**Collection interview Question 51.**

Collection **Output** interview question **1.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            //INSET HERE              l.add("2");            l.add("21");     }  } |

**Insert the correct one from given 4 lines -**

List<String> l=new ArrayList<String>();

List<String> l=new ArrayList<Integer>();

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

List<Integer> l=new ArrayList<String>();

**Answer.**

List<String> l=new ArrayList<String>();

**Collection interview Question 52.**

Collection **Output** interview question **2.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            //INSET HERE              List<String> list=**new** ArrayList<String>();            list.add("a");            list.add("b");              l.add(list);     }  } |

**Insert the correct one from given 4 lines -**

List<List<String>> l=new ArrayList<List<String>>();

List<List> l=new ArrayList<List<String>>();

List<List<String>> l=new ArrayList<ArrayList<String>>();

List<List,String> l=new ArrayList<List,String>();

**Answer.**

List<List<String>> l=new ArrayList<List<String>>();

**Collection interview Question 53.**

Collection **Output** interview question **3.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ArrayListTest {  **public** **static** **void** main(String args[]) {            List<String> arrayList = **new** ArrayList<String>();            arrayList.add("a");            arrayList.add("b");            arrayList.add("c");            System.*out*.println();            arrayList.add(1,"d");            System.*out*.println(arrayList);       }  } |

**Answer.**

/\*OUTPUT

[a, d, b, c]

\*/

[ArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) maintains insertion order and element can be added at specific index as well.

**Collection interview Question 54.**

Collection **Output** interview question **4.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.ListIterator;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ArrayListTest {  **public** **static** **void** main(String args[]) {            List<String> arrayList = **new** ArrayList<String>();            arrayList.add("a");            arrayList.add("b");            ListIterator<String> listIterator = arrayList.listIterator();  **while** (listIterator.hasNext()) {                   System.*out*.println(listIterator.next());                   listIterator.previous();            }     }  } |

**Answer.**

/\*OUTPUT

a infinite times

\*/

[ArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) provides [listIterator](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html) for traversing in forward and backward direction, so program will compile and run infinitely.

**Collection interview Question 55.**

Collection **Output** interview question **5.**

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.Set;  **import** java.util.TreeSet;  **public** **class** ConcurrentSkipListMapTest {  **public** **static** **void** main(String args[]) {            Set set = **new** TreeSet();            set.add(1);            set.add("2");            set.add(3);            Iterator iterator = set.iterator();  **while** (iterator.hasNext()) {                   System.*out*.print(iterator.next() + " ");            }     }  } |

**Answer.**

/\*OUTPUT

java.lang.ClassCastException

\*/

TreeSet maintains natural order of elements, 1 is a Integer while "2" is a String, so they cannot be compared for sorted, hence [runtime exception](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) -  ClassCastException is thrown.

*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-3 > Q75- Q100***](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

**Collection interview Question 56.**

Collection **Output** interview question **6.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Iterator;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ArrayListTest {  **public** **static** **void** main(String args[]) {            List<String> arrayList = **new** ArrayList<String>();            arrayList.add("a");            arrayList.add("b");            Iterator<String> iterator = arrayList.iterator();  **while** (iterator.hasNext()) {                   System.*out*.println(iterator.next());                   arrayList.add("c");            }     }  } |

**Answer.**

/\*OUTPUT

a

Exception in thread "main" java.util.ConcurrentModificationException

   at java.util.ArrayList$Itr.checkForComodification(Unknown Source)

   at java.util.ArrayList$Itr.next(Unknown Source)

   at ArrayListTest.main(ArrayListTest.java:23)

\*/

[ConcurrentModificationException](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) is thrown because ArrayList cannot modified during iteration.

