유무
System.out.println(map.containsKey("PHP"));
```

```
총 6개의 Map Entry가 있습니다.
Java : 1
C++ : 3
C : 2
JavaScript : 4
Delphi : 6
Python : 5
==Lambda 식====
Java
C++
C
JavaScript
Delphi
Python
====Method reference====
Java
C++
C
JavaScript
Delphi
Python

false
```

HashMap 응용 프로그램

- 실습 예제

다음 코드에서 CarTest의 테스트 결과가 true, true, false가 되도록 HashMap을 사용하여 CarFactory 클래스를 구현해 보세요

```
public class Car {
    String name;

    public Car() {}

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

HashMap 응용 프로그램

```
public class CarFactory {
    private static CarFactory instance;
    private HashMap<String, Car> carMap = new HashMap<>();

    private CarFactory() {}

    public static CarFactory getInstance() {
        if(instance==null) {
            instance = new CarFactory();
        }
        return instance;
    }

    public Car createCar(String name) { //차이름을 매개변수로 전달함
        if(carMap.containsKey(name)) { //카맵에 차이름이 있다면
            return carMap.get(name);   //차이름을 가져와서
        }
        Car car = new Car();
        carMap.put(name, car);   //카맵에 이름과 차 객체를 추가
        return car;
    }

    @Override
    public String toString() {
        return "carMap=" + carMap;
    }
}
```

HashMap 응용 프로그램

```
public class CarTest {
    public static void main(String[] args) {
        CarFactory factory = CarFactory.getInstance();
        Car sonata1 = factory.createCar("수소차");
        Car sonata2 = factory.createCar("수소차");
        System.out.println(sonata1==sonata2);    //true

        Car avante1 = factory.createCar("전기차");
        Car avante2 = factory.createCar("전기차");
        System.out.println(avante1==avante2);    //true

        System.out.println(sonata1==avante1);    //false
    }
}
```

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++) {
        Member member = arrayList.get(i);
        int tempId = member.getMemberId();  //이미 저장된 회원아이디
        if(tempId==memberId) {    //저장 Id와 외부 입력 Id가 같다면
            arrayList.remove(i);  //member 객체 삭제
            return true;
        }
    }
    System.out.println(memberId + "가 존재하지 않습니다.");
    return true;
}
```

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.
```

```java
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();
```

Set 인터페이스

◆ HashSet을 활용한 회원관리 프로그램

```
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;
    }

    public String getMemberName() {
        return memberName;
    }

    public void setMemberName(String memberName) {
        this.memberName = memberName;
    }
```

Set 인터페이스

◆ Member 클래스

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

@Override
public boolean equals(Object obj) {
    if(obj instanceof Member) {  //Member의 인스턴스 객체가 obj라면
        Member member = (Member)obj; //다운 캐스팅 - Member로 형 변환
        if(this.memberId == member.memberId) //외부에서 입력한 memberId와 같다면
            return true;
    }
    return false;
}

@Override
public int hashCode() { //equals와 hashCode 모두 재정의되어야 중복이 허용 안됨
    return memberId;
}
```

Set 인터페이스

```
public class MemberHashSet {
    //자료형이 Member인 HashSet 멤버 변수 선언
    private HashSet<Member> hashSet;

    public MemberHashSet() {
        hashSet = new HashSet<>();
    }

    //회원 추가
    public void addMember(Member member) {
        hashSet.add(member);
    }

    //회원 목록 출력
    public void showAllMember() {
        /*for(Member member: hashSet) {
            System.out.println(member);
        }*/
        Iterator<Member> ir = hashSet.iterator();
        while(ir.hasNext()) {
            Member member = ir.next();
            System.out.println(member);
        }
    }
}
```

Set 인터페이스

◆ MemberHashSet 클래스

```
//회원 삭제
public boolean removeMember(int memberId) {//매개변수는 memberId
    Iterator<Member> ir = hashSet.iterator();
    while(ir.hasNext()) {
        Member member = ir.next();
        int dbMemberId = member.getMemberId(); //이미 저장된 memberId를 가져옴
        if(dbMemberId == memberId) { //외부 입력 memberId와 같다면
            hashSet.remove(member); //회원 객체 삭제
            return true;
        }
    }
    System.out.println(memberId + "가 존재하지 않습니다.");
    return false;
}
```

Set 인터페이스

```
public class MemberHashSetTest {

    public static void main(String[] args) {
        //객체 생성
        MemberHashSet memberHashSet = new MemberHashSet();

