Haute Ecole de la Province de Liège

Programmation .NET et C#

Enoncé de laboratoire Phase I

2019-2020

François Caprasse
Cécile Moitroux
Daniel Schreurs

Table des matières

1 Introduction

Le laboratoire de programmation orientée objet vise à mettre en pratique les différents concepts montrés dans le cadre du cours de théorie.

Il est découpé en trois parties. La première, propose la création d'un ensemble de classes manipulées dans le cadre d'une application console de test. Les deux suivantes sont destinées à la création d'applications fenêtrées.

Echéance des évaluations :

	Sujet	Echéance 22xx	Echéance B38
Phase I	Développement de classes et outils de base	Semaine du **2/3/2020**	Semaine du **2/3/2020**
Phase II	Application WPF - ????	Semaine du **30/3/2020**	Semaine du **30/3/2020**
Phase III	Application WPF - ????	**Evaluation de juin 2020**	

Les fonctionnalités à développer et l'architecture à mettre en place sont précisées de façon très claire et sont à respecter. En cas de retard, des sanctions seront prises.

1.1 Procédure de remise des dossiers

L'énoncé est à réaliser individuellement.

1.1.1 Création d'un compte étudiant sur la plateforme GitHub

Chaque étudiant doit créer un compte sur la plateforme GitHub (voir procédure dans la rubrique Git sur la plateforme Moodle). Ce compte sera utilisé par l'étudiant pour réaliser la sauvegarde de son code dans des repository/dépôts privés.

https://education.github.com/students

1.1.2 Procédure de remise d'un dossier

Les étudiants utiliseront activement l'outil Git ainsi que la plateforme GitHub pour gérer leur code source. Les dossiers seront remis par l'intermédiaire de cette plateforme.

Un lien sera donné en temps utile pour créer le repository lié à chaque travail demandé.

1.1.3 Ressources

Les ressources mise à la disposition des étudiants sont disponibles via la plateforme Moodle et GitHub.

Dans le dossier ci-dessous, se trouve la procédure d'installation des logiciels et composants ainsi que les énoncés de laboratoire.

https://github.com/HEPL-Moitroux-Programmation-CSharp-Base/Laboratoire

Les outils à utiliser pour les développements sont :

- Visual Studio Community 2017 ou 2019 en **ANGLAIS**
- Framework .NET 4.6 au minimum

L'ensemble des outils seront installés et configurés **avant** la première séance de laboratoire.

2 Fonctionnalités Phase I (Semaine du 2/3/2020)

Créer un ensemble de classes visant à caractériser des objets graphiques utilisés dans le domaine de la cartographie : des points d'intérêt, des lignes, des surfaces. En fonction des sujets, un point d'intérêt peut représenter une taque d'égout, un arbre remarquable ou un point de passage d'un trajet. Une ligne peut représenter le bord d'une route, un trajet effectué à pied ou en voiture. Une surface peut représenter un champ cultivé ou l'espace occupé au sol par un bâtiment par exemple.

L'ensemble de ces classes est contenu dans un projet librairie de classes situé dans la même solution que le projet console de test. **L'application console est créée puis construite en même temps que les différentes classes**. Les éléments spécifiques de chaque classe sont donc testés au fur et à mesure. Les fonctionnalités à tester sont décrites en fin de paragraphe.

2.1 Librairie de classe MyCartographyObjects

Il est important de commencer l'application de test en même temps que la création des classes. Chaque classe est créée puis immédiatement testée. Les tests des classes doivent toujours fonctionner même si les classes ont été modifiées.

Classe **Coordonnees**	Créer une classe « Coordonnees » décrite par : • Deux coordonnées de type double représentant la latitude et la longitude (variable membre et propriété). • Un constructeur d'initialisation. • Un constructeur par défaut qui utilise le constructeur d'initialisation. La surcharge de la méthode ToString() en utilisant le format suivant : (latitude, longitude) *TESTER LA CLASSE*
Classe **POI**	POI = abréviation de Point of Interest (point d'intérêt) Créer une classe **POI** qui hérite de **Coordonnees** et qui est décrite par • Une description (objet de type string), • Un constructeur d'initialisation, • Un constructeur par défaut qui initialise la HEPL comme point d'intérêt (adresse GPS du Parc des Marêt ou du Quai Gloesener) La surcharge de la méthode ToString() qui construit une chaîne de caractères et affiche les donnees de l'objet, sur une seule ligne. Les valeurs numériques affichent au maximum trois décimales. *TESTER LA CLASSE*

