CSCI 1933 Lab 4

Problem Solving with Arrays

Rules

You may work individually or with *one* other partner on this lab. All steps should be completed primarily in the lab time; however, we require that at least the starred steps (★) to be completed and checked off by a TA during lab time. Starred steps cannot be checked off during office hours. You have **until the last office hours on Monday** to have non-starred steps checked by ANY TA, but we suggest you have them checked before then to avoid crowded office hours on Monday. There is nothing to submit via Moodle for this lab. This policy applies to all labs.

Reminder: You **cannot** use an IDE (IntelliJ, Netbeans, or Eclipse) on labs or projects. We will not check you off if we see you using an IDE.

Want more practice? Gopher it!

In this lab and some future labs, we will have `Gopher it problems`. If you want more practice with Java or problem solving relating to lab, you should Gopher these problems. Gopher it problems are not graded. They are simply here for you to complete at your own leisure. Since these Gopher it problems are not graded, you should only start on these problems **AFTER** you get all of the graded steps checked off first. You cannot use these problems as an excuse for not finishing the project on time. You can find these problems at the end of each lab.

The equals method – We kind of lied

Since the beginning of the course, we've been telling you that you should be using the `equals()` method to check if two objects are equal. This is because using `==` on two objects checks if the two objects share the same memory location. Well, the `equals` method does the same thing **if you don't write it yourself**. Remember the `BankAccount` class? Before reading any further, try using `equals` on two seemingly equal `BankAccounts`, what happens?

Hopefully you tried it out before reading the following. The reason you are seeing that behavior is because if you do not write the `equals` method in your class, e.g., `BankAccount`, the behavior of the `equals` method is the same as using `==`. Therefore, to get the desired behavior, you have to define what it means for two objects, e.g., `BankAccounts`, to be equal. That is, do the `passwords`, `balance`, and `name` have to match, or just a subset of them?

1 Analyzing Owl Dataset ★

One of the tracks offered by the Computer Science degree is Big Data. An important part of Big Data is being able to process a large dataset and perform some sort of analysis on it. For this step, we're going to be doing some statistical analysis on a population of owls. Before we do so, we need to model our owls. Create a `Owl` class with the following member variables: `name`, `age`, and `weight`. Also, include the following method:

- `public boolean equals(Owl other)` – This returns `true` if your name, age, and weight are equal to `other`'s name, age, and weight. Otherwise, this returns `false`

Consider what constructor(s) your `Owl` class needs. Also consider what the appropriate access modifiers for your member variables should be.

To do the statistical analysis for this step, create a `OwlStats` with the following methods:

- `public void analyze(Owl[] owls)` – This will analyze the dataset and save the results. This will overwrite any saved data from previous analysis. The analysis should include computing the average age, finding the youngest `Owl`, and the heaviest `Owl`
- `public Owl getYoungest()` – Returns the youngest `Owl` from the most recent analysis. If no analysis has been performed, return `null`
- `public Owl getHeaviest()` – Returns the heaviest `Owl` from the most recent analysis. If no analysis has been performed, return `null`
- `public String toString()` – This will return a summary of the most recent analysis. The summary should include the name of the youngest `Owl` and the heaviest `Owl`, and the average age. The format of the summary is up to you.

As always, test your code through main. We have a few on the next page to get you started.

Reflection: Imagine that we needed to perform some more statistical analysis in the `analyze` method. For example, suppose we also wanted to find the age and weight that occurs the most in the dataset. How would you prevent your `analyze` method from become too cluttered with analysis code?

