**What is UnsupportedOperationException?**  
This exception is thrown to indicate that the requested operation is not supported.  
Example of UnsupportedOperationException:  
In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection cannot be modified. Hence, UnsupportedOperationException will be thrown.

**public** **class** UnModiFiableList

{

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

{

ArrayList list = **new** ArrayList();

list.add("1");

list.add("amit");

list.add("kumar");

list.add("ranhci");

Collection lis =Collections.*unmodifiableCollection*(list);

System.***out***.println(lis);

lis.add("xyz");

}

}  
 **Suppose there is an Employee class. We add Employee class objects to the ArrayList. Mention the steps need to be taken, if I want to sort the objects in ArrayList using the employeeId attribute present in Employee class.**  
a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)  
b. We will now call Collections class sort method and pass the list as argument , that is ,  
     Collections.sort(empList)    
 **package** collectionInterviewbased;

**import** java.util.ArrayList;

**import** java.util.Collections;

**public** **class** Employee **implements** Comparable<Employee>

{

**private** **int** empId;

**private** String empName;

**private** String empDep;

**private** String empAdd;

**public** Employee(**int** empId, String empName, String empDep, String empAdd)

{

**this**.empId = empId;

**this**.empName = empName;

**this**.empDep = empDep;

**this**.empAdd = empAdd;

}

**public** **int** getEmpId()

{

**return** empId;

}

**public** String getEmpName()

{

**return** empName;

}

**public** String getEmpDep()

{

**return** empDep;

}

**public** String getEmpAdd()

{

**return** empAdd;

}

@Override

**public** **int** compareTo(Employee empObj)

{

**if** (**this**.empId>empObj.getEmpId())

**return** 1;

**else** **if** (**this**.empId<empObj.getEmpId()) {

**return** -1;

}

**return** 0;

}

@Override

**public** String toString()

{

String out = **this**.getEmpId() + "---"+**this**.getEmpName()+"--- "+**this**.getEmpDep()+"---"+**this**.getEmpAdd();

**return** out;

}

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

{

Employee emp1 = **new** Employee(101, "amit", "it", "BANGALORE");

Employee emp2 = **new** Employee(100, "ajit", "cse", "JHARKHAND");

Employee emp3 = **new** Employee(99, "anit", "iit", "BIHAR");

Employee emp4 = **new** Employee(98, "ankit", "cse", "DELHI");

ArrayList<Employee> empList = **new** ArrayList<Employee>();

empList.add(emp1);

empList.add(emp2);

empList.add(emp3);

empList.add(emp4);

System.***out***.println("Befor sorting according to EmpId");

System.***out***.println(empList);

System.***out***.println("After sorting according to EmpId");

Collections.*sort*(empList);

System.***out***.println(empList);

System.***out***.println("====done====");

}

}

**How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)  
  
Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**How do you convert a given Collection to SynchronizedCollection ?**One line code :    Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.

**What is hash-collision in Hashtable ? How it was handled in Java?**  
  
In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.

**When to use ArrayList and when to use LinkedList in application?**  
ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .  
Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.

**Why Map interface does not extend the Collection interface in Java Collections Framework ?**  
  
One liner answer : **Map interface is not compatible with the Collection interface.**  
Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object  . In Collection interface add(Object o) has only one parameter.  
The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.

**Difference between HashMap and Hashtable in Java**

|  |  |  |
| --- | --- | --- |
|  | **HashMap** | **Hashtable** |
|  |  |  |
| Synchronized | No | Yes |
|  |  |  |
| Thread-Safe | No | Yes |
|  |  |  |
| Null Keys and Null values | One null key ,Any null values | Not permit null keys and values |
|  |  |  |
| Iterator type | Fail fast iterator | Fail safe iterator |
|  |  |  |
| Performance | Fast | Slow in comparison |
|  |  |  |
| Superclass and Legacy | AbstractMap , No | Dictionary , Yes |