

# Comp 401 – Assignment 11 (Last!): Error Handling

Date Assigned: Thu April 16, 2009

(Early) Completion Date: April (23) 27, 2009

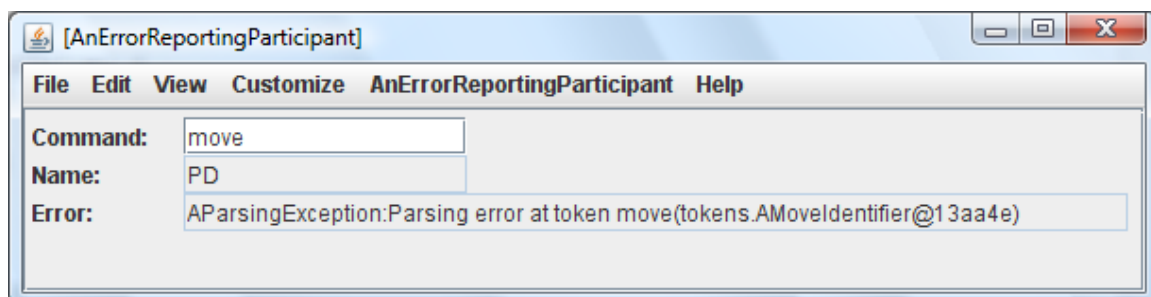
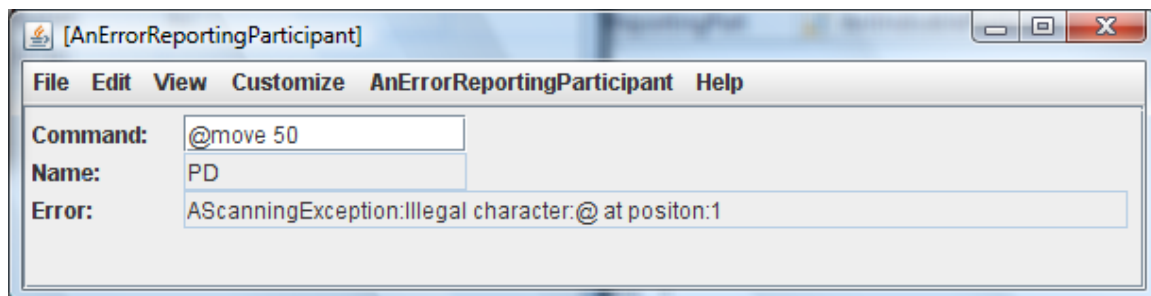
This assignment covers declaration, throwing and catching of exceptions/errors, preconditions/assertions, and visitors. In addition, if you have not implemented undo as part of the previous assignment, you should do so in this assignment. If we do not cover visitors by April 23<sup>rd</sup>, you will not be required to implement them.

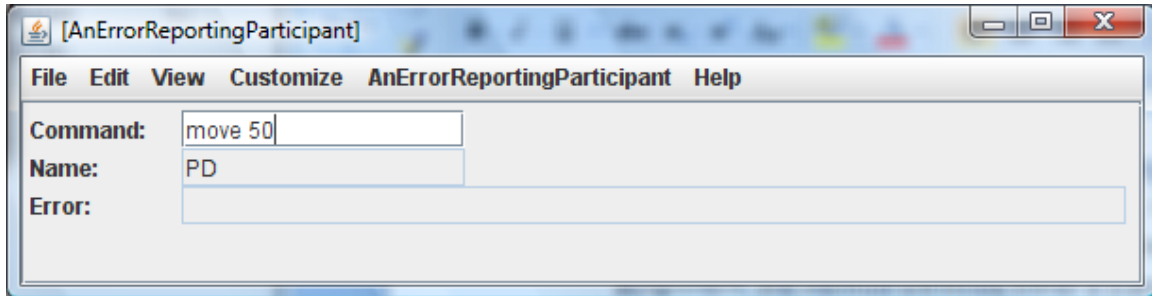
## *Undo*

Finish the undo component from the last assignment, using the undo pattern described in class.