Test Cases

```
// new Owl(name, age, weight)
Owl leslie = new Owl("leslie", 10, 10);
Owl andy = new Owl("andy", 5, 11);
Owl ron = new Owl("ron", 21, 21);
Owl[] dataset = {leslie, andy, ron};

OwlStats pawneePop = new OwlStats();
pawneePop.analyze(dataset);
Owl heaviest = pawneePop.getHeaviest();
Owl youngest = pawneePop.getYoungest();
Owl ron2 = new Owl("ron", 21, 21);
String expectedSummary = "Youngest: andy, Heaviest: ron, Average age: 12";
String actualSummary = pawneePop.toString();

System.out.println("Heaviest Test: " + heaviest.equals(ron));
System.out.println("Youngest Test: " + youngest.equals(andy));
System.out.println("Equals Test: " + ron.equals(ron2));
System.out.println("Summary Test: " + expectedSummary.equals(actualSummary));

Owl jerry = new Owl("jerry/gary", 22, 22);
dataset[2] = jerry;
pawneePop.analyze(dataset);
heaviest = pawneePop.getHeaviest();
expectedSummary = "Youngest: andy, Heaviest: jerry/gary, Average age: 12.3";
actualSummary = pawneePop.toString();

System.out.println("Testing for changes dataset");
System.out.println("Heaviest Test: " + heaviest.equals(jerry));
System.out.println("Summary Test: " + expectedSummary.equals(actualSummary));
```

2 Histogram

Recall that a histogram is a visual representation of a distribution of discrete data. For example, the data set $\{3, 2, 1, 2, 3, 0, 1, 5, 3\}$ over the range $[0, 5]$ will have the following histogram:

```
0:*
1:**
2:**
3:***
4:
5:*
```

Create a `Histogram` class to represent a histogram of integer valued data. Your `Histogram` class should have the constructor `public Histogram(int m, int n)`, where m and n refer to the range of the data (inclusive) in the histogram, where $m \leq n$. If a `Histogram` object is initialized with $m > n$, just swap them within the constructor. In our example above, $m = 0$ and $n = 5$. The histogram data should be stored in an `int` array, which you will initialize in the constructor. You must also implement the following instance methods:

- `public boolean add(int i)` – This method will check if i falls in the range of the histogram. If it does, it will add it and return `true` to indicate success. If it does not, it will not add it and return `false` to indicate failure.
- `public void addOutOfRange(int i)` – If i is in the range of your histogram, then this method will function the same as `add`. If i is out of the range, however, then it will adjust the range of your histogram and add i to your histogram appropriately.
- `public String toString()` – Just like last lab, you will write a `toString()` method that returns a string formatted as in our example above. The first and last line should begin with `m:` and `n:` respectively. Note: `\n` is a newline character.

Write a main method that prompts the user for the range of the histogram, then prompts for what data to put in the histogram, and finally prints out the histogram. If the user's input is not in the range of the histogram, print a message saying so (use the return value of `add` to do this). You will find that you will need use 2 `Scanner` instances to read in the data as you did in the last lab.

A sample program execution can be found on the following page (green text is user input):

```
Range? 10 15

(using regular add)
Data? 10 12 13 11 13 13 25 10 77
25 is not in the range
77 is not in the range

10:**
11:*
12:*
13:***
14:
15:

(using addOutOfRange)
Enter one data point: 17

10:**
11:*
12:*
13:***
14:
15:
16:
17:*

(using regular add)
More Data? 16 25 17
25 is not in the range

10:**
11:*
12:*
13:***
14:
15:
16:*
17:**
```

3 Pe ef, Mcfep?

Back before computers were around, encrypted messages were tough to crack. Nowadays, we can crack many naïve encryptions in a matter of seconds. The Caesar cipher is perhaps one of the most well known encryption techniques, mainly because it's the easiest one to crack and understand.

The Caesar cipher simply shifts the entire alphabet over by some number of letters. Letters at the end, e.g. x, y, and z, that "fall off" the alphabet, wrap around. For example, suppose that our `key = 3`. Then, every letter in the alphabet shifts three spots to the **right**:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

To encrypt a message with `key = 3`, then we "read up" the columns of the table. For example, encrypting the message `"arrays are cool"` becomes `"duudbv duh frro"`. To decrypt a message, we shift every letter 3 spots to the **left** and "read up" the columns. For example, decrypting the message `"dqg qhdw"` gives `"and neat"`

For this step, create a `CaesarCipher` class with the following methods:

- `public CaesarCipher(int key)` – This is your constructor
- `public String encrypt(String message)` – Returns your `message` in its encrypted form.

