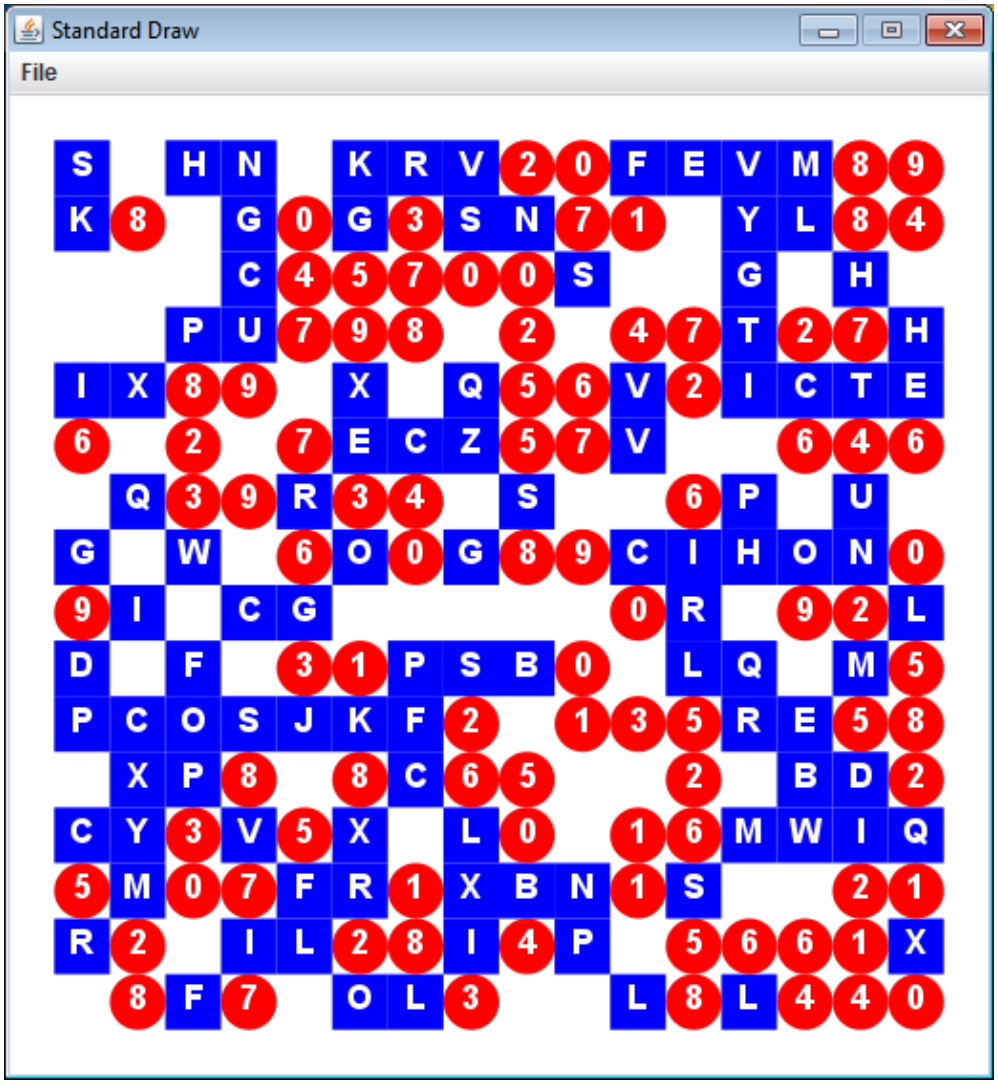
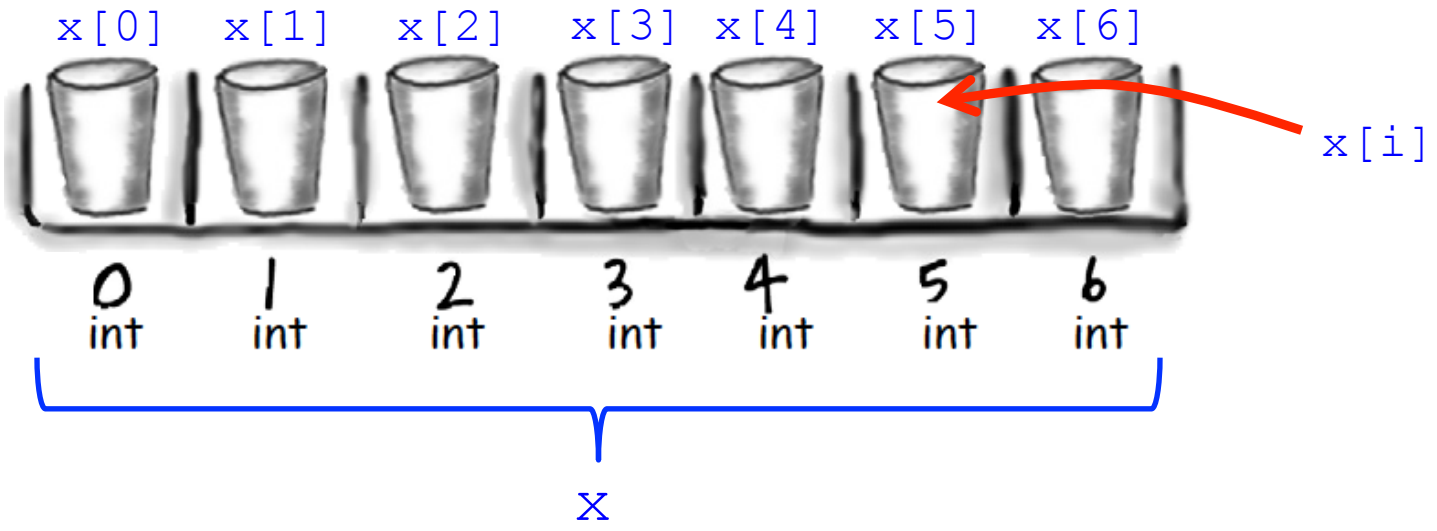
Multidimensional arrays, Object hierarchies



