STUDENT OUTLINE

Lesson 26 – Arrays of Objects

INTRODUCTION: In previous lessons, the arrays sorted primitive types, such as integers and characters. Arrays can also have references to objects as elements. Fairly common complex information management structures can be created using only arrays and other objects. For example, an array could contain objects, and each of those objects could contain several variables and methods that use them. Those variables could themselves be array, and so on. The design of a program should capitalize on the ability to combine these constructs to create the most appropriate representation for all information.

The key topics for this lesson are:

A. Object-Array Elements
B. Comparing Objects
C. Using An Array of Objects

VOCABULARY: HETEROGENEOUS AGGREGATE

DISCUSSION: A. Object–Array Elements

1. An object is a user-defined data type that organizes collections of values having the same or different data types. These objects are characterized as a *heterogeneous aggregate.*

   heterogeneous - of different types
   aggregate - a collection of pieces into a mass or sum (like `struct` in C)

2. This model will be used to model objects that require multiple values, usually of different data types. An example is the information printed on a driver's license. Most of the data types are strings, but weight is stored as an integer, and your birthday could be stored using a date class.

3. Programmers who are familiar with arrays but unfamiliar with object oriented programming sometimes distribute information across separate arrays. For example, a program to manage information about students might be organized using separate arrays for each data type.

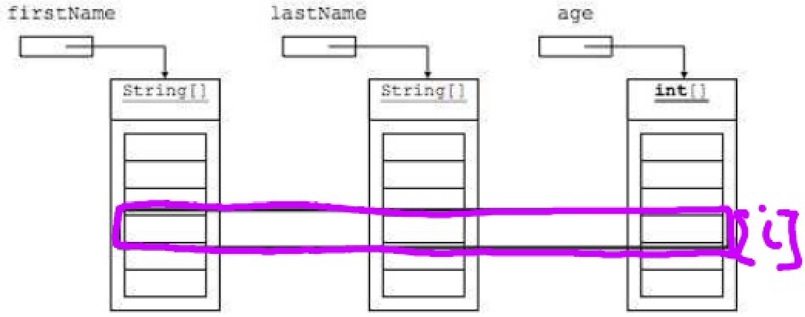
```
// don't do this
String[] lastName;
String[] firstName;
int[] age;
```

PARALLEL ARRAYS

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

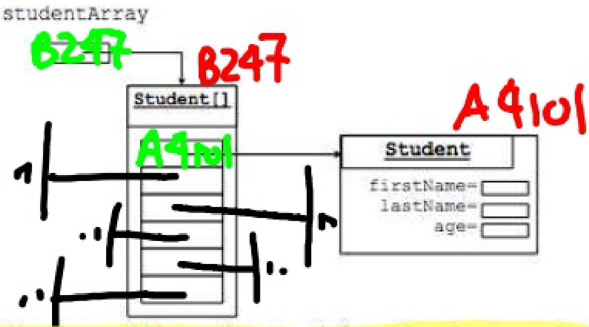
In such a program, related information is distributed in separate arrays. The ith slice (firstName[i], lastName[i], and age[i]) contains data that needs to be processed together. These arrays are called *parallel arrays* and should be avoided.

ith slice
(ith Person)
PRE OOP



4. In situations where you use multiple arrays of the same length, you should consider replacing them with a single array of a class type. Look at a slice and find the concept that it represents and then make the concept into a class. In our example, each slice contains a firstName, lastName and age describing a student.

THE OOP
APPROACH



5. Here is a class that could be used to store information about a student in an array:

INSTANCE
VARIABLES
(ATTRIBUTES OR
NOUNS)

CONSTRUCTOR
(INITIALIZE)

```
class Student
{
  String lastName;
  String firstName;
  int age;

  Student (String lName, String fName, int initAge)
  {
    lastName = lName;
    firstName = fName;
    age = initAge;
  }
}
```

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

6. Instantiating (creating) an array of objects reserves room to store references only. The objects that are stored in each element must be instantiated separately.

B. Comparing Objects

1. In previous lessons, we learned how to sort arrays of integers. In real programming, there is rarely a need to sort or search through a collection of integers. However, it is easy to modify these techniques to search or sort real data.

NOT ALIASES (only with Strings)

2. Suppose we create two `Student` objects and compare them as shown in the following segment of code:

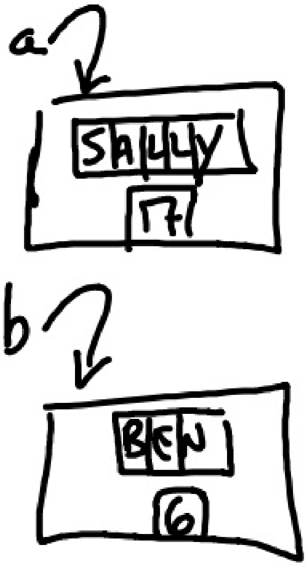
```
Student student1 = new Student ("Thawk", "Nigh", 17);
Student student2 = new Student ("Thawk", "Nigh", 17);

if (student1 == student2)
{
  System.out.println("This will NOT print");
}
else
{
  System.out.println("This WILL print");
}
```

CHECKS THE LOCATION (ADDRESS)

LOCATION! NOT CONTENT.

