# Exercises

## EXERCISES

Define each of the following terms:

1. `Collection`
2. `Collections`
3. `Comparator`
4. `List`
5. load factor
6. collision
7. spacetime trade-off in hashing
8. `HashMap`

---

Explain briefly the operation of each of the following methods of class `Vector`:

1. `add`
2. `insertElementAt`
3. `set`
4. `remove`
5. `removeAllElements`
6. `removeElementAt`
7. `firstElement`
8. `lastElement`
9. `isEmpty`
10. `contains`
11. `indexOf`
12. `size`
13. `capacity`

---

Explain why inserting additional elements into a `Vector` object whose current size is less than its capacity is a relatively fast operation and why inserting additional elements into a `Vector` object whose current size is at capacity is a relatively slow operation.

By extending class `Vector`, Java's designers were able to create class `Stack` quickly. What are the negative aspects of this use of inheritance, particularly for class `Stack`?

Briefly answer the following questions:

1. What is the primary difference between a `Set` and a `Map`?
2. Can a two-dimensional array be passed to `Arrays` method `asList`? If yes, how would an individual element be accessed?
3. What happens when you add a primitive type (e.g., `double`) value to a collection?
4. Can you print all the elements in a collection without using an `Iterator`? If yes, how?

Explain briefly the operation of each of the following `Iterator`-related methods:

1. `iterator`
2. `hasNext`
3. `next`

Explain briefly the operation of each of the following methods of class `HashMap`:

1. `put`
2. `get`
3. `isEmpty`
4. `containsKey`
5. `keySet`

Determine whether each of the following statements is **true** or **false**. If **false**, explain why.

1. Elements in a `Collection` must be sorted in ascending order before a `binarySearch` may be performed.
2. Method `first` gets the first element in a `treeSet`.
3. A `List` created with `Arrays` method `asList` is resizable.
4. Class `Arrays` provides `static` method `sort` for sorting array elements.

Explain the operation of each of the following methods of the `Properties` class:

1. `load`

2. `store`
3. `getProperty`
4. `list`

---

Rewrite lines 1726 in Example 19.4 to be more concise by using the `asList` method and the `LinkedList` constructor that takes a `Collection` argument.

---

Write a program that reads in a series of first names and stores them in a `LinkedList`. Do not store duplicate names. Allow the user to search for a first name.

---

Modify the program of Example 19.20 to count the number of occurrences of each letter rather than of each word. For example, the string `"HELLO THERE"` contains two `H`s, three `E`s, two `L`s, one `O`, one `T` and one `R`. Display the results.

---

Use a `HashMap` to create a reusable class for choosing one of the 13 predefined colors in class `Color`. The names of the colors should be used as keys, and the predefined `Color` objects should be used as values. Place this class in a package that can be imported into any Java program. Use your new class in an application that allows the user to select a color and draw a shape in that color.

---

Write a program that determines and prints the number of duplicate words in a sentence. Treat uppercase and lowercase letters the same. Ignore punctuation.

---

Rewrite your solution to Exercise 17.8 to use a `LinkedList` collection.

---

Rewrite your solution to Exercise 17.9 to use a `LinkedList` collection.

---

Write a program that takes a whole number input from a user and determines whether it is prime. If the number is not prime, display the unique prime factors of the number. Remember that a prime number's factors are only 1 and the prime number itself. Every number that is not prime has a unique prime factorization. For example, consider the number 54. The prime factors of 54 are 2, 3, 3 and 3. When the values are multiplied together, the result is 54. For the number 54, the prime factors output should be 2 and 3. Use `Set`s as part of your solution.

---

Write a program that uses a `StringTokenizer` to tokenize a line of text input by the user and places each token in a `treeSet`. Print the elements of the `treeSet`. [**Note:** This should cause the elements to be printed in ascending sorted order.]

---

The output of Example 19.17 (`PriorityQueueTest`) shows that `PriorityQueue` orders

`Double` elements in ascending order. Rewrite Example 19.17 so that it orders `Double` elements in descending order (i.e., should be the highest-priority element rather than ).

---

- Previous
- Next