Debugging Arrays and Other Collection Types

by Sophia

WHAT'S COVERED

In this lesson, you will learn about common problems that arise when working with arrays and collection types used in Java. Specifically, this lesson covers:

1. Array Index Errors and Debugging
2. Debugging Other Collection Types

1. Array Index Errors and Debugging

When working with arrays and accessing items in an ArrayList, out-of-range indices are one of the most common **array index errors** that can occur. It is important to remember that the first element in an array always has the index 0.

KEY CONCEPT

Array indices are never negative.

The "off-by-one" error is another common error in arrays. This error is generated when forgetting that the last index in an array is always equal to the size of the array minus 1. If an array has the length 5, the indices in the array are 0, 1, 2, 3, and 4 (not 1, 2, 3, 4, and 5). Java will produce errors rather than try to access an invalid memory location, as can happen with languages like C and C++, if the programmer is not careful.

Consider the following sample:

```
class ArrayIndexError {
  public static void main(String[] args) {
    String[] choices = {"coffee", "tea", "water"};
    System.out.println("1. " + choices[1]);
    System.out.println("2. " + choices[2]);
    System.out.println("3. " + choices[3]);
  }
}
```

The results should look like this:

```
1. tea
2. water
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at ArrayIndexError.main(ArrayIndexError.java:6)
```

In the introduction to arrays tutorial, you learned that try and catch blocks can be used to deal with out-of-bounds indices. Although, this case is a programming error. The code should be corrected rather than relying on try and catch blocks. The list in the output above starts with the second element in the array because the code accesses choices[1] for the first item rather than choices[0], so the code produces an inaccurate list in addition to causing a runtime error. The error, an ArrayIndexOutOfBounds exception, is caused by trying to read past the end of the array, choices[3], when the last element in the array is choices[2].

Here is what it looks like if the array is accessed correctly when the list of choices is printed but the input is used incorrectly:

```
import java.util.Scanner;

class ArrayIndexError {
  public static void main(String[] args) {
    String[] choices = {"coffee", "tea", "water"};
    Scanner input = new Scanner(System.in);
    System.out.println("1. " + choices[0]);
    System.out.println("2. " + choices[1]);
    System.out.println("3. " + choices[2]);
    System.out.print("Enter selection number: ");
    int selection = input.nextInt();
    // Next line produces the wrong result. It should be use
    // choices[selection - 1]
    System.out.println("You selected " + choices[selection]);
  }
}
```

This sample shows three runs of the program. Two of the trials do not produce any error messages; however, the outcome is wrong. The third run results in an exception, as seen in the output below:

```
1. coffee
2. tea
3. water
Enter selection number: 2
You selected water

1. coffee
2. tea
3. water
Enter selection number: 1
You selected tea

1. coffee
2. tea
3. water
Enter selection number: 3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at ArrayIndexError.main(ArrayIndexError.java:14)
```

KEY CONCEPT

You must always be mindful of such "off-by-one" errors and code carefully to avoid them.

- The first index in an array always has the value 0.
- The last index in an array always has the value length minus 1 (where length is the number of elements in the array).

When assessing user input to determine which element in an array needs to be accessed, it is important to always verify that the input is in the correct range. A numbered menu list almost always begins with 1; therefore, you will need to remember to subtract 1 when checking that the selection is in bounds, and when accessing the array element via its index.
When using a two-dimensional array, remember that the first index refers to the row, and the second index refers to the column. Keep in mind that a program relying on user input may not have the inputs in this order.

HINT

When running into problems with an array in your code, it can be helpful to add some temporary output statements that use Arrays.toString() (for 1D arrays) and Arrays.deepToString() (for 2D and 3D arrays) to display the contents of an array on the screen. This will help you see what data is where in the array, at a particular point in the program. Do not forget to remove this extra output when you are finished debugging.

TERM TO KNOW

Array Index Error
An error that occurs when a programmer accesses the wrong element in an array (or ArrayList collection) or tries to access an element that does not exist.

---

2. Debugging Other Collection Types

There are similarities between a plain, single-dimensional array and an `ArrayList`. This also means that some similar problems can arise.

While the size of an `ArrayList` can grow and be reduced, as items are added with the `add()` method and removed with the `remove()` method), the `get()` method needs a valid index to access an item in the `ArrayList`. The first item in an ArrayList always has the index 0, and the last item index is equal to the `ArrayList`'s size minus 1.

Below are problems similar to the array example above. This code prints out the list of beverages using 1 to try to access the first item and 3 to access the last, which doesn't work well:

```
import java.util.ArrayList;

class ArrayListIndexError {
  public static void main(String[] args) {
    ArrayList<String> choices = new ArrayList<>();
    choices.add("coffee");
    choices.add("tea");
    choices.add("water");
    System.out.println("1. " + choices.get(1));
    System.out.println("2. " + choices.get(2));
    System.out.println("3. " + choices.get(3));
```

```
  }
}
```

Running this code in a file named ArrayListIndexError.java produces this output:

```
1. tea
2. water
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 3 out of bounds for length 3
    at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
    at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
    at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:266)
    at java.base/java.util.Objects.checkIndex(Objects.java:359)
    at java.base/java.util.ArrayList.get(ArrayList.java:427)
    at ArrayListIndexError.main(ArrayIndexError.java:11)
```

As with the array example, we can see that the first item in the list is "tea" rather than the correct value "coffee," since this line actually accesses the second item in the `ArrayList` rather than the first:

EXAMPLE

```
System.out.println("1. " + choices.get(1));
```

This line needs to get the item at index 0 instead:

EXAMPLE

```
System.out.println("1. " + choices.get(0));
```

The output shows "water" next, since the code that produces this output reads:

EXAMPLE

```
System.out.println("2. " + choices.get(2));
```

In an ArrayList of three items, though, index 2 points to the last item of the three, not the second. Finally, consider the following statement:

EXAMPLE

```
System.out.println("3. " + choices.get(3));
```

This example results in an **IndexOutOfBoundsException**, since the last index in an `ArrayList` of three items is 2. When checking the size of an `ArrayList`, remember that you need to use the `size()` method, not the array's `length` property.

