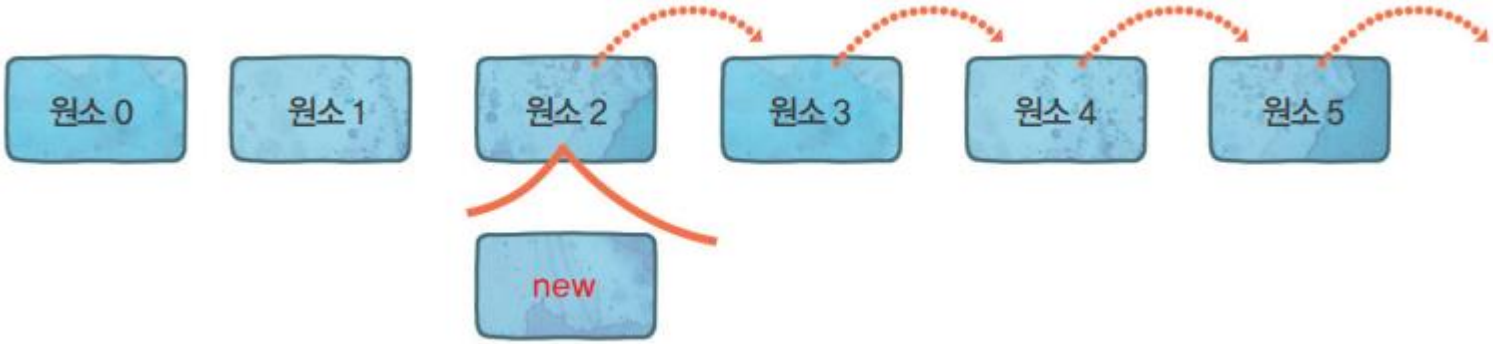
Chapter 11

컬렉션 프레임워크

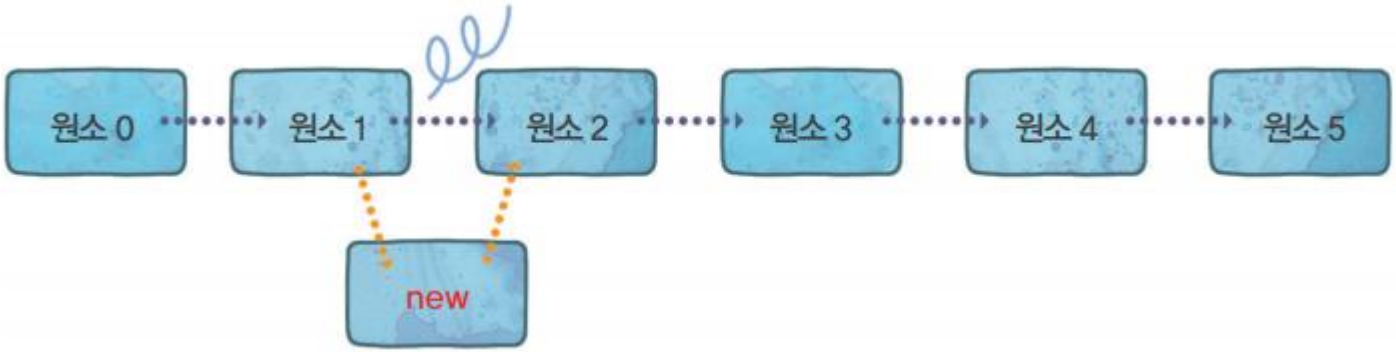
Contents

컬렉션 프레임워크의 필요성

- 유사한 객체를 여러 개 저장하고 조작해야 할 때가 빈번
- 고정된 크기의 배열의 불편함

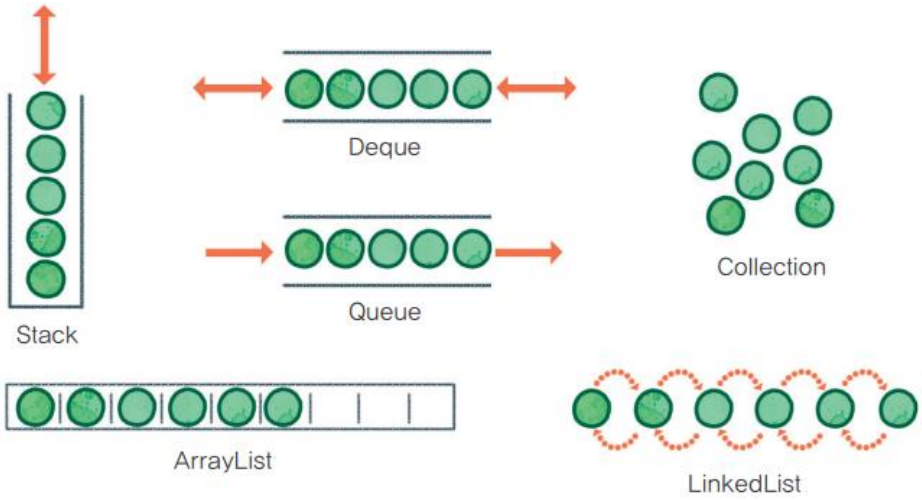


