COLLECTIONS

1.
```
public abstract class Pet{
  public abstract void type();
}
public class Dog extends Pet{
 @override
      public void type(){
           System.out.println("Bark");
      }
}

public class Cat extends Pet{
      @override
      public void type(){
           System.out.println("Meow");
      }
}
```

Given the below code in main method

```
ArrayList<Pet> group = new ArrayList<Pet>();
group.add(new Dog());
group.add(new Cat());
for (Pet sound : group) sound.type();
```

What is the outcome when you execute the sample code above?

a. Bark
   Meow

b. Dog
   Cat

c. Cat
   Meow

d. Dog
   Bark

2. Which declaration do you use to create a list of String Object?

a. List list<String> = new List<String>();
b. List<String> list = new List()<String>;
c. List<String> list = new ArrayList()<String>;
d. List list<String> = new ArrayList <String>();
e. List<String> list = new ArrayList<String>();

3. List <Double> doubles = new LinkedList<>();
   List <Interger> integers = new LinkedList<Integer>();
   …
   List <String> doubleValues = getValues(doubles);
   List <String> intValues = getValues(integers);

   Given that Double and Integer are subclasses of Number, which declaration of the getValues() method in the sample code above will compile without errors and warnings?
   a. List <String> getValues(List <? extends Number> list)
   b. public List <String> getValues(getlist)
   c. public List <Number> getValues(List <String> list)
   d. public List <String> getValues(List list)
   e. public List <String> getNumbers(List <Value> list)

4. List myList = new ArrayList();
   // ????

   Which code snippet do you insert in the place of // ???? to make the myList thread safe in the sample code above?
   a. myList = myList.getSynchronizedList();
   b. myList = List.SynchronizedList(myList);
   c. myList = Collections.SynchronizedList(myList);
   d. myList = new SynchronizedList(myList);
   e. myList.setSynchronizedList(true);

5. Which collection do you use to implement RandomAccess?
   a. LinkedBlockingDeque
   b. Linkedlist
   c. ArrayList
   d. LinkedHashSet
   e. TreeSet

6. Which class do you use to store a set of <key,value> pairs that are sorted by the key and not inserted randomly?
   a. HashMap
   b. LinkedList
   c. TreeMap
   d. TreeSet
   e. LinkedHashMap

7. Which collection has an upper bound that you can limit from growing beyond a certain size?
   a. ArrayQueue
   b. ConcurrentSkipListMap
   c. CopyOnWriteArraySet
   d. LinkedList
   e. LinkedBlockingQueue

8. Code
```
Iterator it = map.entrySet().iterator();
while (it.hasNext()) {
      Entry item = it.next();
      map.remove(item.getKey());
}
```
Based on the sample code above, how do you resolve the ConcurrentModificationException?

   a.
```
Iterator it = map.entrySet().iterator();
while (it.hasNext()) {
      Entry item = itemlist.addblock();
       it.remove();
}
```
   b.
```
Iterator it = map.entrySet().iterator();
while (it.hasNext()) {
      Entry item = it.next();
      it.remove();
}
```
   c.
```
Iterator it = map.entrySet().iterator();
while (it.hasNext()) {
      Entry item = it.next();
      it.remove(item.getKey());
}
```
   d.
```
Iterator it = map.getitemList().iterator();
while (it.getNext()) {
      Entry item = it.next();
      it.remove();
}
```
   e.
```
Iterator it = map.entrySet().iterator();
while (it.hasNext()) {
      Entry item = it.next();
      map.remove(itemlist);
 }
```

9. Enhanced For Statement: for (Type Identifier : Expression) Statement
In the sample code above, what interface must the Expression implement in order to be used in an enhanced for loop construct?
   a. Iterable
   b. Enumeration
   c. Collection
   d. Map
   e. Hashtable

10.
```
public class Example<B> {
        B b;
        public <B> void printMe(B b) {
              System.out.println(b.getClass().getName());
        }
         public static void main(String args[]) {
              Example<Example> b = new Example<Example>();
             b.printMe("Hello, World");
        }
}
```
What is printed after you call the printMe() method in the sample code above?
   a. String
   b. Example.class
   c. java.lang.String
   d. Hello, World
   e. String "Hello, World"

11. You use a java.util.ArrayList as the implementation for a java.util.List collection.
Referring to the scenario above, what happens when you add an element that exceeds the ArrayList's capacity?
   a. The code throws an Error at runtime.
   b. The ArrayList expands automatically to fit the addition.
   c. The virtual machine terminates the application.
   d. The call throws an ArrayIndexOutOfBoundsException.
   e. The add() call returns with a value of -1 rather than the index.

12.
```
public static void main(String [] args) {
     list.add(args); list.add(args); list.add(args);
     for( String [] strings : list) {
          for( String string : strings) {
               System.out.println(string);
          }
     System.out.println();
}
```
In sample code above, when the variable list is of type java.util.List, how must you declare it so there are no compilation warnings ?

a. List list = new ArrayList<String[]>();
b. List <String> list = new ArrayList<String[]>();
c. List <String[]> list = new ArrayList ();
d. List list = new ArrayList ();
e. List <String[]> list = new ArrayList <String[]>();

13. Which of the following statements is true of a class that implements the Iterator interface?
   a. It contains implementation of hasNext() and next() methods
   b. It can be used to store associative arrays
   c. It is an implementation of the Collection Interface
   d. It has implementation of toString() and getNextString() methods
   e. It can work with StringTokenizer class

14. itemList is a list of items contained in an ArrayList. The application is running two threads – the Reader thread and the Adder thread. The program does not use any synchronized methods or synchronized() blocks. The Reader thread is in the process of traversing the itemList, using a ListIterator. In the scenario above what happens when you add an item to the itemList from the Adder thread?
   a. The Reader thread's next call to an iterator method blocks until the addition is complete
   b. The new item appears as the last item in the iterator
   c. The Reader thread does not know about the new item.
   d. The Reader thread's next call to the iterator methods throws ConcurrentModificationException
   e. Adder thread's call to itemList.add() blocks until the Reader thread finishes using the iterator.