Arrays
NULL References
Printing Objects of your own Classes

Lecture 24

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner
News

• Midterm 2, today 6:30-7:30
  – Note: the rooms are different form 1st midterm
    • List is now online
    • The list was corrected yesterday
      – Woodward 2 was incorrectly listed as Woodward 5

  – Material: conditionals, loops
    • Arrays will NOT be on the midterm
Reading Assignments

• Reading for this week: sorting & searching
  – Edition 2: Ch 19.1, 19.3
  – Can wait until after the midterm...
Recap: Multidimensional Arrays

- Arrays can be built of any data type
  - including arrays of arrays!

```java
int[][] x = new int[3][4];
```
Recap: Multidimensional Arrays

• What is the value of this expression?

\[ x_{[0][0]} \times (x_{[2][1]} + x_{[1][3]}) \]
Recap: 2D Arrays

• People think of “arrays of arrays” as 2D arrays

```java
int[][] x = new int[3][4];
```

![2D Array Diagram]
Recap: Arrays of Objects

- Usually, it is better to create an array of objects that concentrate the data for one student into one object:
Objectives for Today

• Practice arrays with another example
• Understand NULL pointers/references
• See how to make sure you can output your own classes with System.out.println()
My Empire Strikes Back
How do I keep track of my sales?

Cans of pop sold this month

185 92 370 485 209 128 84 151 32 563
How do I keep track of my sales?

Cans of pop sold this month

185
92
370
485
209
128
84
151
32
563

In other words, how can I organize the data above in my computer so that I can access it easily and do the computations I need to do?
How to store different data types?

<table>
<thead>
<tr>
<th>cansSold</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
How to store different data types?

<table>
<thead>
<tr>
<th>cansSold</th>
<th>cashIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  185</td>
<td>0  201.25</td>
</tr>
<tr>
<td>1   92</td>
<td>1  100.50</td>
</tr>
<tr>
<td>2  370</td>
<td>2  412.75</td>
</tr>
<tr>
<td>3  485</td>
<td>3  555.25</td>
</tr>
<tr>
<td>4  209</td>
<td>4  195.00</td>
</tr>
<tr>
<td>5  128</td>
<td>5  160.00</td>
</tr>
<tr>
<td>6   84</td>
<td>6  105.00</td>
</tr>
<tr>
<td>7  151</td>
<td>7  188.75</td>
</tr>
<tr>
<td>8   32</td>
<td>8   40.00</td>
</tr>
<tr>
<td>9  563</td>
<td>9  703.75</td>
</tr>
</tbody>
</table>
### Arrays of objects

<table>
<thead>
<tr>
<th>cansSold</th>
<th>cashIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 185</td>
<td>0 201.25</td>
</tr>
<tr>
<td>1 92</td>
<td>1 100.50</td>
</tr>
<tr>
<td>2 370</td>
<td>2 412.75</td>
</tr>
<tr>
<td>3 485</td>
<td>3 555.25</td>
</tr>
<tr>
<td>4 209</td>
<td>4 195.00</td>
</tr>
<tr>
<td>5 128</td>
<td>5 160.00</td>
</tr>
<tr>
<td>6 84</td>
<td>6 105.00</td>
</tr>
<tr>
<td>7 151</td>
<td>7 188.75</td>
</tr>
<tr>
<td>8 32</td>
<td>8 40.00</td>
</tr>
<tr>
<td>9 563</td>
<td>9 703.75</td>
</tr>
</tbody>
</table>

Now we can put references to Strings in our String array:

```java
location[0] = "Chan Centre";
location[1] = "Law School";
location[2] = "Main Library";
...and so on...
```
Arrays of objects

<table>
<thead>
<tr>
<th>cansSold</th>
<th>cashIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 185</td>
<td>0 201.25</td>
</tr>
<tr>
<td>1 92</td>
<td>1 100.50</td>
</tr>
<tr>
<td>2 370</td>
<td>2 412.75</td>
</tr>
<tr>
<td>3 485</td>
<td>3 555.25</td>
</tr>
<tr>
<td>4 209</td>
<td>4 195.00</td>
</tr>
<tr>
<td>5 128</td>
<td>5 160.00</td>
</tr>
<tr>
<td>6 84</td>
<td>6 105.00</td>
</tr>
<tr>
<td>7 151</td>
<td>7 188.75</td>
</tr>
<tr>
<td>8 32</td>
<td>8 40.00</td>
</tr>
<tr>
<td>9 563</td>
<td>9 703.75</td>
</tr>
</tbody>
</table>

Or we could have done this:

```java
String[] location = 
{ "Chan Centre", "Law School", "Main Library", .... };
```
Arrays of objects

