Chapter 8 - Arrays

Basic arrays

- Arrays are a sequence of a data type
- Two components
  - Datatype
  - Size

Chapter 8

- Common to want to deal with collection of items
  - Keep information about all 302 students (250)
  - Information about all video files on computer
  - All of my grades since kindergarten
  - Books in a library
- Infeasible to enumerate an identifier for each

Datatype

- What type of element do we want as the base of the array?
- Can be a primitive or reference type

Syntax:
<datatype>[] <identifier>;
<datatype> <identifier>[][];

8.1 Arrays

- Problem: Want to display differences in scores from average
  - We have to keep track of each score
  - We could just create one identifier for all 10 scores (score1, score2...)
    - But what if we wanted to take in more scores?
  - Solution: Use an array of scores to store all scores
    - double[] scores; // OR
double scores[];

Component 2: Size

- Must use new operator

Syntax:
<identifier> = new double<size>;

Example:
scores = new double[10];
double[] scores = new double[10];

- What data type is an array?
Arrays

- Can access an individual item in the collection
  - Use a single identifier for the whole collection (scores) and then use an **indexed expression** to access an **array element**

- Zero based indexing
  - Index numbered 0 through size-1
  - scores[0]

- Can use expressions → scores[i+1]

Output Difference

double[] scores = new double[10];
double avg, sun = 0;

for(int i=0; i < 10; i++){
    System.out.print("Enter Score");
scores[i] = in.nextDouble();
sun += scores[i];
}

avg = sun/10;

for(int i=0; i < 10; i++){
    System.out.print("Score "+i+" ");
    System.out.println(score[i] - avg);
}

Sample Run

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter score: 22</td>
<td>Score 0 22.0 -48.0</td>
</tr>
<tr>
<td>Enter score: 24</td>
<td>Score 1 24.0 -46.0</td>
</tr>
<tr>
<td>Enter score: 90</td>
<td>Score 2 90.0 20.0</td>
</tr>
<tr>
<td>Enter score: 88</td>
<td>Score 3 88.0 18.0</td>
</tr>
<tr>
<td>Enter score: 75</td>
<td>Score 4 75.0 5.0</td>
</tr>
<tr>
<td>Enter score: 95</td>
<td>Score 5 95.0 25.0</td>
</tr>
<tr>
<td>Enter score: 65</td>
<td>Score 6 65.0 -5.0</td>
</tr>
<tr>
<td>Enter score: 80</td>
<td>Score 7 80.0 10.0</td>
</tr>
<tr>
<td>Enter score: 92</td>
<td>Score 8 92.0 22.0</td>
</tr>
<tr>
<td>Enter score: 69</td>
<td>Score 9 69.0 -1.0</td>
</tr>
</tbody>
</table>

Initial Values

- When array is created, all values are initialized depending on array type:
  - Numbers: 0
  - Boolean: false
  - Object References: null

- Means compiler does not force initialization like with primitives
Length

■ We assumed size of array was 10...what if we aren’t sure? Or want to change size?

■ Every array has a public constant data member length (no parenthesis)

```java
for(int i=0; i < scores.length; i++){
    System.out.print("Score "+i+" ");
    System.out.print(score[i]+" ");
    System.out.println(score[i] - avg);
}
```

Adv 8.1 Initializing arrays

■ You can also initialize all values right off the bat instead of individually as before

```java
```

■ No need to use new, or specify size
  ■ Knows size from number of values in list

Initialization

■ Say we wanted to store all the month names?

■ Can keep array of Strings

```java
String[] monthName = new String[12];
monthName[0] = "Jan";
monthName[1] = "Feb";
monthName[2] = "Mar";
monthName[3] = "Apr";
monthName[4] = "May";
monthName[5] = "Jun";
monthName[6] = "Jul";
monthName[7] = "Aug";
monthName[8] = "Sep";
monthName[9] = "Oct";
monthName[10] = "Nov";
monthName[11] = "Dec";
```

Array sizes

■ So far, we have used constants to define the size of the array
  ■ Fixed-size array declaration

■ What about if we aren’t sure how many scores we want?

