# CONCEPTS AVANCÉS EN PROGRAMMATION ORIENTÉE OBJET

## GUIDE DE L'ÉTUDIANTE ET DE L'ÉTUDIANT

## S2 – APP3GI

***HIVER 2025***

Département de génie électrique et de génie informatique
Faculté de génie
Université de Sherbrooke

Auteur : Charles-Antoine Brunet
Version : H2025-01 (2025-02-03)

Ce document est réalisé avec l'aide de LaTeX et de la classe `genie-app-guide`.

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

## NOTICE

Ce guide a été généré sans la partie référence, car elle se retrouve en ligne sur le site de l'activité.

# 1 GUIDE DE LECTURE

**Langage C++ :** Les lectures pour le C++ sont tirées du livre de référence [2].

- Leçon 9 (pages 237 à 244) : constructeur de copie (révision de l'APP 1) et aussi copie superficielle (*shallow copy*) et copie profonde (*deep copy*).
- Leçon 12 (pages 336 à 365) : surcharge d'opérateurs (*operator overloading*)
- Leçon 14 (pages 402 à 411) : les modèles (*templates*)
- Leçon 18 (pages 476 à 483) : les listes, lecture surtout pour les concepts
- Leçon 24 (pages 604 à 613) : les files (*queues*) et les piles (*stacks*), lecture surtout pour les concepts
- Leçon 27 (pages 650 à 664) : entrées et sorties de fichiers

**UML :** Le lien entre les modèles (*templates*) en C++ et les classes dans un diagramme de classes est abordé dans le livre d'UML [1] aux pages 79 et 80. Ce sont les *classes paramétrables*.

**Pseudocode :** Le standard de pseudocode utilisé, si nécessaire, est celui de S1. Le guide est redonné sur la page web de l'unité.

**Aide-mémoire :** L'aide-mémoire du C++ de l'APP1 est redonné sur la page web de l'unité.

# 2 LOGICIELS ET MATÉRIEL

Des indications pour l'installation des logiciels utilisés sont disponibles sur la page web de la session. Les logiciels nécessaires sont les suivants :

**Visual Studio 2022 :** Consultez le site web de session pour l'installation. Visual Studio 2022 est déjà installé dans les laboratoires du département.

**Qt :** Consultez le site web de session pour des directives et conseils d'installation. Qt est déjà installé dans les laboratoires du département.

**GraphicusGUI :** GraphicusGUI est une librairie distribuée sur la page web de l'unité. Pour les détails d'installation, consultez le fichier *readme.txt* distribué avec GraphicusGUI.

**TestGraphicusGUI :** TestGraphicusGUI est un exemple d'utilisation de GraphicusGUI distribué sur la page web de l'unité. Consultez le fichier *readme.txt* distribué avec TestGraphicusGUI pour les détails de compilation.

# 3 SOMMAIRE DES ACTIVITÉS

**Lundi**

- Tutorat en grand groupe : présentation et séminaire (présence obligatoire).
- Support au laboratoire : dépannage à l'installation Visual Studio et Qt (optionnel).
- Support équipe Teams : dépannage à l'installation Visual Studio et Qt (optionnel).
- Étude et travaux.

**Mardi**

- Support au laboratoire.
- Formation à la pratique procédurale.
- Étude et travaux.

**Mercredi**

- Support au laboratoire.
- Évaluation formative.
- Étude et travaux.

**Jeudi**

- Support au laboratoire.
- Étude et travaux.

**Jeudi**

- Remise des livrables.
- Consultation préexamen.
- Évaluation sommative théorique.

# 4 PRODUCTIONS À REMETTRE

- Les livrables sont réalisés en équipe de 2. La même note sera attribuée à chaque membre de l'équipe.
- L'identification des membres de l'équipe doit être faite avant le mardi à 16h30. Passé cette date limite, les personnes seules ou dans des équipes ne respectant pas les contraintes de formation d'équipe peuvent être affectées à une équipe par la personne enseignante responsable. Le cas échéant, cette assignation est sans appel.
- La date limite du dépôt des livrables est le vendredi à 08h30 (pas 20h30!). Tout retard entraine une pénalité immédiate de 20% et de 20% par jour supplémentaire.
- La remise des livrables se fait uniquement avec le moyen annoncé sur le site de session. Toute remise de livrables, complète ou partielle, faite autrement que par ce moyen sera ignorée.
- Le non-respect des directives et des contraintes, comme le nom d'un répertoire ou d'un fichier, peut entrainer des pénalités.
- Seules les collaborations intraéquipe sont permises. Cependant, vous devez résoudre la problématique de façon individuelle pour être en mesure de réussir les évaluations.
- Les productions soumises à l'évaluation doivent être originales pour chaque équipe, sinon l'évaluation sera pénalisée.

