9. 컬렉션 프레임워크 - Map, Stack, Queue

#1.인강/0.자바/4.자바-중급2편

컬렉션 프레임워크 - Map 소개1

중복X

Key	Value
학생1	90
학생2	80
학생3	80
학생4	100

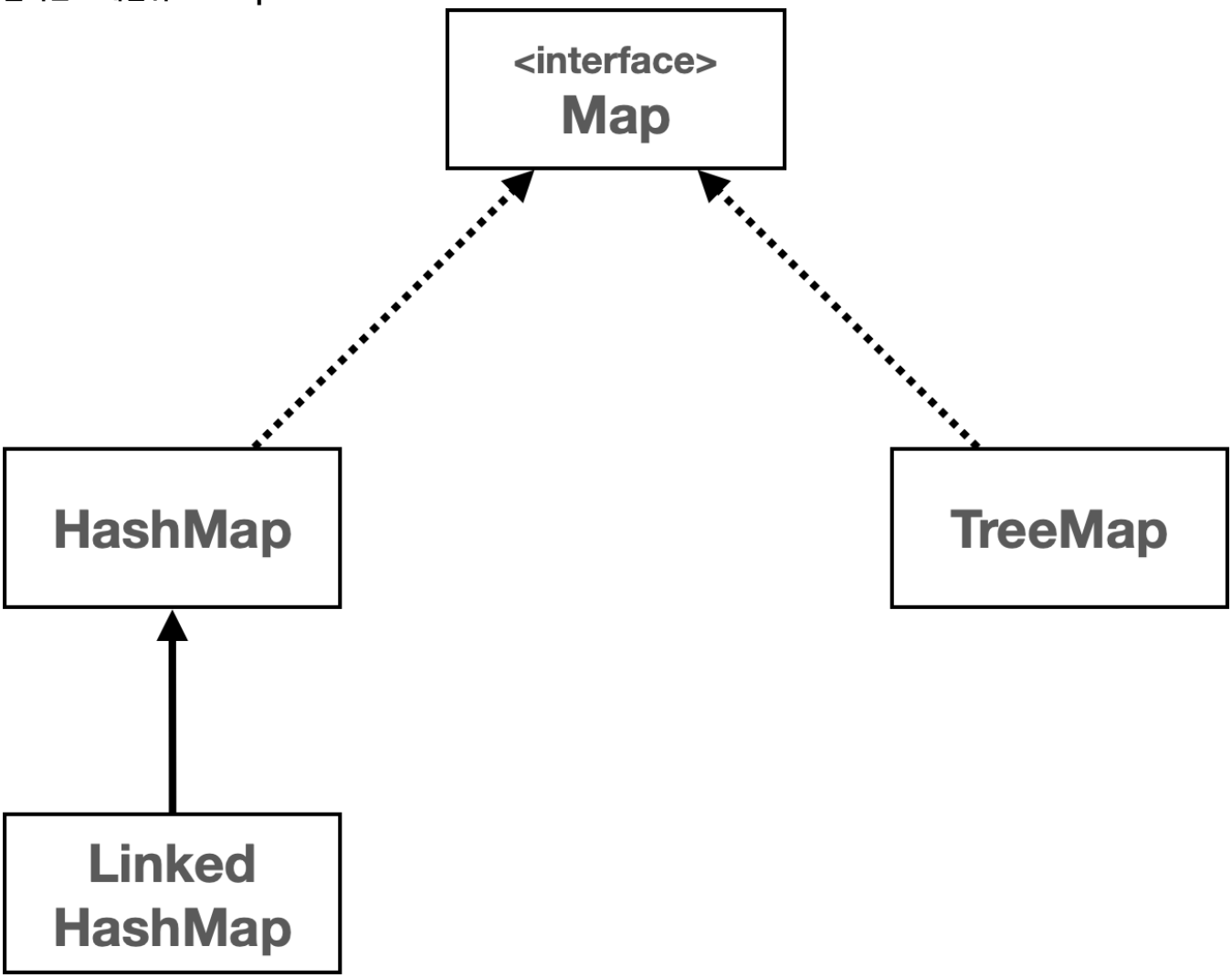
중복 허용

Key	Value
"APPL"	"애플"
"TSLA"	"테슬라"
"CPNG"	"쿠팡"
"NVDA"	"엔비디아"

`Map`은 키-값의 쌍을 저장하는 자료 구조이다.

- 키는 맵 내에서 유일해야 한다. 그리고 키를 통해 값을 빠르게 검색할 수 있다.
- 키는 중복될 수 없지만, 값은 중복될 수 있다.
- `Map`은 순서를 유지하지 않는다.

컬렉션 프레임워크 - Map



자바는 `HashMap`, `TreeMap`, `LinkedHashMap` 등 다양한 `Map` 구현체를 제공한다. 이들은 `Map` 인터페이스의 메서드를 구현하며, 각기 다른 특성과 성능 특징을 가지고 있다.

Map 인터페이스의 주요 메서드

메서드	설명
`put(K key, V value)`	지정된 키와 값을 맵에 저장한다. (같은 키가 있으면 값을 변경)
`putAll(Map<? extends K,? extends V> m)`	지정된 맵의 모든 매핑을 현재 맵에 복사한다.
`putIfAbsent(K key, V value)`	지정된 키가 없는 경우에 키와 값을 맵에 저장한다.
`get(Object key)`	지정된 키에 연결된 값을 반환한다.

`getOrDefault(Object key, V defaultValue)`	지정된 키에 연결된 값을 반환한다. 키가 없는 경우 `defaultValue`로 지정한 값을 대신 반환한다.
`remove(Object key)`	지정된 키와 그에 연결된 값을 맵에서 제거한다.
`clear()`	맵에서 모든 키와 값을 제거한다.
`containsKey(Object key)`	맵이 지정된 키를 포함하고 있는지 여부를 반환한다.
`containsValue(Object value)`	맵이 하나 이상의 키에 지정된 값을 연결하고 있는지 여부를 반환한다.
`keySet()`	맵의 키들을 `Set` 형태로 반환한다.
`values()`	맵의 값들을 `Collection` 형태로 반환한다.
`entrySet()`	맵의 키-값 쌍을 `Set<Map.Entry<K,V>>` 형태로 반환한다.
`size()`	맵에 있는 키-값 쌍의 개수를 반환한다.
`isEmpty()`	맵이 비어 있는지 여부를 반환한다.

이 중에 `HashMap`을 가장 많이 사용한다. 자세한 사용법을 예제 코드로 알아보자.

```
package collection.map;

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

public class MapMain1 {

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

        // 학생 성적 데이터 추가
        studentMap.put("studentA", 90);
        studentMap.put("studentB", 80);
        studentMap.put("studentC", 80);
        studentMap.put("studentD", 100);
```

```
        System.out.println(studentMap);

        // 특정 학생의 값 조회
        Integer result = studentMap.get("studentD");
        System.out.println("result = " + result);

        System.out.println("keySet 활용");
        Set<String> keySet = studentMap.keySet();
        for (String key : keySet) {
            Integer value = studentMap.get(key);
            System.out.println("key=" + key + ", value=" + value);
        }

        System.out.println("entrySet 활용");
        Set<Map.Entry<String, Integer>> entries = studentMap.entrySet();
        for (Map.Entry<String, Integer> entry : entries) {
            String key = entry.getKey();
            Integer value = entry.getValue();
            System.out.println("key=" + key + ", value=" + value);
        }

        System.out.println("values 활용");
        Collection<Integer> values = studentMap.values();
        for (Integer value : values) {
            System.out.println("value = " + value);
        }

    }
}
```

실행 결과

```
{studentB=80, studentA=90, studentD=100, studentC=80}
result = 100

KeySet 활용
key=studentB, value=80
key=studentA, value=90
key=studentD, value=100
key=studentC, value=80

entrySet 활용
```

```
key=studentB, value=80
key=studentA, value=90
key=studentD, value=100
key=studentC, value=80

values 활용
value = 80
value = 90
value = 100
value = 80
```

