Arrays

or handy lists of “things”
Motivation

- Need sets of things.
  - I want to keep track of all your grades in a program.
    - double grade1 = 0.99;
    - double grade2 = 0.86;
    - double grade3 = 0.89;
    - etc.
  - What a pain!
    - What if I assign 2000 homeworks?
    - Ugh. 2000 variables.

- Arrays are simpler way to store sets or lists.
Arrays as a Set

- Consider a list or set of integers
  - \{1, 5, 34, -5, 19, 271\}
- We might say that this set is called an array “a”.
- We say it has size \( n \) if there are \( n \) elements in the set.
- The array elements are numbered from 0 to \( n-1 \).

\{1, 5, 34, -5, 19, 271\}

- 0\(^{\text{th}}\) element
- 1\(^{\text{st}}\) element
- 5\(^{\text{th}}\) element
Array Numbering

Why not from 1 to n? Historical…

So we say that

- the 0th element of the array is \( a[0] \)
- the 3rd element of the array is \( a[3] \)
- the 2nd element is \( a[2] \)
- the ith element is \( a[i] \)

\( \{1, 5, 34, -5, 19, 271\} \)

\( a[0] = 1 \)
\( a[1] = 5 \)
\( a[5] = 271 \)
Arrays Live in Consecutive Memory

\{1, 5, 34, -5, 19, 271\} is stored in consecutive memory blocks inside the computer.

lives in memory block 249

lives in memory block 250

lives in memory block 251

If you’re really curious… Each block takes up some amount of space in your memory. For example, each “int” takes 4 bytes of space. (And each byte is 8 bits. Every bit is a single 0 or 1.)
Declaring Arrays

- int[] iArray = new int[14];
  - This is an array of 14 integers (numbered 0 to 13)
- double[] dArray = new double[3];
  - This is an array of 3 doubles (numbered 0 to 2)
- char[] alphabet = new char[26];
  - This is an array of 26 characters (numbered 0 to 25)
- Person[] faculty = new Person[2000];
  - This is an array of 2000 Persons (numbered 0 to 1999)

When we do this, it tells the computer to please reserve memory blocks that are this long. We will fill them up…
Can Break Up the Declaration

- These are all the same!
  - `Person[] faculty = new Person[2000];`
  - `Person[] faculty;`  
    `faculty = new Person[2000];`
  - `Person[] faculty = null;`  
    `faculty = new Person[2000];`  //re-assigns
  - `Person[] faculty = new Person[173];`  
    `faculty = new Person[2000];`  //re-assigns
Filling an Array (Option 1)

- Use it just like any variable.

```java
int[] a = new int[5];
a[0] = 34;
```
- This assigns 34 to the 0\textsuperscript{th} array element

- Can fill up many array values…
  - `a[0] = 34;`
  - `a[1] = 2;`
  - `a[2] = 45;`
  - `a[3] = -9;`
  - Called “initializing” the array.
Filling an Array (Option 2)

- Here’s another option:

```java
int[] a = new int[10];
for(int i =0; i<=9; i++)
{
    a[i] = 0;
}
```

- This initializes the array so that all values are 0.
Filling an Array (Option 3)

```java
char[] cArray = {'a', 'T', 'c', 'd', '4'};

double[] grades = {1.00, 0.99, 0.75, 0.86};
```

So now what values print out?
```java
System.out.println(cArray[1]);
System.out.println(grades[3]);
```
Initializing Arrays

- Suppose have array of record high temperatures for each day of year.

- Can I say … ?
  ```java
  int[] recordTemperature = new int[365];
  System.out.println("Record temp on Jan 1 is" + recordTemperature[0]);
  ```

- No! Not until we’ve filled it up!
  - We’ve only reserved the space in memory.

- Instead do
  ```java
  int[] recordTemperature = new int[365];
  recordTemperature[0] = 89; //sets the temperature
  System.out.println("Record temp on Jan 1 is" + recordTemperature[0]);
  ```
Initializing Complex Arrays

- Suppose have
  ```java
  Person[] student = new Person[200];
  and suppose Person has a method called “void setGrade(char grade)”
  ```

- Can I say … ?
  ```java
  Person[] student = new Person[200];
  student[3].setGrade(‘A’);
  ```

- No! We’ve reserved the space, but we haven’t initialized!
  - Same as last example.

- Instead do
  ```java
  Person[] student = new Person[200];
  student[3] = new Person();
  student[3].setGrade(‘A’);
  ```
Initializing Problems

What’s wrong with these?

- `int n[4] = {2, 2, -91, 5};`
  - **ERROR!** Wrong format. (But some languages do this!)
  - `int[] n = {2, 2, -91, 5};`

- `int[] n = new int[4];`
  - `for(int i=1; i<=4; i++)`
    - `{`
      - `n[i] = 12;`
    - `}`
  - **ERROR!** Need “for(int i=0; i<4; i++)”
How Big Is an Array?

- Can always get the size with “length”.
  - Kind of like a method, but *don’t use parentheses*.

- `int[] geese = {4,2,1};
  System.out.println(geese.length); //prints 3`
How Use Arrays?

