Object Adventure Game ("Stateful" Items)

REMINDERS and NOTICES:

- See the note in the previous assignment; the same information applies here.
- **IMPORTANT: DO NOT START THIS ASSIGNMENT BEFORE COMPLETING THE PREVIOUS ONE!**
- **Also, don't change any code *you* didn't write**! If you wish to modify the behavior of an item created by someone else, use the "Decorator Pattern," which we'll see next week.

Objective:

Implementing more "complex" behavior for objects utilizing the "State Pattern".

---

1 – Make sure your branch is up to date with the trunk.

E.g.:

```
svn merge ^/101/trunk
```

Address any merge conflicts and test your resulting code. Once it works, then…

```
svn commit
```

Note: If you encounter "merge conflicts" that you can't resolve, talk to me before applying any solutions you find online. More often than not, those "solutions" will only worsen the problem!

2 – Implement the State Pattern on at least one Item:

(*Note: this assignment does not use any new repository management syntax, so there is no reason to repeat all the* ***svn*** *command syntax here. Refer to previous assignment descriptions if you need a syntax refresher.*)

a) **You should <u>not</u> begin this part until the *previous* assignment has been finished and your branch has been committed, with the commit's revision number included in the Blackboard submission.)**

b) Implement the ***State Pattern*** *(as discussed in class, the Example set, and the text/slides)* for at least one of your items.
   - Be sure to commit changes back to your branch as you proceed, not all at once at the end. In other words, if you don't ever want to lose it, commit it!

MINIMAL IMPLEMENTATION:

This assignment is an opportunity to use your own creativity to make an object (one of your "items") more "interactive". Feel free to use your imagination or try to mirror the behavior of some "real-world" object.

At a ***minimum,*** allow the Item to move through at least three different "states" each time the Item is "**used**", varying its own description each time. For example, this is what it might look like in gameplay:

```
> inspect lamp
The lamp is off

> use lamp
The lamp is now on its lowest setting

> inspect lamp
The lamp is very dim

> use lamp
The lamp is now on its highest setting

> inspect lamp
The lamp is very bright

> use lamp
The lamp is now off

> inspect lamp
The lamp is off
```

(Optional) BETTER IMPLEMENTATION:

Be more creative by using item events such as ACTIVATE, DEACTIVATE, OPEN, CLOSE, USE, ENABLE, etc., to allow for logical "state" alterations based on the item's nature. Use the event list in the source sample item to see the possibilities.

```
> use lamp
You can't use the lamp unless you are holding it.

> take lamp
You have taken the lamp.

> use lamp
You can't use the lamp until it has been activated.

> activate lamp
The lamp is now on!

> use lamp
You don't like what you see!!!

> drop lamp
You cannot drop an activated lamp.

> deactivate lamp
The lamp is off.

> throw lamp
The lamp is now in a myriad of pieces.

> use lamp
The lamp is broken and can't be used until repaired.

Etc…
```

Of course, you may also change the description of the "Room" in response to the item state changes.

For example:

```
...
> look
It is too dark to see.

> activate lamp
The lamp is now on!

> use lamp
You don't like what you see!!!

> look
This room is a mess!
...
```

Idea: You may also " spawn" new items in the room (like the gumball machine, for example).

3 – Commit Your Final Changes

Commit your final changes back to your branch.

5 – Indicate that you are done with a brief description

When you are finished, indicate that you are done using “write submission” and provide two or three sentences describing your changes. I.e., briefly describe the state transitions.

Please show the revision number reported after performing the last commit with the “Done!” comment.

E.g., “**Done at revision 338. Added the following states to** <*item_name*>**: …**”

Note: The revision number is given during the commit:

```
...
Committing transaction...
Committed revision 338.
```

If you forgot to make a note of it, the following command can be used to see the revision info for your most recent commit:

```
svn log --limit 1
```

The first line will look something like this (with r9999 being the revision number):

```
r338 | aconover | 2024-04-02 22:27:29 -0400 (Tue, 02 Apr 2024) | 1 line
```

Appendix:

An FYI On the Customary Command-Line Usage Instruction Symbols

This is only here to illustrate the notion I use when referring to command-line commands.

Square Brackets `[]`: Square brackets usually indicate optional components of a command. For instance, `mkdir [-p] <directory_name>` suggests that the `-p` option is optional; you can create the directory without it, but using `-p` allows the creation of parent directories if they don't already exist.

Angle Brackets <>: Angle brackets often denote a placeholder for a required argument. For example, in a command like `cp <source> <destination>`, you must replace `<source>` and `<destination>` with the actual paths or filenames you want to use.

Curly Braces `{}`: Curly braces indicate a choice between multiple options. For example, in a command like `chmod {+|-|=} <permissions> <file>`, the braces indicate that you can choose one of the options `+`, `-`, or `=` to modify the permissions of the file.

Examples in the real world:

```
$ svn --help
usage: svn <subcommand> [options] [args]
```

```
$ git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--config-env=<name>=<envvar>] <command> [<args>]
```

```
$ grep --help
Usage: grep [-HhnlLoqvsrRiwFE] [-m N] [-A|B|C N]
            { PATTERN | -e PATTERN... | -f FILE... } [FILE]...
```