CS100J Lab 04. Writing simple static methods Fall 2003

Name ____________________ Section time ______________ Section instructor __________________

In this lab, you will gain experience with writing simple static methods. As in the previous lab (and in future labs!), if you are working with a partner you should each get a chance to be "driver" (the one actually typing the code) and "navigator" (the one helping the driver decide what to write).

We have provided the skeleton code for a game called GuessMyNumber. When complete, the program will choose a random number between 1 and n (where n is a number provided by the player) that the player can try to guess. As the player interacts with the program, the program will respond with appropriate messages.

Task 1. Understanding the skeleton code

Download the skeleton program from the web, store it in a directory somewhere, and open it in DrJava. Study the program.We have defined the following variables necessary for the game:

private static Integer myNumber	the random number chosen by the computer
private static int maximum	the random number is chosen in the range 1..n, where n >= 1
private static int numGuesses	the number of guesses attempted by the user
private static boolean playerGuessedMyNumber	true if and only if the player guessed the computer's number correctly. We can say this more succinctly, by saying that its value is the value of this sentence: "the player guess the number correctly"

Note that we call the collection of definitions of these variables the **class invariant**. "Invariant" means "unchanging". The class invariant is true when the variable specifications (listed in the right column of the table) are accurate. We expect this class invariant to be true when the program starts and to remain true before and after each call on a method of the class. So, whenever you are writing one of the method bodies, you can assume that this class invariant is true, and the method body that you write must terminate with the class invariant true again. So you have to continually look at these variables and their definitions when writing method bodies.

We have also defined the following method:

public static void chooseNumber(int n)	Choose a number between 1 and n. If n < 1, then tell the user the number is not good and simply return without choosing a number.

A player who wants to start a game calls this method, given as argument the range of integers in which they are willing to guess. So, for a call chooseNumber(10), the program chooses a number in the range 1..10 and the user will try to guess it. For a call chooseNumber(2), the number to be guessed is in the range 1..2 --i.e. it is either 1 or 2.

Take a look at the body of method chooseNumber and read the beginning of Section 5.6 and section 5.6.1 of the class text (you can do the latter after doing this lab, perhaps back in your room). Those sections tell you about random numbers on the computer and show you how a random double number that is nonnegative and less than 1 is changed into an integer in the range 1..n.

Temporarily, make the four fields of the class public, so you can reference them from the Interactions pane of DrJava while testing. Now, in the Interactions pane, call procedure chooseNumber a few times and see what values it puts in field myNumber. This way, you will get an idea how the method is working.

Note that since the method and variables are static, you don't have to create a GuessMyNumber object.

Task 2. Guessing a number

The player needs a way to tell the computer the number that the player guesses. Implement a procedure called guess(int), and make sure it satisfies the following points:

- It should print an error message and return if the computer has not chosen a number yet. (*Hint: Why do you think we made myNumber type Integer instead of type int?*).
- It should print an error message and return if the player has already guessed correctly. (*Hint: Use class variable playerGuessedMyNumber*).
- It should make sure the guess is in the range of the game (1..maximum). If not, print a message indicating what the range is and that the guess isn't being counted.
- It should print a message saying whether the guess is correct, too high, or too low (and count this guess).

The actual messages are up to you, but they should be something like the following:

> *"That's right! My number is <myNumber>"*
> *"Sorry, my number is lower than <the player's guess>"*
> *"Sorry, my number is higher than <the player's guess>"*
> *"The computer hasn't chosen a number yet!"*
> *"You've already guessed the correct number!"*

The messages shouldn't mention the private variables by name, because the player doesn't know about them.

Before you write the method body, type in the specification-comment for the method and the method header. The specification must be precise and thorough. It should say WHAT the method does. It shouldn't mention the private variables by name, because the player doesn't know about them.

Before beginning the method body, compile the program to make sure the header (and braces { }) are syntactically correct.

Write and test the method body incrementally! For example, above, we listed three tasks the method should perform. Write code for the first and then test what you have done by writing appropriate calls on the method. Only when you are satisfied that that part is correct should you go on to the next one. This business of incremental coding and testing is extremely important.

Task 3. Rating the player

After guessing correctly, the played may want to know how well they played the game. Implement a procedure rateMe(), which satisfies the following points:

- If the player has not yet guessed correctly, the method prints an error message and returns.
- If the player guessed correctly, the method prints a message to the player telling them how many guesses they attempted, along with some additional message specific to the number of guesses.

