Announcements

Program 4 up, due next week  
(Questions?) 
Next OWL assignment up, due next Tuesday  
Drop date: 5 PM Wednesday  

CS-Women’s lunch - Thursday, 12:30
We’ve seen Java at two levels:

- the statement level - mechanisms for getting specific, often low-level jobs done, e.g. assignment stmts, println, etc.

- the (class and) object level - mechanisms for modeling things (objects) according to an” Objects” model -(repositories of state) - served by methods (machinery for realizing behaviors) scheme

Now we’re back to a new and very important idea in statement-level thinking: arrays. Basically, arrays give us a new way to think about variables.
Think about: students in a class; seats on an airplane, rooms in a motel, positions in deli line at supermarket. In all cases:

- Many similar, nearly anonymous, variables required
- There’s an indexing scheme for locating / identifying the variables in question:

Student 7  
Seat 23B  
Room 201  
Deli-line position 77

- some indexing schemes are more natural than others  
- some are two-dimensional
Variables in algebra -
\(x_0\)  \(x_1\)  \(y_3\) and so forth

Java notation just a variant:
\(x[0]\), \(x[1]\), \(y[3]\)

Algebra: \(x_0 = 2 * x_1\)

---------------------------------
Java: \(x[0] = 2* x[1];\)
k = 7;

nums[3] = 8;

public class ArrayTest{
    public static void main(String[] args){
        int[] firstArray = new int[10];
        for(int j = 0; j < 10; j++){
            firstArray[j] = j*j;
        }

        System.out.println("here they come");

        for(int j = 0; j < firstArray.length; j++)
            System.out.println(firstArray[j]);
    }
}
Arrays - the mental picture..

```java
int[] firstArray = new int[10];

firstArray[6] = 17;
```

firstArray.length -> 10
These sorts of expressions are possible:

```c
firstArray[4] = 9*firstArray[4];
firstArray[3] = 11;

int j = firstArray[3]/2;
firstArray[j] = 9*firstArray[j/2];
```
int[] nums = {2, 4, 6, 8, 10} makes an array of 5 ints:
System.out.println(nums[4]); -> prints 10

First 10 Fibonacci numbers:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55

-------------------------------------------
Fibonacci #s: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

int[] fibos = new int[10];
fibos[0] = 1;
fibos[1] = 1;
for(int j = 2; j < fibos.length; j++)
fibos[j] = fibos[j-1] + fibos[j-2];
Arrays are objects

When you say “length” you are invoking a constant (public final value) associated with the array.

The size of an array is determined when “new” is invoked:

    int[] someArray = new int[66];

    int[] nums;  // this is ok - variable is named

Array indices are always int, and always start at 0

Array indices end at cell length - 1
We’re going to write an application that rolls a pair of dice some number of times and reports the results as a profile of the rolls (e.g. how many 2,3, etc came up).

```java
import javax.swing.JOptionPane;
public class DiceExperiment {
    public static void main(String[] args) {
        String tossString = JOptionPane.showInputDialog("enter toss count");
        int tossCt = Integer.parseInt(tossString);
        Dice d = new Dice();
        d.multiToss(tossCt);
        d.showScoreboard();
    }
}
```
public class Dice{
    private int[] scoreboard = new int[13];
    public Dice(){
        initializeScoreboard();
    }

    public void initializeScoreboard(){
        for(int j = 0; j < 13; j++) scoreboard[j] = 0;
    }

    public int tossDie(){
        return (1+ (int)(6*Math.random()));
    }

    public int throwDice(){
        return(tossDie() + tossDie());
    }
}
public void multiToss(int tossCount){
    int score;
    for (int j = 0; j < tossCount; j++){
        score = throwDice();
        scoreboard[score]++;
    }
}

public int[] getScoreboard(){return scoreboard; }

public void showScoreboard(){
    for(int j = 2; j < 13; j++)
        System.out.println("toss of " + j + " " + scoreboard[j]);
}
} // ends class
The Scoreboard

...  23  ...

5   6   7   8   9   ...

score = throwDice();
scoreboard[score]++;;

Suppose throwDice() returns 6  -> then what?
Results: (10,000 tosses)

toss of 2 303
toss of 3 543
toss of 4 807
toss of 5 1123
toss of 6 1432
toss of 7 1630
toss of 8 1389
toss of 9 1129
toss of 10 808
toss of 11 557
toss of 12 279
Arrays of objects

Infant[] kids = new Infant[10]; // array of 10 infants

Infant littleMikey = new Infant(“mike”, 3);
kids[4] = littleMikey;
Places littleMikey into cell with index 4 of the kids array.

Kid at cell 2 has wrong name; should be lilly
kids[2].setName(“lilly”);

public void allOlder(Infant[] kiddo){
    for(int j = 0; j < kiddo.length; j++)
        kiddo[j].anotherMonth();
}
A typical array problem:

Find the name of the oldest kid in an array of Infants

Assume zeroth kid is the oldest - set aside her position (0), and her age

Walk down the array (may as well start with 1)
When you find someone older:
set aside her position (j), her age

When you’re done, j holds the index of the oldest kid (who could be older???)
Get that kid, return her name!
public String oldest(Infant[] kiddo){
    // what’s name of oldest kid?
    int oldestSoFar = 0;
    int oldAge = kiddo[0].getAge();
    int curAge;
    for(int j = 1; j < kiddo.length; j++){
        curAge = kiddo[j].getAge();
        if (curAge > oldAge){
            oldAge = curAge;
            oldestSoFar = j; // location of oldest kid so far
        } // location of oldest kid so far
    } // location of oldest kid so far
    return(kiddo[oldestSoFar].getName());
}
public boolean anyBabies(Infant[] kiddo){
// are any kids less than 2 months old?
    boolean aBaby = false;
    for(int j = 0; j < kiddo.length; j++){
        if (kiddo[j].getAge() < 2){
            aBaby = true;
            break;
        }
    }
    return aBaby;
}
public boolean majorityOld(Infant[] kiddo, int a) {
// are strict majority in array older than age a?
    int old = 0;
    for (int j = 0; j < kiddo.length; j++) {
        if (kiddo[j].getAge() > a) {
            old++;
        } else old--;
    }
    return (old > 0);
}