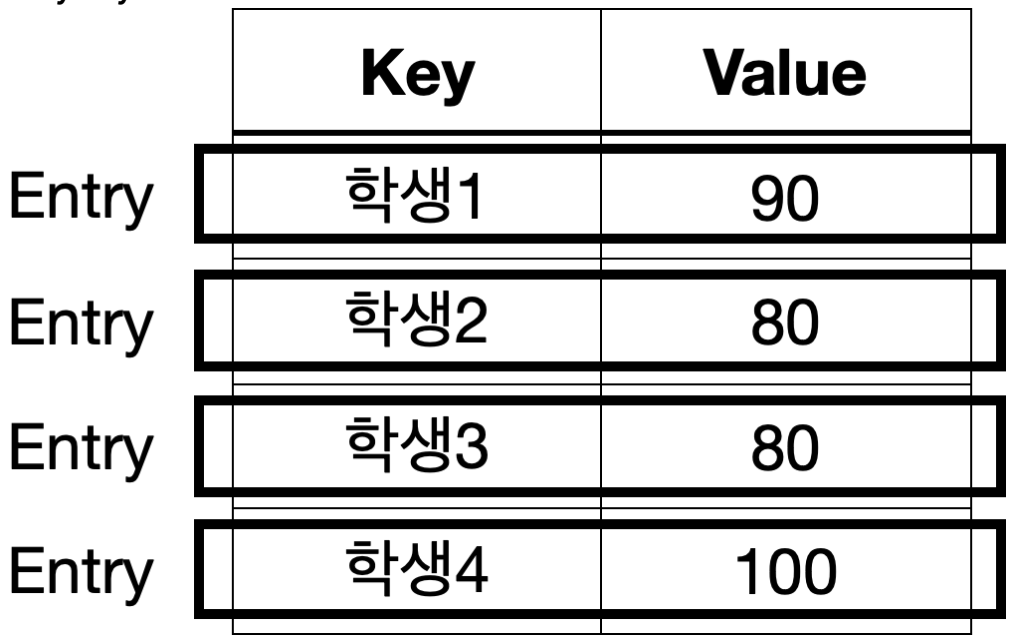
키 목록 조회

`Set<String> keySet = studentMap.keySet()`

`Map`의 키는 중복을 허용하지 않는다. 따라서 `Map`의 모든 키 목록을 조회하는 `keySet()`을 호출하면, 중복을 허용하지 않는 자료 구조인 `Set`을 반환한다.

키와 값 목록 조회

Entry Key-Value Pair



`Map`은 키와 값을 보관하는 자료 구조이다. 따라서 키와 값을 하나로 묶을 수 있는 방법이 필요하다. 이때 `Entry`를 사용한다.

`Entry`는 키-값의 쌍으로 이루어진 간단한 객체이다. `Entiry`는 `Map` 내부에서 키와 값을 함께 묶어서 저장할 때 사용한다.

쉽게 이야기해서 우리가 `Map`에 키와 값으로 데이터를 저장하면 `Map`은 내부에서 키와 값을 하나로 묶는 `Entry` 객체를 만들어서 보관한다. 참고로 하나의 `Map`에 여러 `Entry`가 저장될 수 있다.

참고로 `Entry`는 `Map` 내부에 있는 인터페이스이다. 우리는 구현체보다는 이 인터페이스를 사용하면 된다.

값 목록 조회

`Collection<Integer> values = studentMap.values()`

`Map`의 값 목록은 중복을 허용한다. 따라서 중복을 허용하지 않는 `Set`으로 반환할 수는 없다. 그리고 입력 순서를 보장하지 않기 때문에 순서를 보장하는 `List`로 반환하기도 애매하다. 따라서 단순히 값의 모음이라는 의미의 상위 인터페이스인 `Collection`으로 반환한다.

컬렉션 프레임워크 - Map 소개2

`Map`의 기능을 더 알아보자.

같은 키로 다른 데이터를 저장하면 어떻게 될까?

```
package collection.map;

import java.util.HashMap;
import java.util.Map;

public class MapMain2 {

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

        // 학생 성적 데이터 추가
        studentMap.put("studentA", 90);
        System.out.println(studentMap);

        studentMap.put("studentA", 100); //같은 키에 저장시 기존 값 교체
        System.out.println(studentMap);

        boolean containsKey = studentMap.containsKey("studentA");
        System.out.println("containsKey = " + containsKey);

        // 특정 학생의 값 삭제
        studentMap.remove("studentA");
        System.out.println(studentMap);
    }
}
```

실행 결과

```
{studentA=90}
{studentA=100}
containsKey = true
{}
```

`Map`에 값을 저장할 때 같은 키에 다른 값을 저장하면 기존 값을 교체한다.
`studentA=90`에서 `studentA=100`으로 변경된 것을 확인할 수 있다.

만약 같은 학생이 `Map`에 없는 경우에만 데이터를 저장하려면 어떻게 해야할까?

```
package collection.map;

import java.util.HashMap;
import java.util.Map;

public class MapMain3 {

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

        // 학생 성적 데이터 추가
        studentMap.put("studentA", 50);
        System.out.println(studentMap);

        // 학생이 없는 경우에만 추가1
        if (!studentMap.containsKey("studentA")) {
            studentMap.put("studentA", 100);
        }
        System.out.println(studentMap);

        // 학생이 없는 경우에만 추가2
        studentMap.putIfAbsent("studentA", 100);
        studentMap.putIfAbsent("studentB", 100);
        System.out.println(studentMap);
    }
}
```

실행 결과

```
{studentA=50}
{studentA=50}
{studentB=100, studentA=50}
```

`putIfAbsent()`는 영어 그대로 없는 경우에만 입력하라는 뜻이다. 이 메서드를 사용하면 키가 없는 경우에만 데이터를 저장하고 싶을 때 코드 한줄로 편리하게 처리할 수 있다.

컬렉션 프레임워크 - Map 구현체

자바의 `Map` 인터페이스는 키-값 쌍을 저장하는 자료 구조이다. `Map`은 인터페이스이기 때문에, 직접 인스턴스를 생성할 수는 없고, 대신 `Map` 인터페이스를 구현한 여러 클래스를 통해 사용할 수 있다. 대표적으로 `HashMap`, `TreeMap`, `LinkedHashMap`이 있다.

Map vs Set

그런데 `Map`을 어디서 많이 본 것 같지 않은가? `Map`의 키는 중복을 허용하지 않고, 순서를 보장하지 않는다.
`Map`의 키가 바로 `Set`과 같은 구조이다. 그리고 `Map`은 모든 것이 `Key`를 중심으로 동작한다.
`Value`는 단순히 `Key` 옆에 따라 붙은 것 뿐이다. `Key` 옆에 `Value`만 하나 추가해주면 `Map`이 되는 것이다.
`Map`과 `Set`은 거의 같다. 단지 옆에 `Value`를 가지고 있는가 없는가의 차이가 있을 뿐이다.

Set 자료 구조

중복X

Key
학생1
학생2
학생3
학생4

중복X

Key
"APPL"
"TSLA"
"CPNG"
"NVDA"

Map 자료 구조

중복X 중복 허용

Key	Value
학생1	90
학생2	80
학생3	80
학생4	100

중복X

Key	Value
"APPL"	"애플"
"TSLA"	"테슬라"
"CPNG"	"쿠팡"
"NVDA"	"엔비디아"

이런 이유로 `Set`과 `Map`의 구현체는 거의 같다.

- `HashSet -> HashMap`
- `LinkedHashSet -> LinkedHashMap`
- `TreeSet -> TreeMap`

참고: 실제로 자바 `HashSet`의 구현은 대부분 `HashMap`의 구현을 가져다 사용한다. `Map`에서 `Value`만 비워두면 `Set`으로 사용할 수 있다.

각각의 특징을 알아보자.
참고로 앞서 `Set`에서 학습한 내용과 거의 같다.

1. HashMap:

- **구조**: `HashMap`은 해시를 사용해서 요소를 저장한다. 키(`Key`) 값은 해시 함수를 통해 해시 코드로 변환되고, 이 해시 코드는 데이터를 저장하고 검색하는 데 사용된다.
- **특징**: 삽입, 삭제, 검색 작업은 해시 자료 구조를 사용하므로 일반적으로 상수 시간(`O(1)`)의 복잡도를 가진다.
- **순서**: 순서를 보장하지 않는다.

2. LinkedHashMap:

- **구조**: `LinkedHashMap`은 `HashMap`과 유사하지만, 연결 리스트를 사용하여 삽입 순서 또는 최근 접근 순서에 따라 요소를 유지한다.
- **특징**: 입력 순서에 따라 순회가 가능하다. `HashMap`과 같지만 입력 순서를 링크로 유지해야 하므로 조금 더 무겁다.
- **성능**: `HashMap`과 유사하게 대부분의 작업은 `O(1)`의 시간 복잡도를 가진다.
- **순서**: 입력 순서를 보장한다.

3. TreeMap:

- **구조**: `TreeMap`은 레드-블랙 트리를 기반으로 한 구현이다.
- **특징**: 모든 키는 자연 순서 또는 생성자에 제공된 `Comparator`에 의해 정렬된다.
- **성능**: `get`, `put`, `remove`와 같은 주요 작업들은 `O(log n)`의 시간 복잡도를 가진다.
- **순서**: 키는 정렬된 순서로 저장된다.

참고: 트리에 대한 이론적인 내용은 여기서 다루지 않는다. 트리 구조에 대해서 자세히 알고 싶다면 자료 구조와 알고리즘을 학습하자.

