컬렉션 프레임워크

컬렉션(collection) : 여러 객체(데이터)를 모아 놓은 것 (객체의 저장)

프레임워크(framework) : 표준화, 정형화된 체계적인 프로그래밍 방식 (사용 방법을 정해 놓은 라이브러리)

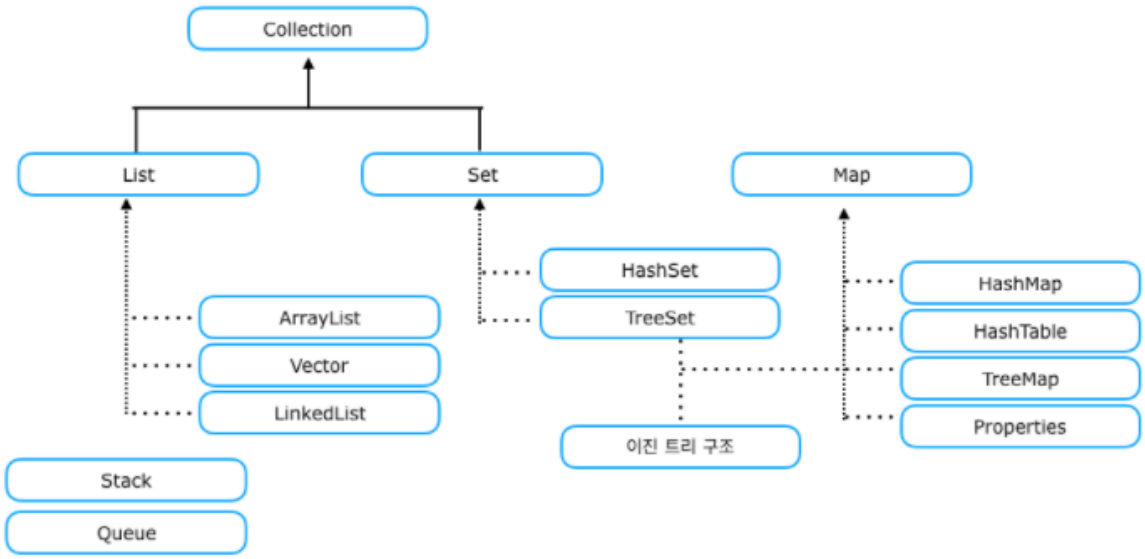
컬렉션 프레임워크

- 다수의 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공하는 클래스의 집합
- 자바는 널리 알려져있는 자료구조를 사용해 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 인터페이스와 구현 클래스를 java.util 패키지에서 제공합니다.

왜 생겨났나?

- 배열은 크기가 고정되어 있어서 (배열 생성시 크기 결정) 불편함
- 데이터 삭제시 그 인덱스 부분을 비워 두기에 메모리상 낭비 심함, for문으로 가져올때도 문제가 생긴다.

핵심 인터페이스



List

- 배열과 비슷하게 객체를 인덱스로 관리 하지만 다른점은 용량이다. 용량이 자동으로 증가한다. (객체 저장시 인덱스 자동 부여)
- 객체 자체 저장이 아니라 객체의 번지를 참조한다. (동일 객체 중복 저장 가능)
- null도 저장 가능
- 순서가 있으며 중복을 허용
- 구현 클래스 : ArrayList / LinkedList / Stack / Vector

Set(집합)

- 순서가 없으며 중복 허용x
- 구현클래스 : HashSet / TreeSet / properties

Map

- key와 value의 쌍으로 이루어진 데이터의 집합
- 순서 유지 x , key는 중복 허용x, value는 중복 허용
- 구현 클래스 : HashMap / TreeMap / HashTable / properties