  > **Hint:** You might want to check out the String API
- `public String decrypt(String message)` – Returns the encrypted `message` in its decrypted form. Do not decrypt spaces.
- `public static void crack(String message)` – This method will crack an encrypted `message` by printing out every decryption possible with its corresponding `key` value.

  > **Hint:** Are there `keys` that will produce the same encryption?
- `public void displayEncryptionStats()` – This will print out how often each letter in the alphabet has been encrypted. The format is up to you.

Continued onto the next page

Write a main method that prompts the user for a `key`, then continuously prompts them whether they want to `encrypt`, `decrypt`, `crack`, or `quit`. If the user chooses `encrypt, decrypt, or crack`, then prompt the user for a `message` that they want to `encrypt/decrypt/crack`. If the user chooses `quit`, then print out the number of times each letter has been encrypted by your `CaeserCipher` before ending the main method.

A sample program execution can be found below (green text is user input):

```
What is your key? 11

What would you like to do?
encypt
decrypt
crack
quit

encrypt
Please enter the message The quick brown fox jumped over the lazy sheepdog
Wkh txlfn eurzq ira mxpshg ryhu wkh odcb vkhhsgrj

What would you like to do?
encypt
decrypt
crack
quit

decrypt
Please enter the message Pe ef Mcfep
Et tu Brute

What would you like to do?
encypt
decrypt
crack
quit

crack
Please enter the message Pe ef Mcfep
Pe ef Mcfep - key = 0
Od de lbedo - key = 1
Mc cd kadcn - key = 2
// Omitting the rest because it somewhat gives away the solution
```

4 Matrices Using 2-D Arrays

Matrices are a fundamental component in mathematics and its subfields, and come up often in computer science. Conveniently, matrices are just 2-dimensional arrays.

For this step, we want you to create a `Matrix` class and implement the the following methods:

- `static int[][] add(int[][] a, int[][] b)` – Returns the sum of matrices `a` and `b`
- `static int[][] multiply(int[][] a, int[][] b)` – Multiplies the two matrices together. Since matrix multiplication is not necessarily commutative, we want `a` × `b`
- `static int[][] transpose(int[][] m)` – Returns the transpose of `m`.

Requirement: For methods that take in two matrices, you cannot assume that their dimensions agree. If they do not, then return `null`

Matrix Review

Define an $m \times n$ matrix (m rows, n columns) as follows:

$$A := \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{(m-1)0} & a_{(m-1)1} & a_{(m-1)2} & \cdots & a_{(m-1)(n-1)} \end{bmatrix}$$

- Recall that the sum of matrices $X = A + B$ will have $x_{ij} = a_{ij} + b_{ij}$.
- Recall that the product of two matrices $Y = C \times D$ will have $y_{ij} = c_{i*} \cdot d_{*j}$, where c_{i*} is the ith row of C as a vector, and d_{*j} is the jth column of D as a vector, and $(\cdot)$ is the dot product.

  Note: Matrix multiplication is not necessarily commutative*, so the dimensions must agree. That is, the dimensions of C take the form $m \times n$ and the dimensions of D take the form $n \times p$.

