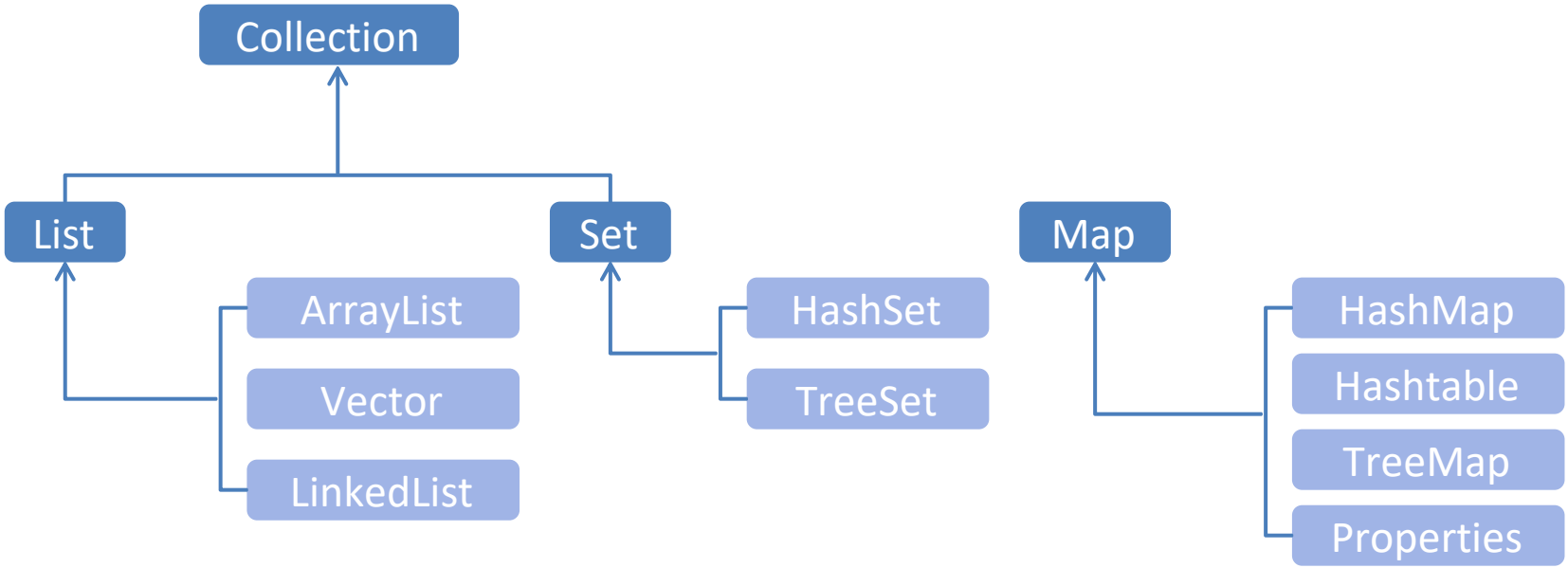
컬렉션 프레임워크

01. 컬렉션 프레임워크

컬렉션 프레임워크

- 널리 알려진 자료구조를 사용해서 객체를 효율적으로 추가,삭제,검색할 수 있도록 미리 구현된 인터페이스와 클래스
- java.util 패키지에서 제공
- 컬렉션 : 객체의 저장 , 프레임워크 :사용 방법을 정해놓은 라이브러리

