CS18000: Problem Solving and Object-Oriented Programming

Arrays
Video 1
Introduction to Arrays
Arrays

Concepts
Syntax
Examples
Lots of Temperatures

Three temperatures:
double tempAM, tempNoon, tempPM;

24 temperatures:
double temp1AM, temp2AM, ... tempNoon, temp1PM, ... tempMidnight;

Mathematicians have solved this problem
Subscripted variables
\(temp_1, temp_2, \ldots, temp_{24}\)

Java uses
\[
double[] temp = \text{new double [24]};
\]
which creates temp[0], temp[1], ..., temp[23]
Lots of Temperatures

<table>
<thead>
<tr>
<th>temp</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Our Next Advance

• Single variables holding single values is not enough

• Need...
  – Ways to deal with data in bulk
  – Treat a collection of values as single unit

• Also known as “data structures” and “aggregate data types”

• Arrays are just one example
Data Structure Characteristics

• Contents: types of the values
  – heterogeneous data values, or
  – homogeneous data values

• Size: number of data values it contains
  – static (fixed size)
  – dynamic (can grow or shrink)

• Element access: how efficiently can different elements be inserted, deleted, or changed?
  – Sequential access
  – Random access
Arrays

• An array is a list of values
• A kind of “container object” in Java
• In Java, arrays are
  – homogeneous
  – static
  – random access
  – reference types
• Analogy: a String is like an array of characters
Video 2
Using Arrays
A Familiar Example

```java
public static void main(String[] args) {
    // The parameter to main is an array of Strings
    // Array args initialized from the space-separated “words” on the command line
    String[] args = new String[3];

    args[0]: first argument
    args[1]: second argument
    args[i]: ith argument
    args.length == number of arguments
}
```
A Familiar Example

```plaintext
args

length 3

0 1 2

"add"
"5"
"17.25"
```
An Example of the Familiar

public class Calculate {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.printf(
                "args[%d] = %s
", i, args[i]));
    }
}
Declaring an Array Variable

- Example:
  
  ```java
  String[][] args
  ```

- In general:
  
  ```java
  type_name [ ] variable_name
  ```

- `type_name`: any primitive or reference type
- `variable_name`: a standard Java identifier
Creating an Array Object

• Example:
  new String[10]
• In general:
  new type_name[size]
• type_name: any primitive or reference type
• size: an int-valued expression
• String[] students = new String[10];
  or
• var students = new String[10];
Accessing an Array Element

• Example:
  args[0] or args[i]

• In general:
  \texttt{array\_object\_reference [ int\_value ]}

• \texttt{array\_object\_reference}: an expression (e.g., variable) that references an array object

• \texttt{int\_value}: an expression that yields an int
```java
int[] list = new int[5];
```
Video 3
WordList Array Example
Problem: WordList

• Create a WordList class that reads words from a file into an array
• Create a computeHistogram method in WordList that computes the number of words of each length in the WordList
• Need to
  – Declare array variables for the main word list and to hold the histogram counts
  – Choose suitable maximum lengths and allocate
Step 1: WordList

• Create WordList class
• What information (fields) should be stored in a WordList object?
  – words: an array containing the words
  – size: the number of actual words in the array
• How should the object be initialized?
  – Constructor takes a Scanner object
  – Allocates “big” array
  – Reads words from Scanner and stores in array
Step 2: WordList

• Create a main method for testing
• Create a Scanner object with String of words
• Create a WordList object
• Print
  – Number of words found
  – List of words
Step 3: Create Histogram

