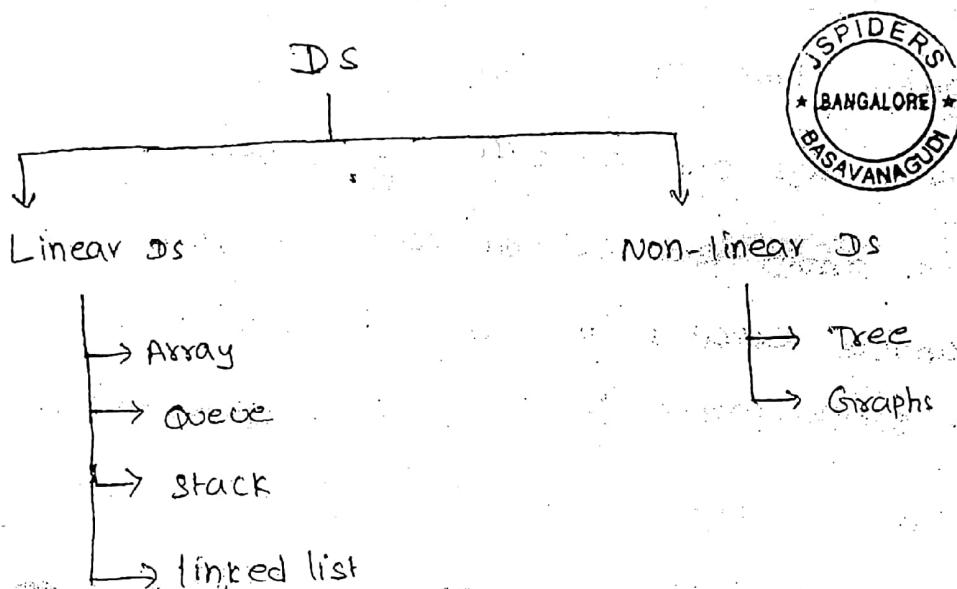


①

* 9686114422
* 9686995511

DATA STRUCTURES

→ It is the particular way of organising the data in the program to use them efficiently in the applications.



ISPIDERS Basavangudi

Linear Data Structure:

→ It is a type of data structure where the data is stored one next to another in linear order.

↔

Non-Linear Data Structure:

→ It is a type of data structure where the data is stored according to some relations.

Arrays (continued) :-

9686114422

9686 995511

- The size of the array should be always specified using Byte (or) short (or) int data types only
- Array size can never be specified as -ve number.
If we specify array size as -ve number then JVM throws Negative Array Size Exception.
- If we specify the array size as zero, there will be no compile time errors (or) run time Exceptions
- * In java arrays are considered as objects.
- In java arrays are created new using new operator
- The variable created for array is actually a reference variable which points to array object.
- If you try to print the reference variable of the array then it always prints the string representation.

* Object array:-

- It is a type of array where every bucket of the array is a reference variable.
- Using the object array we can manage the objects easily.
- Object array will be very useful whenever we want perform the same operations on the every object present in object array.

class program

3

9686 114422

9686 995511

PSV main (String[3] args)

{

Object [] ar1 = new Object[5];

ar1[0] = new Object();

ar1[1] = new Object();

ar1[2] = new Object();

ar1[3] = new Object();

ar1[4] = new Object();

for (int i=0 ; i< ar1.length ; i++)

{

hi= ar1[i].hashCode();

SOP (hi);

}

}

3

Output:

366712642

1829164700

2018699554

1311053135

118352462



9686 114422

9686 998511

class Student

4

{ String name;

int eid;

double marks;

Student (String name, int eid, double marks)

{

 this. name = name;

 this. eid = eid;

 this. marks = marks;

}

@Override

public String toString()

{

 String info = sid + " " + name + " " + marks;

 return info;

}

3

Class Program

{

 public static void main (String [] args)

{

 Student [] st = new Student [5];

 st[0] = new Student ("smith", 7893, 451);

 st[1] = new Student ("martin", 7641, 357);

 st[2] = new Student ("miller", 7614, 400);

 st[3] = new Student ("allen", 7454, 550);

 st[4] = new Student ("scott", 7111, 521);

 for (int i=0; i < st.length; i++)

{

 System.out.println (st[i]);

}

3

Output

7893 Smith 451.0

Java Collection Framework (JCF)

