Le Patron de Conception

Façade

# En bref

## Type de patron de conception

Structurel & Pattern d’objets

## Définition

Le patron de conception façade fournit une interface unifiée à l’ensemble des interfaces d’un sous-système. La façade fournit une interface de plus haut niveau rendant le sous-système plus facile à utiliser.

## Schéma de principe

Façade

Refactoring

# Introduction

## Présentation

Le patron de conception façade procure une interface simplifiée à une librairie, un Framework ou un ensemble complexe de classes. C’est un patron de conception structurel car il permet de composer des objets pour former des structures plus vastes. Il est aussi décrit comme Pattern d’objets car il décrit des relations entre objets qui sont définies à l’exécution du programme.

## Problématique

Face à des librairies ou Framework complexes, le client qui n’utilise pas de façade doit lui-même gérer les instanciations de classes et les appels de méthodes. Lorsque le système est incrémenté par de nouvelles fonctionnalités ou est modifié, le client doit réécrire son code ce qui provoque une maintenabilité difficile.

## Solution

Lorsque le fournisseur de logiciels implémente une façade, il offre à l’utilisateur une interface simplifiée d’un sous-système complexe. Les fonctionnalités sont limitées car la façade est conçue pour le client.

# Étude

## Simplification

La première motivation d’une façade est de simplifier une interface.

## Découplage

La seconde motivation d’une façade est de découpler des sous-systèmes. Ainsi lors de l’incrémentation du système, la façade est réimplémentée par le fournisseur et non le code du client.

## La façade n’est pas une encapsulation

Le client peut toujours, s’il a un besoin spécifique, accéder au sous-système directement, avec toutes les problématiques liées à la non-utilisation d’une façade qui en découlent. Ce n’est donc pas une encapsulation car les classes du système sont exposées.

## La façade peut être une composition de façade

Pour simplifier la gestion d’une façade lorsqu’elle fait appel à de nombreux sous-systèmes, il est possible de créer des façades *ad-hoc* pour découpler les sous-systèmes.

## Le principe « ne parlez pas aux inconnus »

Il est important de limiter au maximum les dépendances entre les classes pour éviter les effets de cascade. Voici une règle de bonne pratique à suivre :

* Soit un objet du système. Cet objet ne peut appeler que des méthodes qui appartiennent :
  + A l’objet lui-même
  + Aux objets donnés en arguments de la méthode appelée
  + Aux objets instanciés par la méthode appelée
  + Aux composants de l’objets (un composant est un objet référencé par une variable).

En suivant cette règle, le principe « ne parlez pas aux inconnus » est respecté.

L’inconvénient de ce principe est qu’il pousse à écrire plus de classes enveloppes ce qui augmente la complexité, le temps de développement et diminue les performances à l’exécution.

# Exemple d’utilisation du patron de conception Façade

## Diagramme de classes

Dans cette section, il est présenté un cas où un client interagit avec un système avec et sans façade.

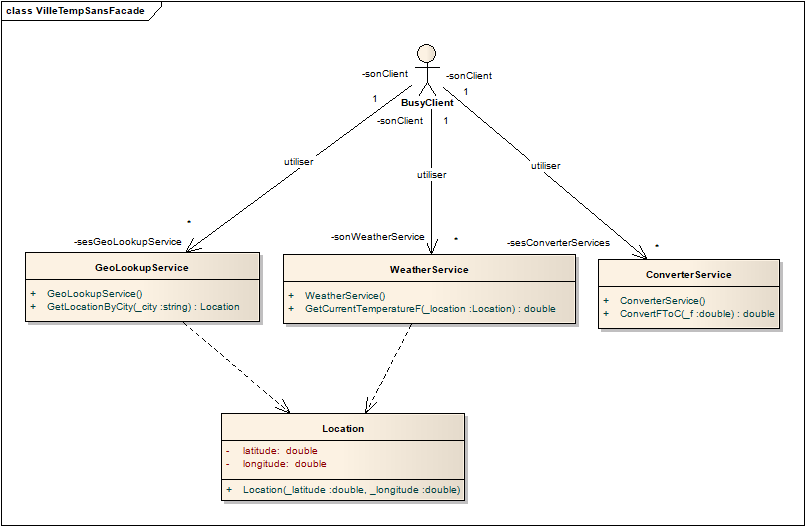
Contexte

Soit un client souhaitant obtenir la température d’une ville à partir de son nom. La température doit être en degré Celsius. L’éditeur du programme offre une bibliothèque de classes métiers au client et lui propose d’implémenter en supplément une façade pour lui simplifier l’accès à cette bibliothèque.