Overview

- Multidimensional arrays
  - 2D arrays, a grid of variables
- Object hierarchy
  - Several classes inheriting from same base class
  - Concrete versus abstract classes

Single dimensional arrays

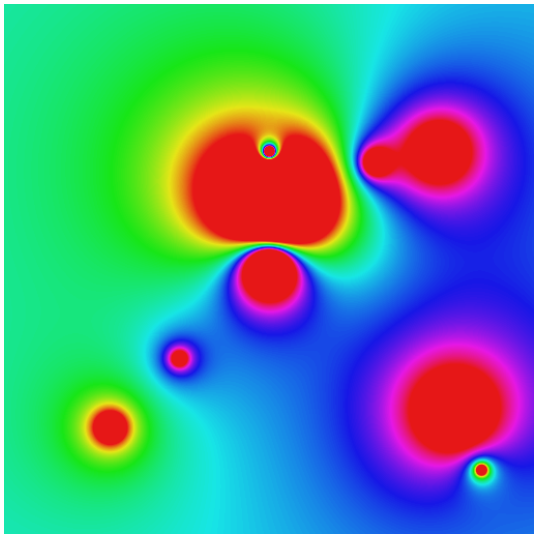


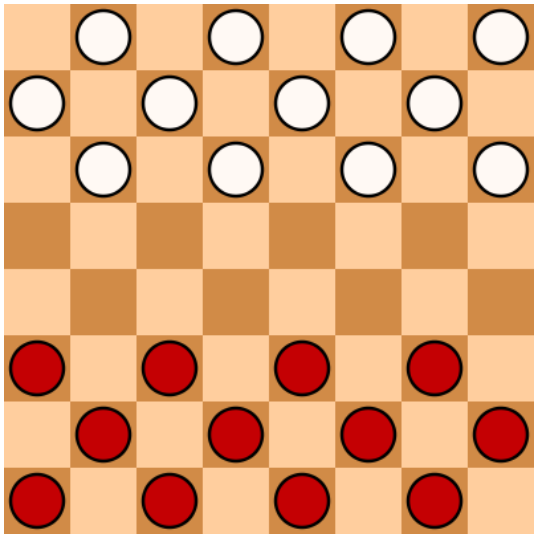
- Declare and create: `int [] x = new int[7];`
- Check length: `x.length`
- Obtain a value: `x[0], x[1], ..., x[6]`

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

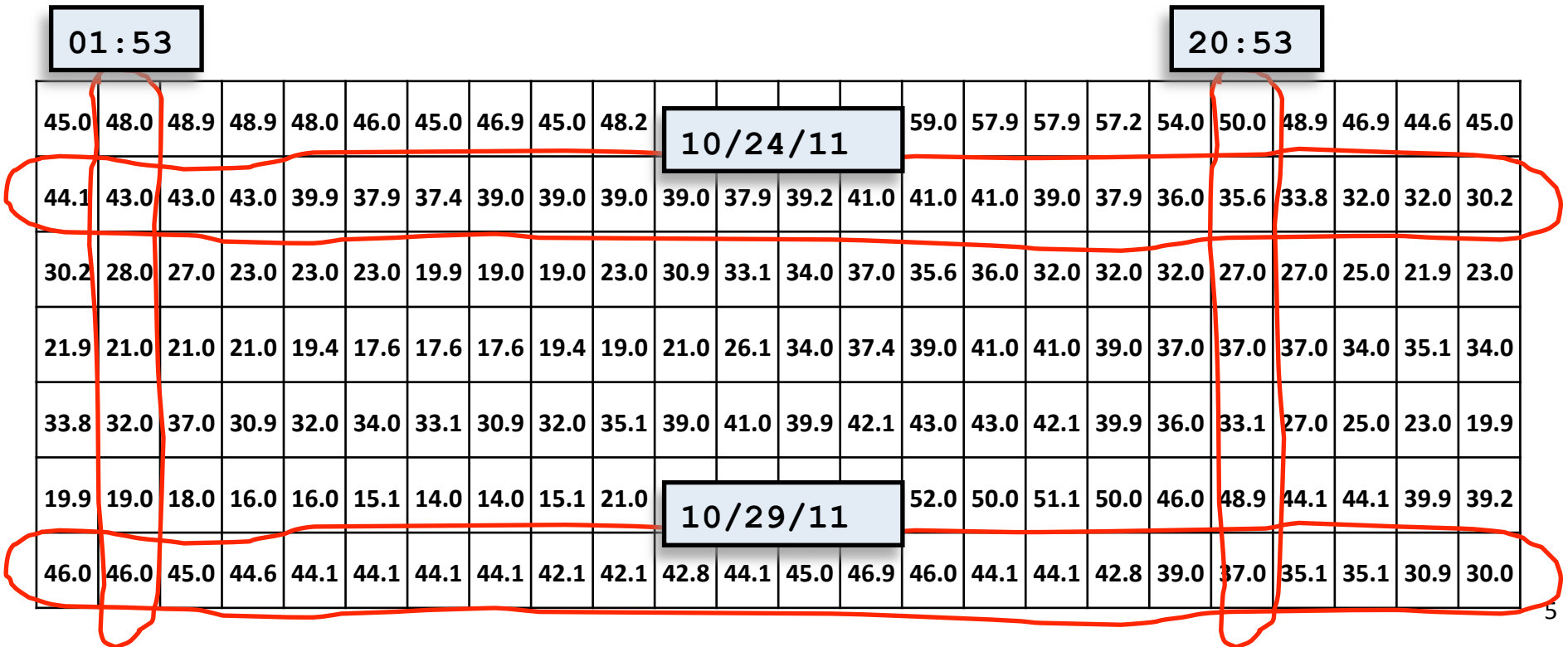
0h	1h	…	23h
32.5	30.0		45.6
…			
59.5	62.1	…	60.0
60.7	61.8	…	70.5
62.6	62.0	…	68.0