- 연결 리스트를 사용하면



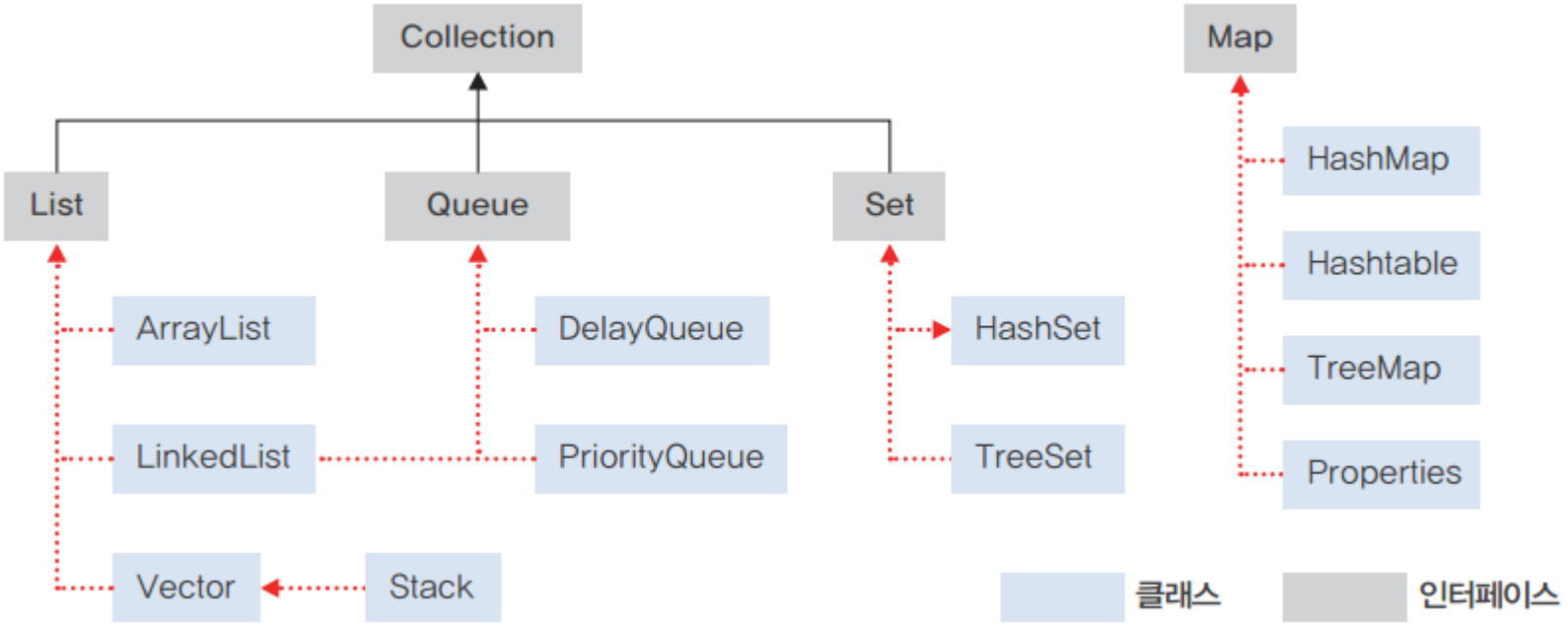
컬렉션 프레임워크의 의미

- 유사한 객체의 집단을 효율적으로 관리할 수 있도록 컬렉션 프레임워크를 제공
- 컬렉션
  - 데이터를 한곳에 모아 편리하게 저장 및 관리하는 가변 크기의 객체 컨테이너
- 컬렉션 프레임워크
  - 객체를 한곳에 모아 효율적으로 관리하고 편리하게 사용할 수 있도록 제공하는 환경



