Java Journal Template

Directions: Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

Name:Alexander Ahmann

Date: 14 Jan., 2023

Final Replit Program Share Link: https://replit.com/@Alekseyyy1/Number-Guesser

Complete the following template. Fill out all entries using complete sentences.

PART 1: Defining Your Problem

Task
State the problem you are planning to solve.

Requirements
- Describe any input data you expect to use.
- Describe what the program will do to solve the problem.
- Describe any outputs or results the program will provide.

Inspiration
When writing your entry below, ask yourself the following questions:
- Why do you want to solve this particular problem?
- What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source?
- Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue?
- What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete?

The problem I am trying to solve is to create a simple “number’s guessing” game where the programme generates a random number between 1 and 100, stores it in an integer variable, and challenges the player to guess the random number. It initiates a while loop, takes input from the user that is presumed to be an integer, compares it to the generated random number, and congratulates the user if they managed to guess the number correctly.

If the user did not guess the number correctly, it will inform the user whether or not the guess is larger or smaller than the generated random number, and ask for another guess. Winning the game will break the for loop. There will also be a counter that will increment by one (1) every time the programme will ask for a guess.

PART 2: Working Through Specific Examples

Task
Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.

Requirements
Complete the three steps below **for at least two distinct examples/scenarios**.

- State any necessary input data for your simplified problem.
- Write clear and specific steps in English (not Java) detailing what the program will do to solve the problem.
- Describe the specific result of your example/scenario.

Inspiration
When writing your entry below, ask yourself the following questions:

- Are there any steps that you don't fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots.
- Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don't understand. Rewrite these parts as clearly as you can.
- Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work?

Scenario 1: Player guesses correctly:

1. Generate a random number
2. Ask for the user's input
3. The user gives a correct input
4. The programme prints out a congratulations message and number of times that it asked for a guess
5. The while loop gets terminated

Scenario 2: Player guess is too large:

1. Generate a random number
2. Ask for the user's input
3. The user gives an integer too large
4. The programme prints out a message informing the user that their guess is too large and to try again.

Scenario 3: Player guess is too small:

1. Generate a random number 2. Ask for the user's input 3. The user gives an integer too small 4. The programme prints out a message informing the user that their guess is too small and to try again.

PART 3: Generalizing Into Pseudocode

Task
Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.

Requirements
- Write pseudocode for the program in English but refer to Java program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.

Help With Writing Pseudocode
- Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it, other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert to code later.
    - https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/
    - https://www.wikihow.com/Write-Pseudocode

Inspiration
When writing your entry below, ask yourself the following questions:
- Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well.
- Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them.
- When you are finished with your pseudocode, does it make sense, even to a person that does not know Java? Aim for the clearest description of the steps, as this will make it easier to convert into program code later.

```
Function PrintMessage():
    Print out welcome message

Function MainLogic(arguments):
    PrintMessage()
    Integer magicNumber <- Random integer between 1 and 100
    Integer countGuesses <- 0

    Boolean done = False
    while not done:
        x <- GetInput("Guess: ")
        Increment countGuesses by 1
        If x = countGuesses Then
            Print out congratulations message and number of guesses needed to get the correct number
            done <- True
```

```
Else If x > magicNumber Then
   Print out that the guess is too large
Else If x < magicNumber Then
   Print out that the guess is too small
End If
```

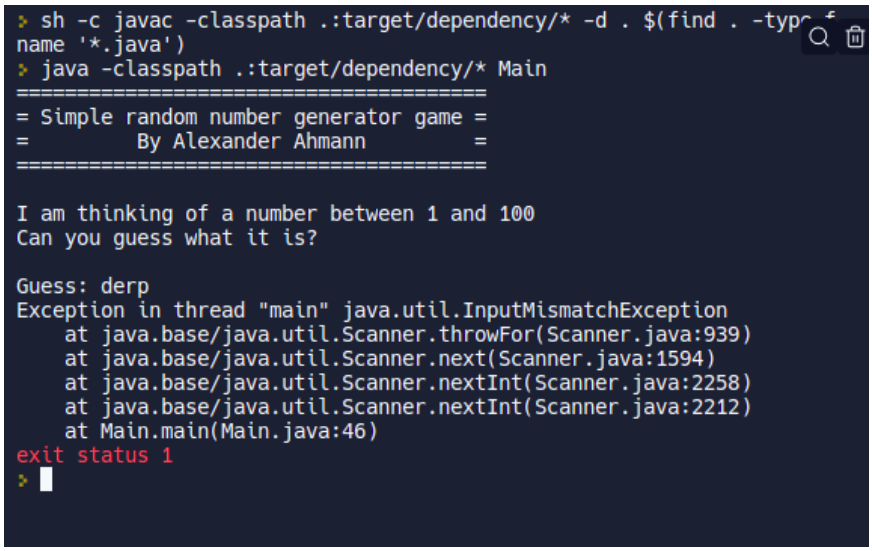
PART 4: Testing Your Program

Task
While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.

