

# 5장. 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.box2;

public class Box {

    private Object obj;

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

    public Object get() {
        return obj;
    }

}
```



# 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);  
    }  
}
```



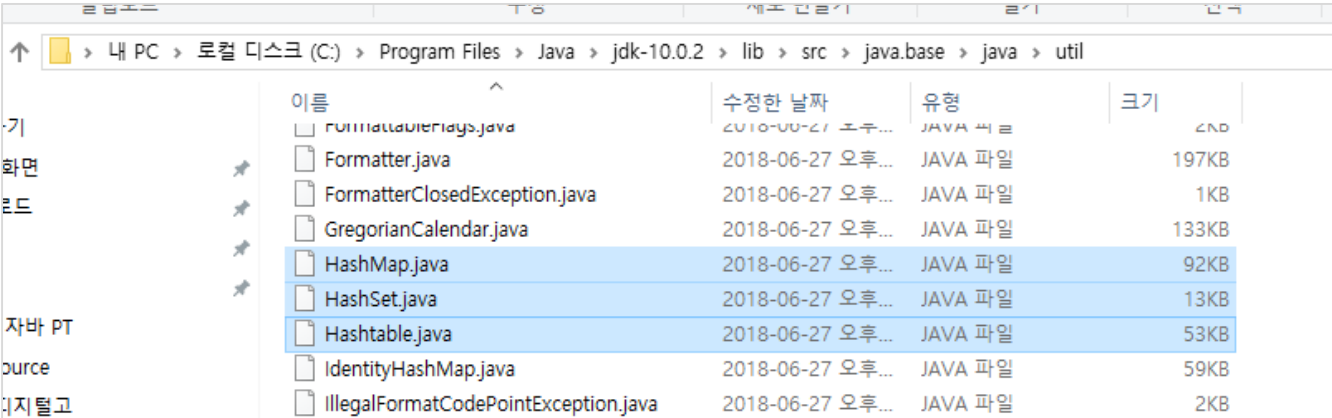
# 컬렉션 프레임워크

## ● 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?



The screenshot shows a Windows File Explorer window with the address bar displaying the path: `> 내 PC > 로컬 디스크 (C:) > Program Files > Java > jdk-10.0.2 > lib > src > java.base > java > util`. The main pane shows a list of files in the `util` package. The files are listed in a table with columns for Name, Modified Date, Type, and Size.

이름	수정된 날짜	유형	크기
<code>FormattableArrays.java</code>	2018-06-27 오후...	JAVA 파일	4KB
<code>Formatter.java</code>	2018-06-27 오후...	JAVA 파일	197KB
<code>FormatterClosedException.java</code>	2018-06-27 오후...	JAVA 파일	1KB
<code>GregorianCalendar.java</code>	2018-06-27 오후...	JAVA 파일	133KB
<code>HashMap.java</code>	2018-06-27 오후...	JAVA 파일	92KB
<code>HashSet.java</code>	2018-06-27 오후...	JAVA 파일	13KB
<code>Hashtable.java</code>	2018-06-27 오후...	JAVA 파일	53KB
<code>IdentityHashMap.java</code>	2018-06-27 오후...	JAVA 파일	59KB
<code>IllegalFormatCodePointException.java</code>	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:

