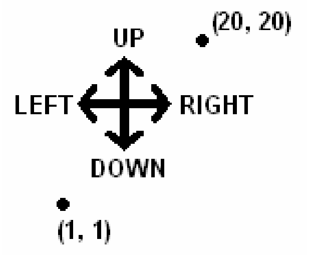
CSE 142, Spring 2007
Programming Assignment #5: Guess 2D (20 points)
Due: Tuesday, May 1, 2007, 4:00 PM

Problem Description:



This assignment will give you practice with `while` loops, random numbers, and using objects. The program allows the user to play a guessing game where the user guesses until a random 2D point is found. Turn in a class named `Guess2D` in a file named `Guess2D.java`.

Program Behavior:

```
This program is a 2-D guessing game.
I will think of a point somewhere
between (1, 1) and (20, 20)
and give hints until you guess it.

Guess x and y: 5 7
You're cold. Go right down
Guess x and y: 18 5
You're cold. Go left down
Guess x and y: 15 2
You're warm. Go left up
Guess x and y: 12 3
You're hot! Go left up
Guess x and y: 11 4
You got it right in 5 guesses!
Play again? y

Guess x and y: 10 10
You're cold. Go left down
Guess x and y: 5 5
You're warm. Go right
Guess x and y: 8 5
You're hot! Go right
Guess x and y: 9 5
You got it right in 4 guesses!
Play again? YES

Guess x and y: 17 10
You're cold. Go left down
Guess x and y: 6 1
You're cold. Go right up
Guess x and y: 9 3
You're warm. Go right up
Guess x and y: 20 20
You're cold. Go left down
Guess x and y: 11 5
You're hot! Go up
Guess x and y: 11 8
You're warm. Go down
Guess x and y: 11 6
You got it right in 7 guesses!
Play again? n

Overall results:
Games played  = 3
Total guesses = 16
Guesses/game  = 5.33
```

In this game, the computer chooses a point between (1, 1) and (20, 20) inclusive. The user is repeatedly asked to guess the point. For each incorrect guess, the program will give the user one of the following three hints:

- hot (a distance ≤ 1.5 from the correct answer)
- warm (a distance ≤ 5.0 from the correct answer)
- cold (a distance > 5.0 away from the right answer)

The game also hints about which direction the user should go from the current guess to find the right answer:

- right (the user should increase x)
- left (the user should decrease x)
- up (the user should increase y)
- down (the user should decrease y)

When the game ends, the program reports how many guesses were needed.

After each game, the program asks the user to play again. You may assume that the user will give a one-word answer. The program should continue playing if the user's response begins with a lower- or upper-case Y. That is, answers such as `"y"`, `"Y"`, `"YES"`, `"yes"`, `"Yes"`, or `"yeehaw"` would all indicate that the user wants to play again. Otherwise, assume that the user does not want to play again. For example, responses such as `"n"`, `"N"`, `"no"`, `"okay"`, `"0"`, and `"hello"` would end the game.

Once the user ends a game and chooses not to play again, the program prints overall statistics about the games played. The total number of games, total guesses for all games, and average number of guesses per game (as a real number rounded to two decimal places) are displayed.

The log of execution on this page demonstrates your program's behavior. Your program may generate different random points, but your output structure should match this one exactly. Your program should present correct information regardless of how many guess(es) were needed or game(s) were played.

Implementation Guidelines:

We suggest that you develop this program in stages. One useful stage is to write a program that plays just one game. You can even put in code to print out the correct answer for debugging purposes, though you should delete such code before you turn in the assignment. The following might be logs of execution for two in-progress stages of the program, though <u>you do not have to develop the program this way and should not turn in a program that prints the answer</u>:

Log of execution from first hypothetical in-progress version (gives cold/warm/hot hints only)	*Log of execution from second hypothetical in-progress version (also gives direction hints and counts guesses)*
`ANSWER IS (11, 4)`	`ANSWER IS (11, 4)`
`Guess x and y:` **`5 7`**	`Guess x and y:` **`5 7`**
`You're cold.`	`You're cold. Go right down`
`Guess x and y:` **`15 5`**	`Guess x and y:` **`15 5`**
`You're warm.`	`You're warm. Go left down`
`Guess x and y:` **`12 3`**	`Guess x and y:` **`12 3`**
`You're hot!`	`You're hot! Go left up`
`Guess x and y:` **`11 4`**	`Guess x and y:` **`11 4`**
`You got it right!`	`You got it right in 4 guesses!`

The hints about guesses being hot, warm, or cold are based on distances between points. The formula to compute the distance between two points is to take the square root of the squares of the differences in x and y between the two points. For example, if the correct point is (11, 4) and the user guesses (5, 7), their distance is $\sqrt{(11-5)^2+(4-7)^2}$ or roughly 6.71, so the hint is "cold". If the user then guesses (12, 3), the distance is $\sqrt{(11-12)^2+(4-3)^2}$ or about 1.41, so the hint is "hot".

You should represent the correct answer and user's guesses as `Point` objects (as shown in Chapter 3) and compute distances between points using the `Point` object's `distance` method. Remember to `import java.awt.*;` to use `Point` in your program. Since performing mathematical operations can introduce tiny errors to doubles, you should not compare the distance to `0.0`, because it might never be equal to `0.0`. Instead, you should check if the user's guess is equal to the correct point.

If the user guesses the number correctly in one try, you can still print the text `"You got it right in 1 guesses"` although the word `"guess"` would be more appropriate. We will not test this case when grading, so either output is fine.

Assume valid user input. When the user is prompted for numbers, the user will type valid integers in the proper range. When the user is prompted to play again, the user will type a one-word string as their answer. To deal with the yes/no response from the user, you may want to use some of the `String` class methods described in Chapters 3 and 4 of the book. To round numbers to 2 decimal places, use the `round2` method shown in previous homework assignments.

For reasons you don't yet understand, do not use the `Scanner`'s `nextLine` method. You have been warned.

Stylistic Guidelines:

Structure your solution using static methods that accept parameters and return values where appropriate. For full credit, you must have at least 2 methods other than main in your program: **a method to play a single game** with the user (not multiple games), and **a method to report the overall statistics** to the user.

You may define other methods if they are useful to eliminate redundancy. If you like, you can place the loop to play multiple games and prompt the user to play another game in your `main` method. As a reference, our solution includes 5 methods other than `main` and `round2` and occupies 105 lines including comments and blank lines.

For full credit, you must define a class constant for the maximum x/y value used in the guessing game. The sample log shows the user making guesses from 1 to 20, but by introducing a constant for 20, you should be able to make the program play the game over any other range starting with (1, 1) just by changing the constant's value. For example, if it is changed to 5, your program picks (x, y) points between (1, 1) and (5, 5). See the course web site for example output.

For this assignment you are limited to the language features in Chapters 1 through 5 of the textbook. Format your code properly as always. Since this program has longer methods than past programs, also put brief comments inside the methods explaining the more complex sections of your code.