گزارش کار

HW5

دانش عبداللهی
9723053

ابتدا کلاس Ingredient را در فایل ingredient.h تعریف می‌کنیم. در تعریف ایک کلاس از کد گذاشته شده در صورت سوال استفاده می‌کنیم. در تعریف ایت کلاس نکتهٔ خاصی وجود ندارد فقط اینکه طبق صورت سوال تابع get_name() را به صورت زیر تعریف می‌کنیم تا یک تابع Pure Virtual باشد و در نتیجه کلاس Ingredient هم کلاس مجازی (Abstract) باشد.

```
virtual std::string get_name() = 0; // Pure virtual function
```

- جواب سوال پرسیده شده در صورت سوال :

اگر Constructor و متغیرهای این کلاس را به جای Protected به صورت Private تعریف می‌کردیم، دیگر نمی‌توانستیم در کلاس‌های مشتق‌شده از این کلاس به Constructor و متغیرهای این کلاس دسترسی داشته باشیم و چون که این کلاس یک کلاس مجازی (Abstract) می‌باشد ، Private تعریف کردن Constructor و متغیرهایش کار مناسبی نیست چرا که حتما این کلاس ، یک

کلاس پایه (Base) برای کلاس دیگری خواهد بود و آن کلاس نیاز به دسترسی به Constructor و شاید متغیرهای این کلاس خواهد داشت.

در ادامه 8 کلاس مختلف را در فایل Sub_ingredients.h تعریف می‌کنیم. تمام 8 کلاس را با استفاده از Macro به صورت زیر تعریف می‌کنیم: (دقت شود که تمام این کلاس‌ها تعریف مشابه هم داشتند و فقط نام و مقدار price_unit آنها با هم تفاوت داشت.)

```
// Define All Sub_Ingredients Classes With Macro

#define DEFCLASS(Class_Name, Price_Unit)                \
                                                        \
    class Class_Name : public Ingredient {              \
    public:                                             \
        Class_Name(size_t units)                        \
            : Ingredient { Price_Unit, units }          \
        {                                               \
            name = #Class_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);
```

در ادامه به سراغ نوشتن کلاس EspressoBassed می‌رویم. این کلاس هم ، یک کلاس مجازی است.

نکتهٔ قابل توجه در نوشتن این کلاس این است که در پیاده‌سازی Operator = و Copy Constructor ، نباید مقادیر متغیرهای ingredients ها را عینا داخل هم بریزیم. در صورت انجام این کار ، به دلیل اینکه مقادیر متغیر ingredients از نوع اشاره‌گر هستند ، به ارورر double free برخورد خواهیم کرد.

برای حل مشکل بالا ، باید در هنگام ریختن متغیر ingredients ، نام آنرا با استفاده از تابع get_name() بخوانیم و با توجه به نامش ، یک اشاره‌گر از جنس آن کلاس با مقادیر معلوم (که با استفاده از تابع get_units() می‌گیریم) می‌سازیم و آنرا در متغیر ingredients دیگری می‌ریزیم. به صورت زیر :

```
for (const auto& item : esp.ingredients) {
    std::string Temp_Name { item->get_name() };
    size_t Temp_Units { item->get_units() };

    // Add the New Sub_ingredient to the Ingredients vector
    if (Temp_Name == "Cinnamon")
        ingredients.push_back(new Cinnamon { Temp_Units });

    else if (Temp_Name == "Chocolate")
        ingredients.push_back(new Chocolate { Temp_Units });

    else if (Temp_Name == "Sugar")
        ingredients.push_back(new Sugar { Temp_Units });

    else if (Temp_Name == "Cookie")
        ingredients.push_back(new Cookie { Temp_Units });

    else if (Temp_Name == "Espresso")
        ingredients.push_back(new Espresso { Temp_Units });

    else if (Temp_Name == "Milk")
        ingredients.push_back(new Milk { Temp_Units });

    else if (Temp_Name == "MilkFoam")
        ingredients.push_back(new MilkFoam { Temp_Units });

    else if (Temp_Name == "Water")
        ingredients.push_back(new Water { Temp_Units });
}
```

در تعریف این کلاس ، برای نوشتن تابع brew() ، از کتابخانهٔ FTXUI استفاده می‌کنیم.

برای این ابتدا فایل CMakeLists.txt را با استفاده از راهنمایی‌های سایت بالا تغییر می‌دهیم و همچنین کتابخانه‌های مورد نیازمان را در فایل espressobassed.h اضافه می‌کنیم. این بخش نکتهٔ خاص دیگری برای گزارش ندارد.

***Gif قسمتی از تابع brew()

نکتهٔ دیگر آن که Destructor این کلاس را با توجه به این که یک کلاس Base است ، به صورت Virtual (مجازی) تعریف می‌کنیم.

