COSC436/COSC716 – Assignment 1

There is no actual submission to Blackboard (aside from showing you completed it).

IMPORTANT NOTES:

- *All* **SVN** *commands should be run from within the "top-level" directory (the "project folder"), which you will see below.*
  - *For example, if you check out a "Working Copy" into a folder/directory called* **MY_PROJECT***, your* **SVN** *commands should be run directly from within the* **MY_PROJECT** *folder. (Not a directory "under" that directory or "above" it.)*

- The terms **Directory** and **Folder** may be used interchangeably.

- Anything in <*angle brackets*> describes a placeholder for your own unique text. Yes, believe it or not, I occasionally get "<*your name here*>" literally instead of a real name.

Assignment Objective:

This assignment continues the example started in class and aims to help students understand how to work with code in a repository.

This assignment will use the following repository:

```
svn://cosc436.net:65436/SeeAndSay/101/trunk
```

The actual Instructions start on the next page.

1. Determine where you want to keep your projects, and then from a terminal window (in that location), "checkout" the trunk of the "SeeAndSay" repository using the following command:

```
svn checkout svn://cosc436.net:65436/SeeAndSay/101/trunk SeeAndSay
```

(The last parameter, *SeeAndSay*, can be changed to whatever folder name you want!)

NOTE: *You should **never** run the checkout command more than once per project, or you will create one project inside another. Unless you know how to manage nested projects, doing this will likely cause a LOT of confusion, especially since separate **projects should be in separate folders.** If you need a NEW checkout, checkout in a new folder outside your project. (FYI, git will also allow you to "clone inside of a clone," creating the same issues.)*

FYI for GIT users:

- ***svn checkout** is effectively the same as **git clone.***
- ***svn update** is essentially the same as "**git pull**", but with svn, the incoming changes are integrated into your working directory/folder, as svn does not "clone" entire repositories, but only the "head" of the specified branch.*
- ***svn commit** is essentially a **git commit** followed by a **push**.*

---

2. From the terminal window, change your working directory to the project (`cd SeeAndSay`), then run:

```
svn info
```

If you get an error message saying "not a working copy," you probably skipped the "**cd**" step! Otherwise, you should see general information about the checkout itself… which indicates that you have done everything correctly.

It should look something like this:

```
Working Copy Root Path: T:\COSC436\SeeAndSay
URL: svn://cosc436.net:65436/SeeAndSay/private
Relative URL: ^/101/trunk
Repository Root: svn://cosc436.net:65436/SeeAndSay
Repository UUID: 6f48c927-2fe6-434b-a085-1605cdde8762
…
```

3. **Open the Java project in the environment/IDE of your choosing.**

- For ***VSCode*** and ***IntelliJ IDEA***, you can use "Open Folder" to open the folder ***containing*** the project's "**`src`**" folder. You will likely have problems if you just "open" the "src" folder itself.
    - VSCode users should have the "***Extension Pack for Java***" plugin installed. Also, the VSCode SVN extension (the one by "Chris Johnson") is very useful but not strictly necessary, as everything can be done from the terminal.
- ***NetBeans*** has a few more manual steps, which are still straightforward, but I can help you with this if need be. (NetBeans is still an underrated Java Development Tool! It's been developed longer than almost any other Java IDE.)
- You are on your own with ***Eclipse***, as I don't use it… but you are welcome to do so if you are already comfortable with it. (FYI, "Subclipse" seems to be the standard SVN plugin for Eclipse.)

NOTES:

Complete coverage of creating new projects in IDEs is beyond the scope of the assignment. Preferred development environments are highly personal choices, and every IDE has a different process for creating new projects from scratch and around existing sources. (While the instructor can help with Netbeans, IntelliJ IDEA, and VS Code… if you don't know how to "create a project from existing sources," search engines can be your friend here!).

For those familiar with Maven, Gradle, Etc., this is an "unmanaged" / "no build tools" project. If that doesn't mean anything to you, don't worry; following the directions above is sufficient.

