Project MindNumber

Collaboration: Solo. Complete this project by yourself with optional help from section leaders. Do not work with anyone else, do not copy any code directly, do not copy code indirectly through reading a computer screen, and do not give your code to anyone. Begin early.

Turnin: Submit three well-tested required functions described below to WebCat for the bulk of the score. You will see your score, be given some hints, and will be allowed to submit as often as you desire until you have 100% for 60 /100 points. Include the actual game that we will grade by hand by running it as a Java Application.

Goals
In this assignment, you are going to implement a number guessing game that is often known as the game of Mastermind. This assignment will give you more experience with

1. Strings  
2. User input  
3. If statements  
4. While statements  
5. Testing with JUnit  
6. Problem solving  
7. Submitting projects to WebCat and learning how to read feedback to get a perfect score

First Get the start of the Eclipse Project

Download this archive file http://www.cs.arizona.edu/~mercer/Assignments/MindNumberStart.zip and import it into Eclipse like this:

- Select File > Import > General > Existing Projects into Workspace > Next
- Click the radio button Select Archive File and then click the Browse button to the right
- Browse to where you downloaded MindNumberStart.zip, click Open, click Finish
- You will see many errors in MindNumberTest.java. Hover your cursor over any @Test and select the quick fix Add JUnit 4 library to the build path

There are several files including the start of a unit test and the method stubs in MindNumber.java. Here is a preview of what you will do:

1. Write assertions to test your code  
2. Change three methods from stubs to correct methods  
3. Implement the game of MindNumber in a main method so anyone with your program can play the game  
4. Optionally, play the game with the GUI

Using a Main Method to Play the Game (only after you get three methods working)

To first give you an overall feeling for the finished game, we first present a dialog to show how the game will be played. The user will first be prompted to enter a -1 to let the secret number for the program itself pick a secret number. You will also allow users to enter their own secret number, which is done for easier testing of your game (it really helps testing when you know the
A program generated secret number is a five digit number stored as a String as "01234" or "92745" but not "1234", "12344", or "123z7". This number can be generated using the existing well-tested method `public String generateSecretNumber()`.

The game should then prompt the “game player” to guess the number. The input is error checked only to ensure the user enters a string of length 5. Entering "what?" will not help at all, but it should be allowed and should count as a try at the secret number. If the user enters "123456" or "1234", notify the user they must enter 5 digits but do not count this as an attempt at the secret number.

The game rules insist that you give the player some feedback on guesses. Based on that feedback, the player makes more guess. Guessing continues until the “secret” number is guessed or until the maximum number of tries (32) is reached. Here is one sample dialog that plays one game where we, the users, actually input the 'secret' number as 12345. Later on, enter -1 to let the real MindNumber experiences begin.

Enter the secret number as a 5 digit number where all digits are unique, or enter -1 to have this program select the secret number.

Select your own valid secret number or enter -1: what? 'what?' is not a valid secret number.
Select your own valid secret number or enter -1: 123456 '123456' is not a valid secret number.
Select your own valid secret number or enter -1: 12233 '12233' is not a valid secret number.
Select your own valid secret number or enter -1: 12345

Enter your 5 digit guess: 11111
Try number: 1
Digits found: 1
Correct position: 1

Enter your 5 digit guess: 22222
Try number: 2
Digits found: 1
Correct position: 1

Enter your 5 digit guess: 99999
Try number: 3
Digits found: 0
Correct position: 0

Enter your 5 digit guess: 12333
Try number: 4
Digits found: 3
Correct position: 3

Enter your 5 digit guess: 12344
Try number: 5
Digits found: 4
Correct position: 4

Enter your 5 digit guess: what?
Try number: 6
Digits found: 0
Correct position: 0

Enter your 5 digit guess: 123456
'123456' must have a length of 5
Enter your 5 digit guess: 54321
Try number: 7
Digits found: 5
Correct position: 1

Enter your 5 digit guess: 12345
Try number: 8
Digits found: 5
Correct position: 5

You won in 8 tries!

**TopDown Design**
Before implementing the game method, you are asked (okay you are required if you want full credit for this assignment) to write three well-tested methods first that will make writing the actual game itself much easier with far fewer bugs. These three methods listed next should be tested well and have the same exact method signatures. They are all given in class MindNumber as method stubs (they compile but don't work). Sample assertions are included below each method heading to help explain the methods use and behavior (these same assertions are included in starter project MindNumberStart.zip).

