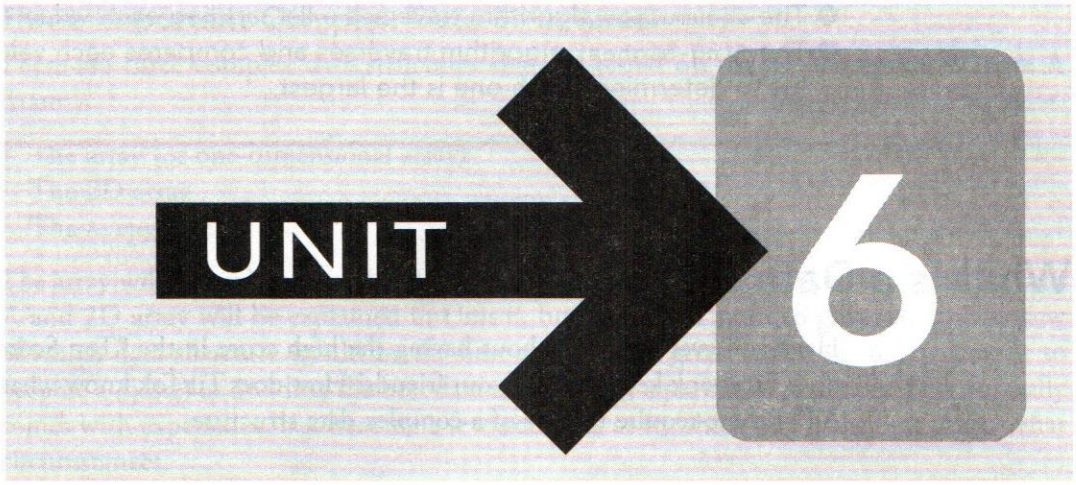


Array

IN THIS UNIT

Summary: Sometimes a simple variable is not enough. To solve certain problems, you need a list of variables or objects. The array, the two-dimensional array, and the `ArrayList` are three complex data structures that are tested on the AP Computer Science A Exam. `ArrayList` and 2D array will be discussed in Units 7 and 8. You must learn the advantages and disadvantages of each of these data structures, know how to work with them, and be able to choose the best one for the job at hand. Programmers use algorithms extensively to write the code that will automate a process in a computer program. The algorithms described in this unit are some of the fundamental algorithms that you should know for the AP Computer Science A Exam.

Key Ideas

- An array is a data structure that can store a list of variables of the same data type.
- An array is not resizable once it is created.
- To traverse a data structure means to visit each element in the structure.
- The enhanced for loop (`for-each` loop) is a special looping structure that can be used by either arrays or `ArrayLists`.
- An algorithm is a set of steps that when followed complete a task.
- Pseudocode is a hybrid between an algorithm and actual Java code.
- Efficient code performs a task in the fastest way.

- The `accumulate` algorithm traverses a list, adding each value to a total.
- The `find-highest` algorithm traverses and compares each value in the list to determine which one is the largest.

What Is a Data Structure?

Have you ever dreamed about having the high score in the "Top Scores" of a game? How does Facebook keep track of your friends? How does TikTok know what video to loop next? All of these require the use of a complex data structure.

Simple Data Structure Versus Complex Data Structure

As programmers, we use **data structures** to store information. A primitive variable is an example of a **simple data structure** as it can store a single value to be retrieved later. A variable that can store a list of other variables is called a **complex data structure**.

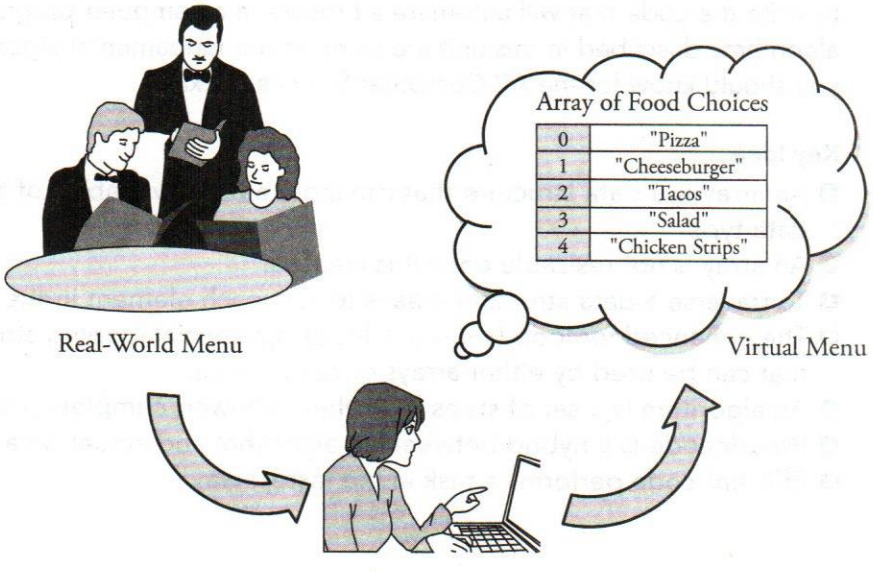
Simple Data Structure
(can hold one thing)