• Add method “int[] computeHistogram()”
• Allocate an array of ints to store the histogram
• How big? Pick a number
• Loop through all the words in the list
  – Get the length of the word
  – Update the histogram counter of that length
• Return the histogram array
• In main: call, then print results
import java.util.Scanner;

public class WordList {
    final static int MAXWORDS = 300000;  // Problem we will discuss later!
    final static int MAXHIST = 50;

    private String[] words;
    private int size;

    public WordList(Scanner in) {
        words = new String[MAXWORDS];
        size = 0;

        while (in.hasNext()) {
            words[size++] = in.next();
        }
    }
}
Solution: WordList (2)

```java
public int getSize() {
    return size;
}

public int[] computeHistogram() {
    var histogram = new int[MAXHIST];
    for (int i = 0; i < size; i++)
        histogram[words[i].length()]++;
    return histogram;
}
```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    WordList w = new WordList(scan);
    System.out.printf("read %d words\n", w.getSize());

    int[] wordLengths = w.computeHistogram();
    for (int i = 1; i < MAXHIST; i++)
        if (wordLengths[i] > 0)
            System.out.printf("%2d: %5d\n", i, wordLengths[i]);
}
Solution: WordList

<table>
<thead>
<tr>
<th>length</th>
<th>300000</th>
</tr>
</thead>
<tbody>
<tr>
<td>words</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>&quot;It&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;was&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;dark&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;and&quot;</td>
</tr>
<tr>
<td>null</td>
<td>&quot;stormy&quot;</td>
</tr>
<tr>
<td>69999</td>
<td>&quot;night&quot;</td>
</tr>
<tr>
<td>null</td>
<td>&quot;end&quot;</td>
</tr>
<tr>
<td>299999</td>
<td>null</td>
</tr>
<tr>
<td>size</td>
<td>0</td>
</tr>
</tbody>
</table>
Solution: WordList

- **length**: 50
- **histogram**: e → 0
  - 1 → 1
  - 2 → 2
  - 3 → 3
  - 4 → 4
  - 5 → 5
  - ...
Video 4
TreeTracker Array Example
Array Initialization

• Default: elements initialized with type-specific default
  – Integer types: 0
  – Real types: 0.0
  – Reference types: null
• Compile-time array initialization possible
• Example:
  ```
  char[] vowels = { 'a', 'e', 'i', 'o', 'u' };
  ```
• In general:
  ```
  array_declaration = { v_1, v_2, ... v_n };
  int[] days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  ```
• Note: Initialization must be done in conjunction with (at
time of) declaration
Problem: TreeTracker

• The National Forest Service needs our help...
• Create a TreeTracker class that creates random Tree objects (for testing)
  – Generate random species and circumferences
  – Store them in an array and print out using the describe method
• Illustrates: array initialization, pseudo-random numbers, length field in arrays
public class Tree {
    private int serial;
    private double circumference;
    private String species;

    public Tree(int serial, double circumference, String species) {
        this.serial = serial;
        this.circumference = circumference;
        this.species = species;
    }

    public String describe() {
        return String.format("Tree number %d has a circumference of %.2f and is of species %s.", serial, circumference, species);
    }

    public double getDiameter() {
        return circumference / Math.PI;
    }

    public double getCircumference() {
        return circumference;
    }
}
Random Class

A Random Class object can be used to generate a random number

You must...

import java.util.Random;
...and declare a Random object...
Random r = new Random();

r.nextDouble() returns a value in the range [0.0-1.0) (same as Math.random())
r.nextInt(int n) returns a value in the range [0-n)
r.nextInt() returns any possible integer positive or negative number
import java.util.Random;

public class TreeTracker {
    final static int NUMTREES = 100;

    public static void main(String[] args) {
        Random r = new Random();
        String[] species = { "pine", "elm", "spruce", "oak", "walnut" };

        var trees = new Tree[NUMTREES];

        for (int i = 0; i < trees.length; i++) {
            String specie = species[r.nextInt(species.length)];
            trees[i] = new Tree(i, r.nextDouble()*100, specie);
        }

        for (int i = 0; i < trees.length; i++)
            System.out.println(trees[i].describe());
    }
}
Solution: TreeTracker

```
species 0 1 2 3 4
  "pine" "elm" "spruce" "oak" "walnut"
```
Solution: TreeTracker

```
length: 100

trees: 0

1: null
2: null
...
99: null
```
Video 1
Two-dimensional arrays, for-each loops, varargs
Array Elements Can Be Other Arrays!

