# Production finale

## Automne 2022

420-201-RE
Introduction à la programmation


## Auteurs :

- **Patrice Roy**
- **Pierre Prud'homme**
- **Maxime Barakatt**


## Table des matières

# A. Préambule

Dans le cours 420-201-RE comme dans tous les cours que vous suivrez au niveau collégial, il est requis de vérifier votre atteinte d'un niveau de compétence suffisant dans les différents aspects de la matière enseignée au cours de la session. Votre capacité à réaliser le travail décrit dans l'énoncé ci-dessous nous permettra d'évaluer ce niveau de compétence. L'examen final qui suivra la remise de cette production finale d'intégration (PFI) permettra de valider notre évaluation de votre compétence.

Évidemment, ce travail est strictement individuel et ne peut donc pas être réalisé, même partiellement, à plusieurs.

# B. Description générale du problème

Vos enseignants œuvraient à la conception d'une première version d'un logiciel nommé **Paysagiste** permettant l'édition de jolis paysages faits de formes géométriques toutes plus jolies les unes que les autres. Dans cette première version, le logiciel devait permettre l'ajout, dans un canevas, de trois formes géométriques : le carré, le triangle et le rectangle.

Lorsqu'on affiche le canevas, les formes créées sont affichées en ordre de profondeur, c'est-à-dire que la forme créée en premier est située le plus profondément dans le canevas, la forme créée en deuxième est située moins profondément dans le canevas que l'autre, et ainsi de suite jusqu'à la forme créée en dernier qui est ainsi la moins profonde dans le canevas. Cet ordre est de grande importance puisque certaines formes moins profondes peuvent avoir une intersection avec une forme située plus profondément dans le canevas, ce qui détermine laquelle on verra par-dessus l'autre dans l'affichage final du canevas.

En plus de pouvoir ajouter des formes au canevas, il est aussi possible d'y supprimer des formes qu'on y aurait déjà inscrit. La suppression d'une forme fait en sorte que toutes les autres seront décalées en profondeur. Par bonheur, la programmation requise pour gérer ce décalage a survécu à « l'accident ».

On peut également faire avancer et reculer une forme dans le canevas, c'est-à-dire modifier l'ordre des formes dans le canevas. Cependant, on ne peut faire avancer ou reculer une forme que d'un seul rang à la fois (pour déplacer une forme de plusieurs rangs, il suffit de la déplacer plusieurs fois). Enfin, il est possible de déplacer une forme à l'écran.

Alors que le travail était pratiquement achevé, un des enseignants qui travaillait sur ce projet a très maladroitement détruit une partie non négligeable du fruit de leur labeur, ce qui est connu depuis comme « l'accident ». La postérité taira le nom de cet illustre maladroit.

Étonnamment, les commentaires associés aux fonctions et autres passages élagués par cette maladresse sont demeurés inscrits dans le code source, ce qui permettra à qui sait lire de saisir l'intention derrière chacun des passages disparus et, qui sait, peut-être même pallier cette bêtise. Plus surprenant encore, il reste aussi, quelque part dans les archives de vos enseignants, un exécutable fonctionnel de l'époque où les sources étaient encore intactes.

Étant trop occupés à leurs autres tâches, vos enseignants font preuve à votre égard d'une confiance à toute épreuve, et vous demandent de reconstituer les fonctions et les passages manquants de manière à retrouver un programme ayant le comportement attendu mais temporairement perdu.

## C. *Structure de la solution*

La solution qui vous est fournie se compose de trois fichiers sources, dont l'un est nommé **`PartieÉtudiante.cs`**. Ce fichier est proprement commenté, mais– tel qu'annoncé – des passages en ont été détruits.

Dans certains cas, une fonction encore en place pourra vous inspirer à rédiger une fonction manquante. Dans d'autres cas, il se peut que vous trouviez des pistes dans vos notes de cours. Enfin, dans certains cas, il vous faudra tout simplement faire preuve de créativité.

