CSC 1051 Algorithms & Data Structures I

More with Arrays

MORE WITH ARRAYS

Command-Line Arguments

The main method accepts an array of strings as a parameter

```
public static void main(String[] args)
{
    // whatever
}
```

This parameter is often ignored

But it can be used to provide command-line arguments to a program

Any values added to the command line when a program is run are put into a String array and passed to main

Command-Line Arguments

In the JDK command-line environment, the java command runs a program

```
> java DoThis
```

Command-line arguments are added after the program name

```
> java DoThis one another "yet another"
```

args[0] would contain "one"

args[1] would contain "another"

args[2] would contain "yet another"

Command-Line Arguments

Command-line arguments are often used to specify file names

Suppose a program reads and input file and writes to an optional output file

It's common for such a program to check the arguments and issue a usage message (and terminate) if there's a problem

A usage message indicates what arguments can be provided when the program is run

Arguments in square brackets are considered optional

Command-Line Arguments

```
public class FileProcessor
{
    public static void main(String[] args)
    {
        if (args.length < 1 || args.length > 2)
        {
            System.out.println("Usage: java FileProcessor "
                + "inputfile [outputfile]");
            System.exit(1);
        }

        String inputFileName = args[0];
        String outputFileName = null;
        if (args.length == 2)
            outputFileName = args[1];

        System.out.println("Processing...");
    }
}
```

Command-Line Arguments

Here's the FileProcessor program executed with various arguments:

```
> java FileProcessor input.txt
Processing...

> java FileProcessor input.txt output.txt
Processing...

> java FileProcessor
Usage: java FileProcessor inputfile [outputfile]

> java FileProcessor one too many
Usage: java FileProcessor inputfile [outputfile]
```

Command-Line Arguments

Command-line arguments are also typically used to specify program options

They customarily begin with a dash

Suppose our file processing program used a –v option to indicate verbose output and –b to eliminate blank lines

We'll allow options to be specified anywhere on the command line

This time, we won't check the number of arguments

Instead, we'll assume that the first non-option argument is the input file and the second (if present) is the output file

Command-Line Arguments

```
...
        for (String arg : args)
        {
            if (arg.startsWith("-"))  // program option
            {
                if (arg.equals("-v"))
                    verboseOutput = true;
                else if (arg.equals("-b"))
                    supressBlankLines = true;
            }
            else if (inputFileName == null)
                inputFileName = arg;
            else
                outputFileName = arg;
        }
...
```

Command-Line Arguments

```
> java FileProcessor2 -v input.txt
Processing...

> java FileProcessor2 -b -v input.txt output.txt
Processing...

> java FileProcessor2 input.txt -v output.txt
Processing...

> java FileProcessor2 input.txt -v -b
Processing...
```

Variable-Length Argument Lists

A method can be declared so that it accepts a variable number of arguments when called

Ellipsis (…) in the parameter list of the method declaration indicates 0 or more parameters

The parameters passed are put into an array

```
public void setLevels(double... list)
{
    // treat list as an array of double
}
```

Variable-Length Argument Lists

Now setLevels can be called with any number of double parameters

```
setLevels(1.23, 4.56);
setLevels(9.8, 7.6, 5.4, 3.2);
setLevels(107.4825);
```

This technique provides flexibility without forcing the caller to pack data into an array or collection

Variable-Length Argument Lists

The following method computes the sum of all integers passed into it

```
public static int sum(int... numbers)
{
    int total = 0;
    for (int num : numbers)
        total += num;
    return total;
}
```

Variable-Length Argument Lists

Calling the sum method:

```
public static void main(String[] args)
{
    System.out.println("Sum of 0 arguments: " + sum());
    System.out.println("Sum of 2 arguments: " +
        sum(10, 20));
    System.out.println("Sum of 3 arguments: " +
        sum(10, 20, 30));
    System.out.println("Sum of 5 arguments: " +
        sum(10, 20, 30, 40, 50));

    int[] nums = {10, 20, 30, 40, 50, 60, 70, 80};
    System.out.println("Sum of array: " + sum(nums));
}
```

