C++ – Travaux Pratiques

Compte Rendu de TP 4 : Gérer un PLU

Liam SMALL, Fatoumata SEYE, Milo SOULARD, Nirmine KORTAM

1er décembre 2024

Table des matières

1 Introduction

Dans ce TP, nous allons explorer la gestion d'un Plan Local d'Urbanisme (PLU) à l'aide du langage C++.

2 Objectifs du TP

Les principaux objectifs de ce TP étaient les suivants :

— Concevoir des classes en C++ pour modéliser des parcelles de terrain sous différentes catégories.

— Appliquer les concepts de l'héritage en programmation orientée objet (simple et multiple)

— Manipuler des objets géométriques comme des polygones et calculer des surfaces en utilisant des méthodes adaptées.

— Utiliser des exceptions pour gérer des erreurs, comme des surfaces invalides ou des polygones mal formés.

— Implémenter la surcharge d'opérateurs pour faciliter l'affichage des objets.

— Travailler avec des fichiers pour sauvegarder et charger les données liées aux parcelles et à la carte.

— Comprendre la gestion de mémoire avec des pointeurs et des objets dynamiques, en particulier avec des collections d'objets comme les vecteurs.

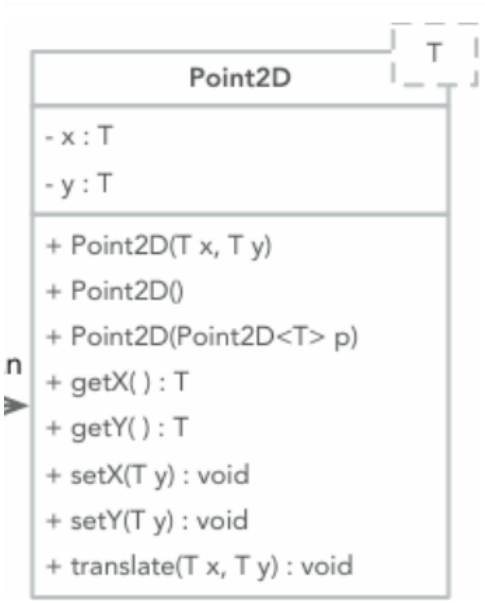
— Apprendre à travailler en groupe sur un github commun.

— S'exercer à rédiger une documentation, notemment avec doxygen.

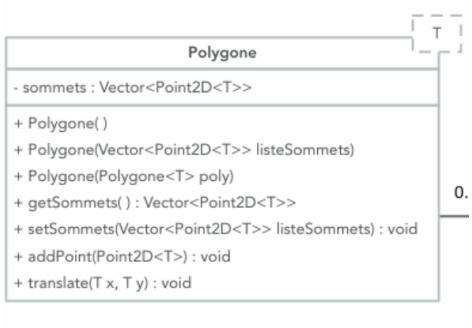
3 Diagrammes de classes

Les premiers étaient donnés dans l'énoncé du TP.

