|  |  |
| --- | --- |
| **1.** | is it possible to read the elements of ArrayList through for-each loop |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **2.** | is it possible to read the elements of ArrayList through Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **3.** | is it possible to read the elements of ArrayList through ListIterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **4.** | What is the return type of hasNext() method in the Iterator? |
| |  | | --- | | A.  double |  |  | | --- | | B.  int |  |  | | --- | | C.  boolean |  |  | | --- | | D.  String |  |  | | --- | | E.  Object | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **5.** | What is the return type of next() method in the Iterator? |
| |  | | --- | | A.  double |  |  | | --- | | B.  int |  |  | | --- | | C.  boolean |  |  | | --- | | D.  String |  |  | | --- | | E.  Object | | | |
| **Correct Answer: E** | | |
| **6.** | Which method is not available in the Iterator? |
| |  | | --- | | A.  hasNext() |  |  | | --- | | B.  next() |  |  | | --- | | C.  add() |  |  | | --- | | D.  forEachRemaining | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **7.** | Is it possible to read elements in the reverse order by using Iterator? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **8.** | Is it possible to read elements multiple times by using Iterator? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **9.** | Iterator is a class. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **10.** | | Iterator is a pointer not a container. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |  |
| --- | --- | --- |
| **11.** | Which exception occurs while adding an element into ArrayList after getting Iterator and before iterating. | |
| |  | | --- | | A.  ClassCastException |  |  | | --- | | B.  NullPointerException |  |  | | --- | | C.  ConcurrentModificationException |  |  | | --- | | D.  ArithmaticException | | | | |
| **Correct Answer: C** | | | |
| **12.** | Fail Fast Iterators are not allowing concurrent operations. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **13.** | Fail Safe Iterators are not allowing concurrent operations. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **14.** | ListIterator extending Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **15.** | What is the return type of hasPrevious() method in the Iterator? |
| |  | | --- | | A.  double |  |  | | --- | | B.  int |  |  | | --- | | C.  boolean |  |  | | --- | | D.  String |  |  | | --- | | E.  Object | | | |
| **Correct Answer: C** | | |
| **16.** | What is the return type of previous() method in the Iterator? |
| |  | | --- | | A.  double |  |  | | --- | | B.  int |  |  | | --- | | C.  boolean |  |  | | --- | | D.  String |  |  | | --- | | E.  Object | | | |
| **Correct Answer: E** | | |

|  |  |
| --- | --- |
| **17.** | Which method is not available in the Iterator? |
| |  | | --- | | A.  hasNext() |  |  | | --- | | B.  next() |  |  | | --- | | C.  add() |  |  | | --- | | D.  forEachRemaining |  |  | | --- | | E.  size() | | | |
| **Correct Answer: E** | | |
| **18.** | Is it possible to read elements in the reverse order by using ListIterator? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **19.** | Is it possible to read elements multiple times by using ListIterator? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **20.** | ListIterator is an interface. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **21.** | ListIterator is a container not a pointer. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **22.** | Which exception occurs while adding an element into ArrayList after getting ListIterator and before iterating. | |
| |  | | --- | | A.  ClassCastException |  |  | | --- | | B.  NullPointerException |  |  | | --- | | C.  ConcurrentModificationException |  |  | | --- | | D.  ArithmaticException | | | | |
| **Correct Answer: C** | | | |

|  |  |
| --- | --- |
| **23.** | which method is in only ListIterator but not in Iterator |
| |  | | --- | | A.  remove() |  |  | | --- | | B.  hasNext() |  |  | | --- | | C.  next() |  |  | | --- | | D.  set() | | | |
| **Correct Answer: D** | | |
| **24.** | Enumeration is only for ? |
| |  | | --- | | A.  reading |  |  | | --- | | B.  reading and remove |  |  | | --- | | C.  reading, adding and remove | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **25.** | Entry is a combination of key and value in a Map |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **26.** | What is the return type of entrySet() in HashMap |
| |  | | --- | | A.  Entry |  |  | | --- | | B.  Set |  |  | | --- | | C.  Key |  |  | | --- | | D.  Value | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **27.** | Which method is not from Entry? |
| |  | | --- | | A.  getKey() |  |  | | --- | | B.  getValue() |  |  | | --- | | C.  getEntryKey() | | | |
| **Correct Answer: C** | | |
| **28.** | there is a compare(Object o1, Object o2) method in Integer wrapper class |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **29.** | what is the default sort in TreeMap |
| |  | | --- | | A.  based on values |  |  | | --- | | B.  based on keys | | | |
| **Correct Answer: B** | | |
| **30.** | Which is the method from Comparator? |
| |  | | --- | | A.  compareTo(Object o1) |  |  | | --- | | B.  compare(Object o1, Object o2) | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **31.** | which class is used to sort ArrayList content |
| |  | | --- | | A.  TreeList |  |  | | --- | | B.  Collections | | | |
| **Correct Answer: B** | | |
| **32.** | How to supply Comparator to the TreeSet |
| |  | | --- | | A.  to the constructor |  |  | | --- | | B.  to the addAll method | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **33.** | which one is proper while implementing compare method to achive sorting based on values? | |
| |  | | --- | | A.  ((Integer)o1.getValue()).compareTo((Integer)o2.getValue()) |  |  | | --- | | B.  ((Integer)o1.getValue()).compare((Integer)o2.getValue()) | | | | |
| **Correct Answer: A** | | | |
| **34.** | Which type is not a Collection type |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | | |
| **Correct Answer: D** | | | |

|  |  |
| --- | --- |
| **35.** | List is a type of Collection |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **36.** | Set is a type of Collection |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **37.** | Queue is a type of Collection |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **38.** | Map is a type of Collection |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **39.** | Which is maintaining oreader of addition. |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: A** | | |
| **40.** | Which is maintaining unique elements |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **41.** | where elements are adding with a key |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: D** | | |
| **42.** | Which can be best to maintain a stack |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **43.** | ArrayList comes under? |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: A** | | |
| **44.** | LinedList mainly comes under? |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **45.** | Vector mainly comes under? |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: A** | | |
| **46.** | Which is alo a Queue |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList |  |  | | --- | | C.  Vector | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **47.** | Which is synchronized one? |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList |  |  | | --- | | C.  Vector | | | |
| **Correct Answer: C** | | |
| **48.** | Which is the best container while reading elements frequently.? |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **49.** | Which is the best container while inserting elements frequently.? |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList | | | |
| **Correct Answer: B** | | |
| **50.** | Collections class is used to sort elements of List type. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **51.** | Every element is storing with an index inside List type containers. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **52.** | every element getting an index automatically under List type |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **53.** | Every element is storing with a key inside List type containers. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **54.** | Every element is storing with an object association inside List type containers. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **55.** | while reading an element from a list type container, we should supply an int value to the get method. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **56.** | while reading an element from a list type container, we should supply string value to the get method. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **57.** | which class is also queue type from the list type. |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList | | | |
| **Correct Answer: B** | | |
| **58.** | This method retrieves, but does not remove, the head (first element) of this queue. |
| |  | | --- | | A.  peek() |  |  | | --- | | B.  poll() |  |  | | --- | | C.  get() | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **59.** | This method retrieves, and also remove, the head (first element) of this queue. |
| |  | | --- | | A.  peek() |  |  | | --- | | B.  poll() |  |  | | --- | | C.  get() | | | |
| **Correct Answer: B** | | |
| **60.** | PriorityQueue is a default sorted one. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **61.** | PriorityQueue is a default sorted one. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **62.** | get method is not available in the PriorityQueue |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **63.** | Inside a PriorityQueue all the elements will be in the sorted manner. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **64.** | Inside a PriorityQueue only head elements is guarenteed in the sorted manner. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **65.** | which one not allowing duplicates? |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: B** | | |
| **66.** | which are the Set type classes |
| |  | | --- | | A.  HashSet |  |  | | --- | | B.  LinkedHashSet |  |  | | --- | | C.  ArrayList |  |  | | --- | | D.  HashMap | | | |
| **Correct Answer: A,B** | | |

|  |  |
| --- | --- |
| **67.** | which methods of element class executing to identikfy duplicates? |
| |  | | --- | | A.  toString() |  |  | | --- | | B.  hashCode() |  |  | | --- | | C.  equals() |  |  | | --- | | D.  compareTo() | | | |
| **Correct Answer: B,C** | | |
| **68.** | which method is executing first for avoiding duplicates? |
| |  | | --- | | A.  toString() |  |  | | --- | | B.  hashCode() |  |  | | --- | | C.  equals() |  |  | | --- | | D.  compareTo() | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **69.** | which method is giving perfect results incase of ineqality and may not be perfect incase of equality | |
| |  | | --- | | A.  toString() |  |  | | --- | | B.  hashCode() |  |  | | --- | | C.  equals() |  |  | | --- | | D.  compareTo() | | | | |
| **Correct Answer: B** | | | |
| **70.** | which method is giving perfect results incase of ineqality and also equality |
| |  | | --- | | A.  toString() |  |  | | --- | | B.  hashCode() |  |  | | --- | | C.  equals() |  |  | | --- | | D.  compareTo() | | | | |
