- Previous Lecture:
  - Review of polymorphism
  - Two-dimensional array of numbers
- Today's Lecture:
  - 2-d array of objects—a `String` is an object
  - Review arrays of objects (1- and 2-d)
- Reading:
  - Review type String in Sec 2.9.
  - Sec 10.3 (No need to memorize the methods! Just be aware of the kinds of `String` methods available for future reference.)

If there may be a "missing row," check for null!

Given a 2-d integer array `x`, calculate the sum of all entries in the array.

```
int sum = 0;  // sum so far

for (int r=0; r<x.length; r++)
   for (int c=0; c<x[r].length; c++)
      sum += x[r][c];
```

x[r]!=null &&

When might I use `public` fields?

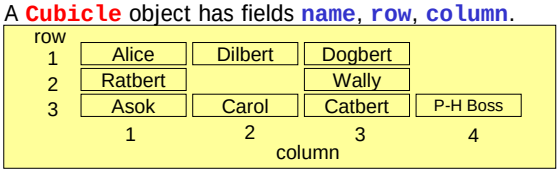
- Client needs easy access to fields
- The only "service" that the class provides is to collect related data under one (class) name
- One should still consider using `private` fields though!

Example: cubicle world

Implement a class `CubicleWorld` that has a 2-d array of `Cubicle`s, so a `CubicleWorld` is like a floor plan.

The array has dimensions just big enough to store the entire floor plan including internal spaces.

A `Cubicle` object has fields `name`, `row`, `column`.



Instantiating 2-d arrays

- A 2-d array is a 1-d array of 1-d arrays
- You can create one dimension at a time:
  1. Declare a reference variable for the 2-d array
  2. Set 1st dimension (# rows): create a 1-d array to hold the row references
  3. Set 2nd dimension (# columns) one row at a time: create the individual arrays that store the values (or object references) of interest
  4. Now you can assign values (or references) into the cells of the array

What we learned...

- Develop/implement algorithms for problems
- Develop programming skills
  - Design, implement, document, test, and debug
- Apply programming languages
  - Control structures
  - Function/methods for reducing redundancy
  - Data structure
  - Fundamentals of object oriented programming, including inheritance
- Specific tasks
  - Simulating systems
  - Sorting
  - Searching
  - Plotting numeric data

```
/* A Cubicle in some office.  Row and column numbers start at 1 */
// The only "service" that this class provides is to collect related
// data in a Cubicle object.  (Notice that there are no methods other
// than the most basic ones: constructor and toString.)  In such a case,
// one may choose to make the fields public.
class Cubicle {
  public String name;  //name of person who uses the Cubicle
  public int row;      //row number of the Cubicle
  public int column;   //column number of the Cubicle

  /* Constructor: Person n uses this Cubicle which is in row r, column c */
  public Cubicle(String n, int r, int c) {
    name= n;
    row= r;
    column= c;
  }

  /* = a String containing the data values of this Cubicle */
  public String toString() {
    return  name + "'s cubicle is at row " + row + ", column " + column ;
  }
} //class Cubicle
```

```
/* A CubicleWorld is a a 2-d array of Cubicles */
public class CubicleWorld {
  private Cubicle[][] floorPlan;  //Refers to 2-d array of Cubicles
  private int rows;               //Number of rows in floor plan
  private int[] columns;        //columns[i] is # of columns in row i of floor plan

  /* Constructor: set the values of the fields */
  public CubicleWorld(int rows, int[] cols) {




    //Set 1st dimension of floor plan (number of rows)


    //Set 2nd dimension of floor plan one row at a time




  }

  /* Fill this CubicleWorld's floor plan */
  public void fillFloorPlan(Cubicle[] cubes) {




  }

  /* =Get Cubicle at row r, column c.  Row, column numbers start at 1 */
  public Cubicle getCubicle(int r, int c) {


  }

  //class CubicleWorld continues on next page
```

```
  //class CubicleWorld, continued

  /* ={Person with name s is found in this CubicleWorld}, true or false.
   * Display the Cubicle location(s) of person(s) with name s */
  public boolean findPerson(String s) {









  }


  public static void main(String[] args) {

    int rows= 3;               //Number of rows of Cubicles
    int[] columns= {3, 3, 4};  //Number of columns of Cubicles

    //Cubicle data collected as a 1-d array
    //(Remember that Cubicle row and column numbers start at 1)
    Cubicle[] workers= new Cubicle[] { new Cubicle("Alice", 1, 1),
                                       new Cubicle("Dilbert", 1, 2),
                                       new Cubicle("Dogbert", 1, 3),
                                       new Cubicle("Ratbert", 2, 1),
                                       new Cubicle("Wally", 2, 3),
                                       new Cubicle("Asok", 3, 1),
                                       new Cubicle("Carol", 3, 2),
                                       new Cubicle("Catbert", 3, 3),
                                       new Cubicle("P-H Boss", 3, 4)
                                     };

    //Create a CubicleWorld that is just big enough for all the workers
      CubicleWorld cw= new CubicleWorld(rows, columns);
    //Now put the workers (Cubicles) into the floorPlan
      cw.fillFloorPlan(workers);

    //Let's test a few cases:
      System.out.println(cw.getCubicle(1,3));
      System.out.println(cw.getCubicle(2,2));
      System.out.println(cw.getCubicle(3,4));

      boolean foundPerson;
      foundPerson= cw.findPerson("Ratbert");
      foundPerson= cw.findPerson("Garfield");
  }
} //class CubicleWorld
```