## Rapport et livrables associés

Les livrables sont un rapport en format PDF, le code C++ et le *fichier pro* que vous avez produits en réponse à la problématique. La partie principale de votre rapport ne devrait pas dépasser 5 ou 6 pages, en incluant les diagrammes, et comporte uniquement les éléments suivants :

- Un diagramme de cas d'utilisation de Graphicus-03.
- Un diagramme de classes de Graphicus-03. Ce diagramme doit évidemment inclure la classe GraphicusGUI, mais **ne mettez pas** ses méthodes et ses attributs, ni ses classes mères ou autres classes de Qt dans votre diagramme.
- Un diagramme de séquence de Graphicus-03 montrant les interactions entre l'usager, l'interface usager (GraphicusGUI) et le canevas pour un scénario typique.
- Un plan de tests de Graphicus-03 et les résultats de 2 tests significatifs.

— Une réponse à la question suivante : Quels concepts de programmation orientée-objet permettent d'associer les actions de l'usager posées dans GraphicusGUI et le code spécifique à votre application ? Expliquez.

## Dépôt électronique

**Veuillez limiter la taille de votre dépôt** en vous assurant de déposer uniquement que ce qui est demandé dans ce qui suit.

Votre dépôt est un fichier d'archive (zip) qui contient les 2 éléments suivants :

— une version PDF de votre rapport ;
— un répertoire nommé Graphicus-03 qui contient uniquement votre code source C++ pour la résolution de la problématique, votre *fichier pro* et la librairie GraphicusGUI. À partir de cela, votre solution à la problématique est recompilée pour être testée. **Assurez-vous que tout compile correctement à partir de votre fichier pro**.

# 5 ÉVALUATIONS

## Utilisation de l'intelligence artificielle générative

Dans le cadre de cette unité d'APP, afin d'assurer un apprentissage réussi des compétences visées, incluant la recherche documentaire, l'analyse, la synthèse et la rédaction, **l'usage des IAG n'est pas permis pour toutes les activités de formation et d'évaluation.**
**L'utilisation des IAG lors des évaluations sommatives et finales n'est pas permise.**

Certains détails des évaluations sont donnés ici, mais leur synthèse est dans le guide de référence sur la page web de l'unité et le barème des cotes est retrouvé sur le site de session.

## 5.1 Rapport et livrables associés

L'évaluation du rapport porte sur les compétences dans la description des activités pédagogiques du guide de référence et elle est directement liée aux livrables demandés à la section 4. La pondération de l'évaluation du rapport est donnée au tableau 5.1. La qualité

Tableau 5.1 – Sommaire de l'évaluation du rapport et des livrables associés

| Élément | GIF242 |
|---|---|
| Diagramme de cas d'utilisation | 5 |
| Diagramme de classes | 5 |
| Diagramme de séquence | 5 |
| Plan de tests et résultats | 5 |
| Réponse à la question | 5 |
| Fonctionnement | 15 |
| Implémentation C++ | 10 |
| Total | 50 |

de la communication technique ne n'est pas évaluée de façon sommative, mais si votre rapport est fautif sur ce plan et de la présentation, il vous sera retourné et vous devrez le reprendre pour être noté.

## 5.2 Évaluation sommative et évaluation finale

L'évaluation sommative et l'évaluation finale sont un examen écrit sans documentation qui porte sur tous les éléments de compétences de l'APP.

## 5.3 Qualités de l'ingénieur

La grille d'indicateurs utilisée aux fins de l'évaluation est donnée au tableau 5.2.

