Chapter 3
Supplemental Topics

3.6 Processing Array of Objects

So far, we have dealt with user-defined objects one at a time: instantiating new objects, naming them, and invoking their methods. To work with multiple objects of the same type effectively, we can use an array. Similar to the type `int[]` when we used arrays of integers, the data type here will be the name of the class, followed by square brackets. To create an array of a specified type of object, use the syntax below:

Syntax: naming and creating array of objects

```
className[] nameOfArray = new className[length];
```

This syntax will create an array of the given class, with all elements initialized to `null`. Before we can work with the objects, we need to populate the array so that elements are not `null`. To do this, we iterate through the array and invoke the appropriate constructor at each element of the array.

Syntax: initializing array of objects

```
for (int i = 0; I < length; i++)
    nameOfArray[i] = new className(parameters...)
```

Recall that while the array variable name lives on the stack, the variable refers to a heap memory location. In contrast to arrays of primitives, each element of the array in heap memory would refer to an object in another heap memory location. A sketch of the stack and heap memory might look like this (see Figure 3-5):

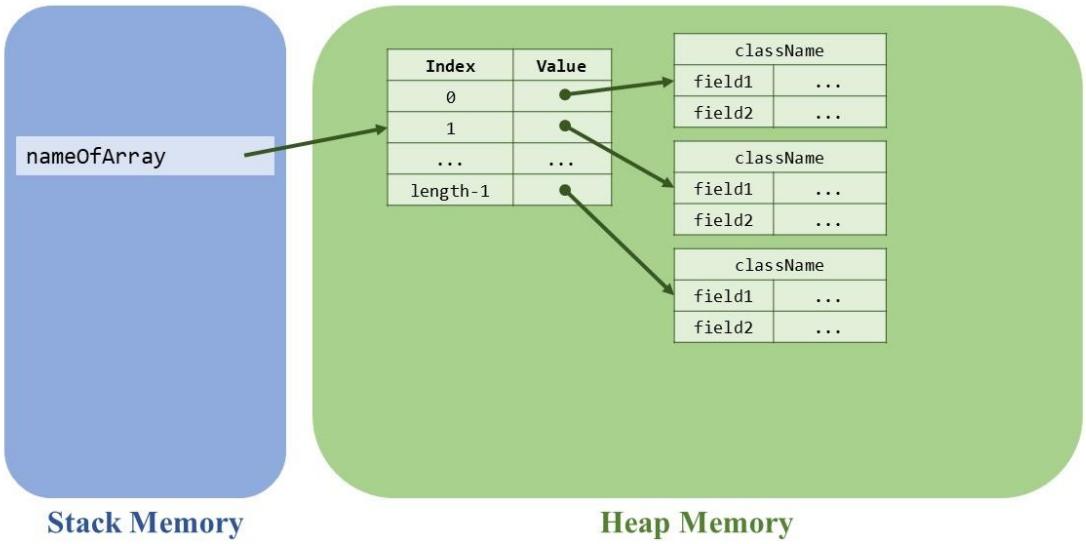

Figure 3-5 Example of stack and heap memory for an array of objects

After the array and its associated objects are created, we can perform operations on the objects by accessing the associated array elements.

We'll walk through a quick example (JavaClassEx4_arrayofobj), using the `Person` and `Outfit` classes that we completed earlier.

JavaClassEx4_arrayofobj

Goal: Work with an array of objects

```
public class JavaClassEx4_arrayofobj{
    public static void main(String[] args) {
        String[] colors = new String[5];
        colors[0] = "green";
        colors[1] = "blue";
        colors[2] = "pink";
        colors[3] = "gray";
        colors[4] = "magenta";

        /* create an array of Outfit objects:
           first, need to create an array of type of Outfit
           next, for each outfit array element, need to create an outfit
           object */
        Outfit[] arrOutfits = new Outfit[5];
        for(int i = 0; i < 5; i++)
            arrOutfits[i] = new Outfit(Outfit.TYPES[i], colors[i], i % 3);
```

A1

```
        Person[] arrPersons = new Person[5];
        for(int i = 0; i < 5; i++) {
            arrPersons[i] = new Person("student"+ i, 10+i, 100.0+i*10,
                5.0+i*0.1, arrOutfits[i], "parents" + i);
        }
```

A2

```
        for(Person var: arrPersons)
            System.out.println(var);
```

