CSE 142, Spring 2012
Programming Assignment #6: Bagels (20 points)
Due: Tuesday, May 15, 2012, 11:30 PM

Program Description:

This interactive program focuses on arrays and usage of a `Random` object to generate random outcomes. You will submit a file called `Bagels.java`.

The program plays the game Bagels, a variant of the board game Mastermind. In this game, the computer generates a random number that the user has to guess. The computer provides hints based on the value and position of digits guessed. The solution will always be a number made up of digits 1-9 (no 0). Its length is determined by a class constant and shared with the user in the introduction.

You must exactly reproduce the format and behavior of this sample run (user input is bold and underlined):

```
Welcome to CSE 142 Bagels!
I'm thinking of a 4 digit number.
Each digit is between 1 and 9.
Try to guess my number, and I'll say "fermi"
for each digit you get right, and "pica"
for each correct digit in the wrong place.

Your guess? 1234
bagels
Your guess? 5678
fermi fermi pica
Your guess? 5566
fermi
Your guess? 5587
fermi pica pica
Your guess? 7578
You got it right in 5 guesses.

Do you want to play again? y
Your guess? 1234
fermi pica
Your guess? 5678
pica
Your guess? 2936
You got it right in 3 guesses.

Do you want to play again? NOPE
```

After displaying an introduction, your program should repeatedly prompt the user for guesses and provide a clue after each incorrect guess.

If no digits match between the guess and the answer, the program prints "bagels." The program prints "fermi" for each correct digit in a correct position and "pica" for each correct digit in the wrong position. You may assume that the user always makes a valid guess with the right number of digits from 1-9.

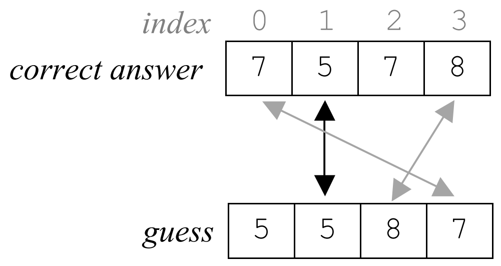
Once the user guesses the answer, the program reports the number of guesses needed and prompts the user to play again.

Because this program uses pseudorandom numbers and user input, you may not be able to exactly recreate the log at left. You must exactly match its format. Please **look at all example outputs on the web site** and do your own testing.

Implementation Details:

Since hints are given based on matching digits, it is helpful to represent the correct answer and a guess as arrays of digits. You can get digits from the user's guess by repeatedly dividing and modding the number by 10. You may want to adapt the `toDigits` method from section on 5/10 to convert a number into an array of digits.

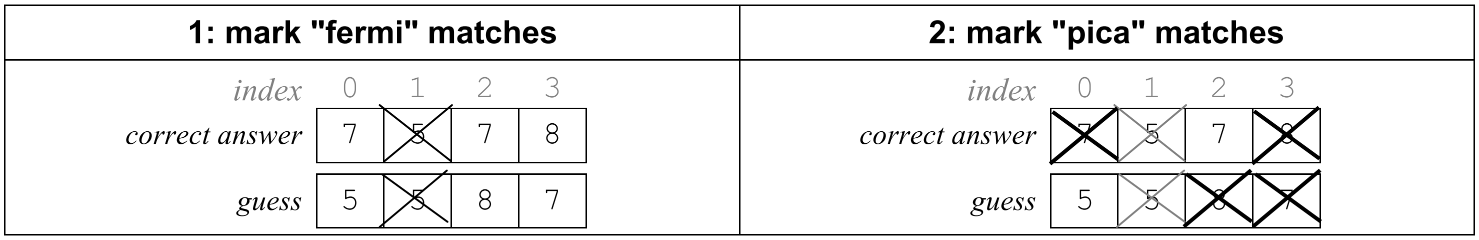
Consider the fourth guess in the first round played above. The following diagram shows the fermi match in black and the pica matches in gray:



Notice that the 5 at index 0 of the guess is neither a fermi match or a pica match; there is a digit 5 in the correct answer, but it is already claimed by its fermi match with index 1. Notice also that the 7 at index 3 of the guess only matches with the 7 at index 0 in the answer and not also with the 7 at index 2.

In this case, the computer's hint is "fermi pica pica." All of the fermi clues are reported first so as not to give any extra information to the player.

To determine fermi and pica hints, we recommend doing two passes over the arrays of digits: the first one to identify fermi matches (matching digits at matching locations) and the second one to identify pica matches (matching digits at different locations). You must be sure that the same digit is never used for two different matches. You will therefore need to mark digits as "used" so that they will not be used again in another match. For example, in the figure below, the numbers with an 'X' through them should no longer be considered:



You must be sure that the process of marking digits doesn't change the correct answer that the user needs to guess. Consider making a copy of it or keeping separate temporary arrays that can safely be modified.

When asking the user whether or not to play again, use the `next` method of the Scanner class to read a one-word answer from the user. Continue playing if this answer begins with the letter "y" or the letter "Y". Notice that the user is allowed to type words like "yes". You are to look just at the first letter of the user's response and see whether it begins with a "y" (either capitalized or not) to determine whether to play again.

Your program should include a **class constant** representing the number of digits for the computer to include in the correct answer. See the course website for additional sample output with different constant values.

Development Strategy:

We suggest starting by writing code to play a single game with the user. Initially, use a fixed value for the correct answer until you get the guessing and clue logic working. Get fermi clues working before attempting pica clues. Then, make the answer randomly generated but print it out at the start of the game so that you can easily evaluate the clues your program is providing.

You will want to use methods from the `Arrays` class. In particular, review `toString`, `copyOf` and `equals`.

Stylistic Guidelines:

This program is more difficult to decompose into methods than previous ones, so you may end up having methods that are longer than 15 lines. You may also want to include more code in main. In particular, you are required to have a while loop in main that plays multiple games and prompts the user for whether or not to play another game. You must have at least 3 non-trivial methods other than `main`, including the following:

- a method to introduce the game
- a method to play exactly one game with the user
- a method to determine and report appropriate hints as described in implementation details

For this assignment you are limited to the language features in Chapters 1-7 shown in lecture or the textbook, although you are not allowed to use the break statement or what are described as "forever loops." Some students try to achieve repetition without properly using while loops, by writing a method that calls itself; this is not appropriate on this assignment. You may want to refer to the **case studies** from chapter 5 and chapter 7.

We will also check strictly for redundancy on this assignment. If you have a very similar piece of code that is repeated in your program, eliminate the redundancy by creating a method, using for loops, etc.

We will once again expect you to use good programming style and to include useful comments throughout your program. We will expect you to make appropriate choices about data types, which if/else constructs to use, what parameters to pass, and so on. Follow past style guidelines about indentation, names, variables, types, line lengths, and comments (at the beginning of your program, on each method, and on complex sections of code). For reference, our solution has around 100 lines without comments and 5 methods other than `main`.