University of British Columbia
CPSC 111, Intro to Computation
2009W2: Jan-Apr 2010
Tamara Munzner

Arrays II

Lecture 22, Wed Mar 10 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10

News

- Assignment 2 out
  - due Fri Mar 26
  - start now, it's challenging!

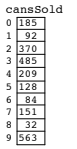
Recap: Array Declaration and Types

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

```
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

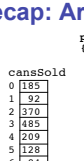



```
public class ArrayTest1
{
  public static void main(String[] args)
  {
    final int ARRAYSIZE = 10;
    int[] cansSold = new int[ARRAYSIZE];

    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



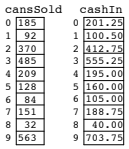

```
public class ArrayTest2
{
  public static void main(String[] args)
  {

    int[] cansSold = {185, 92, 370, 485, 209,
                      128, 84, 151, 32, 563};

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

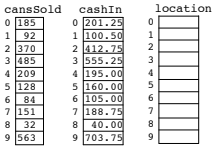
- Can also use initializer list
- Right side of declaration does not include type or size
  - Java figures out size by itself
- Types of values on right must match type declared on left
- Initializer list may only be used when array is first declared

Arrays With Non-Primitive Types



- 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

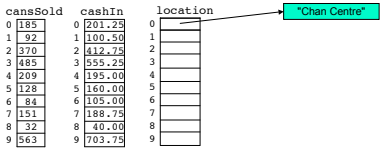
Arrays With Non-Primitive Types



- 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];
```

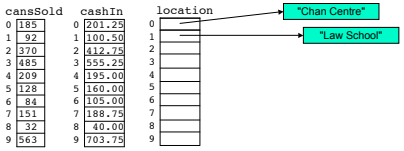

Arrays of Objects



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

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

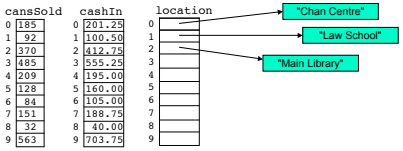

Arrays of Objects



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

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

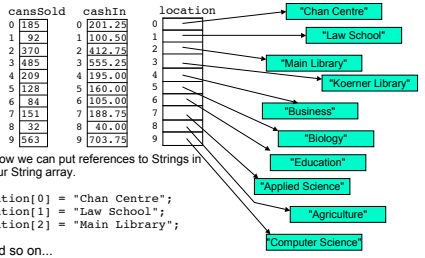

Arrays of Objects



- 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

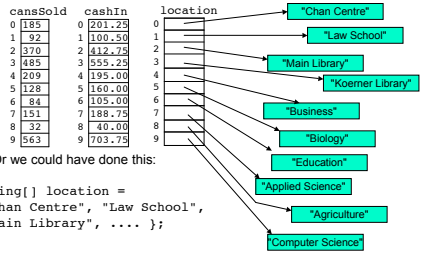


- Now we can put references to Strings in our String array.

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

...and so on...

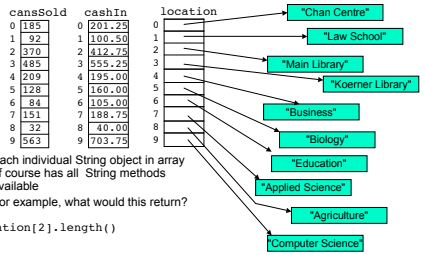
Arrays of Objects



- Or we could have done this:

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

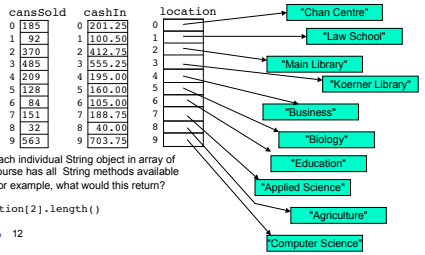

Arrays of Objects



- 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

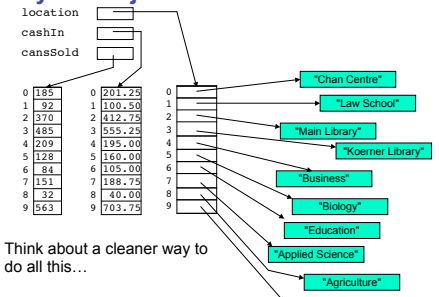


- Each individual String object in array of course has all String methods available
- For example, what would this return?

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

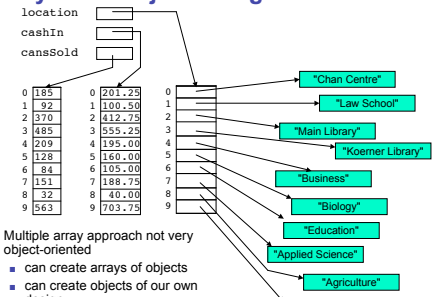
- 12

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

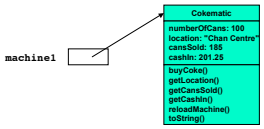
```
Cokematic machine1 = new Cokematic(100, "Chan Centre",
                                   185, 201.25);
```


Cokematic

- In driver, executing

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

- Results in

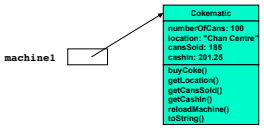


Cokematic

- In driver, executing

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

- Results in



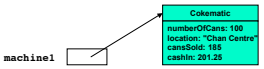
- Note: leaving out methods in UML diagrams from now on to fit on page

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:

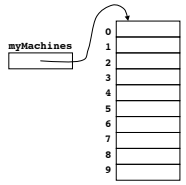
```
CokeEmpire myMachines = new CokeEmpire();
```


