Building Java Programs

Chapter 7: Arrays

Lecture outline

- array traversal algorithms
  - printing an array's elements
  - searching and reversing an array

Why are arrays useful?

- Storing a large amount of data
  - Example: Read a file of numbers and print them in reverse order.

- Grouping related data
  - Example: Tallying exam scores from 0 through 100.

- Accessing data multiple times, or in random order
  - Example: Weather program.

Array initialization statement

- Quick array initialization, general syntax:

  <*type*> [] **<*name*>** = {**<*value*>**, **<*value*>**, ..., **<*value*>**};

  - Example:

    ```
    int[] numbers = {12, 49, -2, 26, 5, 17, -6};
    ```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

  - Useful when you know what the array's element values will be.
  - The compiler figures out the size by counting the values.

Array practice problem

- What element values are stored in the following array?

```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = 1; i < a.length; i++) {
    a[i] += a[i - 1];
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	2	7	8	14	28	35	44

Array practice problem

- What element values are stored in the following array?

```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = a.length - 1; i >= 1; i--) {
    if (a[i] > a[i - 1]) {
        a[i - 1] = a[i - 1] * 2;
    }
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	4	5	2	12	14	14	9

The Arrays class

- The `Arrays` class in package `java.util` has several useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`*array*, *value*`)`	returns the index of the given value in a sorted array (< 0 if not found)
`equals(`*array1*, *array2*`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`*array*, *value*`)`	sets every element in the array to have the given value
`sort(`*array*`)`	arranges the elements in the array into ascending order
`toString(`*array*`)`	returns a string representing the array, such as `"[10, 30, 17]"`

Arrays.toString

- `Arrays.toString` accepts an array as a parameter and returns its data as a `String`, which you can print.

  - Example:

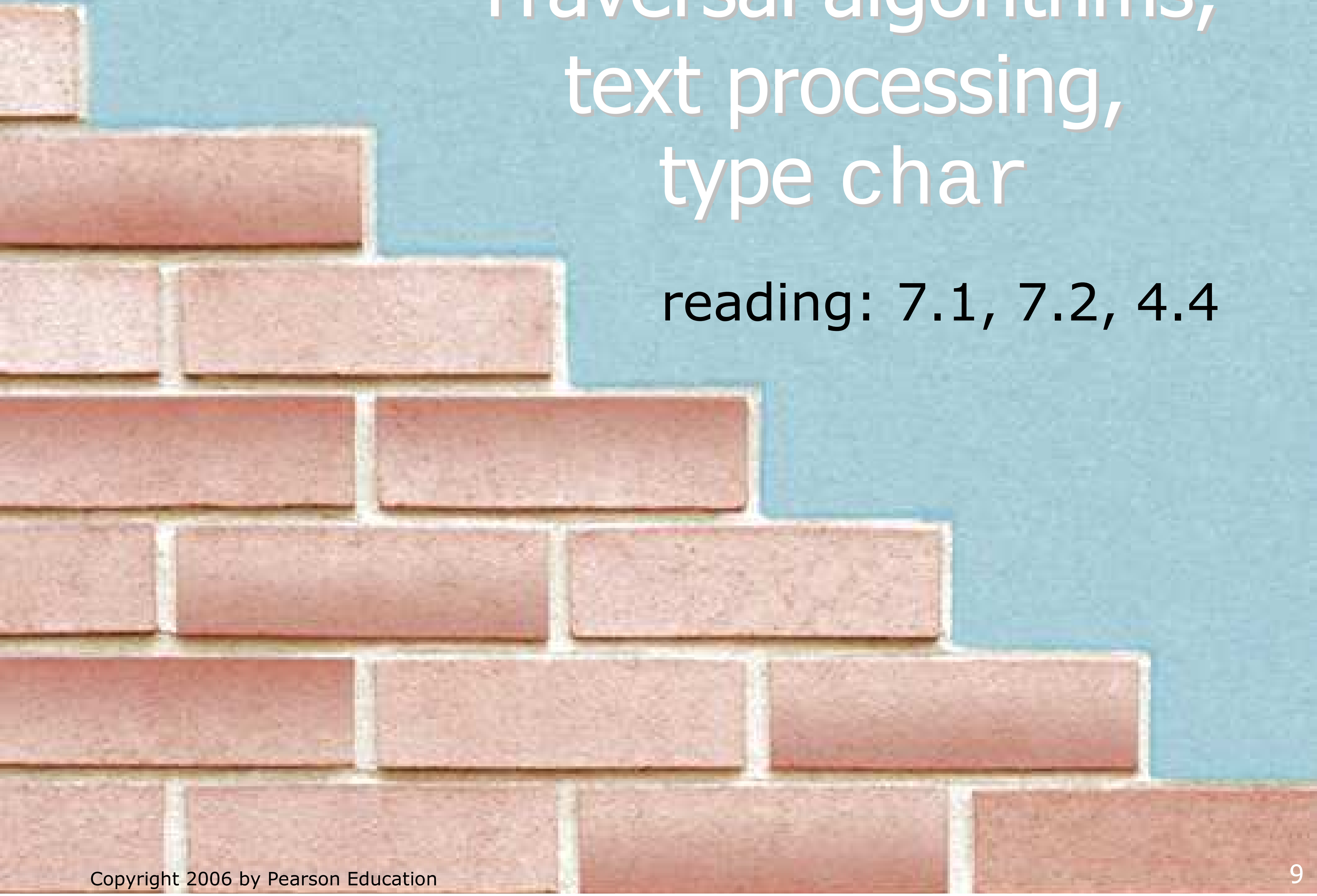
```
int[] a = {2, 5, 1, 6, 14, 7, 9};
for (int i = 1; i < a.length; i++) {
    a[i] += a[i - 1];
}
System.out.println("a is " + Arrays.toString(a));
```

