sorted. Many common types (such as `String` and `Integer`) implement `Comparable`.

You can implement the `Comparable` interface in your own classes by writing a method `compareTo`.

Self-Check Problems

Section 10.1: ArrayLists

1. What is an `ArrayList`? In what cases should you use an `ArrayList` rather than an array?
   ArrayList is like an Array, but more dynamic
2. Which of the following is the correct syntax to construct an `ArrayList` to store integers?

   a. `ArrayList list = new ArrayList();`
   b. `ArrayList[int] list = new ArrayList[int]();`
   c. `ArrayList list<integer> = new ArrayList<integer>();`
   d. `ArrayList<Integer> list = new ArrayList();`
   e. `ArrayList<Integer> list = new ArrayList<Integer>();`

   ArrayList<String> lst = new ArrayList<>();
   lst.add("It");
   lst.add("was");
   lst.add("a");
   lst.add("stormy");
   lst.add("night");

3. The next five questions refer to the following `String` elements:

   `["It", "was", "a", "stormy", "night"]`

   Write the code to declare an `ArrayList` containing these elements. What is the size of the list? What is its type?
   5 ArrayList<String>

4. Write code to insert two additional elements, `"dark"` and `"and"`, at the proper places in the list to produce the following `ArrayList` as the result:

   `["It", "was", "a", "dark", "and", "stormy", "night"]`

   lst.add(3, "dark")
   lst.add(4, "and")

5. Write code to change the second element's value to `"IS"`, producing the following `ArrayList` as the result:

   `["It", "IS", "a", "dark", "and", "stormy", "night"]`

   lst.set(1, "IS");

6. Write code to remove from the list any `Strings` that contain the letter `"a"`. The following should be the list's contents after your code has executed:

   `["It", "IS", "stormy", "night"]`

   for (int i = lst.size()-1; i >= 0; i++) {
   if (lst.get(i).contains("u")) lst.remove(i);
   }

7. Write code to declare an `ArrayList` holding the first 10 multiples of 2: 0, 2, 4, . . ., 18. Use a loop to fill the list with the proper elements.
   ArrayList<Integer> lst = new ArrayList<>(); for(int i=0; i<10; i++) lst[i] = 2*i;

8. Write a method called `maxLength` that takes an `ArrayList` of `Strings` as a parameter and that returns the length of the longest `String` in the list. If your method is passed an empty `ArrayList`, it should return `0`.
   public static int maxLength (ArrayList<String> lst) { int max = 0; for(String s : lst) { max = Math.max(max, s.length()); } return max; }

9. Write code to print out whether or not a list of `Strings` contains the value `"IS"`. Do not use a loop.
   System.out.println(lst.contains("IS"));

10. Given the `ArrayList` from problem 4, write code to print out the index at which your list contains the value `"stormy"` and the index at which it contains `"dark"`. Do not use a loop.
    SOP("%d & %d%n", lst.indexOf("stormy"), lst.indexOf("dark"));

11. Given the `ArrayList` from problem 4, write a for-each loop that prints the uppercase version of each `String` in the list on its own line.
    for (String s: lst) SOPln(s.toUpperCase());

12. When the code that follows runs on an `ArrayList` of `Strings`, it throws an exception. Why?

```
for (String s : words) {
    System.out.println(s);
    if (s.equals("hello")) {
```

You cannot modify an array while iterating over it

```
        words.add("goodbye");
    }
}
```

13. The code that follows does not compile. Why not? Explain how to fix it.

```
ArrayList<int> numbers = new ArrayList<int>();
numbers.add(7);
numbers.add(19);
System.out.println(numbers);
```

A generic cannot be a primitive. Use the Integer wrapper class to fix it.

14. What is a wrapper class? Describe the difference between an `int` and an `Integer`.

A wrapper class can be used in a generic, as an object an "wraps" the primitive

15. Write the output produced when the following method is passed each of the following lists:

```
public static void mystery1(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i > 0; i--) {
        if (list.get(i) < list.get(i - 1)) {
            int element = list.get(i);
            list.remove(i);
            list.add(0, element);
        }
    }
    System.out.println(list);
}
```

a. `[2, 6, 1, 8]` // [2, 6, 1, 8]
b. `[30, 20, 10, 60, 50, 40]` // [10, 30, 40, 20, 60, 50]
c. `[-4, 16, 9, 1, 64, 25, 36, 4, 49]` // [64, 49, 10, 36, -4, 9, 1, 25, 4]

16. Write the output produced when the following method is passed each of the following lists:

```
public static void mystery2(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        if (i % 2 == 0) {
            list.add(list.get(i));
        } else {
            list.add(0, list.get(i));
        }
    }
    System.out.println(list);
}
```

a. `[10, 20, 30]` // [10, 20, 30, 10, 20, 30]
b. `[8, 2, 9, 7, 4]` // [7, 7, 8, 2, 9, 7, 4, 8, 2, 9]
c. `[-1, 3, 28, 17, 9, 33]` // [3, 17, 33, 3, 17, 33, -1, 3, 28, 17, 9, 33]