## D. À remettre

Vous devrez modifier puis remettre le fichier **`PartieÉtudiante.cs`**. Évidemment, si toutes et tous remettent un fichier de ce nom, vos enseignants ne sauront pas à qui donner des points. Ainsi, votre remise se fera en deux temps :

- Par ColNet, chaque étudiante et chaque étudiant devra envoyer la version électronique de ce fichier en pièce jointe d'un message à son enseignant, **mais seulement après l'avoir renommé** ***`NoGroupe-NomPrénom.cs`*** (p. ex. : `05-NguyenKim.cs` si votre nom est Kim Nguyen et vous faites partie du groupe `05`).

- En format papier, chaque étudiante et chaque étudiant devra aussi remettre une copie de ce fichier modifié. L'en-tête du fichier devra être identifié dans le respect des règles indiquées dans vos notes de cours (voir les documents du début de la session).

## E. Évaluation

Vous serez évalué(e) sur la qualité de vos réalisations en lien avec le fonctionnement du projet, de la conception de votre solution et la qualité de son implantation. Nous vous noterons principalement sur les aspects suivants :

- Fonctionnement lors des tests d'exécution (50% de la note finale). Le comportement attendu avec votre version du programme doit correspondre en tout point au comportement obtenu avec le programme original
- Respect des normes et des conventions
- Documentation de votre solution, incluant les aspects relevant de l'auto-documentation
- Respect du découpage en fonctions et du rôle de chacune d'elles. Notez que vous pouvez ajouter des fonctions et des types si vous le souhaitez, mais que les fonctions et les types déjà présents (ou à tout le moins déjà commentés) doivent demeurer dans le programme, et y jouer le rôle décrit dans les commentaires
- Utilisation judicieuse et habile des structures de contrôles
- Utilisation des fonctions et des mécanismes d'échange d'information. En particulier, la répétition de code semblable devrait laisser place à l'écriture de fonctions correctement paramétrées – appliquez le principe DRY, pour *Don't Repeat Yourself*[1]
- Choix approprié des types de variables
- Respect des consignes indiquées dans le présent document

[1] http://h-deb.clg.qc.ca/Sujets/Developpement/Lois-principes.html#principe_dry

## F. Structure de `PartieÉtudiante.cs`

Le fichier `PartieÉtudiante.cs` que vous devez compléter est de prime abord structuré par des commentaires délimitant les zones où se trouvent des fonctions et des types ayant une vocation commune. Une section à la fin du fichier est réservée pour l'ajout de fonctions ou de types que vous estimeriez utiles mais qui ne sont pas explicitement demandés; prenez donc soin d'y loger vos ajouts personnels s'il y a lieu.

## G. Date de remise

Cette activité synthèse doit être complétée et remise avant le début du dernier cours régulier de la session, soit le **mardi 13 décembre 2022**.

# H. Fonctionnement général de l'application

Le `Paysagiste` est un éditeur de paysages composés de formes géométriques. À l'aide d'un canevas (essentiellement, un tableau `2D` rectangulaire de caractères), il permet à des artistes comme vous d'exprimer leur créativité en dessinant des formes à la hauteur de la poésie qui se loge dans leur cœur, en les superposant et en les agençant dans le respect de leur légendaire sens de l'esthétique.

Ce logiciel est une application de type console basée sur un menu d'options permettant de réaliser certaines opérations de base sur un canevas. Au démarrage, l'usager choisit les dimensions du canevas, puis exprime sur ce canevas sa créativité à partir des options que le menu lui propose.