==========

인터페이스의 메서드 (추가 삭제 검색)

ch11-4 List인터페이스 – 순서O, 중복O

메서드	설 명
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치(index)에 객체(element) 또는 컬렉션에 포함된 객체들을 추가한다.
Object get(int index)	지정된 위치(index)에 있는 객체를 반환한다.
int indexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 첫 번째 요소부터 순방향으로 찾는다.)
int lastIndexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 마지막 요소부터 역방향으로 찾는다.)
ListIterator listIterator( ) ListIterator listIterator(int index)	List의 객체에 접근할 수 있는 ListIterator를 반환한다.
Object remove(int index)	지정된 위치(index)에 있는 객체를 삭제하고 삭제된 객체를 반환한다.
Object set(int index, Object element)	지정된 위치(index)에 객체(element)를 저장한다
void sort(Comparator c)	지정된 비교자(comparator)로 List를 정렬한다.
List subList(int fromIndex, int toIndex)	지정된 범위(fromIndex부터 toIndex)에 있는 객체를 반환한다.

* Set인터페이스의 메서드 - Collection인터페이스와 동일

메서드	설 명
boolean add(Object o)	지정된 객체(o)를 Collection에 추가한다.
void clear( )	Collection의 모든 객체를 삭제한다.
boolean contains(Object o)	지정된 객체(o)가 Collection에 포함되어 있는지 확인한다.
boolean equals(Object o)	동일한 Collection인지 비교한다.
int hashCode( )	Collection의 hash code를 반환한다.
boolean isEmpty( )	Collection이 비어있는지 확인한다.
Iterator iterator( )	Collection의 Iterator를 얻어서 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
int size( )	Collection에 저장된 객체의 개수를 반환한다.
Object[ ] toArray( )	Collection에 저장된 객체를 객체배열(Object[ ])로 반환한다.
Object[ ] toArray(Object[ ] a)	지정된 배열에 Collection의 객체를 저장해서 반환한다.

* 집합과 관련된 메서드(Collection에 변화가 있으면 true, 아니면 false를 반환.

메서드	설 명
boolean addAll(Collection c)	지정된 Collection(c)의 객체들을 Collection에 추가한다.(합집합)
boolean containsAll(Collection c)	지정된 Collection의 객체들이 Collection에 포함되어 있는지 확인한다.(부분집합)
boolean removeAll(Collection c)	지정된 Collection에 포함된 객체들을 삭제한다.(차집합)
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만을 남기고 나머지는 Collection에서 삭제한다.(교집합)

ch11-6 Map인터페이스

메서드	설 명
void clear( )	Map의 모든 객체를 삭제한다.
boolean containsKey(Object key)	지정된 key객체와 일치하는 Map의 key객체가 있는지 확인한다.
boolean containsValue(Object value)	지정된 value객체와 일치하는 Map의 value객체가 있는지 확인한다.
Set entrySet( )	Map에 저장되어 있는 key-value쌍을 Map.Entry타입의 객체로 저장한 Set으로 반환한다.
boolean equals(Object o)	동일한 Map인지 비교한다.
Object get(Object key)	지정한 key객체에 대응하는 value객체를 찾아서 반환한다.
int hashCode( )	해시코드를 반환한다.
boolean isEmpty( )	Map이 비어있는지 확인한다.
Set keySet( )	Map에 저장된 모든 key객체를 반환한다.
Object put(Object key, Object value)	Map에 value객체를 key객체에 연결(mapping)하여 저장한다.
void putAll(Map t)	지정된 Map의 모든 key-value쌍을 추가한다.
Object remove(Object key)	지정한 key객체와 일치하는 key-value객체를 삭제한다.
int size( )	Map에 저장된 key-value쌍의 개수를 반환한다.
Collection values( )	Map에 저장된 모든 value객체를 반환한다.

