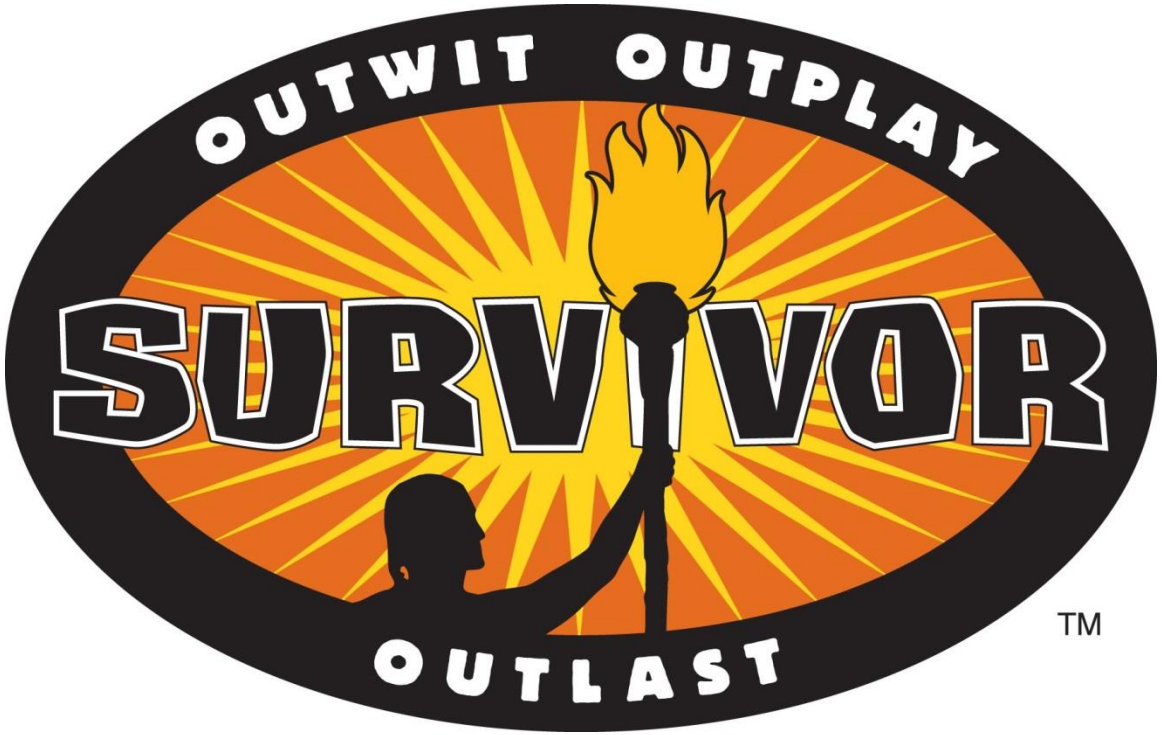
Survivor: CSCI 135



Arrays to the rescue!

- We've already seen arrays:

```
public static void main(String [] args)
```

```
% java CostCalc bananas 12 0.21
To buy 12 bananas you will need $2.52
```

identifier	meaning	value	type
`args[0]`	1st thing on command line after Java class name	`"bananas"`	`String`
`args[1]`	2nd thing on command line	`"12"`	`String`
`args[2]`	3rd thing on command line after Java class	`"0.21"`	`String`
`args.length`	# of things on command line	`3`	`int`

Arrays: creating many things

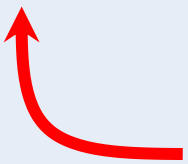
- Arrays: create many variables of same type
- Goal: Ten variables of same type
  - e.g. To hold the values 0-9

```
int a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0;
a1 = 1;
a2 = 2;
a3 = 3;
a4 = 4;
a5 = 5;
a6 = 6;
a7 = 7;
a8 = 8;
a9 = 9;
```

Arrays: creating many things

- Arrays: create many variables of same type
- Goal: Ten variables of same type
  - e.g. To hold the values 0-9

```
int [] a = new int[10];
a[0] = 0;
a[1] = 1;
a[2] = 2;
a[3] = 3;
a[4] = 4;
a[5] = 5;
a[6] = 6;
a[7] = 7;
a[8] = 8;
a[9] = 9;
```