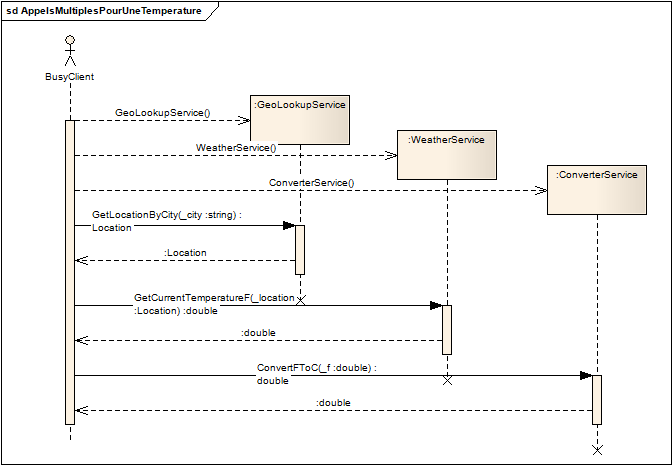
La bibliothèque de classes est constituée de :

* La classe Localisation, qui a pour attribut un couple de double correspondant à la longitude et la latitude d’un lieu.
* La classe GeoLookupStation, qui a une opération permettant de renvoyer une localisation à partir d’un nom de ville.
* La classe WeatherService, qui a une opération permettant de renvoyer une température en degré Fahrenheit à partir d’une localisation.
* La classe ConverterService, qui a une opération permettant de convertir une température d’unité égale au degré Fahrenheit en une température en degré Celsius.

Sans façade

Le client ne souhaite pas utiliser de façade. Voici le diagramme de classe correspondant : 

Dans ce cas, le client gère lui-même son accès à la bibliothèque de classes avec toute la complexité que cela implique. Si la bibliothèque de classe venait à être modifiée, le client aurait l’obligation de réimplémenter son programme.

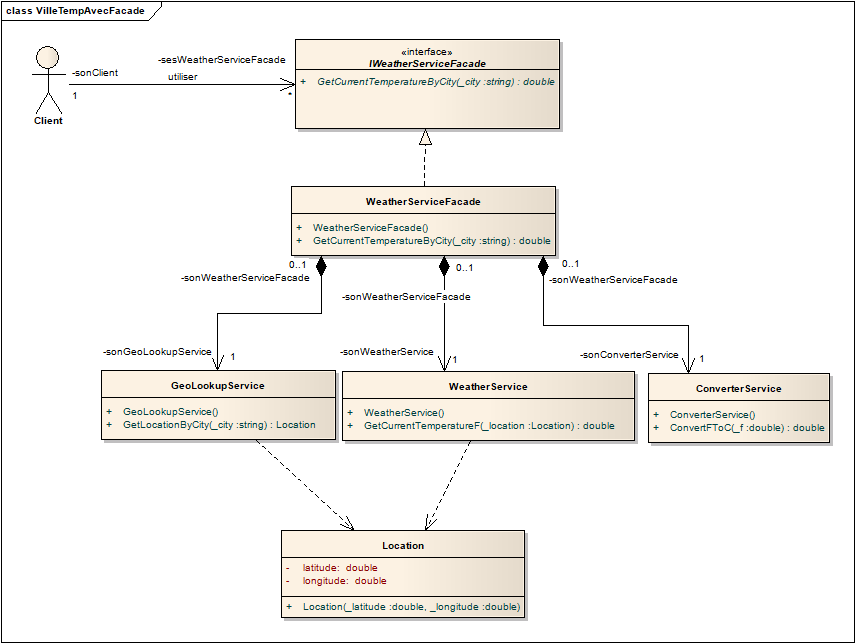
Voici le diagramme de séquence correspondant : 

