Assignment MindNumber

Collaboration Solo! Complete this assignment yourself. 50pts

Turnin: Submit the program to WebCat for testing of the required methods described below. You will see your score, be given some hints, and will be allowed to submit as often as you desire until you have 100%.

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 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 over @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 so anyone with your program can play the game

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

To give you an overall feeling for the 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 'secret' number that you seek). A program generated secret number is a five digit number stored as a String as in "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!
Functional Decomposition

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) uniqueDigitsFound

/**
 * 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)
    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"));

3) isValid

/**
 * 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)
    MindNumber m = new MindNumber();
    assertTrue(m.isValid("01987"));
    // These three examples are not valid secret numbers:
```java
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
```

**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.

```java
/**
 * 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 as 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 -1), 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.

**Grading Criteria 50 pts max**

__/ +10 MindNumberMain.java has the code to run the game using standard IO and the dialog is very close to what you see at the beginning of this document.

__+40 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.

**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 (you may be using the wrong version of Java)
* 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 tests cannot find the expected method. Consider the following example:

The following feedback shows lots of green. But do you see the Total Score is 0.0/100.0? Look hard because it is small (and the green is so dominant)!

Scroll Down a bit to see that `numberOfPairs` was expected in your project, but the symbol `numberOfPairs` was not found. This user in this case named that method `numberOfPairs` (note the lower case `o`)
Code Coverage from Your Tests (100%)

* Estimate of Problem Coverage (Unknown!)

Problem coverage: unknown

Web-CAT was unable to assess your test cases.

For this assignment, the proportion of the problem that is covered by your test cases is being assessed by running a suite of reference tests against your solution, and comparing the results of the reference tests against the results produced by your tests.

Your code failed to compile correctly against the reference tests.

This is most likely because you have not named your class(es) as required in the assignment, have failed to provide one or more required methods, or have failed to use the required signature for a method.

Failure to follow these constraints will prevent the proper assessment of your solution and your tests.

The following specific error(s) were discovered while compiling reference tests against your submission:

```
cannot find symbol
  symbol : method numberOfPairs(java.lang.String[])
  assertEquals(1, studentsFan.numberOfPairs());

cannot find symbol
  symbol : method numberOfPairs(java.lang.String[])
  assertEquals(2, studentsFan.numberOfPairs());

cannot find symbol
  symbol : method numberOfPairs(java.lang.String[])
  assertEquals(1, studentsFan.numberOfPairs());

cannot find symbol
```