COMP 110-003
Introduction to Programming
*Miscellaneous and More Arrays*

April 02, 2013

Haohan Li
TR 11:00 – 12:15, SN 011
Spring 2013
Daily Joke

**CAT:** Good boy.

Hey, is that a robot? Not just any robot. THIS is the current state-of-the-art in artificial intelligence.

We trained a 9-layered locally connected sparse autoencoder with pooling and local contrast normalization on a dataset of 10 million images. It was trained for 3 days on a cluster of 1000 machines comprising 16,000 cores.

It can now recognize 22,000 different object categories with 15.8% accuracy.

15.8%? Impressive, huh? **CAT**

Impressive? Didn't you AI guys program a machine that defeated a world chess champion way back in 1997?

And didn't you AI guys build a computer system that kicked the respective asses of the two best Jeopardy! champions of all time?

What happened to 'HAL 9000 by 2001'? Where are our Series 800 Terminator robots?

You call this progress?!!

Sorry we annoyed you with our awesomeness. **CAT**
Daily Joke – Behind the Scenes

• How powerful is a computer nowadays?
  – A computer can perform billions of arithmetical operations in every second
  – You’ve written a program computing π. In a millisecond, it computes so many digits on which many mathematicians spent their whole lives in 19th Century

• How smart is a computer nowadays?
  – You know, it’s still hard for it to recognize a cat
  – If you know how to do it accurately, you can definitely become a professor in our department
You Wonder Why?

- Computers do things in a deterministic way!
  - The algorithms have to be precise and deterministic
    - Computer scientists don’t have tools significantly better than Java
  - When you see a cat, you know it is a cat, but you don’t know the rule for telling that truth
    - That’s your instinct, which is hard to be logical
  - Computer excels at board games because these games can be deterministic
    - You will write the logic behind it, and make people feel it smart
    - But still, it has only logic
Miscellaneous

• Initialization of instance variables
• Evaluation of boolean expressions
• **break** statement (and **return** statement)
• Random number generator
Initialization of Instance Variables

• In Lab 4, initialization was required
  – Many of you ignored this requirement
  – Those who did it didn’t do it right
Initialization of Instance Variables

- You can declare default values for instance variables

```java
public class Rectangle {
    public int width = 1;
    public int height = 1;
    public int area = 1;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

Rectangle box = new Rectangle();
System.out.println(box.getArea());

// Output: 1
What’s the Point of Initialization

• When people call your methods, they won’t get an error
  – Because from outside of your class, they can not see implementation details
  – You must guarantee that every object in your class works, starting from it’s created
public class Statistics {

    private int Goals_Made, Free_Throws, Three_Pointers, Goal_Attempts, Free_ThrowAttempt, Three_Attempts;

    public void setFieldGoalsMade(int FG_Made) {
        Goals_Made = FG_Made;
    }

    public double getFieldGoalPercent() {
        return (((double) Goals_Made) / ((double) Goal_Attempts)) * 100;
    }

}
public static void main(String[] args) {
    Statistics unc = new Statistics();
    DecimalFormat df = new DecimalFormat("0.00");
    unc.setFieldGoalsMade(1);
    unc.setFieldGoalAttempts(2);

    int points = (int) unc.getTotalPoints();
    double field = unc.getFieldGoalPercent();
    double free = unc.getFreeThrowPercent();
    double three = unc.get3PointPercent();
    System.out.print("UNC has scored " + points + " points\n"
        + "UNC has a field-goal percentage of " + df.format(field)
        + "%\n" + "UNC has a free-throw percentage of "
        + df.format(free) + "%\n"
        + "UNC has a 3-point field-goal percentage of "
        + df.format(three) + "%\n");
}
If UNCStats Call It This Way

• In a game with only two field attempts, what is the free throw and three pointer percentage?
  – It should be 0.0% (or 100.0% if you want to)

• The uninitialized version outputs:

  Exception in thread "main" java.lang.ArithmeticException: / by zero
  at Statistics.getFreeThrowPercent(Statistics.java:87)
  at UNCStats2.main(UNCStats2.java:15)

  – Divide-by-zero run-time error
If UNCStats Call It This Way

- Obviously, the problem is you are calculating 0/0
- For both integer and floating-point instance variables, the initial value is always 0
  - For boolean variables, it is always false
  - Actually, it makes no difference if you initialize everything as 0
  - You have to initialize variables so that there is no error
Correct Initialization