Tableau 5.2 – Grille d'indicateurs utilisée pour les évaluations

| Libellé | Qualité | Insuffisant (N0) | Passable (seuil) (N1) | Bien (N2) | Très bien (cible) (N3) | Excellent (N4) |
|---|---|---|---|---|---|---|
| Notions du C++ | Q01.1 | ... ne démontre pas une compréhension des notions du C++. | ... démontre une compréhension insuffisante des notions du C++. | ... démontre une compréhension minimale des notions du C++. | ... démontre une bonne compréhension des notions du C++. | ... démontre une excellente compréhension des notions du C++. |
| Utilisation du C++ | Q01.2 | ... ne démontre pas la capacité d'utiliser les notions du C++. | ... est en mesure d'utiliser de façon insuffisante les notions du C++. | ... est en mesure d'utiliser minimalement les notions du C++. | ... est en mesure d'utiliser adéquatement les notions du C++. | ... est en mesure d'utiliser adéquatement efficacement les notions du C++. |
| Notions de structures de données | Q01.1 | ... ne démontre pas une compréhension des structures de données. | ... démontre une compréhension insuffisante des structures de données. | ... démontre une compréhension minimale des structures de données. | ... démontre une bonne compréhension des structures de données. | ... démontre une excellente compréhension des structures de données. |
| Utilisation des structures de données | Q01.2 | ... ne démontre pas la capacité d'utiliser les structures de données. | ... est en mesure d'utiliser de façon insuffisante les structures de données. | ... est en mesure d'utiliser minimalement les structures de données. | ... est en mesure d'utiliser adéquatement les structures de données. | ... est en mesure d'utiliser adéquatement et efficacement les structures de données. |
| Valider la solution | Q04.5 | ... ne démontre pas la capacité de valider une solution. | ... valide inadéquatement ou de manière non pertinente que la solution répond aux exigences. | ... valide minimalement que la solution répond aux exigences. | ... valide que la solution répond aux exigences, sauf dans certains cas non critiques. | ... valide complètement que la solution répond aux exigences. |
| Utiliser les techniques et outils sélectionnés selon les protocoles établis | Q05.2 | ... ne démontre pas la capacité d'utiliser les techniques et outils spécifiés. | ... ne démontre pas suffisamment la capacité d'utiliser les techniques et outils spécifiés. | ... connait et est en mesure d'utiliser avec assistance mineure les techniques et outils spécifiés. | ... connait et est en mesure d'utiliser les techniques et outils spécifiés. | ... connait et est en mesure d'utiliser efficacement les techniques et outils spécifiés. |
| Bonnes pratiques de programmation | Q01.2 | ... ne démontre pas la capacité à appliquer les bonnes pratiques de programmation. | ... démontre de la difficulté à appliquer les bonnes pratiques de programmation. | ... sait appliquer la majorité des bonnes pratiques de programmation. | ... sait appliquer efficacement la majorité des bonnes pratiques de programmation. | ... sait appliquer efficacement les bonnes pratiques de programmation. |

# 6 ÉNONCÉ DE LA PROBLÉMATIQUE

## GRAPHICUS – partie III

Suite à l'étape de réalisation du prototype Graphicus-02, l'étape suivante est l'intégration du code de la gestion du canevas de Graphicus-02 avec le prototype d'interface graphique nommé GraphicusGUI. Cette interface est montrée à la figure suivante :

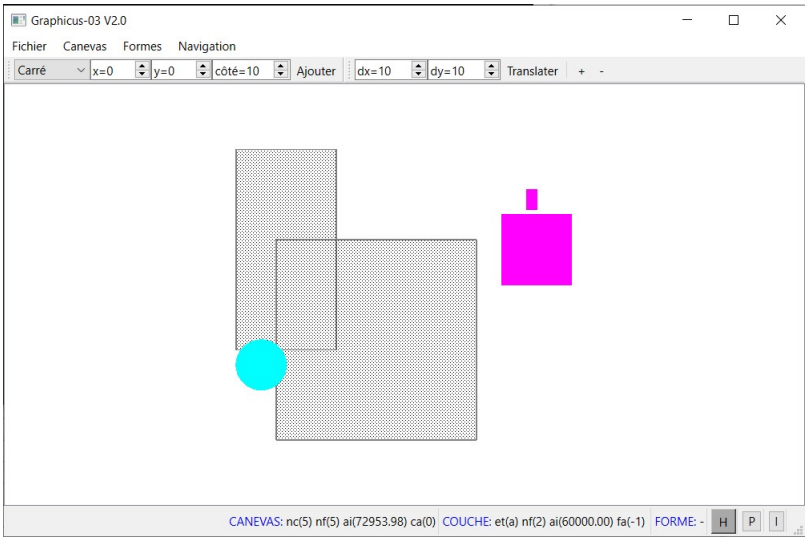


FIGURE 6.1 – L'interface graphique GraphicusGUI.