■ Solution 1 – declare array of size 100 and get input until quit or 100 sizes are input

Array sizes

■ 2 problems
  ■ Inefficient – wastes space (what if we get 1 score? Wasted space for other 99) underutilization
  ■ Limiting – what if we want more than 100 scores?

■ Solution 2 – variable size array – non-constant values for size
  ■ But you still cannot change the size once created (length is constant)

User defined size

```java
int size;
double [] scores;
System.out.print("Enter number of scores to enter: ");
size = in.nextInt();
scores = new double[size];
```

■ Can use any mathematic expression
  ■ Must give a whole number (integer)
Array of Objects

BankAccount[] ba;  
System.out.print("Enter number of accounts: ");
int numAccounts = in.nextInt();
ba = new BankAccount[numAccounts];
ba[0].deposit(500);

What's wrong?

Objects and arrays

- We have created the array (i.e. the bins)
- What's in the bins?
- We still need to create an object for each bin

Memory Diagram

Create objects

- Each “bin” (index of the array) is a reference
- When we declare an array, we declare numAccounts identifiers in essence
- Now we need to create a BankAccount object for each bin

Array of Objects

BankAccount[] ba;  
System.out.print("Enter number of accounts: ");
int numAccounts = in.nextInt();
ba = new BankAccount[numAccounts];
ba[0] = new BankAccount();
ba[0].deposit(500);

Memory Diagram
Arrays of Objects

- We will use the Person class to illustrate this concept.

- An array of objects is declared and created just as an array of primitive data types is.
  
  ```java
  Person[] person;
  person = new Person[20];
  ```

Fig. 10.4 Wu

- An array of Person objects after the array is created.

Arrays of Objects

- The person array with one Person object added to it.

- To assign data values to this object, we can execute

  ```java
  person[0].setName("Ms. Latte");
  person[0].setAge (20);
  person[0].setGender ('F');
  ```

- The syntax here is the one used to call an object’s method; we are using an indexed expression to refer to the object instead of a simple variable.

The following program illustrates various aspects of array processing:

- Creating a new person array.
- Creating a new Person object and assigning values to it.
- Finding the average age of the persons in the array.
- Finding the youngest and oldest persons in the array.
- Finding a particular person in the array.

```java
/* Chapter 10 Sample Program: Illustrate the processing of an array of Person objects
File: Ch10ProcessPersonArray.java */

class Ch10ProcessPersonArray {
    public static void main (String[] args) {
        Person[] person; //declare the person array
        person = new Person[5]; //and then create it
Create person Array

```java
String name, inpStr;
int age;
char gender;

for (int i = 0; i < person.length; i++) {
    // read in data values
    System.out.println("Enter info for a person");
    name = in.nextLine();
    age = in.nextInt();
    inpStr = in.next().charAt(0);
    // create a new Person and assign values
    person[i] = new Person();
    person[i].setName(name);
    person[i].setAge(age);
    person[i].setGender(gender);
}
```

Compute Average Age

```java
float sum = 0, averageAge;
for (int i = 0; i < person.length; i++) {
    sum += person[i].getAge();
}
averageAge = sum / (float) person.length;
System.out.println("Average age: " + averageAge);
```

Find the youngest and oldest person

```java
Person youngest, oldest;
youngest = person[0];
oldest = person[0];
for (int i = 1; i < person.length; i++) {
    if (person[i].getAge() < youngest.getAge()) {
        // found a younger person
        youngest = person[i];
    } else if (person[i].getAge() > oldest.getAge()) {
        // found an older person
        oldest = person[i];
    }
}
System.out.println("Oldest: " + oldest.getName() + " is " + oldest.getAge() + " years old.");
System.out.println("Youngest: " + youngest.getName() + " is " + youngest.getAge() + " years old.");
```

Fig. 10.6

- An array of Person objects with two Person variables.

Deleting object

- How do we delete an object?

