Objects and Arrays

CS 99 – Summer 2000

Michael Clarkson

Lecture 6

Administration

- Read clarified grading policies
- Lab 6 due tomorrow
  - Submit .java files in a folder named Lab6
- Lab 7
  - Posted today
  - Upson Lab closed (today?, tomorrow, Wednesday?) – class cancelled on Tuesday
  - Michael will hold office hours during lab time

Agenda

- Objects
- Arrays

Objects

- What is a class?
- What is an object?
- Object reference variables
- Invoking methods on objects

What is a Class?

- Java represents data as either primitive types (`int`, `double`, `boolean`, `char`) or user-defined types
- A *type* is a set of values and operations that can be performed on those values
- A *class* is a user-defined type in Java
  - Defines methods (i.e., operations) that can be performed on objects of that class

What is an Object?

- An *object* is an *instance* of a class
  - instance: a case or occurrence
- Consider cars:
  - There many 2000 Chevrolet Cavaliers
  - Each was made from a specification for the car
  - The specification is the class
  - Each of the cars is an object of that class

Objects have Methods

- Can invoke methods on objects, e.g.:
  - `System.out.println("Hello, world!");`
  - `println()` is a method of the `System.out` object
  - `System.out` is an instance of the class `PrintStream`
- Printing is an operation performed on the `System.out` object
- Or, printing is a service performed for us by the `System.out` object
- So calling a method is like sending a message to the object, telling it to do something for us

Objects Methods

- `Random` is another class we've used
- We created an instance of it called `generator`
- We used a method of `generator` to get it to generate random numbers for us
- We've also used the `equals` and `charAt` methods of the `String` class

Object Methods [2]

- What about a car? What methods (operations on it, services it performs) can we identify?

```
class Car {

  // ??

}
```

Objects have Fields

- *Fields* are data contained by objects
- For example, the `System.out` object has to keep track of where it is printing on the screen
- Objects of the `Random` class have to know what the last number they generated was in order to generate the next number
- Fields are not usually available to the users of the object

Object Fields

- What fields would a `Car` have?

```
class Car {

  // ??

}
```

Object Reference Variables

- We name objects by declaring *object reference variables*
  - All variables that hold objects, instead of primitive types, are reference variables
  - *Reference* means the variable refers to an object, but is not the object itself – more on that later

Declaring References

- We declare references in an identical manner to declaring primitive variables:

  `type name;`
- Default value is the keyword `null`

e.g.:

```
    Random generator;
    Car myCar;
    String firstName;
```

Creating Objects

- **Not** the same thing as declaring a reference!
- Objects are created (constructed) with the `new` operator
- Examples:

```
  new Random()    // note these
  new Car()       // look like
  new String()    // method calls
```

Creating Objects [2]

- Declaring a primitive reserves memory and gives it a name
- Declaring a reference gives it a name but reserves no memory for an object
- Creating an object reserves memory but gives it no name
- So we have to declare and create objects

Creating Objects [3]

- The usual way to declare and create objects is therefore to declare the reference and initialize it to an object.

```
 Random generator = new Random();
 Car myCar = new Car();
 String firstName = new String();
```

Strings

- Why haven't we ever written something like:

  `String firstName = new String();`
- Java treats String literals as an implicit object creation, so

  `"Hello"`

  is essentially replaced by

  `new String("Hello")`

Invoking Methods

- To invoke a method on an object, we write:

  `objectReference.methodName``(...)`
- Examples:
  - `System.out.println();`
  - `generator.nextBoolean();`
  - `myCar.honkHorn();`

Arrays

- What is an array?
- Array indexing
- Declaring and instantiating arrays
- Array initializers
- Using arrays

Lists of Data

- List of 5 integers (say, test scores)
  - 45, 66, 78, 82, 95
  - How would we represent these in a program?
    - Declared 5 variables with distinct names, say: a, b, c, d, e.
    - Would have input values from the user
  - This produces 5 named locations in memory:

45	66	78	82	95
a	b	c	d	e

Lists of Data [2]

- But what if we had 100, or 1000 scores?
- There's a better way of working with lists of data: arrays

scores
45
66
78
92
95

What is an Array?

