

CSE 142, Summer 2009
Programming Assignment #6: Bagels (20 points)
Due: Friday, August 7, 2009, 11:30 PM

This assignment focuses on arrays. Turn in a file named `Bagels.java`.

Bagels is a variant of the game Mastermind where the computer thinks of a random number and the user tries to guess it. Each digit of the random number is a value from 1 through 9. No 0 digits are included, but the same digit might appear more than once. The player is told how many digits the number has. (We will use a constant for the number of digits.)

Program Behavior:

```
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? 8877  
fermi pica pica  
Your guess? 5587  
fermi pica pica  
Your guess? 5778  
fermi fermi pica pica  
Your guess? 7578  
You got it right in 7 guesses!
```

At left is a sample log of execution (user input is underlined). Your program's output begins with an introduction message that explains the game.

Next your program repeatedly prompts the user for guesses, giving a clue after each incorrect guess. Once the user has correctly guessed the number, the program prints how many guesses were needed. You may assume that the user will always type a valid guess with the right number of digits from 1-9.

In the previous assignment, the player was given clues about whether the right answer was higher or lower than the guess. In this game, the clues are instead based on digits that match between the correct answer and the guess.

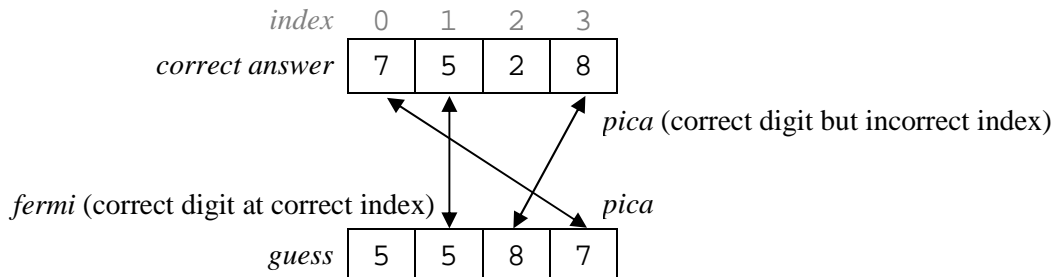
For each digit in the guess that exactly matches the corresponding digit in the answer, the program outputs

"fermi". For each digit in the guess that appears in the answer but at a different index, the program outputs "pica". If the guess does not contain any digits that appear in the correct answer (no "fermi" or "pica" matches), the output is "bagels".

Clues:

It is helpful to think of the correct answer and the guess as arrays of digits. Suppose the correct answer is 7578 and the user guesses 5587 (the fifth guess in the log above). The above guess has one "fermi" match: the digit 5 at index 1. It also has two "pica" matches: the digit 8 at index 2 of the guess matches the 8 at index 3 of the correct answer; and the digit 4 at index 3 of the guess matches the 4 at index 0 of the correct answer.

Notice that the digit 5 at index 0 of the guess is neither a fermi match nor a pica match; there is a digit 5 in the correct answer, but it is already claimed by its fermi match with index 1.



The computer's clue consists of the word "fermi" once for each fermi match (correct digit at correct index), followed by the word "pica" match for each pica match (correct digit, wrong index). All of the fermi clues are reported first so as not to give any extra information to the player. The clue for the above guess would be "fermi pica pica".

Implementation Details:

The main purpose of this assignment is to test your understanding of arrays. Therefore you should use arrays to store the computer's random number and the user's guesses as appropriate, as shown in the diagrams on the previous page. This will be a useful way to represent the numbers so that you can compare them one digit at a time. Recall that you can pull digits off of an integer by repeatedly dividing and modding the number by 10.

The most difficult part of the assignment is comparing the user's guess to the correct answer and providing the correct clue. We suggest that you use a "two phase" algorithm that compares the two arrays of digits in two passes:

- The first pass looks for "fermi" matches, where a digit in the guess is correct and in the correct place.
- The second pass looks for "pica" matches, where a digit in the guess exists in the answer, but at a different index.

The same digit cannot be used for two different matches. For example, using the correct answer of 7578 and the guess of 5587 from the previous page, the digit 5 at index 0 of the guess doesn't match anything, because the digit 5 in the guess is already claimed as part of a fermi match. To implement this functionality correctly, you should find a way to mark individual digits as being "used" so that they will not be used again in another match.

First pass ("fermi pass")	Second pass ("pica pass")
<p><i>index</i> 0 1 2 3</p> <p><i>correct answer</i> 7 5 2 8</p> <p><i>guess</i> 5 5 8 7</p>	<p><i>index</i> 0 1 2 3</p> <p><i>correct answer</i> 7 5 2 8</p> <p><i>guess</i> 5 5 8 7</p>

One way to "mark" a digit would be to change its value. But you should be careful not to damage the original array for the correct answer, so you may want to make a copy of it or implement some way to restore its state later. Another way to "mark" digits would be to keep a separate temporary array of marking values.

Development and Debugging Hints:

The `Arrays` class from `java.util` has several methods that may help you on this assignment. For example:

- `Arrays.toString(array)` converts an array to a printable string. This can be very helpful when debugging to see the contents of an array. For example, to print an array named `a1`, you could write.

```
int[] a1 = {9, 2, 7};
System.out.println("a1 stores " + Arrays.toString(a1)); // a1 stores [9, 2, 7]
```

- `Arrays.copyOf(array, length)` accepts an array and a length and returns a new copy of the original array.

```
int[] a2 = Arrays.copyOf(a1, a1.length);
```

- `Arrays.equals(array1, array2)` accepts two arrays and returns `true` if they contain the same elements.

```
if (Arrays.equals(a1, a2)) { ...
```

A useful trick for debugging is to print out the correct answer at the start of the game. That way you can examine your clues to make sure they are correct for that answer and a given guess. Another idea is to initially use a fixed value for the correct answer (such as 7578 in the log in this document) until you get the guessing and clue logic working properly.

Class Constant (number of digits):

You should have an class constant for the number of digits to use in the computer's random number. The log in this document uses a 4-digit number, but by changing your constant and recompiling, it should be possible to play a game to guess any number of digits. The log below shows a 3-digit game. See the course web site for other sample logs.

```
Welcome to CSE 142 Bagels!
I'm thinking of a 3 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? 123
pica
Your guess? 456
pica
Your guess? 789
bagels
Your guess? 234
fermi
Your guess? 147
bagels
Your guess? 258
fermi
Your guess? 369
fermi
Your guess? 266
fermi fermi
Your guess? 262
You got it right in 9 guesses!
```

Style Guidelines:

Good method structure is important on this assignment. You must use at least **three nontrivial methods besides main**. These methods should use parameters and returns as appropriate. The methods should be well-structured and avoid redundancy. No one method should do too large a share of the overall task. If you are writing methods that use arrays as parameters or returns, see Section 7.1 of the textbook about Arrays and Methods. Chapter 7's Case Study program in textbook section 7.5 is a good example of how to write a larger program with several methods that use arrays.

Your `main` method should be a concise summary of the overall program. It is okay for `main` to contain some code such as `println` statements or the overall game loop. But the `main` method should not perform too large a share of the overall work by itself. Also avoid "chaining," which is when many methods call each other without returning to `main`. For reference, our solution is around 100 lines long and has 4 methods besides `main`, though you don't need to match this.

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 such as by creating a method, by using `for` loops, or by factoring your code.

You are limited to features in Chapters 1 through 7. Follow past style guidelines such as indentation, names, variables, types, line lengths, and comments (at the beginning of your program, on each method, and on complex sections of code).