

Unit 6: Array

Unit 6 covers arrays, Java's fixed-size ordered collections. You will learn to create arrays, read and write elements by index, walk through them with loops, and build the standard search-and-modify algorithms that Units 7 and 8 are built on.

AP Computer Science A

Array

About 14 minutes to read

How to use this guide

Read it in order the first time because the topics build on each other. Creation and access set up the index rules, traversal turns those rules into loops, the enhanced for loop is a special case of traversal, and the algorithms in 6.4 are traversal with a job to do. Exam questions often hand you a loop and ask you to trace it or spot the error, so trace every example in this guide by hand before you read the explanation.

After the first read, use the trap boxes and the tables to review the distinctions that exam questions test most often. Finish with the practice questions, then complete the recall check on the last page out loud and note any items you cannot explain yet.

What this unit is worth. Unit 6 is about 10 to 12.5 percent of the AP CSA exam. Array traversal is the foundation for ArrayList (Unit 7) and 2D arrays (Unit 8), and array algorithms appear in FRQ 1 or 3 most years. The off-by-one errors you fix here stay fixed everywhere else.

6.1 Array Creation and Access

An **array** holds a fixed number of values of one type, stored in order. Each position has an **index**, and indexing starts at 0. Once you create an array, its length never changes.

To create an array, give the type, the brackets, and the size.

```
int[] scores = new int[5];
```

This creates five slots, numbered 0 through 4. Every slot is filled immediately with the default value for the type, which for `int` is 0. If you already know the values, use an **initializer list** instead.

```
int[] nums = {3, 7, 1};
```

This creates an array of length 3 holding 3, 7, and 1. You can only use the short form when you declare the array.

Read or write an element with its index in brackets. `nums[0]` is 3, and `nums[0] = 10;` changes the first slot to 10. The valid indices run from 0 to `arr.length - 1`. Using an index outside that range throws an **ArrayIndexOutOfBoundsException** when the program runs.

The size of an array comes from `arr.length`, which is a **field**, not a method. There are no parentheses. This is different from strings, where you call `length()` with parentheses. Writing `arr.length()` does not compile.

ARRAY TYPE	DEFAULT VALUE IN EACH NEW SLOT
int, long, and other integer types	0
double and other decimal types	0.0
boolean	false
Any reference type, such as String	null

Trap. A new array is never empty. Every slot already holds a default value, so `new int[5]` gives you five 0s. For a `String[]`, every slot holds `null`, and calling a method on one of those slots throws a `NullPointerException`, which is a different error from going out of bounds.

6.2 Traversing Arrays

Traversing an array means visiting each element in order, usually with a for loop. The standard pattern starts the index at 0, continues while the index is strictly less than the length, and adds 1 each time.

```
for (int i = 0; i < arr.length; i++) {  
    System.out.println(arr[i]);  
}
```

The bound is `i < arr.length` because the last valid index is `arr.length - 1`. The loop must stop before `i` reaches `arr.length`. This hand trace shows each step of a sum.

```
int[] vals = {4, 9, 2};  
int total = 0;  
for (int i = 0; i < vals.length; i++) {  
    total = total + vals[i];  
}  
// i = 0: total = 0 + 4 = 4  
// i = 1: total = 4 + 9 = 13  
// i = 2: total = 13 + 2 = 15  
// i = 3: 3 < 3 is false, loop ends. total is 15.
```

Two small changes break the pattern in two different ways. Changing the condition to `i <= arr.length` lets `i` reach `arr.length`, and `arr[arr.length]` throws an `ArrayIndexOutOfBoundsException` on the final pass. Starting at `i = 1` instead of 0 compiles and runs fine but silently skips the first element. The crash is loud. The skipped element is quiet, and quiet bugs cost more points.

Sometimes you need to walk backward. Start at the last valid index and count down to 0.

```
for (int i = arr.length - 1; i >= 0; i--) {  
    System.out.println(arr[i]);  
}
```

Trap. The most common runtime error in this unit is `ArrayIndexOutOfBoundsException`, and it almost always comes from a loop bound. Read every traversal bound as a question. Does the loop touch index 0? Does it stop before `arr.length`? If you cannot answer yes to both, the loop is wrong.