Dans un premier temps, le client instancie les trois services nécessaires à l’accomplissement de sa tâche. Une fois les objets créés, le client utilise le comportement GetLocationByCity de l’objet :GeoLookupService. Celui-ci renvoie une :Localisation qui est utilisée par le comportement GetCurrentTemperatureF de l’objet :WeatherService, renvoyant une température en degré Fahrenheit. Cette température est convertie par le comportement ConvertFToC de l’objet :ConverterService qui renvoie à son tour une température dont l’unité est le degré Celcius.

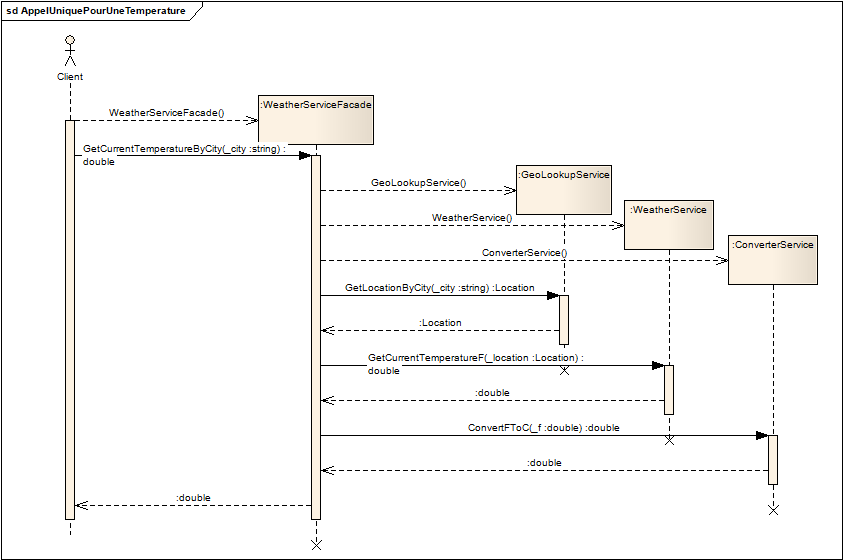
En observant le diagramme de séquence, il est possible de constater que le client doit gérer lui-même l’instanciation des objets et l’ordre dans lequel il appelle les opérations. Le client doit ainsi comprendre la complexité du système.

Il est compréhensible que la conception d’une façade soit nécessaire.

Avec façade

Le client décide de faire appel à l’éditeur du programme pour qu’il implémente une façade. Voici le diagramme de classe correspondant : 

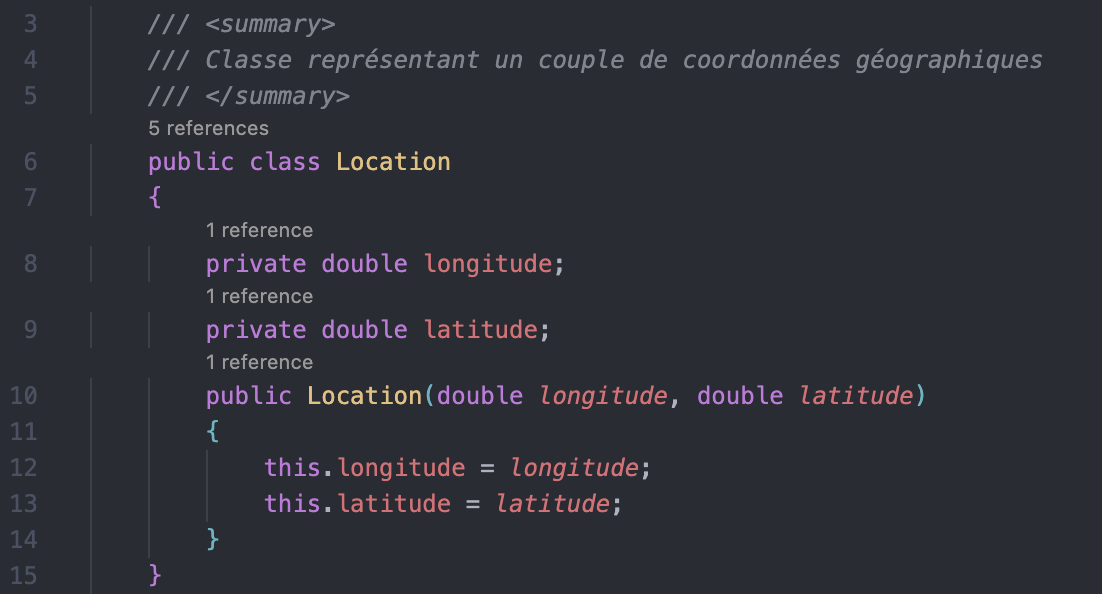
L’incrémentation par rapport au diagramme de classe précédent est la présence d’une interface IWeatherServiceFacade qui est réalisée par la classe WeatherServiceFacade. Cette classe présente une opération GetCurrentTemperatureByCity : à partir d’un nom de ville, elle renvoie sa température en degré Celcius. Le client, une fois l’instanciation de ce service effectuée, n’a qu’à appeler la précédente opération pour obtenir le résultat attendu.

