Comp 401 – Bonus Assignment
Exceptions, Iterator, Assertions

Date Assigned: Wed Nov 30, 2011
Completion Date: Fri Dec 9, 2011
NO LATE SUBMISSIONS ACCEPTED!

This is not a real assignment in that separate points will not be allocated for it. Instead, it is an extra credit assignment in that points you earn on it will be added to the total computed for the other twelve assignments. No late submissions will be accepted for this assignment. There is no early submission.

**Defining and Using an Iterator**

Replace your scanner with one that iterates tokens by implementing an integrator interface. Java actually provides a generic Iterator interface, `java.util.Iterator` interface, which is parameterized by the type of the object returned by `next()`. However, it has a spurious method, `remove`, which you do not need, and reduces your flexibility in throwing exceptions. Therefore, define your own interface.

The scanner should not store the whole collection of scanned tokens. Thus, it no longer has a getter that returns a collection of scanned tokens. In fact, it implements only the two iterator methods. Each time a new string is to be scanned, a new instance of the iterator should be created.

The constraint that the scanner can only implement the two iterator methods means that errors cannot be passed from the scanner to the scanner user. This is ok for this part. In the next part, you will use exceptions to overcome this problem.

Your parser can store the tokens returned by the scanner in a collection before parsing them.

**Recursive-Descent Parsing without Storing**

For additional extra credit, change your parser so that it too does not store the token collection and still does recursive descent parsing.

**Declaring, Throwing, and Catching Exceptions**

If you have not done so already as part of extra credit, define a special readonly dependent `String` property in the parser to show the scanning and/or parser errors encountered while processing the command string. The setting of the error string should be done in the setter for
the command string. Use exceptions to communicate errors between the methods that detect them and the command setter.

When a scanner or parser method detects an error, it should not print the error or return a special value to its caller. Instead it should throw either a scanning or parsing exception, depending on the kind of error, and set the message of the exception to indicate the error details. Thus, the method that catches an error will determine the error message, but not when or how the error message is displayed. This decision will be made by the command setter.

**Preconditions following ObjectEditor Conventions**

Define and assert preconditions for the undo and redo methods of the history undoer. Next, define and assert preconditions for the public methods defined by the simulation to move knights, speak, pass, and fail. You should implement the preconditions in separate public methods that follow the ObjectEditor conventions for these methods. This will allow ObjectEditor to disable the menu items for parameterless methods whose preconditions are not met, and ensure it does not call methods with and without parameters whose preconditions are not met.

ObjectEditor will call precondition methods only for the objects it displays. This means it will not call precondition methods in the undoer, which is not displayed by it.

The status of menu items is disabled/enabled each time ObjectEditor does a refresh of the frame. When you invoke commands in one window, other windows are not refreshed. You can always do a manual refresh of a window to display its correct state.

The precondition methods should be added to the interfaces of the classes that implement them. If a method has no precondition, its precondition method should return true.

Your parser should call the precondition methods in the simulation to check the validity of commands entered by the user. For extra credit to the following: If a precondition of a command entered by the user is not met, the (recursive-descent) parser method that is responsible for calling the associated simulation method should throw a new kind of exception - a semantic exception - which should be caught by the setter of the command string, and an appropriate message displayed. For example, if the user enters the say command, and the precondition for the say simulation method is not met, the parser method that calls the say method in the simulation should throw a semantic exception (to distinguish it from a parsing or scanning exception) with an appropriate message, which is printed by the parser command setter.

**Iterator for your Generic Collection**

Implement the java.util.Iterator interface for your generic collection. That is, create a class that implements this interface and takes as a constructor argument an instance of your collection.
An instance of the class should iterate the elements of the collection. Next change the generic collection to implement the java.util.Iterable interface.