If the program relies on user input to access an item in the ArrayList, it is important to carry out appropriate checking and translation. This is important, since the menu item will be 1 greater than the actual index in the ArrayList, before trying to get a value from the ArrayList.

Here is a version of the code that accesses the items in the `ArrayList` correctly and checks to make sure the selection is within the correct range. While it would be possible to use try and then `catch` an `IndexOutOfBoundsException`, it is usually best to use selection statements to deal with an out-of-bounds selection.

TRY IT

Directions: Type this code into a file named ArrayListIndexRange.java in the IDE and try running it:

```
import java.util.ArrayList;
import java.util.Scanner;

class ArrayListIndexRange {
  public static void main(String[] args) {
    ArrayList<String> choices = new ArrayList<>();
    // Add as many choices as needed, ArrayList grows to fit.
    choices.add("coffee");
    choices.add("tea");
    choices.add("water");
    System.out.println("1. " + choices.get(0));
    System.out.println("2. " + choices.get(1));
    System.out.println("3. " + choices.get(2));

    Scanner input = new Scanner(System.in);
    System.out.print("Enter a selection #: ");
    int selection = input.nextInt();
    // Subtract 1 from menu choice to get index
    int index = selection - 1;
    if(index >= 0 && index < choices.size()) {
      System.out.println("You selected " + choices.get(selection - 1) + ".");
    }
    else {
      System.out.println("Not a valid selection");
    }
  }
}
```

Here is the result of three sample runs of the program:

```
1. coffee
2. tea
3. water
Enter a selection #: 3
You selected water.

1. coffee
2. tea
3. water
Enter a selection #: 0
Not a valid selection
```

```
1. coffee
2. tea
3. water
Enter a selection #: 4
Not a valid selection
```

REFLECT

This program shows how a combination of selection statements can be used when working with a collection to avoid exceptions and program crashes if the user makes an incorrect selection. A message about the problem is a better user experience than a crash caused by an unhandled exception. Using selection statements like this is better than using try and catch blocks. Why do you think this might be the case?

TRY IT

Directions: When working with `ArrayLists`, you can also add temporary output statements that use the `toString()` method to show the current contents:

```
import java.util.ArrayList;
import java.util.Scanner;

class ArrayListIndexRange {
  public static void main(String[] args) {
    ArrayList<String> choices = new ArrayList<>();
    // Add as many choices as needed, ArrayList grows to fit.
    choices.add("coffee");
    choices.add("tea");
    choices.add("water");
    // A temporary output statement for debugging
    System.out.println("DEBUG: ArrayList = " + choices.toString());
    System.out.println("1. " + choices.get(0));
    System.out.println("2. " + choices.get(1));
    System.out.println("3. " + choices.get(2));

    Scanner input = new Scanner(System.in);
    System.out.print("Enter a selection #: ");
    int selection = input.nextInt();
    // Subtract 1 from menu choice to get index
    int index = selection - 1;
    if(index >= 0 && index < choices.size()) {
      System.out.println("You selected " + choices.get(selection - 1) + ".");
    }
    else {
      System.out.println("Not a valid selection");
    }
  }
}
```

Note the "Debug" line in the output that allows the programmer to confirm that the collection contains the expected values:

```
DEBUG: ArrayList = [coffee, tea, water]
1. coffee
2. tea
3. water
Enter a selection #: 2
You selected tea.
```

REFLECT

This code shows how a temporary output statement and a collection's toString() method can be used to confirm the contents of a collection when developing and debugging an application. Of course, such temporary output statements should be removed when development is done, since they are likely to confuse someone using the program.

HINT

Be sure to remove such extra output statements when you're done with the debugging process.

When working with `HashSet` and `HashMap` collections, such index-out-of-bounds errors will not occur, since these collections don't allow access to individual elements by index. Almost all collection types use temporary output statements that make use of `toString()`. They can be helpful for checking the contents of the collection and making sure that they match the programmer's expectations.

TERM TO KNOW

IndexOutOfBoundsException
An exception (runtime error) thrown when code tries to access an element that is outside the bounds of an array.

SUMMARY

In this lesson, you learned about common problems that can arise when working with arrays and collections in Java. These include **array index errors**. "Off-by-one" errors are a common problem. You learned that it is important to keep in mind that while people generally start counting at 1, computers and programming languages usually start with 0. Finally, you learned that when accessing data in arrays and collections and **when debugging other collections**, it is important to check that the index is in bounds. This includes cases where the index is based on user input.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Array Index Error
An error that occurs when a programmer accesses the wrong element in an array (or ArrayList collection) or tries to access an element that does not exist.

IndexOutOfBoundsException
An exception (runtime error) thrown when code tries to access an element that is outside the bounds of an array.