Variable-Length Argument Lists

Note that a method that accepts variable arguments can also be passed an array that's been previously created

```
Sum of 0 arguments: 0
Sum of 2 arguments: 30
Sum of 3 arguments: 60
Sum of 5 arguments: 150
Sum of array: 360
```

Variable-Length Argument Lists

```
public static void main(String[] args)
{
    System.out.print("Ittttttttt's ");
    System.out.println(concat("super", "cali", "fragi",
        "listic", "expi", "ali", "docious", "!"));
}

public static String concat(String... stringList)
{
    String result = "";
    for (String str : stringList)
        result += str;
    return result;
}
```

```
Ittttttttt's supercalifragilisticexpialidocious!
```

Variable-Length Argument Lists

A method that accepts a variable number of arguments can also accept other parameters

The variable-length list must come last

The following method accepts a String, a double, and zero or more integers

```
public void doThis(String str, double d, int... list)
{
    // whatever
}
```

Common Array Algorithms

Explore the **Common Array Algorithms** topic in Rephactor to see code in action for each of these:

- Filling and printing
- Computing an Average
- Finding a minimum or maximum
- Linear search
- Swapping elements

Two-Dimensional Arrays

A basic array is a single-dimensional array

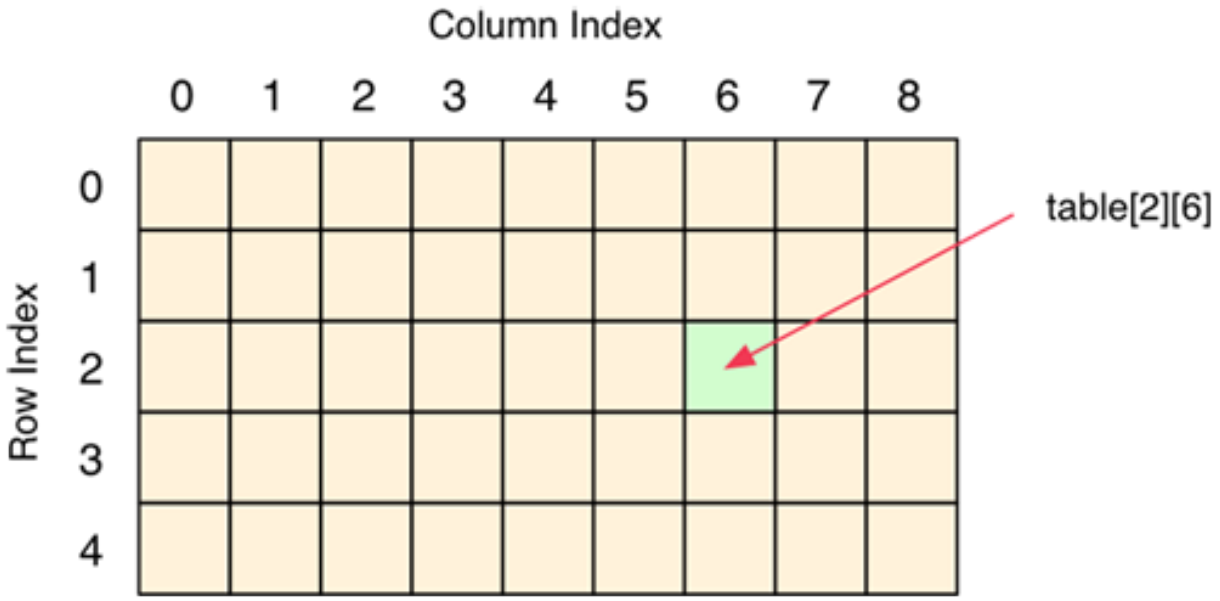
A two-dimensional array is accessed via row and column and appropriate for tabular data

Movie	Rev 1	Rev 2	Rev 3	Rev 4	Rev 5
Godzilla	3	3	4	2	3
Guardians of the Galaxy	4	4	5	5	4
Jersey Boys	3	3	3	4	3
Maleficent	4	3	3	4	3
The Expendables 3	2	1	3	2	2
The Fault in our Stars	4	3	4	4	3
X-Men: Days of Future Past	4	3	4	3	3