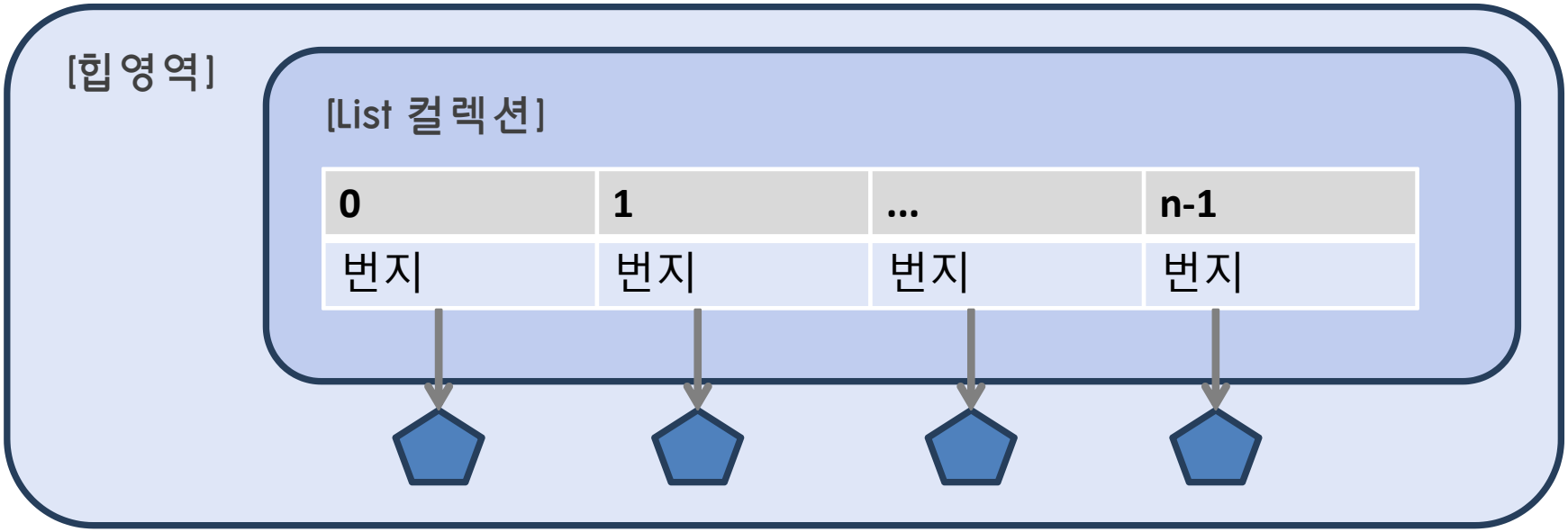


List 컬렉션

01. List 컬렉션

List 컬렉션

- 자료를 일렬로 나열하여 저장하는 방식
- 배열과 유사하나 데이터 저장시 필요한 만큼의 공간이 자동증가
- 객체자체를 저장하는 것이 아니라 객체가 존재하는 주소번지를 저장



02. List 공통메서드

공통 메서드

- List 컬렉션에는 ArrayList,Vector,LinkedList 등이 있는데 해당 클래스에서 공통적으로 사용하는 메서드는 다음과 같다

기능	메소드	설명
객체추가	boolean add(E e)	주어진 객체를 맨 끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	E set(int index,E element)	주어진 인덱스에 저장된 객체를 주어진 객체로 변경
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지를 조사
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

