FH University of Applied Sciences TECHNIKUM WIEN

Application or use cases of various design patterns:

Home Assignment 1: Implementing Design Patterns

Group Participants:

Báez, Mauro Leandro

Manzur, Matías

Date: March 20th, 2025

General Description of our Application

The application is a prototype of a simple character creator, where you are able to create humanoid creatures but with animal body parts. Therefore, this app is of great use if the user needs a fast way of creating weird animal creatures for whatever reason they may need them (games, illustrations for stories, or just for fun).

This app offers a graphical user interface (GUI) where users can choose different body parts for the creature, combining them as they like. Also, users can undo or redo changes made to their fictional characters, as well as just generating a random combination of body parts if they are out of ideas.

Design Decisions and Patterns Applied

The main three design patterns we decided to implement were: Singleton, Memento and Command.

We decided to contain the main application state in one Singleton class Creator. This class is in charge of persisting all the available body parts for selection, as well as which are the currently selected body parts.

To implement the undo and redo feature, we decided to apply the Memento pattern. This involves making the Creator class implement two methods: the first one generates a Snapshot of the current state of the application, while the second one loads a given Snapshot to restore the state described in it.

All this needs an external manager or “caretaker” that is responsible for managing the Snapshot history. This is where the VersionManager class comes in. It is also a Singleton class, since it is responsible for saving all the Snapshots part of the version history, which is part of the global state of the application.

The VersionManager contains two stacks of Snapshots: the first one contains all the previous states, while the second one is used to pile up the reverted states (for when the user does the undo operation). This allows the navigation through the whole Snapshot history with the undo() and redo() methods.

All the available body parts are instances of Head, Torso, Legs or Wings. All these classes extend the BodyPart abstract class. This abstract class defines all the attributes a body part contains (name, type, sprite and index).

On the other hand, we have the BaseCommand interface (which is implemented as an Abstract class with no internal state in Python). This allows us to define that all Commands should implement the execute() method. This method is the one responsible for making the actual operation the user triggered. This implementation corresponds to the Command design pattern, which allows us to centralize in one place all the logic executed by each command.

For example, we want to trigger the Undo operation both when the “undo” button is clicked and then the user presses Ctrl+Z. With this pattern, we do not need to implement the Undo logic in both UI calls, since we just implement it in the UndoCommand and both inputs will call the execute method of this Command. The same goes for the Redo and Random commands.

In addition, we also have the NewVersionCommand, which is used when the user wants to generate a new version of the creature by changing one of its components. Since the command can be called with different combinations of body parts as attributes, often not expliciting them all simultaneously, we decided to generate an instance of NewVersionCommand using a Builder.

Insights

We have learnt how useful some design patterns which we had never used before are. For instance, thinking of the Snapshots as a Memento pattern, the coding itself was trivial, as we knew exactly what behaviour it should follow. Singletons are always useful but we learnt a new way of creating them that we had never seen before. However, in the end, we decided to only use one way of making a Singleton (through a static method) so as to be consistent throughout the codebase.

Command was a great way of repeating as little code as possible when there are different ways of triggering the same action. We thought it would be useful to use the Builder pattern in order to generate some of the commands, and it most definitely was. Should we want to add a new type of command, the builder would most likely come in handy.

Possible Improvements

Containerization would have been useful, but there are some issues with running ‘tkinter’ when using Docker so we opted against containerizing the project. As the project is relatively small we hope there are no compatibility issues depending on the system.

Adding more options and animals would be nice too! As well as making the GUI prettier…