```java
tms[x] = null;
```

- Remember garbage collection takes any objects no longer being referenced to (i.e. can never be accessed again)

8.3 Wrappers

<table>
<thead>
<tr>
<th>Primitve Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
</tbody>
</table>
Wrapper classes

- Allow us to treat primitives as objects
- Each wrapper class has a primitive data member

8.4 Enhanced for Loop

- The traditional for loop – iterating through arrays

```java
double[] data = ...;  
double sum = 0;  
for (int i = 0; i < data.length; i++)  
{
    double e = data[i];  
    sum = sum + e;  
}
```

8.5 Simple Array Methods

- Finding Average

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

Count

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

Find Value

```java
double[] scores = ...;  
double search = ...;  
boolean found = false;  
for (int j = 0; j < scores.length; j++)  
{
    if (Math.abs(scores[j], search <= 1E-14)  
    {  
        found = true;  
    }  
}  
```
**Maximum/Minimum**

```java
double max = scores[0];
double min = scores[0];
for(int i = 1; i < scores.length; i++)
{
    if(scores[i] > max)
    {
        max = scores[i];
    }
    else if(scores[i] < min)
    {
        min = scores[i];
    }
}
```

---

**For Objects**

```java
public class Bank {
    public BankAccount find(int accountNumber) {
        for (int j = 0; j < accounts.length; j++)
        {
            if (accounts[j].getAccountNumber() == accountNumber) // Found a match
                return accounts[j];
        }
        return null; // No match in the array list
    }
}
```

---

**10.5 Dimensional Arrays**

- So far: Only one dimensional arrays
- Many problems may require us to represent data in a “table” format, or along two-dimensions
  - Two-dimensional arrays
- What do we need to add?

---

**2D Arrays**

- Two indices – x and y
- Example:
  - We have multiple students, and each student has multiple scores – row for student, column for test

```java
int[][] scores;
scores = new int[numStudents][numTests];
```
10.5 Two-Dimensional Arrays

To declare our example array, we state:

```java
double[][] payScaleTable;
```
or

```java
double payScaleTable[][];
```

and create the array as

```java
payScaleTable = new double[4][5];
```

---

2D Arrays

- Arrays are a specific structure
- Two dimensional arrays are not
  - They are an array of arrays
- How do you check number of rows? Columns? (using the length member)

---

10.5 Two-Dimensional Arrays

- The sample array creation
  ```java
  payScaleTable = new double[4][5];
  ```

is a shorthand for

```java
payScaleTable = new double[4][];
payScaleTable[0] = new double[5];
payScaleTable[1] = new double[5];
payScaleTable[2] = new double[5];
```

- and so on.

---

Fig. 10.16 Wu

- Accessing an element of a two-dimensional array.

<table>
<thead>
<tr>
<th>Row#</th>
<th>Column#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>36.50</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

---

Fig. 10.17A Wu

- Executing the statements on the left in sequence will create the array of arrays shown on the right.

1. ```java
   payScaleTable = new double[][];
   ```
2. ```java
   payScaleTable[0] = new double[];
   ```
3. ```java
   payScaleTable[1] = new double[];
   ```
4. ```java
   payScaleTable[2] = new double[];
   ```
5. ```java
   payScaleTable[3] = new double[];
   ```
6. ```java
   payScaleTable[4] = new double[];
   ```
10.5 Two-Dimensional Arrays

The expression

\[
payScaleTable.length
\]

refers to the length of the payScaleTable array itself.

<table>
<thead>
<tr>
<th>payScaleTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>payScaleTable.length == 4</td>
</tr>
</tbody>
</table>

10.5 Two-Dimensional Arrays

- The expression
  \[
payScaleTable[1].length
\]
  refers to the length of the array stored at row 1 of payScaleTable.

8.7 Copying Arrays

- Arrays are reference variables, treat like other objects
- Copying an array variable yields a second reference to the same array