==========

List

구분	순차적으로 추가/삭제	중간에 추가/삭제	검색
ArrayList	빠르다	느리다	빠르다
LinkedList	느리다	빠르다	느리다

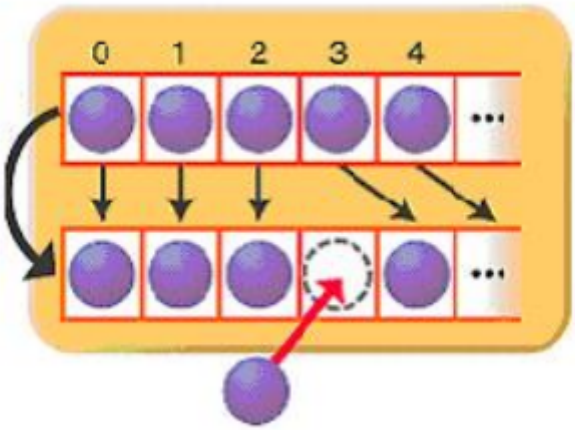
1. ArrayList

   https://coding-factory.tistory.com/551

   - Vector의 new 버전이다. (Vector는 동기화가 되어 있으나 ArrayList는 동기화 되어 있지 않다.)

     --> Vector는 동기화된 메소드로 구성되어 있기 때문에 멀티 스레드가 동시에 이 메소드들을 실행할 수 없고, 하나의 스레드가 실행을 완료해야만 다른 스레드들이 실행할 수 있습니다. 그래서 멀티 스레드 환경에서 안전하게 객체를 추가하고 삭제할 수 있습니다.
   - 데이터의 저장공간으로 배열을 사용한다.
   - List 인터페이스를 상속 받은 클래스
   - 용량 초과시 자동으로 용량 늘어남
   - 저장순서가 유지되며 중복 허용
   - 중간에 데이터를 insert 할 경우가 많다면 되도록이면 사용하지 말자

     --> index중간에 값을 추가하면 해당 인덱스부터 마지막 인덱스까지 모두 1씩 뒤로 밀려난다. 이럴 경우 성능에 악영향을 끼친다.



선언 방법

```
ArrayList list = new ArrayList();//타입 미설정 Object로 선언된다.
ArrayList<Student> members = new ArrayList<Student>();//타입설정 Student객체만 사용가능
ArrayList<Integer> num = new ArrayList<Integer>();//타입설정 int타입만 사용가능
ArrayList<Integer> num2 = new ArrayList<>();//new에서 타입 파라미터 생략가능
ArrayList<Integer> num3 = new ArrayList<Integer>(10);//초기 용량(capacity)지정
ArrayList<Integer> list2 = new ArrayList<Integer>(Arrays.asList(1,2,3));//생성시 값추가
```

ArrayList list = new ArrayList(); 라고 사용할 경우 값을 뽑아낼때 Casting 연산이 필요하다. 그러므로 우리는 Generics를 사용해 ArrayList를 선언해주자

> Generics는 선언할 수 있는 타입이 객체 타입입니다. int는 기본자료형이기 때문에 들어갈수 없으므로 int를 객체화시킨 wrapper클래스를 사용해야 합니다.

- 래퍼클래스(wrapper class)

기본타입(primitive type)	래퍼클래스(wrapper class)
byte	Byte
char	Character
int	Integer
float	Float
double	Double
boolean	Boolean
long	Long
short	Short

```
Integer num = new Integer(17); // 박싱
int n = num.intValue(); //언박싱
```

ArrayList의 예제

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

```java
// ArrayList 값 추가
list.add(3); //값 추가
list.add(null); //null값도 add가능
list.add(1,10); //index 1에 10 삽입

// ArrayList 값 삭제
list.remove(1);  //index 1 제거
list.clear();  //모든 값 제거

// ArrayList 크기 구하기
System.out.println(list.size());

// ArrayList 값 출력
ArrayList<Integer> list = new ArrayList<Integer>(Arrays.asList(1,2,3));
System.out.println(list.get(0));//0번째 index 출력
for(Integer i : list) { //for문을 통한 전체출력
    System.out.println(i);
}

// ArrayList 값 검색
System.out.println(list.contains(1)); //list에 1이 있는지 검색 : true
System.out.println(list.indexOf(1)); //1이 있는 index반환 없으면 -1


// iterator() 메소드와 get() 메소드를 이용한 요소의 출력
Iterator<Integer> iter = arrList.iterator();
while (iter.hasNext()) {
    System.out.print(iter.next() + " ");
}

// set() 메소드를 이용한 요소의 변경
arrList.set(0, 20);

for (int e : arrList) {
    System.out.print(e + " ");
}

// size() 메소드를 이용한 요소의 총 개수
System.out.println("리스트의 크기 : " + arrList.size());
```

