Chapter 5

Arrays

One-Dimensional Arrays

- In an array, all data items (known as ***elements***) must have the same type.
- An array can be visualized as a series of boxes, each capable of holding a single value belonging to this type:

- An array whose elements are arranged in a linear fashion is said to be ***one-dimensional.***

Creating Arrays

- An array declaration contains `[]`, element can be of any type, e.g., objects:

```
int[] a;                    or int a[];
String[] b;                 or String b[];
```

- Declaring an array variable doesn't allocate space for the array's elements. One way to allocate this space is to use `new` keyword: `a = new int[10];`
- Be careful not to access the elements of an array before the array has been allocated. Doing so will cause a **`NullPointerException`** to occur.

Creating Arrays

- Allocate space when the array is declared:

```
int[] a = new int[10];
int n = 10;
int[] a = new int[n];
```

- An array can be **initialized** at the time it's declared:

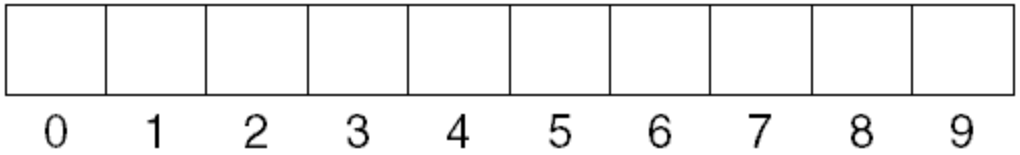
```
int[] a = {3, 0, 3, 4, 5};
```

  - The word `new` isn't used if an initializer is present.

- When an array is created using `new`, the elements of the array are given default values:
  - Numbers are set to zero.
  - `boolean` elements are set to `false`.
  - Array and object elements are set to `null`.

Visualizing Arrays

- Each array element has an ***index,*** or ***subscript,*** that specifies its position within the array.
  - only the numbers between 0 and $n - 1$ are valid indexes.
  - $n=9$



- `a[i]` represents the `i`*th* element in array `a`.
- An array subscript can be any expression, provided that it evaluates to an `int` value.

```
a[0], a[i], a[2*i-1]
```

Array Subscripting

- Access a nonexistent array element causes an error named `ArrayIndexOutOfBoundsException`.
- An array element behaves just like a variable of the element's type:

```
a[i] = 10;
System.out.println(a[i]);
```

- If the elements of an array are objects, they can call instance methods.
  - if the array `b` contains `String` objects, the call `b[i].length()` would return the length of the string stored in `b[i]`.

Processing the Elements in an Array

- The number of elements in an array `a` is given by the expression `a.length`.
- A loop that adds up the elements in the array `a`, leaving the result in the `sum` variable:

```
int sum = 0;
int i = 0;
while (i < a.length) {
  sum += a[i];
  i++;
}
```

Exercise: Write a Program

- The `MaxScores` program generates n=10 random scores in 0 – 100, computes the average and finds the max score. Output the average and max scores. of a series of scores entered by the user:

- Try 1: ask the users to type in n and the scores from the keyboard
- Try2: check the validity of user's type and repeat until legal input is obtained.

General Form of the `for` Statement

- Form of the `for` statement:

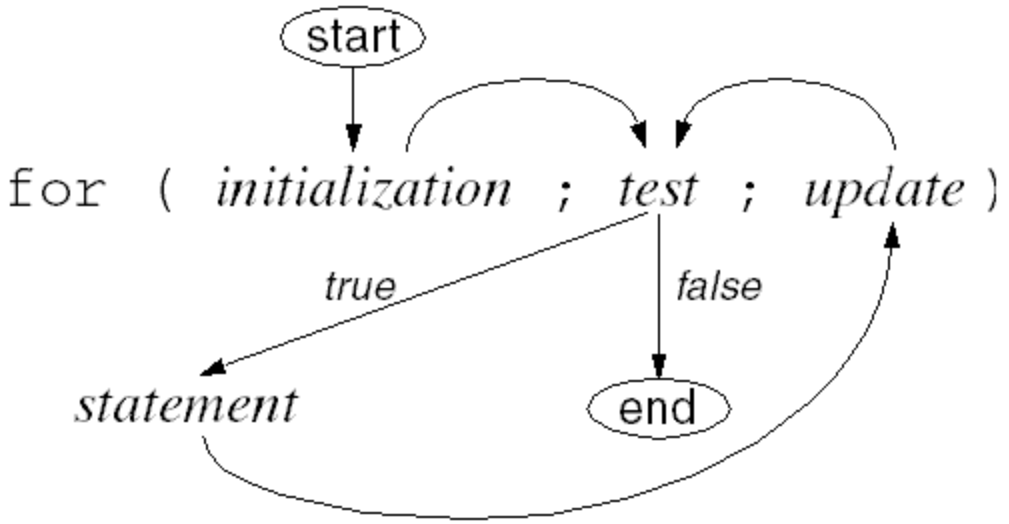
  `for` (*initialization* ; *test* ; *update*)
  statement

- *Initialization* is an initialization step that's performed once, before the loop begins to execute.
- *Test* controls loop termination (the loop continues executing as long as *test* is true).
- *Update* is an operation to be performed at the end of each loop iteration.

`for` Statements Versus `while` Statements

is equivalent to the following `while` loop:

```
initialization ;
while ( test ) {
    statement
    update ;
}
```



`for` Statements Versus `while` Statements

- A `for` statement:

```
for (i = 10; i > 0; i--)
   System.out.println("T minus " + i +
                      " and counting");
```

- An equivalent `while` statement:

```
i = 10;
while (i > 0) {
   System.out.println("T minus " + i +
                      " and counting");
   i--;
}
```

- uses `--i` instead of `i--`

`for` Statement Idioms

- Typical ways to write a `for` statement

Counting up from 0 *to* `n` – 1:	`for (i = 0; i < n; i++) …`
Counting up from 1 *to* `n`:	`for (i = 1; i <= n; i++) …`
Counting down from `n` – 1 *to* 0:	`for (i = n - 1; i >= 0; i--) …`
Counting down from `n` *to* 1:	`for (i = n; i >= 1; i--) …`

- *initialization, test,* and *update* parts need not be related.
- The three parts that control a `for` loop are optional—any or all can be omitted.
  - When both *initialization* and *update* are omitted, same as `while` loop
  - If the *test* part is missing, it defaults to `true`, so the `for` statement doesn't terminate, `for (;;)`
  - The `break` cause the loop to terminate.

Declaring Control Variables

- For convenience, the *initialization* part of a `for` statement may declare a variable:

```
for (int i = 0; i < n; i++)
   …
```

- A variable declared in this way can't be accessed outside the loop. (The variable isn't ***visible*** outside the loop.)
- It's illegal for the enclosing method to declare a variable with the same name. It *is* legal for two `for` statements to declare the same variable, however.

Declaring Control Variables

- Having a `for` statement declare its own control variable is usually a good idea. It's convenient, and it can make programs easier to understand.
- More than one variable can be declared in initialization, provided that all variables have the same type:

```
for (int i = 0, j = 0; i < n; i++)
  …
```

Commas in `for` Statements

- In a `for` statement, both *initialization* and *update* are allowed to contain commas:

```
for (i = 0, j = 0; i < n; i++, j += i)
  ...
```

- Any number of expressions are allowed within *initialization* and *update,* provided that each can stand alone as a statement.
- When expressions are joined using commas, the expressions are evaluated from left to right.
- Using commas in a `for` statement is useful primarily when a loop has two or more counters.