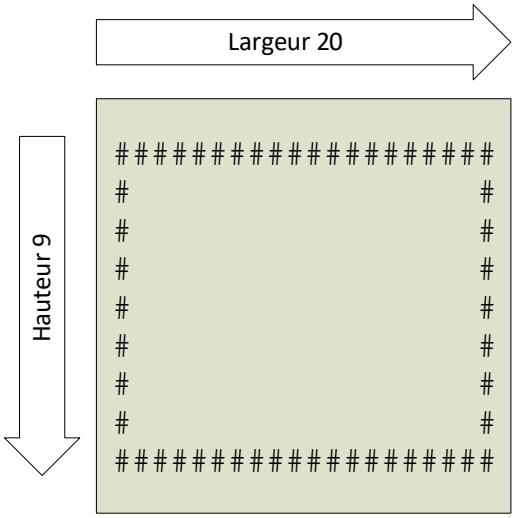


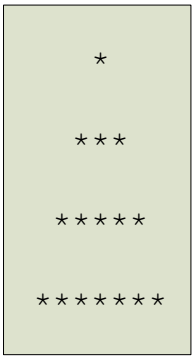

**Figure 1 Canevas vide (hauteur 9, largeur 20)**

## 1. Cadre

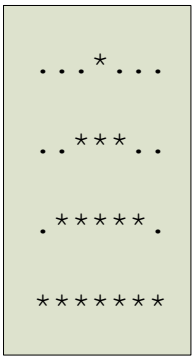
Dans l'exécution du `Paysagiste`, des points et diverses formes seront placé(e)s sur le canevas. Dans tous les cas, ce qui est placé sur le canevas doit se situer en entier à l'intérieur du canevas : par exemple, pour le canevas donné en exemple ci-dessus, les positions `0` et `8` sur l'axe des $y$ sont réservées pour le cadre du canevas, et il en va de même pour les positions `0` et `19` en $x$.

Pour être dans le cadre, une forme doit se situer entièrement à l'intérieur du canevas. Pour un carré ou un rectangle, il est relativement simple de valider cela : le coin en haut et à gauche de la forme doit être dans le cadre, et il doit en être de même du coin en bas et à droite.

Les règles pour un triangle méritent une attention particulière. Lorsqu'il est dessiné, un triangle doit être centré. À titre d'exemple, ce qui suit est un triangle de hauteur `4` dessiné à l'aide du symbole `*` :




L'espace occupé par un triangle est défini par son rectangle englobant le plus petit. À titre d'illustration, avec le triangle ci-dessus, l'espace occupé par le triangle est décrit par le rectangle suivant :




La position du triangle sur le canevas est le coin en haut et à gauche de son rectangle englobant.

## 2. Axes

Le système d'axes d'un canevas n'est pas un système cartésien classique, en ce sens que le point à la position `0,0` est en haut et à gauche. Les coordonnées vont en croissant vers le bas (hauteur) et vers la droite (largeur).

Par exemple, le canevas suivant contient deux formes, soit un rectangle de hauteur `2` et de largeur `5` dessiné à l'aide du symbole % et situé à la position `x:3, y:3` et un triangle de hauteur `3` dessiné à l'aide du symbole `$` et situé à la position `x:5, y:6`, le tout sur un canevas de hauteur `10` et de largeur `20` :

```
####################
#                  #
#                  #
#   %%%%%          #
#   %%%%%          #
#                  #
#       $          #
#      $$$         #
#     $$$$$        #
####################
```

## 3. Profondeur

Dans un canevas, des formes sont dessinées. Celles-ci sont dans un tableau, ordonnancées de la plus profonde à la moins profonde. Ainsi, pour dessiner sur le canevas, la forme à la profondeur `0` sera dessinée d'abord, puis celle à la profondeur `1`, puis celle à la profondeur `2`, etc. Changer la profondeur d'une forme consistera donc à permuter deux formes dans un tableau.

## 4. Type `Forme`

Certaines fonctions utilisent un type `Forme`. Ce type représente une forme au sens large, mais vous n'avez pas besoin d'en connaître le détail.

