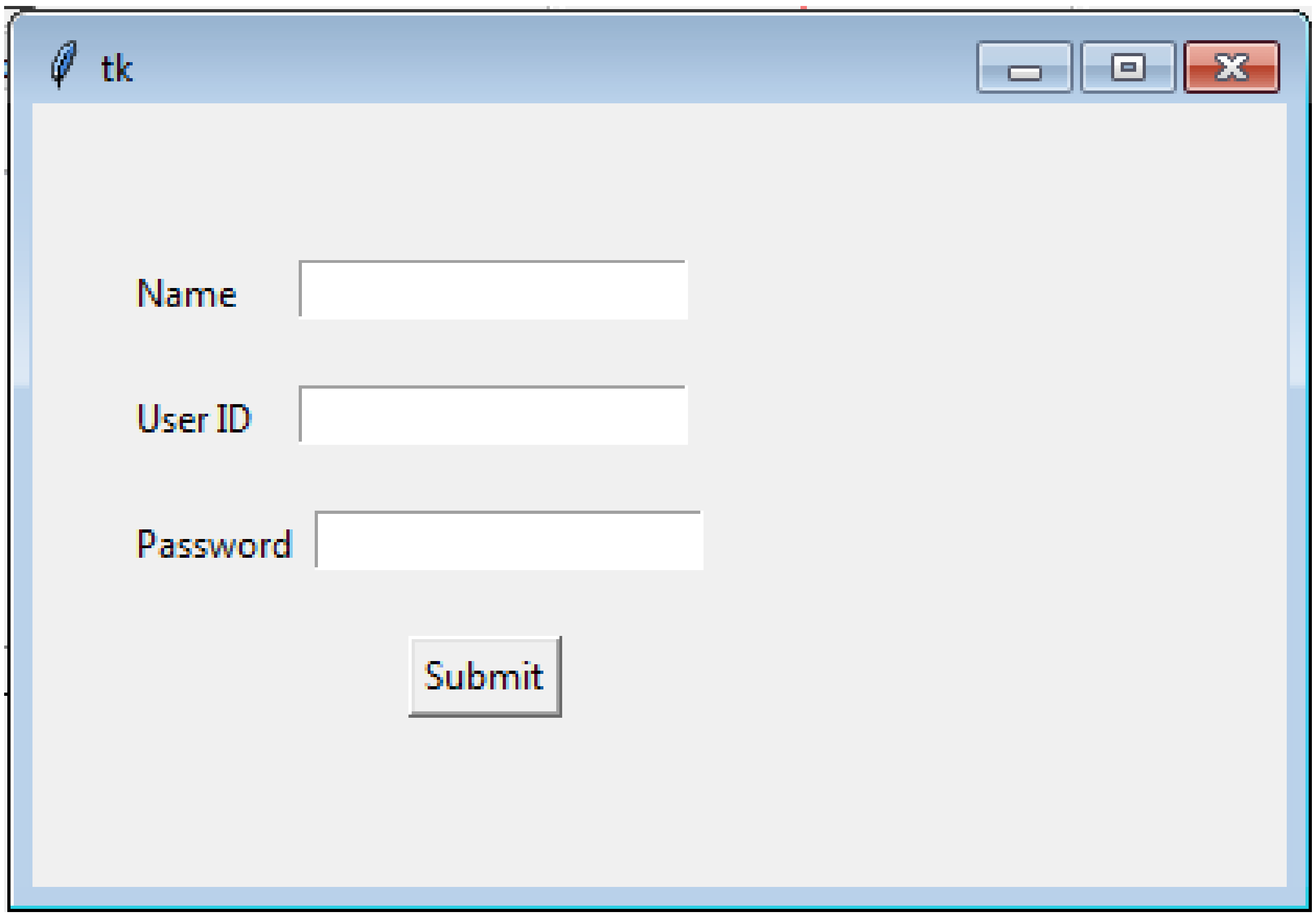
COMP 2012 Object-Oriented Programming and Data Structures

