CSE 190D, Winter 2013
Programming Assignment #8: Earthquake/Point (12 points)
Due Monday, March 4, 2013, 11:30 PM

This program focuses on classes and objects. Turn in two files named `Earthquake.java` and `Point.java`.

The assignment has two parts: a client program that uses `Point` objects, and alterations to the `Point` class that we wrote in lecture. You will also need the `cities.txt` input files from the class webpage. Make sure you place all files in the same folder.

Part A (`Point.java`, class of objects):

The first part of this assignment asks you to add methods to the `Point` class we wrote in lecture. The code that we wrote in lecture is linked from the assignments page.
Your `Point` class should implement the following additional methods:

- `public void translate(int dx, int dy)`
  This method should change the point's x position by a given dx and dy amount.
- `public double distance(Point other)`
  Returns the distance between a `Point` and another `Point`. You can access the other point's fields by writing `other.x` and `other.y`. Use the formula: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- `public double distanceFromOrigin()`
  Returns the distance between a `Point` and the origin, (0, 0). Use the same formula as shown above.

None of the methods listed above should print any output to the console.

Part B (`Earthquake.java`, client program):

The second part of this assignment asks you to write a client that uses the `Point` class you wrote in Part A. The goal of Part B is to give you a bit of practice creating and using `Point` objects from a client's perspective and to give you an appreciation for the usefulness of objects in general.

Begin by prompting the user for an input file name, epicenter x, epicenter y and affected radius. Read the cities from the input file with the name the user typed. Use this information to print the cities with "- hit" following the cities closer in distance from the epicenter than the affected radius.

Below is a sample input file and example log of execution from the program; user input is bold and underlined. Your program's output should match this example exactly given the same input. See the course web site for more logs with other input.

cities.txt:

```
6
50 20
90 60
10 72
74 98
5 136
150 91
```

console output:

```
Input file? cities.txt
Epicenter x? 100
Epicenter y? 100
Affected radius? 75
(50, 20)
(90, 60) - hit
(10, 72)
(74, 98) - hit
(5, 136)
(150, 91) - hit
```

You may assume that the user has entered the name of a file that exists and is in valid format. The valid format for a file is a series of lines with the first containing a single number representing the number of the rest of the lines in the file and the rest containing two numbers, the first representing a city's x coordinate and the second representing a city's y coordinate.

Solve this problem using `Point` objects. The methods and behavior of each `Point` object are described above. You can construct a `Point` object as follows:

```
    Point name = new Point();
```

You may want to create an array of `Point` objects. You can construct an array of `Point`s as follows:

```
    Point[] name = new Point[size];
```

Style Guidelines:

For Part B, you are to solve the problem by creating and using `Point` objects as much as possible. This is because a major goal of this assignment is to demonstrate understanding of using objects and defining new classes of objects. Part B should be divided into static methods. For reference, our solution to Part B has two static methods besides `main`.

On both parts of the assignment, you should follow general past **style guidelines** such as: appropriately using control structures like loops and if/else statements; avoiding redundancy using techniques such as methods, loops, and `if/else` factoring; properly using indentation, names, types, variables; and not having lines longer than 100 characters.

You should properly **comment** your code with a proper heading in each file, a description on top of each method, and on any complex sections of your code. Specifically, place a comment heading at the top of each method of the `Date` class, written in your own words, describing that method's behavior, parameters, and return values if any. Your comments should be written in your own words and not taken verbatim from this spec or elsewhere.