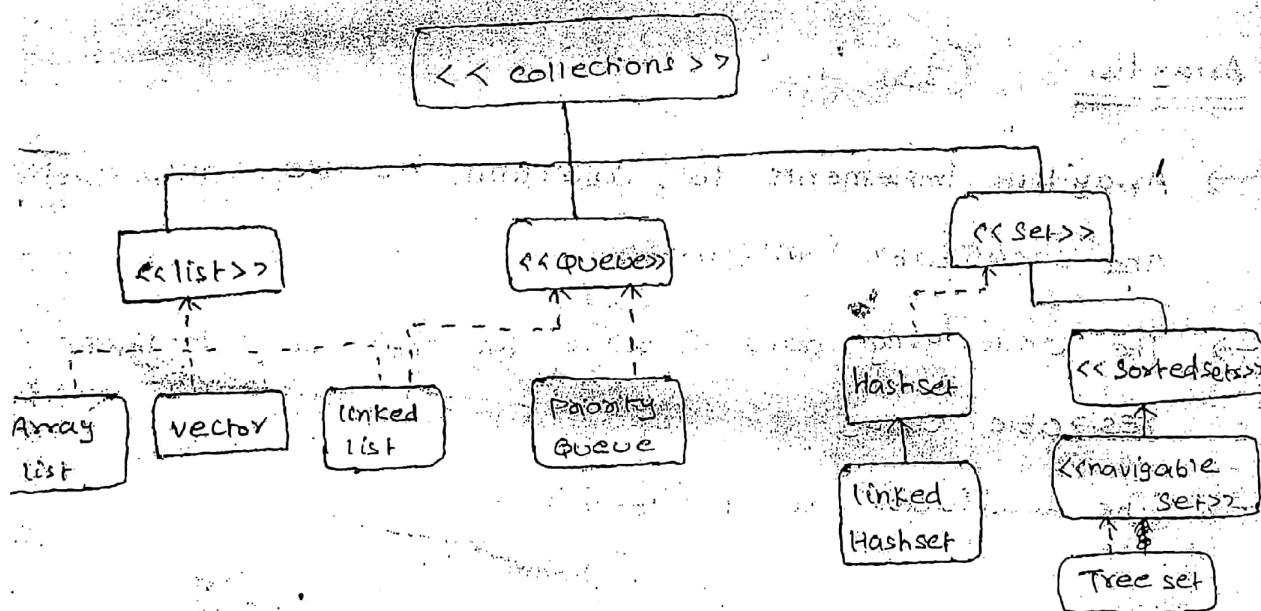
Collection :-

→ Collection is a group of references which is represented as one unit.

Properties of Collection :-

- Collection supports heterogeneous data (references).
- Collections are growable in nature, the size of the collection can be increased dynamically at run time.
- Every collection will have atleast one underlying data structure.
- Every collection will have inbuilt methods for basic operations like add, remove, replace etc. on the collections.

Collections framework :-



example for compile
time polymorphism

***example for constructor

6 overloading (oral)***

#9686114422

Constructors of ArrayList :-

#9686995511

1. ArrayList () :-

→ constructs an empty list with an initial capacity
of 10

2. ArrayList (int initialCapacity) :-

→ constructs an empty list with specified initial
capacity

3. ArrayList (Collection c) :-

→ constructs an ArrayList from the given collection.

→ The new capacity of ArrayList is calculated as
follows

$$\text{new capacity} = (\text{old capacity} * 3/2) + 1$$

Load factor :-

→ It is the threshold capacity which indicates
when the new collection should be created with
the new capacity.



- collection is the root interface of the Java Collection Framework.
- Java Collection framework is a group of classes and interfaces which are used to manage the reference present in different collections.
- Every collection will have its own properties and hence every collection will contain the methods which specific to that type of collection.

<< list >>



- It allows duplicates
- It allows null values (multiple null values)
- List has index
- List preserves insertion order.

