Chapter 6: Arrays
Objectives

After studying this chapter, Student should be able to:

- Manipulate a collection of data values, using an array.
- Declare and use an array of primitive data types in writing a program.
- Declare and use an array of objects in writing a program.
- Define a method that accepts an array as its parameter and a method that returns an array.
- Describe how a two-dimensional array is implemented as an array of arrays.
Array Basics

- An array is a collection of data values.
- If your program needs to deal with 100 integers, 500 Account objects, 365 real numbers, etc., you will use an array.
- In Java, an array is an indexed collection of data values of the same type.
Arrays of Primitive Data Types

- **Array Declaration**

  
  \[
  \text{<data type> [ ] <variable>}
  \quad \text{//variation 1}
  \]
  
  \[
  \text{<data type> <variable>[ ]} \quad \text{//variation 2}
  \]

- **Array Creation**

  
  \[
  \text{<variable> = new <data type> [ <size> ]}
  \quad \text{Variation 1}
  \]

  
  \[
  \text{<variable> = new <data type> [ <size> ]}
  \quad \text{Variation 2}
  \]

  
  \[
  \text{double[ ] rainfall;}
  
  \text{rainfall}
  
  \text{= new double[12];}
  \]

  
  \[
  \text{double rainfall [ ];}
  
  \text{rainfall}
  
  \text{= new double[12];}
  \]

  
  An array is like an object!
Accessing Individual Elements

- Individual elements in an array accessed with the indexed expression.

```java
double[] rainfall = new double[12];
```

The index of the first position in an array is 0.

This indexed expression refers to the element at position #2.

rainfall[2]

0 1 2 3 4 5 6 7 8 9 10 11
double[] rainfall = new double[12];

double annualAverage,
sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {
    rainfall[i] = Double.parseDouble(
        JOptionPane.showInputDialog(null,
        "Rainfall for month " + (i+1) )
    );
    sum += rainfall[i];
}

annualAverage = sum / rainfall.length;

The public constant length returns the capacity of an array.
double[] rainfall = new double[12];
String[] monthName = new String[12];
monthName[0] = "January";
monthName[1] = "February";
...
double annualAverage, sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {
    rainfall[i] = Double.parseDouble(
        JOptionPane.showMessageDialog(null, 
            "Rainfall for " + monthName[i]));
    sum += rainfall[i];
}
annualAverage = sum / rainfall.length;
Array Processing – Sample 3

- Compute the average rainfall for each quarter.

```java
//assume rainfall is declared and initialized properly

double[] quarterAverage = new double[4];

for (int i = 0; i < 4; i++) {
    sum = 0;
    for (int j = 0; j < 3; j++) {
        //compute the sum of
        sum += rainfall[3*i + j];  //one quarter
    }
    quarterAverage[i] = sum / 3.0;  //Quarter (i+1) average
}
```
Array Initialization

Like other data types, it is possible to declare and initialize an array at the same time.

```java
int[] number = { 2, 4, 6, 8 };

double[] samplingData = { 2.443, 8.99, 12.3, 45.009, 18.2, 9.00, 3.123, 22.084, 18.08 };

```

- `number.length` → 4
- `samplingData.length` → 9
- `monthName.length` → 12
Variable-size Declaration

- In Java, we are not limited to fixed-size array declaration.
- The following code prompts the user for the size of an array and declares an array of designated size:

```java
int size;

int[] number;

size = Integer.parseInt(JOptionPane.showInputDialog(null, "Size of an array:");

number = new int[size];
```
Arrays of Objects

