

Unit 3: Arrays and Algorithms

This unit introduces you to data structures to store primitive values and object references. You use one-dimensional (1D) arrays to store multiple related values while expanding your knowledge of loops and conditionals to analyze and process data in a 1D array. You learn to use **for** loops to traverse arrays and discover that an algorithm involving loops can be implemented with either a **for** loop or a **while** loop. Throughout the unit, you develop and modify algorithms to find and manipulate elements in a 1D array while also discovering the concept of polymorphism when traversing arrays of objects. While developing algorithms, you identify preconditions and postconditions and implement solutions to ensure that these conditions are satisfied. Throughout this unit, you continue to develop software engineering skills as you learn to make design decisions and use 1D arrays to store and analyze data.

Objectives

In this unit, you learned:

- Declare and initialize one-dimensional (1D) arrays to store primitive values and objects using the **new** keyword and initializer lists
- Traverse the elements in one-dimensional (1D) arrays using **for** loops, **while** loops, and enhanced **for** loops
- Write algorithms to find and modify elements in one-dimensional (1D) arrays
- Implement solutions to satisfy the preconditions and postconditions of an algorithm
- Debug common errors that occur when traversing one-dimensional (1D) arrays
- Use polymorphism to create and traverse one-dimensional (1D) arrays of a superclass type

Lesson 1: One-Dimensional (1D) Arrays

How can I store multiple related values without creating multiple variables?

You have used variables to store literal values and references to objects. In this lesson, you discover the need for one-dimensional (1D) arrays to store multiple related values. You explore the syntax and functionality of 1D arrays and learn how to declare and initialize a 1D array using the `new` keyword. You learn how to determine the length of a 1D array, then practice creating 1D arrays using the `new` keyword and assigning and accessing values at specific positions.

Key Terms

Term	Definition
data structure	a structure for organizing, processing, retrieving, and storing data
element	a single value or object in a data structure
one-dimensional (1D) array	a data structure that holds multiple values of the same data type

Syntax

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

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

```
nameOfArray.length
```

Lesson 2: Modifying Elements

How can I easily assign multiple elements to a one-dimensional (1D) array?

In this lesson, you explore using initializer lists to create one-dimensional (1D) arrays and learn how to access and modify elements. You discover that the size of a 1D array cannot be changed after it is created and identify similarities and differences between assigning values to a 1D array and to a variable. You then practice creating a 1D array using initializer lists and modifying the values stored in a 1D array.

Key Terms

Term	Definition
index	an integer value that indicates the position of a value in a data structure
initializer list	a comma-separated list of values or objects given inside curly braces ({ })

Syntax

```
nameOfArray[index]
```

```
nameOfArray[index] = value;
```

```
dataType[] nameOfArray = { value1, value2, value3, . . . };
```

Lesson 3: Traversing 1D Arrays

How can I easily access all elements in a one-dimensional (1D) array?

You have learned how to declare and initialize one-dimensional (1D) arrays and access and modify elements one at a time. In this lesson, you discover the need for loops to repeatedly access multiple elements and revisit **while** loops. You learn how to traverse 1D arrays using a **while** loop and discover the cause of the **ArrayIndexOutOfBoundsException** and strategies for debugging this error. You then practice using **while** loops to traverse one and multiple 1D arrays.

Key Terms

Term	Definition
traverse	to access elements in a data structure one by one

Syntax

```
int index = startingValue;

while (index < nameOfArray.length) {
    // code to execute
    // change of index
}
```

Lesson 4: For Loops

How can I write more efficient code to traverse a 1D array?

You learn about the structure and functionality of a **for** loop and explore how it can be used to traverse a one-dimensional (1D) array. You identify similarities and differences between **while** loops and **for** loops and practice tracing code segments involving a **for** loop to determine the output. You then practice converting **while** loops to **for** loops and using **for** loops to traverse 1D arrays to access and modify elements and identify ideal scenarios for using each type of loop.