| **Correct Answer: C** | | | |

|  |  |  |
| --- | --- | --- |
| **71.** | if two elements hasCode value same, then which method should execute to identify whether both are same or different | |
| |  | | --- | | A.  toString() |  |  | | --- | | B.  hashCode() |  |  | | --- | | C.  equals() |  |  | | --- | | D.  compareTo() | | | | |
| **Correct Answer: C** | | | |
| **72.** | hash bucket containing elements which are having same hashCode value. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **73.** | Which container used to store elements under one hash bucket |
| |  | | --- | | A.  HashSet |  |  | | --- | | B.  LinkedHashSet |  |  | | --- | | C.  ArrayList |  |  | | --- | | D.  HashMap |  |  | | --- | | E.  LinkedList | | | |
| **Correct Answer: E** | | |
| **74.** | elements under one hash bucket are storing in the order wise |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **75.** | elements under one hash bucket are storing in the randum order |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **76.** | equals() method is calling on elements of same bucket one by one in the order wise |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **77.** | equals() method is calling on elements of same bucket one by one in the randum order |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **78.** | which class is maintaining order of addition under Set |
| |  | | --- | | A.  HashSet |  |  | | --- | | B.  LinkedHashSet |  |  | | --- | | C.  ArrayList |  |  | | --- | | D.  HashMap |  |  | | --- | | E.  LinkedList | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **79.** | in which containers, element is adding with a key? |
| |  | | --- | | A.  List |  |  | | --- | | B.  Set |  |  | | --- | | C.  Queue |  |  | | --- | | D.  Map | | | |
| **Correct Answer: D** | | |
| **80.** | Under Map, how many arguments required to put method |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  4 | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **81.** | what is first argument for put method under Map |
| |  | | --- | | A.  key |  |  | | --- | | B.  value or element | | | |
| **Correct Answer: A** | | |
| **82.** | what is 2nd argument for put method under Map |
| |  | | --- | | A.  key |  |  | | --- | | B.  value or element | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **83.** | what is the argument for get method under Map |
| |  | | --- | | A.  key |  |  | | --- | | B.  value or element | | | |
| **Correct Answer: A** | | |
| **84.** | what is returning by get method under Map |
| |  | | --- | | A.  key |  |  | | --- | | B.  value or element | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **85.** | What returns get method under map while supplying a key which is not associated to any element | |
| |  | | --- | | A.  0 |  |  | | --- | | B.  0.0 |  |  | | --- | | C.  null |  |  | | --- | | D.  false | | | | |
| **Correct Answer: C** | | | |
| **86.** | key can be any type under Map. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **87.** | value/element can be any type under Map. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **88.** | key always String type under Map. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **89.** | how to read all keys of Map? |
| |  | | --- | | A.  getKeys() |  |  | | --- | | B.  keySet() |  |  | | --- | | C.  readKeys() | | | |
| **Correct Answer: B** | | |
| **90.** | how to read all Entry objects of Map? |
| |  | | --- | | A.  entrySet() |  |  | | --- | | B.  getEntries() |  |  | | --- | | C.  readEntries() | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **91.** | how to read key from the Entry object under Map? |
| |  | | --- | | A.  readKey() |  |  | | --- | | B.  getKey() |  |  | | --- | | C.  getKey() | | | |
| **Correct Answer: B** | | |
| **92.** | how to read value from the Entry object under Map? |
| |  | | --- | | A.  readValue() |  |  | | --- | | B.  getValue() |  |  | | --- | | C.  get() | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **93.** | which one is synchronized under Map |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  LinkedHashMap | | | |
| **Correct Answer: B** | | |
| **94.** | which one is maintaining order of addition under Map |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  LinkedHashMap | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **95.** | which one allows null as a key |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable | | | |
| **Correct Answer: A** | | |
| **96.** | which one allows null as a value/element |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **97.** | Unser Map duplicate keys are allowed |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **98.** | Unser Map duplicate values/elements are allowed |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **99.** | Iterator is an interface. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **100.** | | Iterator is a container and not a reference to ArrayList object |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **101.** | we can iterate elements any number of times by using one Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **102.** | we can't iterate elements in the reverse order by using Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **103.** | we can remove iterating element while iterating through Iterator. This remove will remove element from an ArrayList itself. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **104.** | ListIterator extending Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **105.** | ListIterator is not a container and just a reference to ArrayList object |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **106.** | we can iterate elements any number of times by using one Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **107.** | we can't iterate elements in the reverse order by using ListIterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **108.** | we can remove iterating element while iterating through ListIterator. This remove will remove element from an ArrayList itself. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **109.** | we can add element into ArrayList through ListIterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **110.** | we can add element into ArrayList through Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **111.** | we can replace element from an ArrayList through Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **112.** | we can replace element from an ArrayList through ListIterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **113.** | we can find next element index through ListIterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **114.** | we can find next element index through Iterator |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **115.** | by default Iterator is a fail safe one |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **116.** | by default Iterator is a fail fast one |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **117.** | by default ListIterator is a fail safe one |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **118.** | by default ListIterator is a fail fast one |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **119.** | is it possible to add any element into ArrayList after getting Iterator and before iterating. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **120.** | is it possible to add any element into ArrayList after getting ListIterator and before iterating. (using add method of ArrayList) | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |  |
| --- | --- | --- |
| **121.** | is it possible to add any element into ArrayList after getting ListIterator and before iterating. (using add method of ListIterator) | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **122.** | Which exception occures in case of fail fast iterators. |
| |  | | --- | | A.  NullPinterException |  |  | | --- | | B.  CuncurrentModificationException |  |  | | --- | | C.  NumberFormatException | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **123.** | Which class is used to sort elements from List type of containers. |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  LinkedList |  |  | | --- | | C.  Vector |  |  | | --- | | D.  Collection |  |  | | --- | | E.  Collections | | | |
| **Correct Answer: E** | | |
| **124.** | Which class is used to sort elements from Set type of containers. |
| |  | | --- | | A.  HashSet |  |  | | --- | | B.  LinkedHashSet |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  Collection |  |  | | --- | | E.  Collections | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **125.** | Which class is used to sort entries from Map type of containers. |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  LinkedHashMap |  |  | | --- | | C.  TreeMap |  |  | | --- | | D.  Hashtable |  |  | | --- | | E.  Collections | | | |
| **Correct Answer: C** | | |
| **126.** | Which class is used to sort elements from Queue type of containers. |
| |  | | --- | | A.  LinkedList |  |  | | --- | | B.  PriorityQueue | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **127.** | Is it required all elements should be same type in a container while sorting all elements from a container? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **128.** | Is it required all elements should be not null in a container while sorting all elements from a container? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **129.** | Is it required all elements should be type of Comparable in a container while sorting all elements from a container and there is no separate Comparator supplying? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **130.** | Is it required all elements should be type of Comparable in a container while sorting all elements from a container and there is a separate Comparator supplying? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **131.** | Comparable is an interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **132.** | Comparator is an interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **133.** | Which method available in Comparable? |
| |  | | --- | | A.  compareTo(Object obj) |  |  | | --- | | B.  compare(Object obj1, Object obj2) | | | |
| **Correct Answer: A** | | |
| **134.** | Which method available in Comparator? |
| |  | | --- | | A.  compareTo(Object obj) |  |  | | --- | | B.  compare(Object obj1, Object obj2) | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **135.** | Comparable is a marker interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **136.** | Comparable is a functional interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **137.** | Comparator is a marker interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **138.** | Comparator is a functional interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **139.** | If class is with a single field, then wich interface is most suitable to achieve sorting? |
| |  | | --- | | A.  Comparable |  |  | | --- | | B.  Comparator | | | |
| **Correct Answer: A** | | |
| **140.** | If class is with multiple fields, then wich interface is most suitable to achieve sorting? |
| |  | | --- | | A.  Comparable |  |  | | --- | | B.  Comparator | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **141.** | wich interface implemented by wrapper classes. |
| |  | | --- | | A.  Comparable |  |  | | --- | | B.  Comparator | | | |
