News

- Assignment 2
  - do not wait until last minute
  - corrections to ASCIIArtiste.java posted (Sun)
  - definitely read WebCT bboards!
- Remember CSLC available!
  - Mon-Thu 10-6, Fri 10-4, x150 (near Reboot)
  - extra TA lab coverage for A2 help:
    - Sun 12-2 Parker
    - Tue 4-6 Hastings, 6-8 Leavitt

Reading

- This week: 8.1, 8.5-8.7, topics 6.3 and 6.4
- Next week: no new reading

Recap: Array Declaration and Types

- Just like ordinary variable, must declare array before we use it
- give array a type
- Since cansSold contains integers, make integer array:
  ```java
  int[] cansSold = new int[10]
  ```
- Looks like variable declaration, except:
  - empty brackets on the left tell Java that cansSold is an array...
  - the number in the brackets on the right tell Java that array should have room for 10 elements when it's created

Recap: Array Test1

```java
public class ArrayTest1 {
    public static void main(String[] args) {
        final int ARRAYSIZE = 10;
        int[] cansSold = new int[ARRAYSIZE];
        cansSold[0] = 185;
        cansSold[1] = 92;
        cansSold[2] = 370;
        cansSold[3] = 485;
        cansSold[5] = 84;
        cansSold[6] = 563;
        System.out.println("Element 4 is " + cansSold[4]);
    }
}
```
Recap: Array Declaration and Types

public class ArrayTest2
{
    public static void main(String[] args)
    {
        int[] cansSold = {185, 92, 370, 485, 209, 128, 84, 151, 32, 563};
        // do useful stuff here
        System.out.println("Element 4 is ", cansSold[4]);
    }
}

![Can also use initializer list! Right side of declaration does not include type or size! Java figures out size by itself! Types of values on right must match type declared on left! Initializer list may only be used when array is first declared]

Objectives

- Understanding when and how to use
  - arrays of objects
  - 2D arrays

Histogram Loop Example

Now use same data as basis for histogram

Write one loop to look at value associated with each row of array

For each value print a line with that many asterisks

Program then reads the value 8, prints a line of 8 asterisks, and so on.

Need outer loop to read individual values in the array
Need inner loop to print asterisks for each value

Storing Different Data Types

Could use two arrays of same size but with different types

Storing Different Data Types

Could use two arrays of same size but with different types

Write program to compare what's been collected from each machine vs. how much should have been collected?
Storing Different Data Types

- Write program to compare what's been collected from each machine vs. how much should have been collected?

```
public class ArrayTest4 {
    public static void main(String[] args) {
        int[] cansSold = {185, 92, 370, 485, 209, 128, 84, 151, 32, 563};
        double[] cashIn = {201.25, 100.50, 412.75, 555.25, 195.00, 160.00, 105.00, 188.75, 40.00, 703.75};
        for (int i = 0; i < cansSold.length; i++) {
            System.out.println("Machine " + (i + 1) + " off by "+
            (expected - cashIn[i]));
        }
    }
}
```

Could use two arrays of same size but with different types

- Storing Different Data Types

- Arrays With Non-Primitive Types

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

- Arrays of Objects

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

- Great if you're always storing primitives like integers or floating point numbers
- What if we want to store String types too?
- Remember that String is an object, not a primitive data type

- Storing Different Data Types

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

- Then we create array of objects
- In this case objects will be Strings
- Array won’t hold actual object
- Holds references: pointers to objects

```
String[] location = new String[10];
```

Arrays With Non-Primitive Types

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

- Now we can put references to Strings in our String array.

```
location[0] = "Chan Centre";
```

Arrays With Non-Primitive Types

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

- Now we can put references to Strings in our String array.

```java
location[0] = "Chan Centre";
location[1] = "Law School";
location[2] = "Main Library";
```

- Or we could have done this:

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

Arrays of Objects

- Each individual String object in array of course has all String methods available.
- For example, what would this return?