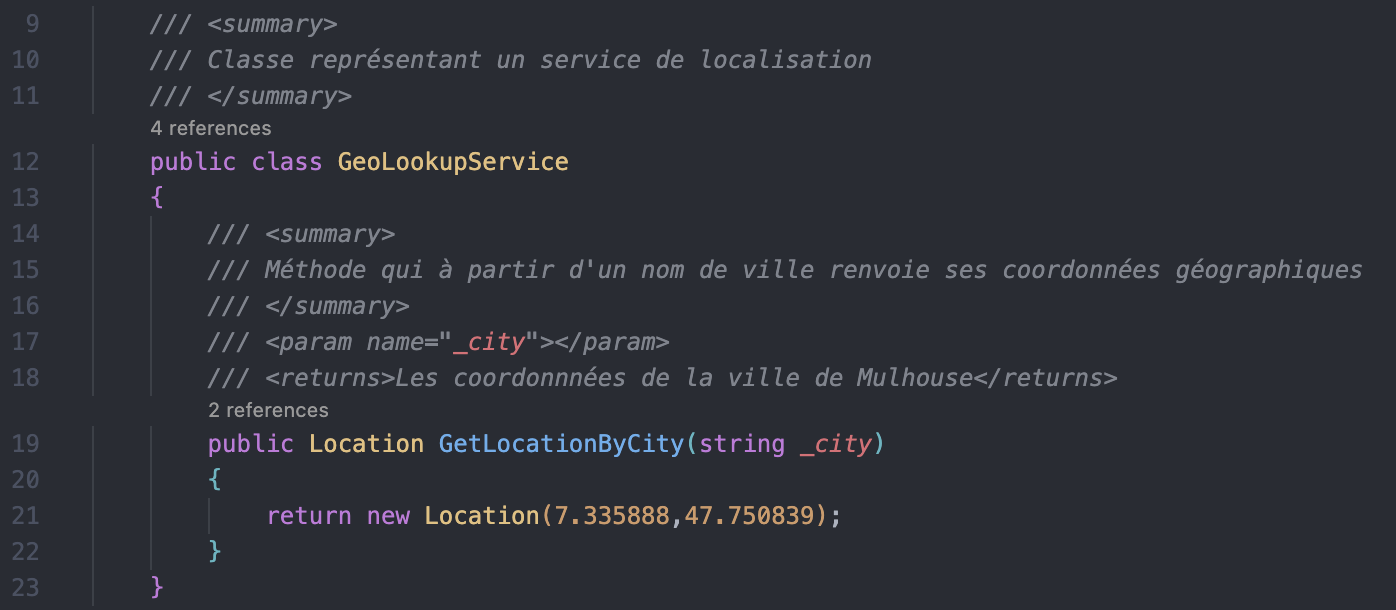
Ainsi, client n’a besoin que d’interagir avec la façade. Le diagramme de séquence correspondant permet de comprendre le peu d’implémentation que le client doit réaliser : 

## Implémentation en C#

Dans un premier temps, il est implémenté les classes métiers présentées dans les diagrammes de classes de la section précédente.

