컬렉션과 제네릭

안 화 수

컬렉션

- 컬렉션 프레임워크(Collection Framework)

1. 컬렉션은 자바에서 지원하는 자료구조를 의미한다.

2. 컬렉션은 배열의 한계를 많이 보완할 수 있다. (동일한 자료형, 배열의 크기)

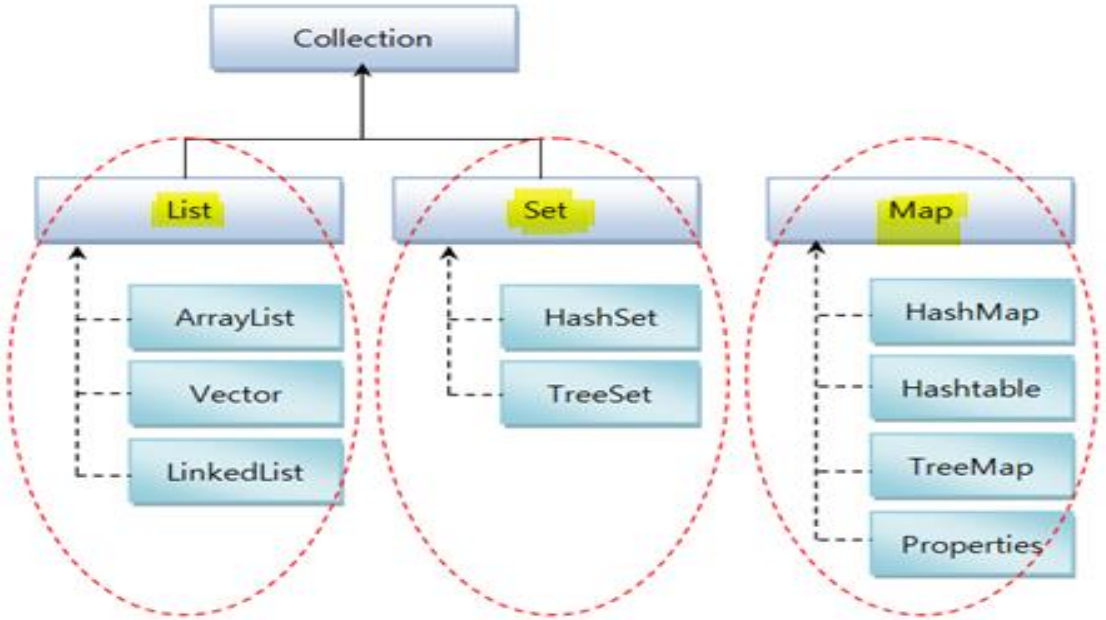
3. 컬렉션은 다양한 자료형의 데이터를 모두 저장할 수 있다.

4. 컬렉션은 동적으로 공간의 크기를 늘려서 저장 할 수 있다.

5. 컬렉션은 java.util 패키지 안에 들어있다.

컬렉션

- 컬렉션 프레임워크의 주요 인터페이스

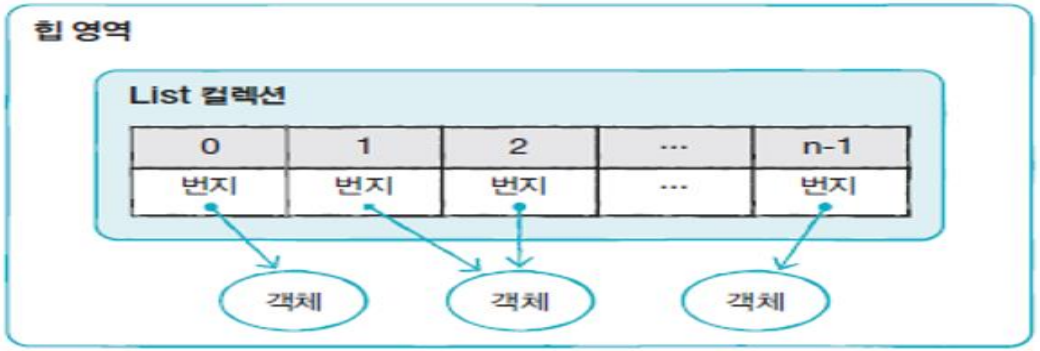


인터페이스 분류		특징	구현 클래스
Collection	List 계열	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set 계열	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet, TreeSet
Map 계열		- 키와 값의 쌍으로 저장 - 키는 중복 저장 안됨	HashMap, Hashtable, TreeMap, Properties

List 컬렉션

- List 인터페이스
- 상속받는 클래스 : ArrayList, Vector, LinkedList

1. 여러가지 자료형의 Data를 모두 저장할 수 있다.
   ex) int, double, char, boolean, String etc
2. 순서 있는 입.출력 처리(index번호 순으로 저장됨)
3. 중복된 Data를 저장 할 수 있다.
4. 동적으로 공간의 크기를 늘려서 저장할 수 있다.