        //회원 추가
        memberHashSet.addMember(new Member(1001, "네이버"));
        memberHashSet.addMember(new Member(1002, "카카오"));
        memberHashSet.addMember(new Member(1003, "엔씨소프트"));
        memberHashSet.addMember(new Member(1001, "네이버"));  //중복 불가

        //회원 목록 조회
        memberHashSet.showAllMember();
        System.out.println("==================================");

        //회원 삭제
        memberHashSet.removeMember(1003);
        memberHashSet.removeMember(1004);

        //회원 목록 조회
        memberHashSet.showAllMember();
    }
}
```

```
네이버 회원님의 아이디는 1001입니다.
카카오 회원님의 아이디는 1002입니다.
엔씨소프트 회원님의 아이디는 1003입니다.
==================================
회원 아이디 1004가 존재하지 않습니다.
네이버 회원님의 아이디는 1001입니다.
카카오 회원님의 아이디는 1002입니다.
```

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();
    }
}
```

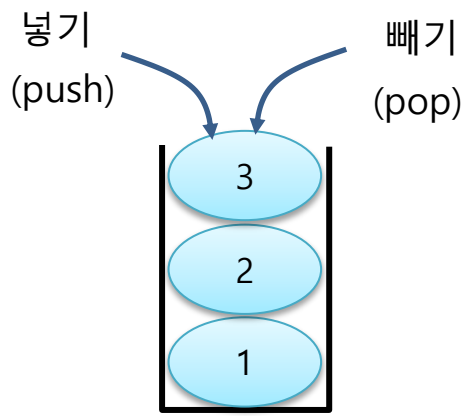
```
삼성전자 회원님의 아이디는 1001입니다.
현대자동차 회원님의 아이디는 1002입니다.
네이버 회원님의 아이디는 1003입니다.
카카오 회원님의 아이디는 1004입니다.
==================================
회원 아이디 1005가 존재하지 않습니다.
현대자동차 회원님의 아이디는 1002입니다.
네이버 회원님의 아이디는 1003입니다.
카카오 회원님의 아이디는 1004입니다.
```

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)

```java
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());
    }
}
```

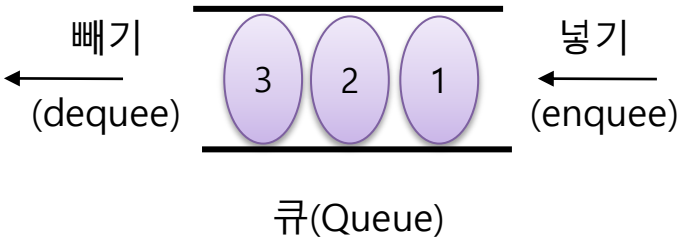
```
소
닭
돼지
스택이 비었습니다.
null
```

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



큐(Queue)

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>,

스택(Stack)과 큐(Queue)

❖ Queue를 이용한 메시지 큐 구현하기

```
package collection.list;

import java.util.LinkedList;
import java.util.Queue;

class Message{
    public String command; //수행하기
    public String to;      //대상

    public Message(String command, String to) {
        this.command = command;
        this.to = to;
    }
}
```

```
이양파님에게 메일을 보냅니다.
박마늘님에게 SMS를 보냅니다.
오감자님에게 Katalk을 보냅니다.
```

스택(Stack)과 큐(Queue)

```java
public class MessageQueueTest {
    public static void main(String[] args) {
        Queue<Message> messageQueue = new LinkedList<>();

        //Message 객체 생성
        Message mail = new Message("sendMail", "이양파");
        Message sms = new Message("sendSMS", "박마늘");
        Message katalk = new Message("sendKatalk", "오감자");

        //객체 넣기
        messageQueue.offer(mail);
        messageQueue.offer(sms);
        messageQueue.offer(katalk);

        //객체 빼기
        while(!messageQueue.isEmpty()) {
            Message message = messageQueue.poll();
            switch(message.command) { //message.command는 String형
            case "sendMail":
                System.out.println(message.to + "님에게 메일을 보냅니다.");
                break;
            case "sendSMS":
                System.out.println(message.to + "님에게 SMS를 보냅니다.");
                break;
            case "sendKatalk":
                System.out.println(message.to + "님에게 Katalk을 보냅니다.");
                break;
            }
        }
```

Queue인터페이스를 구현한 대표적인 클래스는 **LinkedList**이다.

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
```