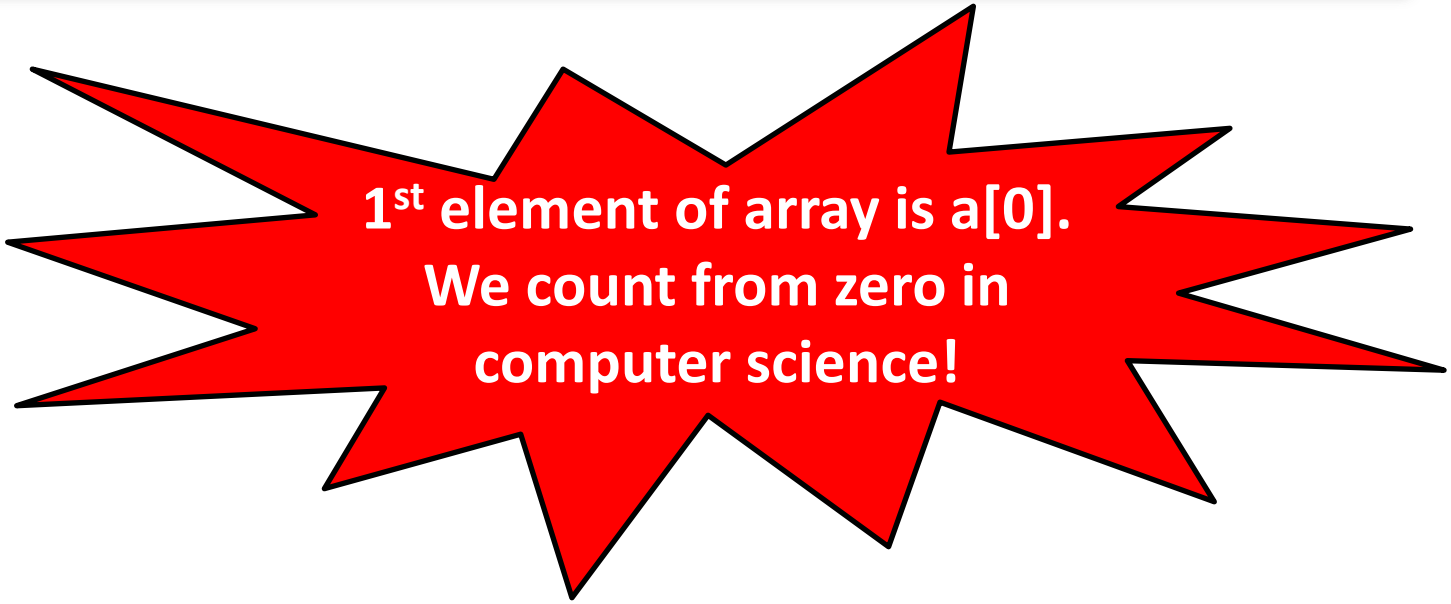


new keyword is used whenever we create an array

Arrays: accessing elements

- Arrays: we can use a variable as the index!
  - Makes code shorter, cleaner, less buggy

```
int N = 10;                              // size of array
int [] a;                                // declare array
a = new int[N];                          // create array

for (int i = 0; i < a.length; i++)       // initialize array elements
   a[i] = i;                             // to be 0 - 9
```

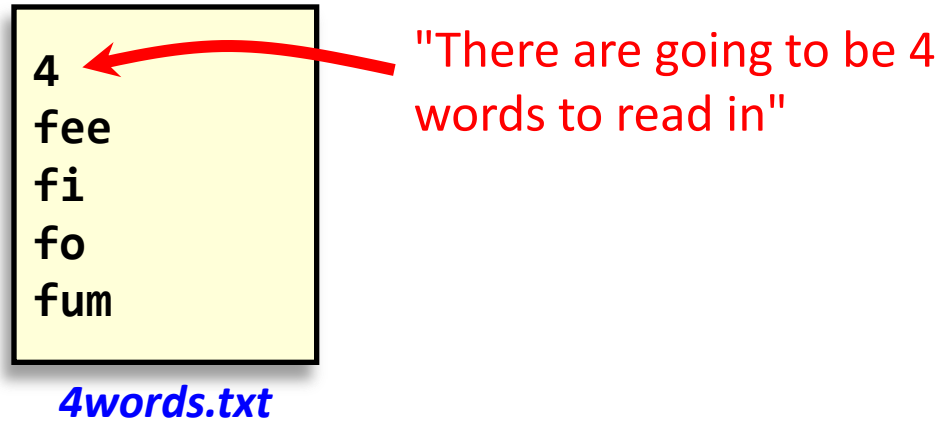


Arrays: easy to extend

- Arrays: can hold lots and lots of data
  - Same code, but now holds 100,000 integers:

```
int N = 100000;                          // size of array
int [] a;                                // declare array
a = new int[N];                          // create array

for (int i = 0; i < a.length; i++)       // initialize array elements
   a[i] = i;                             // to be 0 - 9
```