Searching for a Particular Element

- One common array operation is searching an array to see if it contains a particular value:

```
int i;
for (i = 0; i < scores.length; i++)
  if (scores[i] == 100)
    break;
```

- An `if` statement can be used to determine whether or not the desired value was found:

```
if (i < scores.length)
  System.out.println("Found 100 at position " + i);
else
  System.out.println("Did not find 100");
```

Processing Array Counting Occurrences

- Counting number occurrences

```
int count = 0;
for (int i = 0; i < scores.length; i++)
  if (scores[i] == 100)
    count++;
```

- Finding the largest (or smallest element)

```
int smallest = scores[0];
for (int i = 1; i < scores.length; i++)
  if (scores[i] < smallest)
    smallest = scores[i];
```

Exercise: Write a Program

- The `RepeatedDigits` program will determine which digits in a number appear more than once
- The program will examine `number`'s digits one at a time, incrementing one of the elements of `digitCounts` each time, using the statement

```
digitCounts[number%10]++;
```

- If `number` is originally 392522459, the `digitCounts` array will have the following appearance:

0	0	3	1	1	2	0	0	0	2
0	1	2	3	4	5	6	7	8	9

- Try 1: use Scanner
- Try2: how to count repeated number in a string

RepeatedDigits.java

```
// Checks a number for repeated digits

import jpb.*;

public class RepeatedDigits {
  public static void main(String[] args) {
    // Prompt user to enter a number and convert to int
   form
    SimpleIO.prompt("Enter a number: ");
    String userInput = SimpleIO.readLine().trim();
    int number = Integer.parseInt(userInput);

    // Create an array to store digit counts
    int[] digitCounts = new int[10];

    // Remove digits from the number, one by one, and
    // increment the corresponding array element
    while (number > 0) {
      digitCounts[number%10]++;
      number /= 10;
    }
```

```
    // Create a string containing all repeated digits
    String repeatedDigits = "";
    for (int i = 0; i < digitCounts.length; i++)
      if (digitCounts[i] > 1)
        repeatedDigits += i + " ";

    // Display repeated digits. If no digits are repeated,
    // display "No repeated digits".
    if (repeatedDigits.length() > 0)
      System.out.println("Repeated digits: " +
                         repeatedDigits);
    else
      System.out.println("No repeated digits");
  }
}
```

Using Arrays as Vectors

- A `for` statement scaling: $\alpha\mathbf{A} = [\alpha a_1\ \alpha a_2\ \ldots\ \alpha a_n\ ]$

```
double[] a = new double[n];
for (int i = 0; i < a.length; i++)
  a[i] *= alpha;
```

- The ***inner product,*** or ***dot product,*** of **A** and **B** is defined as follows:

$$\mathbf{A} \cdot \mathbf{B} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

- A loop that calculates the inner product

```
double innerProduct = 0.0;
for (int i = 0; i < a.length; i++)
  innerProduct += a[i] * b[i];
```

Parallel Arrays

- The first technique for storing a database is to use ***parallel*** arrays, one for each field.
- For example, the records in a phone directory would be stored in three arrays:

0	AARON Robin B	0	4011 Stone Mountain St	0	384-7110
1	ABBOTT C Michael	1	981 Glen Arden Way	1	776-5188
2	ABEL A B	2	343 Lakeshore Ct	2	871-7406
3	ABERCROMBIE Bill	3	5810 Lismoor Tr	3	844-9400
4	ABERNATHY C	4	2120 Martin Rd	4	779-7559
5	ABRAHAM Gary	5	585 Chandler Pond Dr	5	582-6630
	⋮		⋮		⋮

Parallel Arrays

```
int[] xCoordinates = new int[100];
int[] yCoordinates = new int[100];
```

- The values of `xCoordinates[i]` and `yCoordinates[i]` represent a single point.
- Parallel arrays can be useful. However, they suffer from two problems:
  - It's better to deal with one data structure rather than several.
  - Maintenance is more difficult. Changing the length of one parallel array requires changing the lengths of the others as well.