Requirements
- For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.

For each error that occurs while writing and testing your code:
- Record the details of the error from Replit. A screenshot or copy-and-paste of the text into the journal entry is acceptable.
- Describe what you attempted in order to fix the error. Clearly identify which approach was the one that worked.

Inspiration
When writing your entry below, ask yourself the following questions:
- Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program.
- Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa?
- Did the outcome look the way you expected? Was it formatted correctly?
- Does your output align with the solution to the problem you coded for?



Here, I was prompted to add a try-catcc block to handle any bad input:

```
try {
    System.out.print("Guess: "); // asking for the user's guess
    int inputGuess = inputMechanism.nextInt(); // parse the user's input as an integer
    countGuesses++; // increment the countGuesses by 1

    [... snip ...]
}
catch  (InputMismatchException ex) {
        System.out.println("Input is not a valid number.");
        inputMechanism.nextLine(); //clear out the input buffer
}
```

PART 5: Commenting Your Program

Task
Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.

Requirements

- The purpose of the program and each of its parts should be clear to a reader that does not know the Java programming language.

Inspiration
When writing your entry, you are encouraged to consider the following:

- Is each section or sub-section of your code commented to describe what the code is doing?
- Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer.

NumbersGuessing.java:

```
/**
 * Classic game that challenges the user to guess the random number
 * By Alexander Ahmann <alexander.ahmann@snhu.edu>
 */

import java.util.*;

public class NumberGuessing {

  /**
   * Function to print a welcome message into the terminal
   */
  public static void printMessage() {
    System.out.println("=======================================");
    System.out.println("= Simple random number generator game =");
    System.out.println("=         By Alexander Ahmann         =");
    System.out.println("=======================================");
    System.out.println("\nI am thinking of a number between 1 and 100");
    System.out.println("Can you guess what it is?\n");
  }

  /**
   * The Main Logic of the Numbers Guessing Game
   */
  public static void main(String[] args) {

    // Declaring the classes with the functionality needed by the game
```

```
    Random random = new Random(); // for random number generation
    Scanner inputMechanism = new Scanner(System.in); // for

    printMessage(); // Print the welcome message

    int magicNumber = random.nextInt(99) + 1; // The target random number that the user has to guess
    int countGuesses = 0; // The numbers of guesses that it took for the user to guess the
magicNumber

    boolean done = false; // the state of the game's while loop

    /**
     * This is the while loop for the game. It asks for an input and will
     * break if the user can input a number that matches the magicNumber.
     */
    while (!done) {
      /**
       * After some rudimentary testing, I added a try-catch exception
       * handling mechanism to handle bad inputs regarding guessing.
       */
      try {
        System.out.print("Guess: "); // asking for the user's guess
        int inputGuess = inputMechanism.nextInt(); // parse the user's input as an integer
        countGuesses++; // increment the countGuesses by 1

        /**
         * This conditional compares the user's guess (inputGuess) and the
         * magicNumber. The game ends if inputGuess equals magicNumber.
         */
        if (inputGuess == magicNumber) {
          System.out.printf("Correct! The random number is %d. It took you %d guesses to guess the
random number!\n",
              magicNumber, countGuesses); // Congratulate the user for correctly guessing the
magicNumber
          inputMechanism.close(); // "Cleanup" by closing the Scanner class.
          done = true; // Break the loop.
        } else if (inputGuess > magicNumber)
          System.out.println("Your guess is greater than what I'm thinking."); // Incorrect guess because
the input is
                                                    // larger than the magicNumber, inform
                                                    // the user that the number is too big.
        else if (inputGuess < magicNumber)
          System.out.println("Your guess is less than what I'm thinking."); // Incorrect guess because the
input is
                                                  // smaller than the magicNumber, inform the
                                                  // user that the number is too small.
```

```
        else
          continue; // this should not execute
      }
      /**
       * Print out an error message in the case of a bad input
       */
      catch (InputMismatchException ex) {
        System.out.println("Input is not a valid number."); // prints error message
        inputMechanism.nextLine(); // empty out the input buffer
      }
    }
  }
}
```

PART 6: Your Completed Program

Task
Provide the Replit link to your full program code.

Requirements
- The program must work correctly with all the comments included in the program.

Inspiration
- Check before submitting your Touchstone that your final version of the program is running successfully.

https://replit.com/@Alekseyyy1/Number-Guesser