CS 330 Homework
Ractor

1 Overview

Your responsibility in this homework is to explore inheritance and polymorphism. You will do this in the context of designing in C++ a library for console-based user interfaces. Your library will wrap around the Ncurses library to allow developers to form a user interface out of buttons, lists, checkboxes, sliders, and labels.

2 Install

Download the Ncurses library and its header files on Ubuntu by entering the following at the command line:

```
sudo apt-get install libncurses5-dev
```

3 Requirements

To receive credit for this homework, you must satisfy these requirements:

1. Place all files in directory `<YOUR-REPOSITORY>/ractor`.
2. All code must run on a standard Linux machine. If you use another operating system, test your code on a Linux machine before submitting.
3. Provide a `makefile` to compile your code with support for debugging (`-g`). Include a `clean` rule that removes all object files and executables.
4. Complete the classes described in the following sections, making all instance variables `private` or `protected`.

3.1 Rectangle

Write class `Rectangle` (declared in `Rectangle.h` and implemented in `Rectangle.cpp`) representing a widget's two-dimensional axis-aligned bounding box with the following interface:

- A constructor that accepts four `int`s representing the box's left, right, top, and bottom coordinates, respectively. The top-left vertex of the rectangle is at (left, top), and the bottom-right vertex is at (right, bottom). Unlike your graphing calculator, the origin in most GUI libraries is at the top-left corner. Assume, then, that the top coordinate is less than bottom.
- Methods `GetLeft`, `GetRight`, `GetTop`, `GetBottom` for getting the rectangle's coordinates. Make these methods `const`, as they do not need to mutate state.
- Methods `SetLeft`, `SetRight`, `SetTop`, `SetBottom` to set the rectangle's coordinates. Each of these methods accepts the new coordinate value as an `int`.

- Methods `GetWidth` and `GetHeight` for getting rectangle's dimensions. A rectangle inclusively spans its coordinates. For example, a rectangle spanning the interval [5, 8] has a width of 4. Make these methods `const`.
- Method `Contains` that accepts two parameters in this order:
  - an `int` x-coordinate
  - an `int` y-coordinate

  It returns a `bool`, which is `true` if and only if the coordinate falls inclusively within the bounds of the rectangle. Make this method `const`.
- Method `Intersects` that accepts a `const` reference to another `Rectangle` as its sole parameter. It returns a `bool`, which is `true` if and only if the two rectangles overlap at some point. You may find it easier to solve the inverse of this problem first: when do two rectangles not intersect?

3.2 Widget

Write class `Widget` (declared in `Widget.h` and implemented in `Widget.cpp`) representing an abstracted widget in the Ractor library. It has the following interface:

- A constructor that accepts four `int`s representing the box's left, right, top, and bottom coordinates, respectively.
- A deconstructor that deletes any dynamically allocated memory. Probably you will have none. However, since this class will serve as a superclass to others, and we may have a `Widget *` pointer or a `Widget &` reference that polymorphically points to an instance of a derived class, we want to make sure that the derived deconstructor gets called. So, provide this method, make it `virtual`, and expect it to be overridden when it needs to be.
- Methods `Contains`, `GetLeft`, `GetRight`, `GetTop`, `GetBottom` that behave just like `Rectangle`'s. In fact, it's tempting to just have `Widget` extend `Rectangle`. Composition should be favored over inheritance, however, as inheritance makes `Widget` vulnerable to changes orchestrated through `Rectangle`'s interface.
- Method `Draw`, which will be implemented by subclasses to draw the widget. At the `Widget` level, there is no drawing to be done. Make this method pure, which designates this class as abstract and effectively forces non-abstract subclasses to implement it. Allow it to be called polymorphically by making it `virtual`.
- Method `OnMouseClick` to handle mouse events. It accepts two parameters in this order:
  - an `int` x-coordinate
  - an `int` y-coordinate

  Implement this method to ignore the event, but do not make it pure. Subclasses will have the option of either inheriting or overriding it. Allow it to be called polymorphically by making it `virtual`.

3.3 Window

Write class `Window` (declared in `Window.h` and implemented in `Window.cpp`) representing a widget that contains other widgets and handles mouse-click events. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts no parameters. The window will not have a fixed bounding box, but will span the entire terminal window. Have it set up its `Widget` foundation so that its bounding box spans (-1, -1) to (-1, -1). Call the following Ncurses functions to initialize our window:
  - `initscr`, to initialize the Ncurses library
  - `noecho`, which disables echoing of keys
  - `curs_set`, to disable the cursor
  - `keypad`, to enable the keypad (and more importantly—and ironically—mouse events) for `stdscr`
  - `mousemask`, to register for `BUTTON1_CLICKED` events
- A deconstructor that deletes all of a window's child widgets.
- Method `Add` that accepts a pointer to a heap-allocated `Widget` as its sole parameter. It takes ownership of the widget and will delete it when the window is destroyed.
- Method `Draw` that first clears the window using Ncurses' `clear` function and then draws all the child widgets in the order they were added.
- Method `Loop` that enters into an event loop, which runs indefinitely, listening for and handling key and mouse events. Follow this pseudocode:

```
while not stopped
  draw the window
  input = getch()
  if input is q
    stop
  else if input is KEY_MOUSE
    get mouse event
    for each widget
      if widgets contains mouse
        delegate event to widget's OnMouseClick
  refresh()
endwin()
```