```java
public class Statistics {
    private int Goals_Made = 0, Free_Throws = 0, Three_Pointers = 0,
                       Goal_Attempts = 0, Free_ThrowAttemp = 0, Three_Attempts = 0;
    private double Goal_Percent = 0, Free_Percent = 0, Three_Percent = 0;
    // The methods will be in charge of checking values
}
```

```java
public class Statistics {
    private int Goals_Made = 0, Free_Throws = 0, Three_Pointers = 0,
                       Goal_Attempts = 1, Free_ThrowAttemp = 1, Three_Attempts = 1;
    // Only works when you don’t have
    // methods like makeAShot() and missAShot()
}
```
Key Point of Initialization

- You class should work as long as an object is created
  - You should try your best to cover all cases
  - If it can’t be done, we will learn “constructor method” in next week, which pushes the responsibility to the user
Evaluation of Boolean Expressions

• Logical operators
  – &&: be false if **ONE** expression is false
  – ||: be true if **ONE** expression is true

• Java doesn’t evaluate all subexpressions if the result is known
  – a && b && c && d
    • The evaluation stops when one subexpression is **false**
  – a || b || c || d
    • The evaluation stops when one subexpression is **true**
Evaluation of Boolean Expressions

- The following code will have a run-time error

```java
if (3 == 3 && 3 / 0 == 1) {
    System.out.println("Something");
}
```

- The following code will print “Something”

```java
if (3 == 3 || 3 / 0 == 1) {
    System.out.println("Something");
}
```
Why is This Useful?

• In certain circumstances, we can make things short

```java
if (i >= 1) {
    if (num[i - 1] > currentValue) {
        num[i - 1] = num[i];
    }
}
```

```java
if (i >= 1 && num[i - 1] > currentValue) {
    num[i - 1] = num[i];
}
```

– The second version won’t have an out-of-bound problem
Break Statement

• We saw it in switch statement
• It can also be used in loops
  – The syntax is very simple
    • `break;`
  – It means: to jump out of current loop
• It is used when
  – You don’t want to execute the remaining loop
  – You must stop executing the remaining loop
public boolean equalStrings(String a, String b) {
    boolean result = true;
    if (a.length() != b.length()) {
        result = false;
    } else {
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != b.charAt(i)) {
                result = false;
                break; // jump out of the loop immediately
            }
        }
    }
    return result;
}
Break Statement

• You can only jump out one loop
  – There is no way to jump out a nested loop using break

```
System.out.println("All possible dice combinations no greater than 8 are:");
for (int i = 1; i <= 6; i++) {
    for (int j = 1; j <= 6; j++) {
        if (i + j >= 8)
            break;
        System.out.print("(" + i + "," + j + "), ");
    }
    System.out.println();
}
```
Break Statement

• You can run the code by yourself
• The results are:

All possible dice combinations no greater than 8 are:
(1,1), (1,2), (1,3), (1,4), (1,5), (1,6),
(2,1), (2,2), (2,3), (2,4), (2,5),
(3,1), (3,2), (3,3), (3,4),
(4,1), (4,2), (4,3),
(5,1), (5,2),
(6,1),
Jump Out of All Loops

• The only way is to use `return` statement in a method
• An example question:
  – Given an array, does it include 3 numbers that add up to 0?
  – For example, if the array is
    • `int[] nums = { 3, 2, 4, 9, -3, -3, -2, -11 };`
    • You should output: 2+9+-11=0
  – If the array is
    • `int[] nums = { 3, 2, 4, 9, -3, -3, -2, -10 };`
    • You should output: No such three numbers
public static void main(String[] args) {
    int[] nums = {3, 2, 4, 9, -3, -3, -2, -10};
    threeSumZero(nums);
}

private static void threeSumZero(int[] num) {
    for (int i = 0; i < num.length; i++)
        for (int j = 0; j < num.length; j++)
            for (int k = 0; k < num.length; k++)
                if (i != j && i != k && j != k
                    && num[i] + num[j] + num[k] == 0) {
                    System.out.println(num[i] + " + "+ num[j] + " + " + num[k] + "=0");
                    return; // JUMP OUT OF EVERYTHING
                }
    System.out.println("No such three numbers");
}
Random Number Generator

• It is very similar to Scanner

