**Collection interview Question 76.**

Collection **Output** interview question **26.**

|  |
| --- |
| **import** java.util.HashMap;  **import** java.util.Map;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String args[]) {            Map<String, String> hashMap = **new** HashMap<String, String>();            hashMap.put(**new** String("a"), "audi");            hashMap.put(**new** String("a"), "ferrari");            System.*out*.println(hashMap);     }  }  /\*OUTPUT  {a=ferrari}  \*/ |

**Answer.** HashMap does not allow duplicate keys. **HashMap** when comparing keys (and values) performs object-equality not reference-equality. In an HashMap, two keys k1 and k2 are equal if and only if (k1==null ? k2==null : k1.equals(k2))

**new** String("a") & **new** String("a") are **different by reference** but **equal by value.**

**Collection interview Question 77.**

Collection **Output** interview question **27.**

|  |
| --- |
| **import** java.util.IdentityHashMap;  **import** java.util.Map;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String args[]) {            Map<String, String> identityHashMap = **new** IdentityHashMap<String, String>();            identityHashMap.put(**new** String("a"), "audi");            identityHashMap.put(**new** String("a"), "ferrari");            System.*out*.println(identityHashMap);     }  }  /\*OUTPUT  {a=audi, a=ferrari}  \*/ |

**Answer.**

[**IdentityHashMap**](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html) when comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2). (In normal Map implementations (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).)

**new** String("a") & **new** String("a") are **different by reference.**

**Must read :** [**Differences and Similarities between HashMap and IdentityHashMap with program in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html)

**Collection interview Question 78.**

Collection **Output** interview question **28.**

|  |
| --- |
| **import** java.util.Map;  **import** java.util.TreeMap;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** TreeMapTest {  **public** **static** **void** main(String args[]) {            Map<Integer, String> m = **new** TreeMap<Integer, String>();            m.put(11, "audi");            m.put(**null**, **null**);            m.put(11, "bmw");            m.put(**null**, "fer");            System.*out*.println(m.size());            System.*out*.println(m);     }  } |

**Answer.**  NullPointerException

[**TreeMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) does not any null key or null value.

**Collection interview Question 79.**

Collection **Output** interview question **29.**

|  |
| --- |
| **import** java.util.Arrays;  **import** java.util.Comparator;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            String[] ar = { "c", "d", "b", "a", "e" };            NestedClass in = **new** NestedClass();            Arrays.*sort*(ar, in);  **for** (String str : ar)                   System.*out*.print(str + " ");            System.*out*.println(Arrays.*binarySearch*(ar, "b"));     }  **static** **class** NestedClass **implements** Comparator<String> {  **public** **int** compare(String s1, String s2) {  **return** s2.compareTo(s1);            }     }  } |

**Answer.**

/\*

e d c b a -1

\*/

>compareTo() method will do the reverse sorting.

>binarySearch() gives –1 because it should have been invoked using the same Comparator as was used during reverse sorting of the array.

Read:

[**COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50**](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

[**COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75**](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

**Collection interview Question 80.**

Collection **Output** interview question **30.**

|  |
| --- |
| **import** java.util.EnumSet;  **import** java.util.Set;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** EnumSetTest {  **private** **enum** Days {  *Monday*, *Tuesday*, *Wednesday*, *Thursday*, *Friday*, *Saturday*, *Sunday*;  **public** **static** Set<Days> *allDays* = EnumSet.*allOf*(Days.**class**);    **public** **static** Set<Days> *weekDays* = EnumSet.*range*(*Monday*, *Friday*);    **public** **boolean** isWeekDay() {  **return** *weekDays*.contains(**this**);     }     }     /\*\* Main \*/  **public** **static** **final** **void** main(**final** String args[]) {            System.*out*.println(Days.*weekDays*.size());              Days day=Days.*Monday*;            System.*out*.println( (day.isWeekDay() ? "is WeekDay" : "is weekEnd"));              day=Days.*Sunday*;            System.*out*.println( (day.isWeekDay() ? "is WeekDay" : "is weekEnd"));         day=Days.*Monday*;     System.*out*.println(Days.*allDays*.contains(day));     System.*out*.println(day.ordinal());     }  } |

**Answer.**

/\*OUTPUT

5

is WeekDay

is weekEnd

true

0

\*/

*Read :* [***EnumSet in java***](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html)

**Collection interview Question 81.**

Collection **Output** interview question **31.**

|  |
| --- |
| **import** java.util.LinkedHashSet;  **import** java.util.Set;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** LinkedHashSetTest {  **public** **static** **void** main(String args[]) {            Set s = **new** LinkedHashSet();            s.add("1");            s.add(1);            s.add(3);            s.add(2);            System.*out*.println(s);     }  } |

**Answer.**

/\* OUTPUT

[1, 1, 3, 2]

\*/

[LinkedHashSet](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) maintains insertion order and does not allow duplicates.

