HW_5

محمدرضا بابایی مصلح

دانشگاه صنعتی امیرکبیر

در این تمرین به استفاده از مسایل چند ریختی و ارث بری پرداختیم.

در ادامه ابتدا به بررسی فایل های h. و سپس به بررسی cpp.
میپردازیم و نکات هرکدام را توضیح میدهیم.

: ingredient.h

```
#ifndef INGREDIENT_H
#define INGREDIENT_H

class Ingredient
{
public:
    double get_price_unit(){return price_unit;}
    size_t get_units(){return units;}
    virtual std::string get_name() = 0;

    double price()
    {
        return price_unit * units;
    }


protected:
    Ingredient(double price_unit, size_t units)
    {
        this->price_unit = price_unit;
        this->units = units;
    }

    double price_unit;
    size_t units;
    std::string name;
};

#endif // INGREDIENT_H
```

در این فایل کل کلاس ingredient تعریف شده است دو نکته که لازم به ذکر است: ابتدا تابع(method) get_name() که با قرار گرفتن

عبارت virtual در پشت ان باعث شده که هر کلاسی که این method را تعریف میکند بتواند از چند ریختی استفاده کند. و با قرار دادن این عبارت مساوی 0 باعث شدیم این کلاس pure virtual بشود به این معنی که هر کلاسی که ازین کلاس ارث میبرد باید این method را تعریف کند و همچنین نمیتوان از این کلاس object درست کرد.

نکته دوم اینکه وجود عبارت protected باعث شده که کلاس هایی که ازین کلاس ارث میبرند به متغیر های داخل ان دسترسی داشته باشند ولی در جاهای دیگر نتوان به ان ها دسترسی پیدا کرد.

:sub_ingredients.h

```
#ifndef SUB_INGREDIENTS_H
#define SUB_INGREDIENTS_H
#include "ingredient.h"
#include <iostream>
#include <string>

#define DEFCLASS(subclass_name, ppu)\
class subclass_name : public Ingredient\
{\
public:\
    subclass_name(size_t units) : Ingredient{ppu, units}\
    {\
        this->name = #subclass_name;\
    };\
    virtual std::string get_name() {return this->name;}\
};

 DEFCLASS(Cinnamon, 5);
 DEFCLASS(Chocolate, 5);
 DEFCLASS(Sugar, 1);
 DEFCLASS(Cookie, 10);
 DEFCLASS(Espresso, 15);
 DEFCLASS(Milk, 10);
 DEFCLASS(MilkFoam, 5);
 DEFCLASS(Water, 1);
```

در این فایل برای تعریف کلاس هایی که از کلاس ingredient ارث میبرند با توجه به اینکه تعریف های مشابهی دارند و فقط یک ورودی ان ها فرق میکند از macro استفاده کردیم.

: Espresso_based

```
#ifndef ESPRESSO_BASED_H
#define ESPRESSO_BASED_H
#include<vector>
#include<string>
#include"sub_ingredients.h"
#include <ftxui/dom/elements.hpp>
#include <ftxui/screen/screen.hpp>
#include "ftxui/dom/node.hpp"      // for Render
#include "ftxui/screen/color.hpp"  // for ftxui
#include <iostream>
#include <thread>

class EspressoBased
{
public:
    virtual std::string get_name() = 0;
    virtual double price() = 0;

    void brew();
    std::vector<Ingredient*>& get_ingredients();

    virtual ~EspressoBased();

protected:
    EspressoBased();
    EspressoBased(const EspressoBased& esp);
    void operator=(const EspressoBased& esp);

    std::vector<Ingredient*> ingredients;
    std::string name;

};

#endif // ESPRESSO_BASED_H
```

در فایل بالا مشاهده میشود که با توجه به تعریف متغیر های داینامیک destructor هم تعریف شده و در پشت ان عبارت virtual قرار گرفته شده است زیرا کلاس هایی که از این کلاس ارث میبرند بایدdestructor خود را هم صدا بزنند.

همچنین تعداد زیادی header با پسوند یکسان هم include شده اند که مربوط به کار کردن با gui میباشند که در ادامه توضیح داده میشود.