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

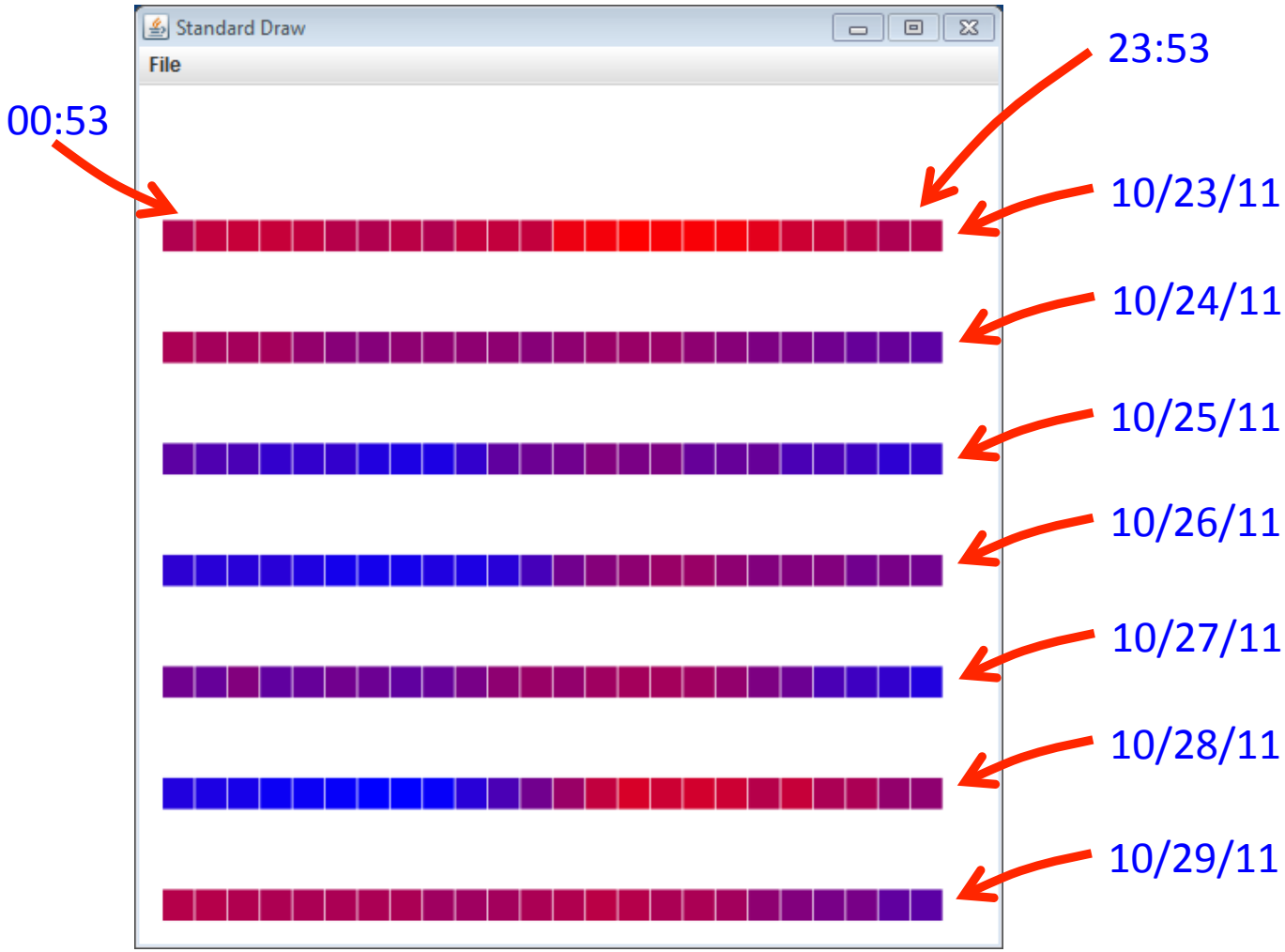
Reading temperature data

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

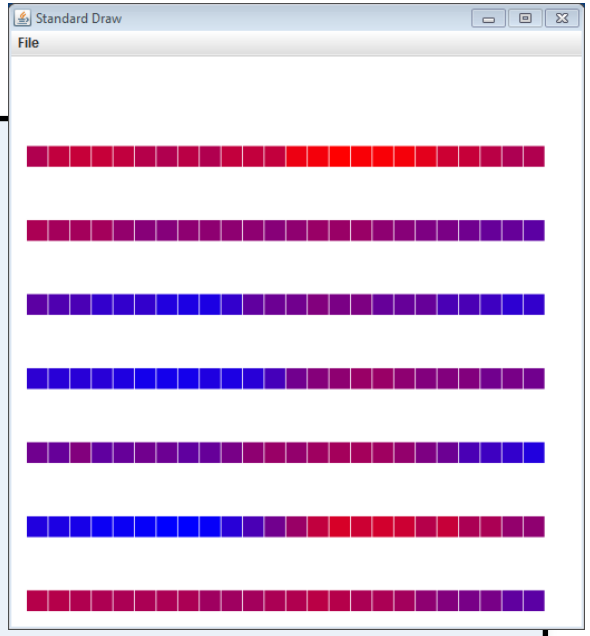
```
final int DAYS  = 7;
final int HOURS = 24;
double [][] a = new double[DAYS][HOURS];
double min = Double.MAX_VALUE;
double max = Double.MIN_VALUE;

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);
```

