15장-1. 제네릭, 컬렉션(List)

Generic / List

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>{....}

```
class Box<T>{
    private T type;

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

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

```
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);
    }
}
```

제네릭(Generic)

비제네릭 타입

```
class Box{
    private Object obj;

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

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

```
public class BoxTest {

    public static void main(String[] args) {
        //문자열(String) type 저장
        Box box = new Box();
        box.set("행운을 빌어요!");
        String msg = (String) box.get();
        System.out.println(msg);

        //객체 type 저장
        box.set(new Apple());
        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>

```
회사가 TV를 제조합니다.
모델 : 스마트TV
회사가 자동차를 제조합니다.
모델 : 전기차
```

```java
package generic.product;
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() {
        return kind;
    }

    public M getModel() {
        return model;
    }

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

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

멀티타입 파라미터 – class<K, V>

```
public class GenericProduct {

    public static void main(String[] args) {
        // <객체, 문자열> 타입
        Product<TV, String> product1 = new Product<>();
        TV tv = new TV();
        tv.making();

        product1.setKind(tv);
        product1.setModel("스마트TV");
        String tvModel = product1.getModel();
        System.out.println("모델 : " + tvModel);

        //<객체, 문자열> 타입
        Product<Car, String> product2 = new Product<>();
        Car car = new Car();
        car.making();

        product2.setKind(new Car());
        product2.setModel("전기차");
        String carModel = product2.getModel();
        System.out.println("모델 : " + carModel);
    }
}
```

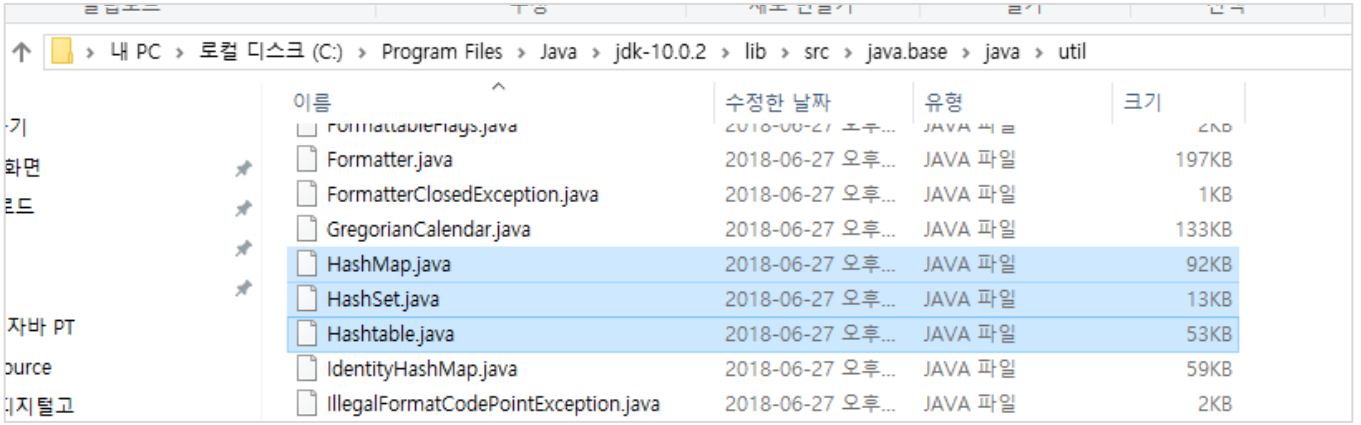
컬렉션 프레임워크

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:

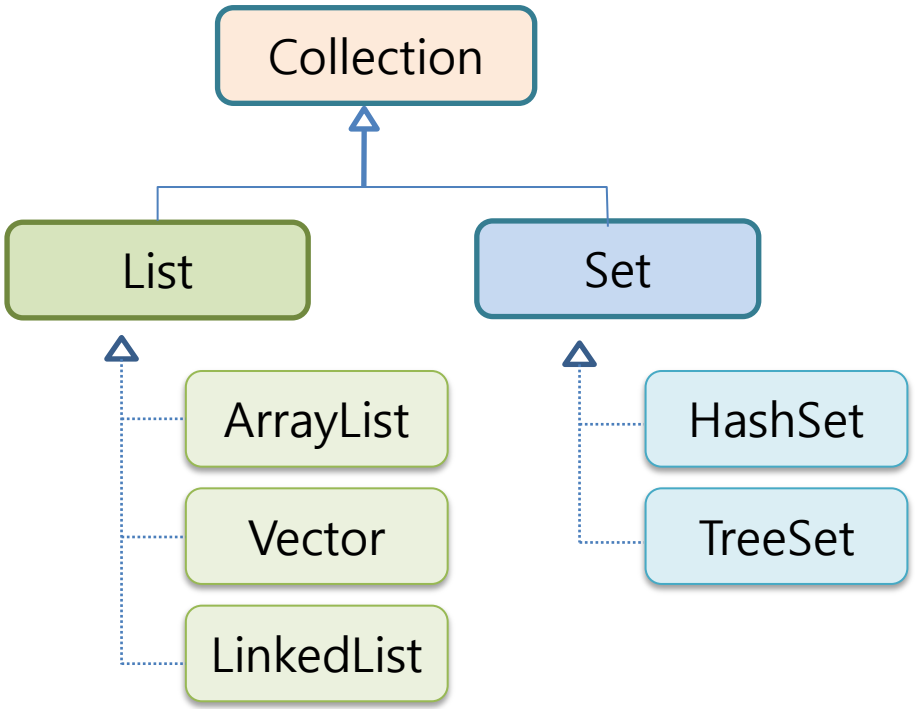
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, EventSet, List<E>,