6.3 Enhanced for Loop for Arrays

The **enhanced for loop** visits every element without an index variable. The loop variable takes each element's value in order, one per pass.

```
int[] vals = {4, 9, 2};
for (int x : vals) {
    System.out.println(x);
}
// prints 4, then 9, then 2
```

The loop variable is a copy of each element, not a window into the array. Assigning to it changes nothing in the array.

```
int[] nums = {1, 2, 3};
for (int x : nums) {
    x = x * 2;
}
// nums is still {1, 2, 3}. The doubling was lost.
```

Use the enhanced for loop when you only need to read each element once, in order. Reach for a regular indexed loop instead when you need to change elements, know the index, compare an element with its neighbor, or walk backward.

Trap. Exam questions show an enhanced for loop that tries to modify an array and ask what the array holds afterward. The answer is always the original array. If the code assigns to the loop variable, that is the signal the array is untouched.

6.4 Developing Algorithms Using Arrays

Most array algorithms are the traversal pattern from 6.2 with a job added inside the loop. Learn the shapes below and you can rebuild them on the exam without memorizing code line by line.

A **running total** starts an accumulator at 0 and adds each element. Dividing by the length gives the average.

```
int sum = 0;
for (int i = 0; i < arr.length; i++) {
    sum = sum + arr[i];
}
double avg = (double) sum / arr.length;
```

To find the **maximum**, start with the first element and keep whichever is larger. The loop can start at 1 because index 0 is already held.

```
int max = arr[0];
for (int i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
        max = arr[i];
    }
}
```

Counting matches is the same idea with a counter and a test.

```
int count = 0;
for (int i = 0; i < arr.length; i++) {
    if (arr[i] > 90) {
        count++;
    }
}
```

Trap. The max pattern assumes the array is not empty, since it reads `arr[0]` before the loop. The sum and count patterns work on any array because their starting values are safe. Match the starting value to the job. A sum starts at 0. A max starts at the first element.

Shifting elements

To shift every element one slot to the right, save the last element first, because the loop will overwrite it. Then walk backward so each element moves into the slot ahead of it. Trace this on {1, 2, 3} before reading on.

```
int[] arr = {1, 2, 3};
int temp = arr[arr.length - 1]; // temp = 3
for (int i = arr.length - 1; i > 0; i--) {
    arr[i] = arr[i - 1];
}
arr[0] = temp; // arr is now {3, 1, 2}
```

Walking forward here would be a mistake. Each element would copy its left neighbor after that neighbor had already moved, and the whole array would fill with copies of the first element.

Reversing an array

To reverse in place, swap pairs from the outside in and stop halfway. Going past the middle swaps everything back to where it started.

```
int[] nums = {1, 2, 3, 4};
for (int i = 0; i < nums.length / 2; i++) {
    int temp = nums[i];
    nums[i] = nums[nums.length - 1 - i];
    nums[nums.length - 1 - i] = temp;
}
// nums is now {4, 3, 2, 1}
```

PATTERN	WHAT THE LOOP DOES
Sum	Accumulator starts at 0. Add each element.
Average	Sum divided by arr.length.
Maximum or minimum	Start at arr[0]. Keep the larger or smaller.
Count matches	Counter starts at 0. Increment when the test passes.
Shift	Save one end. Move the rest. Put the saved value back.
Reverse	Swap outside pairs inward. Stop at arr.length / 2.

Confusions That Cost Points

PAIR	HOW TO KEEP THEM STRAIGHT
<code>arr.length</code> vs <code>arr.length()</code>	<code>arr.length</code> is a field with no parentheses. <code>length()</code> with parentheses is a String method. Writing <code>arr.length()</code> does not compile.
<code>i < arr.length</code> vs <code>i <= arr.length</code>	The last valid index is <code>arr.length - 1</code> , so the condition must be strict. Using <code><=</code> walks one step past the end and throws <code>ArrayIndexOutOfBoundsException</code> .
Indices start at 0, not 1	The first element is <code>arr[0]</code> . Starting a traversal at 1 compiles and runs but silently skips the first element.
Enhanced for vs indexed loop	The enhanced for loop gives you each value but no index, and it cannot change the array. If you need to write elements or know positions, use an indexed loop.
Default values are real values	A new <code>int</code> array holds 0s and a new <code>String</code> array holds nulls. Calling a method on a null element throws <code>NullPointerException</code> , which is a different error from going out of bounds.
Assigning arrays copies the reference	<code>int[] b = a</code> makes <code>b</code> point at the same array, not a new one. Changes through <code>b</code> are visible through <code>a</code> . To copy, build a new array and fill it in a loop.

