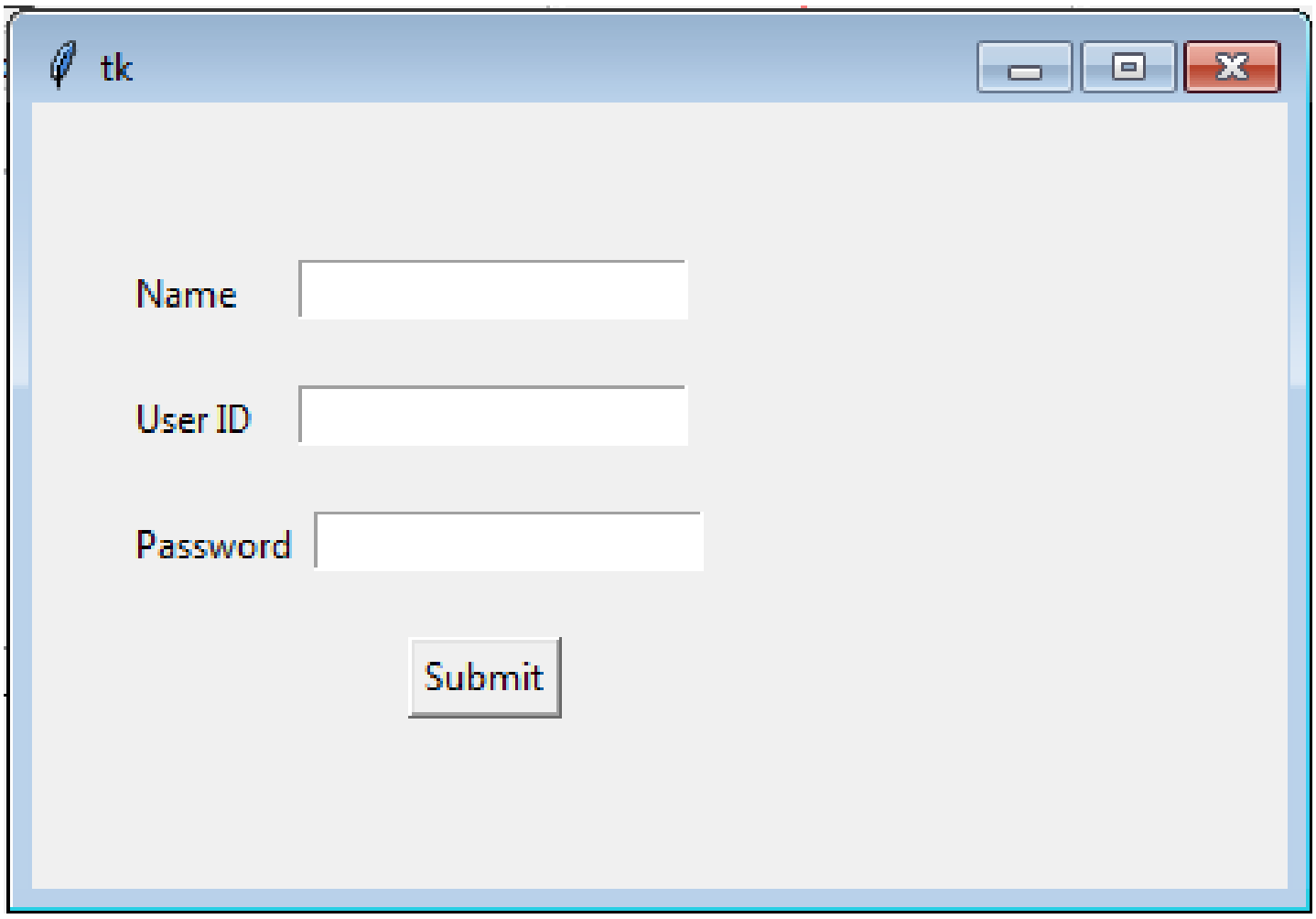
COMP 2012 Object-Oriented Programming and Data Structures

Lab 3 inheritance Basic



A Python Tkinker widget
Image Source: w3resource, retrieved here.

Introduction

In this lab, you will implement a simplified Graphical User Interface (GUI) framework with inheritance. To kick start, we will introduce what a Graphical User Interface is and show a diagram demonstrating the inheritance hierarchy of the classes we defined for this lab.

Introduction to Graphical User Interface (GUI)

Graphical User Interface is an interface in which a user completes certain tasks by manipulating graphical components such as icons, textboxes, buttons, and menus.

To develop an application with GUI, the programmers should write the essential computing codes (backend), and define the interaction between the backend and the User Interface (frontend). For example, a text box displays a line or multiple lines of text on UI, where the text content is usually defined by strings in the backend. You can click a button on the UI to perform an action in the backend.

Textboxes and buttons are important UI objects. It is not easy to define the complete behavior of these UI objects, including the interaction with an operating system, graphics display and working with multiple threads. Therefore, using a GUI framework that offers UI objects' toolsets via a simple program interface is common.