Pendant que votre équipe réalisait un prototype pour valider la conception objet du canevas, une autre équipe réalisait un prototype simplifié d'interface graphique basé sur une conception objet. Il est maintenant temps d'intégrer les deux parties, le canevas et GraphicusGUI, pour créer Graphicus-03. Évidemment, c'est votre équipe qui est mandatée de la tâche d'implémeter les fonctionnalités de GraphicusGUI pour avoir un prototype fonctionnel qui utilise un canevas.

Les spécifications du canevas, et donc aussi des couches, des formes et du vecteur, sont essentiellement les mêmes que dans Graphicus-02. Cependant, certaines spécifications ont changé afin que l'usager ait une plus grande liberté et d'autres ont changé à cause de l'utilisation de l'interface graphique. Évidemment, ces changements pourraient amener des adaptations aux développements qui ont été faits lors de Graphicus-02. À haut niveau, les changements principaux sont les suivants et tous les détails sont donnés à l'annexe A.

— C'est l'interface graphique qui mène maintenant le bal et il n'y a plus de classe de tests qui gère les tests et les interactions avec l'usager. Les commandes données au

canevas ne sont plus contrôlées par la classe de tests, mais bien par l'usager à partir de GraphicusGUI.

— Le vecteur est maintenant basé sur les modèles (*templates*) et il doit utiliser la surcharge des opérateurs. Selon les besoins de l'application, il est donc maintenant possible d'avoir un vecteur contenant tout type de données, pointeur ou non, et non plus uniquement des pointeurs de formes.
— La couche et le canevas sont maintenant basés sur un vecteur. La nouvelle classe de vecteur est donc maintenant utilisée à deux endroits : dans le canevas, pour gérer l'ensemble des couches, et dans la couche, pour gérer l'ensemble des formes.
— Les sorties qui sont liées aux différentes méthodes `afficher` des classes de Graphicus-02 sont changées afin de respecter les spécifications de GraphicusGUI.

Graphicus-03 est réalisé avec l'aide de la librairie GraphicusGUI, de votre nouvelle classe de vecteur et en redéfinissant les méthodes virtuelles de GraphicusGUI. En redéfinissant les méthodes virtuelles, vous pouvez interagir avec votre canevas en accord avec les actions de l'usager et ensuite ajuster ce qui est affiché graphiquement dans GraphicusGUI.

Il existe plusieurs manières de résoudre la problématique, mais on vous suggère les étapes suivantes afin de minimiser les problèmes :

**Étape 1 :** Réaliser la classe de vecteur avec les modèles et la surcharge d'opérateurs. Il vous est fortement suggéré de faire un projet séparé et indépendant de la problématique pour réaliser et tester votre classe. Cela limite les sources potentielles de problèmes et permet de se concentrer uniquement sur le vecteur. C'est une bonne idée de faire vos tests initiaux avec un vecteur d'entiers ou de nombres réels. La lecture de l'annexe C sur la compilation avec les modèles peut s'avérer utile. Une fois votre classe de vecteur complètement fonctionnelle et totalement validée, elle est alors prête à être intégrée dans un autre projet, comme Graphicus-03.

**Étape 2 :** Connaitre et comprendre la librairie GraphicusGUI et les outils dont vous avez besoin. Essentiellement, il s'agit de lire le guide de programmation de GraphicusGUI à l'annexe B et de comprendre, compiler et exécuter l'exemple d'utilisation qui est fourni. Avant de réaliser votre prototype complet de Graphicus-03, faites vos propres expérimentations avec la librairie afin de bien comprendre son fonctionnement.

**Étape 3 :** Réaliser Graphicus-03 en redéfinissant les méthodes virtuelles de GraphicusGUI et en intégrant vos nouvelles classes de vecteur, de canevas et de couche.