```java
int N = 10, print = 5;
Random generator = new Random();
for (int i = 0; i < print; i++) {
    int randomNum = generator.nextInt(N);
    System.out.println(randomNum);
}
```

– If you run the code many times, the output varies
  • 7,6,6,1,9/5,6,8,8,5/5,6,7,3,9/6,7,4,0,2

– `nextInt(N)` generates integers in [0,N)
  • Like arrays, 0 is included but N is not
  • You can get 0 but not 10 in this program
Random Number Generator

• Like scanners, using one generator is sufficient
• You can use it in many ways
  – Create a random size array with random increasing elements
    • The size can be from 10 to 19
    • The values starts from 0 to 100
  – Sample outputs:
    • The array has a size 17. The elements are:
      – 3, 7, 8, 11, 15, 16, 21, 24, 26, 30, 33, 38, 41, 42, 45, 46, 51.
    • The array has a size 15. The elements are:
      – 2, 3, 7, 8, 10, 12, 18, 21, 24, 30, 35, 40, 46, 47, 50.
Random Number Generator

```
int lowSize = 10, highSize = 20, valueRange = 100;
Random generator = new Random();
int size = lowSize + generator.nextInt(highSize - lowSize);
System.out.println("The array has a size " + size + ". The elements are:");
int[] array = new int[size];
array[0] = generator.nextInt(valueRange / size);
for (int i = 1; i < size; i++) {
    array[i] = array[i - 1] + generator.nextInt(valueRange / size) + 1;
    System.out.print(array[i - 1] + ",");
}
System.out.println(array[size - 1] + ".");
```
Avoid “Magic Numbers”

• Magic numbers means
  – It is there. It works. But you don’t know what it is.

```java
int size = 10 + generator.nextInt(10);
```

• Use variables instead

```java
int lowSize = 10, highSize = 20, valueRange = 100;
int size = lowSize + generator.nextInt(highSize - lowSize);
```

  – If you want to change it, you know what to do
Back to Arrays

- `int[] scores = new int[5];`

- This is like declaring 5 strangely named variables of type `int`:
  - scores[0]
  - scores[1]
  - scores[2]
  - scores[3]
  - scores[4]

- Especially, you can use `score[i]` to locate a single one
Review: Array and Index

- Index numbers start with 0. They do NOT start with 1 or any other number.
- The array name represents a memory address, and the $i^{th}$ element can be accessed by the address plus $i$.
Review: Creating an Array

• Create an array with given length saved in constants
  – `public static final int NUMBER_OF_READINGS = 100;`
  – `int[] pressure = new int[NUMBER_OF_READINGS];`
• Create an array with user input length
  – `System.out.println("How many scores?");`
  – `int numScores = keyboard.nextInt();`
  – `int[] scores = new int[numScores];`
Review: Don’t be OUT OF BOUNDS!

• Indices MUST be in bounds
  – `double[] entries = new double[5]; // from [0] to [4]`
  – `entries[5] = 3.7; // ERROR! Index out of bounds`

• Your code will compile if you are using an index that is out of bounds, but it will give you a run-time error!
Arrays as Instance Variables

• Quite straight forward