: cappuccino

```
#ifndef CAPPUCCINO
#define CAPPUCCINO
#include<string>
#include<vector>
#include<espresso_based.h>

class Cappuccino : public EspressoBased
{
public:
    Cappuccino();
    Cappuccino(const Cappuccino& cap);
    virtual ~Cappuccino();
    void operator=(const Cappuccino& cap);

    virtual std::string get_name();
    virtual double price();

    void add_side_item(Ingredient* side);
    std::vector<Ingredient*>& get_side_items();

private:
    std::vector<Ingredient*> side_items;

};

#endif // CAPPUCCINO
```

تنها نکته فایل بالا ارث بری از کلاس espresso_based است

: mocha.h

```h
#ifndef MOCHA_H
#define MOCHA_H

#include<string>
#include<vector>
#include<espresso_based.h>

class Mocha : public EspressoBased
{
public:
    Mocha();
    Mocha(const Mocha& cap);
    virtual ~Mocha();
    void operator=(const Mocha& cap);

    virtual std::string get_name();
    virtual double price();

    void add_side_item(Ingredient* side);
    std::vector<Ingredient*>& get_side_items();

private:
    std::vector<Ingredient*> side_items;

};


#endif // MOCHA_H
```

دقیقا مشابه کلاس cappuccino است.

حال به سراغ فایل های cpp. میرویم.

Espresso_based.cpp

```cpp
#include "espresso_based.h"
#include <cmath>

EspressoBased::EspressoBased()
    :name{""}
    ,ingredients{}
{
};


EspressoBased::EspressoBased(const EspressoBased& esp)
    :name{esp.name}
{
    for (Ingredient* component : esp.ingredients)
    {
        if(component->get_name() == "Cinnamon")
        {
            ingredients.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            ingredients.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            ingredients.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            ingredients.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            ingredients.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
        {
            ingredients.push_back(new Milk{component->get_units()});
        }
        else if(component->get_name() == "MilkFoam")
```

```
        {
            ingredients.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            ingredients.push_back(new Water{component->get_units()});
        }

    }
};


void EspressoBased::operator=(const EspressoBased& esp)
{
    if(this == &esp)
    {
        return;
    }

    ingredients.clear();

    for (Ingredient* component : esp.ingredients)
    {
        if(component->get_name() == "Cinnamon")
        {
            ingredients.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            ingredients.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            ingredients.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            ingredients.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            ingredients.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
        {
```

```
            ingredients.push_back(new Milk{component->get_units()});
        }
        else if(component->get_name() == "MilkFoam")
        {
            ingredients.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            ingredients.push_back(new Water{component->get_units()});
        }

    }
    name = esp.name;
};


EspressoBased::~EspressoBased()
{
    for(const auto& i : ingredients)
        delete i;
    ingredients.clear();
}


void EspressoBased::brew()
{
    using namespace ftxui;
    using namespace std::chrono_literals;

    double sum_unit{};
    for (auto components : ingredients)
    {
        sum_unit += components->get_units();
    }

    auto step{ 1 / sum_unit };
    //std::cout << step << std::endl;

    std::string reset_position{};
    double ratio{};
    std::vector<std::string> sub_component;
    sub_component.push_back("");
    sub_component.push_back("");
    sub_component.push_back("");
    sub_component.push_back("");
```

```
Decorator color1{color(Color::Blue)};
Decorator color2{color(Color::Yellow)};
Decorator color3{color(Color::Green)};
Decorator color4{color(Color::White)};

std::vector< Decorator> color_coll;
std::vector< Decorator> color_coll_itr;

color_coll_itr.push_back(color1);
color_coll_itr.push_back(color2);
color_coll_itr.push_back(color3);
color_coll_itr.push_back(color4);

color_coll.push_back(color(Color::Black));
color_coll.push_back(color(Color::Black));
color_coll.push_back(color(Color::Black));
color_coll.push_back(color(Color::Black));
size_t counter{};
for (auto components : ingredients)
{
    if(size(ingredients) == 4)
        sub_component[3] = sub_component[2];
    sub_component[2] = sub_component[1];
    sub_component[1] = sub_component[0];
    sub_component[0] = components->get_name();

    if(size(ingredients) == 4)
        color_coll[3] = color_coll[2];
    color_coll[2] = color_coll[1];
    color_coll[1] = color_coll[0];
    color_coll[0] = color_coll_itr[counter];

    auto partial_step{step / (components->get_units() * 100)};
    for (int i{} ; i < components->get_units() * 100 ; i++)
    {
            std::string txt{std::to_string((ratio * 100))};
            txt.append("/100");
                Element document = hbox({
            text("brewing: ") | color(Color::Blue)
            ,gaugeRight( ratio ) | color(Color::Magenta)
            ,text(txt)
            ,border(vbox({
                text(sub_component[0]) | border | color_coll[0]
                ,text(sub_component[1]) | border | color_coll[1]
```

```
                    ,text(sub_component[2]) | border | color_coll[2]
                    ,text(sub_component[3]) | border | color_coll[3]


                })))
                ,text("          ")
            });

            ratio += partial_step * components->get_units();
            auto screen = Screen::Create(Dimension::Full(),
Dimension::Fit(document));
            Render(screen, document);
            std::cout << reset_position;
            screen.Print();
            reset_position = screen.ResetPosition();

            std::this_thread::sleep_for(0.01s);
        }
        counter++;
    }
            Element document = hbox({
            text("brewing: ") | color(Color::Blue)
            ,gaugeRight( ratio ) | color(Color::Magenta)
            ,text("100/100")
            ,border(vbox({
             text(sub_component[0]) | border | color_coll[0]
            ,text(sub_component[1]) | border | color_coll[1]
            ,text(sub_component[2]) | border | color_coll[2]
            ,text(sub_component[3]) | border | color_coll[3]
        }))
        ,text("                    ")
    });
    auto screen = Screen::Create(Dimension::Full(), Dimension::Fit(document));
    Render(screen, document);
    std::cout << reset_position;
    screen.Print();
    reset_position = screen.ResetPosition();
    std::this_thread::sleep_for(1s);



    std::cout << "" << std::endl;
}
```

```
std::vector<Ingredient*>& EspressoBased::get_ingredients()
{
    return ingredients;
}
```