| **Correct Answer: A** | | |
| **142.** | wich interface implemented by String class. |
| |  | | --- | | A.  Comparable |  |  | | --- | | B.  Comparator | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **143.** | How to supply Comparator type to Collections's sort method? |
| |  | | --- | | A.  as first arg |  |  | | --- | | B.  as a 2nd arg | | | |
| **Correct Answer: B** | | |
| **144.** | Is it possible to sort elements of a container which contains null value? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **145.** | Is it possible to sort elements of a container which contains different type of elements? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **146.** | How to supply Comparator type to PriorityQueue? |
| |  | | --- | | A.  as an arg to add method |  |  | | --- | | B.  as an arg to constructor | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **147.** | Which class is used to sort elements from Set type of containers. |
| |  | | --- | | A.  HashSet |  |  | | --- | | B.  LinkedHashSet |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  Collection |  |  | | --- | | E.  Collections | | | |
| **Correct Answer: C** | | |
| **148.** | Which class is used to sort entries from Map type of containers. |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  LinkedHashMap |  |  | | --- | | C.  TreeMap |  |  | | --- | | D.  Hashtable |  |  | | --- | | E.  Collections | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **149.** | How to supply Comparator to TreeSet to achive sorting. |
| |  | | --- | | A.  an arg to sort method |  |  | | --- | | B.  an arg to constructor | | | |
| **Correct Answer: B** | | |
| **150.** | How to supply Comparator to TreeMap to achive sorting. |
| |  | | --- | | A.  an arg to sort method |  |  | | --- | | B.  an arg to constructor | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **151.** | Based on which quantity sorting is happening in TreeMap? |
| |  | | --- | | A.  key |  |  | | --- | | B.  value or element | | | |
| **Correct Answer: A** | | |
| **152.** | What is the condition to be followed if TreeMap is not taking Comparator? |
| |  | | --- | | A.  key should be a Comparable type |  |  | | --- | | B.  value or element should be a Comparable type | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **153.** | how to read all keys of Map? |
| |  | | --- | | A.  getKeys() |  |  | | --- | | B.  keySet() |  |  | | --- | | C.  readKeys() | | | |
| **Correct Answer: B** | | |
| **154.** | how to read all Entry objects of Map? |
| |  | | --- | | A.  entrySet() |  |  | | --- | | B.  getEntries() |  |  | | --- | | C.  readEntries() | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **155.** | how to read key from the Entry object under Map? |
| |  | | --- | | A.  readKey() |  |  | | --- | | B.  getKey() |  |  | | --- | | C.  getKey() | | | |
| **Correct Answer: B** | | |
| **156.** | how to read value from the Entry object under Map? |
| |  | | --- | | A.  readValue() |  |  | | --- | | B.  getValue() |  |  | | --- | | C.  get() | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **157.** | in which class, synchronizedSet() available? |
| |  | | --- | | A.  SortedSet |  |  | | --- | | B.  TreeSet |  |  | | --- | | C.  LinkedHashSet |  |  | | --- | | D.  Collections | | | |
| **Correct Answer: D** | | |
| **158.** | Which class is used to make List as a synchronized one. |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  TreeSet |  |  | | --- | | C.  LinkedList |  |  | | --- | | D.  Collections | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **159.** | How to make HashMap as a synhronized one? |
| |  | | --- | | A.  HashMap.synchronizedMap(); |  |  | | --- | | B.  Collections.synchronizedMap(Map map); | | | |
| **Correct Answer: B** | | |
| **160.** | package app1;  import java.util.ArrayList;  public class M1 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(909);  list.add("abc");  list.add(true);  list.add('a');  System.out.println(list);  }  } |
| |  | | --- | | A.  [909, abc, a, true] |  |  | | --- | | B.  [909, abc, true, a] |  |  | | --- | | C.  Compilation error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **161.** | package app1;  import java.util.ArrayList;  public class M2 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(909);  list.add(909);  list.add("abc");  list.add(true);  list.add('a');  System.out.println(list);  list.add(1, "vijay");  System.out.println(list);  }  } |
| |  | | --- | | A.  [909, 909, abc, true, a]  [909, vijay, 909, abc, true, a] |  |  | | --- | | B.  [909, 909, abc, true, a]  [909, vijay, abc, true, a] |  |  | | --- | | C.  Compilation error | | | |
| **Correct Answer: A** | | |
| **162.** | package app1;  import java.util.ArrayList;  public class M3 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(909);  list1.add(909);  list1.add("abc");  list1.add(true);  list1.add('a');  System.out.println(list1);  list1.set(1, "vijay");  System.out.println(list1);  }  } |
| |  | | --- | | A.  [909, 909, abc, true, a]  [909, vijay, 909, abc, true, a] |  |  | | --- | | B.  [909, 909, abc, true, a]  [909, vijay, abc, true, a] |  |  | | --- | | C.  Compilation error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **163.** | package app1;  import java.util.ArrayList;  public class M4 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(909);  list1.add(909);  list1.add("abc");  list1.add(true);  list1.add('a');  System.out.println(list1);  System.out.println(list1.contains(true));  System.out.println(list1.contains(90000));  }  } |
| |  | | --- | | A.  [909, 909, abc, true, a]  false  false |  |  | | --- | | B.  [909, 909, abc, true, a]  true  true |  |  | | --- | | C.  [909, 909, abc, true, a]  true  false | | | |
| **Correct Answer: C** | | |
| **164.** | package app1;  import java.util.ArrayList;  public class M5 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  for(String s1 : args) {  list1.add(s1);  }  System.out.println(list1);  }  }  // If you pass the command line arguments abc xyz xyz 123 500 500  what will be the output |
| |  | | --- | | A.  abc xyz xyz 123 500 500 |  |  | | --- | | B.  abc xyz 123 500 |  |  | | --- | | C.  [] | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **165.** | package app1;  import java.util.ArrayList;  public class M6 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  for(String s1 : args) {  if(! list1.contains(s1))  list1.add(s1);  }  System.out.println(list1);  }  }  // If you pass the command line arguments abc xyz xyz 123 500 500  what will be the output |
| |  | | --- | | A.  abc xyz xyz 123 500 500 |  |  | | --- | | B.  abc xyz 123 500 |  |  | | --- | | C.  [] | | | |
| **Correct Answer: B** | | |
| **166.** | package app1;  import java.util.ArrayList;  import java.util.Scanner;  public class M7 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  Scanner sc = new Scanner(System.in);  do  {  System.out.println("Enter an element");  String s1 = sc.next();  if(! list1.contains(s1))  list1.add(s1);  System.out.println("You want to add one more (Y/N)?");  }while("Y".equalsIgnoreCase(sc.next()));  System.out.println(list1);  }  }  // If you give the inupt as  abc  xyz  xyz  123  500  500  what will be the output |
| |  | | --- | | A.  abc xyz xyz 123 500 500 |  |  | | --- | | B.  abc xyz 123 500 |  |  | | --- | | C.  [] | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **167.** | package app1;  import java.util.ArrayList;  import java.util.Scanner;  public class M8 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  ArrayList list2 = new ArrayList(list1);  System.out.println(list2);  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190, 910, 901, 290] | | | |
| **Correct Answer: B** | | |
| **168.** | package app1;  import java.util.ArrayList;  public class M9 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  ArrayList list2 = new ArrayList();  list2.add("abc");  list2.add(90000);  list2.addAll(list1);  System.out.println(list2);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [ 90, 190, 910, 901, 290, abc, 90000] |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [abc, 90000, 90, 190, 910, 901, 290] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **169.** | package app1;  import java.util.ArrayList;  public class M11 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  Object obj = list1.get(3);  System.out.println(obj);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  901 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  910 |  |  | | --- | | C.  [90, 190, 910, 901, 290]  IndexOutOfBoundsException | | | |
| **Correct Answer: A** | | |
| **170.** | package app1;  import java.util.ArrayList;  public class M12 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  int i = (Integer) list1.get(3);  System.out.println(i);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  901 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  910 |  |  | | --- | | C.  [90, 190, 910, 901, 290]  IndexOutOfBoundsException | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **171.** | package app1;  import java.util.ArrayList;  public class M13 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  int i = list1.size();  System.out.println(i);  }  } |
| |  | | --- | | A.  4 |  |  | | --- | | B.  5 | | | |
| **Correct Answer: B** | | |
| **172.** | package app1;  import java.util.ArrayList;  public class M14 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  int j = list1.size();  for(int i = 0; i < list1.size(); i++) {  System.out.println(list1.get(i));  }  System.out.println(j);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  90  190  910  901  290  5 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  90 190 910 901 290  5 |  |  | | --- | | C.  5 | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **173.** | package app1;  import java.util.ArrayList;  public class M15 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  int j = list1.size();  for(Object obj : list1) {  System.out.print(obj + ", ");  }  System.out.println();  System.out.println(j);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  90  190  910  901  290  5 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  90 190 910 901 290  5 |  |  | | --- | | C.  5 | | | |
| **Correct Answer: B** | | |