## *Declaring, Throwing, and Catching Exceptions*

In assignment 3, you detected scanning errors. In assignment 8, you detected parsing errors. In both cases, a method that detected an error printed a message on the console. In this assignment, the method will 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 user-object. It will now define an extra readonly property that contains the error associated with the last command entered by the user. This is shown below:





You should create a subclass of the proxy user-object you created for implementing the undo part of the project. You do not have to use delegation for this part, but you should not mix error reporting code with the rest of the project. By creating the subclass, you can isolate the error reporting in a separate class.

Define separate checked (non-runtime) exception classes for the two kinds of errors: scanning and parsing. You will need to catch these exceptions in the command setter of the subclass.

As these are checked exceptions, you will have to change the headers of the methods in the chain of calls from the setter to the scanning/parsing methods. This, in turn, means that your scanner iterator interface must change – in particular, it can no longer be an elaboration of the `java.util.Iterator` interface. It must again be a custom interface, this time extended to propagate exceptions.

As you see in the figure above, the length of the text field created for the error property is not the default length. You can override the default length in two ways. If you download the latest version of `ObjectEditor`, you can associate the getter of a property with a `util.ComponentWidth` annotation, which takes the desired width (in pixels) of the component used to display the property, as shown below:

```
@util.ComponentWidth (800)
```

```
public String getError() {  
    return error;  
}
```

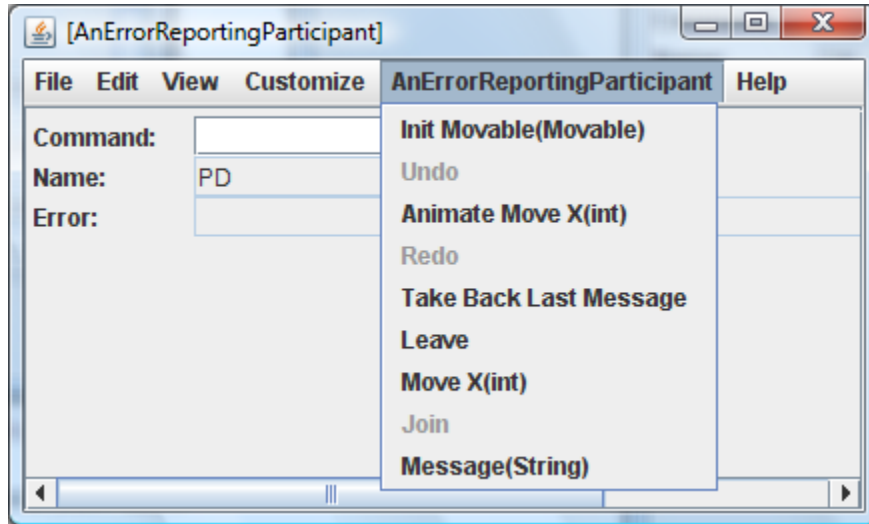
If you are using an earlier version of `ObjectEditor`, you can set the component width of all properties using the following call:

```
ObjectEditor.setDefaultAttribute(AttributeNames.COMPONENT_WIDTH, 800);
```

### ***Preconditions following ObjectEditor Conventions***

Define and assert preconditions of the `join()`, `leave()`, `move(int)`, and `message(String)` methods of your user object. You can do so in the subclass you implement for reporting error – you do not have to create a new subclass or delegator. 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 methods whose preconditions are not met, as shown below, where the join method is disabled because the user is already in the conference.



### ***Quantified Assertions and Visitor Pattern***

Assert the following *post condition* of the method *message (String)* of the user object, which adds a new message shape to the chat history of the avatar associated with the user object:

Each shape in the chat history is above the avatar.

