Oh My Sandwich

Dépôt GitHub : https://github.com/Nouuu/OhMySandwich

Énoncé

1. Informations pratiques

Le code rendu devra être compilé sans erreurs.

Il vaut mieux rendre un code incomplet qui compile qu'un code ne compile pas.

Le projet sera noté selon plusieurs critères :

- Qualité du code fourni
- Qualité des présentations intermédiaires
- Prise en compte des différents retours suite aux présentations
- Qualité de la soutenance finale

Vous n'oublierez pas d'inclure les slides de votre soutenance finale ainsi qu'un rapport PDF précisant vos choix, les problèmes techniques rencontrés et les solutions trouvées.

2. Sujet

Une sandwicherie souhaite simplifier sa prise de commande et l'élaboration de ses factures.

Chaque sandwich est constitué d'une liste précisé d'ingrédients et possède un prix.

On souhaite écrire un programme qui prend en entrée une commande de sandwichs et produit une facture formatée.

Prise en compte des commandes

Votre programme devra récupérer les commandes sous la forme d'entrée textuelle en console.

Les commandes sont de la forme :

- A Sandwich1

- A Sandwich1, B Sandwich2, C Sandwich3

Une commande peut contenir plusieurs occurrences du même sandwich, ainsi une commande de la forme :

- A Sandwich1, B Sandwich2, C Sandwich1 devra être considérée comme :
- A+C Sandwich1, B Sandwich2

Édition d'une facture

Après avoir interprété la commande en entrée, vous produirez une sortie console suivant la forme suivante :

```
A Sandwich1
    Ingredient1
    Ingredient2
    [...]
    IngredientN
B Sandwich2
    Ingredient1
    [...]
[...]
Prix total : XXX€
```

Sandwichs disponibles

La sandwicherie est capable de produire les sandwichs suivants :

- **Jambon beurre** : 1 pain, 1 tranche de jambon, 10g de beurre => 3,50€
- **Poulet crudités** : 1 pain, 1 oeuf, 0.5 tomate, 1 tranche de poulet, 10g de mayonnaise, 10g de salade => 5€
- **Dieppois** : 1 pain, 50g de thon, 0.5 tomate, 10g de mayonnaise, 10g de salade => 4,50€

Comportement attendu du programme

Votre programme devra récupérer l'entrée de l'utilisateur et valider sa conformité.

En cas de commande incorrecte, votre programme produira une erreur compréhensible, mais ne devra pas crasher.

En cas de commande correcte, votre programme écrira dans la console la facture.

Après avoir traité une commande, votre programme attendra la commande suivante, il ne doit pas s'arrêter après avoir écrit une facture.

3. Déroulement du projet

Il vous sera demandé une première implémentation naïve ne vous demandant pas d'utiliser de design patterns.

Cette première implémentation sera présentée et servira de base aux discussions de pistes d'améliorations de votre projet.

Vous devrez ensuite revoir votre implémentation afin d'y implémenter des designs pattern appropriés.

Cette deuxième implémentation vous servira de base pour les modules complémentaires du projet.

Une fois la première implémentation présentée, les différents modules complémentaires vous seront présentées.

Votre rendu final devra contenir le sujet de base ainsi qu'au moins un module complémentaire.

Vous devrez présenter une première fois votre implémentation ainsi que vos choix.

Choix d'implémentation

Solution & Projets

Afin d'avoir une architecture propre, nous avons créé une solution contenant plusieurs projets. Ils sont au nombre de 4 :

- **CLI** : Implémentation d'un Adaptateur de Commande en ligne de commande pour l'application.
- **Domain** : Contient la partie métier, nos objets du domaine ainsi que nos interfaces
- **Infrastructure** : Contient les objets techniques nécessaires à l'implémentation de l'application
- **Test** : Contient les tests unitaires de l'application

Tests

Nous avons créé un projet de tests unitaires afin de pouvoir nous assurer que notre code fonctionne correctement.