`HashMap`, `LinkedHashMap`, `TreeMap`의 특징을 코드로 확인해보자.

```
package collection.map;

import java.util.*;

public class JavaMapMain {

    public static void main(String[] args) {
        run(new HashMap<>());
        run(new LinkedHashMap<>());
        run(new TreeMap<>());
    }

    private static void run(Map<String, Integer> map) {
        System.out.println("map = " + map.getClass());
        map.put("C", 10);
        map.put("B", 20);
        map.put("A", 30);
        map.put("1", 40);
        map.put("2", 50);
```

```
        Set<String> keySet = map.keySet();
        Iterator<String> iterator = keySet.iterator();
        while (iterator.hasNext()) {
            String key = iterator.next();
            System.out.print(key + "=" + map.get(key) + " ");
        }
        System.out.println();
    }
}
```

실행 결과

```
map = class java.util.HashMap
A=30 1=40 B=20 2=50 C=10

map = class java.util.LinkedHashMap
C=10 B=20 A=30 1=40 2=50

map = class java.util.TreeMap
1=40 2=50 A=30 B=20 C=10
```

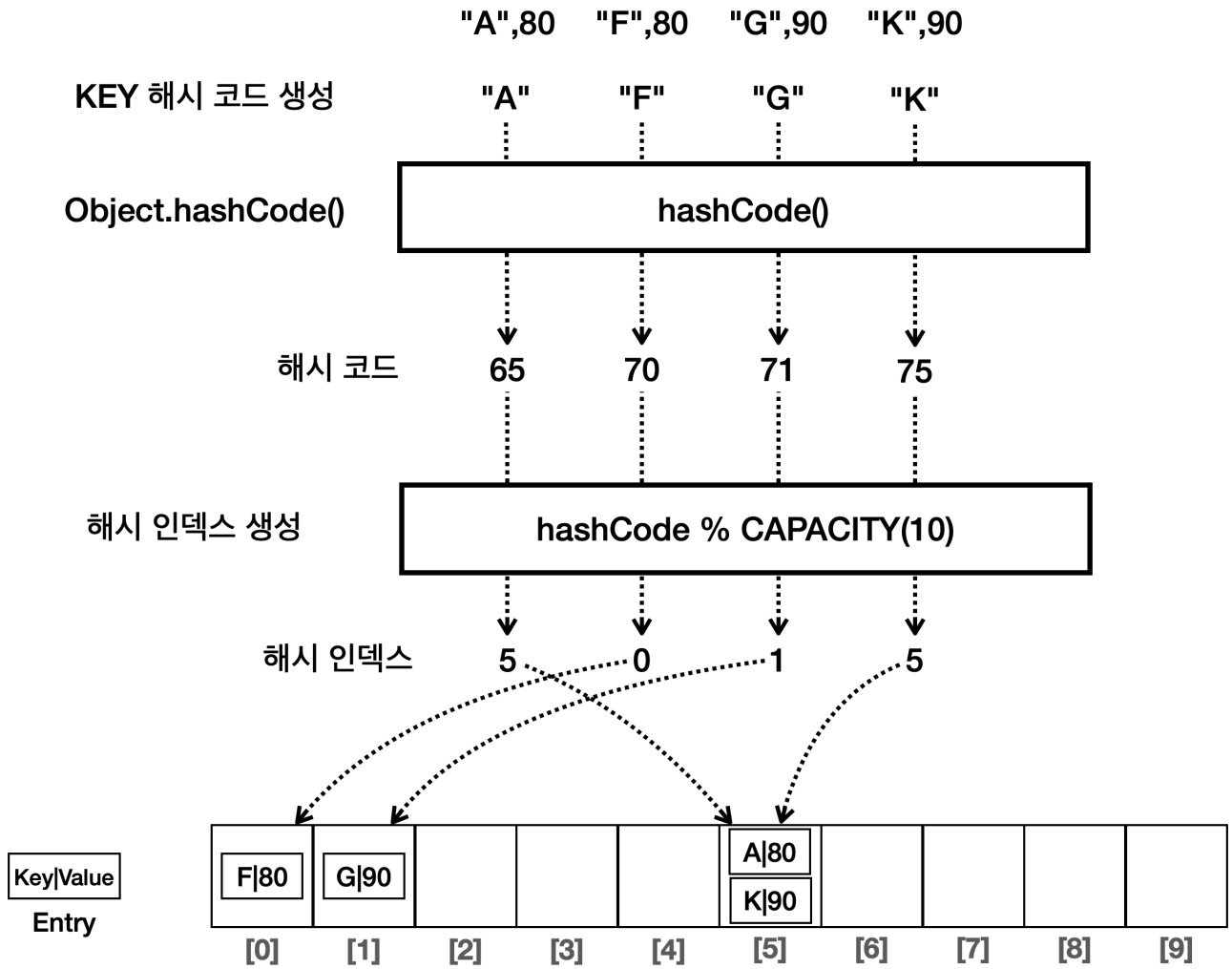
- `HashMap` : 입력한 순서를 보장하지 않는다.
- `LinkedHashMap` : 키를 기준으로 입력한 순서를 보장한다.
- `TreeMap` : 키 자체의 데이터 값을 기준으로 정렬한다.

자바 HashMap 작동 원리

자바의 `HashMap` 은 `HashSet` 과 작동 원리가 같다.

`Set` 과 비교하면 다음과 같은 차이가 있다.

- `Key` 를 사용해서 해시 코드를 생성한다.
- `Key` 뿐만 아니라 값(`Value`)을 추가로 저장해야 하기 때문에 `Entry` 를 사용해서 `Key` , `Value` 를 하나로 묶어서 저장한다.



이렇게 해시를 사용해서 키와 값을 저장하는 자료 구조를 일반적으로 해시 테이블이라 한다.

앞서 학습한 `HashSet`은 해시 테이블의 주요 원리를 사용하지만, 키-값 저장 방식 대신 키만 저장하는 특수한 형태의 해시 테이블로 이해하면 된다.

주의!

`Map`의 `Key`로 사용되는 객체는 `hashCode()`, `equals()`를 반드시 구현해야 한다.

정리

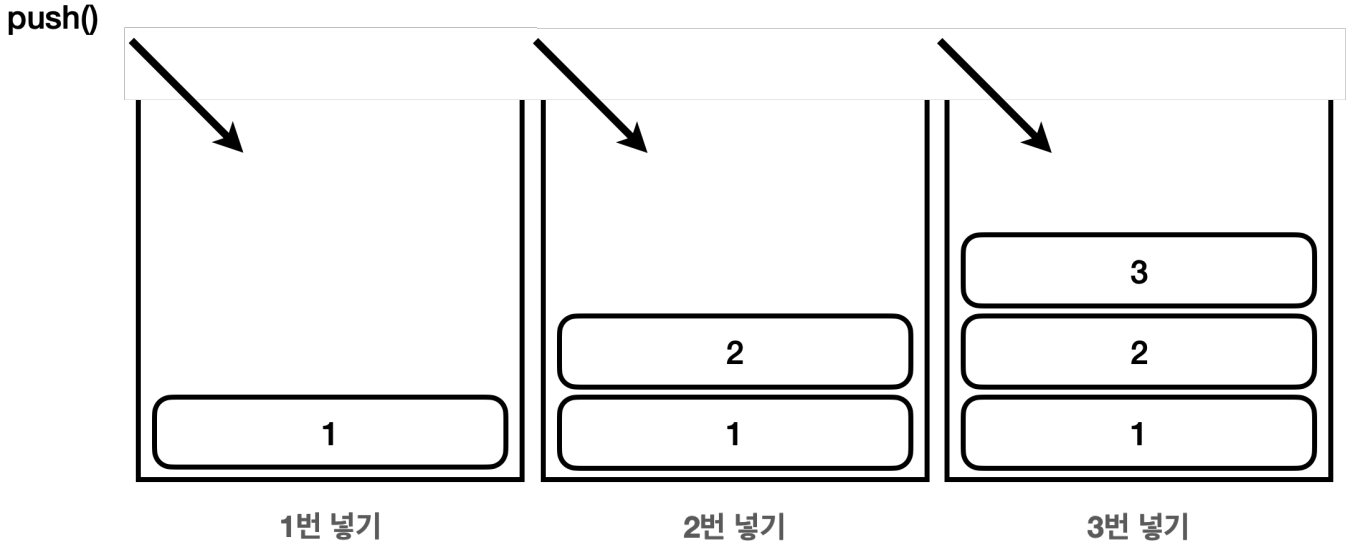
실무에서는 `Map`이 필요한 경우 `HashMap`을 많이 사용한다. 그리고 순서 유지, 정렬의 필요에 따라서 `LinkedHashMap`, `TreeMap`을 선택하면 된다.