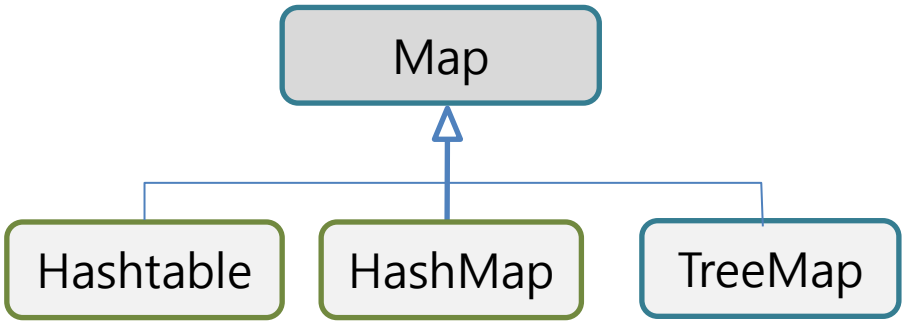
All Known Implementing Classes:

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 + " ");
    }
}
```

회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

Member.java – 회원 클래스(자료)

```
package collection.list.arraylist;

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;
    }
```

회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

Member.java – 회원 클래스(자료)

```
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;
}

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

회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 프로그램

MemberArrayList.java – 회원 추가, 회원 전체 조회

```
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

MemberArrayList.java – 회원 삭제

```
//회원 삭제 - 회원 아이디가 매개변수임
public boolean removeMember(int memberId) {
    for(int i=0; i<memberList.size(); i++) {
        Member member = memberList.get(i);
        int tempId = member.getMemberId();
        //저장된 회원아이디를 임시변수에 저장
        if(tempId==memberId) {//저장된 Id와 외부입력 Id가 같다면
            memberList.remove(i); //remove() 메서드로 삭제
            return true;
        }
    }
    System.out.println(memberId + "가 존재하지 않습니다.");
    return false;
}
```

회원 관리 프로그램

◆ ArrayList를 활용한 회원관리 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 가입
memberArrayList.addMember(chu);
memberArrayList.addMember(son);
memberArrayList.addMember(park);
memberArrayList.addMember(kim);

//회원 목록
memberArrayList.showAllMember();

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

//회원 삭제
memberArrayList.removeMember(park.getMemberId());
memberArrayList.removeMember(1005);

//삭제 후 전체 출력
memberArrayList.showAllMember();
```

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

```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

◆ 벡터(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

```java
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
```

Vector 클래스

◆ Enumeration 인터페이스 사용하기

- 객체들의 집합(Vector)에서 각각의 객체들을 하나씩 처리할 수 있는 메소드를 제공한다.
- Vector 클래스의 elements()는 객체의 모든 요소들을 Enumeration 객체로 반환한다.

```
Vector<String> aniList = new Vector<>();

aniList.addElement("cat");
aniList.addElement("dog");
aniList.addElement("cow");