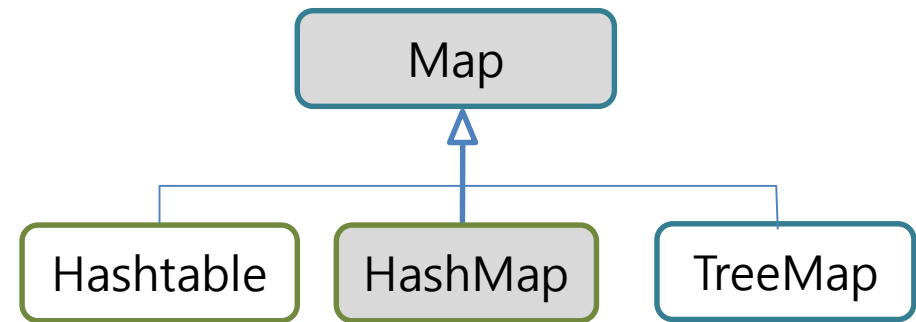
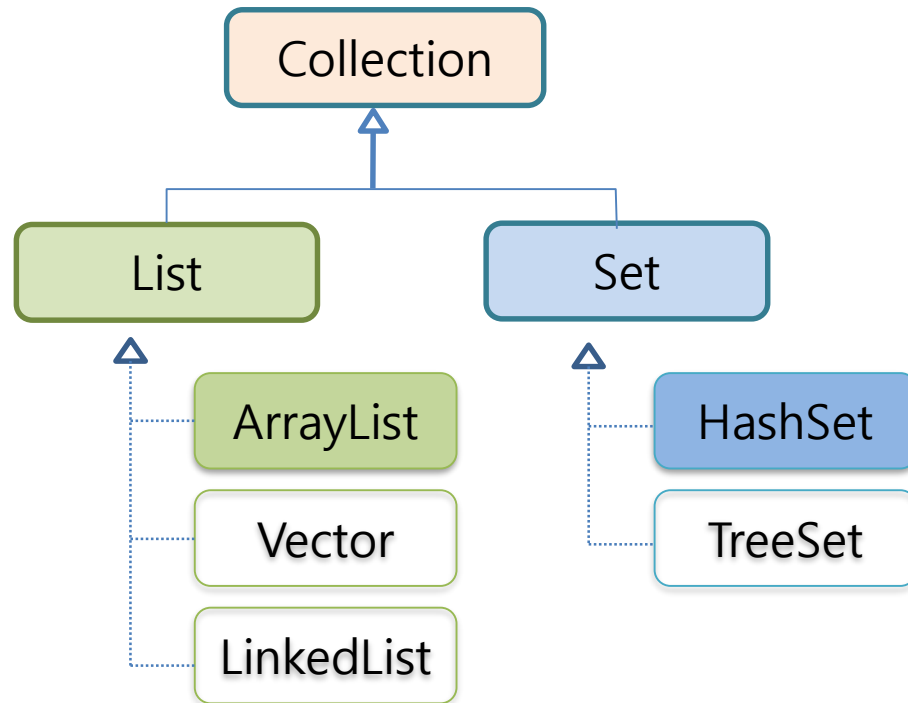
AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, 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"><li>- 순서를 유지하고 저장</li><li>- 중복 저장 가능</li><li>- 구현클래스 : ArrayList, Vector, LinkedList</li></ul>
Set 인터페이스	<ul style="list-style-type: none"><li>- 순서를 유지하지 않고 저장</li><li>- 중복 저장 안됨</li><li>- 구현클래스 : HashSet, TreeSet</li></ul>



# 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



# List 인터페이스

## ● 요소 삽입, 삭제 구현

```
List<String> myList = new ArrayList<>();

//자료 추가
myList.add("A");
myList.add("B");
myList.add("D");

System.out.println(myList);

//특정 위치에 자료 추가
myList.add(2, "C");

System.out.println(myList);

//자료 삭제
if(myList.contains("B")) {
    myList.remove("B");
}

System.out.println(myList);

//전체 요소 출력
for(String list : myList)
    System.out.println(list);
```