17. Write the output produced when the following method is passed each of the following lists:

```
public static void mystery3(ArrayList<Integer> list) {
    for (int i = list.size() - 2; i > 0; i--) {
        int a = list.get(i);
```

```
            int b = list.get(i + 1);
            list.set(i, a + b);
        }
        System.out.println(list);
    }
```

a. [72, 20] // [72, 20]
b. [1, 2, 3, 4, 5, 6] // [21, 20, 16, 15, 11, 6]
c. [10, 20, 30, 40] // [100, 90, 70, 40]

18. Write the output produced when the following method is passed each of the following lists:

```
public static void mystery4(ArrayList<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        int element = list.get(i);
        list.remove(i);
        list.add(0, element + 1);
    }
    System.out.println(list);
}
```

a. [10, 20, 30] // [31, 21, 11]
b. [8, 2, 9, 7, 4] // [5, 8, 10, 3, 9]
c. [-1, 3, 28, 17, 9, 33] // [34, 10, 18, 29, 4, 0]

Section 10.2: The Comparable Interface

19. Describe how to arrange an ArrayList into sorted order. What must be true about the type of elements in the list in order to sort it? Collections.sort(list), the list must be ArrayList of class that implements comparable */)

20. What is a natural ordering? How do you define a natural ordering for a class you've written?
An obvious and fairly universal way of ordering items. You define it using a compareTo method and implementing the Comparable interface

21. Consider the following variable declarations:

```
Integer n1 = 15;
Integer n2 = 7;
Integer n3 = 15;
String s1 = "computer";
String s2 = "soda";
String s3 = "pencil";
```

Indicate whether the result of each of the following comparisons is positive, negative, or 0:

a. n1.compareTo(n2) > 0
b. n3.compareTo(n1) = 0
c. n2.compareTo(n1) < 0
d. s1.compareTo(s2) < 0
e. s3.compareTo(s1) > 0
f. s2.compareTo(s2) = 0

22. Use the `compareTo` method to write code that reads two names from the console and prints the one that comes first in alphabetical order. For example, the program's output might look like the following:

```
Type a name: Tyler Durden
Type a name: Marla Singer
Marla Singer goes before Tyler Durden
```

23. Write code to read a line of input from the user and print the words of that line in sorted order, without removing duplicates. For example, the program output might look like the following:

```
Type a message to sort: to be or not to be that is the question
Your message sorted: be be is not or question that the to to
```

Exercises

1. Write a method called `averageVowels` that takes an `ArrayList` of strings as a parameter and returns the average number of vowel characters (a, e, i, o, u) in all Strings in the list. If your method is passed an empty `ArrayList`, it should return `0.0`.

2. Write a method called `swapPairs` that switches the order of values in an `ArrayList` of strings in a pairwise fashion. Your method should switch the order of the first two values, then switch the order of the next two, then the next two, and so on. If the number of values in the list is odd, the method should not move the final element. For example, if the list initially stores `["to", "be", "or", "not", "to", "be", "hamlet"]`, your method should change the list's contents to `["be", "to", "not", "or", "be", "to", "hamlet"]`.

3. Write a method called `removeEvenLength` that takes an `ArrayList` of strings as a parameter and removes all of the strings of even length from the list.

4. Write a method called `doubleList` that takes an `ArrayList` of strings as a parameter and replaces every string with two of that same string. For example, if the list stores the values `["how", "are", "you?"]` before the method is called, it should store the values `["how", "how", "are", "are", "you?", "you?"]` after the method finishes executing.

5. Write a method called `scaleByK` that takes an `ArrayList` of integers as a parameter and replaces every integer of value *k* with *k* copies of itself. For example, if the list stores the values `[4, 1, 2, 0, 3]` before the method is called, it should store the values `[4, 4, 4, 4, 1, 2, 2, 3, 3, 3]` after the method finishes executing. Zeroes and negative numbers should be removed from the list by this method.

6. Write a method called `minToFront` that takes an `ArrayList` of integers as a parameter and moves the minimum value in the list to the front, otherwise preserving the order of the elements. For example, if a variable called `list` stores `[3, 8, 92, 4, 2, 17, 9]`, the value 2 is the minimum, so your method should modify the list to store the values `[2, 3, 8, 92, 4, 17, 9]`.

7. Write a method called `removeDuplicates` that takes as a parameter a sorted `ArrayList` of strings and eliminates any duplicates from the list. For example, if the list stores the values `["be", "be", "is", "not", "or", "question", "that", "the", "to", "to"]` before the method is called, it should store the values `["be", "is", "not", "or", "question", "that", "the", "to"]` after the method finishes executing. Because the values will be sorted, all of the duplicates will be grouped together. Assume that the `ArrayList` contains only `String` values, but keep in mind that it might be empty.