```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```

Clone

- Use clone to make true copy
  ```
  double[] prices = (double[]) data.clone();
  //Note clone returns an object, so we type // cast
  ```

Figure 7  Two References to the Same Array
Copying Elements

- System.arraycopy() not encouraged
- Hard to manipulate
- Using for loops is more powerful

Common use: insert

//insert x in position i of newData
//we already have an array called
int newData = new int[data.length+1];

//<loop to copy elements before i>
for (int pos=0; pos<i; pos++)
    newData[pos] = data[pos];

//insert the new element
newData[i] = x;

//<loop to copy elements after i>
for (int pos=i+1; pos<newData.length; pos++)
    newData[pos] = data[pos-1];

data = newData;

Common use: insert

//insert x in position i of newData
//we already have an array called
int newData = new int[data.length+1];

//<loop to copy elements before i>
for (int pos=0; pos<i; pos++)
    newData[pos] = data[pos];

//insert the new element
newData[i] = x;

//<loop to copy elements after i>
for (int pos=i+1; pos<newData.length; pos++)
    newData[pos] = data[pos-1];

data = newData;

Common use: insert

//insert x in position i of newData
//we already have an array called
int newData = new int[data.length+1];

//<loop to copy elements before i>
for (int pos=0; pos<i; pos++)
    newData[pos] = data[pos];

//insert the new element
newData[i] = x;

//<loop to copy elements after i>
for (int pos=i+1; pos<newData.length; pos++)
    newData[pos] = data[pos-1];

data = newData;

Copy elements

How do I copy count positions from array from to array to, starting at positions fromStart and toStart respectively?
Growing array

- If the array is full and you need more space, you can grow the array.
- Create a new, larger array.
  
  ```java
  double[] newData = new double[2 * data.length];
  ```
- Copy all elements into the new array.
- Store the reference to the new array in the array variable.
  
  ```java
  data = newData;
  ```

Passing Arrays to Methods

- Exactly the same as objects, the reference is passed to method parameters.
- And for returning an object.

- length is especially important in methods, since we have no other way of knowing the size (except sending another parameter).

```java
public int searchMinimum(double[] number) {
    int indexOfMinimum = 0;
    for (int i = 1; i < number.length; i++) {
        if (number[i] < number[indexOfMinimum]) {
            // found a smaller element
            indexOfMinimum = i;
        }
    }
    return indexOfMinimum;
}
```

To call this method (from a method of the same class), we write:

```java
double[] arrayOne;
// create and assign values to arrayOne...

// get the index of the smallest element of arrayOne
int minOne = searchMinimum(arrayOne);

// output the result
System.out.print("Minimum value in Array One is ");
System.out.print(arrayOne[minOne] + "at position " + minOne);
...
```
Adv 8.4 Partially Filled Arrays

- Use partially filled arrays to deal with sets where size is not known beforehand
- MUST keep track of capacity and size to ensure array does not get overfilled
- Make size large
- Either resize array or enforce a limit when full

```java
int capacity = 100;
double[] data = new double[capacity];
boolean stop = false;
while(!stop)
{
    System.out.print("Enter data: ");
    if(in.hasNextDouble())
    {
        double x = in.nextDouble();
        if(dataSize >= capacity)
        {
            capacity *= 2;
            double[] newData = new double[capacity];
            for(int i = 0; i < data.length; i++)
            { newData[i] = data[i];
            data = newData;
        }
data[dataSize] = x;
dataSize++;
    }
    else
    {
        stop = true;
    }
}
```

Example: Candy in a Machine

- Write a code fragment to increase the price of all items by 5 cents.

```java
for ( int i=0; i < machine.length; i++ )
    for ( int j=0; j < machine[i].length; j++ )
        machine[i][j] = machine[i][j] + 0.05;
```