Lab 4 Inheritence: Virtual Function

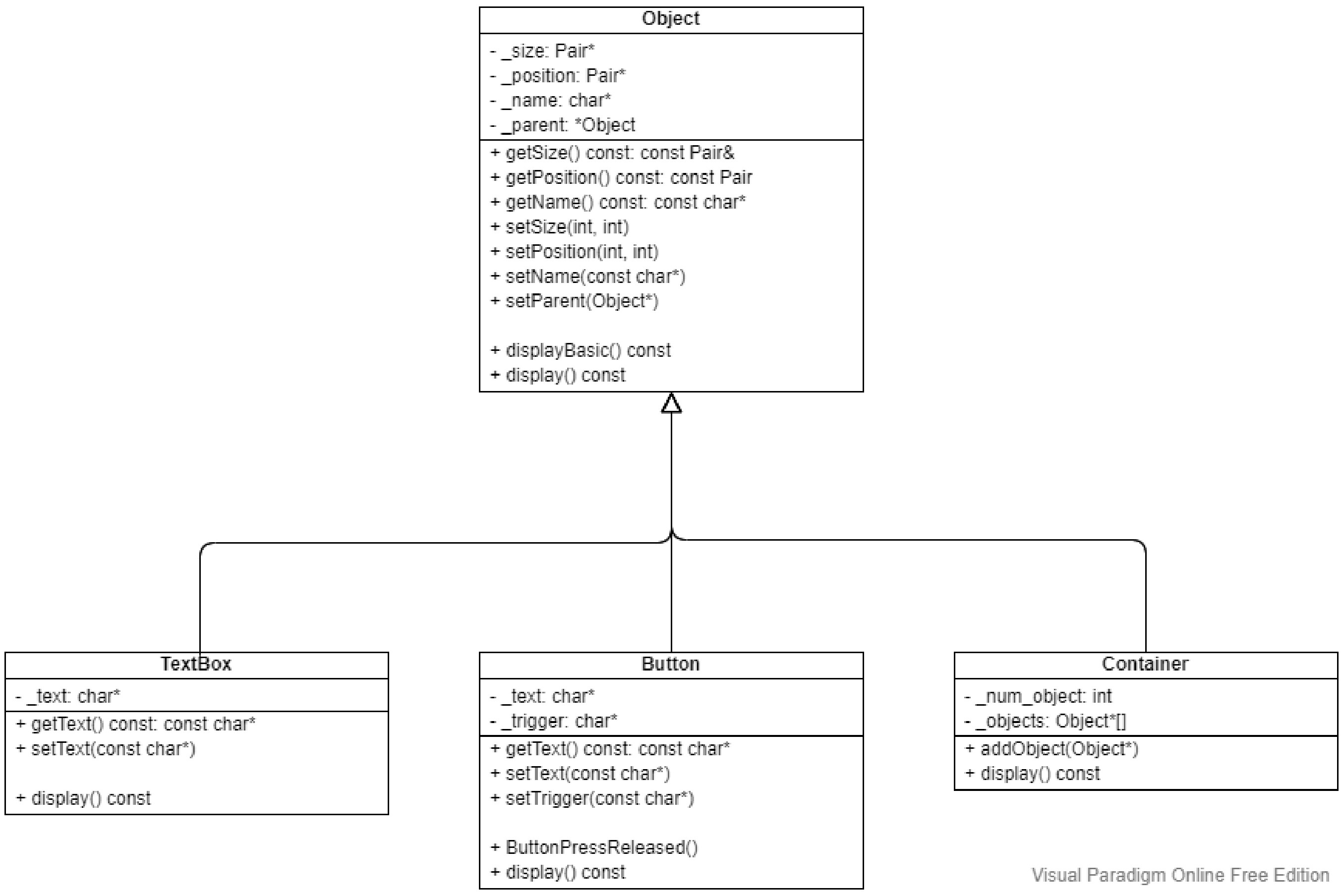


A Python Tkinter widget
Image Source: w3resource, retrieved here.

Introduction

In this lab, we will improve the User Interface (UI) framework in lab3 with virtual functions. We will first provide the class diagram for the updated framework and then summarize the code.

Overview of Code



The UI framework

Same as lab3, there are four classes in lab4.

Class Object

Menu

- Introduction
- Download
- Lab Tasks
- Submission Guidelines

Page maintained by

ZHOU Yuxuan
Last Modified:
10/28/2022 14:00:39

Homepage

Course Homepage

`Object`: The base class of all UI objects. See `object.h/object.cpp`. All UI Objects have properties: size, location and name. They can print these information via the `displayBasic()` function.

Different from lab3, now we try to improve the implementation of most of the classes by inheriting from the `Object` class and using virtual functions. Now the `display()` function in the three derived classes is a pure virtual function in the `Object` class. Each derived class has to implement its own `display()`.

Furthermore, to make the framework more interesting and closer to reality, the data member `_position` represents the relative position to the `Container` that contains this `Object`. If no `Container` contains this `Object`, the `_position` will be the absolute position.

To do this, when calculating the position with `getPosition()`, you have to check whether the object has a `_parent`. If not, then the position is just the value of `_position`; otherwise, you have to add the position of this `Object` and its parent. Consider carefully if its parent also has a parent.