List 컬렉션

- List 주요 메소드

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	set(int index, E element)	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어있는 전체 객체수를 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

List 컬렉션

- List 예1.

```
public class Collections02 {

  public static void main(String[] args) {
    // List list = new ArrayList();     업캐스팅
    ArrayList list = new ArrayList();
    list.add("하나");
    list.add(2);
    list.add(2);
    list.add(3.42);
    list.add("넷");
    list.add("five");
    list.add(6);

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

List 컬렉션

- List 예2. (1/2)

```
public class Collections04 {

    public static void main(String[] args) {
        List list = new ArrayList(); // 업캐스팅
        // ArrayList list = new ArrayList();

        list.add("하나");
        list.add(2);
        list.add(3.42);
        list.add("넷");
        list.add("five");
        list.add(6);
        System.out.println(list);

        System.out.println(list.get(2)); // 인덱스 2번 원소추출 : 3.42
        System.out.println(list.get(4)); // 인덱스 4번 원소추출 : five
```

List 컬렉션

- List 예2. (2/2)

```
            // Object  get(int index)
            for (int i = 0; i < list.size(); i++) {
            // System.out.println( i + " 번째 요소는 " + list.get(i));

                Object s = list.get(i);
            //  String s =(String)(list.get(i));
                System.out.println(s);
            }

            // 향상된 for문
            for (Object s : list) {
                System.out.print(s + "\t");
            }
            System.out.println();

