Introduction to Computer Science
Shimon Schocken
IDC Herzliya

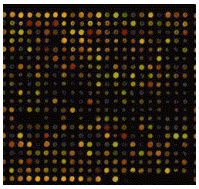
Lecture 4-1

Arrays I

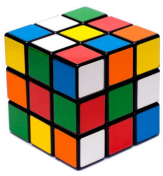
Why arrays?

- So far, our programs contained objects like `grade` and `turtle`
- We will now learn how to create and process objects like `grade[100]` and `turtle[20]`
- Such objects are called <u>arrays</u>
- Arrays come handy when you have to store and process an indexed collection of objects of the same type.

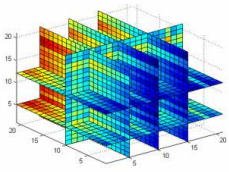
1-dimensional array



2-dimensional array



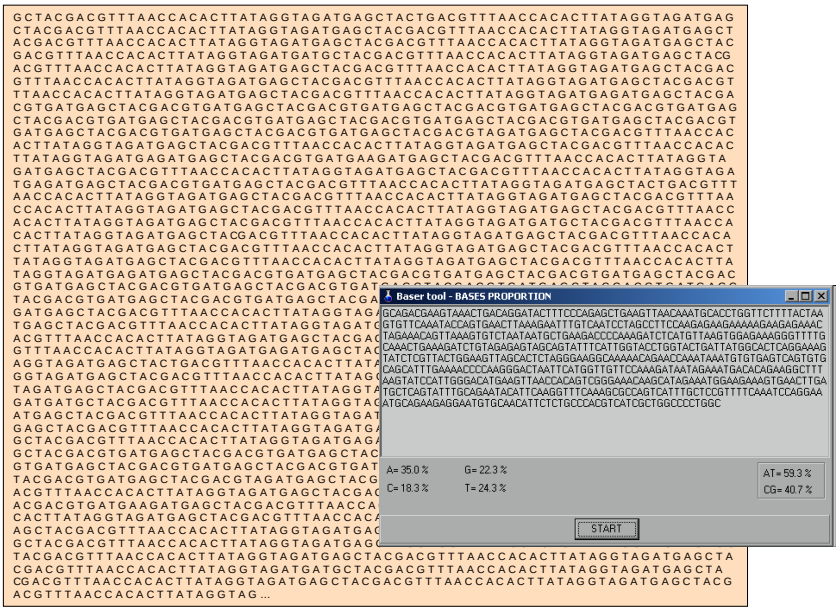
3-dimensional array



n-dimensional array
(impossible to draw)

Why arrays?

DNA file:



Outline

- Array processing examples
- Array declaration
- For … each
- More array processing examples
- Initializer lists
- Strings vs arrays
- Multi-dimensional arrays
- Command line arguments

Array processing: examples

DNA file: G C T A C G A C G T T T A A C C A C A C T T A T A G G T A G A T G A ...

This sequence can be viewed as an indexed collection of bases:

	0	1	2	3	4	5	6	7	8	9	...
DNA array:	G	C	T	A	C	G	A	C	G	T	...

Array processing lends itself to questions like:

- Which base appears in location 512 ?
- Which pattern appears in locations 310 to 315 ?
- Where is the last location of base A ?
- How many times each base appears in the sequence ?
- Does the pattern CGT appear in the sequence ?
- Does the pattern C?T appear (where ? stands for any one base) ?
- Does the pattern C*T appear (where * stands for any one or more bases) ?
- Given two dna sequences s1 and s2, compute the function
  similarity(s1,s2) = % of locations in which s1 and s2 have identical bases
- And so on.

Array processing: examples

	0	1	2	3	4	5	6	7	8	9	...
DNA array:	G	C	T	A	C	G	A	C	G	T	...

```
// Normally, we will get the DNA array data from a file.
char[] dna = {'G','C','T','A','C','G','A','C','G','T'};  // For testing purposes
char base;
String pattern;

// Which base appears in location 5?
base = dna[5];    // 'G'

// Which pattern appears in locations 3 to 6?
pattern = dna[3] + dna[4] + dna[5] + dna[6];     // 'ACGA'

// Mutation ...
dna[7] = 'A'

// Mutation: switch bases 5 and 6
{char temp = dna[5]; dna[5] = dna[6]; dna[6] = temp;}
```

Mutable objects: Have a changeable state

Immutable objects: Once created, their state cannot be changed

Arrays are mutable objects

Array processing: examples

	0	1	2	3	4	...	99
Grades array:	55	78	90	45	71	...	85

```
int[] grades = new int[100];
// Suppose that the array was populated with data

grades[6] = 75;
grades[3] = grade[3] + 5;
grades[4] = grades[4] * 1.1;
int x = (100 + grades[12]) / 2;
```

- If an array is of type `int`, then any one of its elements can be treated as an `int` variable and can play any role that an `int` variable is allowed to play in Java
- Same rule holds for any other array type.