# 7 PRATIQUE PROCÉDURALE

## 7.1 EXERCICES

### P.E1 Structures de données

— Qu'est-ce qu'une pile ?
— Qu'est-ce qu'une file (ou une queue) ?
— Qu'est-ce qu'une liste chainée ?
— Comment peut-on les implémenter ?

### P.E2 Surcharge d'opérateurs

— Qu'est-ce que la surcharge d'opérateur ?
— À quoi sert-elle ?
— Quand doit-on s'en servir ?

### P.E3 Surcharge d'opérateurs : exercice

Réalisez les opérateurs d'insertion (<<), d'addition (+), de test d'égalité (==), d'assignation (=) ainsi que le constructeur de copie (*copy constructor*) pour la classe suivante qui permet d'implémenter des nombres complexes.

```
class Complexe {
  double reel;
  double imaginaire;
public:
  Complexe(const Complexe &);
  Complexe(double r = 0, double i = 0);
};
```

### P.E4 Opérateur : méthode ou fonction ?

Pourquoi est-ce que l'opérateur d'addition peut-il autant être une fonction membre de la classe `Complexe` qu'une fonction régulière alors que les opérateurs d'insertion (<<) et d'extraction (>>) doivent absolument être des fonctions régulières ?

### P.E5 Modèles

— Qu'est-ce qu'un modèle ?
— À quoi sert-il ?
— Quand doit-on s'en servir ?

### P.E6 Modèles : exemple de fonctions

Convertissez le segment code C++ suivant afin qu'il utilise des modèles. Identifiez les opérateurs qui ont besoin d'être surchargés.

```
void permutation( int * const, int * const );

void triBulle( int *tableau, const int taille ) {
  for (int passage=0;passage<taille-1;passage++)
    for (int j=0; j<taille-1 ; j++ )
      if(tableau[j] > tableau[j+1])
        permutation(&tableau[j], &tableau[j+1]);
}

void permutation( int * const ptr1, int * const ptr2 ) {
  int maintien = *ptr1;
  *ptr1 = *ptr2;
  *ptr2 = maintien;
}
```

### P.E7 Modèles : exemple de classe

Réécrivez la définition de classe suivante pour qu'elle utilise les modèles. Le paramètre du modèle détermine le type des éléments du vecteur. Conceptuellement, quel serait un algorithme général pour les opérateurs == et =.

```
class Pile {
  friend ostream &operator<< ( ostream &, const Pile & );
  friend istream &operator>> ( istream &, Pile & );
public:
  Pile ( int laCapacite = 10 );
  Pile( const Pile & );
  ~Pile();
  int getTaille() const;
  bool push ( int val );
  bool pop ( int & val );
  Pile &operator=( const Pile & );
  bool operator==( const Pile & ) const;
private:
  int taille, capacite;
  int *ptr;
};
```

# 8 PRATIQUE EN LABORATOIRE

### But des activités

Le premier but de ces activités est d'offrir du support technique au laboratoire pour l'implémentation de votre solution à la problématique. Le deuxième but est de vous permettre de consulter les personnes ressources présentes au sujet de la modélisation UML et de l'analyse du logiciel Graphicus-03.

# A SPÉCIFICATIONS MODIFIÉES DE GRAPHICUS-02

Les spécifications qui sont changées par rapport à Graphicus-02 sont les suivantes. Elles reprennent aussi ce qui a aussi été dit dans l'énoncé de la problématique. Il est à noter que de nouveaux comportements sont décrits dans le guide de la librairie GraphicusGUI (voir annexe B), car ils relèvent de comportements de l'interface et non pas du canevas lui-même.

Évidemment, selon votre solution à Graphicus-02, les changements mentionnés ici et à l'annexe B pourraient amener d'autres adaptations aux développements qui ont été faits lors de Graphicus-02.

## A.1 Classe de vecteur

La classe de vecteur doit maintenant être basée sur les modèles (*templates*). C'est donc maintenant le programmeur qui utilise la classe vecteur qui décide de ce qui est stocké dans le vecteur : des nombres entiers ou réels, des chaines de caractères, des pointeurs de formes géométriques, une voiture ou tout autre type de données, pointeur ou non.

