Given the following code:

```
1.  import java.util.*;
2.  class Dog implements Comparable<Dog> {
3.        private String name;
4.        private int age;
5.        private String owner;
6.        Dog(String n,String o,int a) {
7.              name = n; owner = o; age=a;
8.        }
9.        public String toString() {
10.             return name;
11.       }
12.       public int compareTo(Dog d) {
13.             return owner.compareTo(d.owner);
14.       }
15. }
16. public class Dtest {
17.       public static void main(String [ ] args) {
18.             ArrayList<Dog> doglist = new ArrayList<Dog>();
19.             doglist.add(new Dog("Lazy","John",3));
20.             doglist.add(new Dog("White","Henry",2));
21.             doglist.add(new Dog("Blaky","Bert",5));
22.             doglist.add(new Dog("Tazan","Jack",1));
23.             Collections.sort(doglist);
24.             System.out.print(doglist);
25.       }
26. }
```

What is the result?

Please select :

- A. [Lazy, White, Blaky, Tazan]
- B. [Blaky, White, Tazan, Lazy]
- C. [Bert ,John, Jack, Henry]
- D. [Bert, Henry, Jack, John]
- E. [Blaky, Lazy, Tazan, White]

```
2.  class Dog {
3.        String name;
4.        int age;
5.        String owner;
6.        Dog(String n,String o,int a) {
7.              name = n; owner = o; age=a;
8.        }
9.        public String toString() {
10.             return owner;
11.       }
12. }
13. public class DTest2 {
14.       public static void main(String [ ] args) {
15.             ArrayList<Dog> doglist = new ArrayList<Dog>();
16.             doglist.add(new Dog("Lazy","John",3));
17.             doglist.add(new Dog("White","Henry",2));
18.             doglist.add(new Dog("Blaky","Bert",5));
19.             doglist.add(new Dog("Tazan","Jack",1));
20.             Sort1 s1 = new Sort1();
21.             Collections.sort(doglist , s1);
22.             System.out.print(doglist);
23.             Sort2 s2 = new Sort2();
24.             Collections.sort(doglist , s2);
25.             System.out.print(doglist);
26.       }
27.       static class Sort1 implements Comparator<Dog> {
28.             public int compare(Dog first,Dog second) {
29.                   return first.name.compareTo(second.name);
30.             }
31.       }
32.       static class Sort2 implements Comparator<Dog> {
33.             public int compare(Dog first,Dog second) {
34.                   return first.owner.compareTo(second.owner);
35.             }
36.       }
37. }
```

Which options are correct ?
(Select two options.)

Please select :

- A. [Bert, John, Jack, Henry][Bert, Henry, Jack, John]
- B. [Henry, Jack, John, Bert] [Bert, John, Jack, Henry]
- C. Compilation succeeds.
- D. In the java.util.Collections class, there is no overloaded version of the “sort()” method which can take a List and a Comparator as parameters
- E. Compilation fails.

```
1.  import java.util.*;
2.  public class Gen {
3.          public static void main(String args[ ]) {
4.                  Map<Key, Child> cmap = new HashMap<Key, Child>();
5.                  cmap.put(new Key(3), new Child("Reka"));
6.                  cmap.put(new Key(2), new Child("Buddhika"));
7.                  cmap.put(new Key(5), new Child("Piumi"));
8.                  cmap.put(new Key(8), new Child("Silva"));
9.                  cmap.put(new Key(2), new Child("Livera"));
10.
11.                 System.out.println(cmap.size());
12.                 System.out.print(cmap.get(new Key(5)));
13.         }
14. }
15. class Child {
16.         String name;
17.          Child(String s){ name = s; }
18.          public int hashCode() {
19.                  return name.length();
20.          }
21.          public boolean equals(Object o) {
22.                  if((o instanceof Child) && ((Child)o).name== name)
23.                      return true;
24.                  else
25.                      return false;
26.          }
27.          public String toString() {
28.                  return name;
29.          }
30.  }
31.  class Key {
32.          int id;
33.          Key(int i){ id = i; }
34.
35.   public boolean equals(Object o){
36.          if((o instanceof Key) && ((Key)o).id == id)
37.              return true;
38.          else
39.              return false;
40.          }
41.
42.          public String toString() {
43.                  return ("key " + id);
44.          }
45. }
```

What is the output?

Please select :

- A. 4 Piumi
- B. 5 null
- C. 5 Piumi
- D. 4 null
- E. Compilation fails.