            // 반복자 : 하나, 2, 3.42,   넷, five, 6
            Iterator elements = list.iterator();
            while (elements.hasNext()) {
                System.out.println("\t\t" + elements.next());
            }
        }
}
```

List 컬렉션

- ArrayList 클래스

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

- 저장용량(capacity)

  초기 용량 : 10

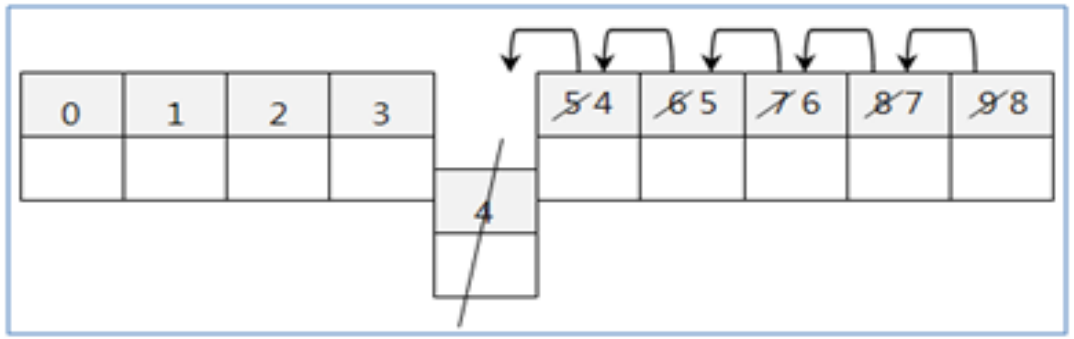
  저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어나다.

List 컬렉션

- ArrayList 클래스

- 객체 제거

  객체를 제거하면 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨진다.



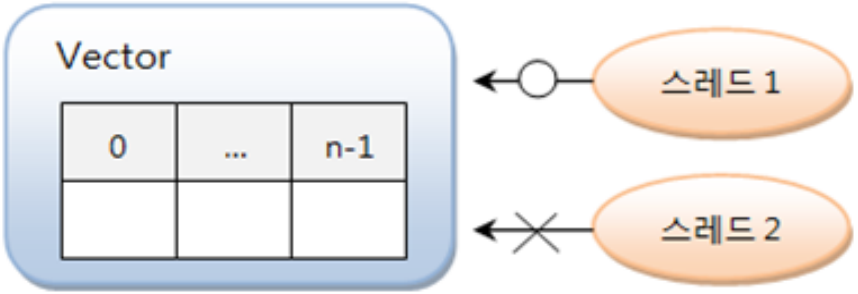
List 컬렉션

- Vector 클래스

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

- Vector 클래스는 스레드에 안전하다.

  Vector 클래스는 동기화된 메소드로 구성되어 있기 때문에 멀티 스레드가 동시에 Vector 클래스의 메소드를 실행할 수 없고, 하나의 스레드가 메소드 실행을 완료해야만 다른 스레드가 메소드를 실행할 수 있다. 그래서 멀티 스레드 환경에서 안전하게 객체를 추가, 삭제 할 수 있다.

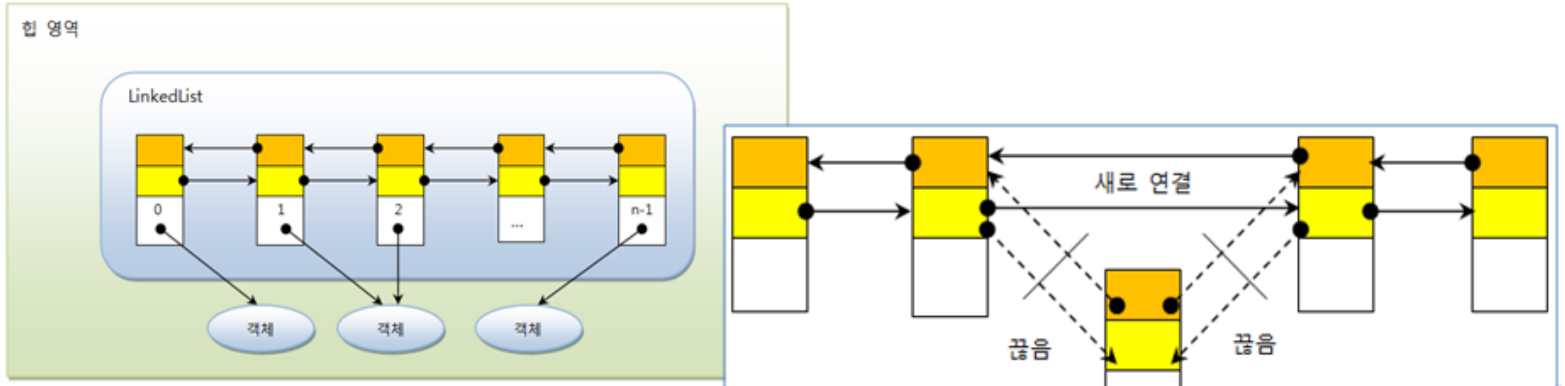


List 컬렉션

- LinkedList 클래스

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

- 인접 참조를 링크해서 체인처럼 관리한다.
- 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경한다.
- 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 성능이 좋다.



Set 컬렉션

- Set 인터페이스
- 상속받는 클래스 : HashSet, TreeSet

1. 여러가지 자료형의 Data를 모두 저장할 수 있다.
   ex) int, double, char, boolean, String etc
2. 순서 없는 입.출력 한다.
3. 중복된 Data를 저장하지 못한다.
4. 동적으로 공간의 크기를 늘려서 저장할 수 있다.

Set 컬렉션

❖ Set 주요 메소드

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 저장, 객체가 성공적으로 저장되면 true 를 리턴하고 중복 객체면 false 를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 조사
	Iterator<E> iterator()	저장된 객체를 한번씩 가져오는 반복자 리턴
	int size()	저장되어있는 전체 객체수 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

Set 컬렉션

- Set 예1.

```
public class Collections01 {
  public static void main(String[] args) {
    Set set = new HashSet();  // 업캐스팅
    // HashSet set = new HashSet();

    System.out.println("요소의 갯수->"+set.size());
    set.add("하나");
    set.add(2);
    set.add(3.42);
    set.add("넷");
    set.add("five");
    set.add(6);
    set.add(6);
    System.out.println("요소의 갯수->"+set.size());
    System.out.println(set);

    Iterator elements=set.iterator();
    while(elements.hasNext()) {
      System.out.println("\t\t" + elements.next());
    }
  }
}
```

Set 컬렉션

- Set 예2.