2. LinkedList

- 인덱스 대신 다음 요소를 가리키는 참조만을 가진다
- 중간에 데이터를 추가나 삭제하더라도 전체의 인덱스가 한 칸씩 뒤로 밀리거나 당겨지는 일이 없어 추가/삭제가 용이하다.
- 인덱스가 없어서 특정 요소에 접근하려면 순차 탐색이 필요하다. (속도 저하)
- 탐색/정렬을 자주할 경우 되도록이면 쓰지 말자

이런 단일 연결 리스트는 이전 요소로 접근하기 어려워서, 이전 요소를 가르키는 참조도 가지는 이중 연결 리스트를 좀 더 많이 사용한다.

LinkedList 선언

```
// ArrayList와 달리 초기 크기 생성 x
LinkedList list = new LinkedList();//타입 미설정 Object로 선언된다.
LinkedList<Student> members = new LinkedList<Student>();//타입설정 Student객체만 사용가능
LinkedList<Integer> num = new LinkedList<Integer>();//타입설정 int타입만 사용가능
LinkedList<Integer> num2 = new LinkedList<>();//new에서 타입 파라미터 생략가능
LinkedList<Integer> list2 = new LinkedList<Integer>(Arrays.asList(1,2));//생성시 값추가
```

```
LinkedList<Integer> linklList = new LinkedList<Integer>();

// LinkedList 값 추가
linklList.addFirst(1);//가장 앞에 데이터 추가
linklList.addLast(2);//가장 뒤에 데이터 추가
linklList.add(3);//데이터 추가
linklList.add(1, 10);//index 1에 데이터 10 추가

// LinkedList 값 삭제
LinkedList<Integer> list = new LinkedList<Integer>(Arrays.asList(1,2,3,4,5));
list.removeFirst(); //가장 앞의 데이터 제거
list.removeLast(); //가장 뒤의 데이터 제거
list.remove(); //생략시 0번째 index제거
list.remove(1); //index 1 제거
list.clear(); //모든 값 제거

System.out.println(list.size()); //list 크기
```

```
// LinkedList 값 출력
LinkedList<Integer> list = new LinkedList<Integer>(Arrays.asList(1,2,3));
System.out.println(list.get(0));//0번째 index 출력

for(Integer i : list) { //for문을 통한 전체출력
    System.out.println(i);
}

Iterator<Integer> iter = list.iterator(); //Iterator 선언
while(iter.hasNext()){//다음값이 있는지 체크
    System.out.println(iter.next()); //값 출력
}
```

==========

Set

1. HashSet
    - Set 컬렉션 클래스에서 가장 많이 사용되는 클래스
    - 해시 알고리즘(hash algorithm)을 사용하여 검색 속도가 빠르다
    - 객체를 중복해서 저장할 수 없고 하나의 null 값만 저장할 수 있습니다. 또한 저장 순서가 유지되지 않습니다
    - 중복 자동 제거 (hashCode() 메소드 호출 -> 해시코드 get -> equals() 메소드로 객체 비교 -> true가 나오면 자동 제거)

```
// HashSet 선언
HashSet<Integer> set1 = new HashSet<Integer>();//HashSet생성
HashSet<Integer> set2 = new HashSet<>();//new에서 타입 파라미터 생략가능
HashSet<Integer> set3 = new HashSet<Integer>(set1);//set1의 모든 값을 가진 HashSet
생성
HashSet<Integer> set4 = new HashSet<Integer>(10);//초기 용량(capacity)지정
HashSet<Integer> set5 = new HashSet<Integer>(10, 0.7f);//초기 capacity,load
factor지정
HashSet<Integer> set6 = new HashSet<Integer>(Arrays.asList(1,2,3));//초기값 지정
```