فایل بالا مهم ترین فایل این تمرین است به ترتیب به توضیحات ان میپردازیم.

اولین نکته در copy constructor ان است که با توجه به اینکه باید برای هر المان یک پوینتر new کنیم و همچینین اجازه ساختن object از کلاس ingredients را نداریم پس به ناچار برای هر حالت یک شرط باید تعریف کنیم.

دومین نکته در operator= است که بسیار شبیه به copy constructor است به غیر از ابتدای ان که باید ingredient را خالی کنیم.

سومین نکته که سخت ترین بخش نیز بود مربوط به gui در متد brew میباشد که برای ان از ftxui استفاده کردیم و با مطالعه documentation ان به تکمیل کردن method پرداختیم

برای این کار برای هر ماده از قبل یک متن خالی قرار دادیم و یک رنگ نیز به ان اختصاص دادیم که در صورت استفاده از ان به لیوان ما اضافه میشوند همچنین برای تکمیل کردن نوارصورتی رنگ از gauge استفاده کردیم که مقدار step ان را از جمع کردن مقدار unit های مواد تشکیل دهنده و صد برابر کردن ان استفاده کردیم سپس با گردش روی ingredients کاری کردیم که نوار به اندازه هر نسبت هر unit به کل ان ها طول بکشد و با این کار ها را با استفاده از hbox انجام دادیم و

در داخل ان با فراخوانیvbox به ساختن cup خود پرداختیم و هربار که ایتم جدید خوانده میشد ایتم قبلی یک واحد به پایین انتقال داده میشود.

نکته اخر مربوط به screen میشود که کلاسی است که بستر نمایش برای ما است از انجایی که ما داریم از ترمینال به عنوان gui استفاده میکنیم تمام عبارت ها و اشکال در اصل به صورت یک string ذخیره و سپس نمایش داده میشوند و طول و ابعاد این screenمطابق با کد بالا مشخص میشود

در انتهای هر حلقه هسته ای که در حال process کردن اطلاعات است را 1 ثانیه خاموش میکنیم تا تصویر ها به صورت farme by frame روی هم بیافتد و تصاویر به ظاهر متحرک دیده شوند.

