Course Topics

Elements of the Java Platform

The Java Language

- Java Arrays, Objects, Methods

Java’s Object Orientation and I/O

Interfaces, Graphical User Interfaces, and Applets

Topics this week:

- Last Week
- Last Week - Control Flow - Branching
- Control Flow - Loops
- Creating Multiple Student Objects
- Java Objects
- References to and Creating Objects
- Creating collections of objects
- What is an "array"?
- Arrays
- Creating an array
- Arrays Can Be Made of Any Type or Class
- Array Manipulation
- Saving Multiple Student Objects
- Objects - Instances of classes
- Java Methods
- Introduction to Inheritance
- Inheritance Example
- Assignment for next time
• Minimal Employee class
Java Arrays, Objects, Methods

Last Week

The Java Language:
The syntax and constructs for writing Java code

The Java Platform

<table>
<thead>
<tr>
<th>Java Program (application, applet, servlet) bytecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Application Programming Interface (standard packages; also bytecode)</td>
</tr>
<tr>
<td>Java Virtual Machine (executes bytecode)</td>
</tr>
<tr>
<td>Hardware, e.g., your PC, OSF1, workstations, rings, ...</td>
</tr>
</tbody>
</table>

Primitive Data Types

- byte
- short
- int
- long
- double
- float
- char

String is a built-in Object type, not a Primitive

User Defined Objects
Creating and initializing instances of classes
Java Arrays, Objects, Methods

Last Week - Control Flow - Branching

"if" statements

```java
if (expression that evaluates to a boolean result) {
    // Perform the enclosed statements if true
    ...
}
else {
    // Perform the enclosed statements if not true
    ...
}
```

"switch" statements

```java
switch (expression that results in an integer value) {
    case 42:  // or whatever a possible value of expression is
        ...
        break;
    case 39:  // or whatever another possible value is
        ...
        break;
    ...
    default:  // (optional) catch all other values
        ...
        break;
}
```
Control Flow - Loops

"for" loops
Perform group of statements for a number of times

    for (initialization; expression that evaluates to boolean; increment) {
        // Code that executes each time through the loop ...
    }

"while" loops
Perform group of statements while a condition is satisfied

    while (expression that evaluates to boolean) {
        // Code that executes each time through the loop ...
    }

"do ... while" loops
Perform group of statements while a condition is satisfied

    do {
        // Code that executes each time through the loop ...
    } while (expression that evaluates to boolean);

Question: How is while different from do ... while
Creating Multiple Student Objects
/** Encapsulate information relating to a single student. ** @author: Jonathan Doughty ***/

public class StudentGroup {

    public static void main(String[] args) {
        if ((args.length == 0) || (args.length % 2) != 0) {
            System.out.println("usage java StudentGroup [student names]"hs);
        } else {
            StudentGroup.makeSomeStudents(args);
        }
    }

    public static void makeSomeStudents(String[] arguments) {

        CSStudent aStudent;
        int nextId = 0;

        for (int arg = 0; arg < arguments.length; arg++) {

            // Each time through the loop a new CSStudent object is
            // created and its reference is assigned to the aStudent
            // field.

            String name = arguments[arg];
            String id = String.valueOf(nextId);
            nextId++;

            aStudent = new CSStudent( name, id);
            aStudent.setGrade(100);

            // Ask each Student to identify itself
            System.out.println(aStudent);
        }

        // Question: Do any of the CSStudent objects created still
        // exist?
    }
}
Java Objects

Classes

**Definition:** A class is a blueprint or prototype that defines the variables and methods common to all objects of a certain kind.


Objects - Instances of classes

**Definition:** An object is a software bundle of variables (fields) and related methods.


Objects instantiate classes

Objects are created (via `new`) from the template that a class defines.

**Question:** Why define classes and create object types?
References to and Creating Objects

Primitives come into existence by declaring them

```
int    count;
float  distance;
boolean guilty;
char   letter;
```

Declaring an Object type establishes space for an Object reference
But does not, by itself, create the object itself.

**Objects must be declared and then instantiated**

Creating (Instantiating) and Using Objects

```
Insect bug;   // declare name for Insect reference
bug = new Insect();   // create and save the instance
```
Creating collections of objects
Several ways to create a group of objects, all of the same type:

**Individual instance variables:**

```java
Student aStudent;
Student anotherStudent;
Student yetAnotherStudent;
...
aStudent = new Student();
aStudent.setName("Fred Flintstone");
anotherStudent = new Student();
anotherStudent.setName("Wilma Flintstone");
yetAnotherStudent = new Student();
yetAnotherStudent.setName("Barney Rubble");
```

**Arrays**
To be discussed in a few minutes

**One of Java’s "Collections" classes**

```java
java.util.Vector;
java.util.Hashtable;
java.util.LinkedList;
java.util.HashMap;
java.util.ArrayList;
```

*We won’t be covering the Collection classes in this course.*
What is an "array"?