| **174.** | package app1;  import java.util.ArrayList;  import java.util.Iterator;  public class M16 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  Iterator it = list1.iterator();  while(it.hasNext()) {  System.out.println(it.next());  }  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  90  190  910  901  290 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  90 190 910 901 290 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **175.** | package app1;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.ListIterator;  public class M17 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  ListIterator it = list1.listIterator();  while(it.hasNext()) {  System.out.println(it.next());  }  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  90  190  910  901  290 |  |  | | --- | | B.  [90, 190, 910, 901, 290]  90 190 910 901 290 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **176.** | package app1;  import java.util.ArrayList;  import java.util.ListIterator;  public class M18 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  Object obj = list1.remove(2);  System.out.println(list1);  System.out.println(obj);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 190, 901, 290] |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190, 901, 290]  910 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **177.** | package app1;  import java.util.ArrayList;  public class M19 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  boolean b1 = list1.remove(new Integer (901));  System.out.println(list1);  System.out.println(b1);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 190, 910, 290]  true |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190, 910, 290] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **178.** | package app1;  import java.util.ArrayList;  public class M20 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  boolean b1 = list1.remove(new Integer (9000000));  System.out.println(list1);  System.out.println(b1);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 190, 910, 901, 290]  true |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190, 910, 901, 290]  false |  |  | | --- | | C.  [90, 190, 910, 901, 290]  [90, 190, 910, 901, 290] | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **179.** | package app1;  import java.util.ArrayList;  public class M21 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);    ArrayList list2 = new ArrayList();  list2.add(90);  list2.add(190);  System.out.println(list2);  System.out.println("-----------------------");    list1.removeAll(list2);    System.out.println(list1);  System.out.println(list2);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  [910, 901, 290]  [90, 190] |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  [90, 190, 910, 901, 290]  [] |  |  | | --- | | C.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  []  [] | | | |
| **Correct Answer: A** | | |
| **180.** | package app1;  import java.util.ArrayList;  public class M22 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);    ArrayList list2 = new ArrayList();  list2.add(90);  list2.add(190);  System.out.println(list2);  System.out.println("-----------------------");    list1.retainAll(list2);    System.out.println(list1);  System.out.println(list2);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  [90, 190, 910, 901, 290]  [90, 190] |  |  | | --- | | B.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  [910, 901, 290]  [90, 190] |  |  | | --- | | C.  [90, 190, 910, 901, 290]  [90, 190]  -----------------------  [90, 190]  [90, 190] | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **181.** | package app1;  import java.util.ArrayList;  public class M23 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  list1.removeRange(1, 3);  System.out.println(list1);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 901, 290] |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  Run time exception | | | |
| **Correct Answer: B** | | |
| **182.** | package app1;  import java.util.ArrayList;  public class M23 extends ArrayList {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  M23 list1 = new M23();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  System.out.println(list1);  list1.removeRange(1, 3);  System.out.println(list1);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290]  [90, 901, 290] |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  Run time exception | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **183.** | package app1;  import java.util.ArrayList;  import java.util.Collections;  public class M24 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  list1.add(500);  list1.add(140);  list1.add(400);  System.out.println(list1);  Collections.sort(list1);  System.out.println(list1);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290, 500, 140, 400]  [90, 140, 190, 290, 400, 500, 901, 910] |  |  | | --- | | B.  [90, 190, 910, 901, 290, 500, 140, 400]  [910, 901, 500, 400, 290, 190, 140, 90] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **184.** | package app1;  import java.util.ArrayList;  import java.util.Collections;  public class M25 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  ArrayList list1 = new ArrayList();  list1.add(90);  list1.add(190);  list1.add(910);  list1.add(901);  list1.add(290);  list1.add(500);  list1.add(140);  list1.add(400);  System.out.println(list1);  Collections.sort(list1);  System.out.println(list1);  int i = Collections.binarySearch(list1, 910);  System.out.println(i);  }  } |
| |  | | --- | | A.  [90, 190, 910, 901, 290, 500, 140, 400]  [90, 140, 190, 290, 400, 500, 901, 910]  910 |  |  | | --- | | B.  [90, 190, 910, 901, 290, 500, 140, 400]  [90, 140, 190, 290, 400, 500, 901, 910]  7 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **185.** | entire collection api is available in |
| |  | | --- | | A.  java.util |  |  | | --- | | B.  java.lang |  |  | | --- | | C.  java.awt | | | |
| **Correct Answer: A** | | |
| **186.** | Collection objects are fixed in size |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **187.** | Arrays are fixed in size |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |
| **188.** | In collection api we can add any type of elements |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **189.** | Collection api allows to store only objects |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |
| **190.** | Is it possible to store Wrapper class objects |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **191.** | By default most of the collections accept duplicates |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |
| **192.** | Is it possible to copy one ArrayList elements into another ArrayList elements |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **193.** | By using which initializer you will copy one ArrayList elements into another ArrayList | |
| |  | | --- | | A.  SIB |  |  | | --- | | B.  constructor |  |  | | --- | | C.  IIB | | | | |
| **Correct Answer: B** | | | |
| **194.** | Is it possible to read collection objects through iterator |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **195.** | Is it possible to use removeRange() method stright away |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: B** | | |
| **196.** | In which collection type order of elements is preserved |
| |  | | --- | | A.  List |  |  | | --- | | B.  Queue |  |  | | --- | | C.  Map | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **197.** | Which collection type does not allows duplicates |
| |  | | --- | | A.  List |  |  | | --- | | B.  Queue |  |  | | --- | | C.  Map |  |  | | --- | | D.  Set | | | |
| **Correct Answer: D** | | |
| **198.** | In order to maintain elements in Queue or stack format which collection type is used |
| |  | | --- | | A.  List |  |  | | --- | | B.  Queue |  |  | | --- | | C.  Map |  |  | | --- | | D.  Set | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **199.** | In which collection type elements having both key value pair |
| |  | | --- | | A.  List |  |  | | --- | | B.  Queue |  |  | | --- | | C.  Map |  |  | | --- | | D.  Set | | | |
| **Correct Answer: C** | | |
| **200.** | In which collection type elements stored without index or key |
| |  | | --- | | A.  List |  |  | | --- | | B.  Queue |  |  | | --- | | C.  Map |  |  | | --- | | D.  Set | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **201.** | Collection is an |
| |  | | --- | | A.  class |  |  | | --- | | B.  interface | | | |
| **Correct Answer: B** | | |
| **202.** | Which class acts as both List and Queue types |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **203.** | Which class can be used in multi threaded environment |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: B** | | |
| **204.** | which class is synchronized |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **205.** | Which class is very much effective in case of reading operation |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: A** | | |
| **206.** | ArrayList internally using arrays |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **207.** | Which class is more preferable for in case of more frequently reading operation |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: A** | | |
| **208.** | Which class is not much effective in case of a reading operation |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **209.** | In which class insertion operation is not advisable |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: A** | | |
| **210.** | In which class insertion operations are more effective |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **211.** | Which class having dual features |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: C** | | |
| **212.** | Which class is a legacy class |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **213.** | which class is non synchronized by default |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector | | | |
| **Correct Answer: A** | | |
| **214.** | Which class is used for sorting purpose |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  LinkedList |  |  | | --- | | C.  Collections | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **215.** | package queue;  import java.util.LinkedList;  public class M1 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.removeFirst());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1190, 1910, 1901, 1290] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  [100, 1190, 1910, 1901, 1290] | | | |
| **Correct Answer: A** | | |
| **216.** | package queue;  import java.util.LinkedList;  public class M2 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.removeLast());  System.out.println(list);  System.out.println(list.removeLast());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  [90, 100, 1190, 1910, 1901]  [90, 100, 1190, 1910] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  1290  [90, 100, 1190, 1910, 1901]  1901  [90, 100, 1190, 1910] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **217.** | package queue;  import java.util.LinkedList;  public class M3 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.peek());  System.out.println(list);  System.out.println(list.peek());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  [90, 100, 1190, 1910, 1901, 1290]  [90, 100, 1190, 1910, 1901, 1290] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  1290  [90, 100, 1190, 1910, 1901, 1290]  1290  [90, 100, 1190, 1910, 1901, 1290] |  |  | | --- | | C.  [90, 100, 1190, 1910, 1901, 1290]  90  [90, 100, 1190, 1910, 1901, 1290]  90  [90, 100, 1190, 1910, 1901, 1290] | | | |
| **Correct Answer: C** | | |
| **218.** | package queue;  import java.util.LinkedList;  public class M4 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.poll());  System.out.println(list);  System.out.println(list.poll());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1190, 1910, 1901, 1290]  1290  [1190, 1910, 1901] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  [100, 1190, 1910, 1901, 1290]  [1190, 1910, 1901, 1290] |  |  | | --- | | C.  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1190, 1910, 1901, 1290]  100  [1190, 1910, 1901, 1290] | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **219.** | package queue;  import java.util.LinkedList;  public class M5 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.peekLast());  System.out.println(list);  System.out.println(list.peekLast());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  [90, 100, 1190, 1910, 1901, 1290]  [90, 100, 1190, 1910, 1901, 1290] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  1290  [90, 100, 1190, 1910, 1901, 1290]  1290  [90, 100, 1190, 1910, 1901, 1290] |  |  | | --- | | C.  [90, 100, 1190, 1910, 1901, 1290]  90  [90, 100, 1190, 1910, 1901, 1290]  90  [90, 100, 1190, 1910, 1901, 1290] | | | |