Arrays of Objects

- The alternative to parallel arrays is to treat each record as an object, then store those objects in an array.
- A `PhoneRecord` object could store a name, address, and phone number.
- A `Point` object could contain instance variables named `x` and `y`. (The Java API has such a class.)
- An array of `Point` objects:

```
Point[] points = new Point[100];
```

Creating a Database

- Consider the problem of keeping track of the accounts in a bank, where each account has an account number (a `String` object) and a balance (a `double` value).
- One way to store the database would be to use two parallel arrays:

```
String[] accountNumbers = new String[100];
double[] accountBalances = new double[100];
```

- A third variable would keep track of how many accounts are currently stored in the database:

```
int numAccounts = 0;
```

Creating a Database

- Statements that add a new account to the database:

```
accountNumbers[numAccounts] = newAccountNumber;
accountBalances[numAccounts] = newBalance;
numAccounts++;
```

- `numAccounts` serves two roles. It keeps track of the number of accounts, but it also indicates the next available "empty" position in the two arrays.

Creating a Database

- Another way to store the bank database would be to use a single array whose elements are `BankAccount` objects.
- The `BankAccount` class will have two instance variables (the account number and the balance).
- `BankAccount` constructors and methods:

```
public BankAccount(String accountNumber,
                   double initialBalance)
public void deposit(double amount)
public void withdraw(double amount)
public String getNumber()
public double getBalance()
```

Creating a Database

- `BankAccount` objects will be stored in the `accounts` array:

```
BankAccount[] accounts = new BankAccount[100];
```

- `numAccounts` will track the number of accounts currently stored in the array.
- Fundamental operations on a database:
  - Adding a new record
  - Deleting a record
  - Locating a record
  - Modifying a record

Adding a Record to a Database

- Adding a record to a database is done by creating a new object and storing it in the array at the next available position:

```
accounts[numAccounts] =
  new BankAccount(number, balance);
numAccounts++;
```

- The two statements can be combined:

```
accounts[numAccounts++] =
  new BankAccount(number, balance);
```

- In some cases, the records in a database will need to be stored in a particular order.

Removing a Record from a Database

- When a record is removed from a database, it leaves a "hole"—an element that doesn't contain a record.
- The hole can be filled by moving the last record there and decrementing `numAccounts`:

```
accounts[i] = accounts[numAccounts-1];
numAccounts--;
```

- These statements can be combined:

```
accounts[i] = accounts[--numAccounts];
```

- This technique works even when the database contains only one record.

Searching a Database

- Searching a database usually involves looking for a record that matches a certain "key" value.
- Statements that search the `accounts` array for a record containing a particular account number:

```
int i;
for (i = 0; i < numAccounts; i++)
  if (accounts[i].getNumber().equals(number))
    break;
```

- Once the loop has terminated, the next step is to test whether `i` is less than `numAccounts`. If so, the value of `i` indicates the position of the record.

Modifying a Record in a Database

- A record can be updated by calling a method that changes the object's state.
- A statement that deposits money into the account located at position `i` in the `accounts` array:

```
accounts[i].deposit(amount);
```

- It's sometimes more convenient to assign an array element to a variable, and then use the variable when performing the update:

```
BankAccount currentAccount = accounts[i];
currentAccount.deposit(amount);
```

Arrays as Objects

- Like objects, arrays are created using the `new` keyword.
- Arrays really *are* objects, and array variables have the same properties as object variables.
- An object variable doesn't actually store an object. Instead, it stores a ***reference*** to an object. Array variables work the same way.

Properties of Object Variables

- Object variables have the following properties:
  - When an object variable is declared, it's not necessary for the variable to refer to an object immediately.
  - The value of an object variable can be changed as often as desired.
  - Several object variables can refer to the same object.
  - When no variable refers to an object, it becomes eligible for garbage collection.
  - Assigning one object variable to another causes only a reference to be copied; no new object is created.
  - Testing whether two object variables are equal or not equal compares the references stored in the variables.