```
public class TreeSetTest {
    public static void main(String[] args) {
        TreeSet hs = new TreeSet();
        if(hs.add("korea")){
            System.out.println("첫 번째 korea 추가 성공");
        }else{
            System.out.println("첫 번째 korea 추가 실패");
        }
        if(hs.add("japan")){
            System.out.println("japan 추가 성공");
        }else{
            System.out.println("japan 추가 실패");
        }
        if(hs.add("america")){
            System.out.println("america 추가 성공");
        }else{
            System.out.println("america 추가 실패");
        }
        if(hs.add("britain")){
            System.out.println("britain 추가 성공");
        }else{
            System.out.println("britain 추가 실패");
        }
        if(hs.add("korea")){
            System.out.println("두 번째 korea 추가 성공");
        }else{
            System.out.println("두 번째 korea 추가 실패");
        }
        Iterator it = hs.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}
```

Map 컬렉션

- Map 인터페이스
  - 상속받는 클래스 : HashMap, HashTable

1. 여러가지 자료형의 Data를 모두 저장할 수 있다.
   ex) int, double, char, boolean, String etc
2. Data를 저장할 때 Key, Value 를 동시에 저장한다.
3. key 값은 중복이 되면 안된다.
   만약에 중복된 key가 있으면, 가장 마지막에 설정된 value만 사용할 수 있다.
4. value값은 중복이 되어도 상관없다.

Map 컬렉션

- Map 주요 메소드

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장이 되면 값을 리턴
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set 에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값 Collection 에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry 삭제, 삭제가 되면 값을 리턴

Map 컬렉션

- Map 예1.

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

        // Map hm = new HashMap();
        HashMap hm = new HashMap();

        //키와 데이터 쌍으로 저장
        hm.put( "woman", "gemini" );
        hm.put( "man", "johnharu" );
        hm.put( "age", new Integer(10) );
        hm.put( "city", "seoul" );

        System.out.println( hm );

        //키 값만 출력
        System.out.println( hm.keySet() );

        //키를 이용해 해당 데이터를 출력
        System.out.println( hm.get( "woman" ));
        System.out.println( hm.get( "city" ));
    }
}
```

Map 컬렉션

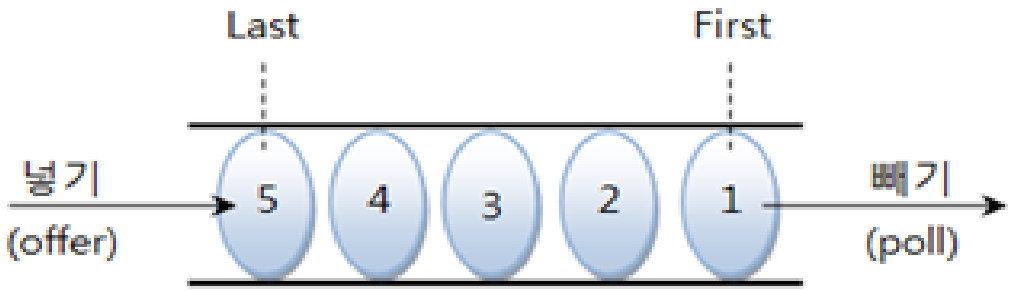
- Map 예2.

```
public class HashTableTest {
  public static void main(String[] args) {
      Map ht = new Hashtable();
      // Hashtable ht= new Hashtable();

      ht.put("딸기", "StrawBerry");
      ht.put("사과", "Apple");
      ht.put("포도", "Grapes");

      //  Object obj = ht.get("포도");
      String Val = (String)ht.get("포도");
      if(Val != null) {
        System.out.println("포도-> " + Val);
      }

      Enumeration Enum = ht.keys();
      while(Enum.hasMoreElements()){
          Object  k = Enum.nextElement();
          Object  v = ht.get(k);
          System.out.println(k + " : "+ v );
      }
   }
}
```

Queue 컬렉션

- 큐(Queue) 인터페이스
- 상속받는 클래스 : LinkedList

- FIFO(First Input First Output) 구조
  먼저 입력된 자료가 먼저 출력되는 구조



Queue 컬렉션

- 큐(Queue) 주요 메소드

리턴타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣는다.
E	peek()	객체 하나를 가져온다. 객체를 큐에서 제거하지 않는다.
E	poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.

Queue 컬렉션

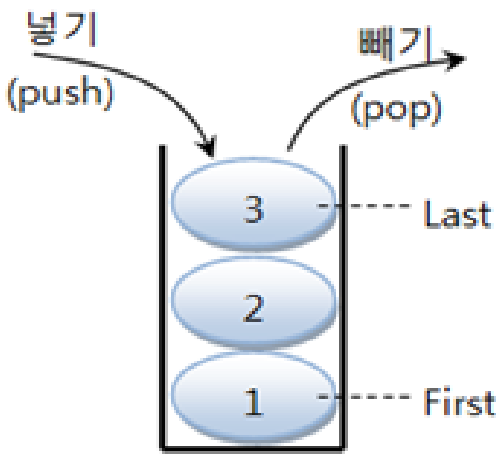
- Queue 예1.

