University of British Columbia
CPSC 111, Intro to Computation
Jan-Apr 2006

Tamara Munzner

Arrays II

Lecture 15, Thu Mar 2 2006

based on slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

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

Scope Revisited, With Diagrams

- Common confusion on midterm: what's wrong?

```
public class RoachPopulation {
 private int myPopulation;
 private static final double KILL_PERCENT = 0.10;
 public RoachPopulation(int population) {
   this.myPopulation = population;
 }
 public void waitForDoubling(int newPopulation)
  newPopulation *= 2;
 }
 public int getRoaches() {
  return this.myPopulation
 }
}
```

```
public class RoachPopulationDriver {
 public static void main (String[] args) {
  RoachPopulation r = new RoachPopulation(10);
  int num = r.getRoaches();
  r.waitForDoubling();
  num = r.getRoaches();
 }
}
```

Recap: Array Declaration and Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

- Just like ordinary variable, must
  - declare array before we use it
  - give array a type
- Since cansSold contains integers, make integer array:

```
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 Declaration and Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

```
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[4] = 209;
    cansSold[5] = 128;
    cansSold[6] = 84;
    cansSold[7] = 151;
    cansSold[8] = 32;
    cansSold[9] = 563;

    // do useful stuff here
    System.out.println("Element 4 is " +
                       cansSold[4]);
  }
}
```

Recap: Array Declaration and Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

```
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

Histogram Loop Example

	numbers	
0	6	******
1	8	********
2	11	***********
3	18	******************
4	20	********************
5	17	*****************
6	14	**************
7	10	**********
8	5	*****
9	2	**

- 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
    - For example, if program reads value 6 from the array, should print line of 6 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

Objectives

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

Storing Different Data Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

Storing Different Data Types

	cansSold		cashIn
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

Could use two arrays of same size but with different types

Storing Different Data Types

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

	cansSold		cashIn
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

Could use two arrays of same size but with different types

Storing Different Data Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	703.75

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?

```
public class ArrayTest4
{
  public static void main(String[] args)
  {
    double expected;
    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++)
    {
      expected = cansSold[i] * 1.25;
      System.out.println("Machine " + (i + 1) +
                         " off by $" +
                         (expected - cashIn[i]));
    }
  }
}
```

Storing Different Data Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	703.75

Could use two arrays of same size but with different types

What happens when we run the program?

- 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)
  {
    double expected;
    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++)
    {
      expected = cansSold[i] * 1.25;
      System.out.println("Machine " + (i + 1) +
                         " off by $" +
                         (expected - cashIn[i]));
    }
  }
}
```

Storing Different Data Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	703.75

```
Machine 0 off by $30.0
Machine 1 off by $14.5
Machine 2 off by $49.75
Machine 3 off by $51.0
Machine 4 off by $66.25
Machine 5 off by $0.0
Machine 6 off by $0.0
Machine 7 off by $0.0
Machine 8 off by $0.0
Machine 9 off by $0.0
```

Somebody has been stealing from the machines after all! We need an anti-theft plan…

Arrays With Non-Primitive Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	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

Arrays With Non-Primitive Types

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	703.75

	location
0	
1	
2	
3	
4	
5	
6	
7	
8	
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

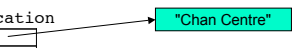
```
String[] location = new String[10];
```

Arrays of Objects

	cansSold
0	185
1	92
2	370
3	485
4	209
5	128
6	84
7	151
8	32
9	563

	cashIn
0	201.25
1	100.50
2	412.75
3	555.25
4	195.00
5	160.00
6	105.00
7	188.75
8	40.00
9	703.75

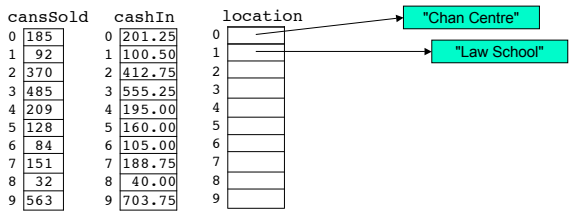
	location
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	



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

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

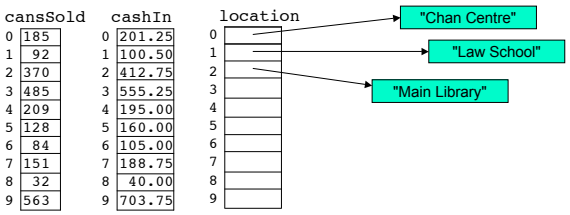
Arrays of Objects



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