Weather visualization

- Goal: Heatmap of temps over the last week



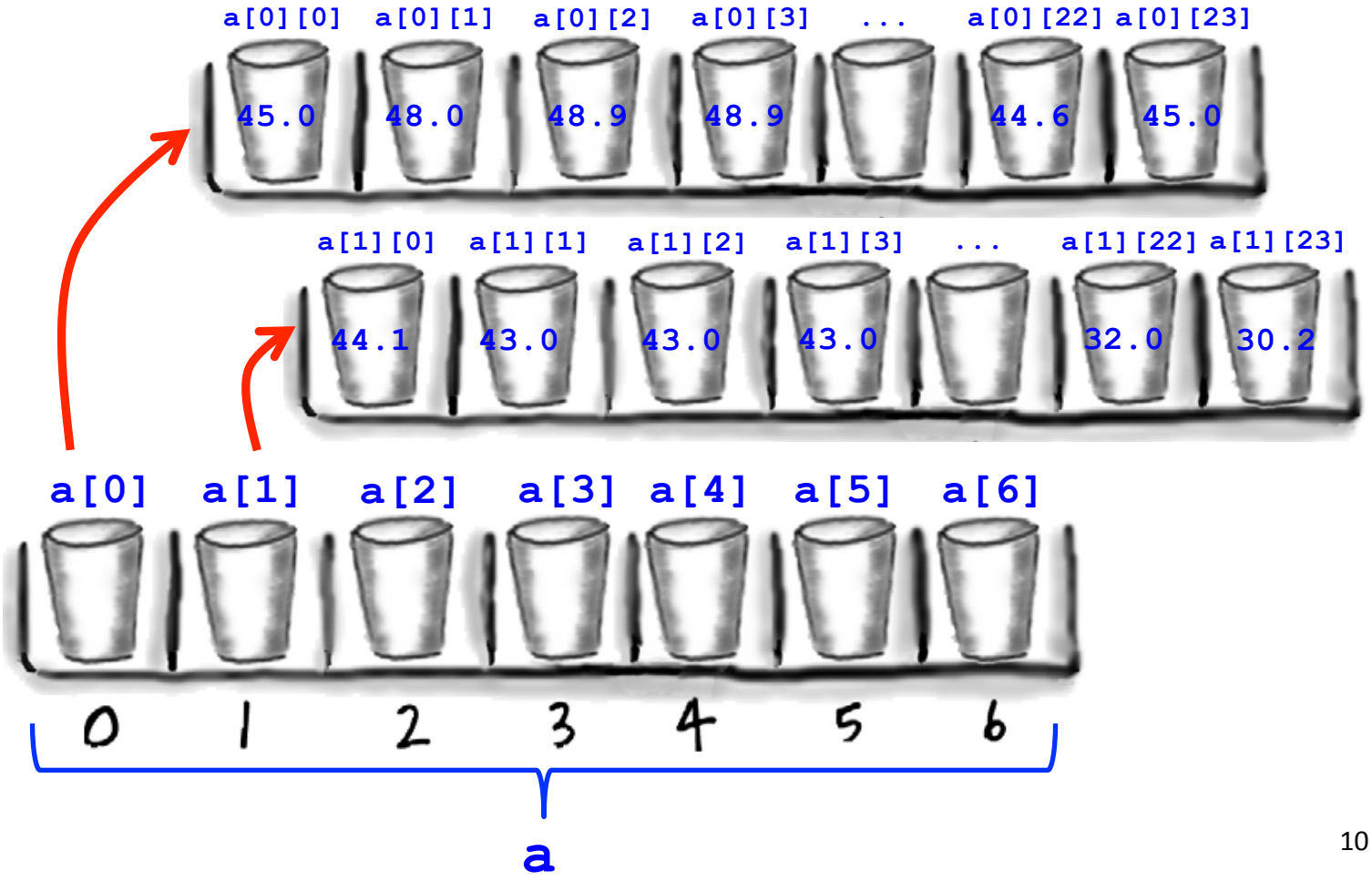
Weather visualization




```
...
System.out.println("min = " + min + ", max =

final double BOX_SIZE = 0.02;
double range = max - min;

for (int row = 0; row < DAYS; row++)
{
   for (int col = 0; col < HOURS; col++)
   {
      float proportion = (float) ((a[row][col] - min) / range);
      Color c = new Color(proportion, 0.0f, 1.0f - proportion);
      StdDraw.setPenColor(c);
      StdDraw.filledRectangle(1.0 / HOURS * col,
                              1.0 / DAYS * (DAYS - row - 1),
                              BOX_SIZE,
                              BOX_SIZE);
   }
}
```

Array of arrays

- An array element can be an array
  - 2D array is really an array of arrays



Lengths in a 2D array

- How many rows? `a.length`
- How many columns in a row? `a[row].length`

Average temperature?