Arrays: loading data from file



- Read words into array
- Print out words in reverse order

```
% java Backwards < 4words.txt
fum fo fi fee
```

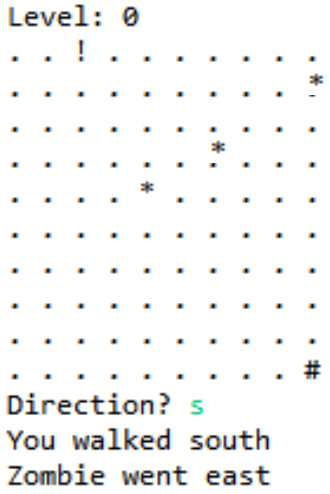
Arrays: loading data from file

```
4
fee
fi
fo
fum
```

```
% java PrintBackward < 4words.txt
fum fo fi fee
```

```
public class Backwards
{
   public static void main(String [] args)
   {
      int num = StdIn.readInt();
      String [] words = new String[num];

      for (int i = 0; i < num; i++)
         words[i] = StdIn.readString();

      for (int i = num - 1; i >= 0; i--)
         System.out.print(words[i] + " ");
      System.out.println();
   }
}
```

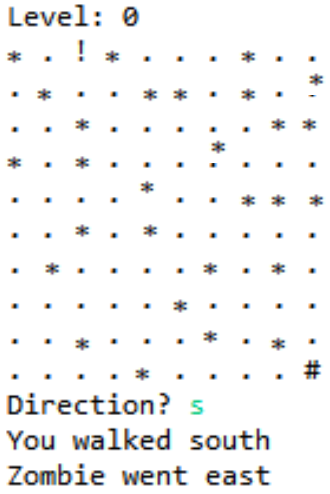
Super Extreme Zombie Apocalypse



What if we need to keep track of three zombies?

```
int personX = 0;
int personY = 0;
final int NUM_ZOMBIES = 3;  // constant defining # of zombies

int [] zombieX = new int[NUM_ZOMBIES]; // declare & create x-pos array
int [] zombieY = new int[NUM_ZOMBIES]; // declare & create y-pos array

// Set random initial location for each zombie (they can overlap)
for (int i = 0; i < NUM_ZOMBIES; i++)
{
   zombieX[i] = (int) (Math.random() * 10); // set i-th zombie's x-pos
   zombieY[i] = (int) (Math.random() * 10); // set i-th zombie's y-pos
}

...

int i = 0;
while ((i < zombieX.length) && (!gameOver))
{
   if ((personX == zombieX[i]) &&
       (personY == zombieY[i]))
   {
       System.out.println("Zombie got your braaaains!");
       gameOver = true;
   }
   i++;
}
```

Super Mega Extreme Zombie Apocalypse



What if we need to keep track of thirty zombies?

```
int personX = 0;
int personY = 0;
final int NUM_ZOMBIES = 30;  // constant defining # of zombies

int [] zombieX = new int[NUM_ZOMBIES]; // declare & create x-pos array
int [] zombieY = new int[NUM_ZOMBIES]; // declare & create y-pos array

// Set random initial location for each zombie (they can overlap)
for (int i = 0; i < NUM_ZOMBIES; i++)
{
   zombieX[i] = (int) (Math.random() * 10); // set i-th zombie's x-pos
   zombieY[i] = (int) (Math.random() * 10); // set i-th zombie's y-pos
}

...

int i = 0;
while ((i < zombieX.length) && (!gameOver))
{
   if ((personX == zombieX[i]) &&
       (personY == zombieY[i]))
   {
       System.out.println("Zombie got your braaaains!");
       gameOver = true;
   }
   i++;
}
```