*Read :* [*HashSet vs LinkedHashSet vs TreeSet - Similarity and Differences*](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)

**Collection interview Question 82.**

Collection **Output** interview question **32.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **class** Employee **implements** Comparable<Employee>{     String name;     String id;  **public** Employee(String name, String id) {  **this**.name = name;  **this**.id = id;     }     @Override  **public** **int** compareTo(Employee otherEmployee) {  **return** **this**.name.compareTo(otherEmployee.name);     }     @Override  **public** String toString() {  **return** "{" + "name=" + name + ", id=" + id  + '}';     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ComparableUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("sam","4");         Employee emp2=**new** Employee("amy","2");         ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list);         System.*out*.println(list);       }  } |

**Answer.**

/\*OUTPUT

[{name=amy, id=2}, {name=sam, id=4}]

\*/

compareTo method of Comparable has been implemented properly and will sort Employee class on basis of name in ascending order.

Read : [**Comparable vs Comparator - differences**](http://www.javamadesoeasy.com/2015/04/comparable-vs-comparator-differences.html) for more.

**Collection interview Question 83.**

Collection **Output** interview question **33.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **class** Employee **implements** Comparator<Employee>{     String name;     String id;    **public** Employee() {}    **public** Employee(String name, String id) {  **this**.name = name;  **this**.id = id;     }       @Override  **public** **int** compare(Employee obj1, Employee obj2) {  **return** obj2.name.compareTo(obj1.name);     }     @Override  **public** String toString() {  **return** "{" + "name=" + name + ", id=" + id  + '}';     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ComparatorUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("sam","4");         Employee emp2=**new** Employee("amy","2");        ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list,**new** Employee());         System.*out*.println(list);     }  } |

**Answer.**

/\*OUTPUT

[{name=sam, id=4}, {name=amy, id=2}]

\*/

compare method of Comparator has been implemented properly and will sort Employee class on basis of name in descending order.

**Collection interview Question 84.**

Collection **Output** interview question **34.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **class** Employee{     String name;  **public** Employee() {}  **public** Employee(String name) {  **this**.name = name;     }  **public** String toString() {  **return** "name=" + name;     }  **static** **class** ComparatorName **implements** Comparator<Employee>{  **public** **int** compare(Employee obj1, Employee obj2) {  **return** obj1.name.compareTo(obj2.name);         }     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ComparatorUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("ankit");         Employee emp2=**new** Employee("brad");          ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list,**new** Employee.ComparatorName());         System.*out*.println(list);     }  } |

**Answer.**

/\*OUTPUT

[name=ankit, name=brad]

\*/

compare method of Comparator has been implemented properly by **static** **class** ComparatorName and will sort Employee class on basis of name in ascending order.

**Collection interview Question 85.**

Collection **Output** interview question **35.**

|  |
| --- |
| **import** java.util.Arrays;  **import** java.util.Comparator;  **class** Sort **implements** Comparator<Integer> {  **public** **int** compare(Integer o1, Integer o2) {  **return** o2.compareTo(o1);     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String...a){         Integer intArray[]={2,3,1};         Arrays.*sort*(intArray, **new** Sort());  **for**(**int** i: intArray){            System.*out*.print(i+" ");         }     }  } |

**Answer.**

/\*OUTPUT

3 2 1

\*/

In program, we sort Integer array by using Arrays.sort (we will define Comparator to sort elements in descending order)

*Read :* [***Arrays.sort to sort arrays by implementing Comparator and how Comparator of superclass can be used by subclasses***](http://www.javamadesoeasy.com/2015/04/arrayssort-to-sort-arrays-by.html)

Read:

[**COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50**](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

[**COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75**](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

**Collection interview Question 86.**

Collection **Output** interview question **36.**

|  |
| --- |
| **import** java.util.Comparator;  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Set<Integer> treeSet = **new** TreeSet<Integer>(**new** Comparator<Integer>() {  **public** **int** compareTo(Integer o1, Integer o2) {  **return** o2.compareTo(o1);                   }            });         treeSet.add(3);         treeSet.add(1);         treeSet.add(2);         System.*out*.println(treeSet);     }  } |

**Answer.**

/\*OUTPUT

compile time exception

\*/

We haven’t implemented compare method of Comparator. If compare would have been there in place of compareTo program would have compiled and executed properly, hence would have sorted elements of treeSet in reverse order..