`String myGame`
`"Mario Kart"`

Complex Data Structure
(can hold a list of things)

`String[] myGames`
`"Mario Kart"`
`"Minecraft"`
`"Pokemon"`
`"Tetris"`

Thinking of ways to represent physical objects by using virtual objects can be challenging. Really awesome programmers have the ability to visualize complex real-world situations and design data structures to represent these physical objects in a virtual way. In the drawing below, the programmer is trying to figure out a way to represent the choices from the lunch menu in a program by creating a complex data structure.



Three Important Complex Data Structures

There are three complex data structures that are tested on the AP Computer Science A Exam:

- The array (or one-dimensional array)
- The 2D array
- The `ArrayList`

The array will be explained in detail later in this unit, `ArrayList` will be explained in Unit 7, and 2D array will be explained in Unit 8, but for now, I want to give you an overview of which data structure would be the best choice for a given situation. You will want to become a master at deciding the best data structure for a particular situation. This normally comes with experience, but here are a few examples as to which one is the best under what circumstances.

Situation	Recommendation	Justification
A program helps an elevator know what floor it is on.	Array	The number of floors is fixed. The elevator can go to any floor by knowing what number it is.
Facebook keeps track of how many friends you have.	`ArrayList`	The number of friends you have on Facebook may increase or decrease. You are allowed to add or remove anyone at any time regardless of where they are in the list.
Your program is going to simulate chess, Candy Crush, or 2048.	2D Array	Each of these games can be simulated on either a square or rectangular grid in which the row and column are used to find out what is in each cell.
A cell phone keeps track of text messages.	`ArrayList`	The number of text messages on a cell phone can increase or decrease. You can even delete all of the messages.
A program keeps track of what classes you have each period of the school day.	Array	The number of class periods in the school day is fixed. Each period is assigned a value (1st hour is math, 2nd hour is science, etc.).

The Array

Definition of an Array

The non-programming definition of an array is "an impressive display of a particular thing." We use the word in common language when we say things like, "Wow, look at the wide *array* of phone cases at the mall!" It means that there is a group or list of things that are of the same kind.

An **array** (**one-dimensional array**) in Java is a **complex data structure** because it can store a **list** of primitives or objects. It also stores them in such a way that each of the items in the list can be referenced by an **index**. This means that the array assigns a number to each of the slots that holds each of the values. By doing this, the programmer can easily find the value that is stored in a slot, change it, print it, and so on. The values in the array are called the **elements** of the array. An array can hold any data type, primitive or object.

Declaring an Array and Initializing It Using a Predefined List of Data

There are two common ways to create an array on the AP Computer Science A Exam. The first uses an initializer list. This technique is used when you already know the values that you are going to store in the list and you have no reason to add any more items. **A pair of brackets**, [ ], tells the compiler to create an array, not just a regular variable.

General Form for Creating an Array Using a Predefined List of Data

```
dataType[] nameOfArray = {dataValue1, dataValue2, dataValue3, . . . };
```

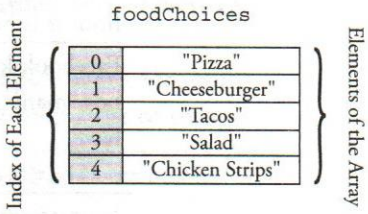
Notice the pair of brackets [ ] after dataType. The brackets signify this variable is a reference to an array object.