Classes métiers

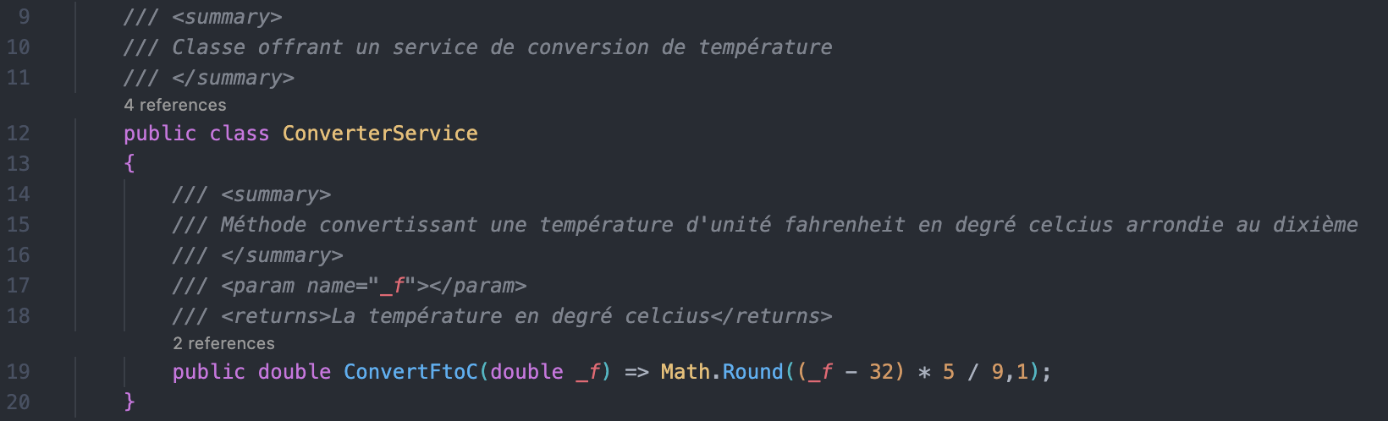
Location : 

GeoLookupService : 

Dans un souci de simplification, il est admis que la localisation est fixée.

WeatherService : 

Dans un souci de simplification, il est admis que la température est aléatoire.

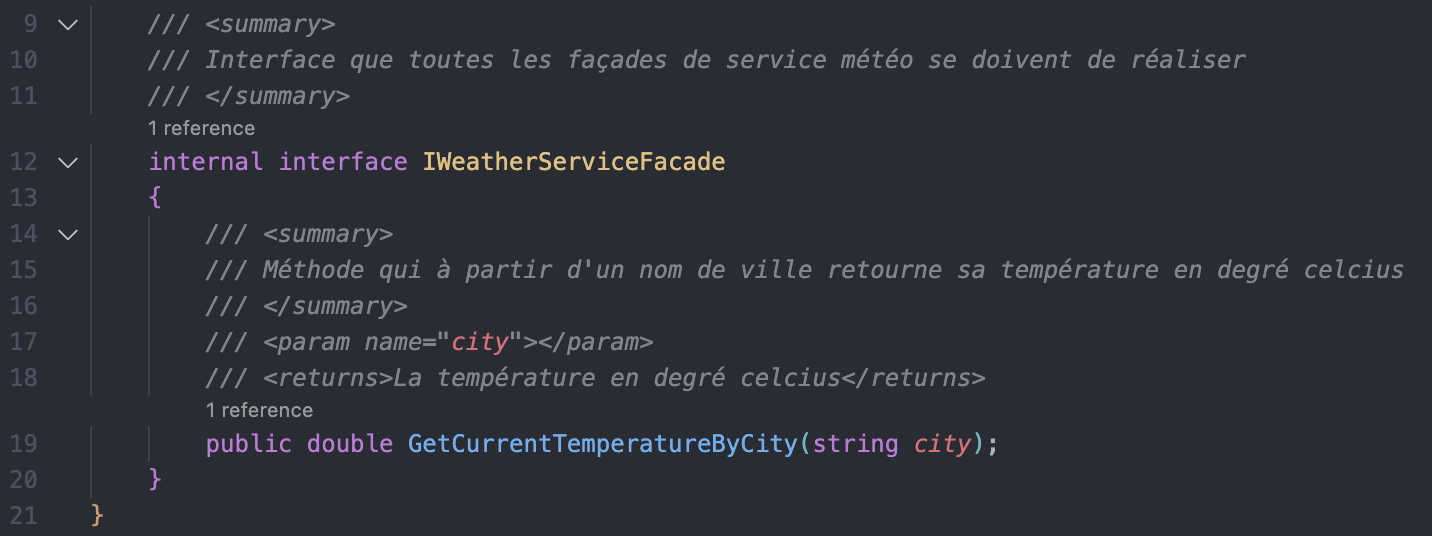
ConverterService : 

