COSC102 Guessing Games Fall 2014

For this assignment, you implement a guessing-game in Java. The game gives users two options: in one the computer picks a number that the user subsequently tries to guess, in the other the user picks and the computer guesses. The first option is in the Javanotes textbook, the second is its "reverse", or dual.

Starting Menu

The program starts with a menu asking the user to play by selecting an option or to exit the program. The menu items can be

- User guesses
- Computer guesses
- Quit

(Feel free to add simple ASCII graphics to make the starting menu like a retro splash screen.)

If the user chooses to play, the chosen option starts. When the play is complete the game returns to the starting menu allowing the user to play again or exit.

When the User Guesses

Play starts with the user picking a range: two integers are entered in any order, one the lowest number in the range, the other the highest.

The computer then secretly picks a random integer from the range. Convince yourself that the following line of code picks any integer in the range, including the end points, with equal probability.

```
int pick = ( (int) ( Math.random() * (high - low + 1) )) + low;
```

The computer then enters the following loop.

1. Get and check the user's guess, which must be an integer in the range.
2. If the guess is correct, play is finished; the number of guesses required is printed; and the starting menu displayed.
3. If the guess is incorrect, the computer tells the user if it is too high or too low.
4. Continue with step 1.

When the Computer Guesses

Play starts with the computer asking the user for a range, after which the user secretly thinks of a number in that range.

The computer then enters the following loop.

1. The computer guesses an integer in the range.
2. If the guess is correct, play is finished; the number of guesses required is printed; and the starting menu displayed.
3. If the guess is incorrect, the user tells the computer whether the guess is too high or too low.
4. Continue with step 1.

How should the computer generate its guesses? Starting at the beginning of the interval and incrementing one by one is not efficient. The computer would be lucky to guess the number in the first few tries. If the interval has n numbers, as many as n guesses might be necessary. It is easy to see that in the average case $n/2$ guesses is necessary.

Binary-search is much more efficient. Each time the middle number is guessed user feedback rejects half of the remaining numbers. This guarantees no more than approximately $\log(n)$ guesses, where log() is the base-2 logarithm, are needed.

So, for an interval of 100 numbers, the linear search could take as manay as 100 guesses, but binary search should never take more than 7 guesses. We will use this type of analysis later in the course when we talk about "efficiency", which usually mean the worst-case amount of running time or data memory used by the procedure.

Helper Functions

To get keyboard input from the user a `Scanner` object is used, made direclty available by the import statement

```
import java.util.Scanner;
```

As the `Scanner` is a resource that uses memory, it is a good practice to use one Scanner object per input stream. Thus the best declaration is to use a class member variable

```
static final Scanner scan = new Scanner(System.in);
```

Add helper functions so that the program is resistant to invalid input (i.e. data type and range).

Grading

Your submission will be graded on both correctness [80%] and style [20%]. Therefore, test your program thoroughly and ensure that your code is readable and well formatted (formating must follow textbook conventions and recommended style). Don't be grossly inefficient, but there's no need to over-optimize.

Submission

Include a header in your source code file that contains

- your student information,
- a short description of the program,
- acknowledgement crediting people and resource(s) that helped you, and
- if you have not fully implemented the assignment a description of each part missing.

Submit your *source code* file using the upload link on Moodle. Remember, it is the text file that contains your Java code and has a `.java` extension (although this extension may not be visible on all computers). Just make sure you do not submit your `.class` (compiled bytecode) or `.java~` (backup) file.

Your submitted file should contain a main function that, when invoked, begins the program as described above. You are expected to use helper functions and methods to structure your code and to include comments when required.

Submit early and often. Late assignments are not accepted, so make sure you have some version of the source code uploaded to Moodle. You can replace the uploaded file with a more up-to-date version until the deadline.

Tips

Plan out your program *before* you start coding: "Minutes of thinking will save you hours of coding." Consider writing an outline of your program using a combination of comments and function declarations for the different parts of the assignment as you see them. Then implement systematically part by part.

Compile often. If you submit a program that does not compile, then there are bugs and the program cannot be tested. A program that does not compile automatically receives a low grade. To prevent this, work on your code in small sections. Compile as you are working on each section to check that there are no syntax errors. A section can be as little as two or three lines of code. By testing small sections of code before moving on, you are preventing a massive debugging exercise to finish what you think is a fully working program. Design your program incrementally, consider the smallest tasks you have to do (printing messages, getting input, picking a random number, etc.) write and test each. Once they are right, you can put them together to form your whole program.

You can add print statements for the purposes of debugging to check the values of integers in your code. Comment these out before your final submission.