ArrayList :- (JDK 1.2)

- ArrayList implements List, Collection, Iterable, Clonable and Serializable interfaces.
- The underlying data structure for the ArrayList is resizable array
- The load factor is 1 (100%).

```
import java.util.ArrayList;  
Class Mainclass  
{
```

```
    . Psvm main (String[] args)  
    {
```

```
        ArrayList al = new ArrayList();
```

```
        al.add(10);
```

```
        al.add("hello");
```

```
        al.add(3.54);
```

```
        al.add(null);
```

```
        al.add(null);
```

```
        al.add(10);
```

```
        for (int index = 0; index < al.size(); index++)
```

```
        for (int index = 0; index < al.size(); index++)
```

```
{
```

```
        System.out.println(al.get(index));
```

```
}
```

```
}
```

```
{
```

O/P:

10

hello

3.54

null

null

10

#9686114422
#9686995511



- Within the collection we cannot store primitive values.
- If we try to add primitive values for the given collection then JVM
 - 1. Converts primitive value to corresponding wrapper class object
 - 2. Stores the address of the wrapper class object within the given collection.
- If we try to add any object reference to any collection then the given reference will be upcasted to object type reference
- While retrieving the reference if we want to access any subclass properties then, we should downcast the reference for the corresponding.

Subclass type:

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

Class MainClass

{

PSV main (String [] args)

{

ArrayList al = new ArrayList();

al.add ("hello");

al.add (new String ("java"));

SOP (al.get(0));

String s1 = (String) al.get(0);

SOP (s1.length());

O/P

hello

5

}



Vector :... (JDK 1.0)

- Vector implements list, collection, iterable, clone and Serializable interfaces.
- The underlying data structure for the Vector is resizable array

→ The load factor is 1 (100%)

Constructors of Vector

1. vector()

→ Constructs an empty list with an initial capacity of 10

2. vector(int initialCapacity)

→ Constructs an empty list with specified initial capacity.

3. vector(Collection c)

→ Constructs a list from the given collection

11

#9686114422
9686995511

```
import java.util.Vector;  
class MainClass  
{  
    public static void main (String [] args)  
    {  
        Vector v1 = new Vector ();  
        v1.add ("10");  
        v1.add ("hello");  
        v1.add (3.54);  
        v1.add (null);  
        v1.add (null);  
        v1.add ("10");  
  
        for (int index = 0; index < v1.size (); index++)  
        {  
            v1.get (index);  
        }  
    }  
}
```

O/P:-

10

hello.

3.54

null

null

10

Set :-

- Set do not preserve insertion order
- Set do not have any index
- Set do not allow duplicates
- Only one null value is allowed

* How set preserves uniqueness?

- Set preserves the uniqueness by comparing the hashCode values of the given object with every other object present in the set.
- If the hashCode values are same then the corresponding object will be rejected.

HashSet :- (JDK 1.2)

- HashSet implements Serializable, Cloneable, Iterable, Collection and Set interfaces.
- The underlined data structure of HashSet is hashtable.

****** ex for constructor overloading *****

HashSet () :- constructs a new, empty set with initial capacity (16) and load factor (0.75).

HashSet (int initialCapacity) :- constructs a new empty set with specified initial capacity and default load factor (0.75)

HashSet (int initialCapacity, float loadFactor) :-

constructs a new empty set with the specified initial capacity and the specified load factor.

HashSet (Collection c) :-

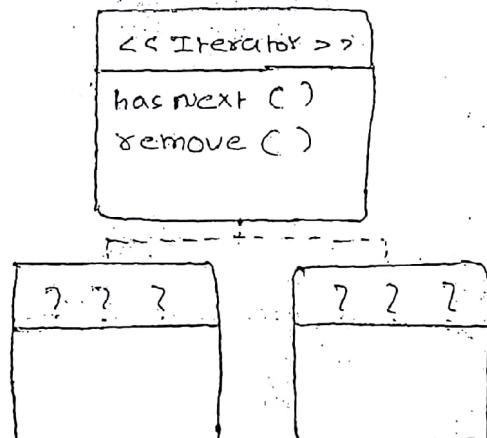
constructs a new set by converting any given collection to set

→ To retrieve the elements from the collection we can use

1. Iterator
2. Advanced for loop (For each loop)

→ Normally iterator (or) for each loop are used with collections which do not have index.