Une des conséquences de cette généralisation est qu'on ne peut pas présumer qu'un élément retiré du vecteur doit être déalloué (`delete`) ou qu'un vecteur doit déallouer chaque item stocké quand il est détruit. La responsabilité de la déallocation des items dans le vecteur, si elle est nécessaire, n'appartient plus au vecteur.

La classe de vecteur doit aussi utiliser la surcharge des opérateurs. Minimalement, les opérateurs suivants doivent être surchargés et utilisés.

**Opérateur `[]` :** pour accéder à un élément du vecteur.

```
cout << vecteur[i]; // Écrit un des items du vecteur à l'écran.
vecteur[i] = x;     // Change la valeur d'un des items du vecteur.
```

**Opérateur `+=` :** pour ajouter un item à la fin du vecteur.

```
vecteur += donnee; // Ajoute un item à la fin de vecteur.
```

**Opérateur d'insertion (`<<`) :** pour écrire les items du vecteur à l'écran, dans un fichier ou une chaine de caractères, comme avec les classes `ostream`, `ofstream` et `ostringstream`. Il est à remarquer qu'il est possible d'utiliser le même opérateur dans les trois cas.

```
cout << vecteur;   // Écrit vecteur à l'écran (ostream).
fich << vecteur;   // Écrit vecteur dans un fichier (ofstream).
str << vecteur;    // Écrit vecteur dans une chaine (ostringstream).
```

**Opérateur d'extraction (`>>`) :** pour lire les items du clavier, d'un fichier ou d'une chaine de caractères, comme avec `istream`, `ifstream` et `istringstream`. Il est à remarquer qu'il est possible d'utiliser le même opérateur dans les trois cas.

```
cin >> vecteur;    // Lit vecteur du clavier (istream).
fich >> vecteur;   // Lit vecteur d'un fichier (ifstream).
str >> vecteur;    // Lit vecteur d'une chaine (istringstream).
```

**Opérateurs ++ et -- :** pour passer à l'item suivant ou précédent du vecteur, en admettant qu'il a un *item courant*. Ce sont les opérateurs de pré-incrémentation et de pré-décrémentation qui sont surchargés.

```
++vecteur;       // Passe à l'item suivant de vecteur.
--vecteur;       // Passe à l'item précédent de vecteur.
```

## A.2 Canevas

Un canevas doit être basée sur un vecteur de couches et chaque couche doit être basée sur un vecteur de formes. Le nombre de couches ou de formes n'est plus fixé et dépend des besoins de l'usager. L'usager peut donc ajouter et retirer une couche ainsi que retirer ou ajouter des formes. Il est possible qu'une couche ne contienne aucune forme. Cependant, il y a tout de même au minimum une couche même s'il n'y a pas de maximum. Lorsqu'une couche est ajoutée, elle est ajoutée à la suite de toutes les autres couches et son état est *initialisée*. La nouvelle classe de vecteur, avec modèles et surcharge d'opérateurs, est donc maintenant utilisée à deux endroits : dans le canevas et dans la couche.

## A.3 Affichage

Les sorties qui sont liées aux différentes méthodes `afficher` des classes de Graphicus-02 sont changées afin de respecter les spécifications de l'interface graphique GraphicusGUI. Le format est donné à l'annexe B.2.

# B LIBRAIRIE GraphicusGUI

C'est maintenant l'interface graphique GraphicusGUI qui mène le bal. Cette section décrit donc l'interface et le comportement désiré pour Graphicus-03.

Il est important de réaliser que GraphicusGUI est une coquille vide. Par défaut, toute action, comme un clic de bouton ou une sélection de menu, a pour effet d'appeler la méthode virtuelle associée à cette action qui affiche un message indiquant que la méthode doit être redéfinie. Toute action de l'usager ne cause donc aucune action utile automatique sur le canevas ou pour l'affichage de formes géométriques.