03. ArrayList

ArrayList

- 생성방법

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

타입 파라미터

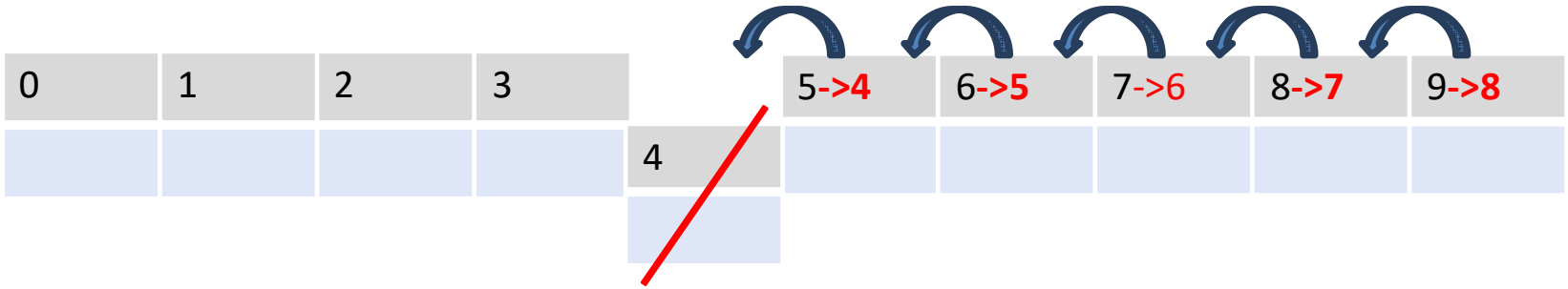
타입 파라미터

- 초기 용량

  - 객체 10 개를 저장할 수 있는 용량

0	1	2	3	4	5	6	7	8	9

- 삭제시 뒤인덱스부터 마지막 인덱스까지 모두 앞으로 1 씩 당겨짐



04. Vector

Vector

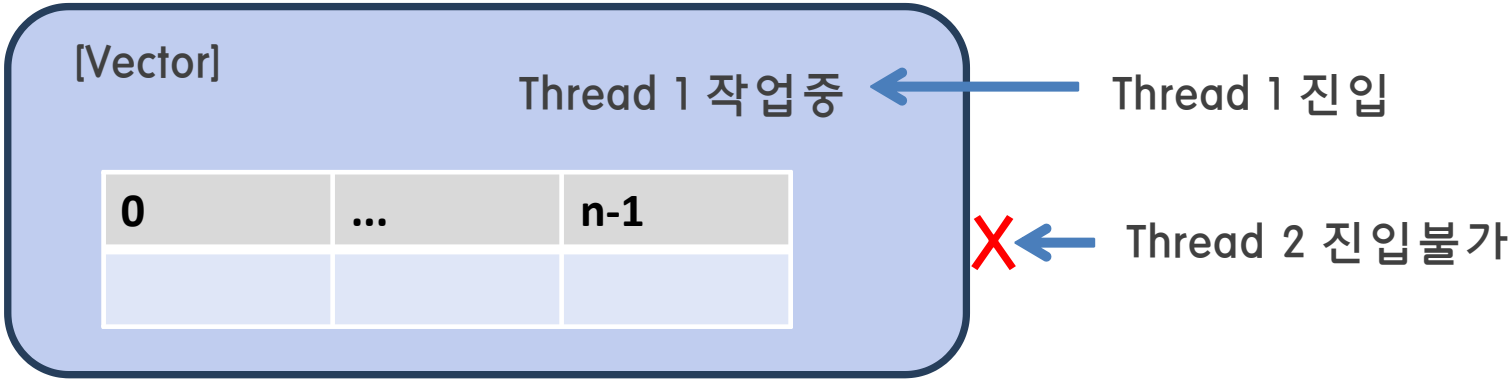
- 생성방법

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

타입 파라미터

타입 파라미터

- ArrayList 와 다른 점은 멀티 스레드환경에서 안전하게 객체를 추가/삭제할 수 있다



05. LinkedList

LinkedList

- 사용방법

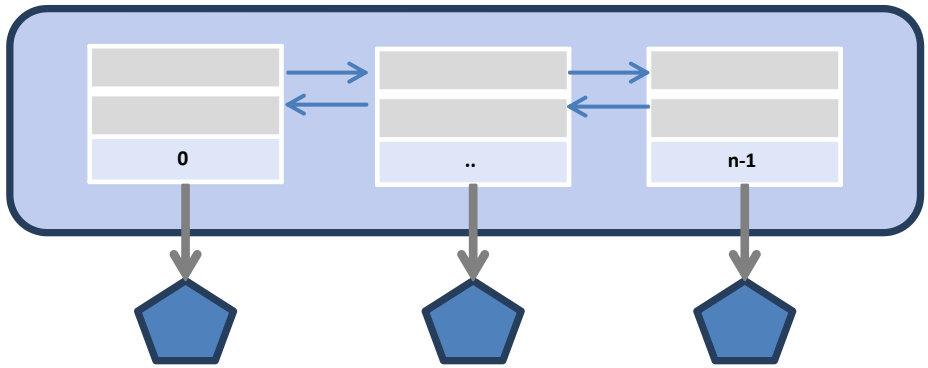
```
List <E> list = new LinkedList <E> ();
```

