Design Patterns: Decorator & GTKmm

CS 247

University of Waterloo

cs247@uwaterloo.ca

July 7th, 2017
Overview

1. Decorator

2. Gtkmm
   - The Main Program
   - Widgets
   - Buttons
   - Events
   - Containers
   - Useful Widgets
   - Images
   - Dialog Boxes
When:
The Decorator pattern is used when you want to add “extra” features on top of a particular base class without creating subclasses.

Examples:
adding toppings to a pizza, putting on different combinations of clothing, customizing your car, add-ons for your internet browser, etc.
Why Decorate?

The decorator pattern lets us do two things:

- Create arbitrarily many combinations without having a subclass for each one
- Swap out decorations at run-time
Subclassing works well for very few combinations of decorations, but is not very extensible. Every time a new concrete decoration is added, the number of possible subclasses *doubles*. 
Question:
Add toppings to a pizza and compute price for the pizza
A few toppings: Pepperoni GreenPeppers
Question:
Add toppings to a pizza and compute price for the pizza
A few toppings: Pepperoni GreenPeppers Mushrooms
The Decorator Pattern is set up like a linked-list. Each decoration, like a linked-list node, has some behaviour of its own, and a pointer to the next object - which could be another decorator, or the base object.
Because of this setup, functions tend to be recursive. A decorator’s “doSomething()” method will typically do a bit of the work, and then call the contained object’s “doSomething()”. The contained object will then do the same thing - do a bit of the work, then let the function call propagate down until it eventually gets to the original object that’s being decorated.
#include <gtkmm.h>

int main(int argc, char * argv[]) {
    auto app
        = Gtk::Application::create( argc, argv, "GTKmm.Example" );
    Gtk::Window window;
    return app->run(window);
}

Initialize Gtk, create the window for your program and run it. A common design choice is to inherit from Gtk::Window.
Widgets

A GUI consists of many different widgets. Different widgets serve different purposes:

- Labels and Images (display something to the user)
- Buttons (to trigger Events; seen a little later...)
- Textboxes, Menus, etc. (to allow the user to input data)
- Frames and Boxes (to organize and contain other widgets)
Buttons are a type of widget.

- You may set the label via the constructor:
  
  `Gtk::Button(std::string label)`

- Or with `Gtk::Button::set_label(std::string label)`

- Stock buttons are available

- Usually used with events
Events allow the user to interact with the program. The code:

```cpp
button.signal_clicked.connect(sigc::mem_fun(
    *this, &onButtonClicked));
```

links the Gtk::Button button’s click event to the function onButtonClicked.
Sometimes, we want to pass arguments to our event handlers. For this, wrap `sigc::bind` around `sigc::mem_fun`.

```cpp
button.signal_clicked().connect(
    sigc::bind(
        sigc::mem_fun(*this, &onButtonClicked),
        "I’m an argument!"));
```

This code will call `onButtonClicked` with the argument "I’m an argument" when the button is clicked.
Containers

Most widgets in gtkmm can contain only one child (including Gtk::Window). To add multiple elements to a widget, you must pack them first in a container. Typical containers:

- **Box**: Gtk::Box
- **Table**: Gtk::Table

Add items to the box containers with Gtk::Box::pack_start() and Gtk::Box::pack_end() Add items to the table with Gtk::Table::attach()
Useful Widgets

- Gtk::Frame
- Gtk::Label
- Gtk::CheckButton
- Gtk::RadioButton
- Gtk::Entry
Images

- Gtk::Image
- Image::Image(string)
- Use set() on Images
- Gtk::Image does not receive events but can be embedded in a button.
Dialog boxes are broken into two segments: a content and an action area.

- Access content area: `get_content_area()`
- Access action area: `get_action_area()`

Transfer control and show the dialog with run.
End