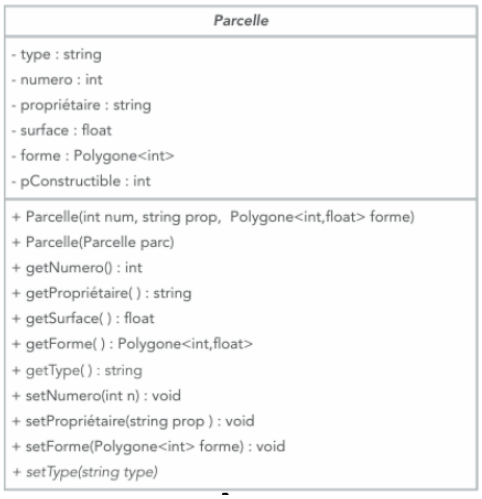
3.1 Classe Point2D



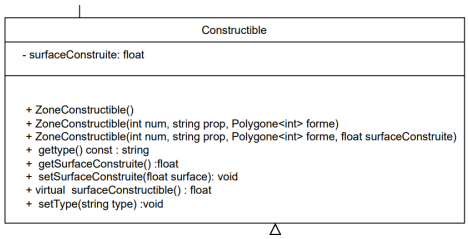
3.2 Classe Polygone



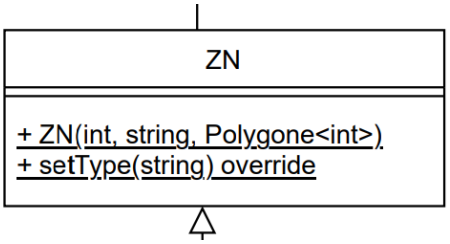
3.3 Classe Parcelle



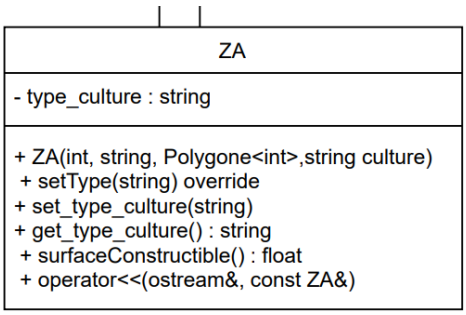
3.4 Classe Constructible



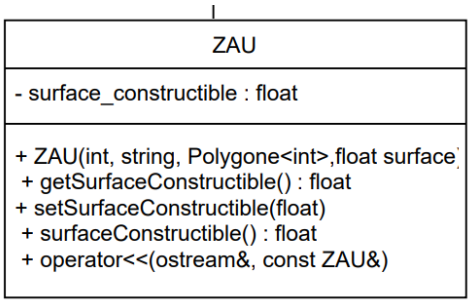
3.5 Classe ZoneNaturelle



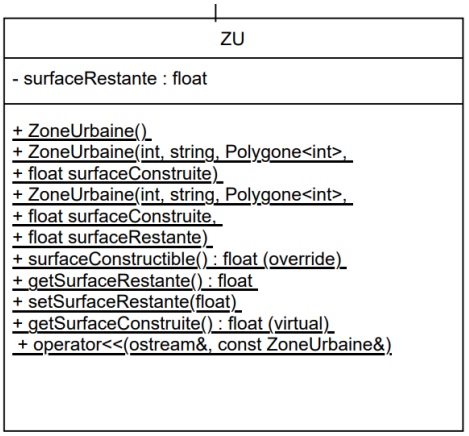
3.6 Classe ZoneAgricole



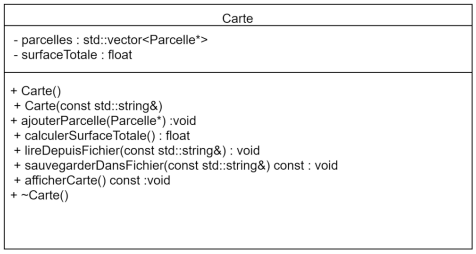
3.7 Classe ZoneAUrbaniser



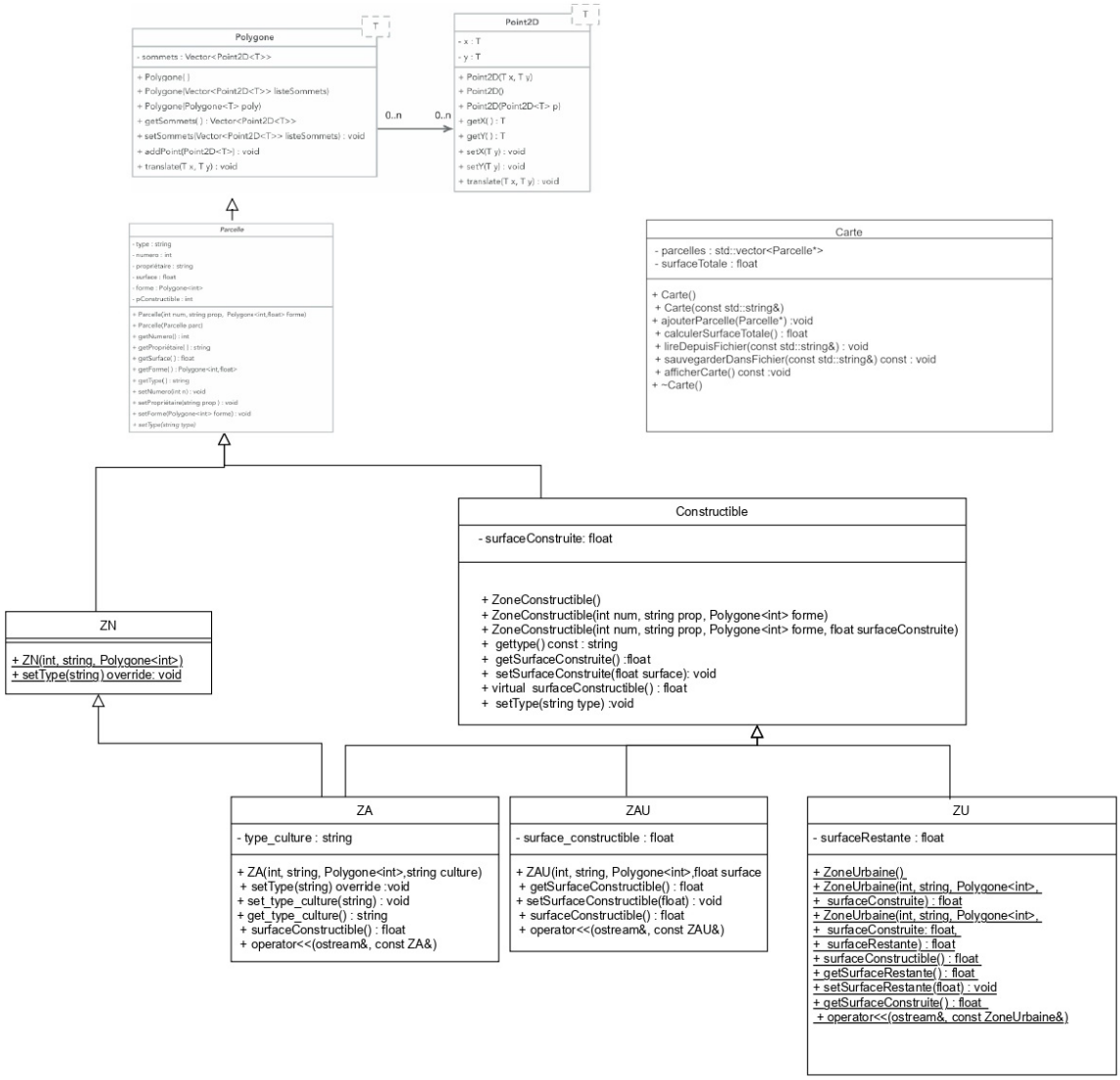
3.8 Classe ZoneUrbaine



3.9 Classe Carte



3.10 Diagramme Entier



4 Explication des codes

4.1 Classe Point2D

Le code est intégralement écrit dans le fichier d'entête .h pour éviter les problèmes avec les template. On s'en sert pour créer des points avec des coordonnées X et Y de types différents. On a accès à des setters et des getters, ainsi qu'une translation et une surcharge d'opérateur pour les afficher en utilisant std : :cout.

4.2 Classe Polygone

La classe Polygone permet de créer et de manipuler des polygones définis par des sommets de type Point2D<T>. Elle inclut un constructeur par défaut pour créer un polygone vide, ainsi que des constructeurs permettant de créer un polygone à partir d'une liste de sommets ou par copie. La méthode getSommets permet de récupérer les sommets du polygone, et setSommets permet de les modifier. La fonction addPoint ajoute un sommet à la liste. La méthode translate(dx, dy)

déplace tous les sommets du polygone en appliquant une translation. La méthode estSensTrigo vérifie si les sommets sont dans le sens trigonométrique, et CheckSensTrigo lève une exception si ce n'est pas le cas. La méthode segmentsIntersect vérifie si deux segments de droite s'intersectent, et estCroise détecte si le polygone présente des croisements. Enfin, la surcharge de l'opérateur « permet d'afficher les sommets du polygone.

