15OOD

Contents

1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.3)
project(lec15 CXX)
include(.cs211/cmake/CMakeLists.txt)

set(COMMON
        src/ISprite_set.cxx
        src/Widget.cxx
        src/Button_widget.cxx
        src/Widget_host.cxx)

include_directories(src)

add_program(button_example example/button_example.cxx ${COMMON})
target_link_libraries(button_example ge211)
```

2 example/button_example.cxx

```cpp
#include "../src/Button_widget.hxx"
#include "../src/Widget_host.hxx"

#include <ge211.hxx>

#include <iostream>

using namespace widgets;

static void print_message()
{
    std::cerr << "Thank you\n";
}

static Widget* make_button()
{
    Button_widget* button = new Button_widget("Click Me");
    button->on_click(print_message);
    return button;
}

int main()
{
    Widget_host(make_button).run();
}
```

3 src/Button_widget.hxx

```cpp
#pragma once

#include "Widget.hxx"

namespace widgets
{

class Button_widget : public Widget
{
public:
    explicit Button_widget(const std::string& title,
                           int font_size = 20);

    void on_click(std::function<void()>);

protected:
    ge211::Dimensions dimensions() const override;

    void draw(ISprite_set& set, Mouse_state) const override;

    bool click(ge211::Position position) override;

private:
    struct Look
    {
        Look(const std::string&,
             const ge211::Font&,
             const Color_pair&);

        void draw(ISprite_set&) const;

        ge211::Text_sprite      fg;
        ge211::Rectangle_sprite bg;
    };

    ge211::Font font_;
    Look normal_, hover_, active_;

    std::function<void()> on_click_      = [] { };
};

} // end namespace widgets
```

4 src/Button_widget.cxx

```
#include "Button_widget.hxx"

using namespace ge211;
using namespace widgets;

static const char* const font_name{"sans.ttf"};

static const Dimensions padding{10, 10};

static const Color gray{192, 192, 192},
                   black{0, 0, 0},
                   nu_purple{59, 47, 119};

static const Color_pair normal_colors{black, gray},
                        hover_colors{nu_purple, gray},
                        active_colors{gray, nu_purple};

Button_widget::Look::Look(const std::string& text,
                          const ge211::Font& font,
                          const Color_pair& colors)
        : fg{Text_sprite::Builder(font).color(colors.fg).message(text).build()},
          bg{fg.dimensions() + 2 * padding, colors.bg}
{ }

void Button_widget::Look::draw(ISprite_set& set) const
{
    Position fg_pos{(bg.dimensions() - fg.dimensions()) / 2};

    set.add_sprite(bg, {0, 0}, 0);
    set.add_sprite(fg, fg_pos, 1);
}

Button_widget::Button_widget(const std::string& title,
                             int font_size)
        : font_{font_name, font_size},
          normal_{title, font_, normal_colors},
          hover_{title, font_, hover_colors},
          active_{title, font_, active_colors}
{ }

void Button_widget::on_click(std::function<void()> on_click)
{
    on_click_ = on_click;
}

Dimensions Button_widget::dimensions() const
{
    return normal_.bg.dimensions();
}

bool Button_widget::click(ge211::Position position)
{
    on_click_();
    return true;
}

```

```
void Button_widget::draw(ISprite_set& set, Mouse_state st) const
{
    switch (st) {
    case Mouse_state::normal:
        normal_.draw(set);
        break;
    case Mouse_state::hover:
        hover_.draw(set);
        break;
    case Mouse_state::active:
        active_.draw(set);
        break;
    }
}
```

5 src/ISprite_set.hxx

```
#pragma once

#include <ge211.hxx>

namespace widgets {

/// Interface (abstract class) for allowing different classes
/// that are used like `Sprite_set`s.
struct ISprite_set
{
    /// Adds the given sprite with the given position, z index,
    /// and transformation.
    virtual void add_sprite(ge211::Sprite const&,
                            ge211::Position,
                            int z,
                            ge211::Transform) = 0;