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

    LinkedList myQue = new LinkedList();
    myQue.offer("1-자바");
    myQue.offer("2-C++");
    myQue.offer("3-API");
    myQue.offer("4-MFC");

    while(myQue.peek() != null)          //큐가 비어있지 않다면
      System.out.println(myQue.poll()); //큐에서 데이터를 꺼내온다.
  }
}
```

Stack 컬렉션

- 스택(Stack) 클래스

- LIFO(Last Input First Output) 구조
  마지막으로 입력된 자료가 가장 먼저 출력되는 구조



스택(LIFO)

Stack 컬렉션

- Stack 주요 메소드

리턴타입	메소드	설명
E	push(E item)	주어진 객체를 스택에 넣는다.
E	peek()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거하지는 않는다.
E	pop()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거한다.

Stack 컬렉션

- Stack 예1.

```
public class StackTest01 {
  public static void main(String[] args) {
    Stack myStack = new Stack();
    myStack.push("1-자바");
    myStack.push("2-C++");
    myStack.push("3-API");
    myStack.push("4-MFC");

    while(!myStack.isEmpty()) {
      System.out.println( myStack.pop());
    }
  }
}
```

Stack 컬렉션

- Stack 예2.

```
public class StackTest {
      public static void main( String[] args ) {
      // Stack 객체 생성
      Stack s = new Stack();
      System.out.println( s.empty() );

      // Stack에 값을 넣음
      s.push( "gemini" );
       Date d = new Date();
      s.push( d );
      s.push( new Integer( 10 ));
      s.push( "johnharu" );

      // Stack의 값을 출력
      System.out.println( s.empty() );
      System.out.println( s.peek() );
      System.out.println( s.pop() );
      System.out.println( s.pop() );
      System.out.println( s.pop() );
      System.out.println( s.pop() );
      System.out.println( s.empty() );
      }
}
```

제네릭

- 제네릭(Generic)

1. 제네릭은 JDK1.5 부터 지원하는 기능이다.
2. 제네릭은 컬렉션에 한가지 자료형의 데이터만 저장 할 수 있는 기능을 제공한다.
3. 제네릭을 사용하면 컬렉션에 저장된 데이터를 구해올 때 컬렉션에 저장된 자료형을 생략할 수 있다.

제네릭

- 제네릭(Generic)

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

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

List<Double> list = new ArrayList<Double>();

Vector<String>  v =  new Vector<String>();

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

제네릭

- 제네릭(Generic) 예1.

  제네릭을 사용하지 않으면, 형변환(다운캐스팅) 할 자료형을 생략할 수 없다.

```
public class Collections05 {
   public static void main(String[] args) {

      // 제네릭을 사용하지 않은 경우
      Vector vec = new Vector();
      vec.add("Apple");
      vec.add("banana");
      vec.add("oRANGE");

      String temp;
      for(int i=0; i<vec.size(); i++){
           temp=(String) vec.get(i); //다운 캐스팅
//         temp=vec.get(i);
           System.out.println(vec.get(i));
           System.out.println(temp.toUpperCase());
      }
   }
}
```

제네릭

- 제네릭(Generic) 예2.

  제네릭을 사용하면, 형변환(다운캐스팅) 할 자료형을 생략할 수 있다.

```
public class Collections06 {
   public static void main(String[] args) {

      // 제네릭을 사용한 경우
      Vector<String> vec = new Vector<String>();

      vec.add("Apple");
      vec.add("banana");
      vec.add("oRANGE");

      String temp;
      for(int i=0; i<vec.size(); i++){
           temp=vec.get(i);   // 자료형 생략 가능함
           System.out.println(temp.toUpperCase());
      }
   }
}
```

제네릭

- 제네릭(Generic) 예3.

  제네릭을 사용하면, 형변환(다운캐스팅) 할 자료형을 생략할 수 있다.

```
public class HashTableTest02{
  public static void main(String[] args) {

    Hashtable<String, String> ht= new Hashtable<String, String>();
    ht.put("사과", "Apple");
    ht.put("딸기", "StrawBerry");
    ht.put("포도", "Grapes");

    String Val = ht.get("포도");
    if(Val != null) {
      System.out.println("포도-> " + Val);
    }

    Enumeration<String> Enum = ht.keys();
    while(Enum.hasMoreElements()){
      String  k = Enum.nextElement();
      String  v = ht.get(k);
      System.out.println(k + " : "+ v );
    }
  }
}
```