Arrays revisited

- Arrays
  - Store a bunch of values under one name
  - Declare and create in one line:

```
int N = 8;
int [] x = new int[10];
double [] speeds = new double[100];
String [] names = new String[N];
```

  - To get at values, use name and index between []:

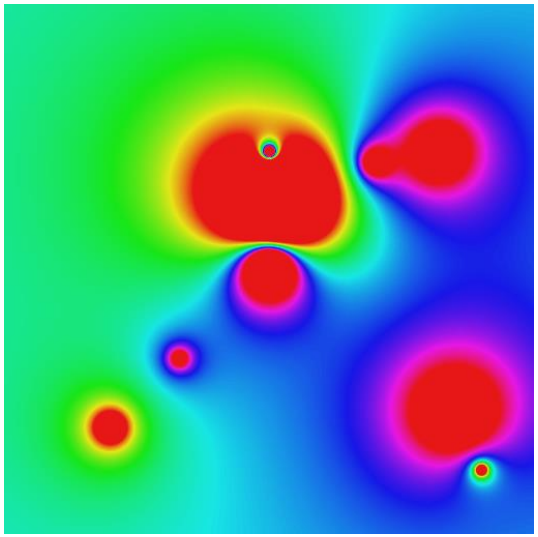
```
int sumFirst2 = x[0] + x[1];
speeds[99] = speeds[98] * 1.1;
System.out.println(names[0]);
```

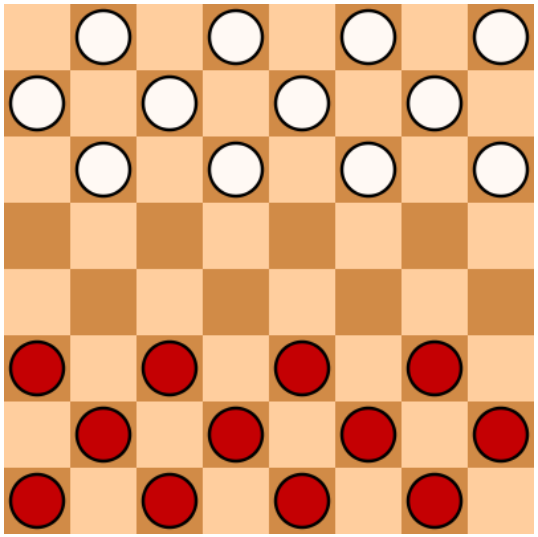
  - Array indexes start at 0!

Two dimensional array examples

- Two dimensional arrays
  - Tables of hourly temps for last week
  - Table of colors for each pixel of a 2D image
  - Table storing piece at each position on a checkerboard

