Object Adventure Game (Room and Item Creation)

NOTE: In the steps below, anything shown in angle brackets '<' and '>' should be replaced by you (without angle brackets). E.g., <`your_TU_username`> should be replaced with your actual username, WITHOUT the angle brackets.

Note to *Windows* Users

The Windows CMD "Command Prompt" treats a '**^**' as a special symbol, meaning you must either quote or "escape" the '**^**' symbol. For example:

`^/trunk` must be written as either **`^^/trunk`** or **`"^/trunk"`**.

This ONLY applies to the *Windows* "command prompt", **not** *PowerShell*, *zsh*, *bash*, etc.

---

1 – If you have not done so already, perform a checkout of the Game repository trunk:

```
svn co svn://cosc436.net:65436/Game/101/trunk COSC436_Game
```

Be sure to "cd" into the Game folder after the checkout:

```
cd COSC436_Game
```

*Note that the project source code is in the "**src**" folder within the working copy.*

---

2 – Switch to your personal branch of the project.

Individual "Branches" have already been created for you, so you may "switch" to the personal branch with the following command:

```
svn switch ^/101/branches/<your TU username>
```

(If you receive any error here, double check the "Note to Windows Users" above and double-check that command was correctly entered.)

Note: Technically, you can check out the branch directly,

```
svn co svn://cosc436.net:65436/Game/101/branches/<your TU username>  COSC436_Game
```

However, as I'll be switching branches in class quite a bit, I wanted to illustrate "branch switching"

3 – Create a project from the source code in your favorite development environment

There are no extra instructions here as every IDE differs slightly (*Eclipse*, *IntelliJ IDEA*, *NetBeans*, *VSCode*, *Vim*, etc.). However, in *IntelliJ* and *VSCode*, you simply need to "**Open Folder**" from within the IDE. The folder to open is the one created in the first step.

4 – Modify/Augment the game environment

The package: `objectAdventure.world.<your TU username>`
is where you'll place all your content.

FOR THIS PART, BE CREATIVE! Remember, we are creating a "game" for all to share!

a) **Create a new "Package" for your work by adding a folder named `<your TU username>` to the `objectAdventure/world` folder.**

b) **Create a new "Room" by adding a new room class to:**
   - It should be placed in the **`objectAdventure.world`** package
   - The class must extend the abstract **`Room`** class and reside in the correct folder.
     - You may call the class whatever you want if it follows the Java class naming conventions. **Class names should always start with an uppercase Letter**.
     - For a sample, see: **objectAdventure.world.aconover.StartRoom**

c) **Make sure your "Room" is `added` to version control. `Commit` your changes thus far.**

d) **Add a new "Item" to your room.**
   - It should be placed in the **`objectAdventure.world`** package
   - The class may be called whatever you like but you must *implement* the **`Item`** interface.
     - For a sample, see the class: **objectAdventure.world.aconover.DemoItem**

e) **Make sure your "Item" is `added` to version control. `Commit` your changes thus far.**

f) **Modify the `objectAdventure.RoomInitializer` class as follows:**
   - Modify this file's **`initRooms()`** method when your room is complete enough to be instantiated without errors.
     - If you look at the existing **`initRooms()`** method, the required modification is documented there. You may follow the example at the top of the **initRooms()** method.

5 – Commit Changes to your branch

a) Verify that your code compiles without errors.
b) Test it within the Game shell to verify that it works.
c) Double-check to make sure you have no uncommitted “added” or “modified” files.
d) Commit your changes back to your branch.

6 – In Blackboard, indicate that you are “Done”:

a) Write “Done” in the Blackboard Assignment “Write Submission” area.

b) Please show the revision number reported back after performing the last commit.
E.g., “**Done at revision 338**”

Note: The revision number is given during the commit:

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

If you forgot to make 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):

```
r9999 | aconover | 2024-04-02 22:27:29 -0400 (Tue, 02 Apr 2024) | 1 line
```

(FYI: For the **git** users, **svn** revision numbers are much like the git commit hashes, they just run in sequence. Being centralized, svn can keep track of the exact ordering of every commit, making histories completely linear.)

NOTES: **You are <u>not</u> committing to the trunk; you will only be committing to your branch.**
I have the “trunk” set as “read-only”. If you received an “access denied” message, you likely did not complete step 2.

I will be merging your “Rooms” and “Items” into trunk as we progress, so keep it “Family Friendly” 😊

OTHER NOTES:

- Be sure you don't "checkout" into a folder containing an existing "checkout"! While both SVN and GIT allow it for particular use cases, it can cause a LOT of confusion if you do it accidentally and without a specific reason.

- Be careful about having multiple checkouts on the same computer. While having a duplicate set of files may be occasionally useful, it can also cause confusion if you are not careful.

- It's very easy (and common) to work with code on multiple machines, such as a laptop and a desktop; you only need to do a single "checkout" on each machine.

    - To keep everything synchronized, you need to make sure you "**`svn commit`**" your changes when finished working on one machine, and then "**`svn update`**" your work on the other machine. If you are working on separate parts of your code on different machines, you don't necessarily need to "commit" any code you don't feel is "ready". Once committed, an "update" on one machine will merge the changes "committed" on the other.

    - (*I use this process to bounce between numerous computers at home, work, classrooms, etc.*)

A quick guide to customary “command help” notation.

(See also: http://docopt.org/, but note that no “official” standard exists... to my knowledge.**)**

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 creating 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 instance, in a command like `chmod {+|-|=} permissions file`, the braces indicate that you can choose one of the options `+`, `-`, or `=` to modify the file's permissions.

Examples of “help” from various tools (as an illustration):

```
$ 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]...
```