Iterator implementation:-



IMPLEMENTATION OF ITERATOR

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

```
import java.util.Iterator;
```

```
Class Mainclass
```

```
{
```

```
    PS U main (String C.I. args)
```

```
{
```

```
    HashSet h1 = new HashSet();
```

```
    h1.add (10);
```

```
    h1.add (20);
```

```
    h1.add (30);
```

```
    h1.add (10);
```

```
    h1.add (50);
```

```

Iterator il = new Iterator();
il.iterator();
while (il.hasNext() == true)
{
    System.out.println(il.next());
}
}

```

Output :-

50
20
10
30



Iterator :-

- Iterator is used to retrieve the elements from any collection with next method.
- To retrieve the elements using the iterator we have to follow 3 steps.
 1. get the iterator object using any collection reference variable by calling iterator method.
 2. use a while loop with hasNext method which returns true if the next element is present else it returns false
 3. use next method within the body of while loop which returns the elements from the collection.
- The next method throws NoSuchElementException, if you try to retrieve the elements from the collection after the last element

For each loop :-

15

9686114422

9686995511

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

```
class MainClass
```

```
{
```

```
    public static void main (String [] args)
```

```
{
```

```
    HashSet hi = new HashSet();
```

```
    hi.add ("Hello");
```

```
    hi.add ("java");
```

```
    hi.add ("Android");
```

```
    hi.add ("apple");
```

```
    for (Object ref : hi)
```

```
{
```

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

```
        String s1 = (String) ref;
```

```
        System.out.println (s1.length());
```

```
}
```

```
}
```

Syntax :-

```
for (classname reference : collection.ref)
```

```
{
```

```
    Stmts ...
```

```
    Stmts ...
```

```
}
```

→ For each loop is used to retrieve the elements from collections and also from object arrays.

→ For each loop internally creates an iterator to retrieve elements from the given collection (or) object array



Output:-

apple

5

java

4

Android

7

apple

5

Linked HashSet :- (JDK 1.4) 16 # 9686114422
9686995511

→ The underlined data structure for LinkedHashSet are

1. Hash table

2. Linked list

→ LinkedHashSet preserves the insertion order

Constructors of linked HashSet :-

LinkedHashSet() :-

→ Constructs a new, empty set with initial capacity (16)

and load factor (0.75)

→ LinkedHashSet(int initialCapacity)

LinkedHashSet(int initialCapacity) :-

→ Constructs a new empty set with specified initial capacity and default load factor (0.75).

LinkedHashSet(int initialCapacity, float loadFactor) :-

→ Constructs a new empty set with specified initial capacity and specified load factor.

→ LinkedHashSet(Collection c) :-

→ constructs a new set by converting any given collection to set

Tree Set :-

17

9686995511

9686114422

Tree :-

- A tree is the non-linear datastructure where the data represented according to some relation.
- The tree is represented with nodes and links
- The node consists of data, Address of left node and address of right node

Binary Tree :-

- It is a type of tree where a parent node can have maximum of 2 child nodes

Properties of TreeSet :-

- The underlying datastructure of TreeSet is Red-black
- TreeSet implements Serializable, Clonable, Iterable, Collection, NavigableSet, Set, SortedSet

Constructors of TreeSet :-

TreeSet() :- Constructs a new, empty tree set, sorted according to the natural ordering of its elements.

TreeSet (Comparator comp) :-

Constructs a new, empty tree set, sorted according to the specified comparator.

- ***
→ Every object added to the TreeSet should be implementing Comparable interface and override compareTo method

⇒ Non-comparable Classes :-

String Buffer

String Builder

Object class etc.

Class Program

```
import java.util.TreeSet;
```

```
import java.util.Iterator;
```

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

```
{
```

```
TreeSet hl = new TreeSet ();
```

```
hl.add ("computer");
```

```
' hl.add ("desktop");
```

```
hl.add ("board");
```

```
hl.add ("monitor");
```

```
hl.add ("apple");
```

```
hl.add ("aaa");
```

```
Iterator ii = new Iterator (hl);
```

```
hl.iterator ();
```

```
while (ii.hasNext () == true)
```

```
{
```

```
System.out.println (ii.next());
```

```
}
```

```
}
```

Output :-

aaa

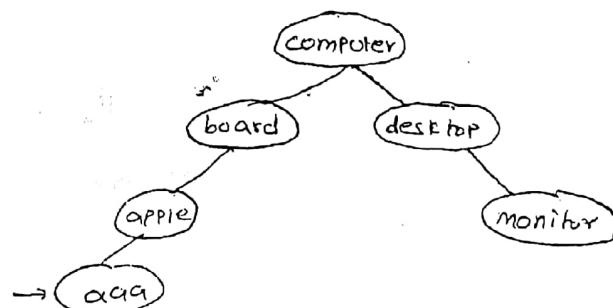
apple

board

computer

desktop

monitor



Left

Parent

right

Comparable interface:

#9686 114422
#9686 995511

- It helps in sorting the java objects.
- It contains only one abstract method by the name `compareTo`
- Whenever the objects needs compared to perform sorting (or) to decide Priority `compareTo` method be overridden to contain the logic for the comparison.
- After comparing given two Objects, `compareTo()` returns
 - +ve integer if the given object value is greater than
 - ve integer if the given object value is lesser than
 - (zero) if the given object value is equal to another

Comparator interface:

- It is used to sort the objects in customized sorting.
- If we don't want to use natural sorting order then we can customize the sorting order according to the application requirement with comparator interface.
- Comparator interface contains an abstract method by name `compare`
- Comparator is a functional interface and supports Lambda functions.
- `compare` method compares given two objects and

```
import java.util.Comparator;  
import java.util.TreeSet;  
import java.util.Iterator;
```

#9686114422
#9686995511

Class Mainclass3

{

```
public main (String[] args)
```

```
Comparator Cmp1 = (Object obj1, Object obj2) ->  
{
```

```
    String s1 = (String) obj1;
```

```
    String s2 = (String) obj2;
```

```
    int value = s2.compareTo(s1);
```

```
    return value;
```

```
}
```

```
TreeSet t1 = new TreeSet(Cmp1);
```

```
t1.add ("computer");
```

```
t1.add ("beer");
```

```
t1.add ("duke");
```

```
t1.add ("makeup kit ");
```

```
t1.add ("apple");
```

```
t1.add ("oaa");
```

```
Iterator ii = t1.iterator();
```

Output:-

makeup kit

duke

computer

beer

apple

oaa

```
while (ii.hasNext() == true)  
{
```

```
    System.out.println(ii.next());
```

```
}
```

Queue :-

- Duplicates are allowed
- Multiple Null Values are allowed
- Insertion order is preserved
- It has Index.

Priority Queue :- (JDK 1.5)

- Duplicates are allowed
- Not even one null value is allowed.
- Insertion order is not preserved.
- It does not have index
- Objects are removed from the priority queue based some priority.



- ~~Priority Queue Implementation~~
- Priority queue implements Serializable, Iterable, Collection interfaces.
 - The underlined datastructure of the priority queue is Priority queue data structure.

Constructors of Priority Queue :-PriorityQueue() :-

- creates a PriorityQueue with the default initial capacity that orders its elements according to their natural order.

PriorityQueue (int initialCapacity) :-

- creates a PriorityQueue with specified initial capacity that orders its elements according to their natural order.

PriorityQueue (collection c) :-

→ Creates a priority queue by converting any collection to priority queue

```
import java.util.PriorityQueue;
```

```
import java.util.Iterator;
```

```
class Customer implements Comparable
```

```
{
```

```
    String name;
```

```
    String address;
```

```
    int priority;
```

```
Customer (String name, int priority)
```

```
{
```

```
    this.name = name;
```

```
    this.priority = priority;
```

```
}
```

```
public int compareTo (Object obj)
```

```
{
```

```
    Customer ct1 = (Customer) obj;
```

```
    int value = ct1.priority - this.priority;
```

```
    return value;
```

```
}
```

```
public String toString ( )
```

```
{
```

```
    String info = name + " " + address + " " + priority;
```

```
    return info;
```

```
}
```



Class mainclass 23

{
PSU main (String[] args)

PriorityQueue q1 = new PriorityQueue();
q1.add (new Customer ("Rama", 3));
q1.add (new Customer ("Bhima", 2));
q1.add (new Customer ("Soma", 5));
q1.add (new Customer ("Shyma", 5));
q1.add (new Customer ("Obama", 3));
SOP (q1.size());

while (q1.isEmpty() == false)
{
 SOP (q1.poll());
}

SOP (q1.size());
}

Output :-

Size = 5

Soma 5

Shyma 5

Obama 3

Rama 3

Bhima 2

Size = 0

#9686114422
9686 995511



IF 9686 114022
IF 9686 995511

Reo Poll () :-

24

- It returns the first element of the queue and also removes it from the given queue.

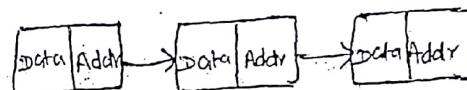
Read Peek () :-



- It returns the first element of the given queue but does not remove it from the given queue.