# I. Description des fonctionnalités à implémenter

Vous avez bien sûr accès à la totalité du code existant (ou, devrait-on dire, restant). Évidemment, vous devrez combler les (nombreux) manques résultant de la maladresse de l'un d'entre nous (qui est – presque – profondément désolé de cette situation). *Tout ce que vous coderez dans* `PartieÉtudiante.cs` *devra être précédé du qualificatif public.*

Prenez soin de respecter les consignes qui suivent avec rigueur; le code existant dépend des noms indiqués dans ce qui suit, de même que de l'ordre des paramètres et de leurs types.

# 1. À propos des sortes de formes

Ce qui suit détaille ce que vous devrez faire pour que le volet « sortes de formes » fonctionne à nouveau.

### a) Énumération `SorteForme`

L'énumération `SorteForme` liste symboliquement les sortes de formes supportées par le logiciel. Cette énumération doit contenir les symboles `Carré`, `Rectangle` et `Triangle`.

### b) Fonction `AfficherSortesFormes()`

**Intrants** : aucun

**Extrant** : aucun

**Rôle** : afficher chaque sorte de forme accompagnée de sa valeur entière (p. ex. : `"Carré : 0"`).

**Note** : les sortes de formes doivent être délimitées à l'affichage par le symbole `'|'`, comme par exemple `"| Carré : 0 | Rectangle : 1 | ... etc."`.

### c) Fonction `LireSorteForme()`

**Intrants** : aucun

**Extrant** : la sorte de forme lue et validée

**Rôle** : lire et valider une sorte de forme

## 2. À propos des outils d'entrée / sortie

Ce qui suit détaille ce que vous devrez faire pour que le volet « entrées / sorties » du logiciel recommence à fonctionner.

### a) Fonction `LireEntier(invite,min,max)`

**Intrants** :

- **`invite`** : message expliquant la nature de l'entrée à réaliser (texte)
- **`min`** : borne minimale des valeurs acceptables en entrée (entier)
- **max** : borne maximale des valeurs acceptables en entrée (entier)

**Note** : les bornes sont inclusives; l'intervalle acceptable est $[min, max]$

**Extrant** : la valeur lue et validée (entier)

**Rôle** : lire et valider un entier devant se situer dans $[min, max]$

### b) Fonction `LirePoint(canevas)`

**Intrant** :

- **`canevas`** : l'espace dans lequel le `Point` lu devra se situer (tableau `2D` rectangulaire de `char`)

**Note** : les contours du `canevas` ne sont pas des endroits acceptables pour un `Point` tel que lu par cette fonction

**Extrant** : le `Point` lu et validé

**Rôle** : lire et valider un `Point` devant se situer dans le `canevas`

### c) Fonction `LireSymbole()`

**Intrants** : aucun

**Extrant** : le symbole lu et validé (caractère)

**Rôle** : lire un symbole qui sera utilisé pour fins d'affichage à l'écran

**Note** : tout symbole (tout `char`) sera acceptable. L'usager est responsable de ne pas entrer un symbole non-affichable ou un blanc

**Note** : le `Paysagiste` affichera les formes en couleur, mais pour ce faire il associera une couleur à chaque symbole. Ainsi, si vous utilisez plusieurs fois le même symbole pour des formes distinctes, elles auront toutes la même couleur.

### d) Fonction `LireCarré(canevas)`

**Intrant** :

- **`canevas`** : l'espace dans lequel le `Carré` lu devra se situer (tableau `2D` rectangulaire de `char`)

**Note** : le `Carré` lu doit se situer en entier à l'intérieur du `canevas`, mais cette validation ne sera faite que partiellement par `LireCarré()`, et sera complétée ultérieurement par `EstDansCanevas(canevas,carré)`

**Extrant** : le `Carré` lu

**Rôle** : lire et valider un `Carré` dont la position se situe dans le `canevas` et dont la hauteur ne dépasse pas celle du `canevas`

### e) Fonction `LireRectangle(canevas)`

**Intrant** :

- **`canevas`** : l'espace dans lequel le `Rectangle` lu devra se situer (tableau `2D` rectangulaire de `char`)