Pour que des actions utiles surviennent, les méthodes virtuelles de GraphicusGUI qui sont associées aux actions (boutons, menus, etc.) doivent être redéfinies afin qu'elles contiennent du code qui spécialise GraphicusGUI pour un comportement particulier. Par exemple, si l'usager ajoute un rectangle avec l'aide de l'interface, alors la méthode correspondante à cette action doit être redéfinie et contenir du code afin que cette forme soit ajoutée dans un canevas (le vôtre) et que les formes dessinées à l'écran soient rafraichies (par vous). Sans la redéfinition des méthodes, aucun canevas n'est géré et aucune forme n'est affichée ou rafraichie. Avec la redéfinition des méthodes, il est possible de *capter* les actions de l'usager dans l'interface et d'implémenter les comportements désirés en gérant votre canevas et en rafraichissant à l'écran les formes dessinées.

Les comportements désirés de Graphicus-03 sont connus, car ils correspondent essentiellement à ceux de Graphicus-02, à peu de choses près. Ces comportements sont décrits du point de vue de GraphicusGUI dans ce qui suit.

## B.1 Description

GraphicusGUI est une librairie graphique qui permet de gérer l'affichage de Graphicus. Elle est illustrée à la figure B.1, qui est une reproduction de la figure 6.1 avec des indications supplémentaires.

La figure B.1 identifie quatre zones utiles dans l'interface : la *zone de menu* (zone 1), la *zone outils* (zone 2), la *zone graphisme* (zone 3) et la *zone informations* (zone 4). Voici quelques explications à leur sujet et les comportements désirés. Il est à noter que la plupart des actions ont des raccourcis au clavier.

### Zone menu

**Menu Fichier :** Le menu a les choix suivants : Ouvrir, Sauvegarder, Sauvegarder sous et Quitter. Le comportement d'Ouvrir est de lire le contenu d'un fichier qui contient les informations d'un canevas. Le canevas qui était affiché est vidé et son contenu est remplacé par ce qui a été lu du fichier. À l'inverse, les choix Sauvegarder et Sauvegarder sous prennent le contenu actuel du canevas et écrivent sont équivalent dans le ficher. Le format des données lues ou écrites est expliqué un peu plus loin à la section B.2.

**Menu Canevas :** Le menu a les choix suivants : réinitialiser le canevas, ajouter une couche, retirer la couche active et afficher en format texte. Il est à noter qu'une couche est ajoutée à la suite de toutes les autres couches. Aussi, la fonctionnalité afficher en format texte

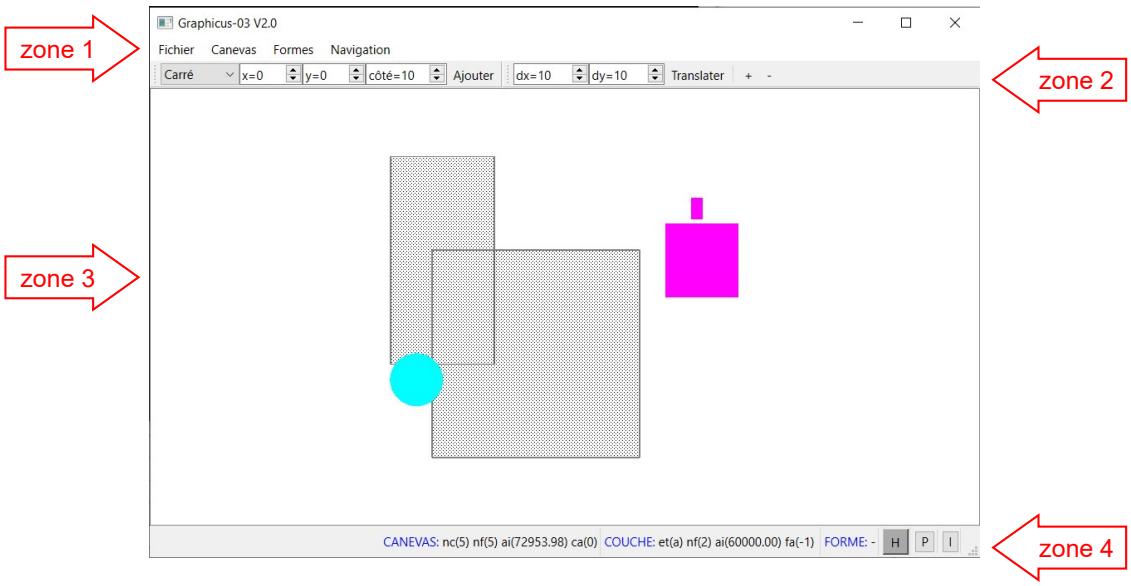


FIGURE B.1 – L'interface graphique GraphicusGUI avec l'identification des quatre zones.

permet d'afficher un dialogue avec le contenu actuel du canevas dans le format décrit à la section B.2. Cette dernière fonctionnalité est complètement gérée par GraphicusGUI et elle est surtout utile pour le débogage.