Array processing: examples

	0	1	2	3	4	5	6	7	8	9	...
DNA array:	G	C	T	A	C	G	A	C	G	T	...

```
// Which pattern appears in locations 4 to 6 ?
String pattern = "";
for (int i = 4; i <= 6; i++)
    pattern = pattern + dna[i];
System.out.println(pattern);

// Which is the last location of base A?
int found = -1;
for (int i = 0; i < dna.length; i++)
    if (dna[i] == 'A')
          found = i;
System.out.println(found);
```

In Java, each array object has a length field holding it's length.

Output:

```
CGA
6
```

for loops provide a natural way to process arrays.

Outline

- Array processing examples
- Array declaration
- For … each
- More array processing examples
- Initializer lists
- Strings vs arrays
- Multi-dimensional arrays
- Command line arguments

Array declaration

Syntax:

```
type[] arrayName = new type [arrayLength]
```

Examples

```
char[] dna = new char[1000];
int[] grades = new int[100];
boolean[] switch;
switch = new boolean[30];

String[] words = new String[500];
Turtle[] turtles = new Turtle[10];
WindMill[] windMills = new windMill[100];
```

Arrays of values of primitive types

Arrays of objects (contain references)

```
// move the third turtle 50 steps forward:
turtles[2].moveForward(50);

// Compute the total gain of the wind mill farm
int yield = 0;
for (int i = 0; i <= windMill.length; i++)
    yield = yield + windMill[i].getYield();
```

For each …

```
// Instead of:
for (int i = 0; i <= grades.length; i++)
    System.out.println(grades[i]);

// Use:
for (int grade : grades)
    System.out.println(grade);

// Instead of:
for (int i = 0; i <= turtles.length; i++)
    turtles[i].hide();

// Use:
for (Turtle t : turtles)
    t.hide();

// Instead of:
for (int i = 0; i <= windMill.length; i++)
  yield = yield + windMill[i].getYield();

// Use:
for (WindMill w : windMills)
  yield = yield + w.getYield();
```

Best practice advice:

Whenever possible, use `for … each` loops

Why?

- Readable
- Elegant
- Avoids variables

But, In some cases the standard `for` loop must be used, e.g. when the array index plays a role in the array processing.

Array processing: examples

```
import java.util.Scanner;
public class ArrayDemo {
  public static void main(String[] args){

    // Declares an array of a user-defined size
    int[] elements;
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter number of elements: ");
    elements = new int[scan.nextInt()];

    // Prompts the user to enter the array elements
    for (int j = 0 ; j < size ; j++) {
      System.out.print("enter element " + j + ": ");
      elements[j] = scan.nextInt();
    }

    // Prints the array elements in reverse order
    for (int j = (elements.length - 1); j >= 0; j--)
      System.out.println(elements[j]);
    }
  }
}
```

Prompts the user for a series of values, then prints them backwards

```
C:\WINDOWS\system32\cmd.exe

D:\demo>java ArrayDemo1
Enter number of elements: 7
enter element 0: 10
enter element 1: 20
enter element 2: 15
enter element 3: 7
enter element 4: 90
enter element 5: 2
enter element 6: 103
103
2
90
7
15
20
10

D:\demo>_
```

Array processing: examples

```
import java.util.Scanner;
public class DigitsCount {

  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter a number: ");
    int number = scan.nextInt();
    int[] digitCounter = new int[10];

   // For each occurrence of each digit, increment its counter
   while (number > 0) {
       int digit = number % 10;
       digitCounter[digit]++;
       number = (int) (number / 10);
   }
    // Print the histogram
    for (int i = 0; i < 10; i++) {
      System.out.print(i + " ");
      for (int j = 0; j < digitCounter[i]; j++)
        System.out.print("*");
      System.out.println();
      }
    }
  }
```

Counts how many times each digit appears in a given number, and prints the results as a histogram

```
C:\WINDOWS\system32\cmd.exe

D:\demo>javac DigitsCount.java

D:\demo>java DigitsCount
Enter a number:
2276326212903200
0 ***
1 *
2 ******
3 **
4
5
6 **
7 *
8
9 *
```


Outline

- Array processing examples
- Array declaration
- For … each
- More array processing examples
- Initializer lists
- Strings vs arrays
- Multi-dimensional arrays
- Command line arguments

Initializer lists

Examples:

```
int[] somePrimes = {3, 5, 7, 11, 13, 17, 19, 23, 29, 31};
char[] vowels = {'a', 'o', 'i', 'u', 'e'};
```

The rules of the game:

- An initializer list is an array
- When defining an initializer list, there is no need to use `new`
- The array length is determined from the size of the initialzer list
- An initializer list can be defined only as part of the array declaration statement (and not later in the code).

A String is not an array of characters!

```
char[] c = {'I','D','C',' ','H','E','R','Z','L','I','Y','A'};

String s = new String(c);

System.out.println(c);  // "IDC HERZLIYA"
System.out.println(s);  // "IDC HERZLIYA"

System.out.println(c[5]);  // 'E'
System.out.println(s[5]);  // Error
System.out.println(s.charAt(5)); // 'E'

c[5] = 'G' // OK
s[5] = 'G' // Error

System.out.println(c.length);  // 12
System.out.println(s.length());  // 12
```

A string can be *constructed from* an array of characters.