CokeEmpire

- In driver, executing

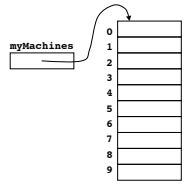
```
CokeEmpire myMachines = new CokeEmpire();
```

- results in



CokeEmpire

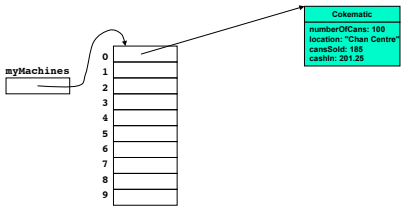
- Populate array with Cokematic objects



CokeEmpire

- Populate array with Cokematic objects

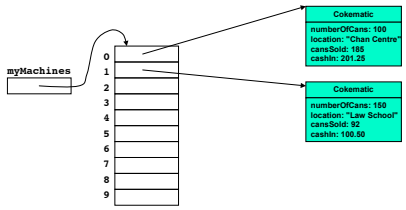
```
myMachines.addCokematic(0, 100, "Chan Centre", 185, 201.25);
```



CokeEmpire

- Populate array with Cokematic objects

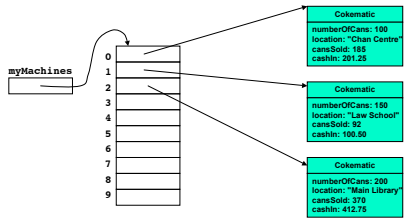
```
myMachines.addCokematic(1, 150, "Law School", 92, 100.50);
```



CokeEmpire

- Populate array with Cokematic objects

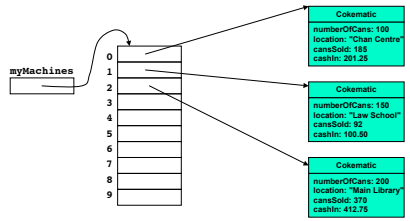
```
myMachines.addCokematic(2, 200, "Main Library", 370, 412.75);
```



CokeEmpire

- What does this return?

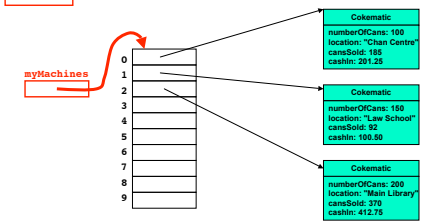
```
myMachines.getCokematic(1).getCansSold()
```



CokeEmpire

- What does this return?

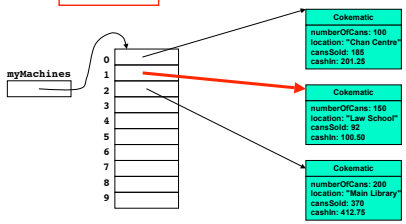
```
myMachines.getCokematic(1).getCansSold()
```



CokeEmpire

- What does this return?

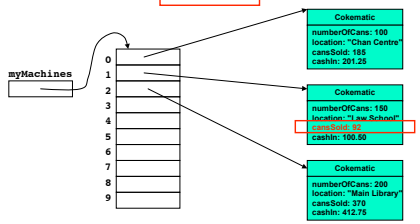
```
myMachines.getCokematic(1).getCansSold()
```



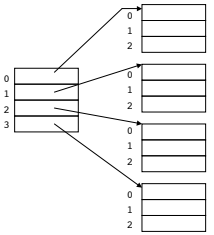
CokeEmpire

- What does this return?

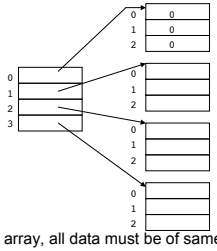
```
myMachines.getCokematic(1).getCansSold()
```



Arrays of Arrays

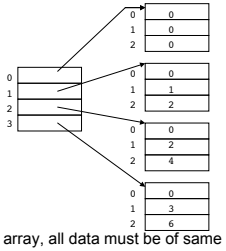


Arrays of Arrays



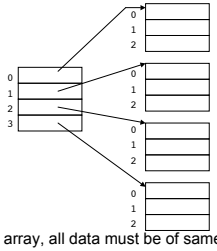
- In any given array, all data must be of same type

Arrays of Arrays



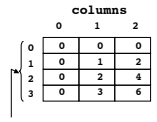
- In any given array, all data must be of same type
- All arrays in array of arrays must be of same type

Arrays of Arrays



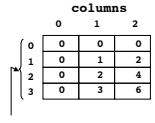
- In any given array, all data must be of same type
- All arrays in array of arrays must be of same type
- So easier to use a two-dimensional array!

Two-Dimensional Arrays



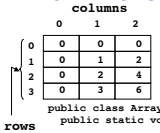
- In Java, 2D array implemented internally as array of arrays
  - but externally syntax of 2D array may seem easier to use

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

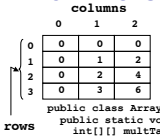
Two-Dimensional Arrays




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

```
}
}
```

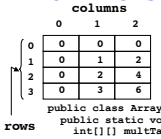

Two-Dimensional Arrays




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

```
}
}
```


Two-Dimensional Arrays

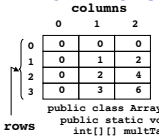



```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

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

```
}
}
```


Two-Dimensional Arrays

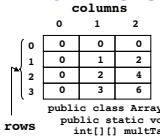



```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

    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



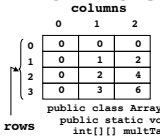

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

    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



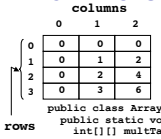

```
public class ArrayTest5 {
  public static void main(String[] args) {
    int[][] multTable = new int[4][3];

    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();
    }
```

```
  }
}
```