  Convert the mouse coordinates from global window coordinates to local widget coordinates before passing them to `OnMouseClick`. For example, suppose a `List` spans (10, 13) to (17, 16) and the user clicks on (13, 14). Pass (3, 1) to the widget as the mouse position.

- Method `Stop` that requests the event loop to stop.

3.4 Label

Write class `Label` (declared in `Label.h` and implemented in `Label.cpp`) representing a read-only text label. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts three parameters in this order:
  - the x-coordinate of the label's first character, as an `int`
  - the label's y-coordinate, as an `int`
  - the label's text, as a `const` reference to a `string`

  Have it set up its `Widget` foundation so that its bounding box fits snugly around the text. That is, the label `"foo"` at (10, 17) will span (10, 17) to (12, 17).
- Method `Draw` to draw the label. Use the Ncurses function `mvprintw` to draw the text.
- Method `SetText` to update the label's text. It accepts a `const` reference to a `string` and adjusts the widget's bounding box accordingly. The next time the label is drawn, it will show the updated text.

3.5 Button

Write class `Button` (declared in `Button.h` and implemented in `Button.cpp`) representing a clickable button. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts three parameters in this order:
  - the x-coordinate of the button label's first character, as an `int`
  - the button label's y-coordinate, as an `int`
  - the button's text label, as a `const` reference to a `string`

  Have it set up its `Widget` foundation so that its bounding box fits snugly around the label. That is, the button with label `"Submit"` at (10, 17) will span (10, 17) to (15, 17).
- Method `Draw` to draw the button in *reverse video*. Use the Ncurses function `mvprintw` to draw the text, but sandwich the call with `attron`/`attroff` to enable/disable the `A_REVERSE` attribute.
- Method `AddListener` to register an observer of clicks. It accepts a listener as a parameter of the type `std::function<void()>`, which stands for a function that returns nothing and accepts no parameters. Clients of this class can register lambda listeners in the following way:

```
Button *button = new Button(3, 10, "Quit");
button->AddListener([&]() {
  window.Stop();
});
```

Include the header `functional` to load this type's declaration. Clients may register multiple listeners.

- Method `OnMouseClick` to notify all listeners that the button was clicked. Notify each listener through its apply method (`operator()`). For example: `listener()`.

3.6 Checkbox

Write class `Checkbox` (declared in `Checkbox.h` and implemented in `Checkbox.cpp`) representing a toggle-able checkbox. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts three parameters in this order:
  - the x-coordinate of the checkbox's left bracket character, as an `int`
  - the checkbox's y-coordinate, as an `int`
  - the checkbox's text label, as a `const` reference to a `string`

  The checkbox will be drawn in this form: `[␣]␣label`, where ␣ represents a space character. Have it set up its `Widget` foundation so that its bounding box fits snugly around the label plus the spaces and brackets. That is, the checkbox with label `"vegan"` at (10, 17) will span (10, 17) to (18, 17).
- Method `IsChecked` that gives the checkbox's current state. It returns a `bool`, which is `true` if and only if the checkbox has been checked. Make this method `const`.
- Method `IsChecked` that sets the checkbox's current state. It returns nothing, but accepts a `bool` that is `true` if and only if the checkbox is to be checked.
- Method `Draw` to draw the checkbox. Use the Ncurses function `mvprintw` to draw the brackets, spaces, and text. If the checkbox is checked, place an `'x'` between the brackets instead of a space.
- Method `AddListener` to register an observer of check or uncheck events. It accepts a listener as a parameter of the type `std::function<void(bool)>`, which stands for a function that returns nothing and accepts a `bool`. Clients of this class can register lambda listeners in the following way:

```
Checkbox *box = new Checkbox(3, 10, "vegan");
box->AddListener([&](bool is_vegan) {
  // ...
});
```

  Clients may register multiple listeners.
- Method `OnMouseClick` to toggle the checkbox's state and notify all listeners that the state has changed. Notify each listener through its apply method (`operator(bool)`).

