به نام خدا

گزارش تمرین سری پنجم AP

محمدمهدی مالوردی

9723079

استاد جهانشاهی

توابع پیاده سازی شده در ingredient.h به صورت زیر می باشد:

include > C ingredient.h

```
#ifndef INGREDIENT_H
#define INGREDIENT_H

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

    double price(){return price_unit*units;}


protected:
    Ingredient(double price_unit, size_t units) :  price_unit{price_unit},units{units}
    {

    }

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


#endif // INGREDIENT_H
```

فراخوانی و تعریف توابع در ingredient.h

include > C sub_ingredients.h

```h
#ifndef SUB_INGREDIENTS_H
#define SUB_INGREDIENTS_H

#include "ingredient.h"

/*
class Cinnamon : public Ingredient
{
public:
    Cinnamon(size_t units) : Ingredient{5, units}
    {
        this->name = "Cinnamon";
    }

    virtual std::string get_name() {return this->name;}
};

class Chocolate : public Ingredient
{
public:
    Chocolate(size_t units) : Ingredient{5, units}
    {
        this->name = "Chocolate";
    }

    virtual std::string get_name() {return this->name;}
};

class Sugar : public Ingredient
{
public:
    Sugar(size_t units) : Ingredient{1, units}
    {
        this->name = "Sugar";
    }

    virtual std::string get_name() {return this->name;}
};

class Cookie : public Ingredient
{
public:
```

```h
public:
    Cookie(size_t units) : Ingredient{10, units}
    {
        this->name = "Cookie";
    }

    virtual std::string get_name() {return this->name;}
};

class Espresso : public Ingredient
{
public:
    Espresso(size_t units) : Ingredient{15, units}
    {
        this->name = "Espresso";
    }

    virtual std::string get_name() {return this->name;}
};

class Milk : public Ingredient
{
public:
    Milk(size_t units) : Ingredient{10, units}
    {
        this->name = "Milk";
    }

    virtual std::string get_name() {return this->name;}
};

class MilkFoam : public Ingredient
{
public:
    MilkFoam(size_t units) : Ingredient{5, units}
    {
        this->name = "MilkFoam";
    }

    virtual std::string get_name() {return this->name;}
};
```

```
};

class Water : public Ingredient
{
public:
    Water(size_t units) : Ingredient{1, units}
    {
        this->name = "Water";
    }

    virtual std::string get_name() {return this->name;}
};
*/

#define DEFCLASS(NAME, PRICE)                          \
    class NAME : public Ingredient                     \
    {                                                   \
    public:                                             \
        NAME(size_t units)                             \
            : Ingredient{PRICE, units}                   \
        {                                               \
            name = #NAME;                              \
        }                                               \
        virtual std::string get_name() { return 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);



#endif // SUB_INGREDIENTS_H
```

فراخوانی و تعریف توابع در challenge + sub_ingredients.h

include > C espresso_based.h

```cpp
#ifndef ESPRESSO_BASED_H
#define ESPRESSO_BASED_H

#include <string>
#include <vector>
#include "ingredient.h"

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
```

فراخوانی توابع در espresso_based.h

src > espresso_based.cpp

```cpp
#include "espresso_based.h"


EspressoBased::EspressoBased(const EspressoBased& esp){


    name = esp.name;
    ingredients = esp.ingredients;

}

EspressoBased::EspressoBased(){
    ingredients.clear();
    name = "";

};

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

void EspressoBased::operator=(const EspressoBased& esp){
    name = esp.name;
    ingredients = esp.ingredients;

}

EspressoBased::~EspressoBased()
{
   for(auto& i : ingredients){
    i = nullptr;
     delete i;}

    ingredients.clear();
}
```

تعریف توابع در espresso_based.cpp

include > C cappuccino.h

```
1   #ifndef CAPPUCCINO
2   #define CAPPUCCINO
3
4   #include <string>
5   #include <vector>
6   #include "sub_ingredients.h"
7   #include "espresso_based.h"
8
9   class Cappuccino : public EspressoBased
10  {
11  public:
12      Cappuccino();
13      Cappuccino(const Cappuccino& cap);
14      ~Cappuccino();
15      void operator=(const Cappuccino& cap);
16
17      virtual std::string get_name();
18      virtual double price();
19
20      void add_side_item(Ingredient* side);
21      std::vector<Ingredient*>& get_side_items();
22
23  private:
24      std::vector<Ingredient*> side_items;
25
26  };
27
28  #endif // CAPPUCCINO
```