  Output:

```
a is [2, 7, 8, 14, 28, 35, 44]
```

Traversal algorithms, text processing, type char

reading: 7.1, 7.2, 4.4



Array traversal

- **traversal**: An examination of each element of an array.
  - Traversal algorithms often take the following form:

```
for (int i = 0; i < <array>.length; i++) {
    do something with <array> [i];
}
```

- Examples:
  - printing the elements
  - searching for a specific value
  - rearranging the elements
  - computing the sum, product, etc.

Examining array elements

- Example (find the largest even integer in an array):

```
int[] list = {4, 1, 2, 7, 6, 3, 2, 4, 0, 9};
int largestEven = 0;
for (int i = 0; i < list.length; i++) {
    if (list[i] % 2 == 0 && list[i] > largestEven) {
        largestEven = list[i];
    }
}
System.out.println("Largest even: " + largestEven);
```

Output:

```
Largest even: 6
```

Strings and arrays

- `String`s are represented internally as arrays.
  - Each character is stored as a value of primitive type `char`.
  - Strings use 0-based indexes, like arrays.
  - We can write algorithms to traverse strings.

  - Example:

```
String str = "Ali G.";
```

index	*0*	*1*	*2*	*3*	*4*	*5*
value	`'A'`	`'l'`	`'i'`	`' '`	`'G'`	`'.'`

Type char

- **char**: A primitive type representing a single character.
  - Literal `char` values are surrounded with apostrophe marks: `'a'` or `'4'` or `'\n'` or `'\''`
  - You can have variables, parameters, returns of type `char`

    ```
    char letter = 'S';
    System.out.println(letter);   // S
    ```

  - You can compare `char` values with relational operators:
    - `'a' < 'b'` and `'Q' != 'q'`
    - You cannot use these operators on a `String` or any other object.
    - An example that prints the alphabet:

      ```
      for (char c = 'a'; c <= 'z'; c++) {
          System.out.print(c);
      }
      ```

The charAt method

- Access a string's characters with its `charAt` method.

```
String word = console.next();
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("That's good enough for me!");
}
```

- We can use `for` loops to examine each character.

```
String name = "tall";
for (int i = 0; i < name.length(); i++) {
    System.out.println(name.charAt(i));
}
```