```
location[0] = "Chan Centre";
location[1] = "Law School";
```

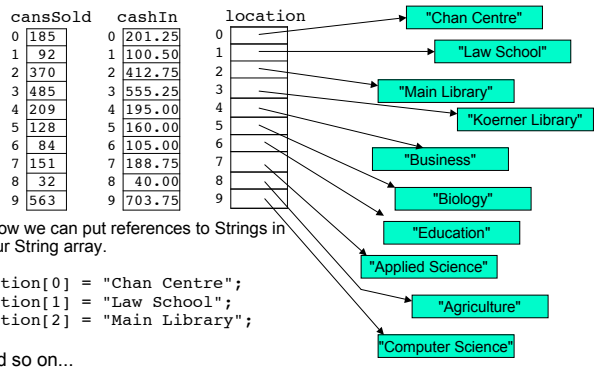
Arrays of Objects



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

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

Arrays of Objects

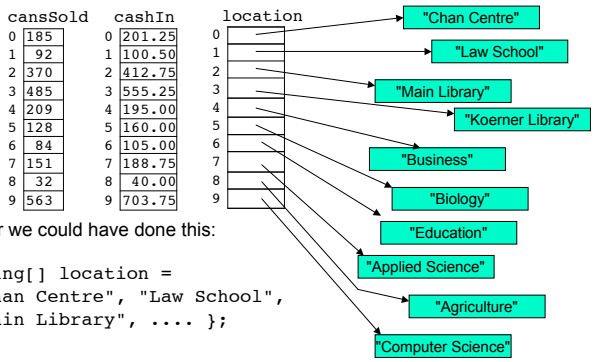


- Now we can put references to Strings in our String array.

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

...and so on...

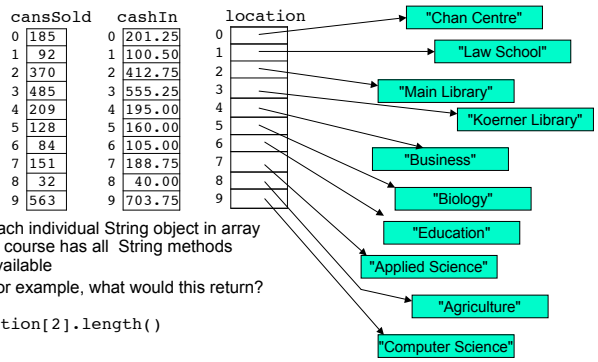
Arrays of Objects



- Or we could have done this:

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

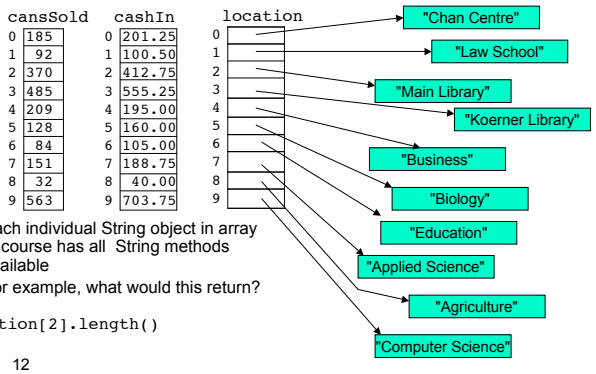
Arrays of Objects



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

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

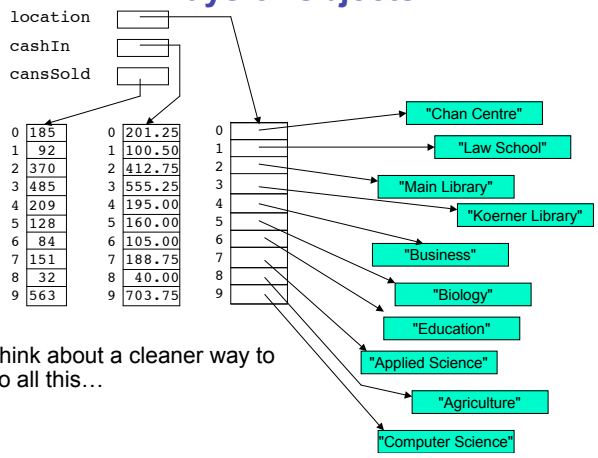
Arrays of Objects



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

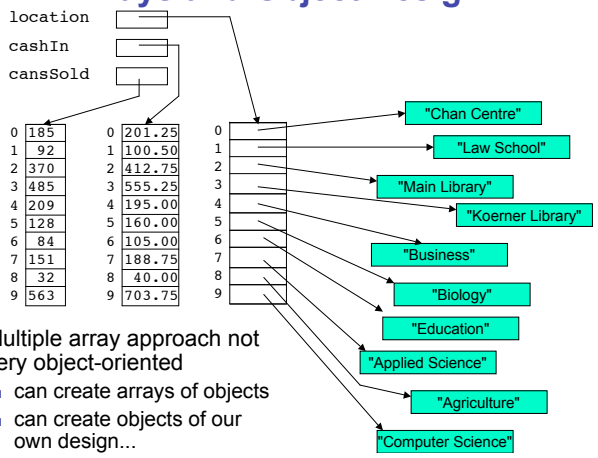
```
location[2].length()
```

Arrays of Objects



- Think about a cleaner way to do all this…

Arrays and Object Design



- Multiple array approach not very object-oriented
  - can create arrays of objects
  - can create objects of our own design...

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");
  }
```

