Using the try-statement in reading from the keyboard

Class JLiveRead provides static methods for reading items from the Java console. Let's look at method readLineInt. This method reads in the next line, which is supposed to contain an integer, and returns it. If the user types something other than an integer, the method prompts the user for an integer, again and again until the user finally satisfies the request.

```
/** The input line is supposed to contain a single int integer, perhaps with whitespace on either side.
  * Read the line and return the integer on it. If line does not contain an int integer, ask user to try again. */
 public static int readLineInt() {
    String input= readString().trim();
   // invariant: input contains the last input line read; previous
   // lines did not contain an recognizable integer.
   while (true) {
       try {
          return Integer.valueOf(input).intValue();
       }
   catch (NumberFormatException e) {
          System.out.println("Input not an int. It must be an int like");
          System.out.println("43 or -20. Try again: enter an integer:");
         input= readString().trim();
          }
   }
```

The method calls a method to read the characters on the next line, trims whitespace from both ends, and stores the result in a fresh variable input.

Now, the value in input has to be converted to an integer. But if it is not an integer, the user has to be requested —again and again— to type an integer. So, a loop is required. Here's the obvious invariant.

invariant: input contains the last input line read;
previous lines did not contain an recognizable integer.

The first statement truthifies the invariant. The body of the loop should return the integer that is in variable input. This String value has to be converted to type int. This is done using a method of wrapper class Integer to convert it to that class-type and then using a second method to extract the int value.

The problem is that method valueOf will throw a NumberFormatException if variable input cannot be converted to an integer —the specification of method valueOf tells us this. Therefore, we enclose the return statement in a try-statement that has a catch-clause to catch this Exception.

The catch-block prints a message that again prompts the user for the integer and then reads the next line and stores it in variable input. Notice that this second statement truthifies the loop invariant.

Note that the loop body returns as soon as the integer is calculated. Therefore, we can use true as the loop condition.

Note how the method did not appear all at once but was developed little by little. First, the obvious statement to read the input line was written. Second, since we knew a loop would be needed, we wrote one and put in the statement to return the integer. Third, the try-statement was inserted because we knew that method Integer.valueOf could throw an error. And fourth, we wrote the catch-block.