| **Correct Answer: B** | | |
| **220.** | package queue;  import java.util.LinkedList;  public class M6 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  LinkedList list = new LinkedList();  list.add(90);  list.add(100);  list.add(1190);  list.add(1910);  list.add(1901);  list.add(1290);  System.out.println(list);  System.out.println(list.pollLast());  System.out.println(list);  System.out.println(list.pollLast());  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1190, 1910, 1901, 1290]  1290  [1190, 1910, 1901] |  |  | | --- | | B.  [90, 100, 1190, 1910, 1901, 1290]  1290  [100, 1190, 1910, 1901]  1901  [1190, 1910, 1901, 1290] |  |  | | --- | | C.  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1190, 1910, 1901, 1290]  100  [1190, 1910, 1901, 1290] | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **221.** | package queue;  import java.util.PriorityQueue;;  public class M7 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(100);  queue.add(1190);  queue.add(1910);  queue.add(1901);  queue.add(1290);  queue.add(10);  System.out.println(queue);  System.out.println(queue.peek());  System.out.println(queue);  System.out.println(queue.peek());  System.out.println(queue);  }  } |
| |  | | --- | | A.  [10, 100, 90, 1910, 1901, 1290, 1190]  10  [10, 100, 90, 1910, 1901, 1290, 1190]  10  [10, 100, 90, 1910, 1901, 1290, 1190] |  |  | | --- | | B.  [100, 90, 1910, 1901, 1290, 1190, 10]  100  [100, 90, 1910, 1901, 1290, 1190, 10]  100  [100, 90, 1910, 1901, 1290, 1190, 10] |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |
| **222.** | package queue;  import java.util.PriorityQueue;  public class M8 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(100);  queue.add(1190);  queue.add(1910);  queue.add(1901);  queue.add(1290);  queue.add(10);  System.out.println(queue);  System.out.println(queue.poll());  System.out.println(queue);  System.out.println(queue.poll());  System.out.println(queue);  }  } |
| |  | | --- | | A.  [10, 100, 90, 1910, 1901, 1290, 1190]  10  [10, 100, 90, 1910, 1901, 1290, 1190]  10  [10, 100, 90, 1910, 1901, 1290, 1190] |  |  | | --- | | B.  [100, 90, 1910, 1901, 1290, 1190, 10]  100  [100, 90, 1910, 1901, 1290, 1190, 10]  100  [100, 90, 1910, 1901, 1290, 1190, 10] |  |  | | --- | | C.  [10, 100, 90, 1910, 1901, 1290, 1190]  10  [90, 100, 1190, 1910, 1901, 1290]  90  [100, 1290, 1190, 1910, 1901] | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **223.** | package queue;  import java.util.PriorityQueue;  public class M9 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(100);  queue.add(1190);  System.out.println(queue.get());  }  } |
| |  | | --- | | A.  90  100  1190 |  |  | | --- | | B.  90 100 1190 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: C** | | |
| **224.** | package queue;  import java.util.PriorityQueue;  public class M9 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(100);  queue.add(1190);  for(Object obj : queue) {  System.out.println(obj);  }  }  } |
| |  | | --- | | A.  90  100  1190 |  |  | | --- | | B.  90 100 1190 |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **225.** | package queue;  import java.util.PriorityQueue;  public class M10 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(100.0);  queue.add(1190);  queue.add(119000);  System.out.println(queue);  }  } |
| |  | | --- | | A.  90  100.0  1190  119000 |  |  | | --- | | B.  90 100 1190 119000 |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  ClassCastException | | | |
| **Correct Answer: D** | | |
| **226.** | package queue;  import java.util.PriorityQueue;  public class M11 {  @SuppressWarnings("unchecked")  public static void main(String[] args) {  PriorityQueue queue = new PriorityQueue();  queue.add(90);  queue.add(null);  queue.add(1190);  queue.add(119000);  System.out.println(queue);  }  } |
| |  | | --- | | A.  90  null  1190  119000 |  |  | | --- | | B.  90 null 1190 119000 |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  NullPointerException | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **227.** | Where ever queue is possible there and all stack also possible |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |
| **228.** | We are achiving perfect stack with the help of poll() method |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **229.** | Which class having default auto sorting |
| |  | | --- | | A.  LinkedList |  |  | | --- | | B.  PriorityQueue | | | |
| **Correct Answer: B** | | |
| **230.** | Is it possible to read elements in the PriorityQueue by using get() method |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **231.** | Is it possible to use enhanced loop to read the elements in LinkedList |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: B** | | |
| **232.** | We can insert any type of elements in PriorityQueue |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **233.** | Is it possible to supply null elements to priorityQueue |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: B** | | |
| **234.** | package set;  import java.util.HashSet;  class A  {  int i;  A(int i){  this.i = i;  }  public String toString() {  return "(i = " + i + ")";  }  }  public class M1 {  public static void main(String[] args) {  HashSet set = new HashSet();  set.add(new A(90));  set.add(new A(90));  set.add(new A(90));  set.add(new A(90));  System.out.println(set);  }  } |
| |  | | --- | | A.  [(i = 90), (i = 90), (i = 90), (i = 90)] |  |  | | --- | | B.  [(i = 90)] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **235.** | package set;  import java.util.HashSet;  class A  {  int i;  A(int i){  this.i = i;  }  public String toString() {  return "(i = " + i + ")";  }  @Override  public int hashCode() {  return i;  }  public boolean equals(Object obj) {  return i == ((A)obj).i;  }  }  public class M1 {  public static void main(String[] args) {  HashSet set = new HashSet();  set.add(new A(90));  set.add(new A(90));  set.add(new A(90));  set.add(new A(90));  System.out.println(set);  }  } |
| |  | | --- | | A.  [(i = 90), (i = 90), (i = 90), (i = 90)] |  |  | | --- | | B.  [(i = 90)] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |
| **236.** | package set;  import java.util.HashSet;  public class M2 {  public static void main(String[] args) {  HashSet set = new HashSet();  set.add(90);  set.add(90);  set.add(90);  set.add(90);  set.add(90);  set.add(90);  System.out.println(set);  }  } |
| |  | | --- | | A.  [90, 90, 90, 90, 90,90] |  |  | | --- | | B.  [90] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **237.** | package set;  import java.util.HashSet;  public class M3 {  public static void main(String[] args) {  HashSet set = new HashSet();  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set);  }  } |
| |  | | --- | | A.  true  false  false  false  false  [90] |  |  | | --- | | B.  false  true  true  true  true  true  [90] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |
| **238.** | package set;  import java.util.HashSet;  public class M3 {  public static void main(String[] args) {  HashSet set = new HashSet();  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(90));  System.out.println(set.add(91));  System.out.println(set);  }  } |
| |  | | --- | | A.  true  false  false  false  false  [90, 91] |  |  | | --- | | B.  false  true  true  true  true  true  [90, 91] |  |  | | --- | | C.  true  false  false  false  true  [90, 91] |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **239.** | package set;  import java.util.HashSet;  public class M4 {  public static void main(String[] args) {  HashSet set = new HashSet();  System.out.println(set.add("hello"));  System.out.println(set.add("hello"));  System.out.println(set.add("abc"));  System.out.println(set.add("abc"));  System.out.println(set.add("xyz"));  System.out.println(set.add("xyz"));  System.out.println(set);  }  } |
| |  | | --- | | A.  true  false  true  false  false  false  [abc, xyz, hello] |  |  | | --- | | B.  true  false  true  false  true  false  [abc, xyz, hello] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |
| **240.** | package set;  import java.util.HashSet;  public class M5 {  public static void main(String[] args) {  HashSet set = new HashSet();  System.out.println(set.add(new StringBuffer("hello")));  System.out.println(set.add(new StringBuffer("hello")));  System.out.println(set.add(new StringBuffer("abc")));  System.out.println(set.add(new StringBuffer("abc")));  System.out.println(set.add(new StringBuffer("xyz")));  System.out.println(set.add(new StringBuffer("xyz")));  System.out.println(set);  }  } |
| |  | | --- | | A.  true  false  true  false  true  false  [abc, xyz, hello] |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  true  true  true  true  true  true  [abc, xyz, xyz, hello, hello, abc] | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **241.** | package set;  import java.util.HashSet;  class B{  int i, j;  B(int i, int j){  this.i = i;  this.j = j;  }  public String toString() {  return "(" + i + ", " + j + ")";  }  public int hashCode() {  String s1 = Integer.toString(i);  String s2 = Integer.toString(j);  int hash = s1.hashCode();  hash += s2.hashCode();  return hash;  }  public boolean equals(Object obj) {  return (i == ((B)obj).i && j == ((B)obj).j);  }  }  public class M6 {  public static void main(String[] args) {  HashSet set = new HashSet();  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,11)));  System.out.println(set.add(new B(90,11)));  System.out.println(set.add(new B(90,11)));  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,10)));  System.out.println(set.add(new B(90,12)));  System.out.println(set);  }  } |
| |  | | --- | | A.  true  false  false  false  true  false  false  false  false  true  [(90, 10), (90, 11), (90, 12)] |  |  | | --- | | B.  true  false  false  false  true  false  false  false  false  false  [(90, 10), (90, 11)] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **242.** | package set;  import java.util.HashSet;  class D  {  int i;  D(int i){  this.i = i;  }  public String toString() {  return "(" + i + ")";  }  }  public class M8 {  public static void main(String[] args) {  D d1 = new D(90);  D d2 = new D(90);  HashSet set = new HashSet();  set.add(d1);  set.add(d2);  System.out.println(set);  }  } |
| |  | | --- | | A.  [(90)] |  |  | | --- | | B.  [(90), (90)] |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **243.** | package set;  import java.util.HashSet;  class E  {  int i;  E(int i){  this.i = i;  }  public String toString() {  return "(" + i + ")";  }  public int hashCode() {  String s1 = Integer.toString(i);  int hash = s1.hashCode();  return hash;  }  public boolean equals(Object obj) {  boolean b1 = (i == ((E)obj).i);  return b1;  }  }  public class M9 {  public static void main(String[] args) {  E d1 = new E(90);  E d2 = new E(90);  HashSet set = new HashSet();  set.add(d1);  set.add(d2);  System.out.println(set);  }  } |
| |  | | --- | | A.  [(90)] |  |  | | --- | | B.  [(90), (90)] |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |
| **244.** | package set;  import java.util.LinkedHashSet;  public class M11 {  public static void main(String[] args) {  LinkedHashSet set = new LinkedHashSet();  set.add(90);  set.add(90);  set.add(190);  set.add(190);  set.add(90);  set.add(190);  set.add(290);  set.add(390);  set.add(390);  set.add(290);  set.add(390);  set.add(500);  System.out.println(set);  }  } |
| |  | | --- | | A.  [90, 290, 190, 500,390] |  |  | | --- | | B.  [90, 190, 290, 390, 500] |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **245.** | Is it possible to sort elements under TreeSet |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |
| **246.** | package set;  import java.util.HashSet;  import java.util.TreeSet;  public class M12 {  public static void main(String[] args) {  HashSet set = new HashSet();  set.add(90);  set.add(190);  set.add(910);  set.add(901);  set.add(290);  set.add(920);  set.add(500);  System.out.println(set);  TreeSet set1 = new TreeSet();  set1.addAll(set);  System.out.println(set1);  }  } |
| |  | | --- | | A.  [290, 500, 901, 920, 90, 190, 910]  [90, 190, 290, 500, 901, 910, 920] |  |  | | --- | | B.  [290, 500, 901, 920, 90, 190, 910]  [90, 190, 290, 500, 920, 901, 910] |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **247.** | package map;  import java.util.HashMap;  public class M1 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put("key1", 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', false);  System.out.println(map1);  }  } |
| |  | | --- | | A.  {key1=2314, a=false, 3.4=true, 345=rty} |  |  | | --- | | B.  {key1=2314, 345=rty, 3.4=true, a=false} | | | |
| **Correct Answer: A** | | |
| **248.** | package map;  import java.util.HashMap;  public class M2 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put(null, 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', null);  System.out.println(map1);  }  } |
| |  | | --- | | A.  NullPointerException |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  {null=2314, a=null, 3.4=true, 345=rty} | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **249.** | package map;  import java.util.HashMap;  public class M3 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put(null, 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', null);  System.out.println(map1);  String s1 = (String) map1.get(345);  System.out.println(s1);  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  {null=2314, a=null, 3.4=true, 345=rty}  rty |  |  | | --- | | C.  {null=2314, a=null, 3.4=true, 345=rty}  345 | | | |
| **Correct Answer: B** | | |
| **250.** | package map;  import java.util.HashMap;  public class M3 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put(null, 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', null);  System.out.println(map1);  String s1 = (String) map1.get(345);  System.out.println(s1);  String s2 = (String) map1.get("xyz");  System.out.println(s2);  }  } |
| |  | | --- | | A.  {null=2314, a=null, 3.4=true, 345=rty}  rty |  |  | | --- | | B.  NullPointerException | | | |
| **Correct Answer: C, , ,** | | |

|  |  |
| --- | --- |
| **251.** | package map;  import java.util.HashMap;  import java.util.Set;  public class M4 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put("abc", 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', "xyz");    Set keys = map1.keySet();  System.out.println(keys);  }  } |
| |  | | --- | | A.  [a, abc, 3.4, 345] |  |  | | --- | | B.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **252.** | package map;  import java.util.HashMap;  import java.util.Set;  public class M5 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put("abc", 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', "xyz");  Set keys = map1.keySet();  for(Object key : keys) {  System.out.println(key + ":" + map1.get(key));  }  System.out.println(keys);  }  } |
| |  | | --- | | A.  a  abc  3.4  345  [a, abc, 3.4, 345] |  |  | | --- | | B.  a:xyz  abc:2314  3.4:true  345:rty  [a, abc, 3.4, 345] |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **253.** | package map;  import java.util.HashMap;  import java.util.Set;  public class M6 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put("abc", 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', "xyz");  Set entries = map1.entrySet();  for(Object entry : entries) {  System.out.println(entry);  }  }  } |
| |  | | --- | | A.  xyz  2314  true  rty |  |  | | --- | | B.  a=xyz  abc=2314  3.4=true  345=rty |  |  | | --- | | C.  Compilation Error | | | | |
| **Correct Answer: B** | | | |
| **254.** | package map;  import java.util.HashMap;  import java.util.Map.Entry;  import java.util.Set;  public class M7 {  public static void main(String[] args) {  HashMap map1 = new HashMap();  map1.put("abc", 2314);  map1.put(345, "rty");  map1.put(3.4, true);  map1.put('a', "xyz");  Set entries = map1.entrySet();  Entry entry;  for(Object obj : entries) {  entry = (Entry) obj;  System.out.println(entry.getKey() + ":" + entry.getValue());  }  }  } | |
| |  | | --- | | A.  xyz  2314  true  rty |  |  | | --- | | B.  a=xyz  abc=2314  3.4=true  345=rty |  |  | | --- | | C.  Compilation Error | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **255.** | package map;  import java.util.Hashtable;  public class M9 {  public static void main(String[] args) {  Hashtable table = new Hashtable();  table.put("abc", 3456);  table.put("a", 345);  table.put(null, true);  table.put(true, 3.4);  System.out.println(table);  }  } |
| |  | | --- | | A.  abc=3456  a=345  null=true  true=3.4 |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  NullPointerExxeption | | | |
| **Correct Answer: C** | | |
| **256.** | package map;  import java.util.LinkedHashMap;  public class M11 {  public static void main(String[] args) {  LinkedHashMap map = new LinkedHashMap();  map.put("abc", 34);  map.put("abc1", 34);  map.put("abc2", 34);  map.put("abc3", 34);  map.put("abc4", 34);  map.put("abc5", 34);  map.put("abc6", 34);  map.put("abc7", 34);  System.out.println(map);  }  } |
| |  | | --- | | A.  {abc=34, abc1=34, abc2=34, abc3=34, abc4=34, abc5=34, abc6=34, abc7=34} |  |  | | --- | | B.  {abc=34, abc1=34, abc3=34, abc2=34, abc4=34, abc7=34, abc6=34, abc5=34} |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **257.** | package map;  import java.util.HashMap;  import java.util.TreeMap;  class A implements Comparable{  int i;  A(int i){  this.i = i;  }  public String toString() {  return "(i = " + i + ")";  }  public int compareTo(Object obj) {  return i - ((A)obj).i;  }  }  public class M13 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put(new A(90), "test1");  map.put(new A(9), "test2");  map.put(new A(0), "test3");  map.put(new A(190), "test4");  map.put(new A(910), "test5");  map.put(new A(901), "test6");  System.out.println(map);  TreeMap map1 = new TreeMap();  map1.putAll(map);  System.out.println(map1);  }  } | |
| |  | | --- | | A.  {(i = 0)=test3, (i = 901)=test6, (i = 910)=test5, (i = 90)=test1, (i = 9)=test2, (i = 190)=test4}  {(i = 0)=test3, (i = 9)=test2, (i = 90)=test1, (i = 190)=test4, (i = 901)=test6, (i = 910)=test5} |  |  | | --- | | B.  {(i = 0)=test3, (i = 9)=test2, (i = 90)=test1, (i = 190)=test4, (i = 901)=test6, (i = 910)=test5}  {(i = 0)=test3, (i = 901)=test6, (i = 910)=test5, (i = 90)=test1, (i = 9)=test2, (i = 190)=test4} |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | | |
| **Correct Answer: A** | | | |
| **258.** | package map;  import java.util.Comparator;  import java.util.HashMap;  import java.util.TreeMap;  class B {  int i, j;  B(int i, int j){  this.i = i;  this.j = j;  }  public String toString() {  return "(" + i + "," + j + ")";  }  }  class SortBasedOnI implements Comparator{  public int compare(Object obj1, Object obj2) {  return ((B)obj1).i - ((B)obj2).i;  }  }  class SortBasedOnJ implements Comparator{  public int compare(Object obj1, Object obj2) {  return ((B)obj1).j - ((B)obj2).j;  }  }  public class M14 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put(new B(90, 0), "test1");  map.put(new B(9, 56), "test2");  map.put(new B(0, 125), "test3");  map.put(new B(190, 45), "test4");  map.put(new B(910, 50), "test5");  map.put(new B(901, 52), "test6");  System.out.println(map);  TreeMap map1 = new TreeMap(new SortBasedOnI());  map1.putAll(map);  System.out.println(map1);    TreeMap map2 = new TreeMap(new SortBasedOnJ());  map2.putAll(map);  System.out.println(map2);  }  } |
| |  | | --- | | A.  {(0,125)=test3, (901,52)=test6, (910,50)=test5, (90,0)=test1, (9,56)=test2, (190,45)=test4}  {(90,0)=test1, (190,45)=test4, (910,50)=test5, (901,52)=test6, (9,56)=test2, (0,125)=test3}  {(0,125)=test3, (9,56)=test2, (90,0)=test1, (190,45)=test4, (901,52)=test6, (910,50)=test5} |  |  | | --- | | B.  {(0,125)=test3, (901,52)=test6, (910,50)=test5, (90,0)=test1, (9,56)=test2, (190,45)=test4}  {(0,125)=test3, (9,56)=test2, (90,0)=test1, (190,45)=test4, (901,52)=test6, (910,50)=test5}  {(90,0)=test1, (190,45)=test4, (910,50)=test5, (901,52)=test6, (9,56)=test2, (0,125)=test3} | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **259.** | Which class is used for sorting in list type elements |
| |  | | --- | | A.  Collections class |  |  | | --- | | B.  Priority class |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  TreeMap |  |  | | --- | | E.  None | | | |
| **Correct Answer: A** | | |
| **260.** | Which class is used for sorting in Queue type elements |
| |  | | --- | | A.  Collections class |  |  | | --- | | B.  Priority class |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  TreeMap |  |  | | --- | | E.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **261.** | Which class is used for sorting in set type elements |
| |  | | --- | | A.  Collections class |  |  | | --- | | B.  Priority class |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  TreeMap |  |  | | --- | | E.  None | | | |
| **Correct Answer: C** | | |
| **262.** | Which class is used for sorting in Map type elements |
| |  | | --- | | A.  Collections class |  |  | | --- | | B.  Priority class |  |  | | --- | | C.  TreeSet |  |  | | --- | | D.  TreeMap |  |  | | --- | | E.  None | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **263.** | Which class is non-Synchronized one |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |
| **264.** | Which one is synchronized one |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **265.** | Which class allows null as key value pair |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |
| **266.** | Which class does not allow null as key value pair |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **267.** | Which class is used for sorting of every entry based on key |
| |  | | --- | | A.  HashMap |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  LinkedHashMap |  |  | | --- | | D.  TreeMap |  |  | | --- | | E.  None | | | |
| **Correct Answer: D** | | |
| **268.** | Map is an |
| |  | | --- | | A.  Classs |  |  | | --- | | B.  interface |  |  | | --- | | C.  identifier |  |  | | --- | | D.  Keyword | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **269.** | In which streams get() method available |
| |  | | --- | | A.  Queue |  |  | | --- | | B.  List |  |  | | --- | | C.  Map |  |  | | --- | | D.  both b and c |  |  | | --- | | E.  only c | | | |
| **Correct Answer: D** | | |
| **270.** | Is it possible to develop interface inside a another interface |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **271.** | package iterators;  import java.util.ArrayList;  public class M1 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  }  } |