How Arrays are Stored

- An array variable contains a reference to where the array's elements are stored.
- Storage for an array named `a` containing 10 integers:

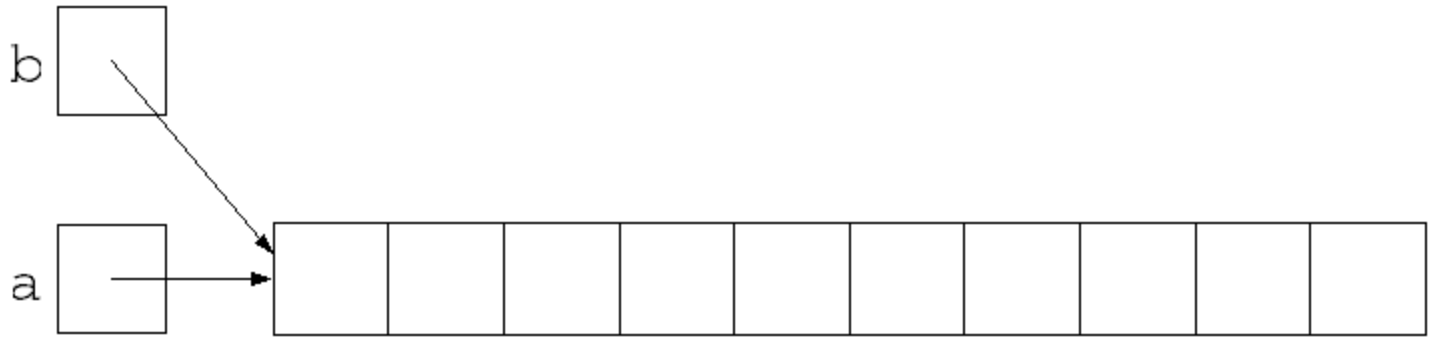
- Arrays are "garbage collected" in the same way as other objects. When there are no more references to an array, the space occupied by the array can be reclaimed automatically.

Copying Arrays

- If `a` and `b` are array variables of the same type, it's legal to write

  ```
  b = a;
  ```

- The effect is that `b` now contains a reference to the same array as `a`:



Copying Arrays

- The assignment operator doesn't make a true copy of an array. To make a genuine copy, there are two strategies:
  - Create a new array of the same length as the old one and copy the elements from the old array to the new one.
  - Use the `clone` method.
- Testing whether two array variables are equal (or not equal) is legal. However, this only checks whether the two variables refer to the same array.
- Checking whether two arrays contain identical elements requires writing a loop.

Resizing an Array

- Although arrays have fixed sizes, it's possible to resize an array if it becomes full. The trick is to create an entirely new array to replace the old one.
- Resizing an array takes three steps:
  1. Create a new array that's larger than the old one.
  2. Copy the elements of the old array into the new array.
  3. Assign the new array to the old array variable.

Resizing an Array

- Code that doubles the size of the `accounts` array:

```
BankAccount[] tempArray =
  new BankAccount[accounts.length*2];
for (int i = 0; i < accounts.length; i++)
  tempArray[i] = accounts[i];
accounts = tempArray;
```

- Doubling the size of an array provides plenty of space for new elements, yet guarantees that there won't be too much unused space.

Resizing an Array

- When an array is resized, the old array can be reclaimed by the garbage collector.
- Copying elements from the old array to the new usually doesn't take that long. If the elements are objects, only references to the objects are copied, not the objects themselves.
- For additional speed, Java provides a method named `System.arraycopy` that can be used to copy elements from one array to another.

Resizing an Array

- A call of `System.arraycopy` that copies the elements of `accounts` into `tempArray`, starting from position 0 in both arrays:

```
System.arraycopy(accounts, 0,
tempArray,
                 0, accounts.length);
```

  The last argument is the number of elements to be copied.

- Instances of Java's `Vector` class behave like arrays that automatically grow when they become full.

Exercise: Write a Program

- In Test2-Studyguide, finish the `NFLTeam3` and `NFLGameDay3` programs
- Add an array to hold player names into class `NFLTeam3`
- Add a method `addAplayer(String name)` into class `NFLTeam3`
- Add at least 2 players to each team in `NFLGameDay3`

- Try 1: add a method `deleteAplayer(String name)` into class `NFLTeam3` and test it.