Each individual String object in the array of course has all the String methods available. For example

location[2].length() would return what?
Each individual String object in the array of course has all the String methods available. For example

```java
location[2].length()
```

would return 12
<table>
<thead>
<tr>
<th>location</th>
<th>cashIn</th>
<th>cansSold</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 185</td>
<td>0 201.25</td>
<td></td>
</tr>
<tr>
<td>1 92</td>
<td>1 100.50</td>
<td></td>
</tr>
<tr>
<td>2 370</td>
<td>2 412.75</td>
<td></td>
</tr>
<tr>
<td>3 485</td>
<td>3 555.25</td>
<td></td>
</tr>
<tr>
<td>4 209</td>
<td>4 195.00</td>
<td></td>
</tr>
<tr>
<td>5 128</td>
<td>5 160.00</td>
<td></td>
</tr>
<tr>
<td>6 84</td>
<td>6 105.00</td>
<td></td>
</tr>
<tr>
<td>7 151</td>
<td>7 188.75</td>
<td></td>
</tr>
<tr>
<td>8 32</td>
<td>8 40.00</td>
<td></td>
</tr>
<tr>
<td>9 563</td>
<td>9 703.75</td>
<td></td>
</tr>
</tbody>
</table>

Note also that because an array of any kind is itself an object, a more accurate picture would look like this.
Are Arrays Objects?

Sort of – but they are weird

Here are some ways in which arrays are like objects:

• Array types are reference types, just like object types.
• Arrays are allocated with the "new" operator, similar to constructors.

Here are some ways in which arrays are not like objects:

• You can't extend an array type to create a new child array type. (You don't know what this means yet, but when you do, it will make sense.)
• Arrays have a different syntax from other object classes.
  e.g.: a.length rather than s.length() for String class
• You can't define your own methods for arrays.
Isn't there an easier way?

<table>
<thead>
<tr>
<th>cansSold</th>
<th>cashIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

0 185 0 201.25
1 92 1 100.50
2 370 2 412.75
3 485 3 555.25
4 209 4 195.00
5 128 5 160.00
6 84 6 105.00
7 151 7 188.75
8 32 8 40.00
9 563 9 703.75

location

0 0
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9

"Chan Centre"
"Law School"
"Main Library"
"Koerner Library"
"Business"
"Biology"
"Education"
"Applied Science"
"Agriculture"
"Computer Science"

Well, of course there is. Since we can create arrays of objects, and we can create objects of our own design, we can just create an array of Coke machines...
Isn't there an easier way?

Let’s have a look at a class called Cokematic, which represents a pop machine that keeps track of the inventory (both cans and money)
Aside:
Printing Objects of your own Classes

Often it would be nice if you could use System.out.println() and related methods to print contents of objects that belong to classes you have created

• This is possible by implementing a method called “toString()” in your class
  – No parameters, returns String
Isn't there an easier way?

Now, executing something like this:

Cokematic machine1 = new Cokematic(100, "Chan Centre", 185, 201.25);
Isn't there an easier way?

Now, executing something like this:

```java
Cokematic machine1 = new Cokematic(100, "Chan Centre", 185, 201.25);
```

results in this:
Isn't there an easier way?

```java
public class CokematicTester {
    public static void main (String[] args) {
        Cokematic[] mymachines = new Cokematic[10];
        mymachines[0] = new Cokematic(100, "Chan Centre", 185, 201.25);
        mymachines[1] = new Cokematic(150, "Law School", 92, 100.50);
        mymachines[2] = new Cokematic(200, "Main Library", 370, 412.75);

        for(int i = 0; i < mymachines.length; i++) {
            mymachines[i].buyCoke();
            System.out.println(mymachines[i]);
        }
    }
}
```
Isn't there an easier way?
Do you see anything that might cause a problem?

```java
public class CokematicTester {
    public static void main(String[] args) {
        Cokematic[] mymachines = new Cokematic[10];
        mymachines[0] = new Cokematic(100, "Chan Centre", 185, 201.25);
        mymachines[1] = new Cokematic(150, "Law School", 92, 100.50);
        mymachines[2] = new Cokematic(200, "Main Library", 370, 412.75);

        for (int i = 0; i < mymachines.length; i++) {
            mymachines[i].buyCoke();
            System.out.println(mymachines[i]);
        }
    }
}
```
NULL Pointers

A NULL pointer is a special reference that indicates “no object”

• Every newly declared variable of a class type initially has the value “null”, until we assign a reference to a real object to it

• We can use this in arrays to detect how many elements we have already initialized

• In our example:
  – Can have at most 10 pop machines (length of array)
  – But maybe I have only 3 installed right now...
Isn't there an easier way?

Adding this test eliminates the null pointer exception:

```java
public class CokematicTester2
{
    public static void main (String[] args)
    {
        Cokematic[] mymachines = new Cokematic[10];
        mymachines[0] = new Cokematic(100, "Chan Centre", 185, 201.25);
        mymachines[1] = new Cokematic(150, "Law School", 92, 100.50);
        mymachines[2] = new Cokematic(200, "Main Library", 370, 412.75);
        for(int i = 0; i < mymachines.length; i++)
        {
            if (mymachines[i] != null)
            {
                mymachines[i].buyCoke();
                System.out.println(mymachines[i]);
            }
        }
    }
}
```
Isn't there an easier way?