- Treat like any other variable.
  - Here’s code that averages grades.

```java
double[] grades = {0.99, 0.83, 0.45, 0.75, 1.00};
double average = 0.0;
for(int i = 0; i<grades.length; i++)
{
    average += grades[i]; //gives 0.99+0.83+0.45+0.75+1.00
}
average /= grades.length; //gives (0.99+0.83+0.45+0.75+1.00)/5
```
public static void main(String[] args) {
    int[] iArray = new int[10];
    int i;

    for(i = 0; i<=10; i++)
    {
        iArray[i] = 2*i;
    }
}

Ahhh, finally!
Just an array!
Pass in from command line as
Java ClassName 1 3 7
with spaces in-between.

ERROR! Array overflow! Walks off the end of the array.

(Details on passing arrays into a method coming soon.)
public static void main(String[] args) {
    int[] iArray = new int[10];
    int i;
    iArray = 34;
    for (i = 0; i <= iArray.length - 1; i++) {
        iArray[i] = 2 * i;
    }
    // ERROR! Bad notation. iArray is not a variable. Only iArray[i] is a variable!
    // Have to say “iArray[7] = 34;” or something similar.
}

But check this out! Does it work? (Yes.)
public class QuoteOfTheDay
{
    public static void main(String[] args)
    {
        String[] quote = new String[3];
        quote[0] = “D’oh!”;
        quote[1] = “Mmmm, free goo.”;

        //generate a random number
        Random r = new Random();
        int randomInteger = r.nextInt();
        randomInteger %= quote.length;

        //print a random quote
        System.out.println(quote[randomInteger]);
    }
}
import javax.swing.JOptionPane;
public class School
{
    public static void main(String[] args)
    {
        String sNumGrades = JOptionPane.showInputDialog("Enter the number of assignments");
        int numGrades = Integer.parseInt(sNumGrades);

        //Array size determined on the fly! Many languages can’t do this!
        double[] dGrades = new double[numGrades];
        for(int i = 0; i<numGrades; i++)
        {
            String sGrade = JOptionPane.showInputDialog("Student’s grade is?");
            dGrades[i] = Double.parseDouble(sGrade);
        }

        //we’ll see how this works next!
        School regis = new School();
        double average = regis.averageGrade(dGrades);  // in a moment we will write this method
    }
    …   //more to come
Passing Arrays to Methods 1

- Can pass array elements one at a time.
- e.g., in previous code, when call the function `averageGrade` could say
  
  ```java
  double average = averageGrade(dGrades[0], dGrades[1], dGrades[2], ..., dGrades[2000]);
  ```

  - What a pain! And what if I decide to assign 2001 grades instead. Code has to change!

  - (But note: sometimes it’s ok to pass in just one element of an array. Maybe that’s all your method needs.)
Just tell it that the variable will be an array…

```java
public class School {
    public double averageGrade(double[] grades) {
        double dAverageGrade = 0.0;
        for (int i = 0; i < grades.length; i++) {
            dAverageGrade += grades[i];
        }
        dAverageGrade /= grades.length;
        return dAverageGrade;
    }
}
```

Tells the function to expect an array!

Use this code by saying:
```
School regis = new School();
double d = regis.averageGrade(studentGrades);
```

where studentGrades is an array of doubles.

Note this isn’t the same variable name that was passed in. Ok???
public class Time
{
    public static void main(String[] args)
    {
        float[] time = new float[3];
        time[0] = getHours();
        time[1] = getMinutes();
        time[2] = getSeconds();
        setClockTime(time);
    }

    public static void setClockTime(float[] fNewTime)
    {
        //…  lots of code …
    }

    //…etc. for other methods
}
Returning Arrays From Methods

- Just what you’d expect

```java
public char[] daveLetters()
{
    char[] dave = {'d', 'a', 'v', 'e'};

    return dave;
}
```

- But there are subtleties.
  - Must learn about pointers.
  - Stay tuned…
Multidimensional Arrays

- Same as regular arrays, but two or more subscripts!

- `int[][] iArray = new int[3][2];`

- What is this?
  - a table of values!

To remember the order, think “RC cola”

<table>
<thead>
<tr>
<th></th>
<th>column 0</th>
<th>column 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>23</td>
<td>-4</td>
</tr>
<tr>
<td>row 1</td>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>row 2</td>
<td>7</td>
<td>-2</td>
</tr>
</tbody>
</table>
Just an Array of Arrays

- int[][] iArray = new int[3][2];

- What is this really?
  - It is an array of size 3.
    - And each element is filled with an array of size 2.
      - e.g., { {23, -4},
                 {7, 49},
                 {7, -2}  };
  
- So these are the same!
  - int[][] iArray = new int[3][2];
  
- int[][] iArray = new int[3][ ];
iArray[0] = new int[2];
iArray[1] = new int[2];
iArray[2] = new int[2];

Same!
Filling Multidimensional Arrays

- Use two for loops.

```java
int[][] data = new int[10][15];
for(int i =0; i<=9; i++)
{
    for(int j =0; j<15; j++)
    {
        data[i][j] = 0;
    }
}
```

- Or fill one by one.
  - data[0][0] = 3;
  - data[0][1] = 2
  - etc.
Example

public static void main(String[] args)
{
    int numStudents = 10;
    int numAssignments = 300;
    float[][] fGrades = new float[numStudents][numAssignments];

    for(int row=0; row<fGrades.length; row++)
    {
        for(int col=0; col<fGrades[row].length; col++)
        {
            String score = JOptionPane.showInputDialog("For student number " + row + " give me their grade for assignment number " + col);
            fGrades[row][col] = Float.parseFloat(score);
        }
    }

    ...
}