Cappuccino.cpp

```cpp
#include "cappuccino.h"



Cappuccino::Cappuccino()
    : EspressoBased{}
    , side_items{}
{
    Milk* milk{new Milk{2}};
    Espresso* espresso{new Espresso{2}};
    MilkFoam* milkfoam{new MilkFoam{1}};
    ingredients.push_back(espresso);
    ingredients.push_back(milk);
    ingredients.push_back(milkfoam);
    name = "Cappuccino";
};


Cappuccino::Cappuccino(const Cappuccino& cap)
    :EspressoBased{cap}
{
    for (Ingredient* component : cap.side_items)
    {
        if(component->get_name() == "Cinnamon")
        {
```

```
            side_items.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            side_items.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            side_items.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            side_items.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            side_items.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
        {
            side_items.push_back(new Milk{component->get_units()});
        }
        else if(component->get_name() == "MilkFoam")
        {
            side_items.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            side_items.push_back(new Water{component->get_units()});
        }

    }
};


void Cappuccino::operator=(const Cappuccino& cap)
{
    if(this == &cap)
    {
        return;
    }

    side_items.clear();

    for (Ingredient* component : cap.side_items)
```

```
    {
        if(component->get_name() == "Cinnamon")
        {
            side_items.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            side_items.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            side_items.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            side_items.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            side_items.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
        {
            side_items.push_back(new Milk{component->get_units()});
        }
        else if(component->get_name() == "MilkFoam")
        {
            side_items.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            side_items.push_back(new Water{component->get_units()});
        }

    }
};


Cappuccino::~Cappuccino()
{
    for(const auto& i : side_items)
        delete i;
    side_items.clear();
};
```

```cpp
std::string Cappuccino::get_name()
{
    return name;
}


double Cappuccino::price()
{
    double value{};
    for (Ingredient* component : ingredients)
    {
        value += (*component).price();
    }

    for (Ingredient* component : side_items)
    {
        value += (*component).price();
    }

    return value;
}

void Cappuccino::add_side_item(Ingredient* side)
{
    side_items.push_back(side);
}

std::vector<Ingredient*>& Cappuccino::get_side_items()
{
    return side_items;
}
```

فایل بالا نکته خاصی ندارد تنها نکته ان new کردن یک اشاره گر برای هر ingredient موجود در ان است که تعداد unit ها را هم همان موقع میگیرد بقیه قسمت ها دقیقا مشابه espresso_based است.

Mocha.cpp

```cpp
#include "mocha.h"
```

```cpp
Mocha::Mocha()
    : EspressoBased{}
    , side_items{}
{
    Milk* milk{new Milk{2}};
    Espresso* espresso{new Espresso{2}};
    MilkFoam* milkfoam{new MilkFoam{1}};
    Chocolate* chocolate{new Chocolate{1}};
    ingredients.push_back(milk);
    ingredients.push_back(espresso);
    ingredients.push_back(milkfoam);
    ingredients.push_back(chocolate);
    name = "Mocha";
};


Mocha::Mocha(const Mocha& cap)
    :EspressoBased{cap}
{
    for (Ingredient* component : cap.side_items)
    {
        if(component->get_name() == "Cinnamon")
        {
            side_items.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            side_items.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            side_items.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            side_items.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            side_items.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
        {
            side_items.push_back(new Milk{component->get_units()});
        }
```

```
        else if(component->get_name() == "MilkFoam")
        {
            side_items.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            side_items.push_back(new Water{component->get_units()});
        }

    }
};


void Mocha::operator=(const Mocha& cap)
{
    if(this == &cap)
    {
        return;
    }

    side_items.clear();

    for (Ingredient* component : cap.side_items)
    {
        if(component->get_name() == "Cinnamon")
        {
            side_items.push_back(new Cinnamon{component->get_units()});
        }
        else if(component->get_name() == "Chocolate")
        {
            side_items.push_back(new Chocolate{component->get_units()});
        }
        else if(component->get_name() == "Sugar")
        {
            side_items.push_back(new Sugar{component->get_units()});
        }
        else if(component->get_name() == "Cookie")
        {
            side_items.push_back(new Cookie{component->get_units()});
        }
        else if(component->get_name() == "Espresso")
        {
            side_items.push_back(new Espresso{component->get_units()});
        }
        else if(component->get_name() == "Milk")
```

```
        {
            side_items.push_back(new Milk{component->get_units()});
        }
        else if(component->get_name() == "MilkFoam")
        {
            side_items.push_back(new MilkFoam{component->get_units()});
        }
        else if(component->get_name() == "Water")
        {
            side_items.push_back(new Water{component->get_units()});
        }

    }
};


Mocha::~Mocha()
{
    for(const auto& i : side_items)
        delete i;
    side_items.clear();
};


std::string Mocha::get_name()
{
    return name;
}


double Mocha::price()
{
    double value{};
    for (Ingredient* component : ingredients)
    {
        value += (*component).price();
    }

    for (Ingredient* component : side_items)
    {
        value += (*component).price();
    }

    return value;
}
```

```cpp
void Mocha::add_side_item(Ingredient* side)
{
    side_items.push_back(side);
}

std::vector<Ingredient*>& Mocha::get_side_items()
{
    return side_items;
}
```

دقیقا مشابه با cappuccino است و فقط یک ingredient بیشتر دارد

پایان.

Github link: https://github.com/ghostoftime111/hw5.git