| |  | | --- | | A.  90  901  910  190  0  9 |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |
| **272.** | package iterators;  import java.util.ArrayList;  public class M2 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  for(int i = 0; i < list.size(); i++) {  System.out.println(list.get(i));  }  }  } |
| |  | | --- | | A.  90  901  910  190  0  9 |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **273.** | package iterators;  import java.util.ArrayList;  public class M3 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  for(Object element : list) {  System.out.print(element + ", ");  }  }  } |
| |  | | --- | | A.  90  901  910  190  0  9 |  |  | | --- | | B.  90,  901,  910,  190,  0,  9, |  |  | | --- | | C.  90, 901, 910, 190, 0, 9, | | | |
| **Correct Answer: C** | | |
| **274.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M4 {  public static void main(String[] args) {    ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  Iterator it = list.iterator();  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  }  } |
| |  | | --- | | A.  90  901  910  190  0  9 |  |  | | --- | | B.  90,  901,  910,  190,  0,  9, |  |  | | --- | | C.  90, 901, 910, 190, 0, 9, | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **275.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M5 {  public static void main(String[] args) {    ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  ListIterator it = list.listIterator();  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  }  } |
| |  | | --- | | A.  90  901  910  190  0  9 |  |  | | --- | | B.  90,  901,  910,  190,  0,  9, |  |  | | --- | | C.  90, 901, 910, 190, 0, 9, | | | |
| **Correct Answer: C** | | |
| **276.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M6 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  while(it.hasNext()) {  System.out.print(it.next() + ", ");  it.remove();  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9, |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **277.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M7 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  Object obj;  while(it.hasNext()) {  obj = it.next();  System.out.print(obj + ", ");  if(obj.equals(190)) {  it.remove();  }  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90,  901,  910,  190,  0,  9,  [90, 901, 910, 0, 9] |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |
| **278.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.function.Consumer;  public class M8 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  Consumer consumer = new Consumer() {  public void accept(Object t) {  System.out.print(t + ", ");  }  };  it.forEachRemaining(consumer);  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90,  901,  910,  190,  0,  9,  [90, 901, 910, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **279.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.function.Consumer;  public class M9 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  it.forEachRemaining(new Consumer() {  @Override  public void accept(Object t) {  System.out.print(t + ", ");  }  });  System.out.println();  System.out.println(list);  }  } | |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90,  901,  910,  190,  0,  9,  [90, 901, 910, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | | |
| **Correct Answer: B** | | | |
| **280.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.function.Consumer;  public class M10 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  it.forEachRemaining((o1) -> System.out.println(o1));  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90,  901,  910,  190,  0,  9,  [90, 901, 910, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **281.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M11 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println("-----------------------");  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  -----------------------  [90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90,  901,  910,  190,  0,  9,  -----------------------  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |
| **282.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M12 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();  list.add(100);  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9, 100] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  ConcurrentModificationException |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **283.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  public class M13 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  Iterator it = list.iterator();    while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  list.add(100);  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9, 100] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  ConcurrentModificationException |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |
| **284.** | package iterators;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.ListIterator;  public class M14 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();    while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println("---------------");  while(it.hasPrevious()) {  System.out.print(it.previous() + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  ---------------  9, 0, 190, 910, 901, 90,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  Reverse reading is not possible |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **285.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M15 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();    while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println("---------------");  while(it.hasPrevious()) {  System.out.print(it.previous() + ", ");  }  System.out.println();  System.out.println("---------------");  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  ---------------  9, 0, 190, 910, 901, 90,  ---------------  [90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  ---------------  ---------------  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | C.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  ---------------  9, 0, 190, 910, 901, 90,  ---------------  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9] |  |  | | --- | | D.  None | | | |
| **Correct Answer: C** | | |
| **286.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M18 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();  it.add(100);  while(it.hasNext()) {  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [100, 90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 910, 190, 0, 9, 100] |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **287.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M19 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();  Object obj;  while(it.hasNext()) {  obj = it.next();  if(obj.equals(910)) {  it.set(300);  }  System.out.print(it.next() + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  910, 190, 0, 9,  [100, 90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  901, 190, 9,  [90, 901, 300, 190, 0, 9] |  |  | | --- | | C.  [90, 901, 910, 190, 0, 9]  [90, 901, 300, 190, 0, 9] | | | |
| **Correct Answer: B** | | |
| **288.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M19 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();  Object obj;  while(it.hasNext()) {  obj = it.next();  if(obj.equals(910)) {  it.set(300);  }  System.out.print(obj + ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]    [90, 901, 910, 190, 0, 9] |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  901, 190, 9,  [90, 901, 300, 190, 0, 9] |  |  | | --- | | C.  [90, 901, 910, 190, 0, 9]  90, 901, 910, 190, 0, 9,  [90, 901, 300, 190, 0, 9] | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **289.** | package iterators;  import java.util.ArrayList;  import java.util.ListIterator;  public class M20 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(90);  list.add(901);  list.add(910);  list.add(190);  list.add(0);  list.add(9);  System.out.println(list);  ListIterator it = list.listIterator();  Object obj;  while(it.hasNext()) {  System.out.print(it.previousIndex() + ":");  obj = it.next();  System.out.print(obj + ":");  System.out.print(it.nextIndex()+ ", ");  }  System.out.println();  System.out.println(list);  }  } |
| |  | | --- | | A.  [90, 901, 910, 190, 0, 9]  -1:90:1, 0:901:2, 1:910:3, 2:190:4, 3:0:5, 4:9:6,  [90, 901, 910, 190, 0, |  |  | | --- | | B.  [90, 901, 910, 190, 0, 9]  0:90:1, 1:901:2, 2:910:3, 3:190:4, 4:0:5, 5:9:6,  [90, 901, 910, 190, 0, | | | |
| **Correct Answer: A** | | |
| **290.** | package iterators;  import java.util.Vector;  public class M21 {  public static void main(String[] args) {  Vector v1 = new Vector();  v1.add(9000);  v1.add(100);  v1.add(9000000);  v1.add(0);  System.out.println(v1);  }  } |
| |  | | --- | | A.  [9000, 100, 9000000, 0] |  |  | | --- | | B.  [0, 100, 9000000, 9000] | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **291.** | package iterators;  import java.util.Enumeration;  import java.util.Vector;  public class M22 {  public static void main(String[] args) {  Vector v1 = new Vector();  v1.add(9000);  v1.add(100);  v1.add(9000000);  v1.add(0);  Enumeration e1 = v1.elements();  while(e1.hasMoreElements()) {  System.out.println(e1.nextElement());  }  System.out.println("----------------------------");  while(e1.hasMoreElements()) {  System.out.println(e1.nextElement());  }  System.out.println("----------------------------");  System.out.println(v1);  }  } |
| |  | | --- | | A.  9000  100  9000000  0  ----------------------------  9000  100  9000000  0  ----------------------------  [9000, 100, 9000000, 0] |  |  | | --- | | B.  9000  100  9000000  0  ----------------------------  ----------------------------  [9000, 100, 9000000, 0] |  |  | | --- | | C.  ----------------------------  9000  100  9000000  0  ----------------------------  [9000, 100, 9000000, 0] | | | |
| **Correct Answer: B** | | |
| **292.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M1 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(900);  list.add(901);  list.add(100);  list.add(990);  list.add(909);  list.add(400);  list.add(904);  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [900, 901, 100, 990, 909, 400, 904]  [100, 400, 900, 901, 904, 909, 990] |  |  | | --- | | B.  [900, 901, 100, 990, 909, 400, 904]  [900, 901, 100, 990, 909, 400, 904] |  |  | | --- | | C.  [100, 400, 900, 901, 904, 909, 990]  [100, 400, 900, 901, 904, 909, 990] |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **293.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M2 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(900);  list.add(901);  list.add(100);  list.add(990);  list.add(909);  list.add(400);  list.add(904);  System.out.println(list);  Collections.sort(list, Collections.reverseOrder());  System.out.println(list);  }  } |
| |  | | --- | | A.  [900, 901, 100, 990, 909, 400, 904]  [100, 400, 900, 901, 904, 909, 990] |  |  | | --- | | B.  [900, 901, 100, 990, 909, 400, 904]  [900, 901, 100, 990, 909, 400, 904] |  |  | | --- | | C.  [900, 901, 100, 990, 909, 400, 904]  [990, 909, 904, 901, 900, 400, 100] | | | |
| **Correct Answer: C** | | |