컬렉션 프레임워크의 구조

- 컬렉션 프레임워크는 인터페이스와 클래스로 구성
- 인터페이스는 컬렉션에서 수행할 수 있는 각종 연산을 제네릭 타입으로 정의해 유사한 클래스에 일관성 있게 접근하게 함
- 클래스는 컬렉션 프레임워크 인터페이스를 구현한 클래스



Collection 인터페이스(1)

- Collection 인터페이스와 구현 클래스

인터페이스		특징	구현 클래스
Collection	List	객체의 순서가 있고, 원소가 중복될 수 있다.	ArrayList, Stack, Vector, LinkedList
	Queue	객체를 입력한 순서대로 저장하며, 원소가 중복될 수 있다.	DelayQueue, PriorityQueue, LinkedList
	Set	객체의 순서가 없으며, 동일한 원소를 중복할 수 없다.	HashSet, TreeSet, EnumSet

Collection 인터페이스(2)

- Collection 인터페이스가 제공하는 주요 메서드

메서드	설명
boolean add(E e)	객체를 맨 끝에 추가한다.
void clear( )	저장된 모든 객체를 제거한다.
boolean contains(Object o)	명시한 객체의 저장 여부를 조사한다.
boolean isEmpty( )	리스트가 비어 있는지 조사한다.
Iterator<E> iterator( )	Iterator를 반환한다.
boolean remove(Object o)	명시한 첫 번째 객체를 제거하고, 제거 여부를 반환한다.
int size( )	저장된 전체 객체의 개수를 반환한다.
T[ ] toArray(T[ ] a)	리스트를 배열로 반환한다.

Map 인터페이스(1)

- 특징과 구현 클래스

특징	구현 클래스
키와 값을 쌍으로 저장하며, 키는 중복될 수 없다.	HashMap, Hashtable, TreeMap, Properties

- Map 인터페이스가 제공하는 주요 메서드

메서드	설명
void clear( )	모든 매핑을 삭제한다.
boolean containsKey(Object key)	주어진 키의 존재 여부를 반환한다.
boolean containsValue(Object value)	주어진 값의 존재 여부를 반환한다.
Set<Map.Entry<K, V>> entrySet( )	모든 매핑을 Set 타입으로 반환한다.
V get(Object key)	주어진 키에 해당하는 값을 반환한다.
boolean isEmpty( )	컬렉션이 비어 있는지 여부를 반환한다.
Set<K> keySet( )	모든 키를 Set 타입으로 반환한다.
V put(K key, V value)	주어진 키-값을 저장하고 값을 반환한다.
V remove(Object key)	키와 일치하는 원소를 삭제하고 값을 반환한다.
int size( )	컬렉션의 크기를 반환한다.
Collection<V> values( )	모든 값을 Collection 타입으로 반환한다.

Map 인터페이스(2)

- Map.Entry<K, V> 인터페이스가 제공하는 주요 메서드

메서드	설명
K getKey( )	원소에 해당하는 키를 반환한다.
V getValue( )	원소에 해당하는 값을 반환한다.
V setValue( )	원소의 값을 교체한다.

List 컬렉션(1)

- 순서가 있는 객체를 중복 여부와 상관없이 저장하는 리스트 자료구조 지원
- 배열과 매우 유사하지만 배열과 달리 크기가 가변적
- List 인터페이스가 제공하는 주요 메서드

메서드	설명
void add(int index, E element)	객체를 인덱스 위치에 추가한다.
E get(int index)	인덱스에 있는 객체를 반환한다.
int indexOf(Object o)	명시한 객체가 있는 첫 번째 인덱스를 반환한다.
E remove(int index)	인덱스에 있는 객체를 제거한다.
E set(int index, E element)	인덱스에 있는 객체와 주어진 객체를 교체한다.
List<E> subList(int from, int to)	주어진 범위에 해당하는 객체를 리스트로 반환한다.

