

# WES-CS GROUP MEETING #5

## Exercise 1: Writing the PigLatin class

The rules for translating an English word to Pig-Latin are as follows:

- If the word starts with a vowel, it is unchanged.
- Otherwise, all of the consonants at the beginning of the word (up to the first vowel) are moved to the end of the word, preceded by a dash (for readability), and followed by “ay”.

For example, the sentence “I love Madison in the springtime” would be translated to “I ove-lay adison-May in e-thay ingtime-spray”.

For this exercise, you will complete the PigLatin class started on the next page, which will allow users of the class to translate an English word to Pig-Latin.

First, make sure you understand the translation rules by translating the following phrases into Pig Latin:

Hello world

I love WESCS

the car says ooga ooga

**Part (a):** Take a look at the incomplete PigLatin class. Make sure you understand what each method is supposed to do, and discuss the reasons for making some of the methods private and others public.

```

import java.io.*;

class PigLatin {
    String englishWord;

    /* constructor */
    public PigLatin( String s ) {
        englishWord = s.toLowerCase();
    }

    /* translate: translate the English word to Pig-Latin */
    public String translate( ) {
        return "translate not implemented yet";
    }

    /* firstVowelPos: return the position of the first vowel
     *                 in the English word; return -1 if there
     *                 is NO vowel in the word
     */
    private int firstVowelPos( ) {
        return -1; // not implemented yet
    }

    /* isVowel: return true if the letter at the given position of
     * the English word is a vowel; otherwise, return false
     */
    private boolean isVowel( int pos ) {
        return false; // not implemented yet
    }

    /* main: main method to be used to test the other methods */
    public static void main(String[] args) throws IOException {
        PigLatin pig;
        String oneWord;
        BufferedReader input =
            new BufferedReader( new InputStreamReader(System.in) );
        System.out.print("enter English word: ");
        oneWord = input.readLine();
        pig = new PigLatin(oneWord);
        /* ADD CODE HERE */
        System.out.println(pig.translate());
    }
}

```

**Part (b):** Write the *isVowel* method, which returns true if the letter at the given position of the *englishWord* data member is a vowel, and otherwise returns false. To get the letter at the given position, use the *charAt* method of the String class:

*charAt( int index )* returns the character at the specified index

Try writing the *isVowel* method using an if statement and also using a switch statement.

To test your code, add some statements to the PigLatin class's *main* method that call *isVowel* (passing different values for the position) and print the returned value. Then compile and run the program.

**Part (c):** Now write the *firstVowelPos* method, which returns the position of the first vowel in the *englishWord* data member of the PigLatin class, or returns -1 if there is no vowel in the word.

Here are some examples:

<b>englishWord</b>	<b>Result of calling <i>firstVowelPos</i></b>
hello	1
ice	0
spring	3
qyzzx	-1

Test your *firstVowelPos* method by replacing the calls to *isVowel* in the *main* method of the *PigLatin* class with a call to *firstVowelPos*.

**Part (d):** Now describe in English how the *translate* method of the PigLatin class should work (using the rules for translating from English to Pig-Latin given above). Then write the actual code, remove the code in the *main* method that prints the first vowel position (leaving the call to the *translate* method), compile your code and run it!

## Exercise 2: The Time and AlarmClock classes

Many groups didn't get to finish the Time and AlarmClock classes two weeks ago. The exercises are repeated below so you can finish them if you'd like.

**Part (a).** First, let's define some new methods for the Time class. Notice that the Time class stores its time in 24-hour format: it has two data members called *hour* and *minute*; *hour* is an integer between 0 and 23 that represents the hours part of the time, and *minute* is an integer between 0 and 59 that represents the minutes part of the time.

Our first new method will be *print12HourTime*, which is like the existing method *print24HourTime*, except that it prints the time in 12-hour format instead of 24-hour format. For example, the 24-hour time 15 20 would be printed as "3:20pm", and the 24-hour time 0 0 would be printed as "midnight".

This method should work as follows:

- If the 24-hour time represents midnight or noon, just print "midnight" or "noon".
- Otherwise, if the time is before noon, print it in 12-hour format (ending with "am").
- Otherwise, print the time in 12-hour format, ending with "pm".

Take a look at the *print24HourTime* method, then write the *print12HourTime* method. Also, add a call to *print12HourTime* in *main*, and compile and run the new program.

**Part (b).** Another useful thing to know is how many minutes there are between two times. For example, it takes 45 minutes to get from noon to 12:45, and it takes 1,395 minutes (23 hours and 15 minutes) to get from 12:45 to noon.

Our next method, *timeRemaining* will tell us how many minutes it takes to get from the time represented by the Time object whose method is called, to a given time. For example, if you have a Time object *now* that represents noon, and another Time object *appointment* that represents 12:45, then the method call *now.timeRemaining( appointment )* will return 45, while the method call *appointment.timeRemaining( now )* will return 1395.

