Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120A - Fall 2004

Lecture Unit 9

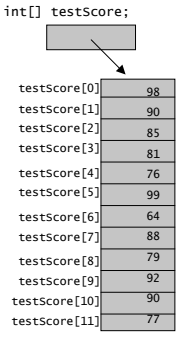
Arrays

Arrays

- Special construct in Java
- Variable that holds a reference to a collection of similar objects
- Arrays are made up of *elements* which are accessed by *position*
  - as opposed to fields in a class definition, which are accessed by name
- Problem: Read 10 words from an input file and print them in reverse order on the screen
  - How about reading 1,000 words?

Array Elements

- Can be either primitive type or object reference
- Indexed starting a position 0 (not 1)
  - Index is often referred to as the subscript
- All elements are homogenous (must be of the same type/class)



Declaring an Array

```
<type> []  <variable-name>;
```

- <type> can be a primitive type (int, char, boolean) or a class (String, Board) or even another array
- Square brackets indicate that you are defining an array variable
- Example:

```
double[] examScores;
```


Creating (Initializing) an Array

```
<variable-name> = new <type>[ <size> ];
```

- Example:

```
examScores = new double[ 10 ];
```

- Tells Java to reserve memory for 10 `doubles`
- If you know the size of the array, you can declare and initialize it at the same time:

```
double[] examScores = new double[ 10 ];
```


Declaring and Creating an Array with an Initializer List

```
double[] examScores = { 99.0, 78.5, 85.2, 91.1 };
```

- Does not use the "new" operator, nor do you need to specify the size of the array
  - Creates an array of four double elements, examScores[0] is 99.0, examScores[1] is 78.5, ...
- Types of the elements in the initializer list must match the type of the array
- If you don't use an initializer list, array elements are initialized to default values:
  - 0 for primitive numeric types
  - false for boolean types
  - null for references types

Accessing and Updating Elements

- Use the subscript operator [ ]

```
double thirdscore = examScores[ 3 ];
examScores[1] = 87.4;
i = 4;
examScores[i] = 94.3;
examScores[i*3/2] = 85.0;
System.out.println(examScores[3]);
```

- The index (or subscript) can be an integer constant, a variable, or any integer expression

Array Example

```
BufferedReader inFile = ... ;
PrintWriter outFile = ... ;

int[] value = new int[1000];
int number = 0;

while (number < 1000) {
  value[number] = Integer.parseInt(inFile.readLine());
  number++;
}

number = 999;
while (number >= 0) {
  outFile.println(value[number]);
  number--;
}
```


For Loops

- Simplified syntax for writing count-controlled loops:

```
number = 0;
while (number < 1000) {
  value[number] = Integer.parseInt(inFile.readLine());
  number++;
}
```

- can be rewritten as

```
for (number = 0; number < 1000; number++) {
  value[number] = Integer.parseInt(inFile.readLine());
}
```


For Loop Syntax

```
for (<init>; <condition>; <update>)  <body>
```

- The <init> statement is only executed once; when the for statement is first encountered
- The <condition> is tested each iteration of the loop, before the <body> is executed
- After executing the <body>, the <update> statement is performed before beginning the next iteration

```
for (number = 0; number < 1000; number++) {
  value[number] = Integer.parseInt(inFile.readLine());
}
```


Nested For Loops

- What does this do?

```
for (lastNum = 1; lastNum <= 7; lastNum++) {
  for (num = 1; num <= lastNum; num ++) {
   System.out.print(num);
  }
  System.out.println();
}
```


For Loop Details

```
for (<init>; <condition>; <update>)  <body>
```

- For loops can be used to write the equivalent of any while loop
- <init>, <condition>, and <update> are optional

```
for ( ; ; ) System.out.println("Hi");
```

... is an infinite loop...

- <init> can begin with a declaration

```
for (int lastNum = 1; lastNum <= 7; lastNum++) {
```

Back to Arrays...

- The length of an array can be obtained by accessing the length field of the array (!)

```
double[] examScores = { 99.0, 78.5, 85.2, 91.1 };
System.out.println(examScores.length);
for (int i=0; i<examScores.length; i++) {
  System.out.println(examScores[4-i]);
}
```

- Accessing an index outside of the bounds of the array will result in your program crashing with an "ArrayIndexOutOfBoundsException"
  - Does this happen above?
  - What should it be instead of [4-i]?