    /// Adds the given sprite with the given position and optional
    /// z index.
    virtual void add_sprite(ge211::Sprite const&,
                            ge211::Position,
                            int z = 0);
};

/// Wraps a `ge211::Sprite_set` so that we can use it as an
/// `ISprite_set`.
```

```
class Sprite_set_adapter : public ISprite_set
{
public:
    explicit Sprite_set_adapter(ge211::Sprite_set&);

    void add_sprite(ge211::Sprite const&,
                    ge211::Position,
                    int z,
                    ge211::Transform) override;

private:
    ge211::Sprite_set& adaptee_;
};

/// The data associated with a sprite placement (used by the next two
/// classes).
struct Placed_sprite
{
    const ge211::Sprite* sprite;
    ge211::Position position;
    int z;
    ge211::Transform transform;
};

/// For testing, a transparent sprite set that just collects all
/// the sprites so we can look at them.
struct Mock_sprite_set : ISprite_set
{
    void add_sprite(ge211::Sprite const&,
                    ge211::Position,
                    int z,
                    ge211::Transform) override;

    std::vector<Placed_sprite> sprites;
};

/// A sprite set that can add its sprites to another sprite set
/// at a specified offset (position and z index).
class Sprite_set_layer : public ISprite_set
{
public:
    void add_sprite(ge211::Sprite const&,
                    ge211::Position,
                    int z,
                    ge211::Transform) override;

    /// Adds the sprites in this layer to `set`, offsetting them
    /// so their lowest z index is the given `z` and translating
    /// them by `offset`. Returns the highest z index used.
    int add_to(ISprite_set& set,
               ge211::Dimensions xyoffset = {0, 0},
               int zoffset = 0) const;

private:
    std::vector<Placed_sprite> sprites_;
    int zmin_ = 0, zmax_ = 0;
};

} // end namespace widgets
```

6 src/ISprite_set.cxx

```
#include "ISprite_set.hxx"

using namespace ge211;
using namespace widgets;

void ISprite_set::add_sprite(Sprite const& sprite,
                             Position position,
                             int z)
{
    add_sprite(sprite, position, z, Transform{});
}

Sprite_set_adapter::Sprite_set_adapter(Sprite_set& adaptee)
        : adaptee_{adaptee}
{ }

void Sprite_set_adapter::add_sprite(Sprite const& sprite,
                                    Position position,
                                    int z,
                                    Transform transform)
{
    adaptee_.add_sprite(sprite, position, z, transform);
}

void Mock_sprite_set::add_sprite(Sprite const& sprite,
                                 Position position,
                                 int z,
                                 Transform transform)
{
    sprites.push_back({&sprite, position, z, transform});
}

void Sprite_set_layer::add_sprite(Sprite const& sprite,
                                  Position position,
                                  int z,
                                  Transform transform)
{
    zmin_ = std::min(zmin_, z);
    zmax_ = std::min(zmax_, z);
    sprites_.push_back({&sprite, position, z, transform});
}

int Sprite_set_layer::add_to(ISprite_set& set,
                             Dimensions xyoffset,
                             int z) const
{
    int zoffset = z - zmin_;
    int zfinal  = z + zmax_;

    for (const auto& placed : sprites_)
        set.add_sprite(*placed.sprite,
                       placed.position + xyoffset,
                       placed.z + zoffset,
                       placed.transform);

    return zfinal;
}
```

7 src/Widget.hxx

```
#pragma once

#include <functional>
#include <ISprite_set.hxx>

namespace widgets {

class Widget_host;

class Widget
{
    friend Widget_host;

public:
    virtual ~Widget() {}

    enum class Mouse_state
    {
        normal,
        hover,
        active,
    };

protected:
    /// Returns the dimensions of the widget.
    virtual ge211::Dimensions dimensions() const = 0;

    /// Adds the widget to a sprite set. Gets the current state
    /// of the mouse so that this can affect how it looks.
    virtual void draw(ISprite_set&, Mouse_state) const = 0;