스택 자료 구조

스택(Stack) 구조

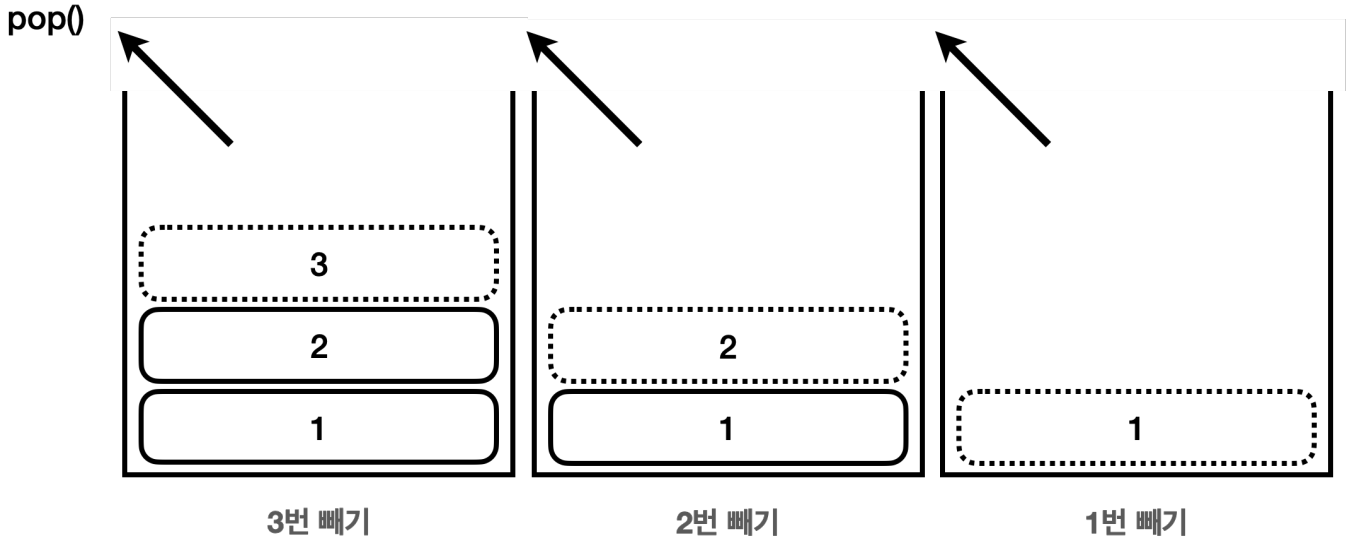
다음과 같은 1, 2, 3 이름표가 붙은 블록이 있다고 가정하자.

이 블록을 다음과 같이 아래쪽은 막혀 있고, 위쪽만 열려있는 통에 넣는다고 생각해보자. 위쪽만 열려있기 때문에 위쪽으로 블록을 넣고, 위쪽으로 블록을 빼야 한다. 쉽게 이야기해서 넣는 곳과 빼는 곳이 같다.



블록은 1➔ 2➔ 3 순서대로 넣는다고 가정하자.

이번에는 넣은 블록을 빼자.



블록을 빼려면 위에서부터 순서대로 빼야한다.
블록은 3➜ 2➜ 1 순서로 뺄 수 있다.

정리하면 다음과 같다.
1(넣기)➜ 2(넣기)➜ 3(넣기)➜ 3(빼기)➜ 2(빼기)➜ 1(빼기)

후입 선출(LIFO, Last In First Out)
여기서 가장 마지막에 넣은 3번이 가장 먼저 나온다. 이렇게 나중에 넣은 것이 가장 먼저 나오는 것을 후입 선출이라 하고, 이런 자료 구조를 스택이라 한다.

전통적으로 스택에 값을 넣는 것을 `push`라 하고, 스택에서 값을 꺼내는 것을 `pop`이라 한다.

자바가 제공하는 스택 자료 구조를 사용해보자.

```
package collection.deque;

import java.util.Stack;

//Stack은 사용하면 안됨 -> Deque를 대신 사용
public class StackMain {

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        stack.push(1);
        stack.push(2);
        stack.push(3);
        System.out.println(stack);

        // 다음 꺼낼 요소 확인(꺼내지 않고 단순 조회만)
        System.out.println("stack.peek() = " + stack.peek());

        // 스택 요소 뽑기
        System.out.println("stack.pop() = " + stack.pop());
        System.out.println("stack.pop() = " + stack.pop());
        System.out.println("stack.pop() = " + stack.pop());
        System.out.println(stack);
    }
}
```

실행 결과

```
[1, 2, 3]
stack.peek() = 3
stack.pop() = 3
stack.pop() = 2
stack.pop() = 1
[]
```

실행 결과를 보면, 1, 2, 3으로 입력하면 3, 2, 1로 출력되는 것을 확인할 수 있다. 나중에 입력한 값이 가장 먼저 나온다.

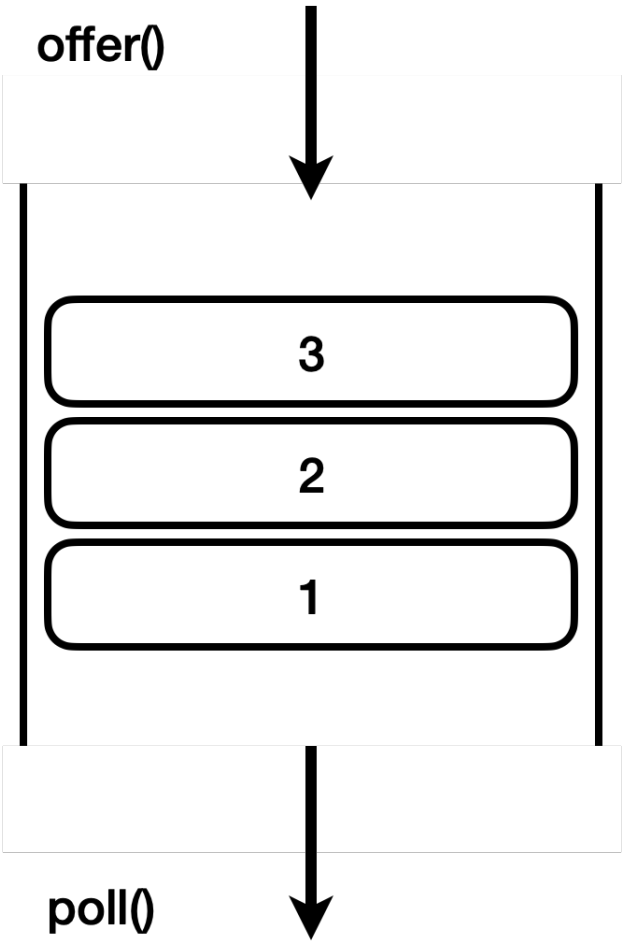
주의!- Stack 클래스는 사용하지 말자

자바의 `Stack` 클래스는 내부에서 `Vector`라는 자료 구조를 사용한다. 이 자료 구조는 자바 1.0에 개발되었는데, 지금은 사용되지 않고 하위 호환을 위해 존재한다. 지금은 더 빠른 좋은 자료 구조가 많다. 따라서 `Vector`를 사용하는 `Stack`도 사용하지 않는 것을 권장한다. 대신에 이후에 설명할 `Deque`를 사용하는 것이 좋다.

큐 자료 구조

선입 선출(FIFO, First In First Out)

후입 선출과 반대로 가장 먼저 넣은 것이 가장 먼저 나오는 것을 선입 선출이라 한다. 이런 자료 구조를 큐(Queue)라 한다.