So now that error has been resolved, and we have an array that holds all our Cokematic objects. We've solved our original problem of keeping track of different types of data for each of our vending machines while holding it all in a single data structure, right?
Isn't there an easier way?

So now that error has been resolved, and we have an array that holds all our Cokematic objects. We've solved our original problem of keeping track of different types of data for each of our vending machines while holding it all in a single data structure, right?

Wrong. The fact that we got that null pointer exception should raise a warning flag for you. Isn't that the sort of thing we might expect to happen if we let any other class written by another programmer (maybe even ourselves) just do whatever he or she wants with our precious array of Cokematic objects? Should we allow anyone to add a new Cokematic object anywhere in the array? Or remove one?

How do we address this issue?
Isn't there an easier way?
We go back to our old friend, the principle of encapsulation. What does "encapsulation" say we should do?
Isn't there an easier way?

We go back to our old friend, the principle of encapsulation. What does "encapsulation" say we should do?

We should strive to preserve the integrity of our data -- the array of Cokematic objects -- by providing a collection of methods that must be used to get at the data. We do that by building a class definition that incorporates an array of Cokematic objects that has private access, appropriate constructor methods for creating objects of this class, and instance methods for accessing the data.

This not only protects our data from being munged* but it also makes it possible for others to use our data without knowing the underlying details.

* "munged" is a technical term.
Isn't there an easier way?

Here's how we could build a class definition that encapsulates our array of Cokematic objects and provides the beginnings of an interface.

```java
public class CokeEmpire
{
    private Cokematic[] collection;     // what does this do?
    private int nextAvailable;

    public CokeEmpire()
    {
        collection = new Cokematic[10];   // what does this do?
        nextAvailable = 0;
    }
}
```
Isn't there an easier way?

Here's how we could build a class definition that encapsulates our array of Cokematic objects and provides the beginnings of an interface.

```java
public void addCokematic(int cans, String loc, int sold, double cash) {
    if (nextAvailable < collection.length) {
        collection[nextAvailable] =
            new Cokematic(cans, loc, sold, cash);
        nextAvailable++;
    } else {
        System.out.println("Sorry, your empire is full.");
    }
}
```
Isn't there an easier way?

Here's how we could build a class definition that encapsulates our array of Cokematic objects and provides the beginnings of an interface.

```java
public Cokematic getCokematic(String loc) {
    Cokematic foundmachine = null;
    for (int i = 0; i < collection.length; i++) {
        // or i < nextAvailable might avoid unnecessary searching
        if ((collection[i] != null) && loc.equals(collection[i].getLocation())) {
            foundmachine = collection[i];
        }
    }
    return foundmachine;
}
```
Isn't there an easier way?

Here's a version of the getCokematic method using a while loop, which might be a better loop for this purpose.

```java
public Cokematic getCokematic(String loc)
{ /* this version assumes that all the array locations up to
the one indicated by nextAvailable hold references to
Cokematic objects. When this method finds the desired
Cokematic object, the reference to the object is stored
in the foundmachine variable. When that variable no longer
holds the null pointer, the while loop stops. */

Cokematic foundmachine = null;
int i = 0;
while ((foundmachine == null) && (i < nextAvailable))
{
    if (loc.equals(collection[i].getLocation()))
    {
        foundmachine = collection[i];
    }
    i++;
}
return foundmachine;
}
```
Isn't there an easier way?

Now, executing something like this:

```java
CokeEmpire myMachines = new CokeEmpire();
```

results in this:
Isn't there an easier way?

We can now start populating this array with Cokematic objects:
Isn't there an easier way?

We can now start populating this array with Cokematic objects:

```java
myMachines.addCokematic(100, "Chan Centre", 185, 201.25);
```
Isn't there an easier way?

We can now start populating this array with Cokematic objects:

```java
myMachines.addCokematic(150, "Law School", 92, 100.50);
```
Isn't there an easier way?

We can now start populating this array with Cokematic objects:

```javascript
myMachines.addCokematic(200, "Main Library", 370, 412.75);
```
Isn't there an easier way?

We can now start populating this array with Cokematic objects:

...and so on...

```plaintext
myMachines

0
1
2
3
4
5
6
7
8
9

Cokematic
numberOfCans: 100
location: "Chan Centre"
cansSold: 185
cashIn: 201.25

Cokematic
numberOfCans: 150
location: "Law School"
cansSold: 92
cashIn: 100.50

Cokematic
numberOfCans: 200
location: "Main Library"
cansSold: 370
cashIn: 412.75
```