- Goal: Static method to compute average of numbers in a 2D array
- Problem: Not sure of array dimensions

```
public static double average(double [][] nums)
{
   double sum = 0.0;
   int count = 0;
   for (int row = 0; row < ??????; row++)
   {
      for (int col = 0; col < ??????; col++)
      {
         sum += nums[???][???];
         count++;
      }
   }
   return ??????;
}
```

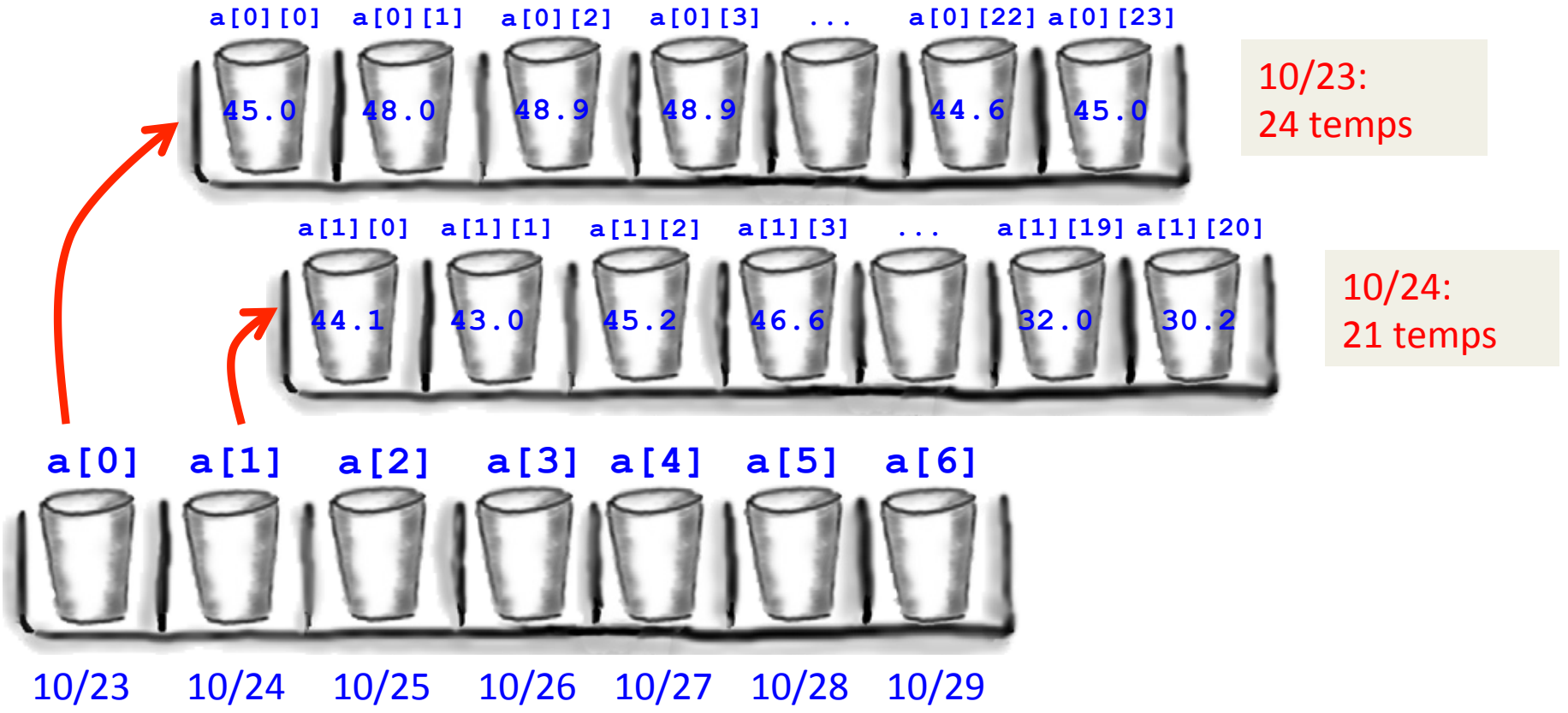
Average temperature?

- Goal: Static method to compute average of numbers in a 2D array
- Problem: Not sure of array dimensions

```
public static double average(double [][] nums)
{
    double sum = 0.0;
    int count = 0;
    for (int row = 0; row < nums.length; row++)
    {
        for (int col = 0; col < nums[row].length; col++)
        {
            sum += nums[row][col];
            count++;
        }
    }
    return sum / count;
}
```

Ragged arrays

- Problem: Not all days have 24 temp measurements
- Solution: Each day's row has variable # of temps



Ragged arrays

- Ragged arrays
  - Create master array to hold "rows"
  - Create array for each row separately

```
// An array with three rows
double [][] a = new double[3][];

// First row can have 2 numbers
a[0] = new double[2];

// Second row can have 3 numbers
a[1] = new double[3];

// Third row can have 4 numbers
a[2] = new double[4];
```

a[0][0]	a[0][1]		
a[1][0]	a[1][1]	a[1][2]	
a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
a.length    = 3
a[0].length = 2
a[1].length = 3
a[2].length = 4
```

Reading ragged data

```
final int DAYS  = 7;
final int HOURS = 24;
double min = Double.MAX_VALUE;
double max = Double.MIN_VALUE;

double [][] a = new double[DAYS][];

for (int row = 0; row < DAYS; row++)
{
   String line = StdIn.readLine();
   String [] temps = line.split("\\s+");

   a[row] = new double[temps.length];