In practice, we usually need to do a sanity check when setting the properties of a class. So in this lab, we will add the sanity check in `setPosition(int x, int y)` and `setSize(int x, int y)`. More specifically, in `setSize(int, int)`, you should check whether the input exceeds the screen resolution (e.g., x > SCREEN_RESOLUTION.x). If the input is invalid, the function should print the following error message:

```
Invalid size of ObjectName
```

and returns without setting the position or size. Similarly, in `setPosition(int,int)`, the function should check whether the input is non-negative (i.e., >=0). If not, please print the following error message:

```
Invalid position of ObjectName
```

and returns. We will also add a sanity check in the class `Container` will introduce that later.

See `object.h/.cpp` for the details.

Class TextBox

The UI object displays a line or lines of plain text. See `textbox.h/textbox.cpp`.

Class 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 pressing a button might generally change something.

Class Container

A UI Object that can organize other UI objects as its components. See `container.h/container.cpp`.

Now the `Container` can add other UI objects as its component via `addObject(Object*)` instead of three different functions in lab3. In the `addObject(Object*)`, you need to check the add the object to the `_objects` if `num_object` is less than `MAX_OBJECT_NUM`. Meanwhile, you need to assign this container as the parent of the object, and add the `num_object`.

As we mentioned before, we will also do the sanity checks in the `addObject(Object*)` function, as the container maybe cannot contain the very large object. So before adding the new object to the `_objects`, you need to check whether the container is big enough to contain the new object, i.e., the new object's size plus position does not exceed the size of the container. If the new object does not pass the check, please print

```
The object ObjectName is too large and cannot be added to ContainerName
```

The 3 derived classes have their own fields and functions besides the members inherited from the base class. As the `display()` is the virtual function, each derived class should implement their own `display()` to print its detailed information.

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 lab 4, every application has a main page that contains all the other UI objects. After constructing every UI objects 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.

There are three test cases. Both expected outputs are provided in the skeleton files. Testcase 3 (the most difficult one) should print the expected output as below.

```
LAB 4: Inheritence: Virtual Inheritance

Invalid position of TextBox13
Invalid position of TextBox13
Invalid size of TextBox13
Invalid size of TextBox13
Invalid size of TextBox13
The object TextBox13 is too large and cannot be added to Page1

Container
  Name: [MainPage]
  Position: (100, 100)
  Size: (1920, 1080)
    #objects: 3


Container
  Name: [Page1]
  Position: (200, 200)
  Size: (960, 640)
    #objects: 6


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

Button
  Name: [Button11]
  Position: (600, 300)
  Size: (100, 300)
    ButtonText=[Press This!]

TextBox
  Name: [TextBox12]
  Position: (300, 500)
  Size: (100, 300)
    TextBoxText=[Back]

Button
  Name: [Button12]
  Position: (300, 600)
  Size: (100, 100)
    ButtonText=[Do not press me!]

Container
  Name: [Page11]
  Position: (500, 500)
  Size: (200, 200)

TextBox
  Name: [TextBox13]
  Position: (300, 300)
  Size: (860, 540)
    TextBoxText=[Welcome]

TextBox
  Name: [TextBox1]
  Position: (200, 200)
  Size: (100, 300)
    TextBoxText=[Welcome Main!]

Button
  Name: [Button1]
  Position: (500, 200)
  Size: (100, 300)
    ButtonText=[Press This Button!]
```

```
Destructing Container MainPage
Destructing Container Page1
Destructing TextBox TextBox11
Destructing Button Button11
Destructing TextBox TextBox12
Destructing Button Button12
Destructing Container Page11
Destructing TextBox TextBox13
Destructing TextBox TextBox1
Destructing Button Button1
```

Lab tasks

You can download the skeleton code HERE.

Task 1

Implement the virtual function `display()` of the class `Button` in `button.cpp`, the class `TextBox` in `textbox.cpp`, and the class `Container` in `container.cpp`. It should print the desired output for buttons according to the expected output. Also implement the `addObject(Object*)` and `~Container()` of the class `Container` in `container.cpp`. Hint: it is similar to the `display()` function and `addButton/addTextbox/addContainer` function in lab 3. You can also get some information about `addObject(Object*)/~Container()` from lab 3.

Task 2

Implement the new `getPosition()` in `object.cpp`. Hint: now the `_position` is relative to the parent, so use a recursive function to get the absolute position of the `object`. Remember to check whether the parent exists!

Task 3

Add the sanity checks in `setPosition(int, int)` and `setSize(int, int)` in `object.cpp`, and `addObject(Object*)` in `container.cpp`. The const int `SCREEN_RESOLUTION` is defined in `object.h`.

Submission

Deadline: 10 minutes after your lab session, Oct 27/28, 2022.

Please submit the following files to ZINC by zipping the following 4 files. ZINC usage instructions can be found here.

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

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 own 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.