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

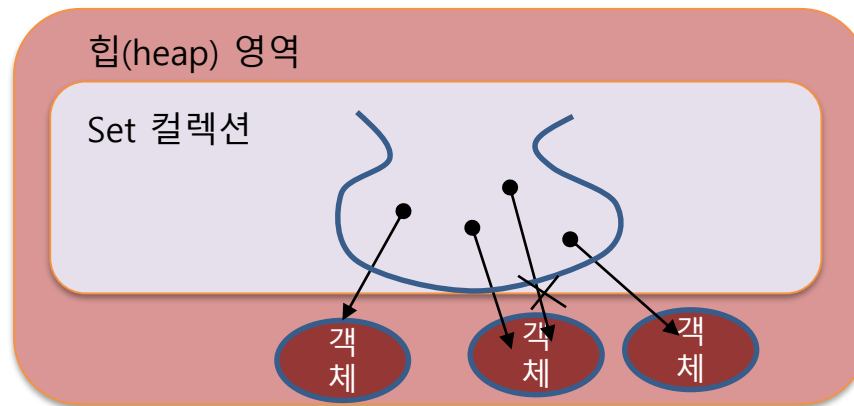


# 컬렉션 프레임워크

## ◆ 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("김선화");  
set.add("고담덕");  
set.remove("김선화")
```

- 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.

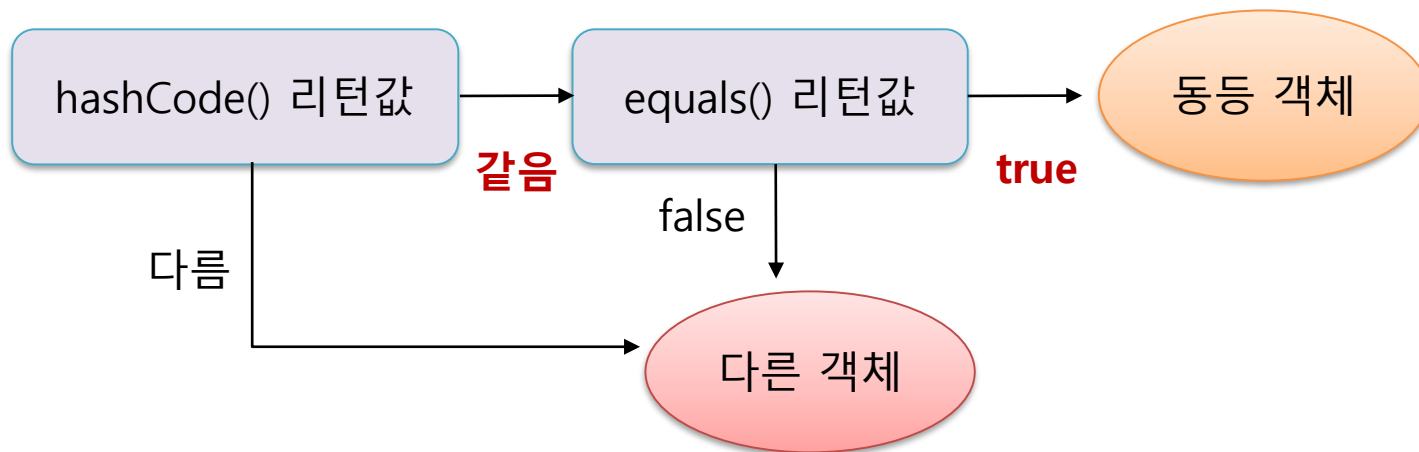
리턴 타입	메소드명	설명
Boolean	hasNext()	가져올 객체가 있으면 true, 없으면 false 리턴
E	next()	컬렉션에서 하나의 객체를 가져온다.
void	remove()	Set 컬렉션에서 객체를 제거한다.



# 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);
}
```

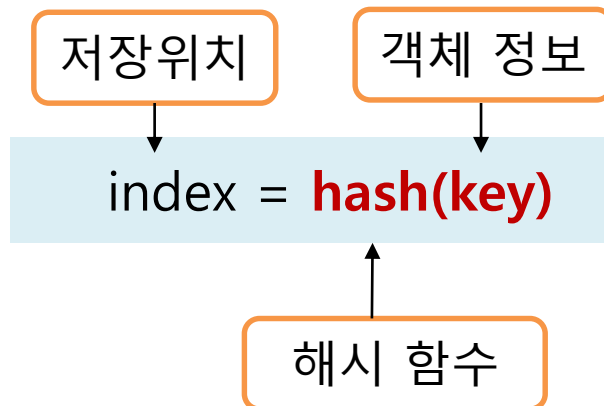
```
[Java, C++, JDBC, Python]
총 객체수: 4
JDBC
        Java
        C++
        JDBC
        Python
=====
        Java
        JDBC
        Python
```



# hashCode() 메서드

## ● hashCode() 메서드

- hash : 정보를 저장, 검색하기 위해 사용하는 자료 구조
- 자료의 특정 값(키 값)에 대해 저장 위치를 반환해 주는 해시 함수를 사용함.  
hash 알고리즘은 검색(search)에 효율적임
- hashCode() 메서드는 인스턴스의 저장 주소를 반환함 : **10진수**로 나타냄



# hashCode() 메서드

## ● hashCode() 메서드

Hash Table

	0
	1
(2025202, "kim")	2
	3
	4
(2025205, "kim")	5
	6
(2025207, "lee") (2025217, "kim")	7
	8
	9

key

value

(2025202, "kim")

$\text{hash}(\text{key}) = \text{key} \% n(\text{테이블 크기})$

$\text{hash}(\text{key}) = 2025102 \% 10$





# hashCode() 메서드

## ▪ hashCode() 재정의 예제

```
package set;

public class Employee {
    int empId;
    String empName;

    Employee(int empId, String empName){
        this.empId = empId;
        this.empName = empName;
    }

    @Override
    public String toString() {
        return empId + ", " + empName;
    }
}
```



# hashCode() 메서드

## ▪ hashCode() 재정의 예제

```
@Override
public boolean equals(Object obj) {
    if(obj instanceof Employee) {
        Employee employee = (Employee)obj;
        if(this.empId == employee.empId)
            return true;
    }
    return false;
}

@Override
public int hashCode() {
    return empId;
}
}
```



# hashCode() 메서드

## ▪ hashCodeTest

```
Employee emp1 = new Employee(101, "장그래");
Employee emp2 = new Employee(101, "장그래");

System.out.println(emp1);
System.out.println(emp2);

//물리적인 주소
System.out.println(System.identityHashCode(emp1));
System.out.println(System.identityHashCode(emp2));

//물리적인 메모리 주소
System.out.println(emp1 == emp2); //false

//논리적으로 문자열을 동일하도록 재정의함
System.out.println(emp1.equals(emp2)); //true

//hashCode() 재정의
System.out.println(emp1.hashCode());
System.out.println(emp2.hashCode());
```

```
101, 장그래
101, 장그래
1237514926
548246552
false
true
101
101
```