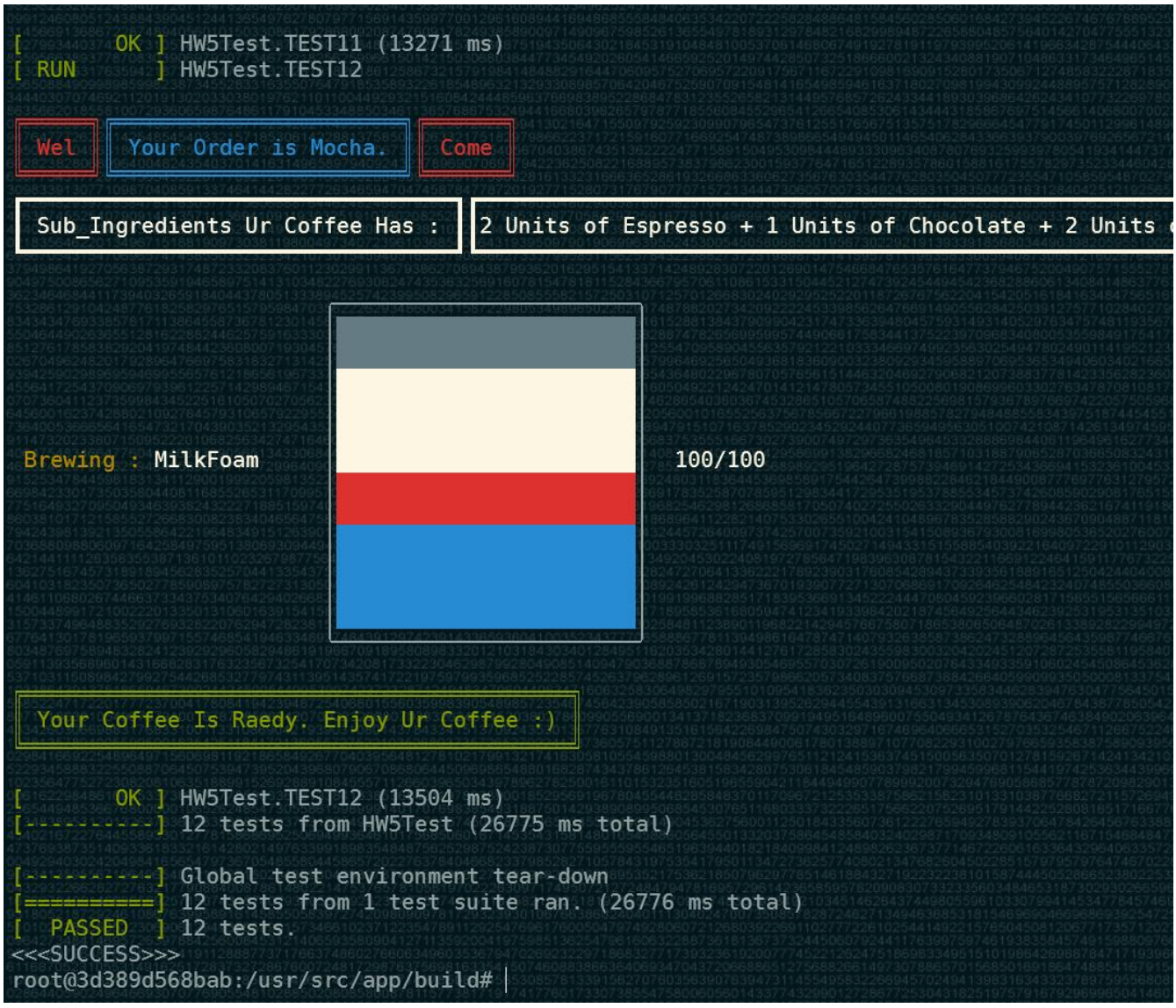
- جواب سوال پرسیده شده در صورت سوال:
  اگر Destructor این کلاس را به صورت Protected تعریف می‌کردیم ، دیگر خارج از این کلاس یا کلاس‌های مشتق گرفته از این کلاس ، نمی‌توانستیم به Destructor این کلاس دسترسی داشته باشیم. (یعنی مثلا نمیتوانستیم به صورت دستی برای پاک گردن یک Object از این کلاس از دستور delete استفاده کنیم !)

در ادامه سراغ نوشتن کلاس‌های Cappuccino و Mocha می‌رویم. این دو کلاس کاملا شبیه به هم هستند و فقط دز جزئیات با هم تفاوت دارند و هر دو کلاس مشتق‌گرفته از کلاس EspressoBassed هستند.

در این دو کلاس توابع get_name() و price() را به صورت override می‌نویسیم. چراکه این دو تابع در کلاس Base آنها به صورت Pure Virtual تعریف شده بودند.

تنها نکتهٔ قابل توجه دیگر در پیاده‌سازی این کلاس‌ها این است که Copy Constructor و = Operato آنها را باید به صورت درستی که در کلاس EspressoBassed توضیح داده شد، بنویسیم.

و در نهایت تمام تست‌ها با موفقیت Pass شدند :



لینک github