**Collection interview Question 57.**

Collection **Output** interview question **7.**

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.Map;  **import** java.util.concurrent.ConcurrentSkipListMap;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ConcurrentSkipListMapTest {  **public** **static** **void** main(String args[]) {            Map<Integer, String> concurrentSkipListMap = **new** ConcurrentSkipListMap<Integer, String>();            concurrentSkipListMap.put(11, "audi");            Iterator<Integer> keyIterator = concurrentSkipListMap.keySet()                         .iterator();  **while** (keyIterator.hasNext()) {                   System.*out*.println(keyIterator.next());                   concurrentSkipListMap.put(13, "bmw");            }     }  } |

**Answer.**

/\*OUTPUT

11

\*/

[ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/concurrentskiplistmap-iterator-on_24.html) can be modified during iteration- i.e. iteration is [fail-safe](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), but newly added keys won’t be a part of that iteration

**Collection interview Question 58.**

Collection **Output** interview question **8.**

|  |
| --- |
| **import** java.util.List;  **import** java.util.concurrent.CopyOnWriteArrayList;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** CopyOnWriteArrayListTest {  **public** **static** **void** main(String args[]) {            List<String> copyOnWriteArrayList = **new** CopyOnWriteArrayList<String>();            copyOnWriteArrayList.add("ind");            copyOnWriteArrayList.add("usa");            copyOnWriteArrayList.add(**null**);  **for** (String string : copyOnWriteArrayList) {                   System.*out*.print(string+" ");                   copyOnWriteArrayList.add("newEle3");            }     }  } |

**Answer.**

/\* OUTPUT

ind usa null

\*/

[CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) allows to add null, maintains insertion order and iteration is [fail-safe](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html).

**Collection interview Question 59.**

Collection **Output** interview question **9.**

|  |
| --- |
| **import** java.util.EnumMap;  **import** java.util.Map;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** EnumMapExample{  **public** **enum** Days{  *Monday*  ;     }  **public** **static** **void** main(String args[]) {           Map<Days, String> daysEnumMap = **new** EnumMap<Days, String>(Days.**class**);         daysEnumMap.put(Days.*Monday*, "Day1");         daysEnumMap.put(Days.Tuesday, "Day2");            System.*out*.println("daysEnumMap.get(Days.Monday) : " +                                              daysEnumMap.get(Days.*Monday*));            System.*out*.println("daysEnumMap.containsKey(Days.Monday) : " +                                              daysEnumMap.containsKey(Days.Tuesday));     }  } |

**Answer.**  Program will fail to compile because a [EnumMap](http://www.javamadesoeasy.com/2015/04/enummap-in-java-with-program.html) is specialized [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) implementation for use with enum type keys. In EnumMap all keys comes from a single enum type that is specified when the Map is created.

*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-3 > Q75- Q100***](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

**Collection interview Question 60.**

Collection **Output** interview question **10.**

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.Set;  **import** java.util.concurrent.CopyOnWriteArraySet;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** CopyOnWriteArraySetTest {  **public** **static** **void** main(String args[]) {            Set<String> copyOnWriteArraySet = **new** CopyOnWriteArraySet<String>();            copyOnWriteArraySet.add("a");            copyOnWriteArraySet.add("b");            Iterator<String> iterator = copyOnWriteArraySet.iterator();  **while** (iterator.hasNext()) {                   copyOnWriteArraySet.add("c");                   System.*out*.println(iterator.next());            }     }  } |

**Answer.**

/\*OUTPUT

a

b

\*/

[CopyOnWriteArraySet](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html) can be modified during iteration- i.e. iteration is [fail-safe](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), but newly added elements won’t be a part of that iteration, and insertion order is not guaranteed.

**Collection interview Question 61.**

Collection **Output** interview question **11.**

|  |
| --- |
| **import** java.util.LinkedHashMap;  **import** java.util.Map;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** LinkedHashMapTest {  **public** **static** **void** main(String args[]) {            Map<Integer, String> m = **new** LinkedHashMap<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.**

/\*OUTPUT

2

{11=bmw, null=fer}

\*/

[LinkedHashMap](http://www.javamadesoeasy.com/2015/02/linkedhashmap-custom-implementation.html) maintains insertion order of keys, and allows one null key and many null values.