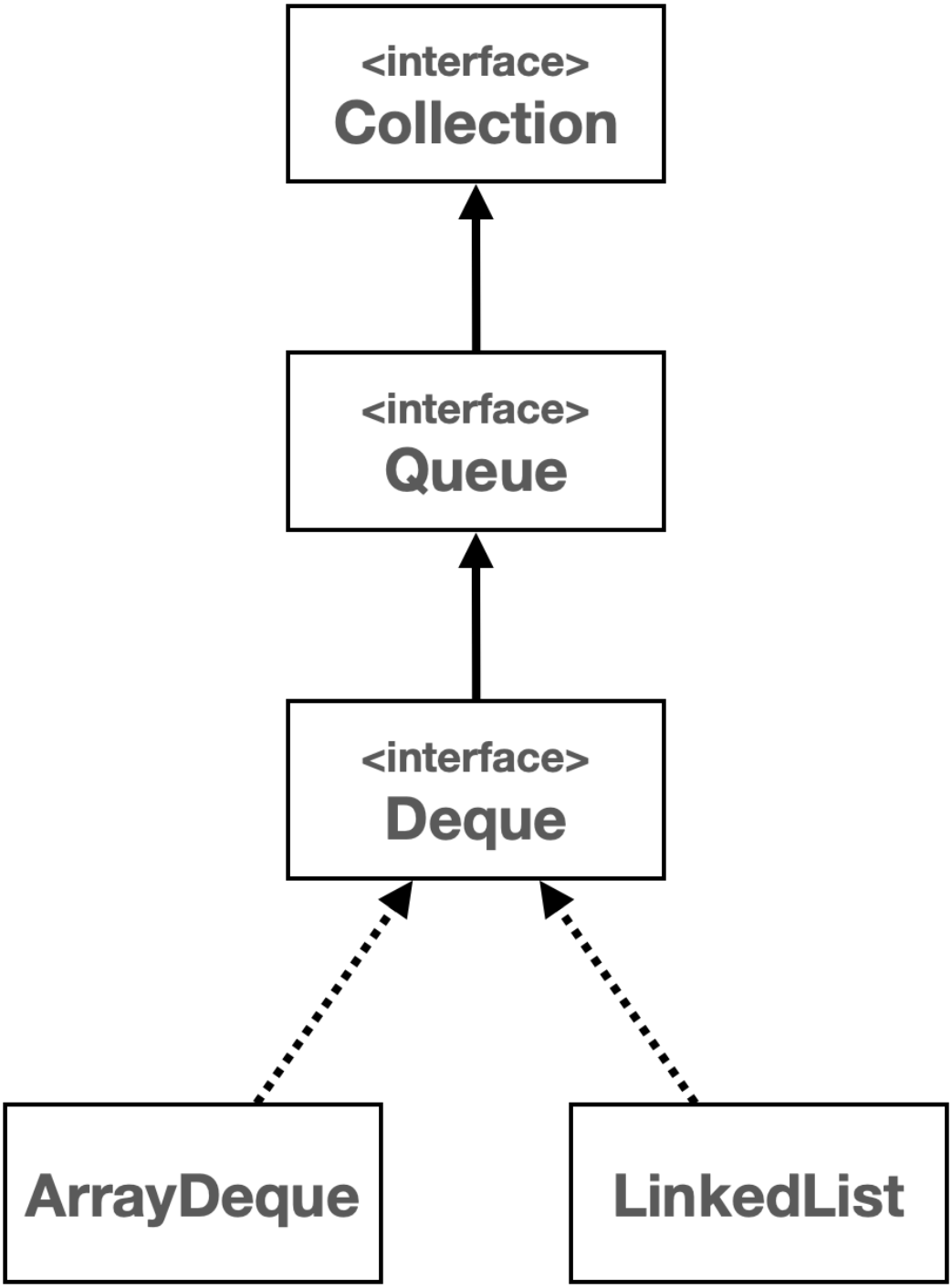


정리하면 다음과 같다.

1(넣기) ➜ 2(넣기) ➜ 3(넣기) ➜ 1(빼기) ➜ 2(빼기) ➜ 3(빼기)

전통적으로 큐에 값을 넣는 것을 `offer`라 하고, 큐에서 값을 꺼내는 것을 `poll`이라 한다.

컬렉션 프레임워크 - Queue



- `Queue` 인터페이스는 `List`, `Set`과 같이 `Collection`의 자식이다.
- `Queue`의 대표적인 구현체는 `ArrayDeque`, `LinkedList`가 있다.
- `Deque`는 조금 뒤에 설명한다.

참고로 `LinkedList`는 `Deque`와 `List` 인터페이스를 모두 구현한다.

```
public class LinkedList<E> extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, java.io.Serializable {}
```

`ArrayDeque`를 통해 `Queue`를 사용해보자.

```java
package collection.deque;

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

public class QueueMain {

    public static void main(String[] args) {
        Queue<Integer> queue = new ArrayDeque<>();
        //Queue<Integer> queue = new LinkedList<>();

        //데이터 추가
        queue.offer(1);
        queue.offer(2);
        queue.offer(3);
        System.out.println(queue);

        //다음 꺼낼 데이터 확인(꺼내지 않고 단순 조회만)
        System.out.println("queue.peek() = " + queue.peek());

        //데이터 꺼내기
        System.out.println("poll = " + queue.poll());
        System.out.println("poll = " + queue.poll());
        System.out.println("poll = " + queue.poll());
        System.out.println(queue);
    }
}
```

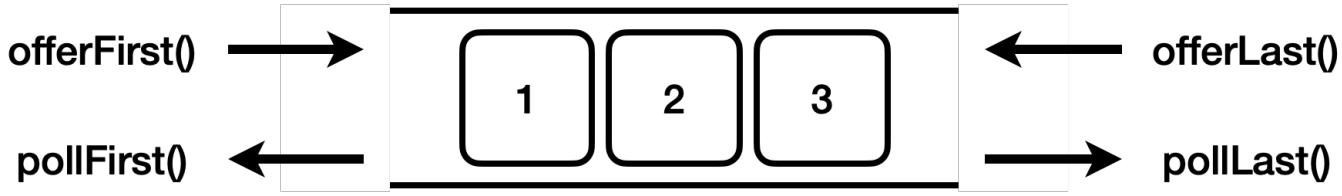
실행 결과

```
[1, 2, 3]
queue.peek() = 1
poll = 1
poll = 2
poll = 3
[]
```

실행 결과를 보면 1, 2, 3으로 입력하면 1, 2, 3으로 출력되는 것을 확인할 수 있다. 가장 먼저 입력한 값이 가장 먼저 나온다.

Deque 자료 구조

"Deque"는 "Double Ended Queue"의 약자로, 이 이름에서 알 수 있듯이, Deque는 양쪽 끝에서 요소를 추가하거나 제거할 수 있다. Deque는 일반적인 큐(Queue)와 스택(Stack)의 기능을 모두 포함하고 있어, 매우 유연한 자료 구조이다.
데크, 덱 등으로 부른다.



- `offerFirst()` : 앞에 추가한다.
- `offerLast()` : 뒤에 추가한다.
- `pollFirst()` : 앞에서 꺼낸다.
- `pollLast()` : 뒤에서 꺼낸다.

`Deque`의 대표적인 구현체는 `ArrayDeque`, `LinkedList`가 있다.

```
package collection.deque;

import java.util.ArrayDeque;
import java.util.Deque;

public class DequeMain {

    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        //Deque<Integer> deque = new LinkedList<>();

        // 데이터 추가
        deque.offerFirst(1);
        System.out.println(deque);
```

```
        deque.offerFirst(2);
        System.out.println(deque);
        deque.offerLast(3);
        System.out.println(deque);
        deque.offerLast(4);
        System.out.println(deque);

        // 다음 꺼낼 데이터 확인(꺼내지 않고 단순 조회만)
        System.out.println("deque.peekFirst() = " + deque.peekFirst());
        System.out.println("deque.peekLast() = " + deque.peekLast());

        // 데이터 꺼내기
        System.out.println("pollFirst = " + deque.pollFirst());
        System.out.println("pollFirst = " + deque.pollFirst());
        System.out.println("pollLast = " + deque.pollLast());
        System.out.println("pollLast = " + deque.pollLast());
        System.out.println(deque);
    }
}
```

실행 결과

```
[1]
[2, 1]
[2, 1, 3]
[2, 1, 3, 4]
deque.peekFirst() = 2
deque.peekLast() = 4
pollFirst = 2
pollFirst = 1
pollLast = 4
pollLast = 3
[]
```

입력 순서는 다음과 같다.

- 앞으로 1을 추가한다. [1]
- 앞으로 2를 추가한다. [2, 1] (앞으로 2를 추가했으므로 기존에 있던 1이 뒤로 밀려난다)
- 뒤로 3을 추가한다. [2, 1, 3]
- 뒤로 4를 추가한다. [2, 1, 3, 4]

앞에서 2번 꺼내면 2, 1이 꺼내진다. 다음으로 뒤에서 2번 꺼내면 4, 3이 꺼내진다.

Deque 구현체와 성능 테스트

`Deque`의 대표적인 구현체는 `ArrayDeque`, `LinkedList`가 있다. 이 둘 중에 `ArrayDeque`가 모든 면에서 더 빠르다.

100만 건 입력(앞, 뒤 평균)

- `ArrayDeque`: 110ms
- `LinkedList`: 480ms

100만 건 조회(앞, 뒤 평균)

- `ArrayDeque`: 9ms
- `LinkedList`: 20ms

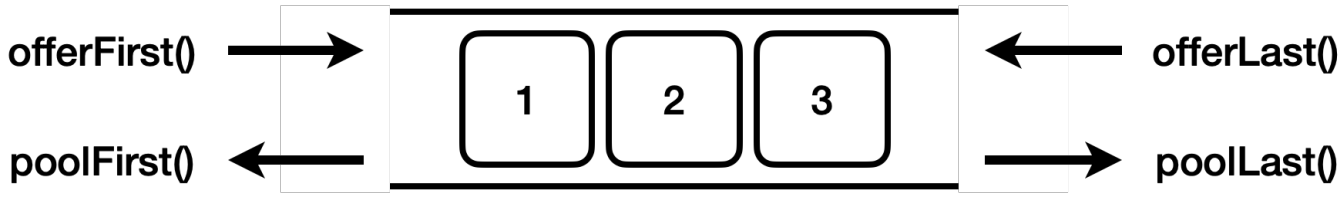
둘의 차이는 `ArrayList` vs `LinkedList`의 차이와 비슷한데, 작동 원리가 하나는 배열을 하나는 동적 노드 링크를 사용하기 때문이다.
`ArrayDeque`는 추가로 특별한 원형 큐 자료 구조를 사용하는데, 덕분에 앞, 뒤 입력 모두 O(1)의 성능을 제공한다.
물론 `LinkedList`도 앞 뒤 입력 모두 O(1)의 성능을 제공한다.
이론적으로 `LinkedList`가 삽입 삭제가 자주 발생할 때 더 효율적일 수 있지만, 현대 컴퓨터 시스템의 메모리 접근 패턴, CPU 캐시 최적화 등을 고려할 때 배열을 사용하는 `ArrayDeque`가 실제 사용 환경에서 더 나은 성능을 보여주는 경우가 많다.