   for (int col = 0; col < temps.length; col++)
   {
      a[row][col] = Double.parseDouble(temps[col]);
      min = Math.min(min, a[row][col]);
      max = Math.max(max, a[row][col]);
   }
}
```

Allocate space for 7 days of temp data

Read line of data and parse into String array

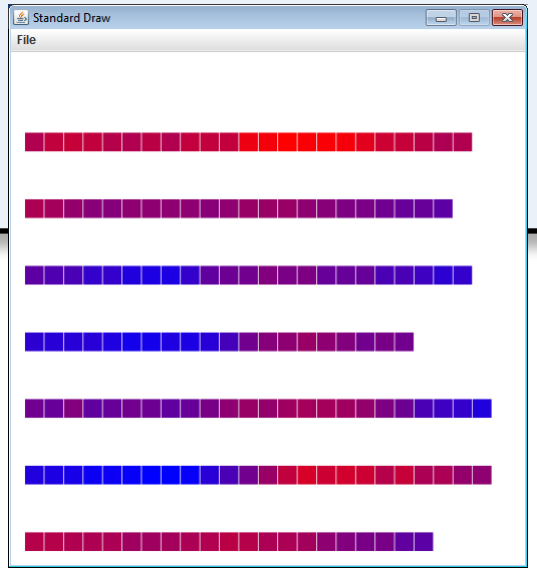
Create array for this day's data

Convert element of String array and put into our 2D array

Drawing ragged data

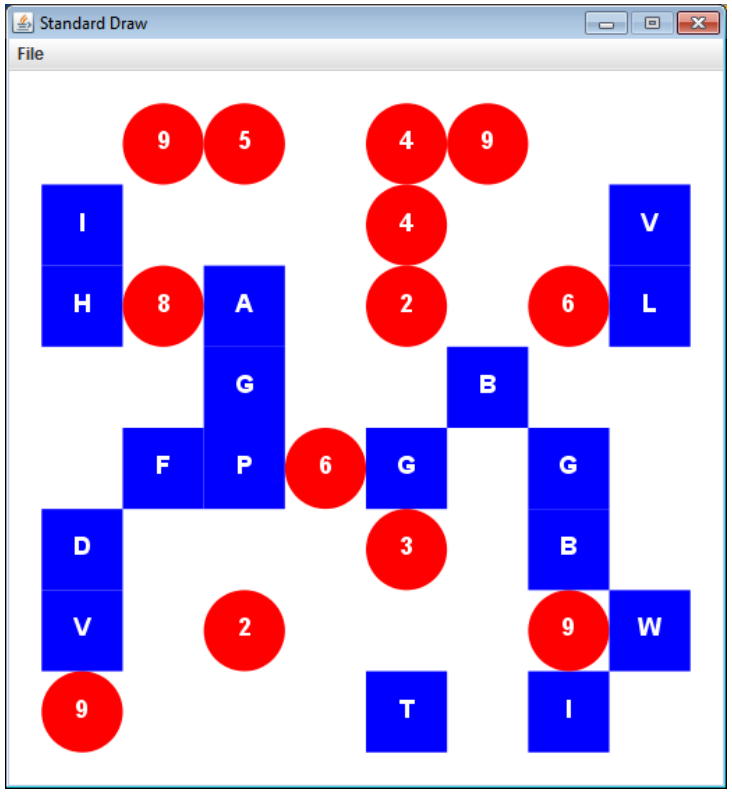
Changed from HOURS, now works with fewer than 24 observations

```
for (int row = 0; row < DAYS; row++)
{
   for (int col = 0; col < a[row].length; col++)
   {
      float proportion = (float) ((a[row][col] - min) / range);
      Color c = new Color(proportion, 0.0f, 1.0f - proportion);
      StdDraw.setPenColor(c);
      StdDraw.filledRectangle(1.0 / HOURS * col,
                              1.0 / DAYS * (DAYS - row - 1),
                              BOX_SIZE,
                              BOX_SIZE);
   }
}
```



A tile game

- Goal: Design classes for use in a tile game
  - Played on a square N x N grid
  - Each grid location can have one thing:
    - Square letter tile
    - Circular number tile
  - Some other rules TBD

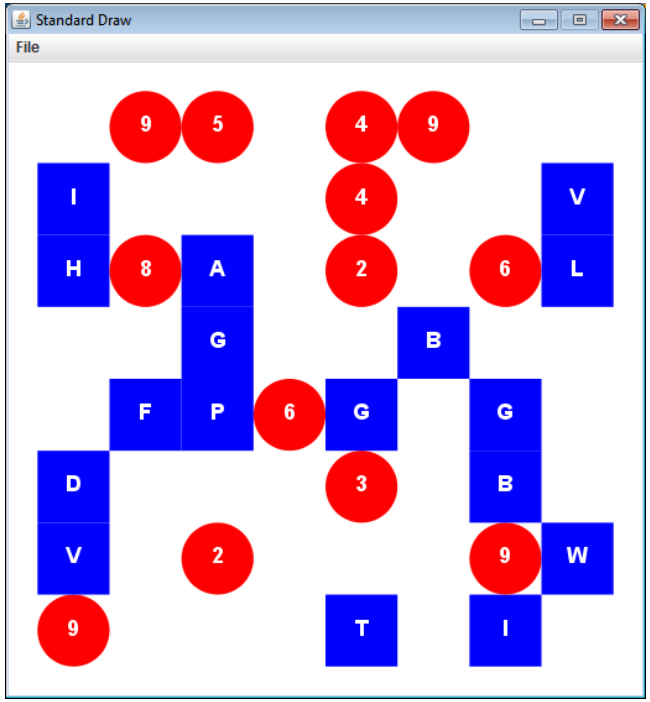


Designing the Tile game

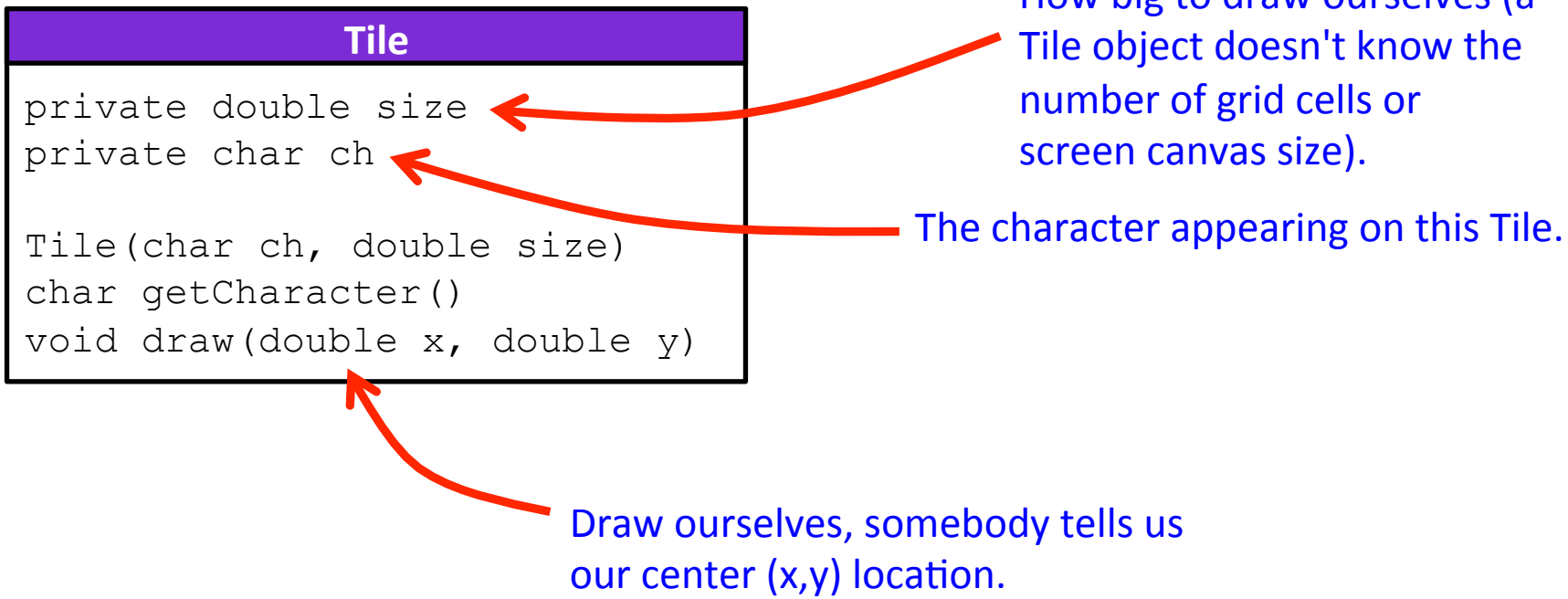
- Use a 2D array for the N x N grid
  - Array element `null` if no tile there
  - Otherwise reference to letter/number tile object
  - Start by randomly placing non-overlapping tiles