# hashCode() 메서드

## ■ hashCodeTest

```
Employee emp1 = new Employee(101, "장그래");  
Employee emp2 = new Employee(101, "장그래");
```

```
System.out.println(emp1 == emp2);  
System.out.println(emp1.equals(emp2));
```

```
//논리적으로 문자열 동일하도록 재정의 함
```

```
System.out.println(emp1.hashCode());  
System.out.println(emp2.hashCode());
```

```
//물리적인 메모리 주소는 다름
```

```
System.out.println(System.identityHashCode(emp1));  
System.out.println(System.identityHashCode(emp2));
```

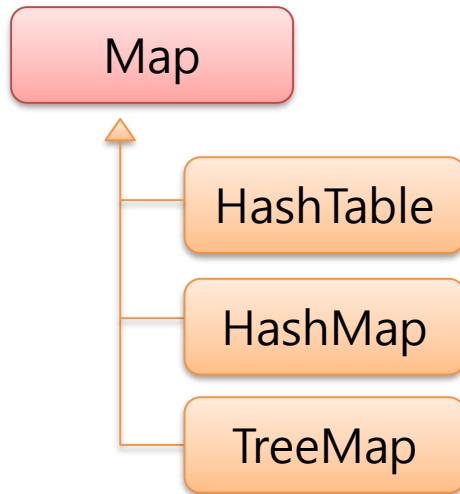
```
}
```

```
false  
true  
49571336  
49571336  
93122545  
2083562754  
=====  
false  
true  
101  
101  
557041912  
1134712904
```



# 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

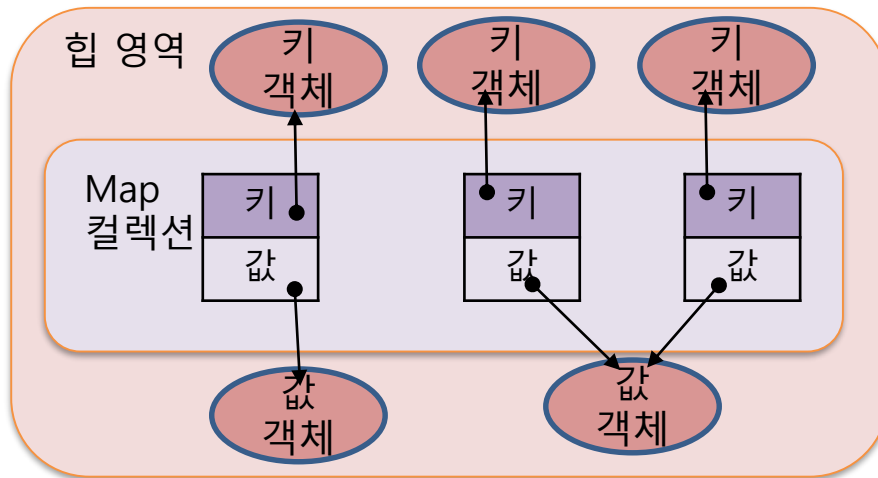
---

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



# Map 인터페이스

## ◆ Map 인터페이스



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



# 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 컴퓨터 용어 사전 프로그램을 구현함
- ✓ HashMap을 사용하여 용어와 그 정의를 저장함
- ✓ 사용자 입력을 통해 용어를 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

## ◆ ArrayList를 활용한 회원관리 프로그램

```
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();
```

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.



# 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

## ◆ HashMap을 활용한 회원관리 프로그램

```
//MemberHashMap 객체 생성
MemberHashMap hashMap = new MemberHashMap();

//회원 추가
hashMap.addMember(new Member(1001, "이정후"));
hashMap.addMember(new Member(1002, "신유빈"));
hashMap.addMember(new Member(1003, "최민정"));
hashMap.addMember(new Member(1003, "임시현")); //중복 발생

//회원 출력
hashMap.showAllMember();

//회원 삭제
hashMap.removeMember(1002);
hashMap.removeMember(1004);

//삭제 후 회원 출력
hashMap.showAllMember();
```

이정후 회원님의 아이디는 1001입니다.  
신유빈 회원님의 아이디는 1002입니다.  
임시현 회원님의 아이디는 1003입니다.

1004가 존재하지 않습니다.  
이정후 회원님의 아이디는 1001입니다.  
임시현 회원님의 아이디는 1003입니다.