Cela nous a aussi été utile au moment du refactoring de notre code pour ne pas faire de regression.

Design Patterns

Command

```
public interface ICommand
{
    ICommand? Execute();

    string GetCommandHelp();

    void Display();
}
```

CLI

- InvoiceGeneratorCommand
- MenuCommand
- NewOrderCommand
- SandwichSelectorComman

Adapter

```
public interface IAdapter
{
    void AcceptInteractions();
}
```

```
public interface IMarshaller<in T>
{
    public string Serialize(T data);
}
```

CLI

- CliAdapter

Infrastructure

- ConsoleBasketMarshaller

- ConsoleIngredientMarshaller
- ConsoleInvoiceMarshaller
- ConsolePriceMarshaller
- ConsoleSandwichMarshaller

Facade + Singleton

```
public interface Context
{
    InvoiceGenerator GetInvoiceGenerator();
    IMarshaller<IngredientStack> GetIngredientMarshaller();
    IMarshaller<Invoice> GetInvoiceMarshaller();
    IMarshaller<Price> GetPriceMarshaller();
    IMarshaller<Sandwich> GetSandwichMarshaller();
    IMarshaller<Basket> GetBasketMarshaller();
    Basket GetBasket();
    List<ICommand> GetAvailableCommands();
    List<Sandwich> GetAvailableSandwichs();
    IAdapter GetAdapter();
}
```

Infrastructure

- CliContext

Factory

```
public interface InvoiceGenerator
{
    Invoice GenerateInvoice(Basket basket);
}
```

Infrastructure

- DefaultInvoiceGenerator

Iterator

```
public interface Iterator<out T>
{
    T NextIteration();
}
```

Infrastructure

- AlphabetIterator

Builder

> SandwichBuilder

```
public class SandwichBuilder
{
    private readonly string? _name;
    private readonly Price? _price;
    private readonly ISet<IngredientStack> _ingredients;

    public SandwichBuilder()
    {
        _ingredients = new HashSet<IngredientStack>();
    }

    public SandwichBuilder(string? name, Price? price, ISet<IngredientStack> ingredientStacks)
    {
        this._name = name;
        this._price = price;
        _ingredients = ingredientStacks;
    }

    public Sandwich GetSandwich()
    {
        if (_name == null || _price == null)
        {
            throw new InvalidOperationException();
        }

        return new Sandwich(_name, _ingredients.ToArray(), _price.Value);
    }

    public SandwichBuilder SetName(string newName)
    {
        return new SandwichBuilder(newName, _price, _ingredients);
```

```
    }

    public SandwichBuilder AddIngredient(IngredientStack ingredientStack)
    {
        var newIngredients = new HashSet<IngredientStack>(_ingredients) { ingredientStack };
        return new SandwichBuilder(_name, _price, newIngredients);
    }

    public SandwichBuilder AddIngredient(Ingredient ingredient, double count)
    {
        var newIngredients = new HashSet<IngredientStack>(_ingredients) { new(ingredient,
count) };
        return new SandwichBuilder(_name, _price, newIngredients);
    }

    public SandwichBuilder SetPrice(Price price)
    {
        return new SandwichBuilder(_name, price, _ingredients);
    }

    public SandwichBuilder SetPrice(double price)
    {
        return new SandwichBuilder(_name, new Price("€", price), _ingredients);
    }

    public SandwichBuilder FromSandwich(Sandwich sandwich)
    {
        return new SandwichBuilder(
            sandwich.Name,
            sandwich.Price,
            new HashSet<IngredientStack>(sandwich.Ingredients)
        );
    }
}
```

Value Object

Domain

- Basket
- Ingredient
- IngredientStack

- Invoice
- Price
- Sandwich
- UnitType

Revision #2
Created 2022-10-02 14:45:43 UTC by Noé Larrieu-Lacoste
Updated 2022-10-02 14:53:24 UTC by Noé Larrieu-Lacoste