Classe **CartoObj**	Créer une classe <u>abstraite</u> **CartoObj** qui décrit tout objet cartographique. Elle doit contenir : • Un identifiant unique « id » de type entier (généré automatiquement à partir d'un compteur « static » du nombre d'instances de chaque sorte d'objet), • Un constructeur par défaut qui initialise l'id unique et est appelé par les constructeurs des classes qui héritent de **CartoObj**. • Une méthode ToString() qui renvoie la chaîne contenant l'id de l'objet. • Une méthode virtuelle Draw() qui affiche cette chaîne de caractères dans la fenêtre Console. • Modifier la / les classe précédentes pour qu'elles héritent de **CartoObj**, adapter les méthodes ToString() pour qu'elles affichent l'id des objets. *TESTER à nouveau LES CLASSES COORDONNEES et POI*
Classe **Polyline**	Créer une classe **Polyline** qui hérite de CartoObj et décrite par • Une collection de coordonnées (références d'objets **Coordonnees**), • Une couleur (voir classe System.Windows.Media.Colors), • Une épaisseur (int), • Un constructeur par défaut • Un constructeur d'initialisation • La surcharge de la méthode ToString() • La redéfinition de la méthode Draw() qui affiche les informations concernant la polyline dans la console TESTER LA CLASSE
Classe **Polygon**	Créer une classe **Polygon** qui hérite de CartoObj et décrite par • Une collection de coordonnées (références d'objets **Coordonnees**), • Une couleur de remplissage, • Une couleur de contour, • Un niveau d'opacité (double compris entre 0 et 1) • Un constructeur par défaut • Un constructeur d'initialisation • La surcharge de la méthode ToString() • La redéfinition de la méthode Draw() qui affiche les informations concernant l'objet polygon dans la console • TESTER LA CLASSE

Interface **IIsPointClose**	Créer une interface **IIsPointClose** implémentée par tous les objets cartographiques. Cette interface contient une méthode **IsPointClose** permettant de déterminer si un point dont les coordonnées (type double) passées en paramètre est proche (précision en paramètre) ou non de l'objet cartographique. Dans le cas du point d'intérêt, révisez Pythagore ☺. Dans le cas de la ligne, nous considérerons qu'un point se trouve proche de la ligne si elle est proche d'un de ses points ou si la distance qui sépare le point d'un des segments de celle-ci est inférieure la précision (voir paramètre ci-dessus). Dans le cas d'un **Polygone**, on utilise le point doit se trouver à l'intérieur de la bounding box (voir §2.3.2)englobant le polygone. L'objectif de cette interface est de pouvoir sélectionner un objet cartographique par clic sur une carte. TESTER tous les cas possibles
Interface **IPointy**	Créer une interface **IPointy** implémentée par les objets cartographiques ayant au moins 2 **Coordonnees** et qui retourne le nombre de points différents qui composent l'objet : Seuls la **Polyline** et le **Polygon** sont concernés. Dans cette interface, ajouter une propriété en lecture seule **NbPoints**. Elle retourne le nombre de points qui compose l'objet. Implémenter ces méthodes lorsque c'est nécessaire. Deux points sont différents s'ils ont des « id » différents. TESTER tous les cas possibles

2.2 Application console de test

Le projet en mode console permet de tester les différentes fonctionnalités des classes. Celles-ci seront complétées au fur et à mesure si nécessaire.

- Créer et afficher 2 objets de chaque sorte. Mettre en évidence l'utilisation des constructeurs, propriétés et méthodes de chaque classe.
- Ajouter ces objets dans une liste générique d'objets de type **`CartoObj`** (`List<`**`CartoObj`**`>`).
- Afficher cette liste en utilisant le mot clé `foreach`.
- Afficher la liste des objets implémentant l'interface `IPointy`.
- Afficher la liste des objets n'implémentant pas l'interface `IPointy`.
- Créer une liste générique de 5 Polyline, l'afficher, la trier par ordre de longueur croissante[1], l'afficher à nouveau. Pour trier, la méthode `Sort()` de la classe `List<T>` utilise la méthode `CompareTo()` définie grâce à l'implémentation dans la classe **`Polyline`** de l'interface `Icomparable<`**`Polyline`**`>`,
- Sans modifier la méthode `CompareTo()`, trier la précédente liste par ordre croissant de taille de surface de la bounding box englobant la polyline. Pour ce faire, il s'agit de créer