저장 용량 초과시 list 처럼 저장 공간을 늘리는데 얘는 두배를 늘려버림. (과부하 발생 조심)

```
HashSet<String> hs01 = new HashSet<String>();
HashSet<String> hs02 = new HashSet<String>();

// add() 메소드를 이용한 요소의 저장
hs01.add("홍길동");
hs01.add("이순신");
System.out.println(hs01.add("임꺽정"));
System.out.println(hs01.add("임꺽정")); // 중복된 요소의 저장

// Enhanced for 문과 get() 메소드를 이용한 요소의 출력
for (String e : hs01) {
    System.out.print(e + " ");
}

// add() 메소드를 이용한 요소의 저장
hs02.add("임꺽정");
hs02.add("홍길동");
hs02.add("이순신");
```

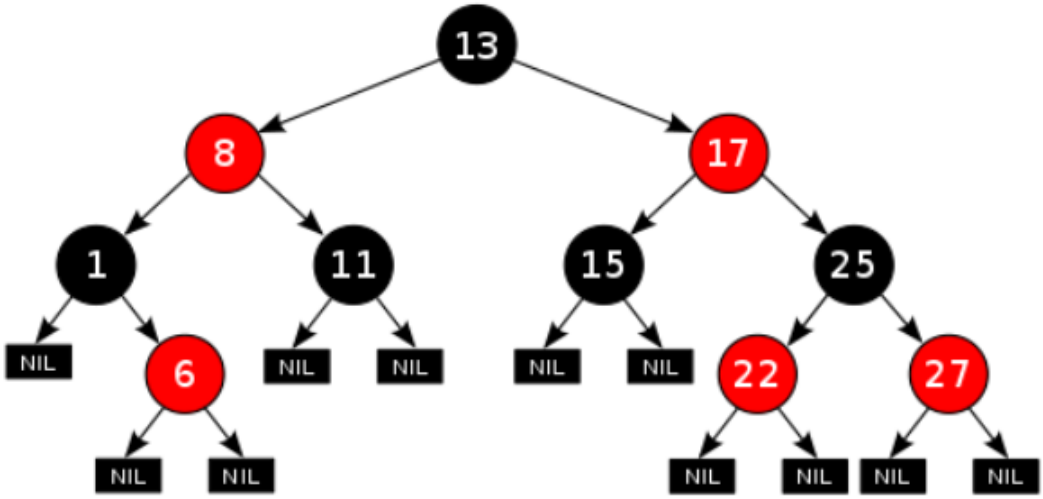
```java
// iterator() 메소드를 이용한 요소의 출력
Iterator<String> iter02 = hs02.iterator();
while (iter02.hasNext()) {
    System.out.print(iter02.next() + " ");
}

// size() 메소드를 이용한 요소의 총 개수
System.out.println("집합의 크기 : " + hs02.size());
```

```
실행 결과 :
// 저장 순서는 상관이 없다, 중복된 값 '임꺽정' 삭제되었다.
true
false
홍길동 이순신 임꺽정
홍길동 이순신 임꺽정
집합의 크기 : 3
```

2. TreeSet

- 데이터가 정렬된 상태로 저장되는 이진 검색 트리(binary search tree)의 형태로 요소를 저장합니다.