Two-Dimensional Arrays

Both row and column indexes begin at 0

Use brackets for each index



Two-Dimensional Arrays

All elements must have the same type

The type of a 2-D array of integers is `int[][]`

```
int[][] matrix;
```

As with a 1-D array, the size of each dimension is specified when the array object is created

```
matrix = new int[12][20];
```

This array has 12 rows (indexed 0 to 11) and 20 columns (indexed 0 to 19)

Two-Dimensional Arrays

You can also use an initialization list to create 2-D arrays

```
int[][] reviews = { {3, 3, 4, 2, 3},
                    {4, 3, 4, 3, 3},
                    {3, 3, 3, 4, 3},
                    {4, 3, 3, 4, 3},
                    {2, 1, 3, 2, 2},
                    {4, 3, 4, 4, 3},
                    {4, 4, 5, 5, 4} };
```

This is essentially a list of lists

Two-Dimensional Arrays

Nested loops are often used to access elements in a 2-D array

The following code fills a 2-D array with random double values

```
double[][] scores = new double[30][5];

for (int row = 0; row < scores.length; row++)
    for (int col = 0; col < scores[row].length; col++)
        scores[row][col] = Math.random() * 100;
```

scores.length is the number of rows

scores[row].length is the number of columns in that row

Two-Dimensional Arrays

Processing the movie reviews:

```
public static void main(String[] args)
{
    int[][] reviews = { {3, 3, 4, 2, 3},
                        {4, 4, 5, 5, 4},
                        {3, 3, 3, 4, 3},
                        {4, 3, 3, 4, 3},
                        {2, 1, 3, 2, 2},
                        {4, 3, 4, 4, 3},
                        {4, 3, 4, 3, 3} };

    String[] movies = {"Godzilla", "Guardians of the Galaxy",
      "Jersey Boys", "Maleficent", "The Expendables 3",
      "The Fault in our Stars", "X-Men: Days of Future Past"};

    int sum;
    double average;
```

Two-Dimensional Arrays

```
...

    for (int i = 0; i < reviews.length; i++)
    {
        System.out.printf("%30s", movies[i]);

        sum = 0;
        for (int j = 0; j < reviews[i].length; j++)
        {
            sum += reviews[i][j];
            System.out.printf("%5d", reviews[i][j]);
        }

        average = ((double)sum) / reviews[i].length;
        System.out.printf("%8.1f%n", average);
    }
}
```

Two-Dimensional Arrays

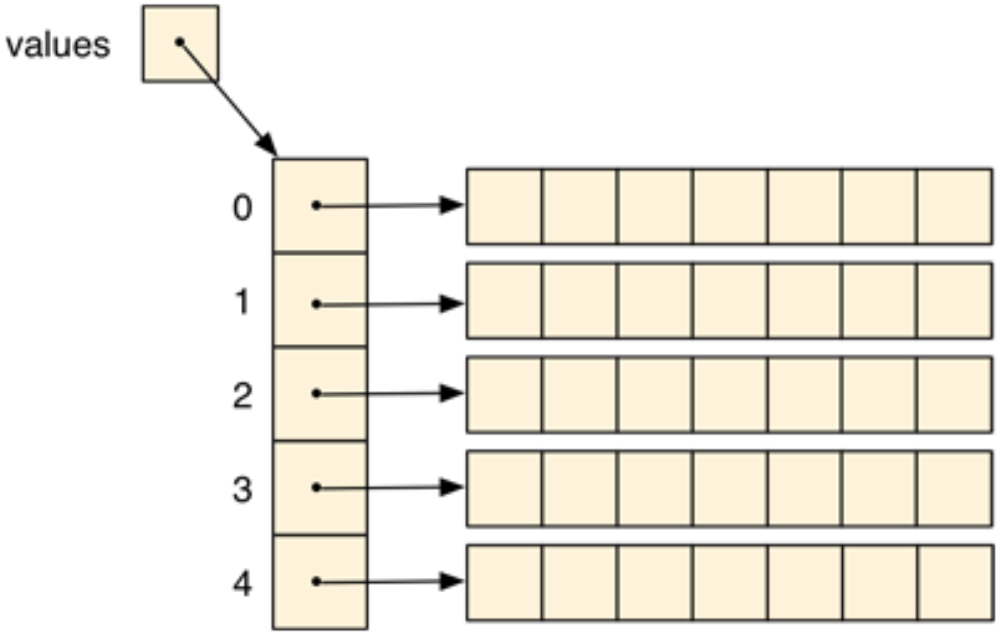
The output includes the reviewer average per movie

```
                  Godzilla   3   3   4   2   3   3.0
Guardians of the Galaxy      4   4   5   5   4   4.4
             Jersey Boys     3   3   3   4   3   3.2
               Maleficent    4   3   3   4   3   3.4
       The Expendables 3     2   1   3   2   2   2.0
 The Fault in our Stars      4   3   4   4   3   3.6
X-Men: Days of Future Past   4   3   4   3   3   3.4
```