    /// Override this to respond to clicks. Returning `true` if
    /// the click is accepted by the widget, or false to allow the
    /// click to pass through this widget.
    virtual bool click(ge211::Position) { return false; }
};

struct Color_pair
{
    ge211::Color fg, bg;
};

} // end namespace widgets
```

8 src/Widget.cxx

```
#include "Widget.hxx"

using namespace ge211;
using namespace widgets;
```

9 src/Widget_host.hxx

```
#pragma once

#include "Widget.hxx"

#include <ge211.hxx>

#include <memory>

namespace widgets {

/// Attaches a widget to a GE211 window.
class Widget_host : public ge211::Abstract_game
{
public:
    /// Takes ownership of the `Widget*`.
    explicit Widget_host(std::function<Widget*()>);

    /// Since `root_` is owned and will be deleted by the destructor,
    /// it wouldn't be safe to copy a `Widget_host`, so we disable
    /// the copy constructor.
    Widget_host(const Widget_host&) = delete;

    /// Disable the copy-assignment operator for the same reason.
    Widget_host& operator=(const Widget_host&) = delete;

    /// Deletes the root widget.
    ~Widget_host() override;

protected:
    void draw(ge211::Sprite_set&) override;

    void on_mouse_down(ge211::Mouse_button, ge211::Position) override;

    void on_mouse_up(ge211::Mouse_button, ge211::Position) override;

    void on_mouse_move(ge211::Position) override;

private:
    bool is_position_inside_(ge211::Position) const;
    ge211::Rectangle bbox_() const;
    Widget& root_() const;
    Widget::Mouse_state mouse_state_() const;

    std::function<Widget*()> create_;
    mutable Widget* root_ptr_ = nullptr;
    bool is_mouse_inside_ = false;
    bool is_mouse_down_   = false;
};

} // end namespace widgets
```

10 src/Widget_host.cxx

```
#include "Widget_host.hxx"

using namespace ge211;
using namespace widgets;

Widget_host::Widget_host(std::function<Widget*()> create)
        : create_{create}
{ }

Widget_host::~Widget_host()
{
    delete root_ptr_;
}

void Widget_host::draw(Sprite_set& set)
{
    Sprite_set_layer layer;
    root_().draw(layer, mouse_state_());

    Sprite_set_adapter adapter{set};
    Rectangle bbox{bbox_()};
    layer.add_to(adapter, {bbox.x, bbox.y});
}

void Widget_host::on_mouse_down(Mouse_button button, Position posn)
{
    if (button == Mouse_button::left)
        is_mouse_down_ = true;
}

void Widget_host::on_mouse_up(Mouse_button button, Position posn)
{
    if (button == Mouse_button::left) {
        if (is_mouse_inside_ && is_mouse_down_)
            root_().click(posn);

        is_mouse_down_ = false;
    }
}

void Widget_host::on_mouse_move(Position posn)
{
    is_mouse_inside_ = is_position_inside_(posn);
}

static bool is_position_inside(Position posn, Rectangle rect)
{
    auto limit = rect.bottom_right();
    return rect.x <= posn.x && posn.x < limit.x &&
           rect.y <= posn.y && posn.y < limit.y;
}

bool Widget_host::is_position_inside_(Position posn) const
{
    return is_position_inside(posn, bbox_());
}

Widget& Widget_host::root_() const
{
    if (!root_ptr_) root_ptr_ = create_();
    return *root_ptr_;
}

Rectangle Widget_host::bbox_() const
{
    Dimensions window = get_window().get_dimensions();
    Dimensions widget = root_().dimensions();
    Position   top_left{(window - widget) / 2};
    return Rectangle::from_top_left(top_left, widget);
}

Widget::Mouse_state Widget_host::mouse_state_() const
{
    using M = Widget::Mouse_state;

    if (is_mouse_inside_)
        if (is_mouse_down_)
```

```
            return M::active;
        else
            return M::hover;
    else
        return M::normal;
}
```