**Must read:** [**Differences between HashMap, Hashtable, LinkedHashMap and TreeMap in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)

**Collection interview Question 62.**

Collection **Output** interview question **12.**

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.Map;  **import** java.util.concurrent.ConcurrentSkipListMap;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ConcurrentSkipListMapTest {  **public** **static** **void** main(String args[]){       Map<Integer,String> concurrentSkipListMap=**new** ConcurrentSkipListMap<Integer,String>();       concurrentSkipListMap.put(11, "audi");     concurrentSkipListMap.put(44, **null**);       Iterator<Integer> keyIterator = concurrentSkipListMap.keySet().iterator();  **while** (keyIterator.hasNext()) {            System.*out*.println(keyIterator.next());     }    }  } |

**Answer.**

/\*OUTPUT

java.lang.NullPointerException

\*/

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

**Must read:** [**Differences and Similarities between TreeMap and ConcurrentSkipListMap with program in java**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)

**Collection interview** Question 63.

Collection **Output** interview question **13.** How many **buckets** will be there and what will be **size of HashMap**?

|  |
| --- |
| **package** p1;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }     //no hashCode() method     //no equals() method       @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program1 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1, Employee[ name=b] =emp2}

HashMap's size> 3

null

\*/

**Buckets**= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

**Size**= As equals() method is not their, size will be **3**.

**get()**=we won’t be able to get object.

**Collection interview Question 64.**

Collection **Output** interview question **14. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p2;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }       @Override  **public** **int** hashCode(){  **return** (**this**.name==**null** ? 0: **this**.name.hashCode() );     }     @Override  **public** **boolean** equals(Object obj){            Employee emp=(Employee)obj;  **return** (emp.name==**this**.name || emp.name.equals(**this**.name));     }     @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program2 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN}

HashMap's size> 2

emp1 OVERRIDDEN

\*/

**Buckets**= As hashCode() method is overridden perfectly, **2** bucket locations will be used.

**Size**= As equals() method is their, size will be **2**,

value corresponding to Employee with id=1 and name=’sam’ was **employee1 data**

& was overridden by value **employee1 data** **OVERRIDDEN**

**get()**=we will be able to get object.

**Collection interview Question 65.**

Collection **Output** interview question **15. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p3;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }       @Override  **public** **int** hashCode(){  **return** 1;     }     @Override  **public** **boolean** equals(Object obj){  **return** **true**;     }     @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program3 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN}

HashMap's size> 1

emp1 OVERRIDDEN

\*/

**Buckets**= As hashCode() method returns 1, only **1** bucket location will be used.

**Size**= As equals() method always returns true, size will be **1**, all three employees will be stored on same bucket location in one Entry (new Entry will keep on overriding previous Entry). We will always get last stored key-value pair only.

**get()**=we will be able to get object.

**Collection interview Question 66.**

Collection **Output** interview question **16.**

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

**Answer.**

/\*OUTPUT

2

{null=null, 11=a}

\*/

[HashMap](http://www.javamadesoeasy.com/2015/02/hashmap-custom-implementation.html) does not maintains insertion order of keys, and allows one null key and many null values.

*Must read :* [***HashMap methods in java***](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)

**Collection interview Question 67.**

Collection **Output** interview question **17.**

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.Vector;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** VectorTest {  **public** **static** **void** main(String args[]) {            Vector<String> vector = **new** Vector<String>();            vector.add("1");            vector.add("2");            Iterator<String> iterator = vector.iterator();  **while** (iterator.hasNext()) {                   vector.add("3");                   System.*out*.println(iterator.next());            }     }  } |

**Answer.**  [**ConcurrentModificationException**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) will be thrown, because Iterator returned by [**Vector**](http://www.javamadesoeasy.com/2015/04/vector-add-add-element-at-specific.html) is fail-fast.

**Collection interview Question 68.**

Collection **Output** interview question **18.**

|  |
| --- |
| **import** java.util.Collections;  **import** java.util.HashMap;  **import** java.util.Map;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** HashMapTest {  **public** **static** **void** main(String args[]){              Map<Integer,String> hashMap=**new** HashMap<Integer,String>();            hashMap.put(11, "a");            Collections.*unmodifiableMap*(hashMap);            hashMap.put(12, "b");            System.*out*.println(hashMap);     }  } |

**Answer.**

/\*OUTPUT

{11=a, 12=b}

\*/

Although map has been made unmodifiable but we haven’t stored that reference anywhere, so we can continue to modify map. Read [making map unmodifiable using Collections.unmodifiableMap](http://www.javamadesoeasy.com/2015/04/hashmap-making-map-unmodifiable-using.html)

**Collection interview Question 69.**

Collection **Output** interview question **19.**

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

**Answer.**  [NullPointerException](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html) will be thrown, because hashtable does not allow to store null key or values.