A graphic representation

A variable (field, reference)

```
int answer;
CSStudent aStudent;
```

An array

```
int[] answers = new int[4];
CSStudent[] someStudents = new CSStudent[4];
```
Java Arrays, Objects, Methods

Arrays

Arrays themselves are objects in Java
Even arrays of primitive data types.

```java
int intArray[];

intArray = new int[4];

float[] fnumbers = new float[8];

CSStudent studentArray[] = new CSStudent[10];
```

Note the last declaration and instantiation of an array of CSStudent objects
Note that array declarations use [], not ()

**Question:** How many CSStudent objects are created by the declaration?

Since arrays are objects they inherit all the characteristics of java.lang.Object
All array objects also have some other characteristics; i.e., each array has an associated field named length.

Notice it is a field named length, unlike the instance method named length() associated with String objects.
Creating an array

... is like creating an object from a class:

- declaration - assigns a name to the reference
- instantiation - creates space for the object
- initialization - gives the objects values for the first time

Arrays of primitive data types are initialized to 0

```java
int[] grades;

grades = new int[60];
```

Arrays of Objects are initialized to `null`

```java
Student[] students;

students = new Student[60];
```

The `students` array has been declared and instantiated, but not yet initialized: no Student object references have been assigned to the array elements.

To initialize the array elements, you need to `instantiate` each individually:

```java
for (int nextStudent = 0; nextStudent < 10; nextStudent++) {
    students[nextStudent] = new CSStudent();
}
```
Java Arrays, Objects, Methods

Arrays Can Be Made of Any Type or Class

"Declaring a variable of array type does not create an array object or allocate any space for array components. It creates only the variable itself, which can contain a reference to an array."


- Arrays are created (instantiated) with **new**, just like other objects.
- Once an array is created, its length never changes.

**Question:** Any idea why?

Examples

```java
int[] intArray = new int[4]; // elements initially set to 0

CreditCard cards[] = new CreditCard[MAXCARDS];
    // elements initially set to null
    // notice the [] can be placed with the field name
    // or the type; though the latter is "better"
```

*Tip: If you need a collection’s size to change consider using java.util.Vector or another collection class instead of arrays*
Java Arrays, Objects, Methods

ArrayExample.java
/** An Example of some array manipulation */

public class ArrayExample {

    public static void main (String args[]) {
        int numberOfElements = 0;
        if (args.length > 0) {
            // Use value from command line
            numberOfElements = Integer.parseInt(args[0]);
        }
        ArrayExample anExample = new ArrayExample();
        anExample.initializeArray(numberOfElements);
        // Notice that method calls can be included in other method
        // calls: in this case, returning primitive values that will
        // be converted to String objects.
        System.out.println("sum = " + anExample.Sum() +
                            " average = " + anExample.Average());
    }

    private int[] intArray; // all instance (non static) methods
    // have access to this 'instance' variable

    /** Initialize the array (which will be made big enough to hold
    size entries) contents with some numbers */
    public void initializeArray(int size) {
        intArray = new int[size];
        int startValue = size * 3; // pick any number
        for (int i = 0; i < intArray.length; i++) {
            intArray[i] = startValue; // put current number in next slot
            startValue = startValue - 2; // and calculate next number
        }
    }

    /** Calculate the sum of the array contents */
    public long Sum() {
        int index;
        int arrayLen;
        long sum = 0;
        // All arrays have a length field that specifies how many
        // elements the array contains
        arrayLen = intArray.length;
        // Calculate the sum the values in all array elements
        for (index = 0; index < arrayLen; index++) {
            sum += intArray[index];
        }
        return sum;
    }

    /** Calculate the average of the array contents */
    public double Average() {
        // Notice that Average calls Sum() to total the values in the
        // array, rather than duplicating that calculation here. What
        // is going on with the "(double)" ?
        double average = (double) Sum() / intArray.length;
        return average;
    }
}

Array Manipulation

In class example of some array manipulation

Write a Java class named Arrays.java. This class should have the following methods.