타입 파라미터

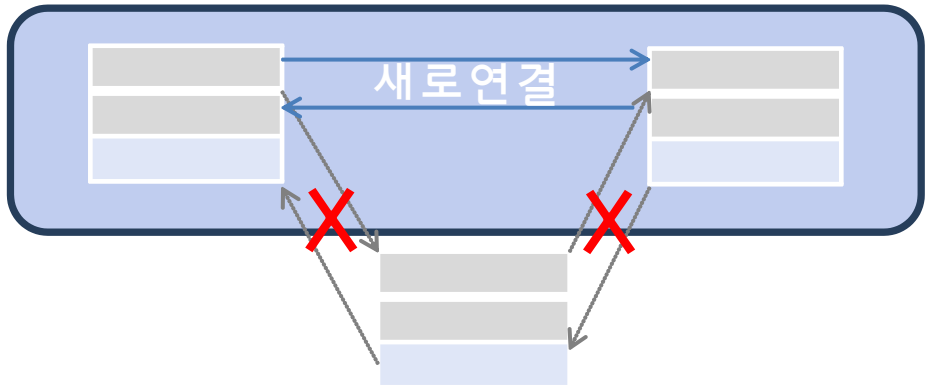
타입 파라미터

- 내부 배열 형태가 아닌 인접 참조를 링크해서 체인처럼 관리
  [힙영역]
- 기존의 ArrayList 에 비해서 데이터 추가 삭제시 성능향상

[LinkedList]



[삭제시]



Set 컬렉션

01. Set 컬렉션

Set 컬렉션

- 저장 순서가 유지되지 않는 형태의 저장
- 수학의 집합과 유사
- 순서와 상관없고 중복이 허용되지 않는다
- Hashset , LinkedHashSet , TreeSet 등이 있다

02. Set 공통메서드

공통 메서드

기능	메소드	설명
객체추가	boolean add(E e)	주어진 객체를 저장 객체가 성공적으로 저장되면 true 리턴 중복 객체면 false 를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 조사
	boolean isEmpty()	컬렉션인 비어있는지 조사
	Iterator<E> iterator()	저장된 객체를 한번씩 가져오는 반복자를 리턴
	int size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

03. 기본 사용법

- 저장 삭제

```
Set <String> set =...;
set.add(" 홍길동" );
set.add(" 남길동" );
set.remove(:홍길동" );
```

- 반복자 가져오기

```
set <String>set = ...;
Iterator<String> iterator = set.iterator();
```

03. 기본 사용법

- 반복자를 통한 자료 검색

리턴 타입	메소드	설명
boolean	hasNext()	가져올 객체가 있으면 true 가져올 객체가 없으면 false
E	next()	컬렉션에서 하나의 객체를 가져옴
void	remove()	Set컬렉션에서 객체를 제거

- 반복자를 통한 자료 검색

```
set <String>set = ...;
Iterator<String> iterator = set.iterator();
while(iterator.hasNext())
{
    String str = iterator.next();
}
```

04.HashSet

HashSet

- 순서 없이 저장, 동일 객체는 중복 저장하지 않는다
- hashCode() 메소드를 호출해서 얻어낸 해시코드를 통해 객체의 동등비교후 중복되지 않은 데이터 Set
- 사용 방법

```
Set <E> set = new HashSet<E>();
```

- Ex

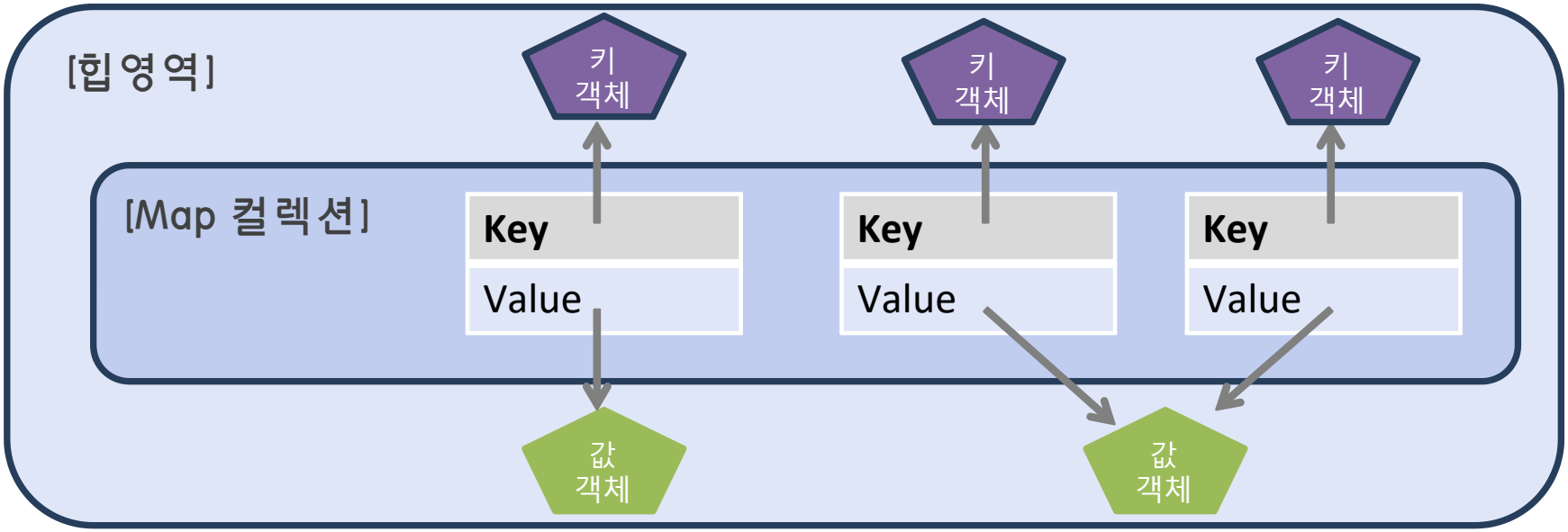
```
Set <String> set = new HashSet<String>();
Set <String>set = new HashSet<>();
```