First, figure out (on paper) how to determine the number of minutes it takes to get from one time to another, then write the method.

Add code to *main* to test your method. You may want to do this by reading a time typed in at the terminal, and figuring out how much time there is between that time and the current time (or vice versa). To read one integer from the terminal, use the code on page 5 (i.e., add it to the Time class). You'll also need to put the following line at the beginning of the Time class: `import java.io.*;`

```

import java.util.*;
public class Time {
    /**
     * data members
     */
    private int hour;
    private int minute;

    /**
     * public methods
     */
    /* constructor */
    public Time(int hr, int min) {
        hour = hr;
        minute = min;
    }

    /* print24HourTime */
    public void print24HourTime() {
        System.out.print(hour + ":");
        if (minute < 10) {
            System.out.print("0");
        }
        System.out.println(minute);
    }

    /* main: create a Time object that represents the current time
     * and print that time in 24-hour format
     */
    public static void main(String[] args) {
        int hour = getCurrHour();
        int min = getCurrMinute();
        Time now = new Time(hour, min);
        now.print24HourTime();
    }

    /**
     * private methods
     */
    /* getCurrHour */
    private static int getCurrHour() {
        Calendar cal = new GregorianCalendar();
        return cal.get(Calendar.HOUR_OF_DAY);
    }

    /* getCurrMinute */
    private static int getCurrMinute() {
        Calendar cal = new GregorianCalendar();
        return cal.get(Calendar.MINUTE);
    }
}

```

Code to read one integer value typed in to the terminal.

```
private static int readInt() {
    int num = 0;
    try {
        Reader in = new InputStreamReader(System.in);
        int ch = in.read();
        while (ch != -1 && ch >= '0' && ch <= '9') {
            num = num * 10 + ch - '0';
            ch = in.read();
        }
    } catch (IOException ex) {
        System.out.println("bad number");
    }
    return num;
}
```

Now we're ready to design a new class that uses the `Time` class. The new class is for your team leader who sometimes parties late into the night and oversleeps the next morning. Your team leader needs an *AlarmClock* class!

The `AlarmClock` class should have three methods:

1. A constructor, whose parameter is the wake-up time (a `Time` object).
2. A *doAlarm* method, that keeps checking the actual time until it is equal to the wake-up time. Then it beeps, and finishes.
3. A *main* method that gets the wake-up time from the user of the program, creates a `Time` object to represent the wake-up time, uses the `AlarmClock` constructor to create a new `AlarmClock` object, tells the user what the current time is and how many minutes remain until the alarm will ring, and finally calls the `AlarmClock` object's *doAlarm* method.

To make the computer beep, use

```
System.out.print('\u0007');
```

Write the `AlarmClock` class and test it.

If you want to improve your `AlarmClock` class, try the following:

1. Your team leader is kind of a heavy sleeper, and so is not likely to be woken up by a single-beep alarm clock. Modify the *doAlarm* method so that it beeps once per second for 60 seconds.
2. Make it easier for your team leader to set the alarm by allowing the wake-up time to be specified in 12-hour format. For example: 1:23pm instead of 13 23. Use the *readInt* method to help you figure out how to read a time in 12-hour format and to convert it to 24-hour format for use by the `Time` constructor.

### **Exercise 3: Google searches**

Google has a news service that provides information about current events, including the upcoming election. A recent study found that the news about John Kerry seemed to have a negative bias. One theory is that this is because Google used searches on “John Kerry” rather than “Senator Kerry”. The idea is that publications that are favorable to Kerry tend to use his title, while publications that are unfavorable tend to use just his name.

First, try out this hypothesis by doing searches on “John Kerry”, “Senator Kerry”, and “Senator John Kerry”. See if you think that the name you use for the search affects the results.

Now think of some other searches that you could do using various different search patterns, try the different searches, and evaluate the results. Do the patterns affect whether the results are more positive or more negative?

## Exercise 4: Logical thinking

Play the following game in pairs:

When it's your turn, choose a number greater than 1.

After a number is chosen, some numbers become unavailable:

1. The multiples of numbers that have been chosen are unavailable. So if the first person chooses 4, then 8, 12, 16, etc. are all unavailable.
2. The sums of all of the multiples are also unavailable. So, if the second person chooses 7, then all of the following are unavailable:

11 ( $7 + 4$ ), 15 ( $7 + 8$ ), 23 ( $7 + 16$ ) ...

14 ( $7 \times 2$ ), 18 ( $7 \times 2 + 4$ ), 22 ( $7 \times 2 + 8$ ) ...

etc

You win if you leave your opponent with nothing to choose.

Play this game several times, try to figure out if there is any winning strategy!