List 컬렉션(2)

- 배열 타입 ⟺ List 타입

```
List 변수 = Arrays.asList(배열);
배열 = List객체.toArray(new 배열타입(List객체.size()));
```

List 객체를 배열로 변환하려면, 먼저 List 객체의 크기만큼 배열 객체를 생성해야 한다.

- [예제 11-1]

```java
import java.util.Arrays;
import java.util.List;

public class ListDemo {
    public static void main(String[] args) {
        String[] names1 = { "사슴", "호랑이", "바다표범", "곰" };

        List<String> list = Arrays.asList(names1);
        list.set(1, "앵무새");
        for (String s : list)
            System.out.print(s + "\t");
        System.out.println();

        list.sort((x, y) -> x.length() - y.length());
        String[] names2 = list.toArray(new String[list.size()]);
        for (int i = 0; i < names2.length; i++)
            System.out.print(names2[i] + "\t");
    }
}
```

```
사슴	앵무새	바다표범	곰
곰	사슴	앵무새	바다표범
```

List 컬렉션(3)

- Stack 클래스에 추가된 메서드

메서드	설명
boolean empty( )	스택이 비어 있는지 여부를 조사한다.
E peek( )	스택의 최상위 원소를 제거하지 않고 엿보기 한다.
E pop( )	스택의 최상위 원소를 반환하며, 스택에서 제거한다.
E push(E item)	스택의 최상위에 원소를 추가한다.
int search(Object o)	주어진 원소의 인덱스 값(1부터 시작)을 반환한다.

- [예제 11-2]

```
체리
체리
바나나
사과

10 100 20
20 100 10
```

```java
import java.util.Stack;

public class StackDemo {
    public static void main(String[] args) {
        Stack<String> s1 = new Stack<>();

        s1.push("사과");
        s1.push("바나나");
        s1.push("체리");

        System.out.println(s1.peek());

        System.out.println(s1.pop());
        System.out.println(s1.pop());
        System.out.println(s1.pop());
        System.out.println();

        Stack<Integer> s2 = new Stack<>();

        s2.add(10);
        s2.add(20);
        s2.add(1, 100);

        for (int value : s2)
            System.out.print(value + " ");
        System.out.println();

        while (!s2.empty())
            System.out.print(s2.pop() + " ");
    }
}
```

List 컬렉션(4)

- ArrayList와 LinkedList의 비교

구분	ArrayList 클래스	LinkedList 클래스
구현	가변 크기 배열	이중 연결 리스트
초기 용량	10	0
get( ) 연산	빠름	느림
add( ), remove( ) 연산	느림	빠름
메모리 부담	적음	많음
Iterator	순방향	순방향, 역방향

- [예제 11-3]

```java
import java.util.ArrayList;
import java.util.LinkedList;

public class PerformanceDemo {
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        LinkedList<Integer> ll = new LinkedList<Integer>();

        long start = System.nanoTime();
        for (int i = 0; i < 100000; i++)
            al.add(0, i);
        long end = System.nanoTime();
        long duration = end - start;
        System.out.println("ArrayList로 처리한 시간 : " + duration);

        start = System.nanoTime();
        for (int i = 0; i < 100000; i++)
            ll.addFirst(i);
        end = System.nanoTime();
        duration = end - start;
        System.out.println("LinkedList로 처리한 시간 : " + duration);
    }
}
```

```
ArrayList로 처리한 시간 : 984324273
LinkedList로 처리한 시간 : 5616035
```

Queue 컬렉션

- 선입선출(FIFO) 방식을 지원

- Queue 인터페이스에 추가된 메서드

기능	예외를 던짐	특별한 값을 반환
삽입	boolean add(E e)	boolean offer(E e)
삭제	E remove( )	E poll( )
검색	E element( )	E peek( )

- [예제 11-4]

```java
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Queue;

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

        // q.remove();

        System.out.println(q.poll());
        q.offer("사과");
        System.out.println("바나나를 추가했나요? " + q.offer("바나나"));

        try {
            q.add("체리");
        } catch (IllegalStateException e) {
        }
        System.out.println("헤드 엿보기 : " + q.peek());
```