```
final int GRID  = 8;
Tile [][] tiles = new Tile[GRID][GRID];
```

null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null
null	null	null	null	null	null	null	null



Single class design?

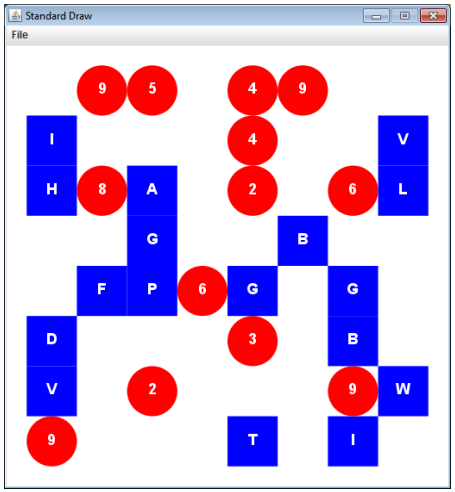


- Problem: draw() has to check character to know how to draw itself
  - Any method whose behavior depends on tile type needs similar conditional logic

Tile class hierarchy

Tile

```
protected double size
protected char ch

char getCharacter()
void draw(double x, double y)
```



LetterTile extends Tile

```
LetterTile(double size)
void draw(double x, double y)
```

Draw a square tile

Construct using a random letter A-Z

NumberTile extends Tile

```
NumberTile(double size)
void draw(double x, double y)
```

Draw a circle tile

Construct using a random number 0-9

Terminology: Concrete vs. Abstract class

- Concrete class
  - Some classes make sense to create
  - e.g. LetterTile, NumberTile
- Abstract class
  - Some classes don't make sense to create
  - Exist so children can inherit things
  - Exist so related objects can live in same array
  - e.g. Tile

Abstract Tile class

Prevents anyone from creating a Tile object

Child classes will get this method for free

```
public abstract class Tile
{
   protected double size = 0.0;
   protected char ch = '\0';

   public char getCharacter()
   {
      return ch;
   }

   public abstract void draw(double x, double y);
}
```

All child classes must implement a method call draw with exactly this signature. This is what makes the polymorphism work (i.e. we can put any child of Tile into the same array and call draw() on every element).

Concrete LetterTile class

```
public class LetterTile extends Tile
{
   public LetterTile(double size)
   {
      this.ch = (char) StdRandom.uniform((int) 'A',
                                         (int) 'Z' + 1);
      this.size = size;
   }

   public void draw(double x, double y)
   {
      StdDraw.setPenColor(StdDraw.BLUE);
      StdDraw.filledRectangle(x, y, size / 2.0, size / 2.0);

      StdDraw.setPenColor(StdDraw.WHITE);
      StdDraw.text(x, y, "" + ch);
   }
}
```

Randomly assign a letter between A and Z

Since we extend Tile, we must implement all abstract methods declared in Tile

Concrete NumberTile class

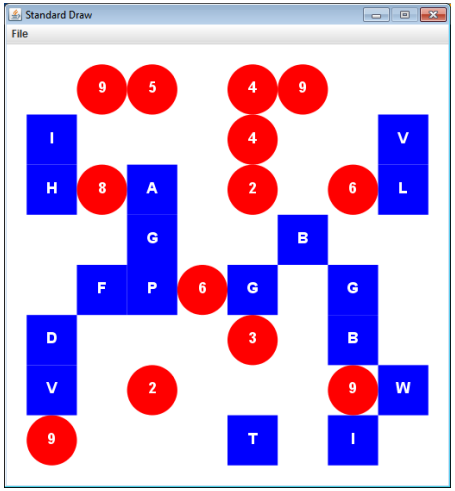
```
public class NumberTile extends Tile
{
   public NumberTile(double size)
   {
      this.ch = (char) StdRandom.uniform((int) '0',
                                         (int) '9' + 1);
      this.size = size;
   }

   public void draw(double x, double y)
   {
      StdDraw.setPenColor(StdDraw.RED);
      StdDraw.filledCircle(x, y, size / 2.0);