Example

Declare an array of `String` variables that represent food choices at a restaurant:

```
String[] foodChoices = {"Pizza", "Cheeseburger", "Tacos", "Salad",
    "Chicken Strips"};
```

The graphic at the right is a visual representation of the foodChoices array. The strings representing the actual food choices are the **elements** in the array (Pizza, Cheeseburger, etc.) and each slot is assigned a number, called the **index** (0, 1, 2, etc.). The index is a number that makes it easy for the programmer to know what element is placed in what slot. The first index is always zero. The last index is always one less than the length of the array.



How to "Speak" Array

When talking about elements of an array, we say things like, "Pizza is the first element in the foodChoices array and its index is zero" or "The element at index 2 of the foodChoices array is Tacos."

The `length` Field

After an array is created and initialized, you can ask it how many slots it has by using the **length** field. Think of the length as an instance variable for an array. Notice that the length does not have a pair of parentheses after it, because it is not a method.

Example

Find the length of the foodChoices array:

```
int result = foodChoices.length;          // result is 5 (not 4!)
```

The `length` of an Array

The `length` field of an array returns the total number of slots that are set aside for that array. It does not return the total number of valid elements in the array. Also, since the first index of an array is always zero, the length of the array is always one more than the last index.

Declaring an Array Using the Keyword new

The second technique for creating an array is used when you know the length of the list, but you may not know all of the values that will go in the list. Again, a pair of **brackets**, [ ], is used to tell the compiler that you want to create an array, not just a regular variable.

General Form for Creating an Array Using the Keyword new

```
dataType[] nameOfArray = new dataType[numberOfSlots];

dataType nameOfArray [ ] = new dataType[numberOfSlots];
```

The number of slots is fixed and all of the elements in the array get the default value for the data type. You can also declare just an array reference variable and then use the keyword new to create the array later.

```
line 1:  dataType[] nameOfArray;                          // no array created
line 2:  nameOfArray = new dataType[numberOfSlots];       // array is created
```

No Resizing an Array

An array cannot be resized after it is created.

Example

Create an array that will represent all of your semester grade point averages for all four years of high school. Let's assume there are two semesters per year. This makes a total of eight semesters. So, let's declare an array to hold eight double values and call the array myGPAs.

```
double[] myGPAs = new double[8];
```

myGPAs	
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
5	0.0
6	0.0
7	0.0

Ouch, all zeros. That hurts. No worries, we can change that. You see, when an array is created in this manner, all of the elements become whatever the **default value** is for that data type. The default value for a double is 0.0, the default value for an `int` is 0, for `boolean` it is false, and for a reference type it is null.

Now that the array is created, let's put some valid numbers into it. In order to replace a value in an array, you need to know the correct index.

Let's let index 0 represent the first semester of your freshman year and let's suppose you earned a GPA of 3.62.

```
myGPAs[0] = 3.62;  // puts 3.62 in slot 0
```

And, let's suppose you earned a 3.75 for the second semester of your sophomore year.

```
myGPAs[3] = 3.75;  // puts 3.75 in slot 3
```

myGPAs		
0	3.62	freshman
1	0.0	
2	0.0	sophomore
3	3.75	
4	0.0	junior
5	0.0	
6	0.0	senior
7	0.0	

Smart, but Not So Smart

The computer has no idea that these numbers represent the GPAs for a student in school. As far as it's concerned, it is just storing a bunch of numbers in an array.

Example

Pick a random color from an array of color names:

```
String[] colors = {"Red","Orange","Yellow","Green","Blue","Indigo","Violet"};
String randomColor = colors[(int)(Math.random() * colors.length)];
System.out.println("My favorite color is " + randomColor);
```

```
OUTPUT
Blue        (Note: The color is chosen randomly and can be different for each run.)
```

The `ArrayIndexOutOfBoundsException`

The valid index values for an array are 0 through one less than the number of elements in the array, inclusive. If you ever accidently use an index that is too big (greater than or equal to the length of the array) or negative, you will get a run-time error called an **ArrayIndexOutOfBoundsException**.