```java
int[][] matrix = new int[5][10];
```
• Creates a 2D matrix with 5 rows and 10 columns
  ```java
  {{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }}
  ```
• Elements of matrix[i] are arrays of (10) ints.
A Java 5x10 Matrix

```
matrix[i][j]
```

```
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

matrix.length is 5
matrix[i].length is 10 (for all i in this example)
Uses of 2D Arrays

• Representing
  – A grid-based game, like tic-tac-toe or chess
  – A set of distances between pairs of cities
  – Matrices in linear algebra
  – Other tabular data

• Generalizable to additional dimensions
  ```java
  double[][][] spaceTime = new double[100][100][100][100];
  ```

• Note: Above requires 100,000,000 storage locations
Quirks

• There is no requirement in Java that all rows of a 2-D matrix have the same number of elements ("columns")
• Allows "ragged right" arrays (aka "jagged right")
• Useful for saving storage
Declaring a Ragged Array

```java
int[][] matrix = new int [5][];

matrix[0] = new int [7];
matrix[1] = new int [10];
matrix[2] = new int [4];
matrix[3] = new int [6];
matrix[4] = new int [9];
```
Problem: Sum Matrix

Compute the sum of a 2D array...

```java
int computeSum() {
    int sum = 0;
    for (int i = 0; i < matrix.length; i++)
        for (int j = 0; j < matrix[i].length; j++)
            sum += matrix[i][j];
    return sum;
}
```

Note: This solution works even for ragged arrays.
The for-each Loop

• Designed for use with arrays (and, more importantly, other data structures to come)

• Allows simplified iteration through an array
  ```java
  int[] bloodPressure = new int[500];
  int sum = 0;
  for (i=0; i<bloodPressure.length; i++)
    sum += bloodPressure[i];
  for (int value : bloodPressure)
    sum += value;
  ```

• Limitation: only read-access to each element
varargs

• varargs is short for variable-length arguments
• method that takes a variable number of arguments is a varargs method
• A variable-length argument is specified by three periods(...) 

```java
public static void func (int ... a)
{
    // method body
}
```
• tells the compiler that func() can be called with zero or more arguments
• a is implicitly declared as an array of type int[]
varargs

```java
public static void main (String args[])
{
    func (100);         // sends array {100}
    func (1, 2, 3, 4);  // sends array {1,2,3,4}
    func ();            // sends array {}
}
```

- varargs has been used and these arguments should be stored in the int array referred to by a
- a.length can be used to find the length of the array
- A method can have variable length parameters with other parameters too, but the varargs parameter should be last in the parameter list of the method declaration
  ```java
  int nums (int a, float b, double ... c)
  ```
Video 2
ArrayLists
Generic Classes

• An advanced topic to be covered more fully later
• Basic idea: A generic class is one that can be parameterized with another class
• ArrayList<E> is parameterized with class (or type) E. It can only “hold” elements that are references to objects of class E
• E must be a reference type; primitive types are not supported
ArrayList Class

• A class provided in the java.util package
• Dynamic array: automatically grows to accommodate new items
• Works with any type of object, but you must specify the type when the ArrayList object is created (just like a Java array)

Example

```java
ArrayList<String> list = new ArrayList<String>();
Creates an empty array of String objects
String[] list = new String[10];
```
ArrayList Class

ArrayList<String> list = new ArrayList<String>();

is correct and complete, but we can use Local Variable Type Inference
var list = new ArrayList<String>();

or the second argument can be inferred
ArrayList<String> list = new ArrayList<>();
ArrayList Class

• Unlike Java arrays, ArrayList does not work with primitive types, only reference types
• Can’t say “ArrayList<int> list”
• Fortunately, Java provides special “wrapper” classes for each primitive type
• Can say “ArrayList<Integer> list”
• Java handles the conversion between wrapper class and corresponding primitive type
Useful ArrayList Methods