Map 컬렉션

01. Map 컬렉션

Map 컬렉션

- 키(key)와 값(Value)으로 구성
- 키는 중복 저장될 수 없지만 값은 중복저장 가능
- 기존의 저장된 키와 동일한 키로 값을 저장하면 기존의 값은 없어지고 새로운 값으로 대체



02. Map 공통 메서드

공통 메서드

- Map 컬렉션에는 HashMap , Hashtable, LinkedHashMap, Properties,TreeMap 등이 있다

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키로 값을 저장 새로운 키일 경우 null을 리턴 동일한 키가 있을 경우 값을 대체하고 이전 값을 리턴
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부를 확인
	boolean containsValue(Object value)	주어진 값이 있는지 여부를 확인
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 set에 담아서 리턴
	V get(Object key)	주어진 키가 있는 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부 확인
	Set<K> keySet()	모든 키를 Set객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<> values()	저장된 모든 값을 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)을 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry를 삭제하고 값을 리턴

03. 기본 사용법

- 생성

```
Map <String,Integer>map=...;
map.put("홍길동",30);                    // 객체 추가
int score = map.get("홍길동");           // 객체 찾기
map.remove("홍길동");                    // 객체 삭제
```

- 검색01 (keySet() 메소드를 이용)

```
Map <K,V> = ...;
Set<K> keySet = map.keySet();          // 모든 키를 Set 컬렉션으로 전달
Iterator<K> keyIterator = keySet.iterator();       //반복자 생성
while(keyIterator.hasNext()){
    K key = keyIterator.next();
    V value = map.get(key);
}
```

02. 기본 사용법

- 검색01 (entrySet() 메소드를 이용)

```
Set<Map.Entry<K,V>> entrySet = map.entrySet(); //모든 Map.Entry를 Set컬렉션으로 전달
Iterator<Map.Entry<K,V>> entryIterator = entrySet.iterator();
while(entryIterator.hasNext()){
    Map.Entry<K,V> entry = entryIterator.next();
    K key = entry.getKey();
    V value = entry.getValue();
}
```

03. HashMap

HashMap

- Map 인터페이스를 구현한 대표적인 Map 컬렉션
- HashMap의 키로 사용할 객체는 hashCode()와 equals() 메소드를 재정의해서 동등객체가 될 조건을 정해야 함
- 사용 방법

```
Map <K   ,   V> map = new HashMap<K   ,   V>( );
```

키 타입 값 타입 키 타입 값 타입

- Ex

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

04. Hashtable

Hashtable

- HashMap 과 동일한 내부구조를 가지고 있음
  그러므로 hashCode() 와 equals() 메소드를 재정의해서 동등객체 조건을 정해야함

- HashMap 과 차이점으로 Hashtable은 동기화된 메소드 구성으로 인해
  스레드환경으로부터 안전

- 사용 방법

```
Map <K   ,   V> map = new Hashtable<K   ,   V>( );
```

키 타입 값 타입 키 타입 값 타입

- Ex

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