The GUI framework is also an excellent example of inheritance, for which we choose GUI as the lab topic for inheritance. The various objects share a lot of common properties and behavior. Some small subsets of objects might share even more commonality. inheritance and poly-morphism can make the UI system well-organized.

This lab explores a toy version of the Qt GUI framework. It is so simplified that it is far from a UI framework. Instead, it is a purely command line application. If you are interested in exploring the real UI framework, feel free to learn it elsewhere.

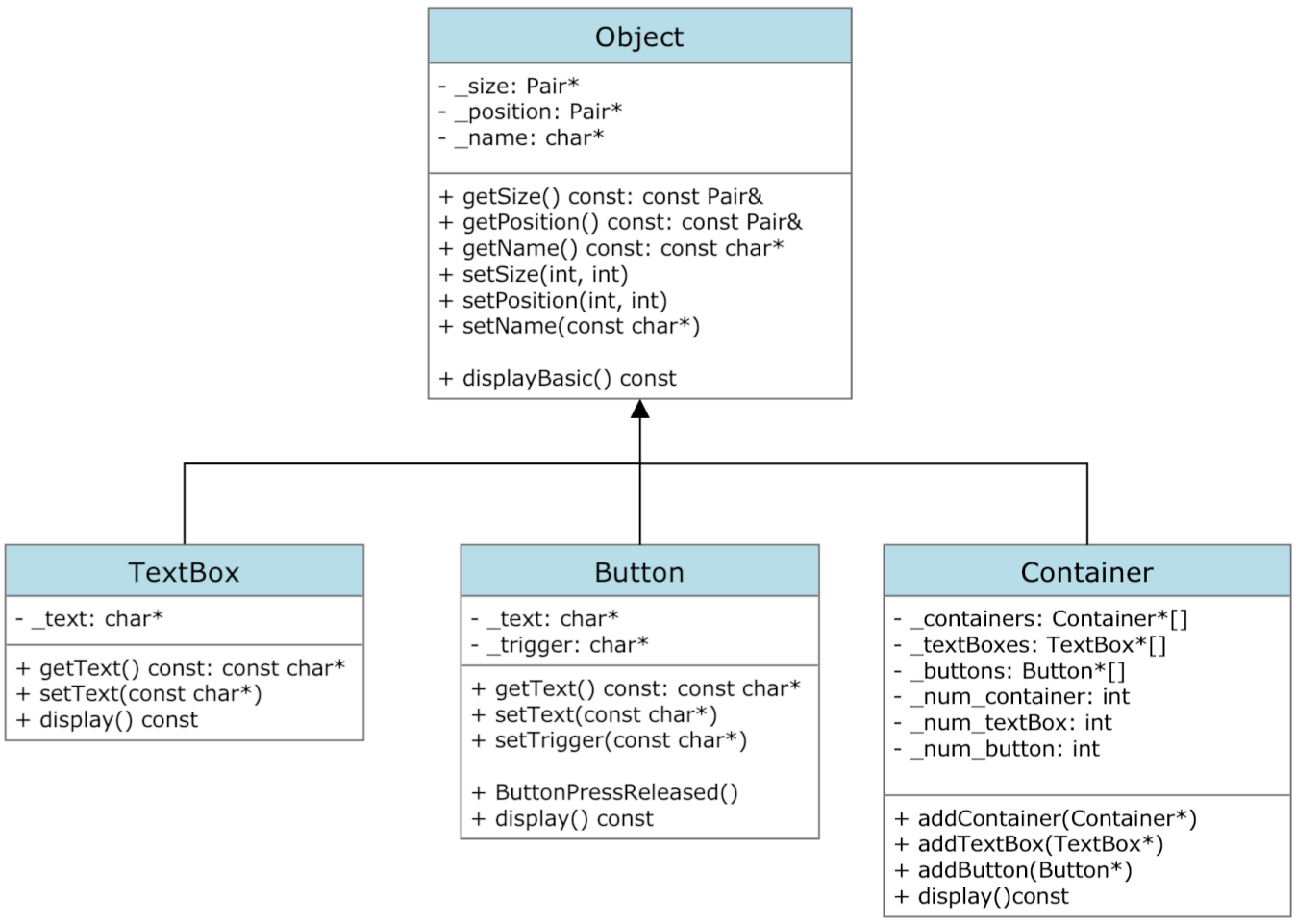
Overview of Code

Page maintained by

CAI Zhuo (Joe)
Last Modified:
10/11/2022 21:01:54

Homepage

Course Homepage



The UI framework

There are four classes in lab3.

- `Object`: The base class of all UI objects. See `object.h/object.cpp`. All UI Objects have properties size, position and name. Their information can be printed by calling `displayBasic()` member function.
- `TextBox`: The UI object displays a line or lines of plain text. See `textbox.h/textbox.cpp`.
- `Button`: The UI object acts as a button. See `button.h/button.cpp`. Note: In this lab, the `ButtonPressReleased()` member function does not modify the class. However, it is designed as a non-const function because in general pressing a button might change something.
- `Container`: A UI Object that contains other UI objects. See `container.h/container.cpp`. The `Container` can add other UI objects as its component via `addTextBox()`, `addButton()` and `addContainer()`.