---

4. **Create your own "Animal" instance.**

Using the **Aconover.java** source file (in the "**SeeAndSay.animalImpl**" package) ***as an example/template***, create your own "Animal Implementation" class in the same package using **your** username (upper-case the first letter).

*You can make it "say" whatever you want; it doesn't have to be a real animal! The goal is to create a class that correctly extends the **Animal** base class, implements the **Talker** interface, and has the resulting code committed to the shared repository*.

---

5. **From a terminal window, run the command:**

```
svn status
```

This will state/status all the files that are (or are NOT) under version control. By default, only modified or un-versioned files are shown. If you see any files with "?" in front of them, they have not been "added" and will not be part of the commit.

Note, you may add a **-v** to the status command to show **all** files, even if they require no action:
`svn status -v`

IMPORTANT: *Only* the **.java** files need to be added! Don't add **.xml**, **.iml**, etc. files to SVN control, as these are generally "meta-data" files for your local project that no one else will need or want. If everyone has different IDEs, etc., committing these files can cause problems for other developers.

---

6. **Add your newly created class to SVN version control from a terminal window with the command:**

`svn add` *`<path_to_your_file_name>`*

For **Example**:

`svn add src/SeeAndSay/AnimalImpl/`*`Aconover.java`*
(Of course, use your own class/file name)

*FYI for GIT users: Once a file is "added," you **never** need to "add" again in SVN (unless you remove it, of course); there is no separate "staging" step in SVN. However, if desired, "staging" can be simulated with SVN via "Change Lists)*

---

7. **Modify the `addAnimals()` method in `AnimalListBuilder.java`**

   Modify the **`addAnimals()`** method in the **`AnimalListBuilder.java`** source file (in the "**`SeeAndSay`**" package) by adding a line that creates an instance of your animal object and adds it to the list. See the first line of the **`addAnimals()`** method for an example!

---

8. **TEST YOUR CODE!!! Ensure everything works as expected AND that your new file(s) have been "added" to version control.**

   Use "**`svn status`**" (as before) to verify that all relevant files are added.

   If you see any **`.java`** files with a '**?**' in front of it, see step 6.

---

9. **If you skipped the previous step, go back to the previous step!**

---

10. **Once it works (*and has been verified to compile*), perform a final commit of your changes:**

```
svn commit -m "<Replace this text with a description of the commit>"
```

(Hopefully, it is obvious, but the "message" above should be your own, not just a copy of the description.)

*Note: If you receive a message that your local "*working copy is out-of-date*", run the command: "`svn update`". This will merge the existing repository state into your working copy, allowing you to commit the combined changes.*

NOTE: If you receive a "conflict," we will discuss conflict resolution separately, but generally, just picking the menu option "accept theirs before mine" will be relatively safe for this assignment. Then, select the "mark as resolved" option to indicate that the problem has been "resolved."

---

11. Indicate that you are “DONE” in Blackboard:

In the "Submission" within Blackboard, indicate "Complete as of Revision: _____" where the revision number of your final commit revision number.

*NOTE: The revision number is shown with each commit but may be found again simply by running the following from within a project **after** performing an "**`svn update`**" to sync the log messages.*

```
svn log -r head -l 1 --search <your user name>
```

For example:

```
C:\Users\aconover\Desktop\SeeAndSay_Public>svn log -r head -l 1 --search aconover
------------------------------------------------------------------------
r249 | aconover | 2025-02-13 20:28:37 -0500 (Thu, 13 Feb 2025) | 1 line
```

r249 is the revision number in this example.

Other Notes:

You may make mistakes, others may “break” your code (hopefully inadvertently), you may have to merge your changes into changes made by others, or even fix something someone else did incorrectly! This is the nature of collaborative development.

You will often need to run "***svn update***" on your local checkout before you commit; svn will tell you that your working copy is outdated and update if so. (The same thing can happen in GIT if you try to “push” into a branch or an upstream repo ahead of your own.)

Also, we are not using individual “Branches” yet but *will* do so as we move forward.