Example

You declare an array of `Circle` objects and attempt to print the radius of the last circle. But you get an `ArrayIndexOutOfBoundsException` because your index is too large. Note: The length of the array is 6. The index of the last `Circle` object is 5. Then, write it the correct way:

```
Circle[] arr = new Circle[6];
double radius = arr[arr.length].getRadius();          // Run-time error
double result = arr[arr.length - 1].getRadius();      // Correct way
```

Common Error

A common error is to accidentally set the index of an array to be the length of the array. The first index of the array is zero, so the last index of the array is **length – 1**.

```
nameOfArray[nameOfArray.length - 1]    // Last element in an array
nameOfArray[nameOfArray.length]        // ArrayIndexOutOfBoundsException
```

Traversing an Array

To **traverse** an array means to move through each slot in the array, one at a time. The most common way to perform a **traversal** is to start with the first index and end with the last index. But that's not the only way. You could start with the last index and move toward the first. You also could do all of the even indices first and then all of the odds. You get what I'm saying? Traversing an array just means that you visit every element at every index in an array.

Print the Contents of an Array by Traversing It

Example

Print out the names of all your friends on Facebook using a `for` loop. By traversing the array from the first element to the last element, you can print each friend's name.

```
String[] faceBookFriends = {"Sierra", "Ciara", "Siarra", "Ciera"};
for (int i = 0; i < faceBookFriends.length; i++)
{
   System.out.println(faceBookFriends[i]);
}
```

```
OUTPUT
Sierra
Ciara
Siarra
Ciera
```

Off-by-One Error

This is a logic error that every programmer has committed at some point. It means that in a loop of some kind, your index is off by one number. You either went one index too high or one index too low. This will result in an `ArrayIndexOutOfBoundsException` being thrown.

The Enhanced `for` Loop (the `for-each` Loop)

The **enhanced for loop** (aka the **for-each loop**) can also be used to traverse an array. It does not work the same way as the standard `for` loop. The `for-each` loop always starts at the beginning of the array and ends with the last element in the array. As the name indicates, the `for-each` loop iterates through each of the elements in the array, and *for each* element, the temporary variable "takes on" its value. The temporary variable is only a copy of the actual value and any changes to the temporary variable are not reflected in the actual value.

An enhanced `for` loop is simpler to write than a standard `for` loop since you don't have to keep track of an index. The enhanced `for` loop is also less error-prone because Java automates the setup and processing of the loop control information. However, the enhanced `for` loop cannot be used to move through an array in reverse order, it cannot assign elements to an array, and since it does not have an index, it can't track any position in the array.

General Form for the Enhanced for Loop (the for-each Loop)

```
dataType[] arrayName =  /* array filled in some way */

for (dataType temporaryVariable : arrayName)
{
   // instructions that use temporaryVariable
}
```

There is no loop control variable for the enhanced `for` loop (it is a **black box**). You only have access to the elements in the array, not the index. Also, the data type of the temporary variable **must** be the same as the data type of the array.

Traversing an Array Using the Enhanced `for` Loop

Example

Print out the names of all your friends on Facebook using a `for-each` loop. Note: `friend` is a temporary variable that is of the same data type as the data stored in the array.

```
String[] faceBookFriends = {"Jordan", "Jordin", "Jordyn"};
for (String friend : faceBookFriends)
{
    System.out.println(friend);
}
```

```
OUTPUT
Jordan
Jordin
Jordyn
```

Mistake When Printing the Contents of an Array

Beginning programmers make the mistake of trying to print the contents of an array by printing the array reference variable. The result that is printed is *garbage*. This is because the reference variable only contains the address of where the array is; it does not store the elements of the array.

```
String[] faceBookFriends = {"Jordan", "Jordin", "Jordyn"};
System.out.println(faceBookFriends);  // Don't do this, it doesn't work
```

```
OUTPUT
@1765f32          // This code does not print the contents of the array
```

Note: The best way to print the contents of an array is to traverse the array using either a `for` loop or a `for-each` loop.

How We Use Algorithms

In our daily lives, we develop and use algorithms all the time without realizing that we are doing it. For example, suppose your grandma calls you because she wants to watch *The Crown* using the Nexflix app on her smart TV, but she can't figure out how to do it. You would respond to her by saying, "Hi, Grandma, well, the first thing you have to do is turn your TV on," then yada, yada, yada until finally you say, "Press play."

What you have just done is developed a process for how to solve a problem. Now, if she insists that you write the steps down on a notecard so she can follow it anytime she wants to, then, you will be writing the algorithm for "How to watch *The Crown* on grandma's smart TV."

> Give a man a fish and he will eat for a day; teach a man to fish and he will eat for a lifetime.
> —Chinese proverb

Why Algorithms Are Important

The word **algorithm** can be simply defined as "a process to be followed to solve a problem." Computer programmers use algorithms to teach the computer how to automate a process.

