600.107 Introductory Programming in Java, Summer 2015
Homework 3
Due: Monday, July 6th at 11:59pm

Please review the **Homework Guidelines** handout for this course before you begin.

Restriction: For all tasks in this assignment, you are not permitted to use repetition statements (while, for, do-while), arrays or ArrayLists.

Task 1. At electric utility companies, electricity is often billed based on the number of kilowatt-hours (kWh) used by a customer. To determine how many kilowatt-hours (kWh) are used by a customer, we find the difference between the meter reading of a customer from the previous month to the meter reading for the current month. For example, if the meter reads 15100 for the previous month and 16232 for the current month, the customer used 16232-15100 = 1132 kWh. Note that a meter reading with fewer than 5 digits is always thought of as having 5 digits (for example, if a reading of 3 is entered, treat it as 00003), and the meter "wraps around" after 99999 back to 00000. There are no valid readings which contain more than 5 digits.

Once usage is calculated, one company computes a customer's charge based on the following table:

kWh Used	Charge
Less than 1000	7 cents per kWh
1000-2500	\$70, plus 5 cents per kWh for each kWh above 1000
More than 2500	\$145, plus 3 cents per kWh for eaach kWh above 2500

Write a Java class called `Electricity` with a main method that uses the Scanner class to read in and echo two integer meter readings (you may assume a meter reading of 00235 would be entered as 235). If either of the readings is not valid (i.e. not between 0 and 99999 inclusive), print out an invalid input message like the one shown in the final example below, and do not perform a calculation. If both readings are valid, output the number of kilowatt-hours used and the user's charge in dollars and cents. Use `printf` to format the dollar amount so that it is printed with exactly two decimal places. You can read about `printf` in Section 5.7 of the text and also here: http://docs.oracle.com/javase/tutorial/java/data/numberformat.html. Be sure to test thoroughly!

See below for sample executions for 3 separate runs of the program (user input shown in bold):

SAMPLE 1

```
Please input last month's meter reading [0-99999]: 15100
Please input this month's meter reading [0-99999]: 16232
You entered readings of 15100 and 16232.
You used 1132 kilowatt-hour(s), and you owe $76.60.
```

SAMPLE 2

```
Please input last month's meter reading [0-99999]: 99900
Please input this month's meter reading [0-99999]: 1
You entered readings of 99900 and 1.
You used 101 kilowatt-hour(s), and you owe $7.07.
```

SAMPLE 3

```
Please input last month's meter reading [0-99999]: 123456
Please input this month's meter reading [0-99999]: 16232
You entered readings of 123456 and 16232.
You have entered an invalid input value, so no bill will be produced.
```

Task 2. Write a Java class named `Dates` which reads a `String` from the keyboard and tests whether that `String` represents a valid date. Display the date and a message which indicates whether or not it is valid. If it is not valid, also display a message explaining *why* it is not valid, as shown below.

The input date will have the form `year/month/day`. You can assume that the format of the input always meets that description, and that aside from the two `/` characters, all other characters in the input `String` are digits. (But you don't know if the digits represent valid years, months, and/or days, so the point of the program is to check.) For the purposes of this program, a valid `year` is any positive integer. A valid `month` is an integer between 1 and 12, inclusive (1 stands for January). A valid `day` is an integer between 1 and a value appropriate for the given month, inclusive. The months April, June, September and November each have 30 days. February has 28 days except for during leap years when it has 29. The remaining months have 31 days each. A *leap year* is any year that is divisible by 4 but not divisible by 100 unless it is also divisible by 400. For example, the years 2016, 2000, and 2400 are leap years, while 2015, 1900, and 2100 are not.

Helpful Tip for Task 2. If you have a `String` value which contains digits, and you'd like to perform arithmetic on it, you can use a library method called `Integer.parseInt()` which will attempt to convert the `String` into an integer value for you. (Of course, if the `String` you're trying to convert contains non-digit characters, an error will result.) See the code below for an example of its use:

```
String ageAsString = "40";                 //note that "40" is a String
int age = Integer.parseInt(ageAsString); //now age contains the int 40
```

Here are a few sample input dates and the output message that your program should report, respectively:

Date collected	Message to output
-2015/7/6	The date -2015/31/6 is invalid since -2015 is not a valid year.
2015/7/6	The date 2015/7/6 is valid.
2015/31/6	The date 2015/31/6 is invalid since 31 is not a valid month.
2015/6/31	The date 2015/6/31 is invalid since 31 is not a valid day for the given month and year.
2015/2/29	The date 2015/2/29 is invalid since 29 is not a valid day for the given month and year.
2016/2/29	The date 2016/2/29 is valid.

Turn-in. Before the due date listed at the top of this file, please submit via the course Blackboard site one complete `.zip` file named `HW3-jhed.zip` (where you replace 'jhed' with your own personal JHED) containing all `.java` files required for the above tasks. Be sure that all submitted code compiles! If you were unable to complete a task, please include as part of your `.zip` submission a text file named `README` explaining anything you'd like the graders to know.

Need to fix something that you already submitted? At any point up until the due date for this assignment, you may re-submit a complete zip file. The submission most recently received is the one that will be graded. Any earlier submissions will be ignored, so please include *all* files in each submission.