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

Arrays of Objects

- Each individual String object in array of course has all String methods available.
- For example, what would this return?

```java
location[2].length()
```
Cokematic object design - contains
- number of cans remaining: integer
- location: String,
- number of cans sold: integer
- cash collected: double

Arrays and Object Design

- Cokematic object design - contains
  - number of cans remaining: integer
  - location: String,
  - number of cans sold: integer
  - cash collected: double

Cokematic

- Cokematic object design - contains
  - number of cans remaining: integer
  - location: String,
  - number of cans sold: integer
  - cash collected: double

  public class Cokematic{
    private int numberOfCans;
    private String location;
    private int cansSold;
    private double cashIn;
    public Cokematic(int cans, String loc, int sold, double cash)
        numberOfCans = cans;
        location = loc;
        cansSold = sold;
        cashIn = cash;
        System.out.println("Adding machine");
    }

    public String getLocation()  {    return location;  }  public int getCansSold()  {    return cansSold;  }  public double getCashIn()  {    return cashIn;  }  public void buyCoke(){
        if (numberOfCans > 0) {
            numberOfCans = numberOfCans - 1;
            cashIn = cashIn + 1.25;
            System.out.println("New a Coke");
            System.out.println(numberOfCans + " remaining");
        } else {
            System.out.println("Sold out.");
        }
    }
    public void reloadMachine(int newCans)
        numberOfCans = numberOfCans + newCans;
        System.out.println("reloading machine");
    }  public int getNumberOfCans()  {
        return numberOfCans;
    }
    public String toString()  {
        return (location + "  sold: " + cansSold + " left: " + numberOfCans + " made: " + cashIn);
    }
}
In driver, executing

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

Results in

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

Note: leaving out methods in UML diagrams from now on to fit on page

In driver, executing

CokeEmpire myMachines = new CokeEmpire();

Contains array of Cokematic objects

public class CokeEmpire
{
    private Cokematic[] collection; // what does this do?
    public CokeEmpire()
    {
        collection = new Cokematic[10]; // what does this do?
    }
    public void addCokematic(int index, int cans, String loc, int sold, double cash)
    {
        collection[index] = new Cokematic(cans, loc, sold, cash);
    }
    public Cokematic getCokematic(int index)
    {
        return collection[index];
    }
}
In driver, executing

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

results in

```java
myMachines
```

Populate array with Cokematic objects

```java
myMachines.addCokematic(0, 100, "Chan Centre", 185, 201.25);
```

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

```java
myMachines.addCokematic(1, 150, "Law School", 92, 100.50);
```

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

```java
myMachines.addCokematic(2, 200, "Main Library", 370, 412.75);
```

CokeEmpire numberOfCans: 200
location: "Main Library"
cansSold: 370
cashIn: 412.75

What does this return?

```java
myMachines.getCokematic(1).getCansSold()
```

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

```java
myMachines.getCokematic(1).getCansSold()
```

CokeEmpire numberOfCans: 150
location: "Law School"
cansSold: 92
cashIn: 100.50
What does this return?

myMachines.getCokematic(1).getCansSold()

CokeEmpire

myMachines

0 1 2 3 4 5 6 7 8 9

Cokematic

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

cansSold: 92
cashIn: 100.50

cansSold: 370
cashIn: 412.75

Arrays of Arrays

In any given array, all data must be of same type

In any given array, all data must be of same type

All arrays in array of arrays must be of same type
Arrays of Arrays

- In any given array, all data must be of same type
- All arrays in array of arrays must be of same type
- So easier to use a two-dimensional array!

Two-Dimensional Arrays

- In Java, 2D array implemented internally as array of arrays
- but externally syntax of 2D array may seem easier to use
- Typical control structure for computing with 2D array is nested loop
- loop within another loop
- Let's write program to
  - load array with values shown
  - print contents of array

```java
public class ArrayTest5 {
    public static void main(String[] args) {
        int[][] multTable = new int[4][3];
        for (int row = 0; row < multTable.length; row++) {
            for (int col = 0; col < multTable[row].length; col++) {
                multTable[row][col] = row * col;
            }
        }
    }
}
```
public class ArrayTest5 {
    public static void main(String[] args) {
        int[][] multTable = new int[4][3];
        for (int row = 0; row < multTable.length; row++) {
            for (int col = 0; col < multTable[row].length; col++) {
                multTable[row][col] = row * col;
            }
        }
        for (int row = 0; row < multTable.length; row++) {
            for (int col = 0; col < multTable[row].length; col++) {
                System.out.print(multTable[row][col] + " ");
            }
            System.out.println();
        }
    }
}