The number of algorithms to learn is infinite, so rather than attempting to learn all possible algorithms, my goal is to teach you *how to write an algorithm*. This will be handy on the AP Computer Science A Exam when you have to either analyze or generate code to accomplish something you've never done before. The two algorithm concepts in this book teach you how to think like a programmer.

> Give developers a line of code and they will smile for a day; teach developers to write their own algorithms and they will smile for a lifetime.
>
> —Coding proverb

Algorithm Versus Pseudocode Versus Real Java Code

Algorithms are great as directions for how to do something; however, they can't be typed into a computer as a real Java program. The integrated development environment (IDE) needs to have the correct syntax so that the compiler can interpret your code properly. Can you imagine typing in the directions for Grandma as lines of code?

So, to solve this problem, programmers translate algorithms into **pseudocode**, which is a code-like language. This pseudocode is then translated into real code that the computer will understand.

Algorithm → **Pseudocode** → **Java code**

This process is really powerful because as programming languages change, the algorithms can stay the same. In the examples that follow, I've written the code in Java, but it could've easily been written in C++, Python, etc.

> **Turning Your Ideas into Code**
>
> Pseudocode is an inner step when turning your ideas into code. It is code-like but does not use the syntax of any programming language.

The Swap Algorithm

Swapping is the concept of switching the values that are stored in two different variables. The ability to swap values is essential for sorting algorithms (explained in Unit 7). To complete a swap, you need to use a temporary variable to store one of the values.

> **Problem:** Swap the values inside two integer variables. For example, if a = 6 and b = 8, then make a = 8 and b = 6.

Algorithm:
Step 1: Let a = 6 and b = 8
Step 2: Create a temporary variable called c
Step 3: Assign c the value of a
Step 4: Assign a the value of b
Step 5: Assign b the value of c
Step 6: Now, the value of a is 8 and b is now 6

Pseudocode:
set a = 6
set b = 8
c = a
a = b
b = c

Java Code:

```
int a = 6;
int b = 8;
int c;
c = a;
a = b;
b = c;
```

The Copy Algorithm for the Array

Making a copy of an array means to create a brand-new object that is a duplicate of the original. This is different from creating a new array variable that references the original array. That was called making an alias. It is easy for new programmers to make the mistake of thinking that these two are the same.

Problem: Suppose you have an array of integers. Make a new array object that is a copy of this array. The two arrays should contain the same exact values but not be the same object. In Unit 7 we will modify the code to work with an `ArrayList` of `Integers`.

Algorithm:
Step 1: Create an array that is the same length as the original
Step 2: Look at each of the values in the original array one at a time
Step 3: Assign the value in the copy to be the corresponding value in the original
Step 4: Continue until you reach the last index of the original array

Pseudocode:
Create an array called duplicate that has the same length as the original array
for (iterate through each element in the original array)
{
    set duplicate[index] = original[index]
}

Java Code 1: Java code using a for loop

```
int[] original = {23, 51, 14, 50};          // or any random numbers
int[] duplicate = new int[original.length]; // create duplicate array

for (int i = 0; i < original.length; i++)   // iterate through the original
{
   duplicate[i] = original[i];              // copy the values one at a time
}
```

Java Code 2: Java code using a for-each loop

```
int[] original = {23, 51, 14, 50};
int[] duplicate = new int[original.length];
int index = 0;                          // create an index for duplicate array

for (int temp : original)               // temp is a copy of the actual object
{
   duplicate[index] = temp;             // for-each loop does not use an index
   index++;                             // update index for the duplicate array
}
```

Copying an Array Reference Versus Copying an Array

The following code demonstrates a common mistake when attempting to copy an array.

```
int[] original = {23, 51, 14, 50};
int[] copy = original;
```

This code makes the reference variable copy refer to the same object that original refers to. This means that if you change a value in copy, it also changes the value in the original. This code *does not create a new object* that is a duplicate of the original. This code makes both reference variables refer to the same object (the object that original refers to).

The Accumulate Algorithm

Suppose you have a list of all the baseball players on a team and you want to find the total number of hits that the team had. As humans, we can do this quite easily. Just add them up. But how do you teach a computer to add up all of the hits?

General Problem: Add up all of the hits for a baseball team.

Refined Problem: Write a method that finds the sum of all of the hits in an array of hits.