Key Terms

Term	Definition
off-by-one error	an error that occurs when a loop repeats one time too many or one time too few
loop control variable	a variable that is changed by a constant value and determines the end of a loop
increment	to increase a value by one
decrement	to decrease a value by one

Syntax

```
for (initialization; condition; change) {  
    // code to execute  
}
```

```
for (int index = 0; index < nameOfArray.length; index++) {  
    // code to execute  
}
```

```
nameOfVariable++
```

```
nameOfVariable--
```

Lesson 5: Preconditions and Postconditions

How can I perform calculations on the values stored in a 1D array?

You have learned how to create one-dimensional (1D) arrays using the `new` keyword and initializer lists and access and modify one or multiple elements. In this lesson, you explore how values from a text file can be read and stored in a 1D array and consider the preconditions and postconditions an algorithm should satisfy. You then apply your knowledge of 1D arrays, loops, and returning values from methods to plan and implement algorithms to calculate the sum and average of the values in a 1D array.

Key Terms

Term	Definition
text file	a file that contains letters, numbers, and/or symbols but has no special formatting
precondition	a condition that must always be true just before the execution of a code segment
postcondition	a condition that must always be true just after the execution of a code segment

Syntax

```
/*  
 * Precondition: The condition that should be  
 * true before the code segment is executed  
 */
```

```
/*  
 * Postcondition: The condition that should be  
 * true after the code segment is executed  
 */
```

```
public dataType nameOfMethod(dataType parameterName) {  
    // code to execute  
    return valueToReturn;  
}
```

Lesson 6: Polymorphism

How can polymorphism improve the efficiency and flexibility of my program code?

In this lesson, you expand on what you know about inheritance to explore how you can create arrays of objects. You learn about polymorphism and discover how it can be implemented to make program code more efficient and flexible. You recall overriding the `toString()` method and how this approach can be applied to override a superclass method in a subclass. You then practice using superclass references to create and work with 1D arrays.

Key Terms

Term	Definition
polymorphism	where the same object or method has more than one form

Syntax

```
SuperclassName nameOfVariable = new SubclassName();
```

```
SuperclassName[] nameOfArray = { nameOfVariable, nameOfVariable, . . . };
```

```
SuperclassName[] nameOfArray = { new ClassName(), new ClassName(), . . . };
```

```
for (int index = 0; index < nameOfArray.length; index++) {  
    nameOfArray[index].methodName();  
}
```

Lesson 7: Enhanced For Loops

When would I want to traverse an array with an enhanced for loop?

You have learned to use **while** and **for** loops to traverse a 1D array. In this lesson, you explore the syntax and functionality of an enhanced **for** loop and identify scenarios where this type of loop would be useful. You practice converting **while** and **for** loops to enhanced **for** loops and implementing algorithms to find the sum and average of the values in a 1D array using an enhanced **for** loop.

Syntax

```
for (dataType variableName : nameOfArray) {  
    // code to execute  
}
```

Lesson 8: Array Algorithms

How can I use what I know about object-oriented programming and 1D arrays to plan and implement algorithms?

In this lesson, you apply what you have learned so far to plan and implement standard algorithms to search or modify a one-dimensional (1D) array. You discover the need for the `equals()` method when comparing `String` objects for equality and explore the difference between executing a return statement inside of a loop and at the end of a method. You then work with a partner and use manipulatives to plan and implement an algorithm to determine the minimum or maximum value in a 1D array, determine if at least one element in a 1D array has a particular property, determine if all elements in a 1D array have a particular property, or determine the number of elements in a 1D array meet specific criteria. After implementing your algorithm, you conduct a code review to refine and improve your code.

Syntax

```
nameOfFirstVariable.equals(nameOfSecondVariable)
```

Lesson 9: Finding Duplicates

How can I check if a 1D array contains duplicate elements?

You have learned how to traverse one-dimensional (1D) arrays and implement algorithms to search and modify elements. You expand on your knowledge of loops to explore using nested loops with 1D arrays and practice tracing these to determine the output of code segments using nested loops. You then work with a partner and use manipulatives to plan and implement an algorithm to find duplicate values in a 1D array.

Key Terms

Term	Definition
nested loop	a loop inside of another loop

Syntax

```
for (int outer = 0; outer < nameOfArray.length; outer++) {  
    for (int inner = 0; inner < nameOfArray.length; inner++) {  
        // code to execute  
    }  
}
```