**Note** : le `Rectangle` lu doit se situer en entier à l'intérieur du `canevas`, mais cette validation ne sera faite que partiellement par `LireRectangle()`, et sera complétée ultérieurement par `EstDansCanevas(canevas,rectangle)`

**Extrant** : le `Rectangle` lu

**Rôle** : lire et valider un `Rectangle` dont la position se situe dans le canevas et dont la hauteur et la largeur ne dépassent pas celles du `canevas`

### f) Fonction `LireTriangle(canevas)`

**Intrant** :

- **`canevas`** : l'espace dans lequel le `Triangle` lu devra se situer (tableau `2D` rectangulaire de `char`)

**Note** : le `Triangle` lu doit se situer en entier à l'intérieur du `canevas`, mais cette validation ne sera faite que partiellement par `Lire Triangle()`, et sera complétée ultérieurement par `EstDansCanevas(canevas,triangle)`

**Extrant** : le `Triangle` lu

**Rôle** : lire et valider un `Triangle` dont la position se situe dans le `canevas` et dont la hauteur ne dépasse pas celle du `canevas`

### g) Fonction `ChoisirForme(formes)`

**Intrants** :

- **`formes`** : un tableau de `Forme` contenant les formes destinées au `canevas`

**Extrant** : l'indice de la forme choisie ou `-1` si le tableau est de taille nulle (zéro)

**Rôle** : lire et valider un indice de forme valide pour le tableau `formes` si le tableau n'est pas de taille nulle (zéro)

## 3. À propos des prédicats utilitaires

Un certain nombre de prédicats utilitaires devront être réécrits suite à la catastrophe qui les a faits disparaître, sans quoi le `Paysagiste` ne fonctionnera pas.

### a) Fonction `EstEntreInclusif(valeur,min,max)`

**Intrants** :

- **`valeur`** : valeur dont on souhaite valider l'appartenance à un intervalle (entier)
- **`min`** : borne minimale de l'intervalle (entier)
- **max** : borne maximale de l'intervalle (entier)

**Note** : les bornes sont inclusives; l'intervalle acceptable est $[min, max]$

**Extrant** : vrai seulement si la valeur est dans l'intervalle $[min, max]$ , faux sinon

### b) Fonction `EstProfondeurValide(formes,profondeur)`

**Intrants** :

- **`formes`** : un tableau des formes destinées au canevas
- **`profondeur`** : la profondeur candidate (entier)

**Extrant** : vrai seulement si `profondeur` est une position valide dans le tableau `formes` et faux sinon

### c) Fonction `EstDansCanevas(canevas,point)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le `Point` reçu en paramètre devra se situer (tableau `2D` rectangulaire de `char`)
- **`point`** : le `Point` candidat

**Note** : les contours du `canevas` ne sont pas des endroits acceptables pour un `Point` tel que celui reçu en paramètre par cette fonction. Un point qui toucherait au contour du `canevas` n'est pas dans le `canevas`.

**Extrant** : vrai seulement si le `point` est dans le `canevas` et faux sinon

### d) Fonction `EstDansCanevas(canevas,carré)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le `Carré` reçu en paramètre devra se situer (tableau `2D` rectangulaire de `char`)
- **`carré`** : le `Carré` candidat

**Note** : les contours du `canevas` ne sont pas des endroits acceptables pour un `Carré` tel que celui reçu en paramètre par cette fonction. Un `carré` qui toucherait au contour du `canevas` n'est pas dans le `canevas`.

**Extrant** : vrai seulement si le `carré` est entièrement dans le `canevas` et faux sinon

### e) Fonction `EstDansCanevas(canevas,rectangle)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le `Rectangle` reçu en paramètre devra se situer (tableau `2D` rectangulaire de `char`)
- **`rectangle`** : le `Rectangle` candidat

**Note** : les contours du `canevas` ne sont pas des endroits acceptables pour un `Rectangle` tel que celui reçu en paramètre par cette fonction. Un `rectangle` qui toucherait au contour du `canevas` n'est pas dans le `canevas`.