참고: 원형 큐에 대한 이론적인 내용은 여기서 다루지 않는다. 해당 내용을 자세히 알고 싶다면 자료 구조와 알고리즘을 학습하자.

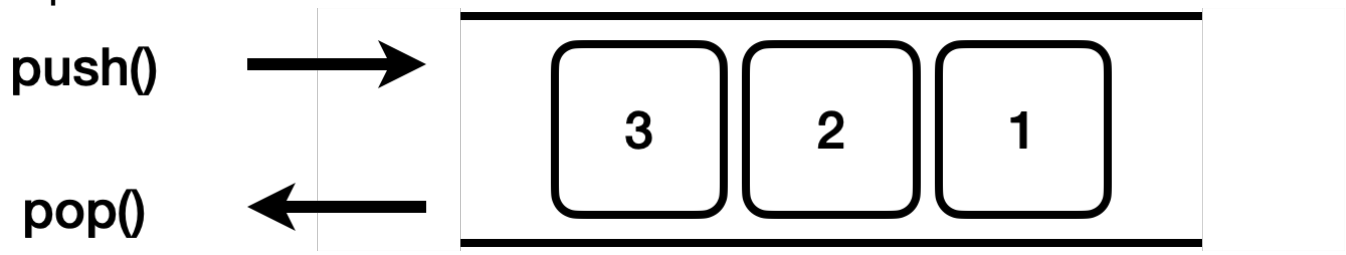
Deque와 Stack, Queue

`Deque`는 양쪽으로 데이터를 입력하고 출력할 수 있으므로, 스택과 큐의 역할을 모두 수행할 수 있다.
`Deque`를 `Stack`과 `Queue`로 사용하기 위한 메서드 이름까지 제공한다.

Deque

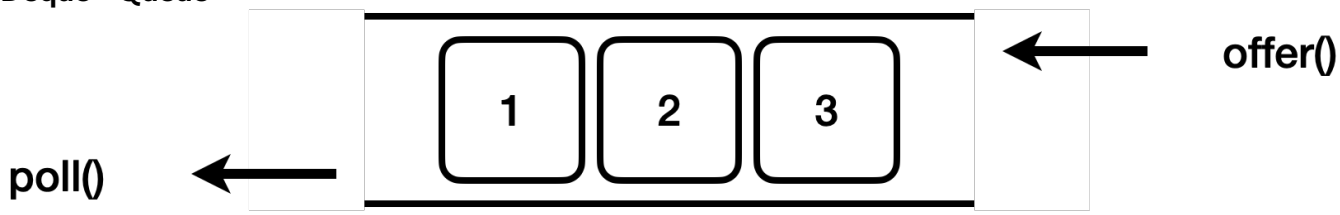


Deque - Stack



- `push()`를 호출하면 앞에서 입력한다.
- `pop()`을 호출하면 앞에서 꺼낸다.

Deque - Queue



- `offer()`를 호출하면 뒤에서 입력한다.
- `poll()`을 호출하면 앞에서 꺼낸다.

Deque - Stack

```
package collection.deque;

import java.util.ArrayDeque;
import java.util.Deque;

public class DequeStackMain {

    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        //Deque<Integer> deque = new LinkedList<>();

        // 데이터 추가
        deque.push(1);
        deque.push(2);
```

```
        deque.push(3);
        System.out.println(deque);

        // 다음 꺼낼 데이터 확인(꺼내지 않고 단순 조회만)
        System.out.println("deque.peek() = " + deque.peek());

        // 데이터 꺼내기
        System.out.println("pop = " + deque.pop());
        System.out.println("pop = " + deque.pop());
        System.out.println("pop = " + deque.pop());
        System.out.println(deque);
    }
}
```

실행 결과

```
[3, 2, 1]
deque.peek() = 3
pop = 3
pop = 2
pop = 1
[]
```

`Deque`에서 `Stack`을 위한 메서드 이름까지 제공하는 것을 확인할 수 있다. 자바의 `Stack` 클래스는 성능이 좋지 않고 하위 호환을 위해서 남겨져 있다. `Stack` 자료 구조가 필요하면 `Deque`에 `ArrayDeque` 구현체를 사용하자.

Deque - Queue

```
package collection.deque;

import java.util.ArrayDeque;
import java.util.Deque;

public class DequeQueueMain {

    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        //Deque<Integer> deque = new LinkedList<>();
```

```
        //데이터 추가
        deque.offer(1);
        deque.offer(2);
        deque.offer(3);
        System.out.println(deque);

        //다음 꺼낼 데이터 확인(꺼내지 않고 단순 조회만)
        System.out.println("deque.peek() = " + deque.peek());

        //데이터 꺼내기
        System.out.println("poll = " + deque.poll());
        System.out.println("poll = " + deque.poll());
        System.out.println("poll = " + deque.poll());
        System.out.println(deque);
    }
}
```

실행 결과

```
[1, 2, 3]
deque.peek() = 1
poll = 1
poll = 2
poll = 3
[]
```

`Deque`에서 `Queue`을 위한 메서드 이름까지 제공하는 것을 확인할 수 있다. `Deque` 인터페이스는 `Queue` 인터페이스의 자식이기 때문에, 단순히 `Queue`의 기능만 필요하면 `Queue` 인터페이스를 사용하고, 더 많은 기능이 필요하다면 `Deque` 인터페이스를 사용하면 된다. 그리고 구현체로 성능이 빠른 `ArrayDeque`를 사용하자.

문제와 풀이1 - Map1

문제1 - 배열을 맵으로 전환

문제 설명

상품의 이름과 가격이 2차원 배열로 제공된다.
다음 예제를 참고해서 2차원 배열의 데이터를 `Map<String, Integer>`로 변경해라.
그리고 실행 결과를 참고해서 `Map`을 출력해라. 출력 순서는 상관없다.

```
package collection.map.test;

public class ArrayToMapTest {
    public static void main(String[] args) {
        String[][] productArr = {{"Java", "10000"}, {"Spring", "20000"},
{"JPA", "30000"}};

        // 주어진 배열로부터 Map 생성 - 코드 작성

        // Map의 모든 데이터 출력  - 코드 작성
    }
}
```

실행 결과

```
제품: Java, 가격: 10000
제품: JPA, 가격: 30000
제품: Spring, 가격: 20000
```

정답

```
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class ArrayToMapTest {
    public static void main(String[] args) {
        String[][] productArr = {{"Java", "10000"}, {"Spring", "20000"},
{"JPA", "30000"}};

        // 주어진 배열로부터 Map 생성
```

```
        Map<String, Integer> productMap = new HashMap<>();
        for (String[] product : productArr) {
            productMap.put(product[0], Integer.valueOf(product[1]));
        }

        // Map의 모든 데이터 출력
        for (String key : productMap.keySet()) {
            System.out.println("제품: " + key + ", 가격: " +
productMap.get(key));
        }
    }
}
```

문제2 - 공통의 합

문제 설명

`map1`과 `map2`에 공통으로 들어있는 키를 찾고, 그 값의 합을 구해라.

실행 결과를 참고하자.

```
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class CommonKeyValueSum1 {
    public static void main(String[] args) {
        Map<String, Integer> map1 = new HashMap<>();
        map1.put("A", 1);
        map1.put("B", 2);
        map1.put("C", 3);

        Map<String, Integer> map2 = new HashMap<>();
        map2.put("B", 4);
        map2.put("C", 5);
        map2.put("D", 6);

        // 코드 작성
    }
}
```

실행 결과

```
{B=6, C=8}
```

정답1

```
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class CommonKeyValueSum1 {
    public static void main(String[] args) {
        Map<String, Integer> map1 = new HashMap<>();
        map1.put("A", 1);
        map1.put("B", 2);
        map1.put("C", 3);

        Map<String, Integer> map2 = new HashMap<>();
        map2.put("B", 4);
        map2.put("C", 5);
        map2.put("D", 6);

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

        for (String key : map1.keySet()) {
            if (map2.containsKey(key)) {
                result.put(key, map1.get(key) + map2.get(key));
            }
        }

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

정답2

```java
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class CommonKeyValueSum2 {
    public static void main(String[] args) {
        Map<String, Integer> map1 = Map.of("A", 1, "B", 2, "C", 3);
        Map<String, Integer> map2 = Map.of("B", 4, "C", 5, "D", 6);

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

        for (String key : map1.keySet()) {
            if (map2.containsKey(key)) {
                result.put(key, map1.get(key) + map2.get(key));
            }
        }

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

- `Map`을 생성할 때 `Map.of()`를 사용하면 편리하게 `Map`을 생성할 수 있다.
- `Map.of()`를 사용해서 생성한 `Map`은 불변이다. 따라서 내용을 변경할 수 없다.