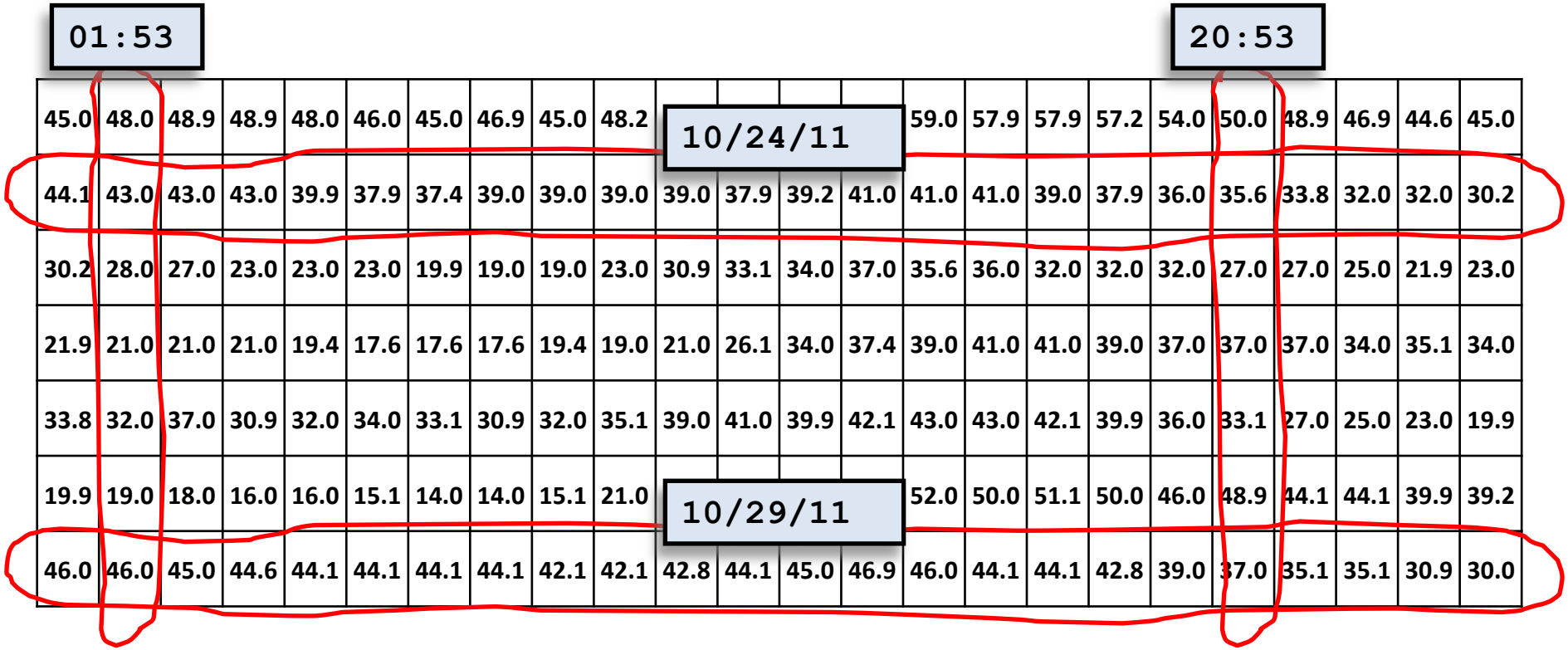
0h	1h	…	23h
32.5	30.0		45.6
…			
59.5	62.1	…	60.0
60.7	61.8	…	70.5
62.6	62.0	…	68.0





Weather data

- Goal: Read in hourly temp data for last week
  - Each row is a day of the week
  - Each column is a particular hour of the day



Two dimensional arrays

- Declaring and creating
  - Like 1D, but another pair of brackets:

```
final int DAYS  = 7;
final int HOURS = 24;
double [][] a = new double[DAYS][HOURS];
```

- Accessing elements
  - To specify element at the i^{th} row and j^{th} column:

```
a[i][j]
```

a[0][0]	a[0][1]	a[0][2]	...	a[0][22]	a[0][23]
a[1][0]	a[1][1]	a[1][2]	…	a[1][22]	a[1][23]
…	…	…	…	….	…
a[6][0]	a[6][1]	a[6][2]	…	a[6][22]	a[6][23]

Temperature on second day of data, last hour of day

Reading temperature data

- Initialize all elements of our 2D array
  - Nested loop reading in each value from `StdIn`
  - Find weekly max and min temp

```
final int DAYS  = 7;
final int HOURS = 24;
double [][] a = new double[DAYS][HOURS];
double min = Double.POSTIVE_INFINITY;
double max = Double.NEGATIVE_INFINITY;

for (int row = 0; row < DAYS; row++)
{
   for (int col = 0; col < HOURS; col++)
   {
      a[row][col] = StdIn.readDouble();
      min = Math.min(min, a[row][col]);
      max = Math.max(max, a[row][col]);
   }
}
System.out.println("min = " + min + ", max = " + max);
```

Start the min at a really high temp.

Start the max at a really low temp.

The new min temp is either the current min or the new data point.

Arrays

You have been given a grade file to read in and calculate two things: the letter grade for each student, and the average score on each exam. The first line in the text file contains the number of students in the file. Then each subsequent line contains the first and last name of a student and that student's scores on exams 1, 2, and 3.

Your program should read the information from the file and print out the letter grade for each student and then the average score for the whole "class". You can assume 90-100% is an A, 80-89% is a B, 70-79% is a C, 60-69% is a D, and the rest an F.

For example, if I ran my program it might look like this:

```
>>java mvandyne5 < gradebook.txt
Santa Claus B
Easter Bunny D
Tooth Fairy B
Bugs Bunny A
Elmer Fudd D

Exam 1 Average: 72.2
Exam 2 Average: 81.6
Exam 3 Average: 81.0
```

VERY IMPORTANT: Name your program <yourusername>5.java

Submit your program to Survivor III on Moodle