

**University of Dhaka**

**Department of Computer Science and Engineering**

**CSE-3216 – Software Design Patterns Lab**

**Lab 3**

**Submitted by:**

Deeparghya Dutta Barua, Roll - 28

Jahir Sadik Monon, Roll - 32

**Submitted to:**

Dr. Md. Samiullah

Assistant Professor

Department of Computer Science and Engineering

University Of Dhaka

**Date of Submission:** 31st August, 2022

**Assumptions:**

During our implementation of a solution for the given problem, we had to make some assumptions. Here is a list of the assumptions we had to make –

1. Application’s Graphical User Interface will only have one window with customizable scenes, which in turn depend on the configuration file. Thus we can make WindowManager and ConfigManager singleton classes which are responsible for the window and the configuration of the scene on the window.
2. Buttons will implement an interface, thus allowing only accessible functions (changeColor(), changeTextSize()) through the interface.
3. To use the Adapter pattern, we assumed that our system already had capability to parse UIElemDescriptors (Description of UI elements) from input lines taken from a file. To extend the system's capability to work with XML files, we added an XML parser that can convert input XML line strings to text lines and made an Adapter class that can convert XML lines to UIElemDescriptors, thus adapting to new file format using the IParseToUIElem interface.
4. We used Abstract Factory pattern using the assumption that currently, only two different types of design styles are supported by the system, and we can have different factory methods for those design styles. In future, different styles can be implemented by other classes that implement the same abstract factory interface. The factory classes will return different style UI Elements, based on the type of factory they are.
5. We assumed that the user will initially choose the configuration file using the terminal, then input the style type, and finally input the color, and the text size (in case of high detailed style), and these parameters will remain the same throughout the program.
6. We assumed that the user will input the correct values for color and text size input. If the color input is incorrect, the GUI will switch to default Javafx colors and the input text size is too low or high, it’ll switch to default size of 14.

**Design Patterns:**

Here is a description of places where we have used the given design patterns –

**Adapter:** It is described in the problem statement that the system already has the capability to parse descriptions of UI elements from .txt files, which we have implemented in our program in the TextParser class. Now, to extend the capability of the system to parse configuration from .xml or any other file format, we have implemented the Adapter pattern.

To achieve this, we have a common interface named IParseToUIElem and made it so that all existing and all new Parser classes implement this interface. For .xml file types, we have implemented an adapter class that converts the xml lines returned by to XMLParser to UI Element descriptions, just like that TextParser. Using the same method, we can extend the capability of the system to parse different file format by adding respective parser class and adapter class, which implements IParseToUIElem interface.

**Singleton Pattern:** As we had assumed that the WindowManager and ConfigManager should only have one instance throughout the execution of the program, we made those classes Singleton classes and have used the getInstance() method to return the initialized instances of the classes.

**Abstract Factory:** We have implemented the abstract factory pattern by making a common interface IUIElemFactory, which defines the create() method which creates UI control objects for Javafx. Specific factory classes, in this case, [SimplisticUIElemFactory](https://github.com/jahirsadik/sdpLab3/blob/master/src/UIElements/SimplisticUIElemFactory.java) and HighDetailedUIElemFactory implements this abstract factory interface and returns Buttons, EditBoxes and TextBoxes according to their respective styles. Using this abstract factory interface, we can add other types of styles in our system without disrupting previous code.

\*(class diagram is added on the next page)

**Class Diagram:**

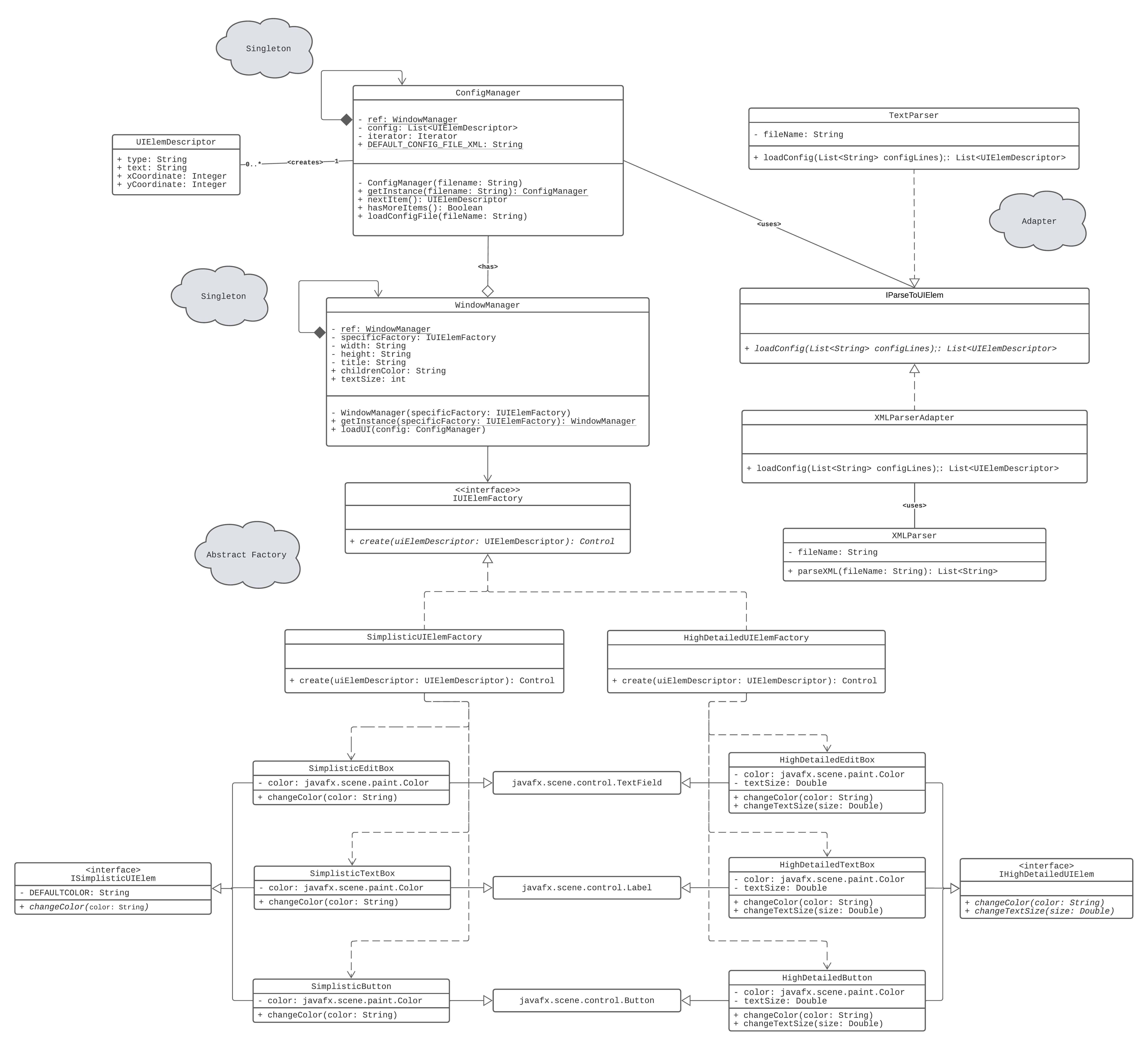


fig. UML Class diagram of our solution