문제3 - 같은 단어가 나타난 수

문제 설명

각각의 단어가 나타난 수를 출력해라.

실행 결과를 참고하자.

```java
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class WordFrequencyTest1 {
    public static void main(String[] args) {
        String text = "orange banana apple apple banana apple";
```

```
        // 코드 작성
    }
}
```

실행 결과

```
{orange=1, banana=2, apple=3}
```

정답1

```
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class WordFrequencyTest1 {
    public static void main(String[] args) {
        String text = "orange banana apple apple banana apple";

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

        String[] words = text.split(" ");

        for (String word : words) {
            Integer count = map.get(word);
            if (count == null) {
                count = 0;
            }
            count++;

            map.put(word, count);
        }

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

정답2

```java
package collection.map.test;

import java.util.HashMap;
import java.util.Map;

public class WordFrequencyTest2 {
    public static void main(String[] args) {
        String text = "orange banana apple apple banana apple";

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

        String[] words = text.split(" ");

        for (String word : words) {
            map.put(word, map.getOrDefault(word, 0) + 1);
        }

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

- `getOrDefault()` 메서드를 사용하면 키가 없는 경우 대신 사용할 기본 값을 지정할 수 있다.

문제4 - 값으로 검색

문제 설명

다음 예제에서 `Map`에 들어있는 데이터 중에 값이 1000원인 모든 상품을 출력해라.

실행 결과를 참고하자.

```java
package collection.map.test;

import java.util.*;

public class ItemPriceTest {

    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("사과", 500);
        map.put("바나나", 500);
        map.put("망고", 1000);
```

```
        map.put("딸기", 1000);

        // 코드 작성
    }
}
```

실행 결과

```
[망고, 딸기]
```

정답

```
package collection.map.test;

import java.util.*;

public class ItemPriceTest {

    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("사과", 500);
        map.put("바나나", 500);
        map.put("망고", 1000);
        map.put("딸기", 1000);

        // 코드 작성
        List<String> list = new ArrayList<>();
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            if (entry.getValue().equals(1000)) {
                list.add(entry.getKey());
            }
        }

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

문제와 풀이2 - Map2

문제5 - 영어 사전 만들기

문제 설명

영어 단어를 입력하면 한글 단어를 찾아주는 영어 사전을 만들자.

- 먼저 영어 단어와 한글 단어를 사전에 저장하는 단계를 거친다.
- 이후에 단어를 검색한다.

실행 결과를 참고하자.

```
package collection.map.test;

public class DictionaryTest {
    // 코드 작성
}
```

실행 결과

```
==단어 입력 단계==
영어 단어를 입력하세요 (종료는 'q'): apple
한글 뜻을 입력하세요: 사과
영어 단어를 입력하세요 (종료는 'q'): banana
한글 뜻을 입력하세요: 바나나
영어 단어를 입력하세요 (종료는 'q'): q

==단어 검색 단계==
찾을 영어 단어를 입력하세요 (종료는 'q'): apple
apple의 뜻: 사과
찾을 영어 단어를 입력하세요 (종료는 'q'): banana
banana의 뜻: 바나나
찾을 영어 단어를 입력하세요 (종료는 'q'): hello
hello은(는) 사전에 없는 단어입니다.
찾을 영어 단어를 입력하세요 (종료는 'q'): q
```

정답

```java
package collection.map.test;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class DictionaryTest {
    public static void main(String[] args) {
        Map<String, String> dictionary = new HashMap<>();
        Scanner scanner = new Scanner(System.in);

        System.out.println("==단어 입력 단계==");
        while (true) {
            System.out.print("영어 단어를 입력하세요 (종료는 'q'): ");
            String englishWord = scanner.nextLine();

            if (englishWord.equals("q")) {
                break;
            }

            System.out.print("한글 뜻을 입력하세요: ");
            String koreanMeaning = scanner.nextLine();

            dictionary.put(englishWord, koreanMeaning);
        }

        System.out.println("==단어 검색 단계==");
        while (true) {
            System.out.print("찾을 영어 단어를 입력하세요 (종료는 'q'): ");
            String searchWord = scanner.nextLine();

            if (searchWord.equals("q")) {
                break;
            }

            if (dictionary.containsKey(searchWord)) {
                String koreanMeaning = dictionary.get(searchWord);
                System.out.println(searchWord + "의 뜻: " + koreanMeaning);
            } else {
                System.out.println(searchWord + "은(는) 사전에 없는 단어입니다.");
            }
        }
```

```
    }
}
```

문제6 - 회원 관리 저장소

문제 설명

- `Map`을 사용해서 회원을 저장하고 관리하는 `MemberRepository` 코드를 완성하자.
- `Member`, `MemberRepositoryMain` 코드와 실행 결과를 참고하자.

```
package collection.map.test.member;

public class Member {
    private String id;
    private String name;

    public Member(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Member{" +
                "id='" + id + '\'' +
                ", name='" + name + '\'' +
                '}';
    }
}
```

```
```

```java
package collection.map.test.member;

public class MemberRepositoryMain {

    public static void main(String[] args) {
        Member member1 = new Member("id1", "회원1");
        Member member2 = new Member("id2", "회원2");
        Member member3 = new Member("id3", "회원3");

        // 회원 저장
        MemberRepository repository = new MemberRepository();
        repository.save(member1);
        repository.save(member2);
        repository.save(member3);

        // 회원 조회
        Member findMember1 = repository.findById("id1");
        System.out.println("findMember1 = " + findMember1);

        Member findMember3 = repository.findByName("회원3");
        System.out.println("findMember3 = " + findMember3);

        // 회원 삭제
        repository.remove("id1");
        Member removedMember1 = repository.findById("id1");
        System.out.println("removedMember1 = " + removedMember1);
    }
}
```

MemberRepository 코드 작성

```java
package collection.map.test.member;

import java.util.HashMap;
import java.util.Map;

public class MemberRepository {

    private Map<String, Member> memberMap = new HashMap<>();

    public void save(Member member) {
        // 코드 작성
```

```
    }

    public void remove(String id) {
        // 코드 작성
    }

    public Member findById(String id) {
        // 코드 작성
    }

    public Member findByName(String name) {
        // 코드 작성
    }
}
```

실행 결과

```
findMember1 = Member{id='id1', name='회원1'}
findMember3 = Member{id='id3', name='회원3'}
removedMember1 = null
```

정답

```
package collection.map.test.member;

import java.util.HashMap;
import java.util.Map;

public class MemberRepository {

    private Map<String, Member> memberMap = new HashMap<>();

    public void save(Member member) {
        memberMap.put(member.getId(), member);
    }

    public void remove(String id) {
        memberMap.remove(id);
    }
```

```
    public Member findById(String id) {
        return memberMap.get(id);
    }

    public Member findByName(String name) {
        for (Member member : memberMap.values()) {
            if (member.getName().equals(name)) {
                return member;
            }
        }
        return null;
    }
}
```

문제7 - 장바구니

문제 설명

- **장바구니 추가 - add()**
  - 장바구니에 상품과 수량을 담는다. **상품의 이름과 가격이 같으면 같은 상품**이다.
  - 장바구니에 이름과 가격이 같은 상품을 추가하면 기존에 담긴 상품에 수량만 추가된다.
  - 장바구니에 이름과 가격이 다른 상품을 추가하면 새로운 상품이 추가된다.
- **장바구니 제거 - minus()**
  - 장바구니에 담긴 상품의 수량을 줄일 수 있다. 만약 수량이 0보다 작다면 상품이 장바구니에서 제거된다.
- `CartMain`과 실행 결과를 참고해서 `Product`, `Cart` 클래스를 완성하자.
- `Cart` 클래스는 `Map`을 통해 상품을 장바구니에 보관한다.
  - `Map`의 Key는 `Product`가 사용되고, Value는 장바구니에 담은 수량이 사용된다.