**Extrant** : vrai seulement si le `rectangle` est entièrement dans le `canevas` (donc faux sinon)

### f) Fonction `EstDansCanevas(canevas,triangle)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le `Triangle` reçu en paramètre devra se situer (tableau `2D` rectangulaire de `char`)
- **`triangle`** : le `Triangle` candidat

**Note** : les contours du `canevas` ne sont pas des endroits acceptables pour un `Triangle` tel que celui reçu en paramètre par cette fonction. Un `triangle` qui toucherait au contour du `canevas` n'est pas dans le `canevas`.

**Extrant** : vrai seulement si le `triangle` est entièrement dans le `canevas` et faux sinon

## 4. À propos des fonctions de dessin

Le `Paysagiste` est un logiciel de dessin, qui a tristement été dépossédé de cette fonctionnalité au cœur de son succès. Heureusement que vous êtes là…

### a) Fonction `DessinerCarré(canevas,symbole,position,hauteur)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le carré devra se situer (tableau `2D` rectangulaire de `char`)
- **`symbole`** : le symbole à utiliser pour dessiner le carré (caractère)
- **`position`** : la position du coin supérieur gauche du carré sur le `canevas` (point)
- **`hauteur`** : la hauteur du carré (entier)

**Extrant** : aucun

**Rôle** : dessiner sur le `canevas` le carré décrit par les paramètres

### b) Fonction `DessinerRectangle(canevas,symbole,position,hauteur,largeur)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le rectangle devra se situer (tableau `2D` rectangulaire de `char`)
- **`symbole`** : le symbole à utiliser pour dessiner le rectangle (caractère)
- **`position`** : la position du coin supérieur gauche du rectangle sur le `canevas` (point)
- **`hauteur`** : la hauteur du rectangle (entier)
- **`largeur`** : la largeur du rectangle (entier)

**Extrant** : aucun

**Rôle** : dessiner sur le `canevas` le rectangle décrit par les paramètres

### c) Fonction `DessinerTriangle(canevas,symbole,position,hauteur)`

**Intrants** :

- **`canevas`** : l'espace dans lequel le triangle devra se situer (tableau `2D` rectangulaire de `char`)
- **`symbole`** : le symbole à utiliser pour dessiner le triangle (caractère)
- **`position`** : la position du coin supérieur gauche du triangle sur le `canevas` (point)
- **`hauteur`** : la hauteur du triangle (entier)

**Extrant** : aucun

**Rôle** : dessiner sur le `canevas` le triangle décrit par les paramètres

### d) Fonction `DécrirePaysage(descriptionFormes)`

**Intrants** :

- **`descriptionFormes`** : tableau de chaînes de caractères décrivant les formes destinées à être affichées sur le canevas

**Note** : les descriptions sont placées en ordre croissant de profondeur, donc de la plus profonde à la moins profonde

**Extrant** : aucun

**Rôle** : afficher les descriptions des formes, en ordre croissant de profondeur, en prenant soin d'indiquer l'indice de chacune

**Note** : si `descriptionFormes` est de taille nulle (zéro), alors cette fonction affiche le texte `"Le paysage est désert"`

### e) Fonction `Afficher(canevas)`

**Intrants** :

- **`canevas`** : le canevas à dessiner (tableau `2D` rectangulaire de `char`)

**Extrant** : aucun

**Rôle** : afficher le `canevas` à l'écran. On comprend qu'afficher le `canevas` permettra de voir les contours ainsi que toutes les formes que ce `canevas` contient dans leur ordre de profondeur.

**Note** : pour afficher un `char` à l'écran, appelez la fonction `AfficherSymbole` (elle se trouve dans `PartieProfs.cs`) plutôt que `Console.Write`, car cette fonction affichera le caractère en utilisant la couleur choisie en créant la forme associée au symbole (s'il y a lieu).

