Haute Ecole de la Province de Liège

Programmation .NET et C#

Enoncé de laboratoire Phases I, II et III

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 - MapData	Semaine du **30/3/2020**	Vendredi **3/4/2020**
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 gestion 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 abstraite **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`**`>`,

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

- 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 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**.

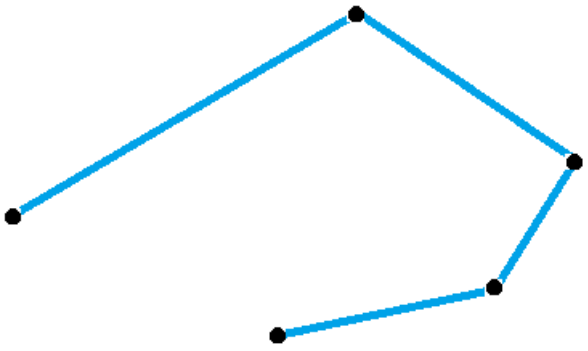
Remarque :

- 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.

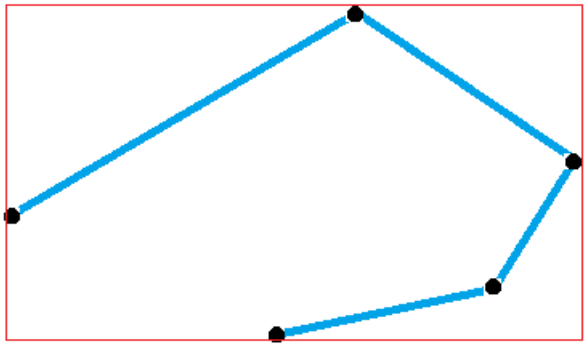
2.3 Quelques dessins pour mieux comprendre

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.

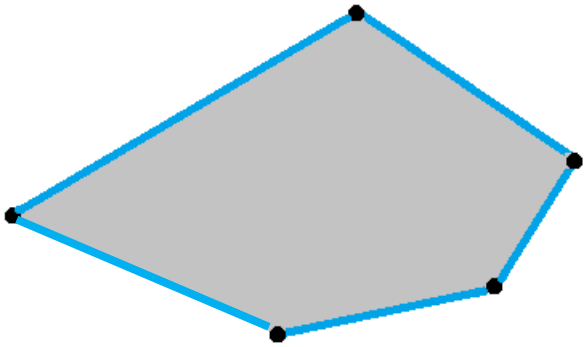


La bounding box d'une polyline est la plus petite boite rectangulaire qui contient tous les points d'une ligne et donc la ligne. Elle est représentée en rouge sur la figure suivante.



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,
- …

3 Fonctionnalités Phase II (Semaine du 30/03/2020)

Il s'agit de développer une application WPF permettant de gérer des objets cartographiques affichés sur une carte. Ces objets sont des points d'intérêt modélisés par la classe POI, des trajets modélisés par la classe Polyline et des surfaces modélisées par la classe Polygon

Développez votre application en fonction de vos connaissances. Le document ci-dessous ne donne pas la liste des choses à faire dans l'ordre où elles doivent l'être, mais donne plutôt une liste des caractéristiques des classes et de l'application finale. Lisez donc bien la totalité de l'énoncé avant de commencer votre nouvelle application. Travaillez étape par étape et testez au fur et à mesure.

3.1 Une nouvelle classe

Il s'agit de créer une classe `MyPersonalMapData` complète respectant le canevas déjà utilisé dans le précédent dossier et permettant de créer et de gérer l'ensemble des objets cartographiques manipulés par l'application graphique.

Les caractéristiques de l'objet MyPersonalMapData sont :

- Nom, Prénom
- Email
- ObservableCollection générique d'objets implémentant une nouvelle interface ICartoObj. Seuls les POI, Polyline et Polygon implémentent cette interface.

La classe `MyPersonalMapData` étant directement liée aux objets qu'elle contient, elle est ajoutée au projet « Class Library (.Net Framework) » MyCartographyObjects.

Cette classe propose

- Une propriété de type « get » qui retourne la collection,
- Des méthodes qui permettent de réinitialiser la collection, la charger ou l'enregistrer au format binaire. Le nom du fichier est donné par la concaténation du prénom et du nom. Il n'est pas demandé de gérer les mots de passe ni les doublons possibles dans les chaînes « prénom + nom ».