Of course, you do not have to print a special message for every possible number of guesses; there could be infinitely many! Instead, form rating groups such as [1-2 guesses], [3-4 guesses], [5-6 guesses], and [7 or more guesses]. Use class variable numGuesses to determine to which group the player belongs and print a message based on that.

For example, you could print the following:

> *"You're amazing! You needed only <numGuesses> guesses!"*

"You're a really good guesser! You needed only <numGuesses> guesses!"
... and so on.

Task 4. Play the game using the Interactions pane of DrJava! :)

When your two methods are finished, and you tested enough to believe they are correct, then make all fields private again. Then, to play a game, do this:

- Call procedure chooseNumber(int) to tell the computer to pick a new number.
- Call procedure guess(int) to guess what the number is.
- Call procedure rateMe() to get your rating message!

Don't forget to show your TA or consultant the code you wrote!

Extra Work (for those who finish early) ***This part is not required; attempt it ONLY if you have finished the stuff above.***

Interfacing with this program is not so nice. You have to write a method call with an argument all the time. The CD ProgramLive contains a GUI, called JLiveWindow, that can provide a better interface. The purpose of this extra work is to introduce you to it and have you implement the same game but using the better interface.

(a) Turn to lesson page 1-5 of ProgramLive (the CD) and listen to activity 3, which is about the GUI JLiveWindow. Notice that method buttonePressed is called whenever the ready button is pressed in the GUI. You will change method buttonPressed to help you play the guessing game.

(b) Read the information on using the GUI (a footnote on lesson page 1-5) if you wish. Then get the two files JLiveWindow.java and MyJLiveWindow.java from a footnote on lesson page 1-5 and place them in a new directory somewhere on your harddrive. Then open MyJLiveWindow in DrJava (you don't have to open the other one).

(c) Scroll down until you see method main. Change the arguments in the call

```
MyJLiveWindow(0, 4, 0)
```

to

```
MyJLiveWindow(2, 0, 2)
```

(c) Copy ALL fields and methods from your correct game that uses the Interactions pane into class MyJLive window and make all the METHODS non-static. Leave the FIELDS static.

(d) Compile to make sure that it is syntactically correct. In the interactions pane, type this: MyJLiveWindow.main(null); This will start the progam going and the GUI should appear in the upper left window, with two integer fields and two string fields. If it is going right, proceed to the next step.

(e) Study this: how the integer fields of the GUI will be used.

> To start a new game using the range 1..n, put 0 in integer field 0 and n in integer field 1.
>
> To make a guess, put 1 in integer field 0 and the guess in integer field 1
>
> To rate yourself, put 2 in integer field 0.

Here is how the string fields will be used.

> If the output is one line of information, put that information in String field 0. This means changing a call on System.out.print or println to a call on setStringField with first argument 0.

If the output is on two lines of information, put the first line in String field 0 and the second line in String field 1.

(f) You have to change methods rateMe, guess, and chooseNumber so that they put their output in the GUI instead of in the Java console. To this end, change each System.out.println(s); statement to either setStringField(0,s); or setStringField(1,s), depending on whether it's the first or second line of output. This may require a bit of fiddling here and there. COMPILE AFTER EACH ONE THAT YOU CHANGE.

(g) Now comes the last piece. You have to change method buttonPressed to read in integer field 0 and call a method depending on what was in the field. Below is what method buttonPressed should do --we have done part of it for you, and you can copy and paste. After copying and pasting, please indent properly.

```
int which= getIntField(0); // 0 is new game, field 2 contains n

// Take care of a new game
if (which == 0) {
chooseNumber(getIntField(1));
setIntField(0, 1);
return null;
}
// Take care of a guess (you fill this in)


// Take care of rate me (you fill this in)


// Take care of bad input
setStringField(0, "input in int field 0 is no good.");
setStringField(1, "0 for new game, 1 for a guess, 2 for rate me");

return null;
```

(h) Now play the game!

Extra Work (for those who finish early) ***This part is not required; attempt it ONLY if you have finished the stuff.***

In rating a player, 7 or more guesses is considered bad. But that doesn't take into account the size of the range. If the range is 1..10000, 7 guesses might be good! If the range is 1..2, 7 guesses is truly bad --the player had to type the wrong guess 8 times! Figure out a method for determining a good number of maximum guesses based on the range of the random numbers generated.