# 실습 문제 – ArrayList

다음 출력 결과가 나오도록 빈 칸을 채우시오

```
class Shape{
    public void draw() {
        System.out.println("도형");
    }
}

class Circle extends Shape{
    public void draw() {
        System.out.println("원");
    }
}

class Triangle extends Shape{
    public void draw() {
        System.out.println("삼각형");
    }
}
```

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

        list.add(new Shape());
        list.add(new Circle());
        list.add(new Triangle());

        for(Shape s : list)
            
    }
}
```

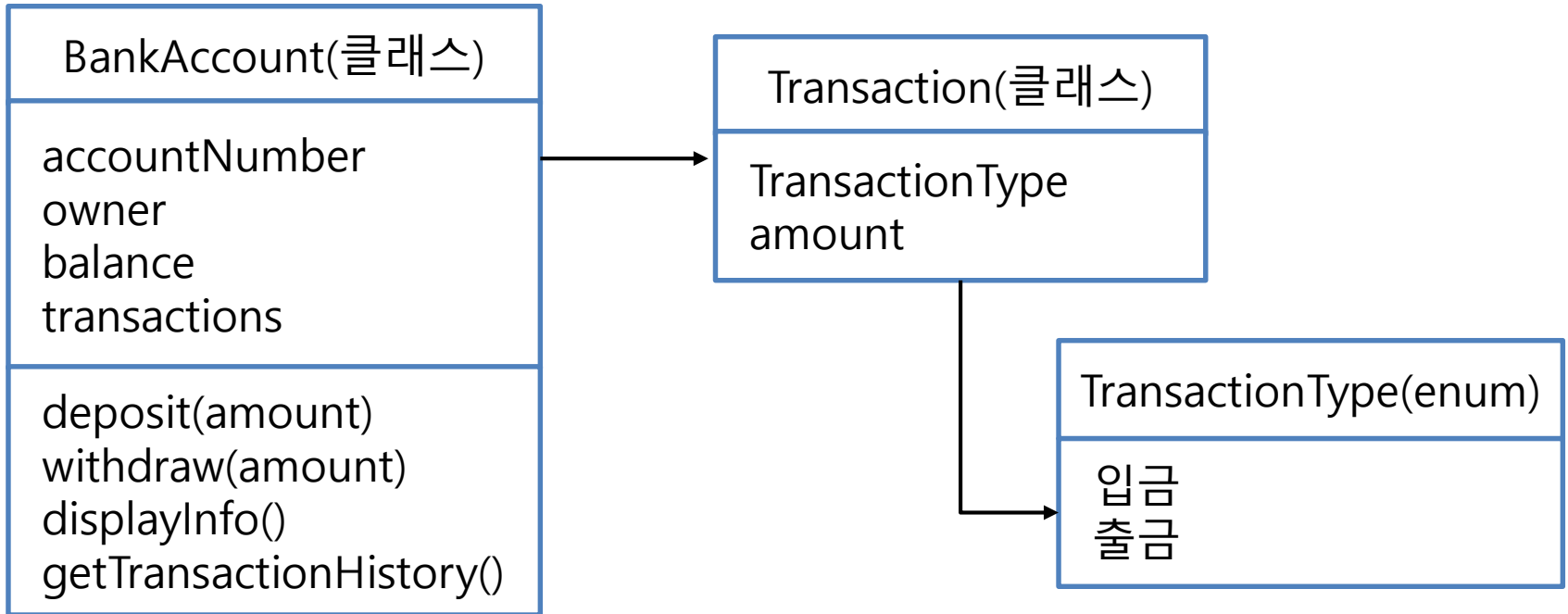
☞ 실행 결과

도형  
원  
삼각형



# 은행 거래 프로젝트 2

## ▪ BankProject > 클래스 다이어그램



# 은행 거래 프로젝트

## ■ 은행 거래 내역 테스트 출력

```
=====
1. 계좌 생성 | 2. 예금 | 3. 출금 | 4. 계좌 검색 | 5. 종료
=====
```

```
선택> 1
```

```
=====
계좌 생성
=====
```

```
계좌 번호 입력: 100-200-3000
```

```
계좌주 입력: kim
```

```
계좌가 생성되었습니다. (계좌 번호: 100-200-3000)
```

```
=====
1. 계좌 생성 | 2. 예금 | 3. 출금 | 4. 계좌 검색 | 5. 종료
=====
```

```
선택> 2
```

```
=====
예금
=====
```

```
계좌 번호 입력: 100-200-3000
```

```
입금액 입력: 10000
```

```
입금이 정상 처리 되었습니다. 현재 잔액: 10000
```

```
=====
1. 계좌 생성 | 2. 예금 | 3. 출금 | 4. 계좌 검색 | 5. 종료
=====
```

```
선택> 3
```

```
=====
출금
=====
```

```
계좌 번호 입력: 100-200-3000
```

```
출금액 입력: 5000
```

```
출금이 정상 처리 되었습니다. 현재 잔액: 5000
```

```
=====
1. 계좌 생성 | 2. 예금 | 3. 출금 | 4. 계좌 검색 | 5. 종료
=====
```

```
선택> 4
```

```
조회할 계좌번호 입력: 100-200-3000
```

```
계좌 정보
```

```
계좌 번호: 100-200-3000
```

```
계좌주: kim
```

```
잔고: 5000
```

```
|입금| 10000원
```

```
|출금| 5000원
```



# 은행 거래 프로젝트

## ▪ 거래(트랜잭션) 유형 - enum

```
package bankapp1_1;  
  
public enum TransactionType {  
    입금,  
    출금  
}
```





# 은행 거래 프로젝트

## ■ 거래(트랜잭션) - 클래스