Cokematic

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

```
public void buyCoke()
{
  if (numberOfCans > 0)
  {
    numberOfCans = numberOfCans - 1;
    cansSold = cansSold + 1;
    cashIn = cashIn + 1.25;
    System.out.println("Have a Coke");
    System.out.println(numberOfCans + " remaining");
  }
  else
  {
    System.out.println("Sold out.");
  }
}
```

```
  public String getLocation()
  {
    return location;
  }

  public int getCansSold()
  {
    return cansSold;
  }

  public double getCashIn()
  {
    return cashIn;
  }

  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);
  }
}
```

Cokematic

- In driver, executing

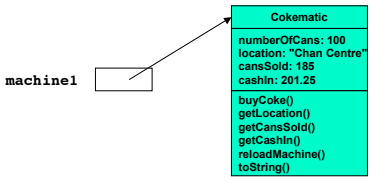
```
Cokematic machine1 = new Cokematic(100, "Chan Centre",
                                   185, 201.25);
```

Cokematic

- In driver, executing

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

- Results in

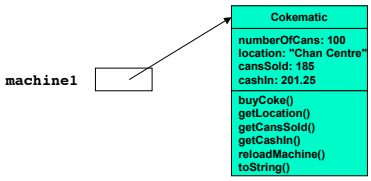


Cokematic

- In driver, executing

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

- Results in



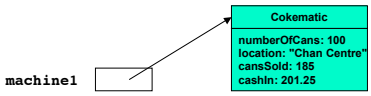
- Note: leaving out methods in UML diagrams from now on to fit on page