4.3 Classe Parcelle

La classe Parcelle permet de modéliser une parcelle de terrain avec des informations sur son type, son numéro, son propriétaire, sa surface, et sa forme géométrique, représentée par un polygone. Le constructeur par défaut initialise une parcelle, tandis que l'autre constructeur permet de créer une parcelle avec un numéro, un propriétaire et une forme spécifique. La méthode getNumero, getProprietaire, getSurface, getForme, et getType permettent de récupérer respectivement le numéro, le propriétaire, la surface, la forme et le type de la parcelle. Les méthodes setNumero, setProprietaire, et setForme permettent de modifier ces informations. La méthode virtuelle setType doit être implémentée dans les classes dérivées. La méthode calculerSurface permet de calculer la surface de la parcelle, bien que son implémentation ne soit pas détaillée dans cette classe. En cas d'erreur dans le calcul de la surface, une exception SurfaceException est levée. La surcharge de l'opérateur « permet d'afficher les informations de la parcelle. Enfin, un destructeur virtuel est présent pour assurer une suppression correcte des objets dérivés de Parcelle.

4.4 Classe ZoneConstructible

La classe ZoneConstructible hérite de la classe Parcelle et représente une parcelle qui peut être construite. Elle ajoute un attribut surfaceConstruite, qui permet de stocker la surface déjà construite sur la parcelle. Les constructeurs permettent de créer une zone constructible avec ou sans la surface construite spécifiée. La méthode gettype permet de récupérer le type de la zone, tandis que getSurfaceConstruite et setSurfaceConstruite permettent respectivement de récupérer et de modifier la surface construite. La méthode virtuelle pure surfaceConstructible doit être implémentée dans les classes dérivées pour calculer la surface constructible restante. Enfin, la méthode setType permet de définir le type de la zone.

4.5 Classe ZoneNaturelle

La classe ZN hérite virtuellement de la classe Parcelle et représente une parcelle désignée comme zone naturelle. Le constructeur permet de créer une zone naturelle en spécifiant son numéro, son propriétaire, et sa forme géométrique. La méthode setType permet de définir le type de la parcelle à "ZN", ce qui est fait automatiquement lors de la création de l'objet. Cette classe hérite de la fonctionnalité de la classe Parcelle et définit le type spécifique à une zone naturelle, sans ajouter d'attributs ou de méthodes supplémentaires.

4.6 Classe ZoneAgricole

La classe ZA hérite virtuellement de ZN et ZoneConstructible et représente une parcelle de terrain dédiée à l'agriculture, avec des caractéristiques spécifiques comme le type de culture. Le constructeur permet de créer une zone agricole en spécifiant son numéro, son propriétaire, sa forme géométrique et le type de culture. La méthode setType redéfinit le type de la parcelle à "ZA". La méthode set type culture permet de définir le type de culture, et get type culture permet de le récupérer. La méthode surfaceConstructible calcule la surface constructible restante, comme défini dans ZoneConstructible. Enfin, la surcharge de l'opérateur « permet d'afficher les informations de la zone agricole, incluant le numéro, le type, la forme, le propriétaire, le type de culture et la surface.

4.7 Classe ZoneAUrbaniser

La classe ZAU hérite de ZoneConstructible et représente une parcelle destinée à être urbanisée. Le constructeur permet de créer une zone à urbaniser en spécifiant son numéro, son propriétaire, sa forme géométrique et la surface constructible, tout en définissant le type de la zone à "ZAU". La méthode getSurfaceConstructible permet de récupérer la surface constructible, et setSurfaceConstructible permet de la modifier, avec une vérification que la surface soit bien comprise entre 0 et 1. La méthode surfaceConstructible calcule la surface constructible en fonction du pourcentage spécifié, en multipliant la surface totale par le facteur de surface constructible. La classe gère ainsi les spécificités d'une zone à urbaniser, tout en héritant des fonctionnalités de ZoneConstructible.