| **294.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M3 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add("xyz");  list.add("abc");  list.add("test");  list.add("java");  list.add("btm");  list.add("check");  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [xyz, abc, test, java, btm, check]  [abc, btm, check, java, test, xyz] |  |  | | --- | | B.  [xyz, abc, test, java, btm, check]  [xyz, abc, test, java, btm, check] |  |  | | --- | | C.  [abc, btm, check, java, test, xyz]  [abc, btm, check, java, test, xyz] | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **295.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M4 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add("xyz");  list.add("abc");  list.add("test");  list.add("java");  list.add("btm");  list.add("check");  System.out.println(list);  Collections.sort(list, Collections.reverseOrder());  System.out.println(list);  }  } |
| |  | | --- | | A.  [xyz, abc, test, java, btm, check]  [abc, btm, check, java, test, xyz] |  |  | | --- | | B.  [xyz, abc, test, java, btm, check]  [xyz, abc, test, java, btm, check] |  |  | | --- | | C.  [xyz, abc, test, java, btm, check]  [xyz, test, java, check, btm, abc] | | | |
| **Correct Answer: C** | | |
| **296.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M5{  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add("xyz");  list.add("abc");  list.add("test");  list.add("java");  list.add("btm");  list.add("check");  list.add("6787");  list.add("98778");  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  [xyz, abc, test, java, btm, check, 6787, 98778]  [6787, 98778, abc, btm, check, java, test, xyz] |  |  | | --- | | C.  [xyz, abc, test, java, btm, check, 6787, 98778]  [xyz, abc, test, java, btm, check, 6787, 98778] | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **297.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M6{  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(900);  list.add(901);  list.add(100);  list.add(990);  list.add(909.0);  list.add(400);  list.add(904);  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [900, 901, 100, 990, 909, 400, 904]  [100, 400, 900, 901, 904, 909, 990] |  |  | | --- | | B.  [900, 901, 100, 990, 909.0, 400, 904]  ClassCastException |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |
| **298.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  public class M7 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(900);  list.add(901);  list.add(100);  list.add(990);  list.add(1);  list.add(400);  list.add(904);  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [900, 901, 100, 990, null, 400, 904]  [null, 100, 400, 900, 901, 904, 990] |  |  | | --- | | B.  [900, 901, 100, 990, null, 400, 904]  NullPointerException |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **299.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  class A{  int i;  A(int i){  this.i= i;  }  @Override  public String toString() {  // TODO Auto-generated method stub  return "i = " + i;  }  }  public class M8 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(new A(90));  list.add(new A(9));  list.add(new A(0));  list.add(new A(80));  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [i = 90, i = 9, i = 0, i = 80]  [i = 0, i = 9, i = 80, i = 90] |  |  | | --- | | B.  [i = 90, i = 9, i = 0, i = 80]  ClassCastException |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: B** | | |
| **300.** | Single method interface is also called as functional interface |
| |  | | --- | | A.  True |  |  | | --- | | B.  False | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **301.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  class B implements Comparable {  int i;  B(int i){  this.i= i;  }  @Override  public String toString() {  // TODO Auto-generated method stub  return "i = " + i;  }  @Override  public int compareTo(Object o) {  return i - ((B)o).i;  }  }  public class M9 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(new B(90));  list.add(new B(9));  list.add(new B(0));  list.add(new B(80));  System.out.println(list);  Collections.sort(list);  System.out.println(list);  }  } |
| |  | | --- | | A.  [i = 90, i = 9, i = 0, i = 80]  [i = 0, i = 9, i = 80, i = 90] |  |  | | --- | | B.  [i = 90, i = 9, i = 0, i = 80]  ClassCastException |  |  | | --- | | C.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **302.** | package sorting;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  class C{  int i, j;  public C(int i, int j) {  super();  this.i = i;  this.j = j;  }  @Override  public String toString() {  return "C [i=" + i + ", j=" + j + "]";  }  }  class SortBasedOnIValue implements Comparator{  @Override  public int compare(Object o1, Object o2) {  return ((C)o1).i - ((C)o2).i;  }  }  class SortBasedOnJValue implements Comparator{  @Override  public int compare(Object o1, Object o2) {  return ((C)o1).j - ((C)o2).j;  }  }  public class M10 {  public static void main(String[] args) {  ArrayList list = new ArrayList();  list.add(new C(10, 20));  list.add(new C(20, 10));  list.add(new C(5, 15));  list.add(new C(15, 8));  System.out.println(list);  Collections.sort(list, new SortBasedOnIValue());  System.out.println(list);  Collections.sort(list, new SortBasedOnJValue());  System.out.println(list);  }  } |
| |  | | --- | | A.  [C [i=10, j=20], C [i=20, j=10], C [i=5, j=15], C [i=15, j=8]]  [C [i=5, j=15], C [i=10, j=20], C [i=15, j=8], C [i=20, j=10]]  [C [i=15, j=8], C [i=20, j=10], C [i=5, j=15], C [i=10, j=20]] |  |  | | --- | | A.  [C [i=10, j=20], C [i=20, j=10], C [i=5, j=15], C [i=15, j=8]]  [C [i=15, j=8], C [i=20, j=10], C [i=5, j=15], C [i=10, j=20]]  [C [i=5, j=15], C [i=10, j=20], C [i=15, j=8], C [i=20, j=10]] | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **303.** | package mapBasedSorting;  import java.util.HashMap;  import java.util.Set;  public class M1 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put("hello", 201);  map.put("abc", 201);  map.put("xyz", 201);  map.put("test", 201);  map.put("btm", 201);  map.put("java", 201);    Set entries = map.entrySet();    for(Object obj : entries) {  System.out.println(obj);  }  }  }  //How the output will be |
| |  | | --- | | A.  same order |  |  | | --- | | B.  in order output |  |  | | --- | | C.  None | | | | |
| **Correct Answer: B** | | | |
| **304.** | package mapBasedSorting;  import java.util.HashMap;  import java.util.Map.Entry;  import java.util.Set;  public class M2 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put("hello", 201);  map.put("abc", 20);  map.put("xyz", 21);  map.put("test", 1);  map.put("btm", 25);  map.put("java", 101);    Set entries = map.entrySet();  Entry entry;  for(Object obj : entries) {  entry = (Entry) obj;  System.out.println(entry.getKey() + ":" + entry.getValue());  }  }  }  //How the output will be | |
| |  | | --- | | A.  same order |  |  | | --- | | B.  in order output |  |  | | --- | | C.  Compilation Error | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **305.** | package mapBasedSorting;  import java.util.HashMap;  import java.util.TreeMap;  public class M3 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put("hello", 201);  map.put("abc", 20);  map.put("xyz", 21);  map.put("test", 1);  map.put("btm", 25);  map.put("java", 101);  TreeMap map1 = new TreeMap();  map1.putAll(map);  System.out.println(map1);  }  } |
| |  | | --- | | A.  {abc=20, btm=25, hello=201, java=101, test=1, xyz=21} |  |  | | --- | | B.  {abc:20, java:101, test:1, xyz:21, btm:25, hello:201} |  |  | | --- | | C.  {  abc=20,  btm=25,  hello=201,  java=101,  test=1,  xyz=21  } | | | | |
| **Correct Answer: A** | | | |
| **306.** | package mapBasedSorting;  import java.util.Comparator;  import java.util.HashMap;  import java.util.Map.Entry;  import java.util.Set;  import java.util.TreeSet;  public class M4 {  public static void main(String[] args) {  HashMap map = new HashMap();  map.put("hello", 201);  map.put("abc", 20);  map.put("xyz", 21);  map.put("test", 1);  map.put("btm", 25);  map.put("java", 101);    Set entries = map.entrySet();    Comparator c1 = new Comparator() {  public int compare(Object o1, Object o2) {  Entry e1 = (Entry) o1;  Entry e2 = (Entry) o2;  return ((Integer)e1.getValue()).compareTo((Integer)e2.getValue()) ;  }  };  TreeSet set = new TreeSet(c1);  set.addAll(entries);  System.out.println(set);  }  } | |
| |  | | --- | | A.  [test=1, abc=20, xyz=21, btm=25, java=101, hello=201] |  |  | | --- | | B.  {abc=20, btm=25, hello=201, java=101, test=1, xyz=21} |  |  | | --- | | C.  Compilation Error | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **307.** | Which are synchronized by default |
| |  | | --- | | A.  vector |  |  | | --- | | B.  Hashtable |  |  | | --- | | C.  Hashmap |  |  | | --- | | D.  Both a and b |  |  | | --- | | E.  only c | | | |
| **Correct Answer: D** | | |
| **308.** | In vector object how many threads are allowed to enter at a time |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  Multiple threads | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **309.** | In Hashtable object how many threads are allowed to enter at a time |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  Multiple threads | | | | |
| **Correct Answer: A** | | | |
| **310.** | Which of the fillowing objects are not possible to use in multi threaded environment stright away | |
| |  | | --- | | A.  ArrayList |  |  | | --- | | B.  Vector |  |  | | --- | | C.  Hashtable |  |  | | --- | | D.  None | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **311.** | Which of the following threads allows only one thread at a time |
| |  | | --- | | A.  Non synchronized thread |  |  | | --- | | B.  Synchronized thread |  |  | | --- | | C.  None | | | |
| **Correct Answer: B** | | |
| **312.** | One object can be shared to how many threads |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  any number of threads |  |  | | --- | | E.  None | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **313.** | Is it possible to make ArrayList as synchronized |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | |
| **Correct Answer: A** | | |
| **314.** | package collectionsSynchronized;  import java.util.List;  import java.util.ArrayList;  import java.util.Collections;  public class M1 {  public static void main(String[] args) {  List list = new ArrayList();  list.add(90);  list.add(90);  list.add(90);  list.add(90);  System.out.println(list);    list = Collections.synchronizedList(list);    }  } |
| |  | | --- | | A.  [90, 90, 90, 90] |  |  | | --- | | B.  Compilation Error |  |  | | --- | | C.  None | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **315.** | package collectionsSynchronized;  import java.util.Collections;  import java.util.HashSet;  import java.util.Set;  public class M2 {  public static void main(String[] args) {  Set set = new HashSet();// line 9    set = Collections.synchronizedSet(set);// line 13  }  }  // Check this program after line 13 it is thread safe are not |
| |  | | --- | | A.  Thread safe |  |  | | --- | | B.  Not safe | | | |
| **Correct Answer: A** | | |
| **316.** | package collectionsSynchronized;  import java.util.Collections;  import java.util.HashMap;  import java.util.Map;  public class M3 {  public static void main(String[] args) {  Map map = new HashMap(); // line 9    map = Collections.synchronizedMap(map); // line 11  }  }  // Check this program after line 9 it is thread safe are not |
| |  | | --- | | A.  Thread safe |  |  | | --- | | B.  Not safe | | | |
| **Correct Answer: B** | | |

Bottom of Form

4