System.out.println("Vector 요소들은 다음과 같다.");
for(int i=0; i<aniList.size(); i++){
    System.out.println(i + "번 : " + aniList.elementAt(i));
}
```

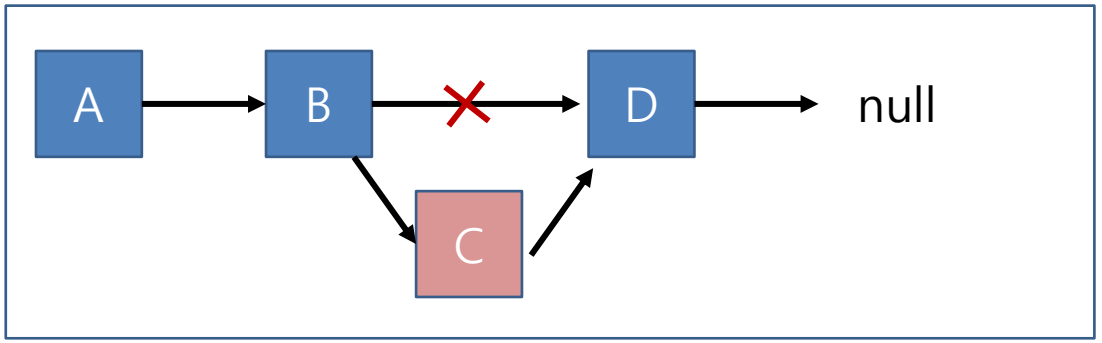
```
Enumeration<String> enu = aniList.elements();
System.out.println("Vector의 객체인 aniList로 생성한
while(enu.hasMoreElements()) {
    System.out.println(enu.nextElement());
}
```

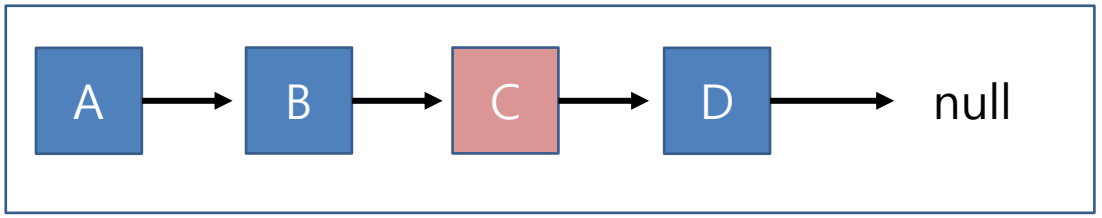
```
Vector 요소들은 다음과 같다.
0번 : cat
1번 : dog
2번 : cow
Vector의 객체인 aniList로 생성한 Enumeration 요소
cat
dog
cow
```

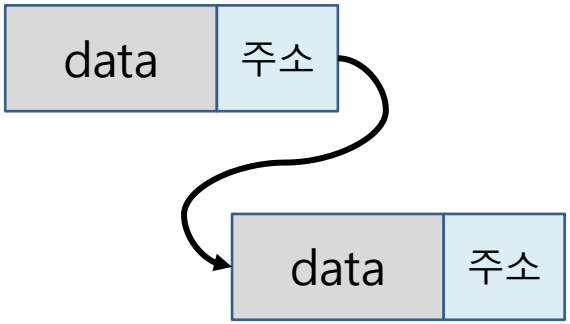
LinkedList 클래스

◆ 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

◆ 링크드 리스트(LinkedList) 클래스

```java
package collection.list;

import java.util.LinkedList;

public class LinkedListTest {
    public static void main(String[] args) {
        LinkedList<String> myList = new LinkedList<>();

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

        //객체 조회
        System.out.println(myList);
```

LinkedList 클래스

◆ 링크드 리스트(LinkedList) 클래스

```
        for(int i=0; i<myList.size(); i++) {
            String vegetable = myList.get(i);
            System.out.print(vegetable + " ");
        }
        System.out.println();

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

        //연결리스트의 맨 뒤에 추가
        myList.addLast("E");
        System.out.println(myList);

        //연결리스트의 맨 앞 요소 삭제
        myList.removeFirst();
        System.out.println(myList);
}
```

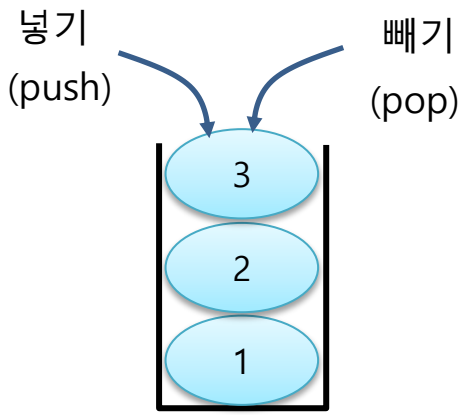
```
[A, B, C]
A B C
[A, B, D, C]
[A, B, D, C, E]
[B, D, C, E]
```

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

Coin.java – Stack에 사용할 자료형

```
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)

❖ Stack 클래스로 동전 꺼내기

```
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

MyStack.java
- Stack에 사용할 자료형

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

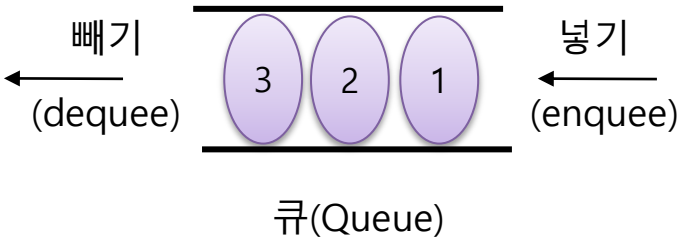
```
소
닭
돼지
스택이 비었습니다.
null
```

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>,

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)

❖ Queue를 이용한 메시지 큐

```
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
```