4.8 Classe ZoneUrbaine

La classe ZoneUrbaine hérite de ZoneConstructible et représente une zone qui est partiellement construite, avec une surface restante encore constructible. Le constructeur permet de créer une zone urbaine en spécifiant son numéro, son propriétaire, sa forme géométrique, la surface déjà construite, et la surface restante. La méthode surfaceConstructible redéfinit celle de ZoneConstructible pour retourner la surface constructible restante. Les méthodes getSurfaceRestante et setSurfaceRestante permettent respectivement de récupérer et de modifier la surface encore disponible pour la construction. La méthode getSurfaceConstruite permet d'accéder à la surface déjà construite. Enfin, la surcharge de l'opérateur « permet d'afficher les informations de la zone urbaine, incluant le numéro, le type, la forme, le propriétaire, la surface, la surface construite et la surface constructible restante.

4.9 Classe Carte

La classe Carte gère une collection de parcelles et calcule la surface totale d'une carte. Elle contient un vecteur de pointeurs vers des objets de type Parcelle, représentant les différentes zones de la carte. Le constructeur par défaut initialise la carte vide, tandis que l'autre constructeur permet de charger la carte à partir d'un fichier. La méthode ajouterParcelle permet d'ajouter une parcelle à la carte. La méthode calculerSurfaceTotale retourne la surface totale de toutes les parcelles de la carte en les additionnant. Les méthodes lireDepuisFichier et sauvegarderDansFichier servent respectivement à charger et sauvegarder les informations de la carte depuis et vers un fichier. La méthode afficherCarte permet d'afficher les détails de toutes les parcelles présentes sur la carte. Enfin, le destructeur libère la mémoire allouée pour les parcelles.

5 Jeux d'essais

5.1 Jeux d'essai Polygone

Le tableau suivant présente les tests effectués sur la classe `Polygone`, comprenant la création, la copie et la translation d'un polygone.

Input (instructions)	Résultat attendu (sommets du polygone)	Validité
Création du polygone original	(0, 0) (4, 0) (4, 3) (0, 3)	Oui
Création du polygone copié (avant translation)	(0, 0) (4, 0) (4, 3) (0, 3)	Oui
Translation du polygone original (+2, +3)	(2, 3) (6, 3) (6, 6) (2, 6)	Oui
Polygone copié après translation de l'original	(0, 0) (4, 0) (4, 3) (0, 3)	Oui

5.2 Jeux d'essai ZAU

Le tableau suivant présente les tests effectués sur la classe `ZAU`, comprenant la création, le test de surface négative, et l'affichage des informations.

Input (instructions)	Résultat attendu	Validité
Création d'une parcelle ZAU avec 2000 m^2 et un taux constructible de 50%	Surface constructible : 1000 m^2	Oui
Test de surface négative pour ZAU	Message d'erreur attendu : "Surface constructible doit être entre 0 et 1."	Oui
Affichage des informations de ZAU	Type : ZAU, Surface constructible calculée en fonction du taux	Oui

5.3 Jeux d'essai ZA

Le tableau suivant présente les tests effectués sur la classe `ZA`, incluant la création, le test des limites, et l'affichage des informations.

Input (instructions)	Résultat attendu	Validité
Création d'une parcelle ZA avec 3000 m^2, culture : "Blé"	Surface constructible : jusqu'à 200 m^2 et max 10% (soit 300 m^2), avec respect des conditions	Oui
Test de dépassement des limites pour ZA (surface construite $>$ 200 m^2)	Surface constructible : 0 m^2 (Non valide, dépassement des limites)	Non
Affichage des détails d'une ZA avec culture "Maïs"	Type : ZA, Culture : Maïs, Surface constructible (sous conditions)	Oui
Création de plusieurs ZA suivies de calculs	Les zones et surfaces sont conformes à la réglementation	Oui

5.4 Jeux d'essai ZN

Le tableau suivant présente les tests effectués sur la classe `ZN`, incluant la création, le test de croisement de segments, et l'affichage des détails.