```
public class Transaction {  
    TransactionType type;    //거래 유형(enum 참조)  
    int amount;              //거래 금액  
  
    public Transaction(TransactionType type, int amount) {  
        this.type = type;  
        this.amount = amount;  
    }  
}
```



# 은행 거래 프로젝트

## ■ 은행계좌(BankAccount) - 클래스

```
public class BankAccount {
    private String accountNumber; //계좌 번호
    private String owner; //계좌주
    private int balance; //잔고
    ArrayList<Transaction> transactions;

    //계좌 번호 정규식 패턴
    private static final String ACCOUNT_PATTERN =
        "^[0-9]{2,4}-[0-9]{2,4}-[0-9]{4,8}$";

    //생성자
    public BankAccount(String accountNumber, String owner) {
        if(!isValidAccountNumber(accountNumber)) {
            throw new IllegalArgumentException("유효하지 않은 계좌번호 형식입니다.");
        }
        this.accountNumber = accountNumber;
        this.owner = owner;
        this.balance = 0;
        transactions = new ArrayList<>();
    }
}
```



# 은행 거래 프로젝트

## ■ 은행계좌(BankAccount) - 클래스

```
//계좌 번호 유효성 검사
private boolean isValidAccountNumber(String accountNumber) {
    return Pattern.matches(ACCOUNT_PATTERN, accountNumber);
}

//setter, getter
public String getAccountNumber() {
    return accountNumber;
}

public void setAccountNumber(String accountNumber) {
    this.accountNumber = accountNumber;
}

public int getBalance() {
    return balance;
}

public void setBalance(int balance) {
    this.balance = balance;
}
```



# 은행 거래 프로젝트

## ■ 거래(transaction)

```
//거래 추가
public void addTransaction(TransactionType type, int amount) {
    Transaction transaction = new Transaction(type, amount);
    transactions.add(transaction);
}

//거래 내역 조회
public void getTransactionHistory() {
    if(transactions.isEmpty()) {
        System.out.println("거래 내역이 없습니다.");
        return;
    }

    for(Transaction transaction : transactions) {
        System.out.print(" | " + (transaction.type == TransactionType.입금 ?
                                "입금" : "출금"));
        System.out.println(" | " + transaction.amount + "원");
    }
}
```



# 은행 거래 프로젝트

## ■ 계좌 정보 출력

```
public void displayInfo() { //계좌 정보 출력
    System.out.println("계좌 정보");
    System.out.println("    계좌 번호: " + accountNumber);
    System.out.println("    계좌주: " + owner);
    System.out.println("    잔고: " + balance);
}
```



# 은행 거래 프로젝트

## ▪ Main 테스트

```
public class Main {  
  
    static List<BankAccount> accountList = new ArrayList<>();  
    static Scanner scan = new Scanner(System.in);  
  
    public static void main(String[] args) {  
        boolean run = true;  
  
        while(run) {  
            System.out.println("=====");  
            System.out.println("1.계좌 생성 | 2.예금 | 3.출금 | 4.계좌 검색 | 5.종료");  
            System.out.println("=====");  
            System.out.print("선택> ");  
  
            int choice = Integer.parseInt(scan.nextLine());
```



# 은행 거래 프로젝트

## ▪ Main 테스트

```
switch(choice) {  
case 1:  
    createAccount();  
    break;  
case 2:  
    deposit();  
    break;  
case 3:  
    withdraw();  
    break;  
case 4:  
    selectAccount();  
    break;  
case 5:  
    System.out.println("프로그램을 종료합니다.");  
    run = false;  
    break;  
default:  
    System.out.println("지원되지 않는 기능입니다.");  
    break;  
}  
}  
} //main() 함수
```



# 은행 거래 프로젝트

## ■ 계좌 생성

```
private static void createAccount() {
    System.out.println("=====");
    System.out.println("                      계  좌  생  성                      ");
    System.out.println("=====");

    while(true) {
        try {
            System.out.print("계좌 번호 입력: ");
            String accNum = scan.nextLine();

            if(searchAccount(accNum) != null) {
                System.out.println("이미 등록된 계좌입니다. 다른 계좌를 입력하세요.");
            }else {
                System.out.print("계좌주 입력: ");
                String name = scan.nextLine();

                //신규 계좌 생성
                BankAccount newAccount = new BankAccount(accNum, name);
                accountList.add(newAccount);
                System.out.println("계좌가 생성되었습니다.(계좌 번호: " + accNum + ")");
                break;
            }
        }
    }
}
```





# 은행 거래 프로젝트

## ■ 계좌 검색 메서드