**Read :** [**Sort Set by using TreeSet and by implementing Comparator and Comparable interface**](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

**Collection interview Question 87.**

Collection **Output** interview question **37.**

|  |
| --- |
| **import** java.util.Collection;  **import** java.util.HashSet;  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Collection<Integer> collection = **new** HashSet<Integer>();         collection.add(3);         collection.add(1);         collection.add(2);         Set<Integer> treeSet = **new** TreeSet<Integer>(collection);         System.*out*.println(treeSet);     }  } |

**Answer.**

/\*OUTPUT

[1, 2, 3]

\*/

*If elements are stored in stored in*[***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*/*[***ArrayList***](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *or any other class that implements*[***Collection***](http://www.javamadesoeasy.com/2015/04/collection-in-java.html)*, then we can use TreeSet’s* ***addAll*** *method or* ***constructor*** *for sorting****.***

***Read :*** [***Sort Set by using TreeSet and by implementing Comparator and Comparable interface in java***](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

**Collection interview Question 88.**

Collection **Output** interview question **38.**

|  |
| --- |
| **import** java.util.Collection;  **import** java.util.HashSet;  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Collection<Integer> collection = **new** HashSet<Integer>();         collection.add(3);         collection.add(1);         collection.add(2);         collection.add(**null**);         Set<Integer> treeSet = **new** TreeSet<Integer>();         treeSet.addAll(collection);         System.*out*.println(treeSet);     }  } |

**Answer.**

/\*OUTPUT

Runtime Exception - NullPointerException

\*/

*If elements are stored in stored in*[***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*, then we can use TreeSet’s* ***addAll*** *method for sorting****, but*** [*TreeSet*](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) *does not allow null.*

***Read :*** [***Sort Set by using TreeSet and by implementing Comparator and Comparable interface in java***](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

**Collection interview Question 89.**

Collection **Output** interview question **39.**

|  |
| --- |
| **import** java.util.Comparator;  **import** java.util.Map;  **import** java.util.TreeMap;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** SortMap {  **public** **static** **void** main(String...a){         Map<Integer, Integer> treeMap = **new** TreeMap<Integer, Integer>(**new** Comparator<Integer>(){  **public** **int** compare(Integer o1, Integer o2) {  **return** o2.compareTo(o1);                   }            });         treeMap.put(4, 1);         treeMap.put(2, 1);         treeMap.put(3, 1);           System.*out*.println(treeMap);     }  } |

**Answer.**

/\*OUTPUT

{4=1, 3=1, 2=1}

\*/

***TreeMap*** is sorted by natural order of keys, but we will implement Comparator interface to change the behaviour to sort TreeMap in descending order of keys.

Here, [Comparator interface has been implemented in form of anonymous inner class.](http://www.javamadesoeasy.com/2015/04/sort-map-by-key-in-ascending-and.html)

Read:

[**COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50**](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

[**COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75**](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

**Collection interview Question 90.**

Collection **Output** interview question **40.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **class** Sort **implements** Comparator<Map.Entry<Integer, Integer>>{     @Override  **public** **int** compare( Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2 ){  **return** (entry2.getValue()).compareTo( entry1.getValue() );     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String...a){         Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();         map.put(1, 2);         map.put(2, 1);         map.put(4, 8);           Set<Entry<Integer, Integer>> set = map.entrySet();         List<Entry<Integer, Integer>> list = **new** ArrayList<Entry<Integer, Integer>>(set);         Collections.*sort*(list, **new** Sort());  **for**(Map.Entry<Integer, Integer> entry:list)          System.*out*.print(entry.getKey());     }  } |

**Answer.**

/\*OUTPUT

412

\*/

**Read :** [**Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method**](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