The comparison will fail since the since the object references, `student1` and `student2` are different. When you compare two variables that are *references* to objects, it is the *references* that are compared, not the objects that they refer to. The equality (==) and relational operators (< and >) compare the values of variables. Hence when variables are references to objects, it is the references that are compared because the references are the values of the variables.



3. Comparing objects requires the implementation of a method that defines the ordering relationship of an object. Java provides a specification for comparison through the implementation of the `Comparable` interface, which specifies one method, `compareTo`:

```
    public int compareTo(Object other)
```

MATCH THIS EXACTLY

The call

```
  a.compareTo(b)
```

must return a negative number if `a` should come before `b`, 0 if `a` and `b` are the same, and a positive number otherwise. To remember the order of comparison, think of `a.compareTo(b)` as "`a-b`."

WHAT WE GET, DEPENDS ON THE CRITERIA!!

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

4. In Java, an **interface** gives a formal specification for objects by listing all the required methods, but does not provide the actual code (interfaces will be covered in more detail in later lessons). A class that **implements** an **interface** must supply all the methods specified by that interface. Many Java library methods also need to compare objects and expect or assume that they deal with comparable objects. Java library classes such as `String`, `Integer`, `Double`, and `Character`, for which "natural" ordering makes sense, implements `Comparable`.

5. You can supply a `compareTo` method for your own class, thus defining a relation of order, a way to compare your class's objects. For example (supposing we were to sort a list of students by age):

SALLY

```
public class Student implements Comparable
{
  ...
  public int compareTo(Object other)
  {
    Student right = (Student)other;
    return myAge - right.~~age~~;
  }                       myAge
  ...
}
```

BEN

Your class's objects become `Comparable` and can be stored in collections and passed to methods that expect the `Comparable` data type.

6. The `compareTo` argument's type is specified as `Object`, to make the method's prototype independent of a particular class that implements it, but usually it is an object of the same class. `compareTo`'s code casts its argument back into its class's type. For example:

```
Student right = (Student)other; // Object other cast to student
```

OF CONTENT

7. Recall that the `Object` class provides a **boolean** method `equals` that compares this object to another object for equality. Therefore, each object has the `equals` method. When a `compareTo` method is defined for a class, it is APPROPRIATE ~~a goodidea~~ to also define an `equals` method that agrees with `compareTo`. For example:

```
  public boolean equals(Object other)
  {
    return compareTo(other) == 0;
  }
```

USE WHAT YOU'VE ALREADY MADE!!

DO BOTH! (HABIT)

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

See Handout H.A.26.1, *ages.java.*

A FANTASTIC TEMPLATE FOR LAB 26.1

C. Using An Array of Objects

1. The handout H.A.26.1, *ages.java*, contains the source code of an example program using an array of objects. The data definitions from the classes are repeated here:

```
public class Student implements Comparable
{
  private String myLastName;
  private String myFirstName;
  private int myAge;

  // additional constructors and methods
}

public class Ages
{
  private Student[] myStudent;

  // additional constructors and methods
}
```

2. The `Student` class contains 3 values: 2 strings and an integer. The `Ages` class contains an array of `Student` to maintain multiple instance of the class.

3. This program keeps track of the first and last names of people and their ages. When an `Ages` object is constructed a text file called *names.txt* is loaded. For the sake of discussion assume *names.txt* has this content:

names.txt

```
4
Smith  Tom  25
Jones  Susan  58
Lee  Cindy  33
White  John  18
```

4. The constructor will initialize data with information from disk.

   a. The constructor receives the pathway to the text file on disk as a string. A `FileInput` object, `inFile` is constructed to access the names data in the file.

   b. The first value read from the file represents the number of `Student` records contained in the file.

   ```
   numStudents = inFile.readInt();
   ```

   reads the 4 from the first line in *names.txt* and stores it in `numStudents`.

   ```
   myStudents = new Student[numStudents];
   ```

   constructs a `Student` array to hold 4 `Student` objects.

   c. The **`for`** loop will traverse the list from positions 0..3, reading appropriate data from *names.txt*.

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

5. Adding new people to the list is accomplished by constructing a new `Student` object for each data record

```
myStudents[studentRec] = new Student(lastName, firstName, age);
```

Instantiating an array of objects reserves room to store references only. The objects that are stored in each element must be instantiated separately.

6. The `quickSort` and `swap` methods were modified to work with a new data type called `Student`. The list of people in the data structure will be ordered in ascending order based on the last name.

7. Finally the structure is saved again to disk including any additional data added to the structure. Saving is the exact opposite of loading the text file from disk.

SUMMARY/ REVIEW: The strategy of using arrays of objects is a powerful organization tool. Combinations of arrays and objects provide excellent tools for organizing and manipulating data.

ASSIGNMENT: Lab Exercise, L.A.26.1, *Store*

OUTPUT TO THE SCREEN FOR SCORING

© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)