## 5. Autres outils

Quelques outils plus difficiles à catégoriser manquent aussi pour que le `Paysagiste` puisse accomplir sa fonction.

### a) Fonction `Permuter(f0, f1)`

**Intrants** :

- **`f0`** : référence sur une `Forme`
- **`f1`** : référence sur une `Forme`

**Extrants** :

- **`f0`** : référence sur une `Forme`
- **`f1`** : référence sur une `Forme`

**Rôle** : permuter `f0` et `f1`

### b) Fonction `ChangerProfondeur(formes,laquelle,déplacement)`

**Intrants** :

- **`formes`** : un tableau de `Forme`
- **`laquelle`** : indice de la `Forme` à déplacer
- **`déplacement`** : déplacement à réaliser

**Notes** :

- le déplacement doit être `1` ou `-1` inclusivement. Un déplacement d'une autre valeur ne doit pas avoir d'effet
- un déplacement qui amènerait à déborder du tableau `formes` ne doit pas avoir d'effet

**Rôle** : permuter la `Forme` située à la position `laquelle` avec la `Forme` située à la position `laquelle + déplacement`

**Extrant** : le tableau `formes` suite à la permutation si elle était réalisable

# J. Description des fonctionnalités déjà implémentées

Heureusement, « l'accident » n'a pas tout détruit. À titre informatif, voici un survol de ce qui a survécu à la maladresse de ce prof anonyme et innommable. Considérez ce qui suit comme un complément d'information, pour les curieuses et les curieux parmi vous.

Ce qui suit décrit sommairement du fichier `PartieProfs.cs`. Notez que vous n'avez pas le droit de modifier ce fichier. Vous n'avez pas non plus le droit de modifier `Program.cs`.

## 1. Classe `Point`

La classe `Point` modélise une coordonnées $\{x, y\}$ sur un canevas.

## 2. Classe `Forme`

La classe `Forme` modélise une forme géométrique de manière abstraite. *Nous n'avons pas vu les techniques pour faire une telle classe encore; ces techniques sont au menu du prochain cours de programmation*. Notez que vous n'avez pas besoin de comprendre la classe `Forme` pour faire ce travail.

## 3. Classes **Triangle, Rectangle** et **Carré**

Les classes `Triangle`, `Rectangle` et `Carré` modélisent les formes géométriques du même nom. Ce que vous devez savoir à leur sujet est (a) comment appeler leur constructeur et (b) quelles sont leurs propriétés. En effet :

Créer un `Triangle` de hauteur `5` avec le symbole `'*'` à la position {2,3}

```
Point pt = new Point(2,3);
Triangle triangle =
   new Triangle('*', pt, 5);
```

Créer un `Rectangle` de hauteur `5` et de largeur `4` avec le symbole `'*'` à la position {2,3}

```
Point pt = new Point(2,3);
Rectangle rect =
   new Rectangle('*', pt, 5, 4);
```

Créer un `Carré` de hauteur `5` avec le symbole `'*'` à la position {2,3}

```
Point pt = new Point(2,3);
Carré carré = new Carré('*', pt, 5);
```

Les trois classes ont une propriété `Symbole` de type `char`. Les trois classes ont une propriété `Position` de type `Point`. Les trois classes ont une propriété `Hauteur` de type `int`. `Rectangle` a en plus une propriété `Largeur` de type `int`.

## 4. Classe `Paysage`

La classe `Paysage` modélise… un paysage. C'est par elle que le programme contrôle et organise le canevas et les formes qui s'y retrouvent. Dans ce travail, votre code ne sera pas en contact direct avec cette classe.

## 5. Fichier `Program.cs`

Le contenu du fichier `Program.cs` est le programme principal; il décrit et organise le fonctionnement à haut niveau du programme `Paysagiste`, et assure le relais des outils existants vers les fonctions et les outils que vous devez programmer vous-mêmes.

# Bon travail!