**Answers to Questions and Exercises:**

**Questions**

1. Question: You plan to write a program that uses several basic collection interfaces: Set, List, Queue, and Map. You're not sure which implementations will work best, so you decide to use general-purpose implementations until you get a better idea how your program will work in the real world. Which implementations are these?  
   Answer:  
   Set: HashSet  
   List: ArrayList  
   Queue: LinkedList  
   Map: HashMap
2. Question: If you need a Set implementation that provides value-ordered iteration, which class should you use?   
   Answer:   
   TreeSet guarantees that the sorted set is in ascending element order, sorted according to the natural order of the elements or by the Comparator provided.
3. Question: Which class do you use to access wrapper implementations?   
   Answer:   
   You use the Collections class, which provides static methods that operate on or return collections.

**Exercises**

1. Exercise: Write a program that reads a text file, specified by the first command line argument, into a List. The program should then print random lines from the file, the number of lines printed to be specified by the second command line argument. Write the program so that a correctly-sized collection is allocated all at once, instead of being gradually expanded as the file is read in. Hint: To determine the number of lines in the file, use [java.io.File.length](https://docs.oracle.com/javase/8/docs/api/java/io/File.html" \l "length--" \t "_blank) to obtain the size of the file, then divide by an assumed size of an average line.   
   Answer:   
   Since we are accessing the List randomly, we will use ArrayList. We estimate the number of lines by taking the file size and dividing by 50. We then double that figure, since it is more efficient to overestimate than to underestmate.
2. import java.util.\*;
3. import java.io.\*;
4. public class FileList {
5. public static void main(String[] args) {
6. final int assumedLineLength = 50;
7. File file = new File(args[0]);
8. List<String> fileList =
9. new ArrayList<String>((int)(file.length() / assumedLineLength) \* 2);
10. BufferedReader reader = null;
11. int lineCount = 0;
12. try {
13. reader = new BufferedReader(new FileReader(file));
14. for (String line = reader.readLine(); line != null;
15. line = reader.readLine()) {
16. fileList.add(line);
17. lineCount++;
18. }
19. } catch (IOException e) {
20. System.err.format("Could not read %s: %s%n", file, e);
21. System.exit(1);
22. } finally {
23. if (reader != null) {
24. try {
25. reader.close();
26. } catch (IOException e) {}
27. }
28. }
29. int repeats = Integer.parseInt(args[1]);
30. Random random = new Random();
31. for (int i = 0; i < repeats; i++) {
32. System.out.format("%d: %s%n", i,
33. fileList.get(random.nextInt(lineCount - 1)));
34. }
35. }
36. }

This program actually spends most of its time reading in the file, so pre-allocating the ArrayList has little affect on its performance. Specifying an initial capacity in advance is more likely to be useful when your program repeatly creates large ArrayList objects without intervening I/O.