*Must Read :* [***Collection - List, Set and Map all properties in tabular form***](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)*,*

[***HashMap and Hashtable - Similarity and Differences***](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-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-3 > Q75- Q100***](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

**Collection interview Question 70.**

Collection **Output** interview question **20. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p4;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }     @Override  **public** **int** hashCode(){  **return** 1;     }     //no equals() method       @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program4 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=b] =emp2, Employee[ name=a] =emp1}

HashMap's size> 3

null

\*/

**Buckets**= As hashCode() method returns 1, only **1** bucket location will be used.

**Size**= As equals() method doesn’t exist, size will be **3**, all three employees will be stored on same bucket location but in different Entry.

**get()**=we won’t be able to get object.

**Collection interview Question 72.**

Collection **Output** interview question **22. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p5;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }       //no hashCode() method       @Override  **public** **boolean** equals(Object obj){  **return** **true**;     }     @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program5 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1}

HashMap's size> 3

null

\*/

**Buckets**= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

**Size**= Though equals() method is their(but because of hashCode() method’s absence) **which always returns true**, we won’t be able to locate correct bucket location for calling equals() method, so, size will be **3**.

**get()**=we won’t be able to get object.

**Collection interview Question 72.**

Collection **Output** interview question **22. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p6;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }       @Override  **public** **int** hashCode(){  **return** (**this**.name==**null** ? 0: **this**.name.hashCode() );     }     //no equals() method         @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program6 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1}

HashMap's size> 3

null

\*/

**Buckets**= As hashCode() method is overridden perfectly, **2** bucket locations will be used.

**Size**= As equals() method is not their, size will be **3**,

**get()**=we won’t be able to get object.

**Collection interview Question 73.**

Collection **Output** interview question **23. How many buckets will be there and what will be size of HashMap?**

|  |
| --- |
| **package** p7;  **import** java.util.HashMap;  **class** Employee {    **private** String name;    **public** Employee(String name) { // constructor  **this**.name = name;     }       //no hashCode() method       @Override  **public** **boolean** equals(Object obj){            Employee emp=(Employee)obj;  **return** (emp.name==**this**.name || emp.name.equals(**this**.name));     }     @Override  **public** String toString() {  **return** "Employee[ name=" + name + "] ";     }  }  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** Program7 {  **public** **static** **void** main(String...a){              HashMap<Employee, String> hm=**new** HashMap<Employee, String>();            hm.put(**new** Employee("a"), "emp1");            hm.put(**new** Employee("b"), "emp2");            hm.put(**new** Employee("a"), "emp1 OVERRIDDEN");              System.*out*.println("HashMap's data> "+hm);            System.*out*.println("HashMap's size> "+hm.size());            System.*out*.println(hm.get(**new** Employee("a")));       }  } |

**Answer**.

/\*OUTPUT

HashMap's data> {Employee[ name=a] =emp1, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=b] =emp2}

HashMap's size> 3

null

\*/

**Buckets**= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

**Size**= Though equals() method is their(but because of hashCode() method’s absence), we won’t be able to locate correct bucket location for calling equals() method, so, size will be **3**.

**get()**=we won’t be able to get object.

**Collection interview Question 74.**

Collection **Output** interview question **24.**

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **public** **class** ArrayListTest {  **public** **static** **void** main(String args[]) {            List<String> arrayList = **new** ArrayList<String>();            arrayList.add("a");            arrayList.add("a");            arrayList.clear();            arrayList.add("b");            arrayList.add("b");            System.*out*.println(arrayList.size());       }  } |

**Answer.**

/\*OUTPUT

2

\*/

ArrayList allows duplicates, clear() method removes all elements from list.

**Collection interview Question 75.**

Collection **Output** interview question **25.**

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

**Answer.**

/\* OUTPUT

[null, 1, 1, null]

\*/

[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) does not store duplicates but >

“1” is a String, while 1 is Integer &

null is nothing, while “null” is a String

Also HashSet does not maintain insertion order and allows null.