Practice Questions

Original questions written for this guide in the style of the AP exam. Answers and explanations are on the next page, so complete the questions before checking them.

1. Consider the following code segment.

```
int[] a = {2, 5, 1, 4};
int s = 0;
for (int i = 0; i < a.length; i++) {
    if (a[i] % 2 == 0) {
        s = s + a[i];
    }
}
System.out.println(s);
```

What is printed as a result of executing the code segment?

- 1. 6
- 2. 12
- 3. 3
- 4. 0

2. Consider the following code segment.

```
int[] b = {3, 1, 4, 1, 5};
int m = b[0];
for (int i = 1; i < b.length; i++) {
    if (b[i] > m) {
        m = b[i];
    }
}
System.out.println(m);
```

What is printed as a result of executing the code segment?

- 1. 3
- 2. 4
- 3. 5
- 4. 1

3. An array is declared as `int[] c = new int[4];` Which of the following loops throws an `ArrayIndexOutOfBoundsException`?

- 1. `for (int i = 0; i < c.length; i++) c[i] = i;`
- 2. `for (int i = 0; i <= c.length; i++) c[i] = i;`
- 3. `for (int i = c.length - 1; i >= 0; i--) c[i] = i;`
- 4. `for (int i = 1; i < c.length; i++) c[i] = i;`

4. Consider the following code segment.

```
int[] d = {10, 20, 30};  
for (int x : d) {  
    x = x + 5;  
}  
System.out.println(d[0] + d[1] + d[2]);
```

What is printed as a result of executing the code segment?

1. 60
2. 75
3. 65
4. 0

5. Consider the following code segment.

```
int[] e = new int[3];  
System.out.println(e[0] + e[2]);
```

What is printed as a result of executing the code segment?

1. 0
2. Nothing, because the code does not compile
3. It throws a `NullPointerException`
4. It throws an `ArrayIndexOutOfBoundsException`

6. Consider the following code segment.

```
int[] f = {1, 2, 3, 4};  
for (int i = 0; i < f.length / 2; i++) {  
    int t = f[i];  
    f[i] = f[f.length - 1 - i];  
    f[f.length - 1 - i] = t;  
}  
System.out.println(f[0] + f[3]);
```

What is printed as a result of executing the code segment?

1. 4
2. 5
3. 10
4. 8

7. Which of the following statements about finding the number of elements in an array `arr` is true?

1. `arr.length` is a method, so you write `arr.length()`.
2. `arr.length` is a field, so you write `arr.length` with no parentheses.
3. `arr.length` and `String.length()` work the same way.
4. You must track the size in a separate variable because arrays cannot report it.

8. Consider the following code segment.

```
String[] words = new String[2];  
System.out.println(words[0].length());
```

What happens as a result of executing the code segment?

1. 0 is printed
2. null is printed
3. A `NullPointerException` is thrown
4. An `ArrayIndexOutOfBoundsException` is thrown