- `add(e)` – adds e to end of list
- `add(i, e)` – adds e at index i (0-based), pushing others down
- `contains(e)` – returns true if e is in the list
- `get(i)` – returns the value at index i
- `remove(e)` – removes e from the list
- `set(i, e)` – adds e at index i, replacing what was there
- `size()` – returns the current size of the list
Useful ArrayList Methods

```java
ArrayList<String> list = new ArrayList<String>();

list.add("A");
list.add("B");
list.add("C");
list.add(1, "D");
list.get(2);
list.set(1, "E");
```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayListDemo {
    public static void main(String[] args) {
        var list = new ArrayList<String>();

        Scanner in = new Scanner(System.in);

        while (in.hasNextLine()) {
            String s = in.nextLine();
            list.add(s);
        }

        System.out.printf("read %d lines\n", list.size());

        for (int i = 0; i < list.size(); i++) {
            System.out.printf("%s\n", list.get(i));
        }
    }
}
TreeList Class

```java
import java.util.ArrayList;

public class TreeList {
    public static void main(String[] args) {
        var forest = new ArrayList<Tree>();

        while (Math.random() < 0.9) {
            Tree t = new Tree(Math.random() * 100);
            System.out.printf("tree has radius %.3f\n", t.getRadius());
            forest.add(t);
        }

        System.out.printf("created %d trees\n", Tree.getNumberOfTrees());

        System.out.printf("list has %d trees:\n", forest.size());
        for (Tree t : forest)
            System.out.printf("tree with radius %.3f\n", t.getRadius());
    }
}
```
Arrays Class

• Useful utility methods for dealing with arrays

```java
import java.util.Arrays;

• Work with arrays of many types (not just int and double)
```

- ```Arrays.binarySearch(int[] array, int value)```
- ```Arrays.copyOf(double[] array, int length)```
- ```Arrays.copyOfRange(double[] array, int from, int to)```
- ```Arrays.equals(int[] array1, int[] array2)```
- ```Arrays.fill(double[] array, double value)```
- ```Arrays.sort(int[] array)```
- ```Arrays.toString(double[] array)```
Generalizations

• A method can take an array as parameter
  – int computeSum(int[] a)
  – public static void main(String[] args)

• A method can return an array as return value
  – int[] sortIntegers(int[] a)
  – int[] computeHistogram()

• An array of type T[], has elements of type T
Video 3
Playing Cards Array Example
Problem: Model Playing Cards

• Create a class that models a deck of playing cards

• Features needed:
  – Draw cards at random from the deck
  – Shuffle
  – Convert card representation to suit and value string
Representing Cards

• Represent suits as ints 0-3
  – H, S, D, C
  – 0, 1, 2, 3

• Represent card values as ints 0-12
  – A, 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K
  – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

• Represent a card in a single int as
  – suit * 13 + value
  – Each card is a value in the range 0-51 (inclusive)
  – suit is (card / 13); value is (card % 13)
Representing Cards

• six of diamonds
• suit = 2
• value = 5
• suit * 13 + value = 31

• What card is 31?
• suit is (card / 13) = 2 (diamonds)
• value is (card % 13) = 5 (six)
Representing a Deck of Cards

- Use an int array with 52 locations
- Locations are (initially) values 0-51
- That is: deck[i] == i for all i
- Thus, deck is initially
  - Sorted by suits then by value
  - All cards are available to be drawn
- To draw a card
  - Choose card at random from available cards
  - Swap chosen card with last card in deck
  - Reduce available cards by 1
- To shuffle
  - Set available cards to 52
Initial Deck Representation

```
deck

size 52

int[]

0 0 (AH)
1 1 (2H)
2 2 (3H)
3 3 (4H)
4 4 (5H)
5 5 (6H)
  . .
  . .
  . .
  . .
51 51 (KC)
```
r.nextInt(size) -> 5
Swap card 5 (6H) and 51 (KC)
Decrement size by 1
Drawing a Card at Random

r.nextInt(size) -> 5
Swap card 5 (6H) and 51 (KC)
Decrement size by 1
Solution: DeckOfCards