The 3 derived classes have their own data members and member functions besides the members inherited from the base class.

Note: All UI objects should have a `display()` function. However, their behavior should be different depending on their features. This is a perfect use case of dynamic binding, which will be covered in the next lab. This lab implements the `display()` functions independently for each UI object class.

The struct `Pair` is defined in `object.h` as:

```
struct Pair
{
  public:
    int x;
    int y;
};
```

The Application

`main.cpp` provides an example application that uses the above "UI framework". In this lab, every application has a main page that contains all the other UI objects. After constructing every UI object and adding them to the desired container, calling the `display()` function of the main page (Container) should be able to display all its components. Calling the destructor of the main page will delete all the UI objects.

Three test cases are given for you to test your program. The expected outputs of these test cases are provided in the skeleton code. Test case 1 should print the following expected output.

```
LAB 3: Inheritence

Container
  Name: [MainPage]
  Position: (0, 0)
  Size: (1920, 1080).
    #textBoxes: 1 |

TextBox
  Name: [TextBox11]
  Position: (100, 100)
  Size: (100, 300).
    TextBoxText=[Welcome]

Destructing Container MainPage
Destructing TextBox TextBox11
```

Test case 2 should print the following expected output.

```
LAB 3: Inheritence

Container
  Name: [MainPage]
  Position: (0, 0)
  Size: (1920, 1080).
    #buttons: 1 |

Button
  Name: [Button11]
  Position: (400, 100)
  Size: (100, 300).
    ButtonText=[Press This!]

Destructing Container MainPage
Destructing Button Button11
```

Test case 3 should print the expected output as below.

```
LAB 3: inheritance

Container
  Name: [MainPage]
  Position: (0, 0)
  Size: (1920, 1080).
    #containers: 1 | #textBoxes: 1 | #buttons: 1 |

Container
  Name: [Page1]
  Position: (100, 100)
  Size: (960, 640).
    #textBoxes: 2 | #buttons: 2 |

TextBox
  Name: [TextBox11]
  Position: (100, 100)
  Size: (100, 300).
    TextBoxText=[Welcome]

TextBox
  Name: [TextBox12]
  Position: (100, 300)
  Size: (100, 300).
    TextBoxText=[Back]

Button
  Name: [Button11]
  Position: (400, 100)
  Size: (100, 300).
    ButtonText=[Press This!]

Button
  Name: [Button12]
  Position: (100, 400)
  Size: (100, 100).
    ButtonText=[Do not press me!]

TextBox
  Name: [TextBox1]
  Position: (100, 100)
  Size: (100, 300).
    TextBoxText=[Welcome Main!]

Button
  Name: [Button1]
  Position: (400, 100)
  Size: (100, 300).
    ButtonText=[Press This Button!]

Destructing Container MainPage
Destructing Container Page1
Destructing TextBox TextBox11
Destructing TextBox TextBox12
Destructing Button Button11
Destructing Button Button12
Destructing TextBox TextBox1
Destructing Button Button1
```

Lab tasks

You can download the skeleton code HERE.

Task 1

Implement the `display()` member function of the class `Button` in `button.cpp`. It should print the desired output for buttons according to the expected output. Hint: it is similar to the `display()` function of class `TextBox`.

Task 2

Implement the `display()` function of the class `Container` in `container.cpp`. The missing part should print the other UI components in the container You can read the expected output to decide how to implement it.

Task 3

Implement the destructor `~Object()` function of the class `Object` in `object.cpp`. It should safely delete its members. Although this function is not directly called in our test cases, it is called when objects of its derived classes are destructed.

Submission and Grading

Deadline: 10 minutes after your lab session, Oct 20/21, 2022.

Please submit the following files to ZINC by zipping the following 3 files. ZINC usage instructions can be found here.

- `button.cpp`
- `container.cpp`
- `object.cpp`

There are 6 test cases in total. The test cases 4~6 are the same as test cases 1~3, respectively, except that test cases 4~6 check memory leakage.

Notes:

- You may submit your file multiple times, but only the last submission will be graded. **You do NOT get to choose which version we grade.** If you submit after the deadline, a late penalty will be applied according to the submission time of your last submission.
- Submit early to avoid any last-minute problems. Only ZINC submissions will be accepted.
- The ZINC server will be very busy on the last day, especially in the last few hours, so you should expect you would get the grading result report not-very-quickly. However, as long as your submission is successful, we will grade your latest submission with all test cases after the deadline.
- In the grading report, pay attention to various errors reported. For example, **under the "make" section, if you see a red cross, click on the STDERR tab to see the compilation errors.** You must fix those before seeing any program output for the test cases below.
- Make sure you submit the correct file yourself. You can download your file back from ZINC to verify. Again, **we only grade what you uploaded last to ZINC**.

Compilation Requirement

It is **required** that your submissions can be compiled and run successfully in our online auto-grader ZINC. If we cannot even compile your work, it won't be graded. Therefore, for parts you cannot finish, just put in a dummy implementation so that your whole program can be compiled for ZINC to grade the other parts you have done.