Given the following code:

```
1.  import java.util.*;
2.
3.  public class Gen {
4.          public static void main(String args[ ]) {
5.                  Map<Key, Child> cmap = new HashMap<Key, Child>();
6.
7.                  cmap.put(new Key(3), new Child("Reka"));
8.                  cmap.put(new Key(2), new Child("Buddhika"));
9.                  cmap.put(new Key(5), new Child("Piumi"));
10.                 cmap.put(new Key(8), new Child("Silva"));
11.                 cmap.put(new Key(2), new Child("Livera"));
12.
13.                 System.out.print(cmap.size()+" ");
14.                 System.out.print(cmap.get(new Key(2)));
15.         }
16. }
17.
18. class Child {
19.         String name;
20.         Child(String s){ name = s; }
21.
22.         public String toString() {
23.                 return name;
24.         }
25. }
26.
27. class Key {
28.         int id;
29.         Key(int i){ id = i; }
30. public int hashCode() {
31.         return id%10;
32. }
33. public boolean equals(Object o) {
34.         if((o instanceof Key) && ((Key)o).id == id)
35.             return true;
36.         else
37.             return false;
38.         }
39.         public String toString() {
40.                 return ("key " + id + " :");
41.         }
42. }
```

What is the Output ?

Please select :

- A. 4 Buddhika
- B. 5 null
- C. 5 Buddhika
- D. 4 Livera
- E. 5 null

```
1.  import java.util.*;
2.
3.  public class Gen3 {
4.
5.      public static void main(String args[ ]) {
6.          TreeMap<String, String> map = new TreeMap<String, String>();
7.          map.put("a", "apple");
8.          map.put("e", "egg");
9.          map.put("g", "gear");
10.         SortedMap<String, String> smap = map.subMap("a","e");
11.         smap.put("b", "ball");
12.         smap.put("f", "fish");
13.         map.put("c", "cat");
14.         map.remove("a");
15.         System.out.println(smap);
16.         System.out.print(map);
17.     }
18. }
```

What is the Output?

Please select :

- A. {a=apple, b=ball, c=cat, f=fish}
  {b=ball, c=cat, e=egg, g=gear}
- B. {b=ball, c=cat, e=egg}
  {b=ball, c=cat, e=egg, f=fish, g=gear}
- C. An exception is thrown at runtime.
- D. Compilation fails because of an error on line 10.
- E. Compilation fails because of an error on line 13.

Consider following code segment :

```
public class Sample {
    public static void main(String [ ] args) {
        Queue<String> row = new LinkedList<String>();
        row.add("OR");
        row.add("CA");
        row.add("MO");
        display(row);
    }

    public static void display(Queue row) {
        row.add(new Integer(94123));
        while (!row.isEmpty ( ) )
            System.out.print(row.poll() + "  ");
    }
}
```

What would be the output, if it is executed as a program?

Please select :

- A. Compile error since Integer cannot be added at line. An exception could be thrown at runtime.
- B. 94123 OR CA MO
- C. OR CA MO 94123
- D. OR CA MO
- E. Exception is thrown

What is the result of compiling and running the following code?

```
import java.util.*;

class Collection1
{
	public static void main(String[] args)
	{
		Set<String> s=new HashSet<String>();
		s.add("apple");
		s.add("Apple");
		s.add("1");
		s.add("1.0");
		String[] ss=s.toArray();
		for(String str : ss)
			System.out.print(str);
	}
}
```

Please select :

- A. Compiler error
- B. Exception
- C. Prints all 4 strings
- D. Prints "apple", "1", and "1.0"

Which collection implementation is suitable for maintaining an ordered sequence of objects when objects are frequently inserted and removed from the middle of the sequence?

Please select :

- A. TreeMap
- B. Vector
- C. ArrayList
- D. LinkedList

What is the return type of the firstEntry() method in the NavigableMap interface?

Please select :

- A. String
- B. Object
- C. Map
- D. Map.Entry

Given the following code line.

Map<Integer,String> map = new HashMap<Integer,String>();

Which of the following are legal?

Select three choices.

Please select :

- [ ] A. for(Map.Entry pairs : map.entrySet()) { }
- [ ] B. Iterator i = map.entrySet().iterator();
- [ ] C. Iterator i = map.iterator();
- [ ] D. Iterator<Map.Entry<Integer,String>> i = map.entrySet().iterator();
- [ ] E. Iterator<Map.Entry> i = map.entrySet().iterator();

Which is the most suitable Java collection class for storing various currencies and their equivalent prices in USD if multiple threads access and modify this data?
It is required that the class should be synchronized inherently.

Please select :

- A. HashSet
- B. Vector
- C. Hashtable
- D. HashMap
- E. TreeMap

Given:

```
34.  HashMap props = new HashMap();
35.  props.put("key45", "some value");
36.  props.put("key12", "some other value");
37.  props.put("key39", "yet another value");
38.  Set s = props.keySet();
39.     // insert code here
```

What, inserted at line 39, will sort the keys in the props HashMap?