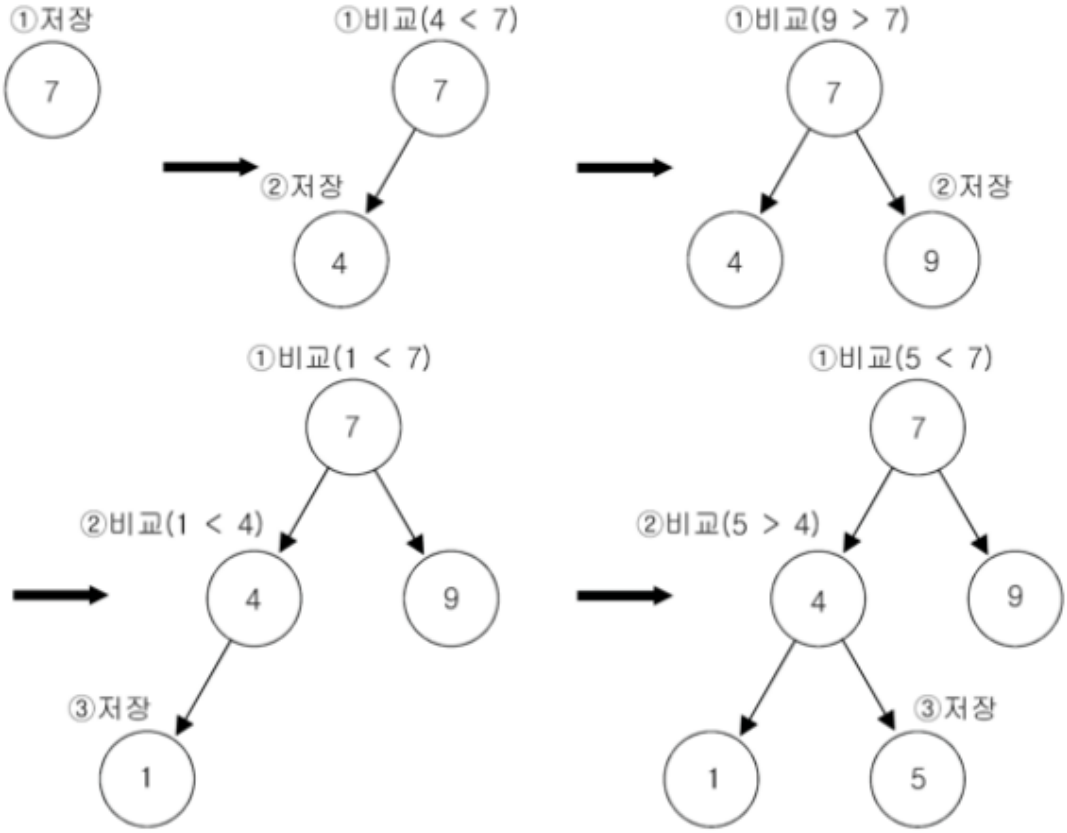
이진검색트리 중에서도 레드-블랙 트리로 구현되어있다.

부모노드보다 작은 값을 가진 노드는 왼쪽으로, 아닌건 오른쪽으로 배치해 데이터의 추가나 삭제시 트리가 한쪽으로 치우쳐지지 않도록 균형을 맞춘다.

- HashSet보다 데이터의 추가와 삭제는 시간이 더 걸리지만 검색과 정렬에는 유리합니다

```java
// TreeSet 선언, 크기 지정 불가
TreeSet<Integer> set1 = new TreeSet<Integer>();//TreeSet생성
TreeSet<Integer> set2 = new TreeSet<>();//new에서 타입 파라미터 생략가능
TreeSet<Integer> set3 = new TreeSet<Integer>(set1);//set1의 모든 값을 가진 TreeSet 생성
TreeSet<Integer> set4 = new TreeSet<Integer>(Arrays.asList(1,2,3));//초기값 지정
```