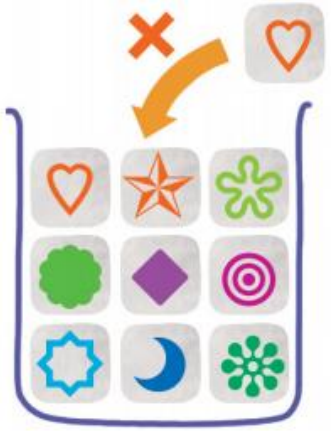
- [예제 11-4]-cont.

```
22
23          String head = null;
24          try {
25              head = q.remove();
26              System.out.println(head + " 제거하기");
27              System.out.println("새로운 헤드 : " + q.element());
28          } catch (NoSuchElementException e) {
29          }
30
31          head = q.poll();
32          System.out.println(head + " 제거하기");
33          System.out.println("새로운 헤드 : " + q.peek());
34
35          System.out.println("체리를 포함하고 있나요? " + q.contains("체리"));
36          System.out.println("사과를 포함하고 있나요? " + q.contains("사과"));
37      }
38  }
```

```
null
바나나를 추가했나요? true
헤드 엿보기 : 사과
사과 제거하기
새로운 헤드 : 바나나
바나나 제거하기
새로운 헤드 : 체리
체리를 포함하고 있나요? true
사과를 포함하고 있나요? false
```

Set 컬렉션

- 순서가 없으며, 중복되지 않는 객체를 저장하는 자료구조 지원
- 추가된 메서드는 없지만 중복 원소를 배제
- 인덱스가 없어 저장 순서 무시



- [예제 11-5]

```java
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class HashSet1Demo {
    public static void main(String[] args) {
        String[] fruits = { "사과", "바나나", "포도", "수박" };
        Set<String> h1 = new HashSet<>();
        Set<String> h2 = new HashSet<>();

        for (String s : fruits)
            h1.add(s);

        System.out.println("1단계 : " + h1);
        h1.add("바나나");
        h1.remove("포도");
        h1.add(null);

        System.out.println("2단계 : " + h1);
        System.out.println(h1.size());
        System.out.println(h1.contains("수박"));

        List<String> list = Arrays.asList(fruits);
        h2.addAll(list);
        System.out.println("3단계 : " + h2);
        h2.clear();
        System.out.println(h2.isEmpty());
    }
}
```

```
1단계 : [포도, 수박, 사과, 바나나]
2단계 : [null, 수박, 사과, 바나나]
4
true
3단계 : [포도, 수박, 사과, 바나나]
true
```

- [예제 11-6]

```java
import java.util.HashSet;
import java.util.Set;

class Element {
    String value;

    public Element(String value) {
        this.value = value;
    }

    public int hashCode() {
        return value.hashCode();
    }

    public boolean equals(Object o) {
        if (o instanceof Element) {
            return ((Element) o).value.equals(value);
        }
        return false;
    }

    public String toString() {
        return "Element[" + value + "]";
    }
}
```

```java
public class HashSet2Demo {
    public static void main(String[] args) {
        Set<Element> h = new HashSet<>();
        h.add(new Element("안녕"));
        h.add(new Element("안녕"));
        System.out.println(h.size());
        System.out.println(h);
    }
}
```

```
1
[Element[안녕]]
```

Map 컬렉션

- 사전처럼 키와 값 쌍으로 구성된 객체를 저장하는 자료구조 지원
- 키와 값도 모두 객체이며 키는 중복되지 않고 하나의 값에만 매핑

키	값
사과	
바나나	
포도	
체리	

- [예제 11-7]

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

public class HashMap1Demo {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<>();

        m.put("사과", 5);
        m.put("바나나", 3);
        m.put("포도", 10);
        m.put("딸기", 1);

        System.out.println(m.size() + "종류의 과일이 있습니다.");

        for (String key : m.keySet())
            System.out.println(key + "가 " + m.get(key) + "개 있습니다.");

        String key = "바나나";
        if (m.containsKey(key))
            System.out.println(key + "가 " + m.get(key) + "개 있습니다.");

        m.remove("사과");
        System.out.println("사과를 없앤 후 과일은 " + m.size() + "종류입니다.");

        m.clear();
        System.out.println("모두 없앤 후 과일은 " + m.size() + "종류입니다.");
    }
}
```