**Collection interview Question 91.**

Collection **Output** interview question **41.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.LinkedHashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **class** Sort **implements** Comparator<Map.Entry<Integer, Integer>>{     @Override  **public** **int** compare( Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2 ){  **return** (entry2.getKey()).compareTo( entry1.getKey() );     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** SortMap {  **public** **static** **void** main(String...a){         Map<Integer, Integer> map = **new** LinkedHashMap<Integer, Integer>();         map.put(4, 1);         map.put(2, 6);         map.put(5, 1);           Set<Entry<Integer, Integer>> entrySet = map.entrySet();         List<Entry<Integer, Integer>> listOfentrySet = **new** ArrayList<Entry<Integer, Integer>>(entrySet);           Collections.*sort*(listOfentrySet, **new** Sort());    **for**(Map.Entry<Integer, Integer> entry:listOfentrySet)          System.*out*.print(entry.getKey());     }  } |

**Answer.**

/\*OUTPUT

542

\*/

**Read :** [**Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method**](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

**Collection interview Question 92.**

Collection **Output** interview question **42.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Number> numberList = **new** ArrayList<Number>();            numberList.add(2);            numberList.add(3);  *m*(numberList);     }  **static** **void** m(List<? **super** Double> l) {            System.*out*.print(l.get(0));            System.*out*.print(l.get(1));     }  } |

**Answer.**

/\*

23

\*/

**Collection interview Question 93.**

Collection **Output** interview question **43.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> l = **new** ArrayList<Integer>();            l.add(2);  *m*(l);       }  **static** **void** m(List<? **super** Double> l) {            System.*out*.println(l.get(0));            System.*out*.println(l.get(1));     }  } |

**Answer.**  Program won’t compile.

List<? **super** Double> can not accept List<Integer>, it can accept list of anySuperClassOfDouble i.e. **List<Number>** or **List<Object>**

**Collection interview Question 94.**

Collection **Output** interview question **44.**

|  |
| --- |
| **class** Abc {     <t> **void** display(t obj[]) {  **for** (t i : obj) {                   System.*out*.print(i + "  ");            }     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **class** MyClass {  **public** **static** **void** main(String... args) {            Abc o = **new** Abc();              Integer i[] = { 1, 2 };            o.display(i);            Double d[] = { 1.1, 2.2 };            o.display(d);     }  } |

**Answer.**

/\*

1  2  1.1  2.2

\*/

because t can of any type may be Integer or double

**Collection interview Question 95.**

Collection **Output** interview question **45.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(2);            list.add(3);            System.*out*.println(*sum*(list));     }  **public** **static** **double** sum(List<? **extends** Number> list) {  **double** sum = 0;  **for** (Number num : list) {                   sum += num.doubleValue();            }  **return** sum;     }  } |

**Answer.**

/\*

5.0

\*/

List<? **super** Number> can accept List<Integer>, it can accept list of anySubClassOfNumber i.e. **List<Double>, List<Float>,** etc.

**Collection interview Question 96.**

Collection **Output** interview question **46.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(2);            list.add(3);  *m*(list);     }  **public** **static** **void** m(List<Number> list) {            System.*out*.println(list);     }  } |

**Answer.**  Program won’t compile.

List<Number> cannot accept List<Integer>, to avoid compilation error we must use List<? **extends** Number>

**Collection interview Question 97.**

Collection **Output** interview question **47.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(1);            list.add(2);            System.*out*.println(*sum*(list));     }  **public** **static** **double** sum(List<? **extends** Number> list) {            list.add(4);  **double** sum = 0;  **for** (Number num : list) {                   sum += num.doubleValue();            }  **return** sum;     }  } |

**Answer.**  Program won’t compile.

List<? **extends** Number> cannot add or remove elements from list. So, list.add(4) will cause compilation error.

**Collection interview Question 98.  Output**

Collection **Output** interview question **question 48.**

|  |
| --- |
| **import** java.util.PriorityQueue;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            PriorityQueue<Integer> q = **new** PriorityQueue<Integer>();            q.add(1);            q.add(2);            q.add(3);            System.*out*.println(q.poll());            System.*out*.println(q.offer(4));            q.add(1);            q.remove(2);            System.*out*.println(q.peek());            System.*out*.println(q);     }  } |

**Answer.**

/\* OUTPUT

1

true

1

[1, 3, 4]