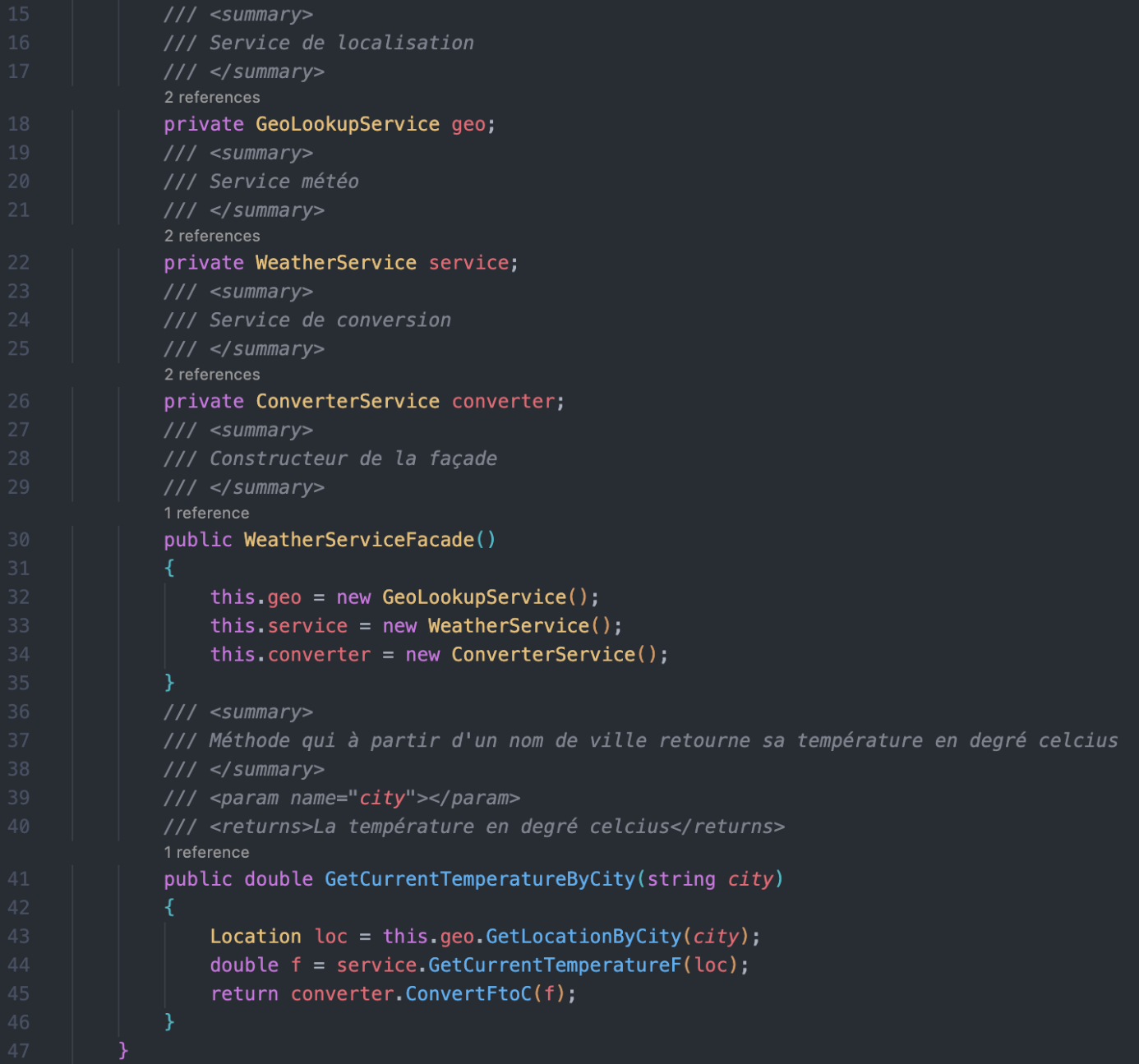
Dans un souci de réalisme, la température est arrondie à l’unité.