Celle-ci doit être testée grâce à un projet console de test avant de pouvoir être utilisée dans l'interface graphique.

3.2 Application WPF

Il s'agit de créer une application WPF « PersonalMap Manager » permettant de créer et gérer des objets cartographiques liés à un utilisateur et de les afficher sur une carte

Pour cela, il faut créer un nouveau projet C# WPF (.Net Framework) qui utilise la librairie de classe créée dans la phase I. Ce nouveau projet est ajouté à la solution de cette Phase I.

La première fenêtre est une fenêtre de connexion dans laquelle l'utilisateur encode son prénom, son nom et son adresse email. Si un fichier à son nom existe, il est ouvert et les données sont chargées. Sinon, les données encodées peuvent servir à un futur enregistrement.

La fenêtre principale est agencée de façon optimale en utilisant au mieux la notion de Layout. Nous attendons une utilisation pertinente de ceux-ci afin d'avoir une interface qui reste utilisable et ergonomique si la taille de la fenêtre est modifiée. Elle doit contenir au minimum un GridSplitter et un StackPanel. Dans celle-ci, on y trouve :

1. Un menu contenant au moins les éléments suivants :
    a. File → Open
    b. File → Save
    c. File → POI → Import
    d. File → POI → Export
    e. File → Trajet → Import
    f. File → Trajet → Export
    g. File → Exit
    h. Tools→ Option
    i. Tools →About box : Créer une « About box » comme il en existe dans de nombreux programmes (Auteur, copyright ☺, date).
2. Plusieurs zones destinées à :
    a. Encoder / modifier les caractéristiques (donc les propriétés) des objets cartographiques (Label, Textbox, TextBlock, ComboBox, …),
    b. Visualiser dans une ListBox, les objets cartographiques chargés / ajoutés. Dans la ListBox, on affiche uniquement la description du POI, du trajet ou de la surface,
    c. Visualiser ces mêmes objets sur une carte BingMap voir §3.5.
3. Une Toolbar est ajoutée juste en dessous du menu. Elle contient des boutons permettant d'activer les différentes actions de création, modification, suppression, sur les objets POI, Trajets et Surface. Choisissez une organisation des boutons dans la Toolbar pour que votre utilisateur comprenne facilement comment il doit s'en servir.
4. Finalement, il doit être possible de :
    a. Créer un « point d'intérêt » par encodage de données ou par clic sur la carte,
    b. Créer un « trajet » par clic sur la carte. Une liste de Coordonnee peut contenir les POI récupérés par clic,
    c. Créer une « surface » par clic (BONUS),
    d. Sélectionner dans la ListBox ou par Clic puis
        i. visualiser les détails,
        ii. supprimer,
        iii. modifier (couleur, description, …).

La fenêtre « Option » est une boîte de dialogue secondaire non modale (non bloquante) permettant de :

1. Initialiser le « dossier de travail »,
2. Choisir et modifier la couleur utilisée pour afficher la ListBox contenant les objets visualisés sur la carte (couleur du texte et couleur du fond).

Elle fonctionne de la même manière que la fenêtre qui permet de gérer la résolution de l'écran sous Windows, elle contient donc 3 boutons :

- OK : valide le choix et ferme la fenêtre,
- Cancel : ferme la fenêtre sans valider le dernier choix (qui n'a pas encore été appliqué),
- Appliquer : valide le choix, ne ferme pas la fenêtre.

La mise à jour de l'interface utilisateur principale se fera par la création d'un **événement personnalisé** envoyé par cette fenêtre à la fenêtre principale au moment opportun.

3.3 Informations complémentaires

- Le « dossier de travail » est un répertoire choisi par l'utilisateur de l'application dans lequel les fichiers de données sont enregistrés. Il est initialisé dans la fenêtre secondaire « Options ». A ce stade du projet, il n'est pas demandé de sauvegarder le dossier de travail mais seulement de le conserver en mémoire pour le cas où l'utilisateur demanderait à ouvrir ou enregistrer un fichier.
- La sauvegarde (File→Save) et le chargement (File→Open) doivent permettre de sauvegarder l'objet MyPersonalMapData courant dans un fichier binaire (Save) ou de charger un fichier binaire contenant des données précédemment sauvegardées (Open).
- Un POI ou un trajet au format texte peut être importé ou exporté de ou vers un fichier portant l'extension « .csv ». Il est possible d'en créer à partir d'un fichier Excel (choisir CSV avec le ; comme séparateur). Si tout le monde a le même format, il doit être possible d'importer les POI ou les trajets d'un autre étudiant. Les fichiers d'exemple (voir §0) se trouvent sur la plateforme Moodle du cours.
- Pensez à la modularité du code et mettez en évidence vos compétences de conception. Permettez de rendre ce code réutilisable si cela vous semble légitime. Aussi, veillez à ne pas réinventer la roue, servez-vous un maximum des fonctionnalités déjà existantes.