Input (instructions)	Résultat attendu	Validité
Création d'une parcelle ZN avec 5000 m²	Surface constructible : 0 m² (Non constructible)	Oui
Test croisement de segments dans une ZN	Message attendu : "Le polygone est croisé."	Oui
Affichage des détails d'une ZN	Type : ZN, Propriétaire : X, Surface : 5000 m²	Oui

5.5 Jeux d'essai ZU

Le tableau suivant présente les tests effectués sur la classe `ZU`, comprenant la création, l'affichage des détails, et la translation.

Input (instructions)	Résultat attendu	Validité
Création d'une parcelle ZU avec 1000 m², 300 m² construits	Surface restante constructible : 700 m²	Oui
Affichage des détails d'une ZU	Type : ZU, Propriétaire : X, Surface construite : 300 m², Surface restante : 700 m²	Oui
Translation d'une parcelle ZU de (2,3)	Sommets déplacés correctement	Oui

5.6 Jeux d'essai Carte

Le tableau suivant présente les tests effectués sur la classe `Carte`, incluant le chargement des parcelles, l'ajout, la sauvegarde et l'affichage.

Type de Parcelle	Input	Résultat Attendu
ZU	ZU 101 Dupont 0.75 200 (5,5) (10,5) (10,10) (5,10)	Surface construite : 200 m², Surface restante constructible : dépend de la surface totale du polygone
ZAU	ZAU 102 Martin 0.50 (3,3) (7,3) (7,7) (3,7)	Surface constructible : 50% de la surface totale du polygone
ZA	ZA 103 Bernard AgricultureBio (1,1) (4,1) (4,4) (1,4)	Type de culture : AgricultureBio, Surface constructible limitée à 10% ou 200 m² max
ZN	ZN 104 Durand (0,0) (5,0) (5,5) (0,5)	Surface non constructible
Fichier Input	Fichier : "parcelles$_{t}est.txt$" $aveccontenu$: $ZU101Dupont0.75200(5,5)(10,5)(10,10)(5,10), ZAU102Martin0.50(3,3)(7,3)(7,7)(3,7), ZA103BernardAgricultureBio(1,1)(4,1)(4,4)(1,4)$	Chargement des parcelles, affichage et calcul corrects des surfaces
Ajout parcelle ZU	Ajouter une parcelle ZU : ajouterParcelle(new ZU(101, "Dupont", polygone, 0.75, 200))	Ajout réussi, la surface totale doit être mise à jour
Ajout parcelle ZA	Ajouter une parcelle ZA : ajouterParcelle(new ZA(103, "Bernard", polygone, "Bio"))	Ajout réussi avec type de culture spécifié
Sauvegarde fichier	Sauvegarder dans le fichier "carte$_{s}auvegarde.txt$"	Le fichier doit contenir les informations de chaque parcelle ajoutée
Affichage carte	afficherCarte()	Affiche toutes les parcelles avec leurs détails
Surface totale	Calculer la surface totale avec plusieurs parcelles ajoutées	Affiche la somme des surfaces des parcelles

6 Conclusion

En conclusion, ce projet nous a permis de combiner théorie et pratique pour aborder la gestion d'un Plan Local d'Urbanisme (PLU) tout en renforçant nos compétences en programmation orientée objet avec C++. Nous avons appliqué des concepts clés comme l'héritage, la surcharge d'opérateurs et la gestion des exceptions, en structurant le code de manière claire.

Le travail collaboratif via GitHub nous a aussi appris à mieux organiser le code en équipe, à gérer les versions et à intégrer les contributions de chacun efficacement.