```
4종류의 과일이 있습니다.
포도가 10개 있습니다.
사과가 5개 있습니다.
바나나가 3개 있습니다.
딸기가 1개 있습니다.
바나나가 3개 있습니다.
사과를 없앤 후 과일은 3종류입니다.
모두 없앤 후 과일은 0종류입니다.
```

- [예제 11-8]

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

public class HashMap2Demo {
    public static void main(String[] args) {
        Map<Element, Integer> map = new HashMap<>();
        map.put(new Element("안녕"), 1);
        map.put(new Element("안녕"), 2);
        map.put(null, 3);
        System.out.println(map.size());
        System.out.println(map);
    }
}
```

```
2
{null=3, Element[안녕]=2}
```

Collections 클래스 의미

- 컬렉션을 다루는 다양한 메서드를 제공하는 java.util 패키지의 클래스
- 컬렉션 원소 정렬, 섞기, 탐색 등 문제를 쉽게 해결

- [예제 11-9] 정렬하기

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class SortDemo {
    public static void main(String[] args) {
        String[] s = { "나는", "조국", "대한민국을", "사랑합니다" };
        List<String> list = Arrays.asList(s);

        Collections.sort(list, Collections.reverseOrder());
        System.out.println(list);

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

```
[조국, 사랑합니다, 대한민국을, 나는]
[나는, 대한민국을, 사랑합니다, 조국]
```

- [예제 11-10] 섞기

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

public class ShuffleDemo {
    public static void main(String[] args) {
        List<Character> list = new ArrayList<>();

        for (char c = 'A'; c <= 'G'; c++)
            list.add(c);

        System.out.println("섞기 전 :\t" + list);
        Collections.shuffle(list);
        System.out.println("섞은 후 :\t" + list);
    }
}
```

```
섞기 전 :	[A, B, C, D, E, F, G]
섞은 후 :	[G, E, B, F, C, D, A]
```

- [예제 11-11] 탐색하기

```java
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class SearchDemo {
    public static void main(String[] args) {
        String[] s = { "황금을", "돌", "같이", "보라" };
        List<String> list = Arrays.asList(s);

        Collections.sort(list);
        System.out.println(list);
        int i = Collections.binarySearch(list, "돌");
        System.out.println(i);
    }
}
```