Please select :

- A. Arrays.sort(s);
- B. s = new TreeSet(s);
- C. Collections.sort(s);
- D. s = new SortedSet(s);

Given the following code :

```
34. HashMap<String, String> props = new HashMap<>();
35. props.put("key45", "some value");
36. props.put("key12", "some other value");
37. props.put("key39", "yet another value");
38. Set s = props.keySet();
39. // insert code here
```

What, inserted at line 39, will sort the keys in the props HashMap?

Please select :

- A. Arrays.sort(s);
- B. s = new TreeSet<>(s);
- C. Collections.sort(s);
- D. s = new SortedSet<>(s);

Given:

```
1.  import java.util.*;
2.  public class SortCollection {
3.          public static void main(String [ ] args) {
4.                  ArrayList<Integer> a = new ArrayList<>();
5.                  a.add(1); a.add(5); a.add(3);
6.                  Collections.sort(a);
7.                  a.add(2);
8.                  Collections.reverse(a);
9.                  System.out.println("Result:" + a);
10.         }
11.  }
```

What is the result?

Please select :

- A. Result: [1, 2, 3, 5]
- B. Result: [2, 1, 3, 5]
- C. Result: [2, 5, 3, 1]
- D. Result: [5, 3, 2, 1]
- E. Result: [1, 3, 5, 2]
- F. Compilation fails.
- G. An exception is thrown at runtime.

Given :

```
10.  interface A { void x(); }
11.  class B implements A { public void x() {} public void y() {} }
12.  class C extends B { public void x() {} }
```

And:

```
20.  java.util.List<A> list = new java.util.ArrayList<>();
21.  list.add(new B());
22.  list.add(new C());
23.  for (A a : list) {
24.      a.x();
25.      a.y();
26.  }
```

What is the result?

Please select :

- A. The code runs with no output.
- B. An exception is thrown at runtime.
- C. Compilation fails because of an error in line 20.
- D. Compilation fails because of an error in line 21.
- E. Compilation fails because of an error in line 23.
- F. Compilation fails because of an error in line 25.

Given:

```
3.  import java.util.*;
4.  public class MyMap {
5.      public static void main(String [ ] args) {
6.          Set<Integer> set = new HashSet<>();
7.          Integer integer1 = 51;
8.          Integer integer2 = 52;
9.          set.add(integer1);
10.         set.add(integer1);
11.         set.add(integer2); System.out.print(set.size() + " ");
12.         set.remove(integer1); System.out.print(set.size() + " ");
13.         integer2 = 47;
14.         set.remove(integer2); System.out.print(set.size() + " ");
15.     }
16. }
```

What is the result?

Please select :

- A. 2 1 0
- B. 2 1 1
- C. 3 2 1
- D. 3 2 2
- E. Compilation fails.
- F. An exception is thrown at runtime.

Given:

```
12. import java.util.*;
13. public class TreeSetDiamondDemo {
14.         public static void main(String[ ] args) {
15.                 TreeSet <Integer> set = new TreeSet<>();
16.                 TreeSet <Integer> subset = new TreeSet<>();
17.                  for (int i = 506; i < 513; i++)
18.                       if(i%2 == 0) set.add(i);
19.                       subset = (TreeSet)set.subSet(508, true, 511, true);
20.                       subset.add(529);
21.                       System.out.println(set + " " + subset);
22.         }
23. }
```

What is the result?

Please select :

- A. Compilation fails.
- B. An exception is thrown at runtime.
- C. [508, 510, 512, 529] [508, 510]
- D. [508, 510, 512, 529] [508, 510, 529]
- E. [506, 508, 510, 512, 529] [508, 510]
- F. [506, 508, 510, 512, 529] [508, 510, 529]

Given that the elements of a PriorityQueue are ordered according to natural ordering,

and:

```
2.  import java.util.*;
3.  public class FruitQueue {
4.      public static void main(String [ ] args) {
5.          PriorityQueue<String> pq = new PriorityQueue<>();
6.          pq.add("banana");
7.          pq.add("pear");
8.          pq.add("apple");
9.          System.out.println(pq.poll() + " " + pq.peek());
10.     }
11. }
```

What is the result?

Please select :

- A. apple pear
- B. banana pear
- C. apple apple
- D. apple banana
- E. banana banana

Given :

```
import java.util.*;
public class Test {
        public static void main(String [ ] args) {
                Set<String> set = new HashSet<>();
                set.add("one");
                set.add("three");
                set.add("two");
                System.out.print(set);
        }
}
```

What is the output of the program?

Please select :

- A. one two three
- B. one three two
- C. two three one
- D. the output is unpredictable

Given a class whose instances, when found in a collection of objects, are sorted by using the compareTo() method, which two statements are true?
(Choose two options.)

Please select :

- [ ] A. The class implements java.lang.Comparable.
- [ ] B. The class implements java.util.Comparator.
- [ ] C. The interface used to implement sorting allows this class to define only one sort sequence.
- [ ] D. The interface used to implement sorting allows this class to define many different sort sequences.