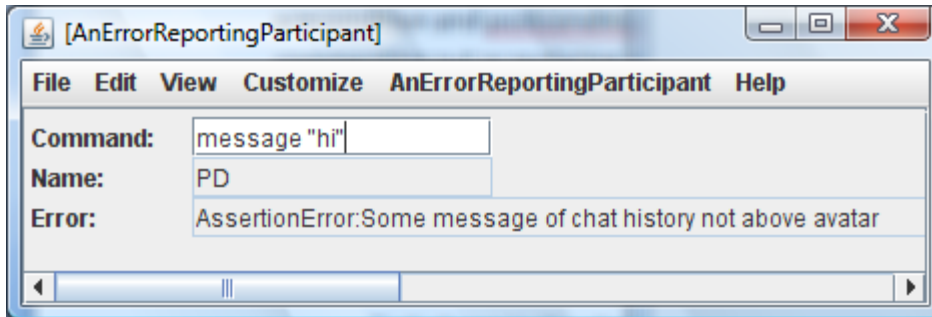
This assertion fails if (the bottom of) any message shape is below the (top) avatar.

You must use the visitor pattern and the assertion library to make this assertion. The library is available as part of the new ObjectEditor version. This assertion should be made in the user-object class you created as part of assignment 8. The `AssertionError` thrown by the failure of this assertion should be processed in the user-object subclass you create in this assignment for handling errors by setting the error property to the message associated with the `AssertionError`.

To use the assertion library, you will have to create an iteration of the elements of the chat history. The easiest (and most inefficient!) way to do so is to copy the elements of the chat History to an `ArrayList`, call the `iterator()` method on this list to create the iteration, and pass this iteration to the library.

The new ObjectEditor is pretty stable at this point. However, if you cannot use it, simply copy the assertion library into your project from the PPT slides.

Introduce an error that causes the above assertion to fail, document it, and create a screen shot showing the assertion fail, as illustrated below:



### *Extra Credit*

Define preconditions of the undo and redo methods of a user-object using ObjectEditor conventions so the corresponding menu items can be dynamically disabled/enabled.

In assignment 8, I put the following constraint on you:

The message sent in the chat history of an avatar should be vertically aligned, as shown in the figure below. It's up to you whether you align the left, right, or middle of each message shape. It is not possible to send messages concurrently, thus the messages sent by the users have a unique order. A message should be displayed above the message sent before it, and the spacing between messages should be constant, as shown in the figure below.

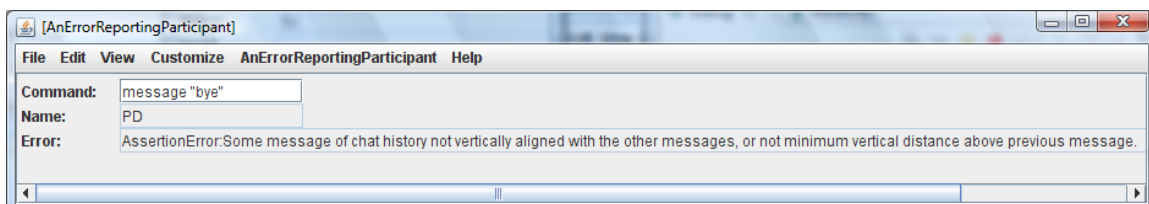
Some of you decided to reduce the spacing between messages as the chat history got bigger. However, you still need to ensure some minimum distance between each pair of successive messages.

Therefore, implement the following assertion as another post condition of the message(String) method of the user-object class of assignment 8:

All messages in the chat history are vertically aligned along the left, right, or middle of each a message shape. Each successive message is displayed some minimum distance above the previous message.

Again, you must use the visitor pattern and the assertion library to implement this assertion. Again, the error property of the user-object should be set to the message stored in the AssertionError object thrown when the above quantifier-based assertion fails.

Again, introduce an error, document it, and create a screen shot showing the assertion fail, as illustrated below:



## *Constraints*

As mentioned above, you must use the undo pattern for undo, put exception and error catching code in a separate user-object class and must use the visitor pattern in the extra credit part.

## *Submission Instructions*

1. Submit a print out of your code together with screen shots showing your code working in various cases, and a document identifying how you support various style and functionality features. You can either submit it in class on the early submission date or put it in your TA's mailbox (ask the Sitterson receptionist where it is) on the regular submission date. **Be sure to indicate the name of the class that runs an error-free version of your entire project.**
2. Upload the assignment directory in blackboard. In general, for all assignments, you should do so by midnight of the day the assignment is due. But do not change the code after you submit it in class.