Answer Key

- 1. A.** Trace it by hand. $i = 0$: $a[0]$ is 2, which is even, so s becomes 2. $i = 1$: $a[1]$ is 5, odd, skipped. $i = 2$: $a[2]$ is 1, odd, skipped. $i = 3$: $a[3]$ is 4, even, so s becomes 6. The loop ends and 6 is printed. B adds all four elements and ignores the even test. C adds the indices of the even elements ($0 + 3$) instead of the values. D assumes the if test never passes.
- 2. C.** This is the maximum pattern. m starts at $b[0] = 3$. $i = 1$: 1 is not greater than 3. $i = 2$: 4 is greater, so m becomes 4. $i = 3$: 1 is not greater. $i = 4$: 5 is greater, so m becomes 5. The loop ends and 5 is printed. A reports the starting value $b[0]$ as if the loop never improved it. B stops paying attention after 4 appears. D is the minimum, the result of flipping the comparison to $<$.
- 3. B.** The array has valid indices 0 through 3. Loop B runs i from 0 to 4 inclusive, and $c[4]$ throws an `ArrayIndexOutOfBoundsException` on the final pass. A is the standard correct traversal. C walks backward through the valid indices correctly. D skips index 0 but every index it touches is legal, so it runs without error.
- 4. A.** The enhanced for loop variable x is a copy of each element. Assigning $x = x + 5$ changes the copy and leaves the array alone, so d is still $\{10, 20, 30\}$ and the sum printed is 60. B assumes the $+5$ stuck, giving $15 + 25 + 35 = 75$. C adds 5 to the total only once. D assumes the loop destroyed the array contents.
- 5. A.** Every slot of a new int array holds the default value 0, so $e[0] + e[2]$ is $0 + 0 = 0$. B assumes unassigned slots are a compile error, but default values mean every slot is assigned from the start. C applies the reference-type default null to an int array, where the default is 0. D is wrong because indices 0 and 2 are both valid for an array of length 3.
- 6. B.** Trace the swap. $i = 0$: swap $f[0]$ and $f[3]$, giving $\{4, 2, 3, 1\}$. $i = 1$: swap $f[1]$ and $f[2]$, giving $\{4, 3, 2, 1\}$. $i = 2$: $2 < 2$ is false, so the loop ends. $f[0] + f[3]$ is $4 + 1 = 5$. A reports only $f[0]$ of the reversed array and drops the second term. C adds all four elements instead of the two the question asks for. D is the result of swapping without a temp variable, which overwrites $f[0]$ before its value is saved.
- 7. B.** `arr.length` is a field, so you write it with no parentheses. A reverses the rule and calls it a method. C is wrong because `String length` really is a method with parentheses, so the two do not work the same way. D is wrong because arrays report their own size through the length field.
- 8. C.** Every slot of a new String array holds null. The code calls `.length()` on `words[0]`, which is null, and calling a method on null throws a `NullPointerException`. A confuses the int default 0 with the reference default null. B would be the output of printing `words[0]` itself, but the code calls a method on it first. D is wrong because index 0 is valid for an array of length 2, so nothing goes out of bounds.

When you check your answers, note which distinction each miss came from. Make a flashcard for that distinction and drill it spaced out over the next few days instead of rereading the whole section. If you missed one of these questions, the same distinction is worth practicing again in Rycal, where the AP Computer Science A Arrays deck has flashcards for it and more practice questions use the same kinds of traps.

One-Page Recall Check

Say each answer out loud before you look back, and mark the ones you cannot finish. Anything you cannot say out loud yet belongs in your flashcard deck. In Rycal, add those items to the AP Computer Science A Arrays deck and let spaced review bring them back over the next few days.

- Declare and create an int array of size 5 two different ways.
- State the index range of an array of length n and name the error thrown outside it.
- Explain why arr.length has no parentheses and what arr.length() would do.
- List the default values for int, double, boolean, and String arrays.
- Write a forward traversal loop and a backward traversal loop from memory.
- Explain what goes wrong with $i \leq \text{arr.length}$ and with starting at $i = 1$.
- State the two things an enhanced for loop cannot do.
- Write the maximum-finding pattern from memory, including where the loop starts.
- Trace a shift-right-by-one loop on {1, 2, 3} by hand.
- Explain why the reverse loop stops at $\text{arr.length} / 2$.
- Describe what `int[] b = a` does and does not do.

Where to go next. Turn every missed item above into flashcards and drill them spaced out over several days rather than in one sitting. In Rycal, open the AP Computer Science A Arrays deck. The deck covers the terms in this guide, and its practice questions target the same traps named here. If you have a test date, add it in the Test Planner. You can also start your next review with a Brain Dump, then check what you missed against this guide.

Key terms for this unit

Array, Index, Element, Array initializer list, arr.length, ArrayIndexOutOfBoundsException, Traversal, Off-by-one error, Enhanced for loop, Accumulator, Default value, NullPointerException, Reference.

About this guide. Written for Rycal and aligned to the College Board AP Computer Science A course framework, Unit 6. All questions and explanations are original Rycal writing. Rycal is independent and is not affiliated with or endorsed by the College Board.