3.7 List

Write class `List` (declared in `List.h` and implemented in `List.cpp`) representing a list of choices that allows no more than one option to be selected. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts three parameters in this order:
  - the x-coordinate of the first character of the first option, as an `int`
  - the y-coordinate of the list's first option, as an `int`
  - the list's options, as a `const` reference to a `vector` of `string`s

  The list will be drawn in this form:

  ```
  option 1
  longer option 2
  option 3
  ```

  Have it set up its `Widget` foundation so that its bounding box fits snugly vertically around the options. Fit it horizontally so that it stretches to include all characters of the longest option. For the given example, the list at (10, 17) will span (10, 17) to (24, 19).
- Method `GetSelectedIndex`, which returns the index of the selected item as an `int`. If no item is selected, -1 is returned. Make this method `const`.
- Method `GetSelected`, which returns the text of the selected item as a `const` reference to a `string`. If no item is selected, the method's behavior is undefined. Make this method `const`.
- Method `Draw` to draw the options. Use the Ncurses function `mvprintw` to draw each. If an option is selected, render it in reverse video as you did with `Button`.
- Method `AddListener` to register an observer of select or deselect events. It accepts a listener as a parameter of the type `std::function<void(int)>`, which stands for a function that returns nothing and accepts an `int`. Clients of this class can register lambda listeners in the following way:

  ```
  List *menu = new List(40, 40, {"slow", "fast", "singularity"});
  box->AddListener([&](int i) {
    // ...
  });
  ```

  Clients may register multiple listeners.
- Method `OnMouseClick` to select or deselect an option and notify all listeners that the state has changed. Use the event's y-coordinate to determine which item was selected. If the item was already selected prior to this click, deselect it. Notify each listener of the index of the selected item through its apply method (`operator(int)`). If no item is selected, report -1 as the selected index.

3.8 Slider

Write class `Slider` (declared in `Slider.h` and implemented in `Slider.cpp`) representing a multiline horizontal slider. It has the following interface:

- `Widget` as its superclass.
- A constructor that accepts three parameters in this order:
  - the x-coordinate of the slider's leftmost tick, as an `int`
  - the y-coordinate of the slider, as an `int`
  - the slider's maximum value, as an `int`
  - the slider's default value, as an `int`

  A slider with a maximum of 10 and a current value of 3 will be drawn in this form:

  ```
  ...o.......
  ```

  The minimum value for a slider is always 0, so this example shows 11 ticks spanning the interval [0, 11]. When the current value is 0, the slider's knob will appear at the leftmost position. Have the constructor set up its `Widget` foundation so that its bounding box fits snugly around the slider. For the given example, the slider at (10, 17) will span (10, 17) to (20, 17).
- Method `GetValue`, which returns the slider's current value as an `int`. Make this method `const`.
- Method `GetMax`, which returns the slider's maximum value as an `int`. Make this method `const`.
- Method `Draw` to draw the slider. Use the Ncurses function `mvprintw` to draw the ticks. Use periods and a lowercase O.
- Method `AddListener` to register an observer of select or deselect events. It accepts a listener as a parameter of the type `std::function<void(int)>`, which stands for a function that returns nothing and accepts an `int`. Clients of this class can register lambda listeners in the following way:

  ```
  Slider *difficulty = new Slider(20, 15, 20, 10);
  difficulty->AddListener([&](int i) {
    // ...
  });
  ```

  Clients may register multiple listeners.
- Method `OnMouseClick` to set the slider's current value and notify all listeners that the state has changed. Use the event's x-coordinate as the slider's new value. Notify each listener of the value through its apply method (`operator(int)`).

3.9 Custom Application

Write in `app.cpp` a `main` function that helps the user complete some task with a useful graphical user interface. Make the task coherent and meaningful—not just a random collection of widgets. Feel free to employ widgets beyond those specified above to achieve your goal. Include a `run` rule in your `makefile` that compiles and runs your application. Share a screenshot (or video!) on Piazza under the `ractor_share` directory with a description of your GUI does.

4 Later Week

To be eligible for later-week submission, you must have a working `makefile` and complete all widgets but `Window`. Your custom application is required for full credit, but not later-week submission.

5 Submission

To submit your work for grading:

1. Run the grading script from your homework directory using `../specs/grade`.
2. Commit and push your work to your repository.
3. Verify that your solution is on Bitbucket by viewing your repository in a web browser.

A passing grading script does not guarantee you credit. Your grade is conditioned on a few things:

- You must meet the requirements described above. The grading script checks some of them, but not all.
- You must successfully submit your code to your repository. Expect to have issues with Git.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. Your instructor employs a vast repertoire of tools to sniff out academic dishonesty, including: drones, moles, and a piece of software called MOSS that rigorously compares your code to every other submission. You don't want to live in a world serviced by those who squeaked by through questionable means. For your future self, career, and family, do your own work.

The grading script allows you to signal your instructor when requirements are met. You only need to send an email if you qualified for later week submission and are resubmitting after the original deadline.