- An array is a sized list of values
- Each value in the array is at a specific, numbered location
  - The number is called an *index* or *subscript*
  - Indexing starts at 0 in Java

	[0]	[1]	[2]	[3]	[4]
scores	45	66	78	82	95

Array Elements

- Each location in an array is called an *element*
- Every element can be treated exactly like a regular variable
- We refer to an element with the name of the array and the element's index:

```
scores[0] = 45;
```

Array Indexing

- `[]` is the index operator in Java
- It returns the value at a particular location in an array
- Arrays are indexed from 0 to N-1, where N is the size of an array
- It is a run-time error to index past the end of an array (`ArrayIndexOutOfBounds`)
  - Bounds checking

Declaring Arrays

- In Java, arrays are objects
  - Remember, all non-primitive types are objects
- So we have to declare a reference to the array, and create the array object itself
- An array reference is declared just like a non-array variable of the same type, with the addition of brackets:

```
int a;          int[] a;
String b;       String[] b;
Random c;       Random[] c;
```

Creating Arrays

- Since arrays are objects, they are created with the `new` operator:

```
new int[5]
new String[50]
new Car[1000]
```

- The number in brackets is the *size* of the array
  - Number of elements in it

Initializing Arrays

- Typically the reference declaration and array creation occur in the same statement:

```
int[] scores = new int[50];
String[] names = new String[50];
Car[] carLot = new Car[300];
```

- Also possible to initialize arrays to specified values:

```
int[] scores = {45, 66, 78, 82, 95};
```

- If there is no initializer, what do the elements equal?

Array Sizes

- It is best – if you know how large of an array you need – to use a constant for declaring the size of an array, e.g.:

```
final int CAR_LOT_SIZE = 1000;
…
Car[] carLot = new Car[CAR_LOT_SIZE];
```

Using Array Elements

- Assign the 6th element of the array `scores` the value 100
- Assign the variable `x` the 1st element of `employees`
- Take the square root of element `i` of `prices` and assign it to `p`
- Assign two times element `j` of `a` to element `k` of `b`

Using Array Elements [2]

- `scores[5] = 100;`
- `x=employees[0];`
- `p = Math.sqrt(prices[i]);`
- `b[k] = 2 * a[j];`

Using Loops with Arrays

- Loops for input, calculation, and output over arrays are very common
  - Input a list of values into an array
  - Calculate based on values in array (sum, average, etc.)
  - Output part or whole of array
- What sort of loop would be best to use?

Input Loop

- Use a for loop to input *n* values into an array named `list`:

```
for (int j = 0; j < n; j++) {
    list[j] = getValueFromUser();
}
```

- We use “0” and “< n” to keep the loop index in bounds

Calculation Loop

- Add up all the values in the `list` array

```
int sum = 0;
for (int k = 0; k < list.length; k++) {
    sum += list[k];
}
```

- The `length` field of an array is always equal to the declared size of the array

Output Loop

- Print out every element in `list` that is positive

```
for (int i = 0; i < list.length; i++) {
    if (list[i] > 0) {
          System.out.println(list[i]);
    }
}
```

Arrays as Arguments

- Arrays can be passed as arguments to methods
- To pass an array, just write its name:

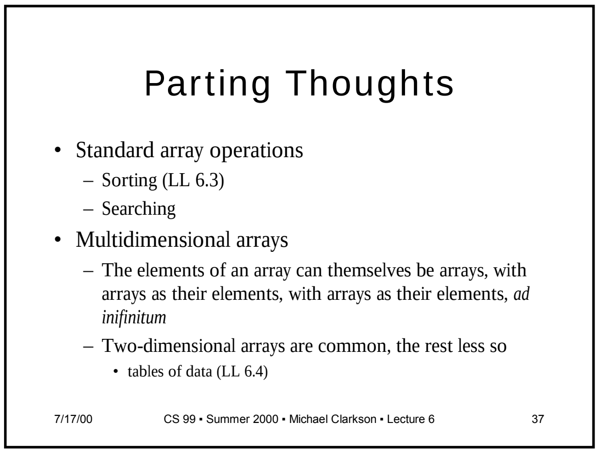
```
double[] scores = new double[NUM_SCORES];
double average = averageArray(scores);
```

- To declare an array as a parameter, include its type in the method header:

```
static double averageArray(double[] arrayToAverage)
```

Array Sizes, Revisited

- What if, when you declare the array, you don’t know how big it needs to be?
- Solutions:
  - Pick a size
    - Must be big enough to hold anticipated data
    - Must not be too big and waste memory
  - Java is nice in that we can input from the user how big to make the array, and then declare it



Parting Thoughts

- Standard array operations
  - Sorting (LL 6.3)
  - Searching
- Multidimensional arrays
  - The elements of an array can themselves be arrays, with arrays as their elements, with arrays as their elements, *ad inifinitum*
  - Two-dimensional arrays are common, the rest less so
    - tables of data (LL 6.4)