Final Problem: Write a method called `findSum` that has one parameter: an `int` array. The method should return the sum of all of the elements in the array. In Unit 7 we will modify the code to work with an `ArrayList` of `Integer`, and in Unit 8 we will modify the code to work with a 2D array of `int` values.

Algorithm:

Step 1: Create a variable called sum and set it equal to zero
Step 2: Look at each of the elements in the list
Step 3: Add the element to the sum
Step 4: Continue until you reach the end of the list
Step 5: Return the sum

Pseudocode:

```
set sum = zero
for (iterate through all the elements in the list)
{
   sum = sum + element
}
return sum
```

Java Code 1: Java code using a for loop

```
public static int findSum(int[] arr)
{
    int sum = 0;
    for (int i = 0; i < arr.length; i++)
    {
        sum += arr[i];
    }
    return sum;
}
```

Java Code 2: Java code using a for-each loop

```
public static int findSum(int[] arr)
{
    int sum = 0;                          // initializing the sum to 0
    for (int value : arr)                 // for every int in arr
    {
        sum += value;                     // Add the element to the sum
    }
    return sum;                           // return the sum
}
```

The Find-Highest Algorithm

What is the highest score on your favorite video game? As more people play a game, how does the computer figure out what is the highest score in the list?

General Problem: Find the highest score out of all the scores for a video game.

Refined Problem: Write a method that finds the highest score in a list of scores.

Final Problem: Write a method called findHigh that has one parameter: an `int` array. The method should return the largest value in the array. In Unit 7 we will modify the code to work with an `ArrayList` of `Integer`, and in Unit 8 we will modify the code to work with a 2D array of `int` values.

Solution 1: Let the highest be the first element in the list.

Algorithm:

Step 1: Create a variable called high and set it to the first element in the list
Step 2: Look at each of the scores in the list

Step 3: If the score is greater than the high score, then make it be the new high score
Step 4: Continue until you reach the end of the list
Step 5: Return the high score

Pseudocode:

```
set high = first score
for (iterate through all the scores in the list)
{
  if (score > high)
  {
    high = score
  }
}
return high
```

Java Code 1: Java code using a for loop

```
public static int findHigh(int[] arr)
{
    int high = arr[0];                          // set high = first element
    for (int i = 1; i < arr.length; i++)        // start with the 2nd element
    {
        if (arr[i] > high)                      // if element is greater than
                                                // high
            high = arr[i];                      // set high equal to that
                                                // element
    }
    return high;                                // return the highest in the
                                                // array
}
```

Java Code 2: Java code using a for-each loop

```
public static int findHigh(int[] arr)
{
    int high = arr[0];
    for (int value : arr)
    {
          if (value > high)
              high = value;
    }
    return high;
}
```

Solution 2: Let the highest be an extremely low value.

There is an alternative solution to the high/low problem that you should know. To execute this alternative, we modify the first step in the previous algorithm.

Step 1: Create a variable called high and set it to *a really, really small number.*

This algorithm works as long as the really, really small number is guaranteed to be smaller than at least one of the numbers in the list. In Java, we can use the public fields from the Integer class to solve this problem.

```
int high = Integer.MIN_VALUE;
```

Likewise, to solve the *find-lowest* problem, you could set the low to be a really, really big number.

```
int low = Integer.MAX_VALUE;
```

Searching for the Smallest Item in a List

If we changed the previous example to find the *lowest* score in the list, then the comparator in the `if` statement would be changed to less than, <, instead of greater than, >.

› Rapid Review

The Array

- An array is a complex data structure that can store a list of data of the same data type.
- An array is also referred to as a one-dimensional (or 1D) array.
- An array can be created by assigning it a predetermined list.
- An array can be created by using the keyword new.
- An array can store either primitive data types or object reference types.
- Even though arrays are objects, they do not have methods.
- An array has one public field, called length.
- The length of the array refers to how many elements can be stored in the array and is decided when the array is created.
- The length of an array cannot be resized once it is declared. That is, an array cannot be made shorter or longer once it is created.
- The length field returns the total number of locations that exist in the array, and not the number of meaningful elements in the array.
- An element of an array refers to the item that is stored in the array.
- The index of an array refers to the position of an element in an array.
- The first index of an array is 0. The last index is its length − 1.
- Unused indices in an array automatically receive the default value for that data type.
- If an array contains objects, their default values are null. You need to instantiate an object for each element of the array before they can be used.
- Using an index that is not in the range of the array will throw an `ArrayIndexOutOfBoundsException`.