Cokematic

- In driver, executing

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

- Results in



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

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];
  }
}
```

CokeEmpire

- In driver, executing:

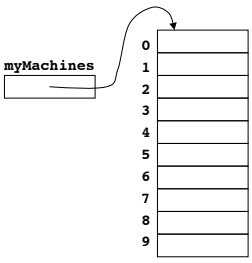
```
CokeEmpire myMachines = new CokeEmpire();
```

CokeEmpire

- In driver, executing

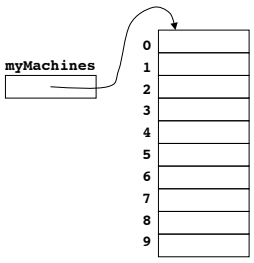
```
CokeEmpire myMachines = new CokeEmpire();
```

- results in



CokeEmpire

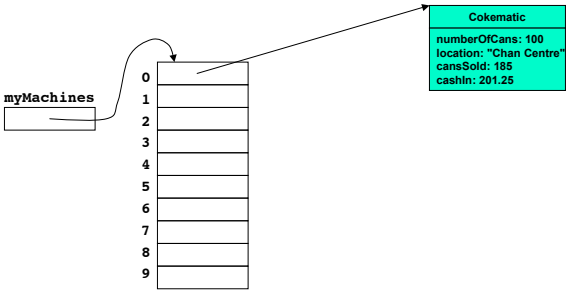
- Populate array with Cokematic objects



CokeEmpire

- Populate array with Cokematic objects

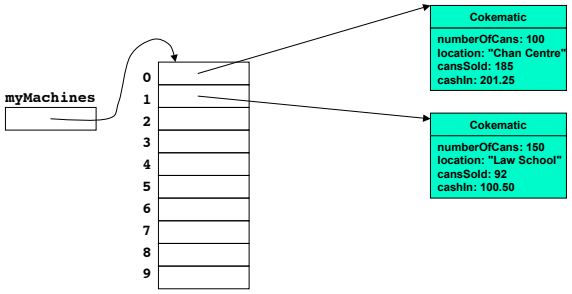
```
myMachines.addCokematic(0, 100, "Chan Centre", 185, 201.25);
```



CokeEmpire

- Populate array with Cokematic objects

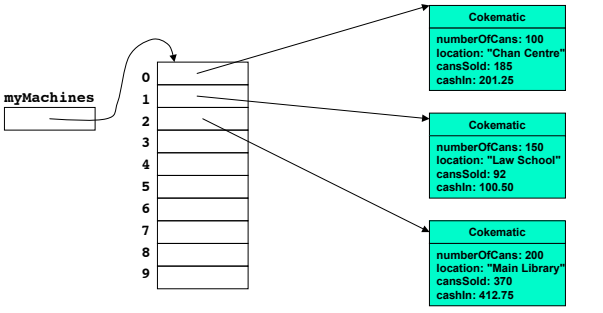
```
myMachines.addCokematic(1, 150, "Law School", 92, 100.50);
```



CokeEmpire

- Populate array with Cokematic objects

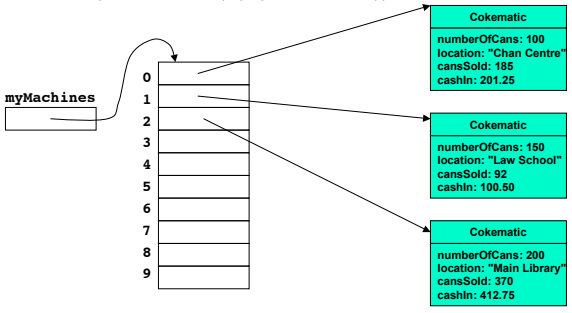
```
myMachines.addCokematic(2, 200, "Main Library", 370, 412.75);
```



CokeEmpire

- What does this return?

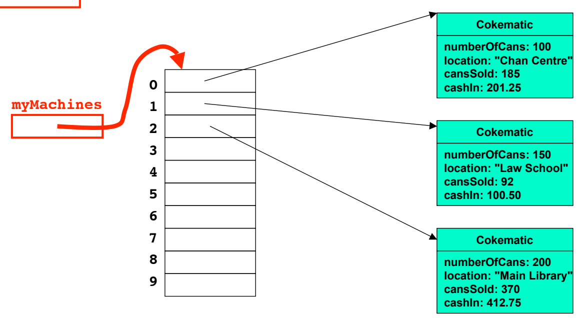
```
myMachines.getCokematic(1).getCansSold()
```



CokeEmpire

- What does this return?

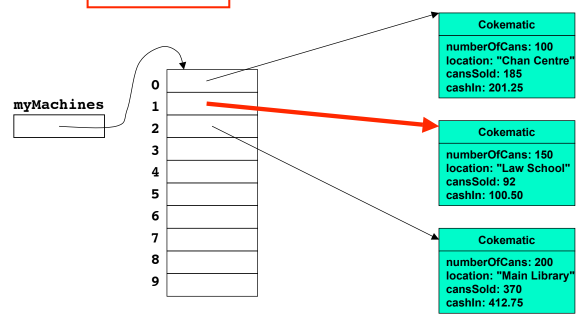
```
myMachines.getCokematic(1).getCansSold()
```



CokeEmpire

- What does this return?

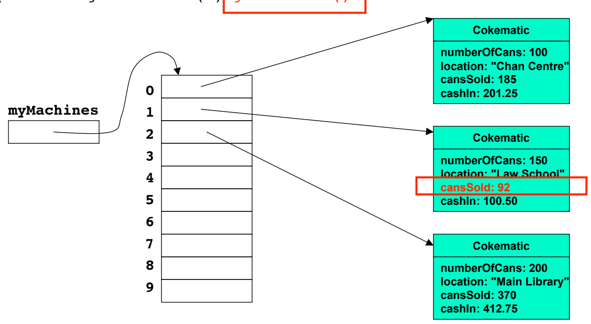
```
myMachines.getCokematic(1).getCansSold()
```



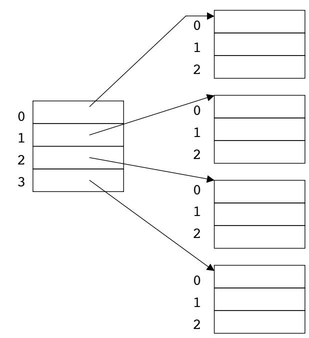
CokeEmpire

- What does this return?

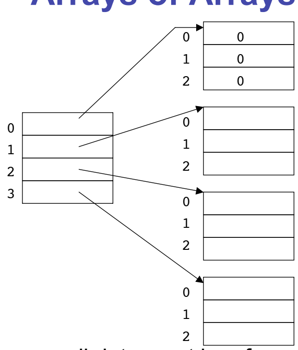
```
myMachines.getCokematic(1).getCansSold()
```



Arrays of Arrays

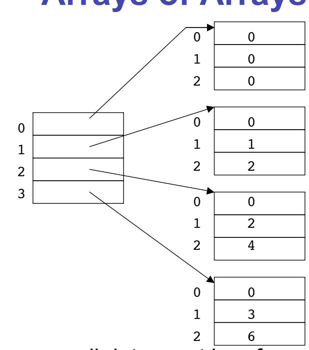


Arrays of Arrays



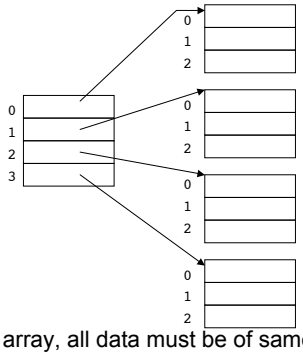
- In any given array, all data must be of same type

Arrays of Arrays



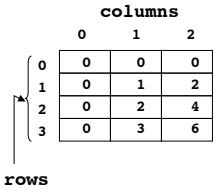
- In any given array, all data must be of same type
- All arrays in array of arrays must be of same type

Arrays of Arrays



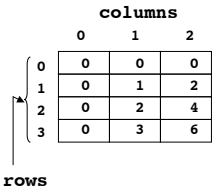
- In any given array, all data must be of same type
- All arrays in array of arrays must be of same type
- So easier to use a two-dimensional array!

Two-Dimensional Arrays



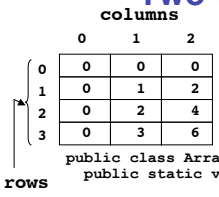
- In Java, 2D array implemented internally as array of arrays
  - but externally syntax of 2D array may seem easier to use

Two-Dimensional Arrays



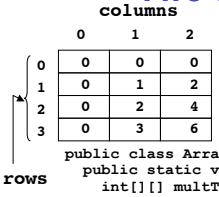
- In Java, 2D array implemented internally as array of arrays
  - but externally syntax of 2D array may seem easier to use
- Typical control structure for computing with 2D array is nested loop
  - loop within another loop
- Let's write program to
  - load array with values shown
  - print contents of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

Two-Dimensional Arrays



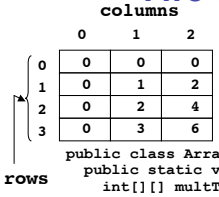