```
        } catch (IllegalArgumentException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}  
  
//계좌 검색  
private static BankAccount searchAccount(String accNum) {  
    BankAccount account = null;  
  
    for(int i = 0; i < accountList.size(); i++) {  
        String dbAccNum = accountList.get(i).getAccountNumber();  
        if(dbAccNum.equals(accNum)) { //계좌 번호가 일치하면  
            account = accountList.get(i); //계좌 인스턴스 저장  
            break;  
        }  
    }  
  
    return account;  
}
```



# 은행 거래 프로젝트

## ■ 예금

```
private static void deposit() {
    System.out.println("=====");
    System.out.println("                        예      금                        ");
    System.out.println("=====");

    while(true) {
        System.out.print("계좌 번호 입력: ");
        String accNum = scan.nextLine();

        System.out.print("입금액 입력: ");
        int amount = Integer.parseInt(scan.nextLine());

        if(searchAccount(accNum) != null) {
            BankAccount account = searchAccount(accNum);
            if(amount < 0) {
                System.out.println("유효한 금액을 입력하세요.");
            }else {
                account.setBalance(account.getBalance() + amount);
                System.out.println("입금이 정상 처리 되었습니다. 현재 잔액: " + account.getBalance());
                account.addTransaction(TransactionType.입금, amount); //입금 거래 추가
                break;
            }
        }else {
            System.out.println("계좌가 없습니다.");
        }
    }
}
```



# 은행 거래 프로젝트

## ■ 출금

```
private static void withdraw() {
    System.out.println("=====");
    System.out.println("출금");
    System.out.println("=====");

    while(true) {
        System.out.print("계좌 번호 입력: ");
        String accNum = scan.nextLine();

        System.out.print("출금액 입력: ");
        int amount = Integer.parseInt(scan.nextLine());

        if(searchAccount(accNum) != null) {
            BankAccount account = searchAccount(accNum);
            if(amount < 0) {
                System.out.println("유효한 금액을 입력하세요.");
            } else if(amount > account.getBalance()) {
                System.out.println("잔액이 부족합니다.");
            } else {
                account.setBalance(account.getBalance() - amount);
                System.out.println("출금이 정상 처리 되었습니다. 현재 잔액: " + account.getBalance());
                account.addTransaction(TransactionType.출금, amount); //출금 거래 추가
                break;
            }
        }
    }
}
```



# 은행 거래 프로젝트

## ■ 계좌 정보 검색

```
//계좌 정보 검색
private static void selectAccount() {

    while(true) {
        System.out.print("조회할 계좌번호 입력: ");
        String accNum = scan.nextLine();

        if(searchAccount(accNum) != null) {
            BankAccount account = searchAccount(accNum);
            account.displayInfo();
            account.getTransactionHistory();
            break;
        }else {
            System.out.println("계좌가 없습니다. 다시 입력하세요");
        }
    }
}
```

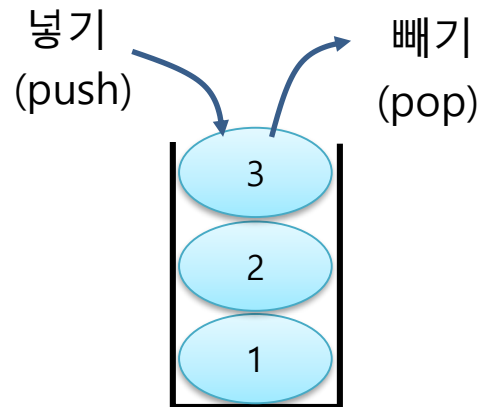


# 스택(Stack)과 큐(Queue)

## ❖ Stack 클래스

- 후입선출(LIFO : Last in First Out) 구조 – (응용 예: JVM 스택 메모리, 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



# 스택(Stack)

## ❖ 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)

## ❖ Stack 클래스로 동전 넣고 빼기 구현

```
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);

//넣기(500 - 100 - 50 - 10)
coinBox.push(coin500);
coinBox.push(coin100);
coinBox.push(coin50);
coinBox.push(coin10);
```



# 스택(Stack)과 큐(Queue)

## ❖ Stack 클래스로 동전 넣고 빼기 구현

```
//빼기(10 - 50 - 100 - 500)
//System.out.println(coinBox.pop().getValue() + "원");
//System.out.println(coinBox.pop().getValue() + "원");