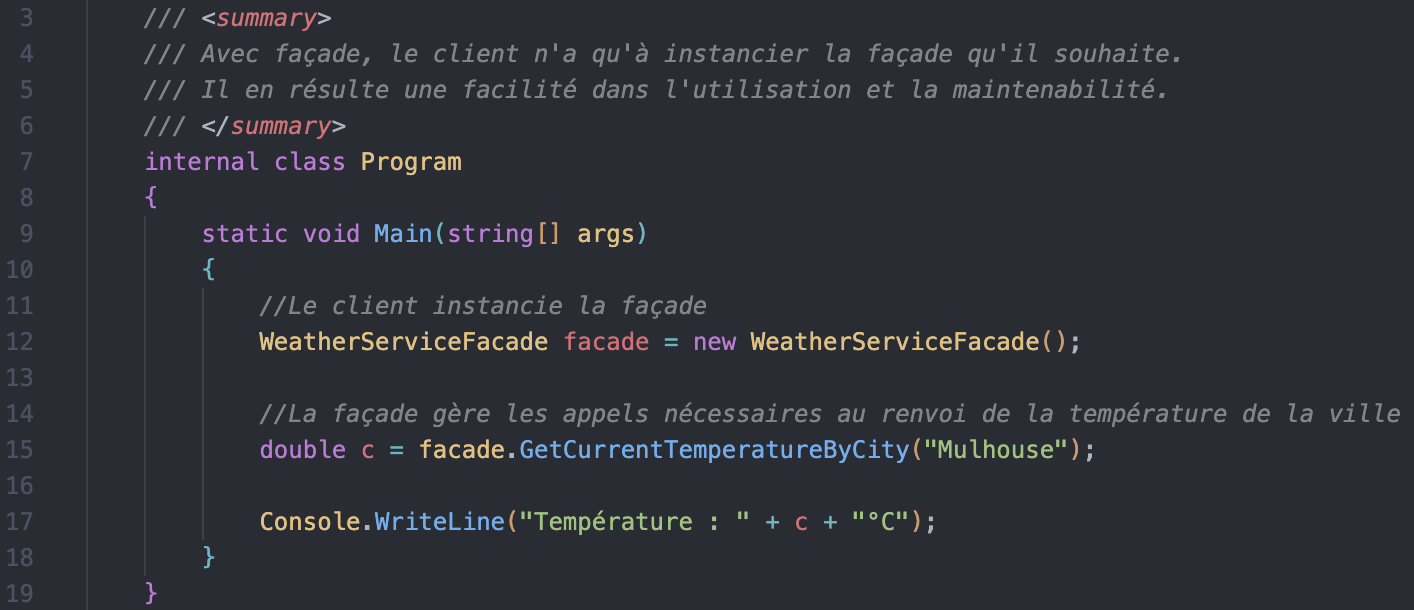
Sans façade

Program : 

Avec façade

IWeatherServiceFacade : 

WeatherServiceFacade : 

Program : 

## Conclusion

Le patron de conception façade permet une meilleure communication avec le client car en masquant la complexité du système et en offrant des méthodes simples, son besoin est répondu.