- In Java, in addition to arrays of primitive data types, we can declare arrays of objects.
- An array of primitive data is a powerful tool, but an array of objects is even more powerful.
- The use of an array of objects allows us to model the application more cleanly and logically.
public class Person
{
    private String name;
    private int age;
    private char gender;

    public Person()
    {age=0; name=" "; gender=' ';}

    public Person(String na, int ag, char gen)
    {setAge(ag); setName(na); setGender(gen); }

    public Person(Person pr)
    { setPerson(pr);}

    public void setPerson(Person p)
    { age=p.age; gender =p.gender;
      name=p.name. substring(0, p.name.length());
    }

    public void setAge (int a) {age=a;}
    public void setGender (char g) {gender=g;}
    public void setName(String na)
    {name=na.substring(0, na.length());}

    public int getAge(){return age;}

    public char getGender () {return gender;}

    public String getName () { return name;}
}
The Person Class

- We will use Person objects to illustrate the use of an array of objects.

```java
public class Person {
    private String name;
    private int age;
    private char gender;
    public Person()   {age=0; name=" "; gender=' ';}
    public Person(String na, int ag, char gen)  {setAge(ag); setName(na); setGender(gen); }
    public Person(Person pr)       { setPerson(pr);}
    public void setPerson(Person p)
    { age=p.age; gender =p.gender;
      name=p.name. substring(0, p.name.length());      }
    public void setAge (int a) {age=a;}
    public void setGender (char g) {gender=g;}
    public void setName(String na)
    {name=na.substring(0, na.length());}
    public int getAge(){return age;}
    public char getGender () {return gender;}
    public String getName () { return name;}
}
```
Creating an Object Array - 1

Code

```java
Person[] person;
person = new Person[20];
person[0] = new Person();
```

Only the name `person` is declared, no array is allocated yet.

State of Memory

After `A` is executed

Only the name `person` is declared, no array is allocated yet.
Now the array for storing 20 Person objects is created, but the Person objects themselves are not yet created.

Code

Person[ ] person;
person = new Person[20];
person[0] = new Person();

State of Memory

After **B** is executed
Creating an Object Array - 3

Person[ ] person;
person = new Person[20];

person[0] = new Person();

One Person object is created and the reference to this object is placed in position 0.

After is executed
Person Array Processing – Sample 2

- Find the youngest and oldest persons.

```java
int minIdx = 0; // index to the youngest person
int maxIdx = 0; // index to the oldest person

for (int i = 1; i < person.length; i++) {
    if (person[i].getAge() < person[minIdx].getAge()) {
        minIdx = i; // found a younger person
    } else if (person[i].getAge() > person[maxIdx].getAge()) {
        maxIdx = i; // found an older person
    }
}

// person[minIdx] is the youngest and person[maxIdx] is the oldest
```
**Object Deletion – Approach 1**

```java
int delIdx = 1;
person[delIdx] = null;
```

Delete Person B by setting the reference in position 1 to `null`.

Before `A` is executed:
```
0 1 2 3
A B C D
```

After `A` is executed:
```
0 1 2 3
A C D
```
Object Deletion – Approach 2

```java
int delIdx = 1, last = 3;
person[delIndex] = person[last];
person[last] = null;
```

Delete Person B by setting the reference in position 1 to the last person.

Before A is executed:

```
A B C D
```

After A is executed:

```
A D C
```
Person Array Processing – Sample 3

• Searching for a particular person. Approach 2
  Deletion is used.

```java
int i = 0;

while ( person[i] != null && !person[i].getName().equals("Latte") ) {
    i++;
}

if ( person[i] == null ) {
    //not found - unsuccessful search
    System.out.println("Ms. Latte was not in the array");
}
else {
    //found - successful search
    System.out.println("Found Ms. Latte at position " + i);
}
```
Passing Arrays to Methods - 1

**Code**

```java
public int searchMinimum(float[] number)
{
    ...
}

minOne = searchMinimum(arrayOne);
```

**State of Memory**

At before `searchMinimum`

- `arrayOne` in memory

A. Local variable `number` does not exist before the method execution
passing arrays to methods - 2

code

```java
public int searchMinimum(float[] number)
{
    ...
}
```

state of memory

The address is copied at B.

B. The value of the argument, which is an address, is copied to the parameter.
Passing Arrays to Methods - 3

Code

```
public int searchMinimum(float[] number)
{
    ...
}
```

State of Memory

While at inside the method

arrayOne

number

C. The array is accessed via number inside the method.
**Passing Arrays to Methods - 4**

**Code**

```java
public int searchMinimum(float[] number)
{
    ...
}
```

**State of Memory**

At after `searchMinimum`

D. The parameter is erased. The argument still points to the same object.