```
package collection.map.test.cart;

public class CartMain {

    public static void main(String[] args) {
        Cart cart = new Cart();
        cart.add(new Product("사과", 1000), 1);
        cart.add(new Product("바나나", 500), 1);
        cart.printAll();
```

```
        cart.add(new Product("사과", 1000), 2);
        cart.printAll();

        cart.minus(new Product("사과", 1000), 3);
        cart.printAll();
    }
}
```

실행 결과

```
==모든 상품 출력==
상품: Product{name='사과', price=1000} 수량: 1
상품: Product{name='바나나', price=500} 수량: 1
==모든 상품 출력==
상품: Product{name='사과', price=1000} 수량: 3
상품: Product{name='바나나', price=500} 수량: 1
==모든 상품 출력==
상품: Product{name='바나나', price=500} 수량: 1
```

```
package collection.map.test.cart;

public class Product {

    private String name;
    private int price;

    // 코드 작성
}
```

```
package collection.map.test.cart;

import java.util.HashMap;
import java.util.Map;

public class Cart {
```

```
    private Map<Product, Integer> cartMap = new HashMap<>();
    // 코드 작성
}
```

정답 - Product

```
package collection.map.test.cart;

import java.util.Objects;

public class Product {

    private String name;
    private int price;

    public Product(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Product product = (Product) o;
        return price == product.price && Objects.equals(name, product.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, price);
    }

    @Override
```

```
    public String toString() {
        return "Product{" +
                "name='" + name + '\'' +
                ", price=" + price +
                '}';
    }
}
```

- `Map`의 Key로 `Product`가 사용된다. 따라서 반드시 `hashCode()`, `equals()`를 재정의해야 한다.

정답 - Cart

```
package collection.map.test.cart;

import java.util.HashMap;
import java.util.Map;

public class Cart {

    private Map<Product, Integer> cartMap = new HashMap<>();

    public void add(Product product, int addQuantity) {
        int existingQuantity = cartMap.getOrDefault(product, 0);
        cartMap.put(product, existingQuantity + addQuantity);
    }

    public void minus(Product product, int minusQuantity) {
        int existingQuantity = cartMap.getOrDefault(product, 0);

        int newQuantity = existingQuantity - minusQuantity;
        if (newQuantity <= 0) {
            cartMap.remove(product);
        } else {
            cartMap.put(product, newQuantity);
        }
    }

    public void printAll() {
        System.out.println("==모든 상품 출력==");
        for (Map.Entry<Product, Integer> entry : cartMap.entrySet()) {
            System.out.println("상품: " + entry.getKey() + " 수량: " +
entry.getValue());
        }
```

```
    }
}
```

문제와 풀이3 - Stack

문제1 - 간단한 히스토리 확인

- 스택에 `push()`를 통해서 다음 데이터를 순서대로 입력해라.
  - `"youtube.com"`
  - `"google.com"`
  - `"facebook.com"`
- 스택에 `pop()`을 통해서 데이터를 꺼내고, 꺼낸 순서대로 출력해라.
  - `"facebook.com"`
  - `"google.com"`
  - `"youtube.com"`

입력 순서와 반대로 출력되는 것을 확인할 수 있다. 가장 마지막에 입력한 데이터가 가장 먼저 출력된다.

```
public class SimpleHistoryTest {

    public static void main(String[] args) {
        Deque<String> stack = new ArrayDeque<>();
        // 코드 작성
    }
}
```

- `Stack`을 사용해도 되지만 `Deque` 인터페이스에 `ArrayDeque` 구현체를 사용하는 것이 성능상 더 나은 선택이다.

실행 결과

```
facebook.com
google.com
youtube.com
```

정답

```
package collection.deque.test.stack;

import java.util.ArrayDeque;
import java.util.Deque;

public class SimpleHistoryTest {

    public static void main(String[] args) {
        Deque<String> stack = new ArrayDeque<>();

        stack.push("youtube.com");
        stack.push("google.com");
        stack.push("facebook.com");

        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
    }
}
```

문제2 - 브라우저 히스토리 관리

`BrowserHistoryTest`와 실행 결과를 참고해서 `BrowserHistory` 클래스를 완성하자.
브라우저의 방문 기록 관리 기능을 개발하자. 다음 기능을 개발해야 한다.

- `visitPage()` : 특정 페이지 방문
- `goBack()` : 뒤로가기

뒤로가기는 가장 나중에 넣은 데이터가 먼저 나온다. 따라서 스택 구조를 고려하는 것이 좋다.

문제 설명

```
package collection.deque.test.stack;

public class BrowserHistoryTest {
```

```
    public static void main(String[] args) {
        BrowserHistory browser = new BrowserHistory();

        // 사용자가 웹페이지를 방문하는 시나리오
        browser.visitPage("youtube.com");
        browser.visitPage("google.com");
        browser.visitPage("facebook.com");

        // 뒤로 가기 기능을 사용하는 시나리오
        String currentPage1 = browser.goBack();
        System.out.println("currentPage1 = " + currentPage1);

        String currentPage2 = browser.goBack();
        System.out.println("currentPage2 = " + currentPage2);
    }
}
```

실행 결과

```
방문: youtube.com
방문: google.com
방문: facebook.com
뒤로 가기: google.com
currentPage1 = google.com
뒤로 가기: youtube.com
currentPage2 = youtube.com
```

BrowserHistory 코드 작성

```
package collection.deque.test.stack;

public class BrowserHistory {
    // 코드 작성
}
```

정답

```java
package collection.deque.test.stack;

import java.util.ArrayDeque;
import java.util.Deque;

public class BrowserHistory {
    private Deque<String> history = new ArrayDeque<>();
    private String currentPage = null;

    public void visitPage(String url) {
        if (currentPage != null) {
            history.push(currentPage);
        }
        currentPage = url;
        System.out.println("방문: " + url);
    }

    public String goBack() {
        if (!history.isEmpty()) {
            currentPage = history.pop();
            System.out.println("뒤로 가기: " + currentPage);
            return currentPage;
        }
        return null;
    }

}
```

문제와 풀이4 - Queue

문제1 - 프린터 대기

- 프린터에 여러 문서의 출력을 요청하면 한번에 모든 문서를 출력할 수 없다. 따라서 순서대로 출력해야 한다.
- 문서가 대기할 수 있도록 큐 구조를 사용하자.
- "doc1", "doc2", "doc3" 문서를 순서대로 입력하고, 입력된 순서대로 출력하자.
- 실행 결과를 참고하자.

문제 설명

```
package collection.deque.test.queue;

import java.util.ArrayDeque;
import java.util.Queue;

public class PrinterQueueTest {
    public static void main(String[] args) {
        Queue<String> printQueue = new ArrayDeque<>();

        // 코드 작성
    }
}
```

실행 결과

```
출력: doc1
출력: doc2
출력: doc3
```

정답

```
package collection.deque.test.queue;

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

public class PrinterQueueTest {
    public static void main(String[] args) {
        Queue<String> printQueue = new LinkedList<>();

        printQueue.offer("doc1");
        printQueue.offer("doc2");
        printQueue.offer("doc3");

        System.out.println("출력: " + printQueue.poll());
        System.out.println("출력: " + printQueue.poll());
        System.out.println("출력: " + printQueue.poll());
```

```
    }
}
```

문제2 - 작업 예약

- 서비스를 운영중인데, 낮 시간에는 사용자가 많아서 서버에서 무거운 작업을 하기 부담스럽다. 무거운 작업을 예약해두고 사용자가 없는 새벽에 실행하도록 개발해보자.
- 다양한 무거운 작업을 새벽에 실행한다고 가정하자.
- 작업은 자유롭게 구현하고 자유롭게 예약할 수 있어야 한다.
- 다음 예제 코드와 실행 결과를 참고해서 `TaskScheduler` 클래스를 완성하자.

문제 설명

```
package collection.deque.test.queue;

public interface Task {
    void execute();
}
```

```
package collection.deque.test.queue;

public class CompressionTask implements Task {
    @Override
    public void execute() {
        System.out.println("데이터 압축...");
    }
}
```

```
package collection.deque.test.queue;

public class BackupTask implements Task {
    @Override
    public void execute() {
        System.out.println("자료 백업...");
    }
```

```
}
```

```
package collection.deque.test.queue;

public class CleanTask implements Task {
    @Override
    public void execute() {
        System.out.println("사용하지 않는 자원 정리...");
    }
}
```

```
package collection.deque.test.queue;

public class SchedulerTest {
    public static void main(String[] args) {
        //낮에 작업을 저장
        TaskScheduler scheduler = new TaskScheduler();
        scheduler.addTask(new CompressionTask());
        scheduler.addTask(new BackupTask());
        scheduler.addTask(new CleanTask());

        //새벽 시간에 실행
        System.out.println("작업 시작");
        run(scheduler);
        System.out.println("작업 완료");
    }

    private static void run(TaskScheduler scheduler) {
        while (scheduler.getRemainingTasks() > 0) {
            scheduler.processNextTask();
        }
    }
}
```

실행 결과

```
작업 시작
```

```
데이터 압축...
자료 백업...
사용하지 않는 자원 정리...
작업 완료
```

TaskScheduler - 코드 작성

```
package collection.deque.test.queue;

import java.util.ArrayDeque;
import java.util.Queue;

public class TaskScheduler {
    private Queue<Task> tasks = new ArrayDeque<>();

    // 코드 작성
}
```

정답

```
package collection.deque.test.queue;

import java.util.ArrayDeque;
import java.util.Queue;

public class TaskScheduler {
    private Queue<Task> tasks = new ArrayDeque<>();

    public void addTask(Task task) {
        tasks.offer(task);
    }

    public void processNextTask() {
        Task task = tasks.poll();
        if (task != null) {
            task.execute();
        }
    }
```

```
    public int getRemainingTasks() {
        return tasks.size();
    }
}
```

정리