```
public class ArrayTest5 {
  public static void main(String[] args) {









  }
}
```

Two-Dimensional Arrays



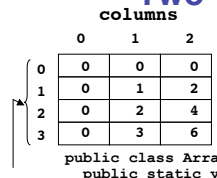

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];









  }
}
```

Two-Dimensional Arrays



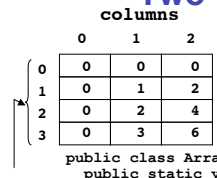

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];


      for (int col = 0; col < multTable[row].length; col++) {
        multTable[row][col] = row * col;
      }









  }
}
```

Two-Dimensional Arrays



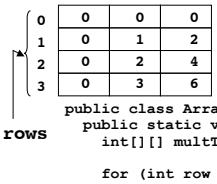

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

    for (int row = 0; row < multTable.length; row++){
      for (int col = 0; col < multTable[row].length; col++) {
        multTable[row][col] = row * col;
      }
    }




  }
}
```

Two-Dimensional Arrays



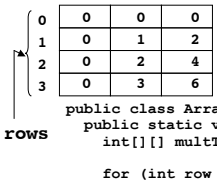

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

    for (int row = 0; row < multTable.length; row++){
      for (int col = 0; col < multTable[row].length; col++) {
        multTable[row][col] = row * col;
      }
    }


    for (int col = 0; col < multTable[row].length; col++){
      System.out.print(multTable[row][col] + " ");
    }


  }
}
```

Two-Dimensional Arrays




```
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] + " ");
      }

    }
  }
}
```

Two-Dimensional Arrays




```
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();
    }
  }
}
```