*unless, of course, we are working in a commutative matrix ring, which we aren't

- Recall that the transpose of the matrix A above is defined as:

$$A^T := \begin{bmatrix} a_{00} & a_{10} & \dots & a_{(m-1)0} \\ a_{01} & a_{11} & \dots & a_{(m-1)1} \\ a_{02} & a_{12} & \dots & a_{(m-1)2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(n-1)} & a_{1(n-1)} & \dots & a_{(m-1)(n-1)} \end{bmatrix}$$

Note:

If we have a matrix $E = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, we would write it in Java as:

```
int[][] e = {{1, 2, 3}, {4, 5, 6}};
```

If we wanted to initialize a 3×4 matrix, we would write it in Java as

```
int[][] m = new int[3][4]
```

We can access or modify the 2nd row, 1st column element of `m` with `m[1][0]`.

5 The Vigenère cipher (Honors)

The Vigenère cipher similar to the Caesar cipher in a sense that it uses the Caesar cipher multiple times. You will want to keep the following table up. Unlike the Caesar cipher, the Vigenére cipher uses a passphrase as a key instead of a numeric value.

Using the `key = "phone"` to encrypt the message `"arraysarecool"`, we repeat the key multiple times until it matches the length of the message we want to encrypt; extra letters are discarded:

a	r	r	a	y	s	a	r	e	c	o	o	l
p	h	o	n	e	p	h	o	n	e	p	h	o

Using the table, to encrypt the letter 'a' in our message, we look at the letter in row 'p' and column 'a', namely 'p'. To encrypt 'r', we look at row 'h' column 'r' which gives us 'y'. As we keep going, the encrypted form of `arraysarecool` is: `pyfnchhfrgdvz`.

To decrypt a message, we go backwards.

p	y	f	n	c	h	h	f	r	g	d	v	z
p	h	o	n	e	p	h	o	n	e	p	h	o

To decrypt 'p' in "**p**yfnchhfrgdvz", we look at row 'p' from "**p**hone", and then find the column where 'p' is located, namely 'a'. To decrypt "p**y**fnchhfrgdvz", we look at row 'h' from "p**h**one", and then find where 'y' is located, namely 'r'. If we keep going, we see that `pyfnchhfrgdvz` decrypts to `arraysarecool`.

Write a `VigenereCipher` class with the following methods:

- `public VigenereCipher(String phrase)` – This is your constructor
- `public String encrypt(String message)` – Returns your `message` in its encrypted form. You may assume that there are no spaces in your `message`
- `public String decrypt(String message)` – Returns the encrypted `message` in its decrypted form. You may assume that there are no spaces in your `message`

Test your methods in a similar fashion to step 3. You don't have to worry about printing out the encryption statistics or cracking the Vigenère cipher.

6 Gopher it! (Optional Problems)

6.1 You will substitute for Q today

Unlike the Caesar cipher, the Substitution cipher does not use a number as a key. Instead, the Substitution cipher uniquely maps each letter in the alphabet to another. Then to encrypt a message, the cipher simply replaces all of the letters in the message with their corresponding mapping. Consider the following mapping:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
x	z	l	s	c	a	q	b	i	v	f	m	h	d	g	w	u	n	e	k	t	o	r	j	y	p

Then to encrypt a message, we "read down" the columns of the mapping. For example, the message `"parks and rec"` becomes: `"wxnfe xds ncl"`. To decrypt, we simply "read up" the columns. So the message `"ie xrceghc"` decrypts to `"is awesome"`. For this problem, create a `SubstitutionCipher` class with the following methods:

- `public SubstitutionCipher(String[][] mapping)` – This is your constructor. You can change the type of the array to `char[][]` if you wish.
- `public String encrypt(String message)` – Returns your `message` in its encrypted form.
- `public String decrypt(String message)` – Returns the encrypted `message` in its decrypted form.

Be sure to test your methods! One way is to find someone else who wrote a `SubstitutionCipher` and trade messages with each other. Be sure to trade keys as well, otherwise you cannot decrypt each other's messages!

Exploration: How can you crack a Substitution cipher? Is there some way you can eliminate some of the mappings before you attempt to crack the message? **Hint:** given the ciphertext (an encrypted message) `"xds"`, we know that it has to decrypt to a three letter word.