1) isValid

```java
/**
 * A secret number is valid if it 5 digits long and contains no duplicates
 * and only contains the digits '0', '1', '2', ..., '9'
 *
 * @param str The secret number being tested for validity
 * @return true if the number is valid
 */
public boolean isValid(String str)
```

// Sample assertions (not a complete test)
```java
MindNumber m = new MindNumber();
assertTrue(m.isValid("01987"));
// These three examples are not valid secret numbers:
assertFalse(m.isValid("1234")); // length is not 5
assertFalse(m.isValid("12x34")); // chars must be digits, 'x' is not
assertFalse(m.isValid("12933")); // duplicates not allow
```

2) uniqueDigitsFound

```java
/**
 * Return the number of digits that are contained in both the secret number and
 * the guess. For example when secretNumber: 12345 and guess: 67821, the two
 * numbers share two digits: 1 and 2.
 *
 * @param secretNumber the secret number the user is trying to guess.
 * @param guess the number the user is comparing to the secret number
 * @return the amount of digits the both the secret number and the guess contain.
 */
public int uniqueDigitsFound(String secretNumber, String guess)
```

// Sample assertions (not a complete test)
```java
MindNumber m = new MindNumber();
assertEquals(5, m.uniqueDigitsFound("12345", "21435"));
assertEquals(0, m.uniqueDigitsFound("12345", "67890"));
```
2) foundInPosition

/**
 * Returns the number of matching digits between the guess and the secret number. For example when secretNumber is "12345" guess is "12675"
 * returns 3 as the 1, 2 and 5 all have the same value at the same location.
 *
 * @param secretNumber
 *   the secret number the user is trying to guess
 * @param guess
 *   the users guess at the secretNumber
 * @return the number of digits in the guess that match the secretNumber
 */
public int foundInPosition(String secretNumber, String guess)

// Sample assertions (not a complete test)
MindNumber m = new MindNumber();
assertEquals(1, m.foundInPosition("12345", "99399"));
assertEquals(3, m.foundInPosition("12345", "19395"));

generateSecretNumber (provided in the starter project)

This method and its full tests are given to you in the starter project. You won’t need it until you implement the console version of the game.

/**
 * Generates a 5 digit, valid secret number. A secret number is valid
 * if it contains no duplicates and all five characters are digits '0'..'9'
 *
 * @return a valid secret number
 */
public String generateSecretNumber() // Implemented and tested for you!

Tips/Hints:

1) The console game cannot allow more than 32 attempts to guess. At the point, the player loses.
2) Manipulation of the guess (the users input) is best done as a string (don’t convert it to an int)
3) The user may enter anything as a guess such as what? as long as it does not affect the game or terminate the program. Count this is a try
4) report an error if the guess is not a length of 5. Do not count this as a try.
5) When the user enters the secret number (rather than -I), the program must check to make sure that is a valid 'secret' number: Five unique digits.
6) Some useful String methods:
   a. int length() how many characters in this String, returned as an integer
   b. char charAt(int index) the character at the specified index into this String
   c. int indexOf(String sub) returns the index of sub in this String or -1 if not found
   d. boolean contains(String other) return true if other is part of this String.
   e. boolean equals(String other) return true if other is exactly the same as this String.
7) Begin your game like this in a file Named MindNumberMain.java

import java.util.Scanner;
public class MindNumberMain {

    public static void main(String[] args) {
        System.out.println("Enter the secret number as a 5 digit number");
        System.out.println("where all digits are unique, or enter -1");
        System.out.println("to have this program select the secret number.");

        MindNumber minNumFun = new MindNumber();

        Scanner keyboard = new Scanner(System.in);
    }

    // Grading Criteria 100 pts max

    ___+70 Web-Cat correctness and method coverage. Your code must compile using the specified
    names, Rick's tests pass, and you have tested all of your methods with assertions in a test
    method. These 40 points are derived from Web-Cat. You may submit an unlimited number of
    times. The final submission will be the one that is graded. Notice that a multiplication feature is
    employed that means 90% problem coverage and 90% code coverage results in 0.9 * 0.9 = 0.81.
    0.81 * 40 pts = 32.4 pts out of the maximum of 40.

    ___/ +20 MindNumberMain.java is turned in with your project and has the code to run the
    game using standard IO. The dialog is very close (if not exactly the same) to what you see at the
    beginning of this document. It must use the three methods you implemented along with the given
    methods in MindNumber.

    ___+10 pts Style and Design:
    • 1pt You named the program that runs the game MindNumberMain.java
    • 2pts You added three methods to MindNumber.java
    • 2pts You have your name as a comment at the top of the all files
    • 3pts You used meaningful identifiers
    • 2pts All code is formatted consistently (use Eclipse’s Source > Format)

    Reasons you can get 0 on WebCat:
    • You did not submit any tests. You need at least one file with the name Test at the end as
      in MindNumberTest.java and there must be @Test methods in that file
    • One of your assertions does not pass when run on WebCat
    • Our tests exposed a case where you have an infinite loop on WebCat. Look for Timeout
      errors on WebCat. Debug your code. Looks at all loops
    • You have an incorrect method signature. Your tests run correctly on your machine, but
      Rick's reference tests on WebCat cannot find the expected method.