1. public void listArgs( String [] args)
   To list out the arguments in an array of Strings

2. public long product( int [] intArray )
   To compute the product of the integers in the array intArray

3. public static void main( String[] args )
   Should have the following code:

   ```java
   if (args.length == 0) {
       System.out.println("usage: Arrays [integers]");
   } else {
       Arrays inst = new Arrays();
       inst.listArgs(args);

       int input[] = new int[args.length];
       for (int i = 0; i < args.length; i++) {
           input[i] = Integer.parseInt(args[i]);
       }
       System.out.print("product = ");
       long answer = inst.product(input);
       System.out.println(answer);
   }
   ```
Saving Multiple Student Objects
Also demonstrates one class using instances of another
/** A class that will make use of another class. 
**/

public class CSClass {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("usage java CSClass [student names]");
        } else {
            int numberOfStudents = args.length;
            CSClass cs161 = new CSClass(numberOfStudents);
            cs161.enroll(args);
            cs161.assignLabPartners();
            cs161.listRoster();
        }
    }

    // Instance fields
    CSStudent students[] = null;
    int last = 0;

    public CSClass(int number) {
        students = new CSStudent[number];
    }

    public void enroll(String[] names) {
        int numberOfStudents = names.length;
        int id = 0;
        for (int arg = 0; arg < numberOfStudents; arg++) {
            String name = names[arg];
            id++;
            String idString = String.valueOf(id); // as a String
            CSStudent aStudent = new CSStudent(name, idString);
            aStudent.setGrade(100);
            // save the reference to the current student in the next array slot
            students[last] = aStudent;
            last++;
        }
    }

    public void assignLabPartners() {
        // Assign every other pair of students as lab partners
        int next = 0;
        int pairs = last / 2;
        while (pairs > 0) {
            students[next].setLabPartner(students[next+1]);
            students[next+1].setLabPartner(students[next]);
            next += 2;
            pairs--;
        }

        // If there were an odd number of students in the array the
        // last one won’t have a lab partner.
    }

    public void listRoster() {
        for (int i = last - 1; i >= 0; i--) {
            System.out.println(students[i]);
        }
    }
}

And the complete CSStudent.java that’s been used in preceding examples
/** Encapsulate information relating to a single student. 
** @author: Jonathan Doughty 
**/

public class CSStudent {

    // Instance variables (fields) that will be associated with 
    // each student
    private String GMU_Id;
    private String name;
    private int homeworkGrade;
    private CSStudent labPartner;

    // Constructors for the class
    private CSStudent() { // why do you think this is?
    }
    public CSStudent(String name, String id) {
        this.name = name;
        GMU_Id = id;
    }

    // An accessor method to set the Student’s name field; not 
    // needed any more but left in because a student’s name could 
    // change.
    public void setName(String studentName) {
        name = studentName;
    }

    // An accessor method to return this Student’s name
    public String getName() {
        return name;
    }

    // An accessor method to set the Student’s GMU_Id field (probably 
    // no longer necessary)
    public void setId(String id) {
        GMU_Id = id;
    }

    // An accessor method to set the Student’s homeworkGrade field
    public void setGrade(int grade) {
        homeworkGrade = grade;
    }

    // An accessor method to assign this Student’s lab partner
    public void setLabPartner(CSStudent s) {
        labPartner = s;
    }

    // Using the toString method to enable an instance of an 
    // object to identify itself usefully.
    public String toString() {
        // Since I’m going to be returning a String made up of various 
        // pieces, I build up those pieces in a StringBuffer.
        StringBuffer sb = new StringBuffer();
        sb.append(name);
        sb.append(" Id# ");
        sb.append(GMU_Id);
        if (labPartner != null) {
            // Notice I don’t just use labPartner.toString(). If I did 
            // I would create an infinite loop: each lab partner calls 
            // its lab partner’s toString() which calls’s its lab 
            // partner’s toString() ... Instead, I just get and append 
            // the lab partner’s name.
            sb.append(" lab partner=");
            sb.append(labPartner.getName());
        }
        return sb.toString();
    }
}

21
Objects - Instances of classes

**Definition:** An object is a software bundle of fields and related methods.

- Notice how each Student object in the previous examples had a name, an id value, a grade, and in the last example, an associated lab partner (another Student.)

- Each Student also had some methods that could be "called on" an instance of the Student class: e.g., setGrade(), setLabPartner()
Java Methods
There are two categories of methods you can write in Java:

Class Methods

- Methods that are associated with a class.
- They are typically "convenience" or utility methods.
- Qualifying a method with the keyword `static` is what makes a method a class method.