Two-Dimensional Arrays

In Java, a 2-D array is really an array of arrays

If values is a 2-D array with 5 rows and 7 columns, it could be depicted like this:



Two-Dimensional Arrays

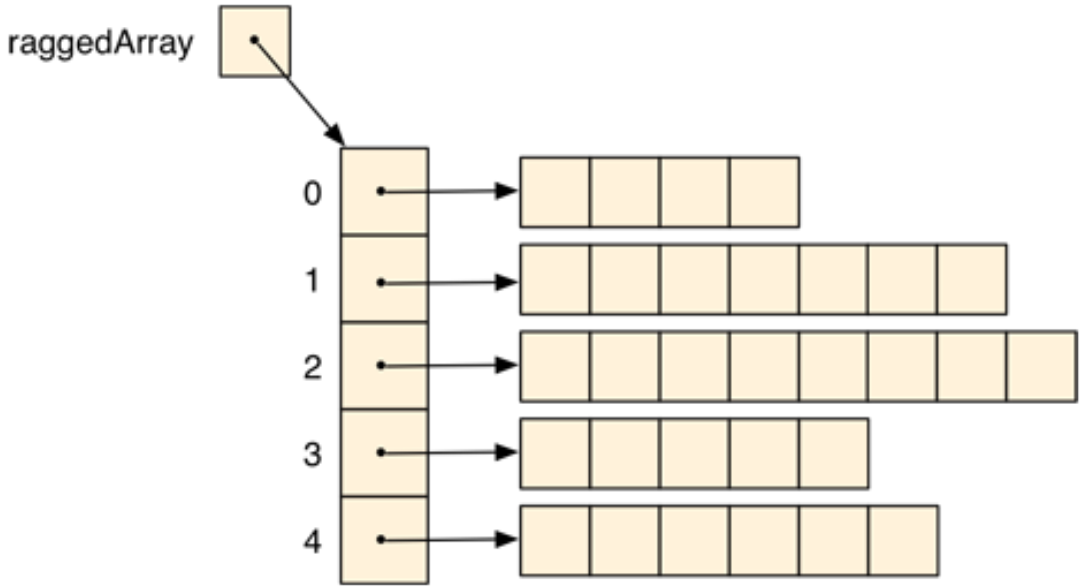
Each row in a 2-D array could be instantiated separately

Each row could therefore have a different number of columns, creating a ragged array

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

Two-Dimensional Arrays

Depicting a ragged array:



Two-Dimensional Arrays

The concept of a 2-D array can be generalized into a multidimensional array

```
int[][][] accidents = new int[12][31][24];
```

This array might be used to store the number of traffic accidents in a town, organized by month, day, and hour

Lists

Explore the **Lists** topic in Rephactor to see descriptions and code in action for each of these:

- Add & Remove
- Get & Set
- Get list Size
- See if list is Empty
- Working with a list in various ways