Comparing Arrays

```
int[] numbers = { 2, 4, 6 };
int[] values = new int[3];
values[0] = 2;
values[1] = 4;
values[2] = 6;

if (numbers == values)
  System.out.println("EQUAL AT POINT A");

numbers = values;
if (numbers == values)
  System.out.println("EQUAL AT POINT B");
```

- Like classes, arrays are reference types

compareArrays Method

```
/* this methods compares the contents of two int arrays for
   equality */
public static boolean compareArrays(int[] arA, int[] arB) {
    if (arA.length != arB.length)
        return false;

    // at this point we know the lengths are the same
    for (int i=0; i<arA.length; i++) {
        if (arA[i] != arB[i])
            return false;
    }

    // we didn't find any discrepancies, so...
    return true;
}
```


Copying Arrays

```
int[] arA = { 2, 4, 6 };
int[] arC = arA;
System.out.println(arA[0]);
arC[0] = 10;
System.out.println(arA[0]);   // !!??

arA[0] = 2;              // restore arA's element
System.out.println(arA[0]);
arC = copyArray(arA);   // try again
arC[0] = 10;
System.out.println(arA[0]);
```


copyArray Method

```
/* this method "clones" an int array */
public static int[] copyArray(int[] ar) {
    int[] copy = new int[ar.length];
    for (int i=0; i<ar.length; i++) {
        copy[i] = ar[i];
    }
    return copy;
}
```


Arrays as Arguments to Methods

- Main method:

```
int[] arA = { 2, 4, 6 };
int a = 5;
System.out.println("a: " + a);
foo(a);
System.out.println("a: " + a);

System.out.println("arA[0]: " + arA[0]);
bar(arA);
System.out.println("arA[0]: " + arA[0]);

arA[0] = 2;
System.out.println("arA[0]: " + arA[0]);
foobar(arA);
System.out.println("arA[0]: " + arA[0]);
```

```
public static void foo(int b) {
        b = 10;
}

public static void bar(int[] ar) {
    ar[0] = 10;
}

public static void foobar(int[] ar) {
    ar = new int[10];
    for (int i=0; i<10; i++) {
        ar[i] = 300;
    }
}
```

(Aside) Methods: "Arguments" and "Parameters"

- Methods are a collection of statements that solve a specific task
  - Have a return type, name, and list of parameters
  - Parameters allow you to pass information to a method to be used in accomplishing the method's task
  - Even if there are no parameters, you have to specify an empty list: double x = Math.random();
- public static void foo(int b) { …
  - In this context, b is called a "parameter" of the method, or a "formal parameter"
- main(…) { … foo(45); … }
  - Here, 45 is called an "argument" being passed to the method, or an "actual parameter"

Command-Line Arguments

- When you run a program from the command line (*e.g.*, "DOS prompt") you type the name of the program and also some arguments
  - javac Arrays.java
    - The program is "javac" (the Java compiler) and you pass as an argument the name of a file to open and compile
  - java Arrays
    - The program is "java" (the Java virtual machine) and you pass as an argument the name of the compiled program to execute
- You can easily extend your Java programs to accept their own command line arguments…

Command-Line Arguments in Java

- A familiar line of Java by now:

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

- What is the red part?
- If you run your program by typing:

```
java Arrays ABC def HIJ 2123 21.93
```

the operating system creates an array of strings storing the command line arguments and passes that array to your main method

Command-Line Arguments Example

```
public class Arrays {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("No command line arguments.");
        } else {
            for (int i=0; i<args.length; i++) {
                System.out.println("args[" + i + "]: "
                        + args[i]);
            }
        }
    }
}
```

- (The name "args" is the conventional name for the array variable containing the command-line arguments…)

Sorting Arrays

```
int[] toSort = { 45, 19, 21, 32, 18, 4, 36, 7, 2, 12 };
sortArray(toSort);
for (int i = 0; i < toSort.length; i++) {
   System.out.print("  " + toSort[i]);
}
System.out.println();

 ...

public static void sortArray(int[] ar) {

   ???

}
```


Selection Sort

```
public static void sortArray(int[] ar) {
   for (int i = 0; i < ar.length; i++) {
       int sel = i;
       for (int j = i+1; j < ar.length; j++) {
           if (ar[j] < ar[sel]) sel = j;
       }
       if (sel != i) {       // minor optimization
           int temp = ar[i];
           ar[i] = ar[sel];  // swap ar[i] and ar[sel]
           ar[sel] = temp;
       }
   }
}
```

Analysis of Algorithms

- Complexity: a measure of the effort expended by the computer in performing a computation, relative to the size of the computation
  - *i.e.*, usually, relative to the size of input
  - Can consider either space or time complexity
  - Examples: constant, linear, logarithmic, quadratic, exponential, factorial, hyperexponential, ...