In the earlier Student examples,

```java
public static void makeSomeStudents( ... )
```

were all Class methods.

Instance Methods

- Methods that are associated with the instances of a class, objects of the class data type.
- They are used to ask an object to do something, e.g., assign itself a value, perform some operation, return some internal information, identify itself, etc.

In the Student and CSClass examples, methods such as
public void setGrade( ... )
public void getName( ... )
public void listRoster()
public String toString( ... )

were all *instance* methods.
Introduction to Inheritance
You say one class **extends** another when you want to *re-use* most of the capabilities of a class, but want to add some additional capabilities, over-ride some, or provide other specialization.
/** encapsulate the characteristics of a Grad student  
** @author: Jonathan Doughty  
***/

`public class CSGradStudent extends CSStudent {

    private static final int MAX = 100;
    private CSStudent[] taFor;
    private int nextStudent = 0;

    // A constructor that will create student objects with specified name and id
    public CSGradStudent(String studentName, String id) {
        super(studentName, id);
        taFor = new CSStudent[MAX];
    }

    // Other methods here unique to CSGradStudents

    public void writeThesis() {
        // mumble
    }

    public String toString() {
        StringBuffer result = new StringBuffer("CSGradStudent

        result.append(super.toString());
        if (nextStudent > 0) {
            result.append("TA for:

            for (int i = 0; i < nextStudent; i++) {
                result.append(taFor[i].getName());
                result.append("\n");
            }
        }

        return result.toString();
    }
}
`
Assignment for next time

Reading

- Chapter 6 - Inheritance - through the section "Constructors in Derived Classes" that ends on page 301 (you may skip the remainder of Chapter 6)
- Chapter 9 - Streams and File I/O - Up to but not including the Programming Example that starts at the bottom of page 495.

Homework

Goal:

- Write a second Java class and use the two classes written so far, this week’s uses last’s.

Purpose:

- Learn about instances of classes and storing them in an array. on them.
- To start building solutions from smaller pieces: Java classes; to get those pieces to interact.

For this assignment you will write a Java class named Boss. This class will make use of the Employee class created last week. If you have not completed that assignment yet, you may use the minimal version on the next page to do this week’s assignment.

- Your Boss class should have the following instance variables:
  - a bossName
○ an array named employees in which to store Employee objects

○ an int field named count to store the number of Employees who the Boss has.

• Write methods with the following signatures
  ○ a main method to start things off with the signature:

    public static void main (String [] args)

    see below for what main should do.

○ a public constructor for the class that will insure that Boss objects always have a name field with the signature.

    public Boss( String name )

    This constructor should also instantiate an array to store Employee objects (at least 60) and initialize the count of employee’s to 0. It should not initialize the array contents; just create the array.

○ a method to add an Employee object to the array of Employees with the signature

    public void hire( Employee emp )

    This method should increase the count of Employees associated with the Boss as each Employee is added.

○ a method that will cause the Boss to hire a group of people. This method should have the signature

    public void staffUp()

    This method should create a number of Employee objects (at least 5) and then use the hire method to add them to the boss. You may use any names and id values you choose. One way to initialize a group of names to use would be to get them from an array that is initialized like:
String [] names = {"John", "Paul", "George", "Ringo", "Fred" };  

You may use these names or any of your choosing and any technique of your choosing to give each Employee a different name as an input parameter for your Employee constructor. Use any id values you choose.

- a method to print out the Boss name followed by the names of the Employees who are managed by a Boss in an order different than they were added. This method should have the signature

  public void list()

- The main method for this class should:
  1. create a single instance of the Boss class,
  2. call the staffUp method on the Boss object
  3. call the list method on the Boss object

Do only what is asked above. Do not add any extra code to prompt for input or otherwise add capabilities not asked for. Be sure to create methods with the signatures requested. Output similar to the following is all the Boss class needs to produce:

  Boss Bruce:
  Fred
  Ringo
  George
  Paul
  John

Hint: To start, create the outline of your class definition by defining empty methods with the signatures listed above. Compile it to be sure the outline is in the right form. Then go back and start filling in the details of each method. Compile often so you can correct errors easily.
Minimal Employee class
The following is a minimal Employee class that you may use to do this week’s assignment to write a Boss class. This does not fulfill all the requirements of last week’s assignment

/** A minimal Employee class to support doing the homework that ** requires you to write a Department class. ***/

public class Employee1 {

    private String empName;
    private String empId;

    public Employee1(String name, String id) {
        empName = name;
        empId = id;
    }

    public String getName() {
        return empName;
    }
}