```
[같이, 돌, 보라, 황금을]
1
```

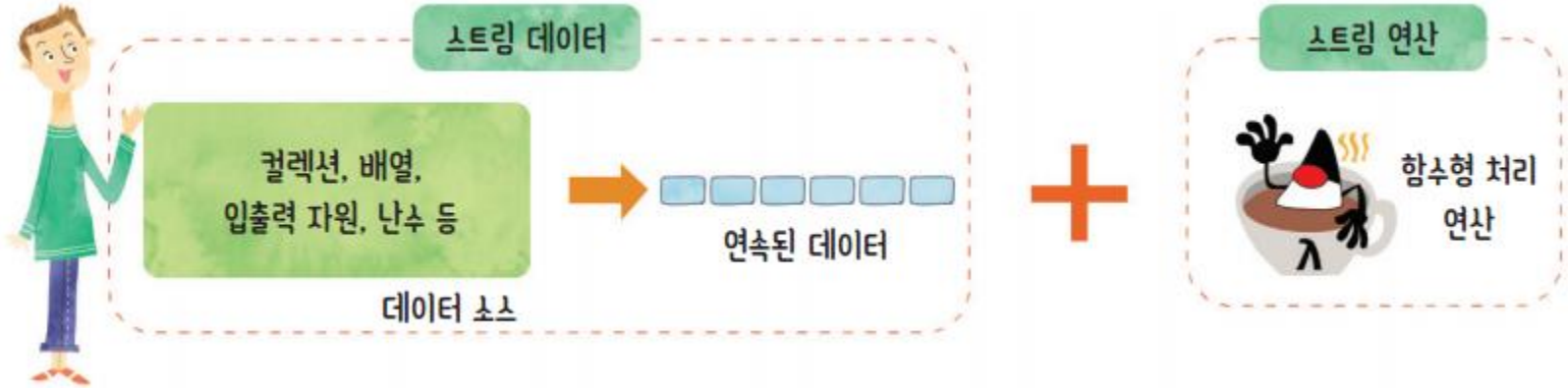
Collections 클래스 기타 메서드

- Collections 클래스

메서드	설명
addAll( )	명시된 원소들을 컬렉션에 삽입한다.
copy( )	리스트를 다른 리스트로 복사한다.
disjoint( )	2개의 컬렉션에서 공통된 원소가 있는지 조사한다.
fill( )	리스트의 모든 원소를 특정 값으로 덮어쓴다.
frequency( )	컬렉션에 주어진 원소의 빈도수를 반환한다.
max( )	리스트에서 최댓값을 반환한다.
min( )	리스트에서 최솟값을 반환한다.
nCopies( )	매개변수 값으로 주어진 객체를 주어진 횟수만큼 복사해 List 객체를 반환한다.
reverse( )	리스트의 원소들을 역순으로 정렬한다.
swap( )	리스트에서 주어진 위치에 있는 두 원소를 교체한다.

스트림의 개념

- JDK 8부터 새롭게 추가된 기능으로 데이터 집합체를 반복적으로 처리
- 스트림을 이용하면 다수의 스레드 코드를 구현하지 않아도 데이터를 병렬로 처리
- 스트림은 스트림 데이터와 스트림 연산의 개념을 모두 포함



스트림 데이터와 스트림 연산

- 스트림 데이터
  - 데이터 소스에서 추출한 연속적인 데이터
  - 스트림 데이터 소스로는 컬렉션, 배열, 입출력 채널, 난수 등이 있으며, 심지어 피보나치 수열 등 연속된 데이터도 가능

- 스트림 연산
  - 매개변수 값이 람다식인 함수형 연산
  - 스트림 데이터를 필터링, 정렬, 매핑, 집계 등의 작업 수행

스트림과 컬렉션 비교

how → 컬렉션, what → 스트림

구분	컬렉션	스트림
처리 방식	다운로드	스트리밍
저장 공간	필요	불필요
반복 방식	외부 반복	내부 반복
코드 구현	명령형	선언형
원본 데이터	변경	변경하지 않고 소비
연산 병렬화	어려움	쉬움

(a) 컬렉션 (b) 스트림

컬렉션의 데이터 반복 처리(1)

- 컬렉션의 종류에 관계 없이 반복자를 이용하면 컬렉션에 포함된 객체를 순차적으로 순회
- Iterator 인터페이스가 제공하는 주요 메서드

메서드	설명
boolean hasNext( )	다음 원소의 존재 여부를 반환한다.
E next( )	다음 원소를 반환한다.
default void remove( )	마지막에 순회한 컬렉션의 원소를 삭제한다.

- 반복자의 이동

컬렉션의 데이터 반복 처리(2)

- 예 : Collection<String> 객체 collection이 있다면

```
Iterator<String> iterator = collection.iterator();
while (iterator.hasNext()) {
  String s = iterator.next();
  // 원소를 처리한다.
}
```

```
for (String s : collection) {
  // 원소를 처리한다.
}
```

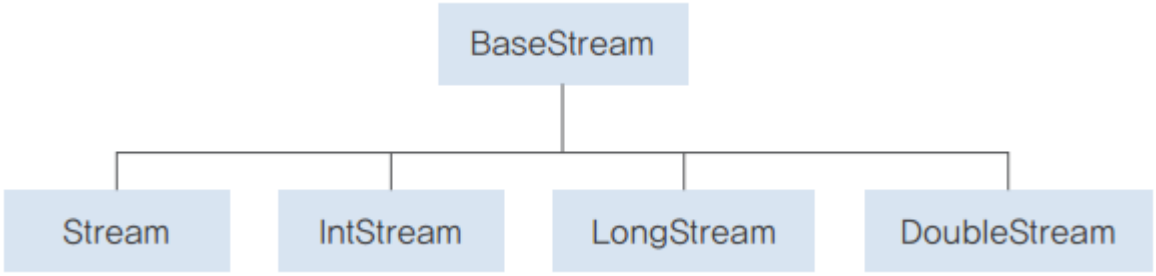
for~each 문은 간접적으로 next( ) 메서드를 호출한다.

- [예제 11-12]

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

public class IteratorDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("다람쥐");
        list.add("개구리");
        list.add("나비");

        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext())
            System.out.print(iterator.next() + "\t");
        System.out.println();

        Collections.sort(list);

        while (iterator.hasNext())
            System.out.print(iterator.next() + ",\t");
        System.out.println();

        iterator = list.iterator();
        while (iterator.hasNext())
            System.out.print(iterator.next() + "->\t");
    }
}
```

```
다람쥐      개구리      나비

개구리->    나비->      다람쥐->
```