Traversing an Array

- Traversing an array refers to the process of visiting every element in the array.
- There are many options for traversing an array; however, the most common way is to start at the beginning and work toward the end.
- Based on the situation, there are times where you may want to traverse an array starting at the end and work toward the beginning.
- Sometimes you may want to traverse only a section of the array.
- When using a `for` loop to traverse an array, be sure to use length − 1 rather than length when accessing the last element in the array.
- The enhanced `for` loop iterates through each element in the array without using an index.
- The enhanced `for` loop is also known as the for-each loop.

- The enhanced `for` loop always starts with the first element and ends with the last.
- Use an enhanced `for` loop when you don't need to know the index for each element and you don't need to change the element in any way.

Algorithms and Pseudocode

- The purpose of this concept is to teach you how to translate an idea into code.
- An algorithm is a sequence of steps that, when followed, produces a specific result.
- Algorithms are an extremely important component of software development since you are teaching the computer how to do something all by itself.
- Pseudocode is a hybrid between an algorithm and real code.
- Pseudocode does not use correct syntax and is not typed into an IDE.
- Pseudocode helps you translate an algorithm into real code since it is code-like and more refined than an algorithm.
- Programmers translate algorithms into pseudocode and then translate pseudocode into real code.

Swap Algorithm

- The swap algorithm allows you to switch the values in two different variables.
- To swap the values in two different variables, you must create a temporary variable to store one of the original variables.

Copy Algorithm

- The copy algorithm is used to create a duplicate version of a data structure.
- It looks at each element in the original list and assigns it to the duplicate list.
- The duplicate is not an alias of the original; it is a new object that contains all of the same values as the original.

Accumulate Algorithm

- The accumulate algorithm finds the sum of all the items in a list.
- It looks at each element in the list one at a time, adding the value to a total.

Find-Highest Algorithm

- The find-highest algorithm finds the largest value in a list.
- It looks at each element in the list one at a time, comparing the value to the current high value.
- Common ways to implement this algorithm include initializing the highest value with the first value in the list or to an extremely small negative number.
- The find-highest algorithm can be modified to find the lowest item in a list.

› Review Questions

Basic Level

1. Consider the following code segment.

```
int number = 13;
int[] values = {0, 1, 2, 3, 4, 5};
for (int val : values)
{
   number += val;
}
System.out.println(number);
```

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

(A) `0`
(B) `13`
(C) `26`
(D) `28`
(E) `56`

2. Consider the following code segment.

```
int total = 3;
int[] values = {8, 6, 4, -2};
total += values[total];
total += values[total];
System.out.println(total);
```

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

(A) `0`
(B) `1`
(C) `3`
(D) `7`
(E) `16`

3. Consider the following code segment.

```
int[] values = {7, 9, 4, 1, 6, 3, 8, 5 };
for (int i = 3; i < values.length; i += 2)
{
   System.out.print(values[i - 2]);
}
```

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

(A) `91`
(B) `746`
(C) `913`
(D) `79416385`
(E) `416385`

4. Which segment of code will correctly count the number of even elements in an array arr?

```
I.  int count = 0;
    for (int i = 0; i < arr.length; i++)
        if (i % 2 == 0)
            count++;

II. int count = 0;
    for (int i = 0; i < arr.length; i++)
        if (arr[i] % 2 == 0)
            count++;

III. int count = 0;
    for (int i = 0; i < arr.length; i+=2)
        if (arr[i] % 2 == 0)
            count++;
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) II and III only

5. Consider the following code segment.

```
double [ ] arr = {9.05, 6.13, 4.20, 2.25, 7.28};
for (int i = arr.length - 1; i > 0; i--)
    arr[i] = arr[i - 1];