فراخوانی توابع در cappuccino.h

src > cappuccino.cpp

```cpp
#include "cappuccino.h"


Cappuccino::Cappuccino() : EspressoBased()
{
    side_items.clear();
    name = "Cappuccino";
    ingredients.push_back(new Espresso{2});
    ingredients.push_back(new Milk{2});
    ingredients.push_back(new MilkFoam{1});
}
Cappuccino::Cappuccino(const Cappuccino& cap){
    side_items = cap.side_items;
    name = cap.name;
    ingredients = cap.ingredients;
}

void Cappuccino::operator=(const Cappuccino& cap){
    side_items = cap.side_items;
    name = cap.name;
    ingredients = cap.ingredients;
}

std::string Cappuccino::get_name(){return name;}
double Cappuccino::price(){
    double sum{0};

    for(auto i : ingredients){
        sum += i->price();
    }
    for(auto i : side_items){
        sum += i->price();
    }
    return sum;

}

void Cappuccino::add_side_item(Ingredient* side){
    side_items.push_back(side);
};
std::vector<Ingredient*>& Cappuccino::get_side_items(){return side_items;}
```

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

Cappuccino::~Cappuccino()
{
    for(auto& i : side_items){
    i = nullptr;
     delete i;}

    side_items.clear();
}
```

تعریف توابع در cappuccino.cpp

include > C mocha.h

```h
#ifndef MOCHA_H
#define MOCHA_H


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

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

    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
```

فراخوانی توابع در mocha.h

src > mocha.cpp

```cpp
#include "mocha.h"

Mocha::Mocha() : EspressoBased()
{
    side_items.clear();
    name = "Mocha";
    ingredients.push_back(new Espresso{2});
    ingredients.push_back(new Milk{2});
    ingredients.push_back(new MilkFoam{1});
    ingredients.push_back(new Chocolate{1});
}
Mocha::Mocha(const Mocha& moc){
    side_items = moc.side_items;
    name = moc.name;
    ingredients = moc.ingredients;
}

void Mocha::operator=(const Mocha& moc){
    side_items = moc.side_items;
    name = moc.name;
    ingredients = moc.ingredients;
}

std::string Mocha::get_name(){return name;}
double Mocha::price(){
    double sum{0};

    for(auto i : ingredients){
        sum += i->price();
    }
    for(auto i : side_items){
        sum += i->price();
    }
    return sum;
}

void Mocha::add_side_item(Ingredient* side){
    side_items.push_back(side);
};
std::vector<Ingredient*>& Mocha::get_side_items(){return side_items;}
std::vector<Ingredient*>& Mocha::get_side_items(){return side_items;}

Mocha::~Mocha()
{
    for(auto& i : side_items){
    i = nullptr;
    delete i;}

    side_items.clear();
}
```

تعریف توابع در mocha.cpp

کلاس ingredient و توابع آن به مانند هر کلاس ساده و توابع معمول آن نوشته شدند و چالش خاصی نداشتند.

کلاس sub_ingredients را ابتدا به صورت کپی کردن از نمونه نوشته شده در صورت تمرین نوشتم و برنامه درست کار کرد. در انتها برای انجام چالش با کمی جستجو، کلاس قالبی به کمک ماکرو ها تعریف کرده و از کپی پیست کردن های بیهوده جلوگیری کردیم.

```
#define DEFCLASS(NAME, PRICE)                                   \
    class NAME : public Ingredient                             \
    {                                                           \
    public:                                                     \
        NAME(size_t units)                                     \
            : Ingredient{PRICE, units}                             \
        {                                                       \
            name = #NAME;                                      \
        }                                                       \
        virtual std::string get_name() { return 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);
```

باید دقت کنیم که چه در sub_ingredients و چه در بقیه کلاس های چایلد و مشتق شده، ارث بری ها به صورت پابلیک اند که دسترسی ها مختل نشود . همچنین هم در مقابل تعریف کلاس و هم کانستراکترش باید این ارث بری اعلان شود.

همچنین مجبور شدم که دیستراکتور کلاس EspressoBased را نیز virtual کنم تا تست های شش و ده به درستی کار کنند.

اگر EspressoBased~ در قسمت protected قرار می گرفت به ارور زیر بر میخوردیم:

```
/usr/src/app/src/unit_test.cpp:68:12: error: 'EspressoBased::~EspressoBased()' is protected within this context
   68 |     delete esp;
      |            ^~~
In file included from /usr/src/app/src/unit_test.cpp:5:
/usr/src/app/include/espresso_based.h:23:5: note: declared protected here
   23 |     ~EspressoBased();
      |     ^
/usr/src/app/src/unit_test.cpp: In member function 'virtual void HW5Test_TEST10_Test::TestBody()':
/usr/src/app/src/unit_test.cpp:111:12: error: 'EspressoBased::~EspressoBased()' is protected within this context
  111 |     delete esp;
```

که چون حافظه دینامیک داریم و در تست ها از دستور delete استفاده می کنیم، این دسترسی ما را محدود می کند و نمی توانیم دیگر به همان شکل از دستوراتی مثل delete استفاده کنیم. برای همین آن را در قسمت پابلیک می گذاریم.

باید حواسمان باشد که برای محاسبه price در کاپوچینو و موکا، قیمت side_items را هم به قیمت بقیه بخش ها اضافه کنیم.

همچنین دستورات دیستراکتور ها را کمی باید تغییر می دادیم و قبل دیلیت کردن حافظه دینامیک، آن را برابر با nullptr نیز قرار می دادیم.

لینک گیت هاب:

https://github.com/mahdimld/AP-HW05