스트림의 종류



- Stream은 객체 원소로 구성된 스트림 데이터를 처리

- 컬렉션이나 배열에서 스트림 구현 객체를 얻는 주요 메서드

메서드	설명
컬렉션	Stream<T> Collection.stream( )
배열	Stream<T> Arrays.stream(T[ ])
	IntStream Arrays.stream(int[ ])
	DoubleStream Arrays.stream(double[ ])

스트림의 메서드(1)

- 스트림 인터페이스가 제공하는 유용한 메서드

메서드	설명
long count( )	스트림에 포함된 원소 개수를 반환한다.
Stream<T> filter(predicate)	predicate에 맞는 원소만으로 구성된 스트림을 반환한다.
void forEach(action)	스트림의 각 원소에서 action을 수행한다.
Stream<T> limit(long max)	처음 max개의 원소로 구성된 스트림을 반환한다.
Stream<T> skip(long n)	처음 n개의 원소를 제외한 원소로 구성된 스트림을 반환한다.
Stream<T> sorted( )	스트림 원소를 정렬한다.

스트림의 메서드(2)

- predicate
  - 매개변수가 있고, 논리 값을 반환하는 메서드를 가진 함수형 인터페이스 타입

```
boolean 메서드이름(1개_이상의_매개변수)
```

  - 예 : 문자열 s의 크기가 0보다 큰지를 조사

```
s -> s.length() > 0
```
→ *predicate*

- action
  - 매개변수는 있지만 반환 값이 없는 메서드를 가진 함수형 인터페이스 타입

```
void 메서드이름(1개_이상의_매개변수)
```

  - 예 : 문자열 s를 콘솔 뷰에 출력

```
s -> System.out.println(s)
```
→ *action*

스트림을 이용한 데이터 처리(1)

```
List<Integer> list = new ArrayList<>();
List<Integer> gt10 = new ArrayList<>();
Random r = new Random();
```

10보다 큰 원소를 임시로 보관할 저장 공간이다.

```
for (int i = 0; i < 10; i++)
   list.add(r.nextInt(30));

for (int i : list)
   if (i > 10)
      gt10.add(i);

Collections.sort(gt10);
System.out.println(gt10);
```

컬렉션을 이용

```
list.stream().filter(i -> i > 10).sorted()
            .forEach(x -> System.out.print(x + " "));
```

스트림을 이용

- [예제 11-13]

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

public class Stream1Demo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        List<Integer> gt10 = new ArrayList<>();
        Random r = new Random();

        for (int i = 0; i < 10; i++)
            list.add(r.nextInt(30));

        for (int i : list)
            if (i > 10)
                gt10.add(i);

        Collections.sort(gt10);
        System.out.println(gt10);

        list.stream().filter(i -> i > 10).sorted().forEach(x -> System.out.print(x + " "));
    }
}
```

```
[11, 15, 17, 28]
11 15 17 28
```

스트림을 이용한 데이터 처리(2)

- 예

```
list.stream()                                   // 컬렉션에서 스트림을 가져온다.
.filter(i -> i > 10)                            // 10보다 큰 원소만 추출한다.
.sorted()                                       // 정렬한다.
.forEach(x -> System.out.print(x + ""));        // 원소를 하나씩 출력한다.
```

- [예제 11-14]

```
3  import java.util.Arrays;
4  import java.util.stream.IntStream;
5
6  public class Stream2Demo {
7      public static void main(String[] args) {
8          int[] ia = {1, 6, 3, 9, 5, 4, 2};
9          IntStream is = Arrays.stream(ia);
10
11         int sum = is.filter(i -> i > 5).sum();
12
13         System.out.println(sum);
14     }
15 }
```

```
15
```