TreeSet에 값이 추가되는 과정




```
TreeSet<Integer> ts = new TreeSet<Integer>();

// add() 메소드를 이용한 요소의 저장
ts.add(30);
ts.add(40);
ts.add(20);
ts.add(10);

// Enhanced for 문과 get() 메소드를 이용한 요소의 출력
for (int e : ts) {
    System.out.print(e + " ");
}

// remove() 메소드를 이용한 요소의 제거
ts.remove(40);

// iterator() 메소드를 이용한 요소의 출력
Iterator<Integer> iter = ts.iterator();
while (iter.hasNext()) {
    System.out.print(iter.next() + " ");
}

// size() 메소드를 이용한 요소의 총 개수
System.out.println("이진 검색 트리의 크기 : " + ts.size());

// subSet() 메소드를 이용한 부분 집합의 출력
System.out.println(ts.subSet(10, 20));
System.out.println(ts.subSet(10, true, 20, true));
```

```
10 20 30 40
10 20 30
이진 검색 트리의 크기 : 3
[10]
[10, 20]
```

==========

Map

1. HashMap

   http://www.tcpschool.com/java/java_collectionFramework_map

   - 해시 알고리즘(hash algorithm)을 사용하여 검색 속도가 매우 빠릅니다.
   - 기존에 저장된 키와 동일한 키로 값을 저장하면 기존의 값은 없어지고 새로운 값으로 대치
   - 많은 양의 데이터를 검색하는 데 있어서 뛰어남

```
// HashMap 선언
HashMap<String,String> map1 = new HashMap<String,String>();//HashMap생성
HashMap<String,String> map2 = new HashMap<>();//new에서 타입 파라미터 생략가능
HashMap<String,String> map3 = new HashMap<>(map1);//map1의 모든 값을 가진
HashMap생성
HashMap<String,String> map4 = new HashMap<>(10);//초기 용량(capacity)지정
HashMap<String,String> map5 = new HashMap<>(10, 0.7f);//초기 capacity,load
factor지정
HashMap<String,String> map6 = new HashMap<String,String>(){{//초기값 지정
    put("a","b");
}};
```

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

// put() 메소드를 이용한 요소의 저장
hm.put("삼십", 30);
hm.put("십", 10);
hm.put("사십", 40);
hm.put("이십", 20);

// Enhanced for 문과 get() 메소드를 이용한 요소의 출력
System.out.println("맵에 저장된 키들의 집합 : " + hm.keySet());
for (String key : hm.keySet()) {
    System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
}

// remove() 메소드를 이용한 요소의 제거
hm.remove("사십");

// iterator() 메소드와 get() 메소드를 이용한 요소의 출력
Iterator<String> keys = hm.keySet().iterator();

while (keys.hasNext()) {
    String key = keys.next();
    System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
}

// replace() 메소드를 이용한 요소의 수정
```

```
hm.replace("이십", 200);

for (String key : hm.keySet()) {
    System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
}

// size() 메소드를 이용한 요소의 총 개수
System.out.println("맵의 크기 : " + hm.size());
```

```
맵에 저장된 키들의 집합 : [이십, 삼십, 사십, 십]
키 : 이십, 값 : 20
키 : 삼십, 값 : 30
키 : 사십, 값 : 40
키 : 십, 값 : 10

키 : 이십, 값 : 20
키 : 삼십, 값 : 30
키 : 십, 값 : 10

키 : 이십, 값 : 200
키 : 삼십, 값 : 30
키 : 십, 값 : 10

맵의 크기 : 3
```

리턴타입	메서드	내용
V	put(K key, V value)	키(key)와 값(value) 추가
V	get(Object key)	지정된 키(key) 값 리턴
V	remove(Object key)	키(key) 값에 해당되는 데이터 삭제
Set<K>	keySet()	저장된 모든 키(key) 리턴
Set<Map.Entry<K, V>>	entrySet()	저장된 모든 키(key)와 값(value) 리턴
boolean	containsKey(Object key)	해당 키(key) 지정 여부 확인
boolean	containsValue(Object value)	해당 값(value) 지정 여부 확인
int	size()	저장된 키(key)의 개수 리턴
void	clear()	저장된 모든 키(key)와 값(value) 삭제
boolean	isEmpty()	컬렉션이 비었는지 확인