for (double element : arr)
    System.out.print(element + "  ");
```

What will be printed after the code segment is executed?

(A) 9.05 9.05 6.13 4.2 2.25
(B) 7.28 2.25 4.2 6.13 9.05
(C) 6.13 4.2 2.25 7.28 7.28
(D) 9.05 6.13 4.2 2.25 7.28
(E) Nothing will be printed. An `ArrayIndexOutOfBoundsException` will occur.

Advanced Level

6. Consider the following code segment.

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

What values are stored in array `nums` after executing the code segment?

(A) `ArrayIndexOutOfBoundsException`
(B) `{2, 2, 3, 3, 4, 4, 5, 5}`
(C) `{0, 0, 1, 3, 5, 7, 9, 11}`
(D) `{0, 0, 1, 1, 3, 4, 5, 6}`
(E) `{0, 0, 1, 2, 2, 3, 4, 5}`

7. A postcondition of a method is: arr[0] > arr[k] for all k such that 0 < k < arr.length. Which of the following is a correct conclusion?

(A) All values in the array are identical.
(B) The array elements are in ascending order.
(C) The array elements are in descending order.
(D) The smallest value in the array is arr[0].
(E) The largest value in the array is arr[0].

8. Consider the following code segment intended to count the number of elements in array `fruits` that contain the letter "e".

```
int count = 0;
for (String element : fruits)
      <statement>
```

Which of the following substitutions for <statement> will cause the segment to work as intended?

(A)
```
if (element.equals("e") > 0)
   count++;
```
(B)
```
if (element.equals("e") >= 0)
   count++;
```
(C)
```
if (element.indexOf("e") > 0)
   count++;
```
(D)
```
if (element.indexOf("e") >= 0)
   count++;
```
(E)
```
if (element.indexOf("e") < 0)
   count++;
```

9. Consider two parallel arrays which contain the names and corresponding scores of players in a trivia game.

```
String [] names = {"Rob", "Pam", "Sandy", "Kelly", "Kim", "J.P."};
int [] scores = {2600, 2420, 1790, 2100, 3100, 3250};
```

Which of the following code segments correctly prints the names and scores of all the players who scored above `high`?

(A)
```
for(int element : scores)
   if (element > high)
      System.out.println(names + " " + element);
```
(B)
```
for(int element : scores)
   if (element >= high)
      System.out.println(names + " " + element);
```
(C)
```
for(int i = 0; i < scores.length; i++)
   if (scores[i] > high)
      System.out.println(names[i] + " " + scores[i]);
```
(D)
```
for(int i = 0; i < scores.length; i++)
   if (scores[i] >= high)
      System.out.println(names[i] + " " + scores[i]);
```
(E)
```
for(int i = 0; i <= scores.length; i++)
   if (scores[i] > high)
      System.out.println(names[i] + " " + scores[i]);
```

10. Count the number of occurrences.

Write a method that counts the number of values in an array that are between a lower bound and an upper bound. The method takes as parameters a double array, and two doubles representing lower and upper

respectively. The method should return a count representing the number of elements in the array that are greater than or equal to the upper bound or less than or equal to the lower bound.

For example, if the lower bound is 250.0, the upper bound is 750.0, and the array passed to the method is:

634.5	521.8	786.6	899.0	509.1	235.4	750.0	806.8	142.5	645.3

The method should return the value 6.

```
//  precondition: lower <= upper
public int countBetween(double [ ] values, double lower, double upper)
{
   // Write the implementation
}
```

11. Determine the relative strength index of a stock.

The relative strength index (RSI) of a stock determines if the stock is overpriced or underpriced. The range of the RSI is from 0 to 100 (inclusive). If the value of the RSI is greater than or equal to 70, then the stock is determined to be overpriced and should be sold.

Write a method that takes an array of RSI values for a specific stock and determines when the stock is overpriced. The method has one parameter: a double array. The method should return a new array of the same length as the parameter array. If the value in the RSI array is greater than 70, set the value in the overpriced array to `true`. If the RSI value is less than or equal to 70, set the value in the overpriced array to `false`.

For example, if the RSI array passed to the method is:

55.6	63.2	68.1	70.1	72.4	73.9	71.5	68.3	67.1	66.2

The overpriced array that is returned by the method will be:

false	false	false	true	true	true	true	false	false	false

```
public boolean[] overpriced(double[] rsiValues)
{
   // Write the implementation
}
```

12. Fill an array with even numbers.

Write a method that fills an array with random even numbers. The method takes two parameters: the number of elements in the array and the range of the random numbers (0–number inclusive).

For example:

```
int[] result = onlyEvens(10, 100);  //result contains ten even #'s [0,100]
```

result contains the following (results will change each time program is run):

56	96	88	30	0	92	12	98	4	28

```
public static int[] onlyEvens(int arraySize, int range)
{
   // Write the implementation
}
```