Output:

```
t
a
l
l
```

char vs. int

- All `char` values are assigned numbers internally by the computer, called **ASCII** values.
  - Examples:
    `'A'` is 65, `'B'` is 66, `'a'` is 97, `'b'` is 98
  - Mixing `char` and `int` causes automatic conversion to `int`.
    `'a' + 10` is 107, `'A' + 'A'` is 130
  - To convert an integer into the equivalent character, type cast it.
    `(char) ('a' + 2)` is `'c'`

char vs. String

- `'h'` is a `char`

  `char c = 'h';`

  - `char` values are primitive; you cannot call methods on them; can't say `c.length()` or `c.toUpperCase()`

- `"h"` is a `String`

  `String s = "h";`

  - Strings are objects; they contain methods that can be called
  - *can* say `s.length()` $\longrightarrow$ `1`
  - *can* say `s.toUpperCase()` $\longrightarrow$ `"H"`
  - *can* say `s.charAt(0)` $\longrightarrow$ `'h'`

- What is `s + s` ? What is `c + c` ?
- What is `s + 1` ? What is `c + 1` ?

Text processing

- **text processing**: Examining, editing, formatting text.
  - Often involves `for` loops that examine the characters of a string
  - Use `charAt` to search for or count a particular value in a string.

```
// Returns the count of occurrences of c in s.
public static int count(String s, char c) {
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == c) {
            count++;
        }
    }
    return count;
}
```

  - `count("mississippi", 'i')` returns `4`

Text processing example

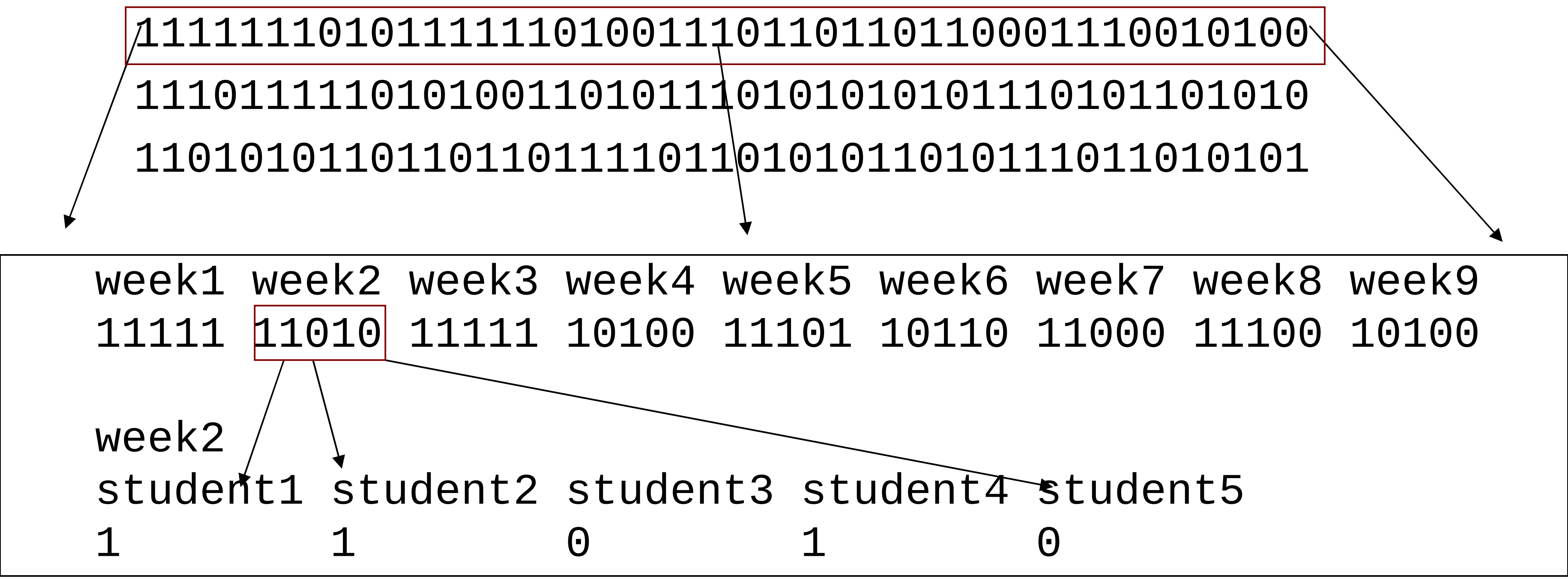
```
// string stores votes: (R)epub., (D)emo., (I)ndep.
String votes = "RDRDRRIDRRRDDDDIRRRDRRRDIDIDDRDDRRDRDIDD";

int[] counts = new int[3];     // R -> 0, D -> 1, I -> 2

for (int i = 0; i < votes.length(); i++) {
    char c = votes.charAt(i);
    if (c == 'R') {             // put vote in proper box
        counts[0]++;
    } else if (c == 'D') {
        counts[1]++;
    } else {// c == 'I'
        counts[2]++;
    }
}
System.out.println(Arrays.toString(counts));
```