3.4 Exemples de fichier à importer / exporter

3.4.1 Point of Interest

Soit le fichier « HEPL Seraing POI.csv »

	A	B	C	D
1	50,610597	5,509206	HEPL Seraing	
2				
3				

La colonne A contient la latitude, la colonne B contient la longitude et la colonne C, la description.

3.4.2 Trajet

Soit le fichier « HEPL Seraing Liege Trajet.csv »

	A	B	C
1	50,610597	5,509206	HEPL Seraing
2	50,610013	5,514195	
3	50,605601	5,543163	
4	50,611906	5,584131	
5	50,619873	5,581654	HEPL Liege
6			

Plusieurs lignes du fichier constituent l'ensemble d'un trajet. Chaque point de passage peut être un **POI** (cas du premier et du dernier point) ou une **Coordonnee**.

3.5 Carte Bing

Il s'agit ici d'utiliser un composant existant permettant de visualiser une carte et de déposer dessus des composants cartographiques tels que des points d'intérêt, des lignes ou des surfaces. La documentation permettant d'utiliser le composant se trouve ici :

https://docs.microsoft.com/en-us/previous-versions/bing/wpf-control/hh750210(v=msdn.10) → lire les différentes sections.

1. Créer un compte développeur pour pouvoir utiliser / afficher les cartes dans une application que vous développez. https://www.bingmapsportal.com/Account Créer une nouvelle clé (donner le nom de votre application, choisir le type « Basic » et Application type = Dev/Test)

   Récupérer la clé de votre application,

2. Télécharger le SDK (https://www.microsoft.com/en-us/download/details.aspx?id=27165) puis ajouter une référence vers l'assembly Microsoft.Maps.MapControl.WPF pour pouvoir ajouter les composants dans votre projet WPF. Voir rubrique « Getting Started » https://docs.microsoft.com/en-us/previous-versions/bing/wpf-control/hh745791(v=msdn.10) puis « Creating an Application ». https://docs.microsoft.com/en-us/previous-versions/bing/wpf-control/hh830433(v=msdn.10),

3. Ajouter le composant graphique à votre projet WPF, voir exemple https://docs.microsoft.com/en-us/previous-versions/bing/wpf-control/hh709042(v=msdn.10) en insérant la clé créée au §1 ci-dessus,
4. Consulter les exemples de code des § qui suivent pour changer l'échelle de la carte, la centrer sur des coordonnées GPS données (https://docs.microsoft.com/en-us/previous-versions/bing/wpf-control/hh830433%28v%3dmsdn.10%29),
5. L'objet Pushpin sera utilisé pour visualiser un POI,
6. L'objet MapPolyline sera utilisé pour visualiser une Polyline,
7. L'objet MapPolygon sera utilisé pour visualiser un Polygon.
8. L'objet Location contient des coordonnées GPS.
9. Les événements sur la carte seront utilisés en fonction des cas pour sélectionner un objet existant sur celle-ci pour créer des objets cartographiques.

Volontairement, nous avons voulu créer un premier ensemble de classes (MyCartographyObject) complètement indépendant des classes proposées par la nouvelle librairie ajoutée. Nous voulons que la carte Bing ne soit utilisée que pour l'affichage et l'interaction avec la carte via des clics sur base de la logique de commande que vous mettrez en place. Il s'agit donc de maintenir en mémoire la liste d'objets de la première librairie ainsi que la liste des objets utiles à la visualisation sur la carte Bing.

3.6 Evaluation

L'évaluation portera sur :

1. La qualité de la solution (découpage en projets)
2. Respect des règles de nommage des composants graphiques, des variables membres, propriétés et méthodes
3. Respect de l'énoncé et des fonctionnalités demandées
4. La gestion de la liste des objets en mémoire et la mise à jour automatique des données lors de modifications
5. L'optimisation de l'utilisation des technologies

4 Fonctionnalités Phase III (juin 2020)

En cours de rédaction