//빼기 범위를 초과하면 EmptyStackException 발생
while(!coinBox.isEmpty()) {
    Coin coin = coinBox.pop(); //삭제
    System.out.println(coin.getValue() + "원");
}
```



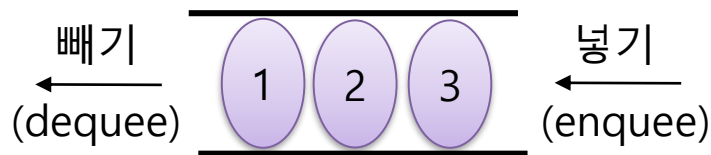


# 스택(Stack)과 큐(Queue)

## ❖ Queue 인터페이스

- 선입선출(FIFO : First in First Out) 구조 – (활용 예: 프린터 대기열, 운영체제 메시지큐)
- 주요 메소드

메서드명	설명
offer()	주어진 객체를 넣는다.
poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.
peek()	큐의 맨 앞 요소 출력
isEmpty()	스택의 객체가 비어있는지 여부



큐(Queue)



# 큐(Queue)

## ❖ Queue(큐)를 이용한 은행 대기줄 구현

```
Queue<String> queue = new LinkedList<>();

queue.offer("고객A");
queue.offer("고객B");
queue.offer("고객C");

//System.out.println(queue.peek()); //고객A
//queue.poll(); //요소 삭제

//System.out.println(queue.peek()); //고객B

while(!queue.isEmpty()) {
    System.out.println(queue.peek() + "님 업무 처리중...");
    queue.poll();
}
System.out.println("모든 고객의 업무가 완료되었습니다.");
```

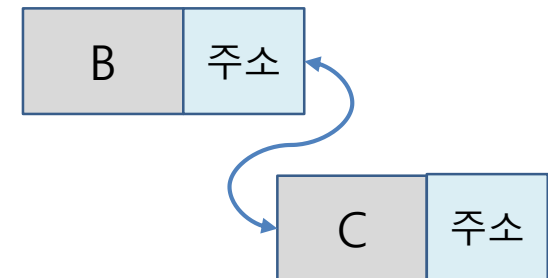
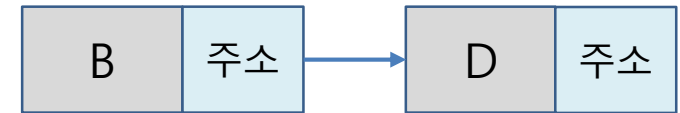
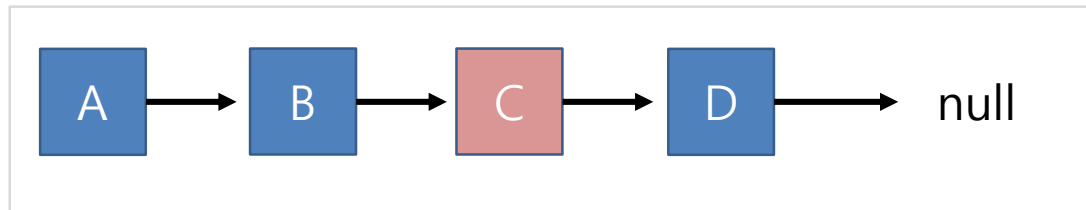
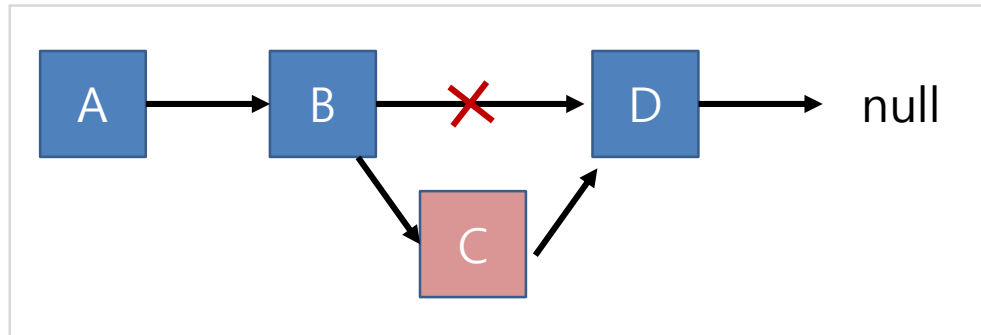
고객A님 업무 처리중...  
고객B님 업무 처리중...  
고객C님 업무 처리중...  
모든 고객의 업무가 완료되었습니다.



# 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 자료 이동이 발생하지 않음
- ArrayList 보다 중간에 자료를 넣고 제거하는 데 시간이 적게 걸리고 크기를 동적으로 증가 시킬수 있다.



# LinkedList 클래스

```
List<String> myList = new LinkedList<>();

//자료 추가
myList.add("A");
myList.add("B");
myList.add("D");

System.out.println(myList);

//특정 위치에 자료 추가
myList.add(2, "C");

System.out.println(myList);

//자료 삭제
if(myList.contains("B")) {
    myList.remove("B");
}

System.out.println(myList);

//전체 요소 출력
for(String list : myList)
    System.out.println(list);
```

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