Output:

```
[17, 18, 5]
```

Section attendance problem

- Consider an input file of course attendance data:



```
111111101011111101001110110110110001110010100
111011111010100110101110101010101110101101010
110101011011011011110110101011010111011010101
```

```
week1 week2 week3 week4 week5 week6 week7 week8 week9
11111 11010 11111 10100 11101 10110 11000 11100 10100

week2
student1 student2 student3 student4 student5
1        1        0        1        0
```

- Each line represents a section (5 students, 9 weeks).
  - 1 means the student attended; 0 not.

Array transformations

- In this problem we convert data from one form to another.
  - This is called *transforming* the data.
  - Often each transformation is stored into its own array.

- We must map between the data and array indexes.

  Examples:

tally	(if input value is *i*, store it at array index *i*)
by position	(store the *i* th value we read at index *i*)
explicit mapping	(count `'R'` at index 0, count `'D'` at index 1)

Section attendance problem

- Write a program that reads the preceding section data file and produces the following output:

```
Section #1:
Sections attended: [9, 6, 7, 4, 3]
Student scores: [20, 18, 20, 12, 9]
Student grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Section #2:
Sections attended: [6, 7, 5, 6, 4]
Student scores: [18, 20, 15, 18, 12]
Student grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Section #3:
Sections attended: [5, 6, 5, 7, 6]
Student scores: [15, 18, 15, 20, 18]
Student grades: [75.0, 90.0, 75.0, 100.0, 90.0]
```

Section attendance solution

```
// This program reads a file representing which students attended
// which discussion sections and produces output of the students'
// section attendance and scores.

import java.io.*;
import java.util.*;

public class Sections {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(new File("sections.txt"));
        int section = 0;    // used to count sections

        while (input.hasNextLine()) {
            String line = input.nextLine();    // one section's data
            section++;
            System.out.println("Section #" + section + ":");

            int[] attended = new int[5];       // count sections attended
            for (int i = 0; i < line.length(); i++) {
                char c = line.charAt(i);
                if (c == '1') {                // student attended section
                    attended[i % 5]++;
                }
            }
            System.out.println("Sections attended: " + Arrays.toString(attended));

            ...
```

Section attendance solution 2

```
        ...

        // compute section score out of 20 points
        int[] scores = new int[5];
        for (int i = 0; i < scores.length; i++) {
            scores[i] = Math.min(3 * attended[i], 20);
        }
        System.out.println("Student scores: " + Arrays.toString(scores));

        // compute section grade out of 100%
        double[] grades = new double[5];
        for (int i = 0; i < scores.length; i++) {
            grades[i] = 100.0 * scores[i] / 20;
        }
        System.out.println("Student grades: " + Arrays.toString(grades));
        System.out.println();
    }
  }
}
```

- The program can be improved:
  - It doesn't have any static methods.
  - To add methods, we'll need to pass arrays as parameters. (seen next time)

Text processing questions

- Write a method named `pigLatin` that accepts a `String` as a parameter and returns that word in simple Pig Latin, placing the word's first letter and *ay* at the end.
  - `pigLatin("hello")` returns `ello-hay`
  - `pigLatin("goodbye")` returns `oodbye-gay`

- Write methods named `encode` and `decode` that accept a `String` as a parameter and return that `String` with each of its letters increased or decreased by 1.
  - `encode("hello")` returns `ifmmp`
  - `decode("ifmmp")` returns `hello`