      StdDraw.setPenColor(StdDraw.WHITE);
      StdDraw.text(x, y, "" + ch);
   }
}
```

Randomly assign a letter between 0 and 9

Since we extend Tile, we must implement all abstract methods declared in Tile

TileBoard class

- Manage the N x N grid inside another class
  - Create for a given number of tiles, grid size, and canvas size
  - Draw itself



TileBoard

```
Tile [][] tiles // 2D array storing LetterTile and NumberTile objs
double size     // Size of tiles in StdDraw coordinates

TileBoard(int numTiles, int gridSize, double canvasSize)
void draw()
```

TileBoard constructor

```
public TileBoard(int numTiles, int gridSize, double canvasSize)
{
   tiles = new Tile[numTiles][numTiles];
   size = canvasSize / gridSize;

   int added = 0;
   while ((added < numTiles) && (added < gridSize * gridSize))
   {
      int x = (int) (Math.random() * gridSize);
      int y = (int) (Math.random() * gridSize);

      if (tiles[x][y] == null)
      {
         if (Math.random() < 0.5)
            tiles[x][y] = new LetterTile(size);
         else
            tiles[x][y] = new NumberTile(size);
         added++;
      }
   }
}
```

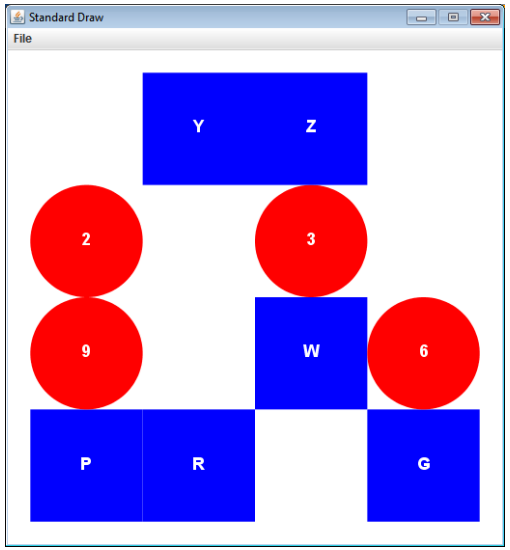
TileBoard drawing

```
public void draw()
{
    StdDraw.setFont(new Font("SansSerif", Font.BOLD, 18));

    for (int x = 0; x < tiles.length; x++)
    {
       for (int y = 0; y < tiles[x].length; y++)
       {
          if (tiles[x][y] != null)
             tiles[x][y].draw(size * (x + 0.5),
                              size * (y + 0.5));
       }
    }
}
```

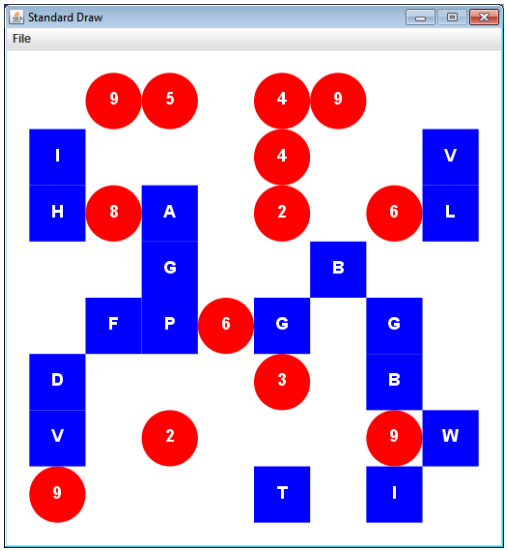
TileGame main program

```
public class TileGame
{
   public static void main(String []args)
   {
      TileBoard board = new TileBoard(Integer.parseInt(args[0]),
                                      Integer.parseInt(args[1]),
                                      1.0);

      board.draw();
   }
}
```

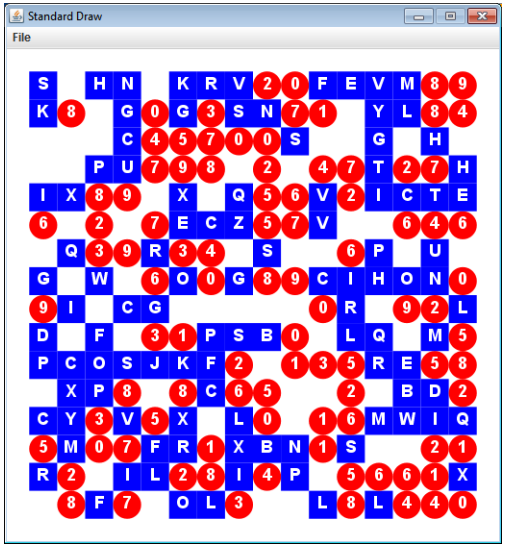
```
% java TileGame 10 4
```




```
% java TileGame 30 8
```




```
% java TileGame 200 16
```



Summary

- Multidimensional arrays
  - An array of arrays
  - Handy when you have a grid of values/objects
  - Focused on 2D arrays
    - But higher dimensional arrays also possible
- Designed classes for a tile game
  - Abstract base class
  - Concrete classes for letter and number tiles
  - Object to manage the entire game board