Composing Data Structures

- Classes whose components are objects
- Classes whose components are arrays
- Arrays whose components are objects
  - Access components using either . (dot – field/method selection) or [] (array index)

StudentRec Example

```
class StudentRec {
   String stuName;
   float gpa;
   int[] examScores = new int[4];
   char courseGrade;

   ...

}

...

StudentRec student = new
   StudentRec();
StudentRec[] members = new
   StudentRec[100];
members[0] = new StudentRec();
```

```
...
student     // a single student
student.stuName  // a name
student.gpa
student.examScores // array
                   // of int
student.examScores[0]
student.examScores[4]
student.courseGrade

members[0]
members[0].stuName
members[0].stuName.length()
members[0].gpa
members[0].examScores
members[0].examScores[1]
```

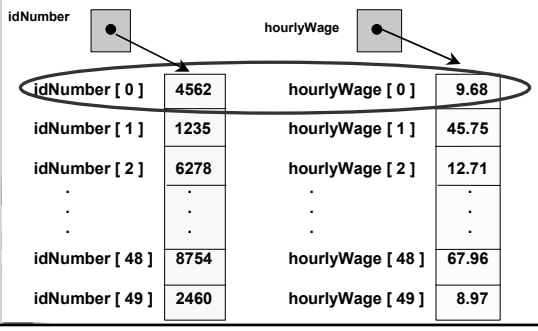

Parallel arrays

- Two or more arrays that have the same index range, and whose elements contain related information, possibly of different data types
- Example: Storing employee ID numbers and hourly wages

```
final int SIZE = 50;
int[] idNumber = new int[SIZE];
float[] hourlyWage = new float[SIZE];
```

- Exercise: Write a method that finds hourly wage given an ID number


```
final int SIZE = 50 ;
int [ ] idNumber = new int [ SIZE ] ;          // parallel arrays hold
float [ ] hourlyWage = new float [ SIZE ] ;    // related information
```



Partial Array Processing

- Can define an array to have more elements than you're actually using
- The length field of an array stores the total number of allocated elements

```
idNumber.length     // 50 in the previous example
```

- You can define another variable to keep track of how many elements are actually being used

```
numEmployees = 35;
  // only idNumber[0]...idNumber[34] contain valid data
```

- Use this counter instead of the array length when processing the array

Array Indexes

- Array indexes can be any integral expression of type `char`, `short`, `byte`, or `int`
- Programmer's responsibility to make sure that an array index does not go out of bounds.
  - The index must be within the range 0 through the array's length minus 1
  - Using an index value outside this range throws an ArrayIndexOutOfBoundsException; prevent this error by using the public instance variable length
- Indices with semantic content
  - Data stored at particular index for some reason, not arbitrarily
  - charCount array in lab
    - charCount['A'] is the number of 'A's encountered

Case Study: Grading True/False Tests (Chapter 10)

- History teacher gives T/F exams
- Asks you to write application to grade them
- Input file:
  - answer key on the first line
  - pairs of lines: student names, answers

```
TFTFTFTFTTTFFFT
Joe Jones
TFTFTFTFTTTFFTT
Janet Jerome
TFTFTFTFTTTFFFF
Jeff Jubilee
TTTFTFTFTFTTTFFFT
```

- Output file: Student names followed by number of correct answers

CRC Cards

- AnswerSheet

Responsibilities	Collaborations
Create itself (# questions)	--
Input name and answers	file (BufferedReader)
Compare with another answer sheet, return # of matches	--
Tell name	

- GradeExams

Data Representation

- AnswerSheet:
  - name : string
  - answers : array of characters ('T' or 'F')
    - could also use array of booleans
- How many questions on the exam?
  - Provide as the first line of the input file
  - To unify view of answer key and student responses, supply "Answer Key" as name of answer key

Testing

- Cases:
  - All answers correct
  - All answers wrong
  - Some correct
    - First and last position okay
    - First and last positions not okay,
    - Okay in the middle
    - Not okay in the middle
- Test cases:
  - Answer key:
    - TTTTT
  - Responses:
    - TTTTT, FFFFF, TFTFT, FTTTF, TTFTT
  - Other answer keys…

Recap: Java Data Types