[1] La longueur d'une polyline se mesure par la somme des longueurs des segments qui la compose.

une classe **`MyPolylineBoundingBoxComparer`** implémentant l'interface `IComparer<`**`Polyline`**`>`,

- Rechercher, parmi les polyline de la liste, celles qui présentent la même taille qu'une polyline de référence. Pour ce faire, il s'agit d'implémenter l'interface `IEquatable<Polyline>`, et d'utiliser la méthode `Find()` ou `FindAll()` de la classe `List<T>`.
- Rechercher, parmi les polyline de la liste, celles qui sont proches d'un point passé en paramètre. Pour ce faire, il s'agit d'utiliser la méthode `IsPointClose()` contenue dans l'interface `IIsPointClose`.
- Mettre en place et tester un mécanisme qui permet de classer une liste d'objets **`CartoObj`** sur base du nombre d'objets **Coordonnees** qui le compose via un **…Comparer**.

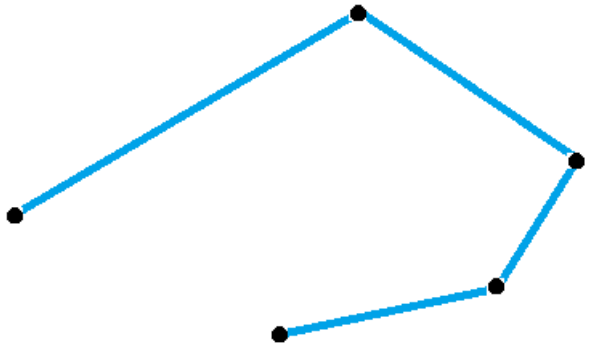
Remarque :

- Les comparaisons de longueur se font « à une précision près » déterminée par l'application. En aucun cas, l'opérateur == ne peut être utilisé pour comparer des variables de type « double ».
- Toutes les méthodes Draw() utilisent explicitement les méthodes ToString() définies dans chacune des classes.
- **Soyez attentifs à la qualité des messages affichés par la console pour la démonstration**. Par exemple, toutes les données d'un objet ou presque, peuvent être affichées sur une ligne en utilisant des tabulations pour aligner les mêmes types de paramètres.

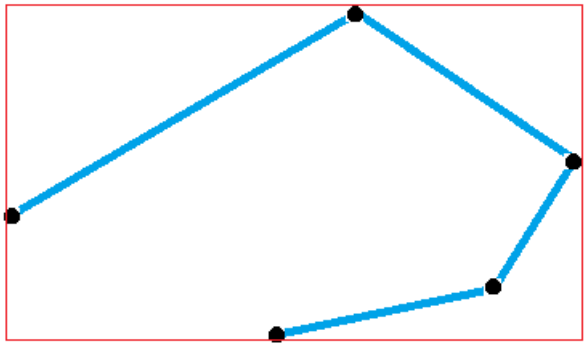
2.3 Quelques dessins pour mieux comprendre

2.3.1 Les objets Polyline

Le dessin suivant présente une polyline. Elle est composée de 5 POI ou Coordonnees sur lesquels s'appuient 4 segments.

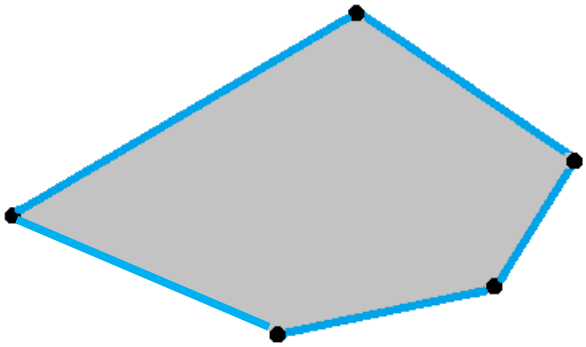


La bounding box d'une polyline est la plus petite boite rectangulaire qui contient tous les points d'une ligne et donc la ligne. Elle est représentée en rouge sur la figure suivante.



2.3.2 Les objets Polygon

Un polygone s'appuye sur une liste de POI ou Coordonnees.



2.4 Librairie MathUtil

Il est suggéré de créer une librairie contenant une classe MathUtil proposant quelques méthodes « static » permettant de :

- Calculer la distance entre deux points donnés par leurs coordonnées (x1, y1, x2, y2),
- Calculer la distance d'un point à un segment,
- …