A3

```
        System.out.println("the average age is: "
```

A4

```
            calcAverageAge(arrPersons));

        // Call your Knowledge check 9 method here.

        // Call your Knowledge check 10 method here.

        // Call your Knowledge check 11 method here.
    }
```

```
    public static double calcAverageAge(Person[] ppl){
        double avgAge = 0.0;
        for(Person var: ppl)
            avgAge += var.getAge();

        return avgAge / ppl.length;
    }

    // Knowledge check 9: write a method to find the tallest Person

    // Knowledge check 10: write a method to find the smallest size of
    // Outfit worn

    // Knowledge check 11: write a method to find all the Person
    // objects who wear the smallest size of Outfit
}
```

A5

We want to work with an array of five `Person` objects. However, since each `Person` has an `Outfit`, we first need to create the `Outfit` objects. We do this efficiently in the client program by using an array. **(See Cell A1.)**

```
String[] colors = new String[5];
colors[0] = "green";
colors[1] = "blue";
colors[2] = "pink";
colors[3] = "gray";
colors[4] = "magenta";

/* create an array of Outfit objects:
   First, create an array of type of Outfit.
   Next, for each Outfit array element, create an Outfit object. */
Outfit[] arrOutfits = new Outfit[5];
for(int i = 0; i < 5; i++)
    arrOutfits[i] = new Outfit(Outfit.TYPES[i], colors[i], i % 3);
```

`arrOutfits` is an array containing five `Outfit` elements. Each `Outfit` element was created with the explicit constructor. The modulo operation in the `Outfit` constructor (`i % 3`) simply converts the element index into an appropriate size integer: 0, 1, or 2.

Once we have the `Outfit` objects, we can create the `Person` objects in **Cell A2**:

```
Person[] arrPersons = new Person[5];
for(int i = 0; i < 5; i++) {
    arrPersons[i] = new Person("student"+ i, 10+i, 100.0+i*10,
        5.0+i*0.1, arrOutfits[i], "parents" + i);
}
```

`arrPersons` is an array containing five `Person` elements. Each `Person` element was created using the `Person` explicit constructor by assigning the corresponding `Outfit` object from the `arrOutfits` array. Then we use for-each loop to output each person in the array.

In Cell A3, we print each of the `Person` objects in the `arrPersons` array by using a for-each loop.

```
for(Person var: arrPersons)
    System.out.println(var);
```

This produces the following output:

```
Name: student0
Age: 10
Weight: 100.0
Height: 5.0
Outfit: The Vest is green and its size is 0
Parents: Mr. and Mrs. parents0

Name: student1
Age: 11
Weight: 110.0
Height: 5.1
Outfit: The Shirt is blue and its size is 1
Parents: Mr. and Mrs. parents1

Name: student2
Age: 12
Weight: 120.0
Height: 5.2
Outfit: The Polo is pink and its size is 2
Parents: Mr. and Mrs. parents2

Name: student3
Age: 13
Weight: 130.0
Height: 5.3
Outfit: The Jacket is gray and its size is 0
Parents: Mr. and Mrs. parents3

Name: student4
Age: 14
Weight: 140.0
Height: 5.4
Outfit: The Coat is magenta and its size is 1
Parents: Mr. and Mrs. parents4
```

We can also aggregate results from an array of objects. As an example, we defined the `calcAverageAge()` method in the client program. This method takes in an array of `Person` objects and calculates the average age of the group. **(See Cell A5.)**

```
public static double calcAverageAge(Person[] ppl){
    double avgAge = 0.0;
    for(Person var: ppl)
        avgAge += var.getAge();

    return avgAge / ppl.length;
}
```

This method iterates through the `Person` array and invokes each `Person` instance's `getAge()` method. After adding all the `Person` objects' ages together, the method divides by the length of the input array (the number of `Person` objects) to find the average age. **Notice that this method assumes that all elements of the input `ppl` array are non-`null`.** If an element had been `null`, the program would attempt to call `getAge()` on an un-instantiated object and throw a `NullPointerException`. We'll discuss `Exceptions` and how to handle them in a later chapter, but for now, understand that when you work with arrays of objects, it is important to make sure that you have filled in all array elements by creating objects.

When we invoke this method on `arrPersons`, we would expect the result to be:

$$(10 + 11 + 12 + 13 + 14) / 5 = 12.$$

As a check, the program produces the following output:

```
the average age is: 12.0
```

Now it's your turn! Try the following **Knowledge Checks for Processing Arrays of Objects**:

(9) Write a method to find the tallest `Person`. Output the *result* in the client program.
(10) Write a method to find the smallest `Outfit` size that any `Person` object is wearing. Output the *result* in the client program.
(11) Write a method to find all the `Person` objects who are wearing `Outfits` of the smallest size. Output the *result* in the client program.