```java
public class Weather {
    private double[] temperature;
    private double[] pressure;

    public void initializeTemperature(int len) {
        temperature = new double[len];
    }
}
```
Arrays of Objects

• When you create an array of objects like this:
  `Student[] students = new Student[35];`
• Each of the elements of `students` is not yet an object
• You have to instantiate each individual one
  `students[0] = new Student();`
  `students[1] = new Student();`
• ...or do this in a loop
Arrays of Objects

Smiley[] smilies = new Smiley[3];
for (int i = 0; i < smilies.length; i++) {
    smilies[i] = new Smiley();
}
Arrays as Parameters

public void changeArray(int[] arr) {
   int len = arr.length;
   arr[len - 1] = 25;
}

changeArray(num)
Arrays as Return Types

- Create an array and return it

```java
public double[] buildArray(int len) {
    double[] retArray = new double[len];
    for (int i = 0; i < retArray.length; i++) {
        retArray[i] = i * 1.5;
    }
    return retArray;
}
```
Indexed Variables as Arguments

• The same as a regular variable

```java
public void printNum(int num) {
    System.out.println(num);
}

public void doStuff() {
    int[] scores = { 15, 37, 95 };
    for (int index = 0; index < scores.length; index++) {
        printNum(index);
        printNum(scores[index]);
    }
}
```
2D Arrays

- Arrays having more than one index are often useful
  - Tables
  - Grids
  - Board games

<table>
<thead>
<tr>
<th></th>
<th>0: Open</th>
<th>1: High</th>
<th>2: Low</th>
<th>3: Close</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0: Apple Inc.</strong></td>
<td>99.24</td>
<td>99.85</td>
<td>95.72</td>
<td>98.24</td>
</tr>
<tr>
<td><strong>1: Walt Disney Co.</strong></td>
<td>21.55</td>
<td>24.20</td>
<td>21.41</td>
<td>23.36</td>
</tr>
<tr>
<td><strong>2: Google Inc.</strong></td>
<td>333.12</td>
<td>341.15</td>
<td>325.33</td>
<td>331.14</td>
</tr>
<tr>
<td><strong>3: Microsoft Corp.</strong></td>
<td>21.32</td>
<td>21.54</td>
<td>21.00</td>
<td>21.50</td>
</tr>
</tbody>
</table>
Declaring and Creating 2D Arrays

- Two pairs of square brackets means 2D
  - `int[][] table = new int[3][4];`
- or
  - `int[][] table;`
  - `table = new int[3][4];`
Declaring and Creating 2D Arrays

- Array (or 1D array) gives you a list of variables
- 2D array gives you a table of variables
  - `int[][] table = new int[3][4];`

<table>
<thead>
<tr>
<th>table[0][0]</th>
<th>table[0][1]</th>
<th>table[0][2]</th>
<th>table[0][3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>table[1][0]</td>
<td>table[1][1]</td>
<td>table[1][2]</td>
<td>table[1][3]</td>
</tr>
</tbody>
</table>
Using a 2D Array

• We use a loop to access 1D arrays

```java
for (int i = 0; i < 5; i++) {
    scores[i] = keyboard.nextInt();
    scoreSum += scores[i];
}
```
Using a 2D Array

• We use nested loops for 2D arrays

```java
int[][] table = new int[4][3];
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 3; j++) {
        table[i][j] = i * 3 + j;
        System.out.println(table[i][j]);
    }
}
```
Multidimensional Arrays

• You can have more than two dimensions
  – `int[][][]` cube = new `int[4][3][4];`

• Use more nested loops to access all elements
  – for (int i...)
    • for (int j...)
      – for (int k...)

Announcement

• Those who haven’t done the make-up assignment, today is the deadline
  – InsertionSort.java will be online tomorrow