Our array list of integers can be converted into a generic class that can store a list of any type of objects. The code is similar, but we must make a few changes—for example, when we construct arrays of type `E[]` or compare objects for equality.

An inner class is declared inside the braces of another (outer) class and has access to the state of an object of that outer class. Our final list iterator is an inner class.

Self-Check Problems

Section 15.1: Simple `ArrayIntList`

1. What is the difference between an array list's size and its capacity? What is the relationship between the two values? (Is one always larger or smaller than the other, for instance?)
The size is how many elements are in it, the capacity is how many elements could be init
2. What fields must be included in the `ArrayIntList` class, and why is each field important? Would the class still work correctly if we removed any of these fields?
elementData to store elements and size to store the size. No
3. How would the output of the `Client1` program shown in this section change if each field from `ArrayIntList` were declared `static`?
It wouldn't compile
4. In this version of the list class, what happens if the client adds too many values to fit in the array?
It throws an ArrayIndexOutOfBoundsException
5. Why does the list class use a `toString` method rather than a `print` method?
It's more standard and more versatile
6. We wrote the class to have public methods called `size` (to read the number of elements of the list) and `get` (to access the element value at a specific index). Why is this approach better than declaring the fields (such as `size`) `public`?
Then the client is unable to change the value of size, which would break the program
7. An element can be inserted at the beginning, middle, or end of an array list. Which of the three insertion points is the most computationally expensive, and why? Which is the most expensive location to remove an element from the list?
Beginning is most expensive because it needs to shift everything over
for removing
8. Write methods called `min` and `max` that return the smallest and largest values in the list respectively. For example, if a variable called `list` stores `[11, -7, 3, 42, 0, 14]`, the call of `list.min()` should return `-7` and the call of `list.max()` should return `42`. If the list is empty, the methods should throw an `IllegalStateException`.

```
public int min() {
  if (size == 0) throw new IllegalStateException();
  int min = Integer.MAX_VALUE;
  for (int i = 0; i < size; i++) min = Math.min(min, elementData[i]);
  return min;
}
```

For max just switch 'min' and 'max'

Section 15.2: A More Complete `ArrayIntList`

9. Describe the overall preconditions placed on the list class in this section. What assumptions do we make about how clients will use the list?
They have to not call an index outside the list
10. What is the purpose of the `checkIndex` method? Where is it called in the list class? Describe a way that the client can utilize an `ArrayIntList` that will be caught by `checkIndex`.
It is to make sure the client isn't accessing data outside the list. It is called anywhere the client can provide an index
11. What is the purpose of the `checkCapacity` method? Where is it called in the list class? Describe a way that the client can utilize an `ArrayIntList` that will be caught by `checkCapacity`.
I didn't include this but the client could try to set the value of -1 in my version, but it throws an error if the list's size exceeds its capacity
12. Once we check thoroughly for preconditions in the code, what data invariants can we now assume about the list?
size <= elementData.length
size isn't between 0 and size
You will never get or set an array element
You will never throw an ArrayIndexOutOfBoundsException
13. Why do we bother to add the `contains`, `isEmpty`, and `remove` methods to the list class, when the client can already perform this same functionality with the `indexOf`, `size`, and `remove` methods, respectively?
This is to add convenience for the client. Technically, the client could do all of this themselves, but our job is to make their job easier by making specific tasks easier.

Section 15.3: Advanced Features

14. When this new version of the class fills to its capacity, it resizes. How much does it grow? Why choose this growth rate, rather than increasing the capacity by a single element or other constant amount?
A single element would cause performance issues if you added often. This program increases to double its size, which isn't huge, but also helps performance.

15. What is the benefit of adding an iterator to the list class?
It allows the client to more easily cycle through values

16. What state does the array list iterator store?
The position and whether it is ok to remove the last value

17. How does the array list iterator know if there are more elements left to examine? What does it do if the client tries to examine a next element but there are none left to examine? It compares the position to the size. It throws an error

18. What is a precondition of the iterator's `remove` method? How does the iterator enforce this precondition, and what does it do if the precondition is violated? There is has not been called since the last call to next(). It keeps track of if it's ok to remove, and throws an error if it's not

19. Write a method called `sum` that returns the sum of all values in the list. For example, if a variable called `list` stores `[11, -7, 3, 42, 0, 14]`, the call of `list.sum()` should return `63`. If the list is empty, `sum` should return `0`.

```
public int sum() {
    int sum = 0;
    for (int i = 0; i < size; i++)
        sum += elementData[i];
    return sum;
```

20. Write a method called `average` that returns the average of the values in the list as a real number. For example, if a variable called `list` stores `[11, -7, 3, 42, 0, 14]`, the call of `list.average()` should return `10.5`. If the list is empty, `average` should return `0.0`.

```
public double average() {
    return size == 0 ? 0.0 : sum() / size;
}
```

Section 15.4: ArrayList<E>

21. What problem do we encounter when we try to construct an array of type `E`? How do we resolve this problem?
It's illegal. We construct an Object[] and perform an unchecked cast to E[]

22. Since our list stores an unfilled array, the empty elements were filled with the value `0` when our array was full of integers. What value occupies the empty cells when our list stores values of type `E`?
null

23. What changes need to be made to the `indexOf` method to search for objects of type `E` in the new list class, and why are these changes necessary? use the equals method instead of == because == works in an unreliable way for objects

24. What is an annotation? How are annotations useful in writing our `ArrayList<E>` class?
Annotations mark metadata for the compiler. Here it is used to get rid of the unchecked cast warning

25. Why is it important to set empty elements to `null` when we are clearing or removing from the list of type `E`, when we didn't need to clear out these elements in the previous `ArrayIntList`? So the unused elements aren't taking up unnecessary space in the heap

26. What is one benefit of making the list iterator into an inner class?
It allows it to access private fields of ArrayList.

Exercises

Each of the following exercises is a method to be added to the `ArrayIntList` class from this chapter.

1. Write a method called `lastIndexOf` that accepts an integer as a parameter and returns the index in the list of the last occurrence of that value, or -1 if the value is not found in the list. For example, if the list stores `[1, 18, 2, 7, 18, 39, 18, 40]`, then the last index of 18 is 6 and the last index of 3 is -1.

2. Write a method called `indexOfSubList` that accepts another list *L* as a parameter and returns the starting index of where *L* first appears in this list, or -1 if it is not found. All elements of *L* must appear in sequence and in the same order. For example, if variables called `list1` and `list2` store `[11, -7, 3, 42, 0, 14]` and `[3, 42, 0]`, respectively, the call of `list1.indexOfSubList(list2)` should return `2`.

3. Write a method called `replaceAll` that accepts two integer values as parameters and replaces all occurrences of the first value in the list with the second value. For example, if a variable called `list` stores `[11, -7, 3, 42, 3, 0, 14, 3]`, the call of `list.replaceAll(3, 999);` should change the list to store `[11, -7, 999, 42, 999, 0, 14, 999]`.