Design Pattern: Singleton Class: Input.java

```
import java.util.Scanner;

/**
 * <i>Singleton</i> class to assist with basic keyboard input operations. Only a single <i>Input</i> object will ever be created. The <i>Input</i> class clusters related input operations and will build one <i>Scanner</i> object to use with input.
 * <i>Scanner</i> objects are big and complex; thus, this approach reduces the overhead associated with the repeated creation of <i>Scanner</i> objects.
 *
 * @author Rex Woollard
 * @version Lab Assignment 1: <i>Lord of the Rings</i>
 */
public final class Input {
   /** Keyword <i>static</i> makes this a <i>class</i> oriented variable, rather than <i>object</i> oriented variable; only one instance of an <i>Input</i> object will ever exist, and the instance is tracked by this reference variable. */
   private static Input referenceToSingleInputObject = null;
   /** Object-oriented instance variable, but since only one <i>Input</i> can ever be created, only one <i>Scanner</i> object will ever be created. */
   private Scanner scannerKeyboard;

   /** A <i>private</i> constructor guarantees that no <i>Input</i> object can be created from outside the <i>Input</i> class; this is essential for the <i>singleton</i> design pattern. */
   private Input() {   scannerKeyboard = new Scanner(System.in); }

   /** The <i>static</i> modifier means this method can be called without the existence of an <i>Input</i> object; if no <i>Input</i> object exists one will be created; if one already exists, it will be re-used. */
   public static Input getInstance() {
      if (referenceToSingleInputObject == null)
         referenceToSingleInputObject = new Input();
      return referenceToSingleInputObject;
   } // end static Input getInstance()

   /**
    * Presents a prompt to the user and retrieves an <i>int</i> value.
    * @param sPrompt   reference to a <i>String</i> object whose contents will be displayed to the user as a prompt.
    * @return <i>int</i> value input from keyboard
    */
   public int getInt(String sPrompt) {
      System.out.print(sPrompt);
      while ( ! scannerKeyboard.hasNextInt()) { // peek into keyboard buffer to see if next token is a legitimate int
         System.out.println("Number is required input.");
         System.out.print(sPrompt);
         scannerKeyboard.nextLine(); // clear bad input data from the keyboard
      }
      return scannerKeyboard.nextInt();
   } // end int getInt(String sPrompt)
```

```
   /**
    * Presents a prompt to the user and retrieves an <i>int</i> value which is within the range of <i>nLow</i> to <i>nHigh</i> (inclusive).
    * @param sPrompt reference to a <i>String</i> object whose contents will be displayed to the user as a prompt.
    * @param nLow lower boundary on the range of legitimate values
    * @param nHigh upper boundary on the range of legitimate values
    * @return <i>int</i> value input from keyboard
    */
   public int getInt(String sPrompt, int nLow, int nHigh) {
      int nInput;
      do {
         System.out.printf("%s (%d-%d): ", sPrompt, nLow, nHigh);
         while ( ! scannerKeyboard.hasNextInt()) { // peek into keyboard buffer to see if next token is a legitimate int
            System.out.println("Number is required input.");
            System.out.print(sPrompt);
            scannerKeyboard.nextLine(); // retrieves input to the next \r\n (line separator) and we choose to ignore the String that is created and returned
         }
         nInput = scannerKeyboard.nextInt();
         if (nInput >= nLow && nInput <= nHigh) // int value is within range, thus it is valid . . . time to break out of loop
            break;
         System.out.println("Value out of range. Try again.");
      } while (true);
      return nInput;
   } // end int getInt(String sPrompt, int nLow, int nHigh)

   /**
    * Presents a prompt to the user and retrieves a <i>reference-to-String</i>.
    * @param sPrompt reference to a <i>String</i> object whose contents will be displayed to the user as a prompt.
    * @return <i>reference-to-String</i> object created by keyboard input
    */
   public String getString(String sPrompt) {
      System.out.print(sPrompt);
      scannerKeyboard.useDelimiter("\r\n"); // Setting this delimiter ensures that we capture everything up to the <Enter> key. Without this, input stops at the next whitespace (space, tab, newline etc.).
      String sInput = scannerKeyboard.next();
      scannerKeyboard.reset(); // The preceding use of useDelimiter() changed the state of the Scanner object. reset() re-establishes the original state.
      return sInput;
   } // end String getString(String sPrompt)

   /**
    * Presents a prompt to the user and retrieves a <i>boolean</i> value.
    * @param sPrompt reference to a <i>String</i> object whose contents will be displayed to the user as a prompt.
    * @return <i>boolean</i> value input from keyboard
    */
   public boolean getBoolean(String sPrompt) {
      System.out.print(sPrompt);
      return scannerKeyboard.nextBoolean();
   } // end boolean getBoolean(String sPrompt)
} // end class Input()
```