However, this does not imply that a string is an array of characters.

Strings are immutable;
Array entries can be changed

Array processing: examples

```
import java.util.Scanner;
public class FindVowels {
    static char[] vowels = {'a', 'e', 'i', 'o', 'u'};

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter a word: ");
        String word = scan.nextLine();
        System.out.println("The word has " +
                    (containsVowel(word) ? "" : "no ") + "vowel(s)");
    }

    public static boolean containsVowel(String s) {
        for (int j = 0 ; j < s.length() ; j++)
            for (int k = 0 ; k < vowels.length ; k++)
                if (s.charAt(j) == vowels[k])
                    return true;
         return false;
    }
}
```

Checks if a given string has one or more vowels in it.

```
C:\WINDOWS\system32\cmd.exe
D:\demo>java FindVowels
Enter a word: baby
The word has vowel(s)

D:\demo>java FindVowels
Enter a word: gypsy
The word has no vowel(s)
```


Outline

- Array processing examples
- Array declaration
- For … each
- More array processing examples
- Initializer lists
- Strings vs arrays
- Multi-dimensional arrays
- Command line arguments

Multidimensional data

Math conventions:

$V = (3 \quad -19 \quad 7 \quad 13 \quad 8)$

V_2 holds the value -19

$$M = \begin{pmatrix} 5 & 2 & 7 & 3 & 8 & 5 \\ 2 & 4 & 8 & 12 & 7 & -3 \\ 3 & 19 & 5 & 8 & 5 & 1 \\ -2 & 3 & 6 & 4 & 9 & 5 \end{pmatrix}$$

$m_{2,4}$ holds the value 12

Java conventions:

v: (3 -19 7 13 8)

`v[2]` holds the value 7

$$m: \begin{pmatrix} (5 & 2 & 7 & 3 & 8 & 5) \\ (2 & 4 & 4 & 12 & 7 & -3) \\ (3 & 19 & 5 & 8 & 5 & 1) \\ (-2 & 3 & 6 & 4 & 9 & 5) \end{pmatrix}$$

`m[2][4]` holds the value 8

2D Array processing: examples

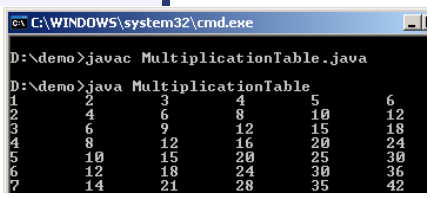
```
public class MultiplicationTable {
   public static void main(String[] args){
        final int N = 10;

        int[][] table = new int[N][N];
        for (int j = 0; j < table.length; j++)
            for (int k = 0; k < table[0].length; k++)
                table[j][k] = j * k;


        for (int j = 1; j < N; j++){
            for (int k = 1; k < N ; k++)
                System.out.print(table[j][k] + "\t");
            System.out.println();
        }
   }
}
```

In Java,
a multidimensional array is implemented as an array of arrays

In this particular example, all the dimensions (array lengths) have the same size: `N`.



2D Array processing: examples

```
public class DiagonalMatrix {
  public static void main(String[] args){

    int[][] mat = new int[10][];


    for (int j = 0; j < mat.length; j++)
       mat[j] = new int[j+1];

    for (int j = 0; j < mat.length; j++)
       for (int k = 0; k < mat[j].length; k++)
          mat[j][k] = j * k;


     for (int j = 0; j < mat.length; j++) {
        for (int k = 0; k < mat[j].length; k++)
           System.out.print(mat[j][k] + " ");
        System.out.println();
     }

  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\demo>javac DiagonalMatrix.java

D:\demo>java DiagonalMatrix
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

In this example, we also have a two-dimensional array, but here each row has a different length.

Initializing a multi-dimensional array

A multidimensional array can be declared and initialized using an initializer list:

Example:

```
int[][] hadamardMatrix = {
              { 1,  1,  1, 1},
              {-1,  1, -1, 1},
              {-1, -1,  1, 1},
              {-1, -1, -1, 1}
        };
```

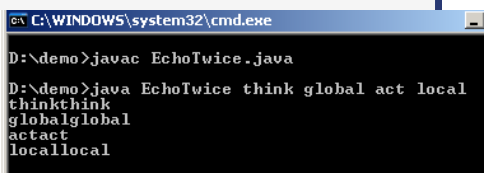
Note that the initializer list is a list of lists:

`{{...},{...}, ... ,{...}}`

Command line arguments

Example:

```
public class EchoTwice {
    public static void main(String[] args) {
        for (String s : args)
            System.out.println(s + s);
    }
}
```



When the user invokes a class from the OS shell, she can pass a line of parameters to the main method using the syntax:

```
Java ClassName parametersLine
```

The compiler breaks `parametersLine` into tokens, and stores them in the entries of the `args` array

The `args` array is accessible to the program's code like any other array.