Linked list :-

- It is a group of nodes which is linked to each other.
- The node of a linked list contains
1. Data
 2. Address of next node
- It, implements list and queue interfaces.
- Since it is implementing both list and queue it can be used as a list (or) normal queue (FIFO)



Constructors of linked list:-

LinkedList () :-

- Constructs an empty LinkedList

LinkedList (collection c) :-

- constructs a linked list by converting any given collection to linked list

- To receive the elements from the linked list we can use get(), poll(), and peek().

- The underlying data structure for the linked list is collection is LinkedList datastructure

import java.util.LinkedList;
 class MainClass
 {
 public static void main (String [] args)
 {
 LinkedList ls1 = new LinkedList();
 ls1.add (50);
 ls1.add (60);
 ls1.add (80);
 ls1.add (20);
 System.out.println (ls1.size());
 for (int index=0; index < ls1.size(); index++)
 {
 System.out.println (ls1.get(index));
 }
 System.out.println (ls1.size());
 while (ls1.isEmpty == false)
 {
 System.out.println (ls1.poll());
 }
 System.out.println (ls1.size());
 }
 }

Output:-

4 (size)

50

60

80

20

4 (size)

50

60

80

20

25



#9686995511

#9686114422

→ Generics are used with the collections to achieve type safety goals

1. Providing type safety (

2. To avoid unnecessary downcasting of objects

Syntax for generic collection :-

CollectionName < Type > ref = new CollectionName < Type > ();

Ex:- ArrayList < String > al = new ArrayList < String > ();

→ Generics are specified with declaration of collection
within angular brackets

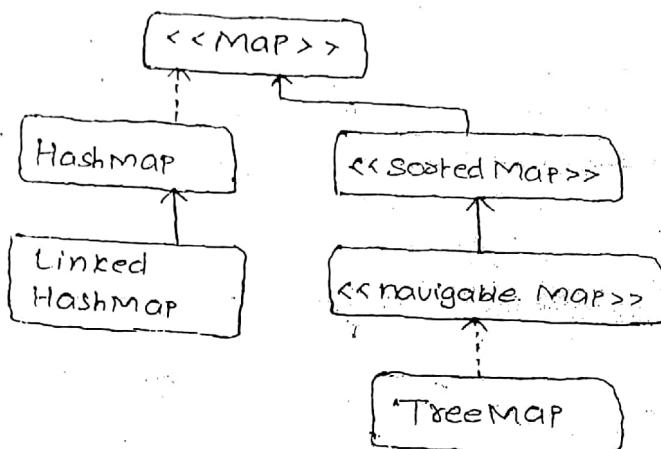
Note:-

→ Java arrays are by default type-safe

```
import java.util.ArrayList;
class MainClass
{
    public static void main (String [] args)
    {
        ArrayList < String > al = new ArrayList < String > ();
        al.add ("Android"); (new String ("Hello"));
        al.add ("Java"); (new String ("Java"));
        al.add ("Android"); (new String ("Android"));
        System.out.println (al);
        // al.add (new StringBuffer ("HTML")) C.T.E //.
        for (int index = 0; index < al.size(); index++)
        {
            String s1 = al.get(index);
            System.out.println (s1.length());
        }
    }
}
```

Output:-

5
4
7



- The data is stored in key-value format
- The keys of map cannot be duplicate
- Only one null can be allowed as key
- Values can be duplicate
- Multiple null can be used as values
- Map interface donot extends collection interface
- We can add the elements to the map using put()
- Syntax: Put (key, value);
- The key and value can be of any datatype
- To retrieve the elements from the hashMap we can use get() Syntax: get(key);
- If we pass an invalid key then get() returns null value.
- If we try to add duplicates then the old value will be replaced with the new value.

9686 114422

9686 995511

import java.util.HashMap;

Class Mainclass

{

PSV main (String [] args)

{

HashMap hi = new HashMap();

hi.put ("key1", 58);

hi.put ("key2", 78);

hi.put ("null", 87);

SOP (hi.get ("key1")); → 58

SOP (hi.get (null)); → 87

SOP (hi.get ("key123")); → null

SOP (hi.get

hi.put ("key1", 12345);

SOP (hi.get ("key1")); → 12345.

{

}

OUTPUT:-

58

87

null

12345