**Menu Formes :** Le menu a un seul choix : retirer la forme active. Une forme peut être retirée, s'il y a une forme active dans la couche active.

**Menu Navigation :** Ce menu permet de poser des actions afin de naviguer de couche en couche. S'il y a une couche d'active, il permet aussi de naviguer de forme en forme dans cette couche. De couche en couche ou de forme en forme, les actions sont d'aller au premier, au précédent, au suivant et au dernier.

Il est important de noter que de se déplacer à une couche a pour effet d'activer cette couche. Par exemple, de passer de la couche 2 à la couche 3 active automatiquement la couche 3. Il en est de même pour le déplacement dans les formes d'une couche.

### Zone outils

Il y a deux outils dans cette zone : Forme et Couche. Un outil peut être montré ou caché avec un clic droit de la souris dans la zone et ils peuvent aussi être déplacés et réorganisés.

**Outil de forme :** C'est avec cet outil qu'il est possible d'ajouter une forme. Dans l'interface, il suffit de choisir le type de forme avec la liste déroulante, de fixer les valeurs et de cliquer sur le bouton Ajouter. La forme doit être ajoutée dans la couche active du canevas, s'il y en a une.

**Outil de couche :** C'est avec cet outil qu'il est possible de translater la couche active du canevas, s'il y en a une. Dans l'interface, il suffit de fixer les valeurs de dx et dy et ensuite de cliquer le bouton Translater. La couche active du canevas doit être translatée en conséquence. Il est à noter que cet outil comporte deux autres boutons qui offrent la possibilité d'ajouter une couche (bouton +) et de retirer la couche active (bouton -).

De cliquer sur un de ces boutons est équivalent à choisir la même action dans le menu correspondant. Évidemment, le canevas doit être ajusté en conséquence de l'action.

### Zone graphisme

C'est dans cette zone que sont dessinées graphiquement les formes. Il est à noter que rien ne se dessine automatiquement. Pour qu'un canevas soit dessiné à l'écran, il faut invoquer la méthode de GraphicusGUI à cet effet à chaque fois. C'est au programmeur de l'invoquer lorsque nécessaire avec les bonnes données pour rafraichir la zone de graphisme en accord avec le contenu actuel du canevas. Les données fournies à la méthode pour dessiner sont dans le format texte expliqué à la section B.2. Il y a aussi possibilité d'effacer d'un seul coup la zone de graphisme.

Dans la zone de graphisme, les coordonnées sont en pixels. Les coordonnées sont donc des entiers. En regardant la figure B.1, les coordonnées en $x$ vont de la gauche vers la droite et les coordonnées en y vont du haut vers le bas. Il faut s'habituer.

Les couleurs des couches sont automatisées. Il y a 8 couleurs possibles, comme montré au tableau B.1. La couleur d'une couche dépend de sa position : la couche 0 a la couleur 0, la couche 1 a la couleur 1, et ainsi de suite jusqu'à la couche 7. Pour les 8 premières couches, c'est simple. S'il y a plus de 8 couches, alors la couleur d'une couche est exprimée par sa position modulo 8. Ainsi, la couche 12 est verte, car 12 modulo 8 donne 4, et 4, c'est le vert.

Tableau B.1 – Couleur des couches.

| Position | Couleur |
|---|---|
| 0 | noir |
| 1 | cyan |
| 2 | rouge |
| 3 | magenta |
| 4 | vert |
| 5 | jaune |
| 6 | bleu |
| 7 | gris |

### Zone informations

La zone d'informations est divisée en quatre parties : canevas, couche, forme et boutons.

**Partie canevas :** Les informations dans cette partie sont des informations sur le canevas. Dans l'ordre, elles sont le nombre total de couches (nc), le nombre total de formes (nf), l'aire totale du canevas (ai) et la couche active (ca). À la figure B.1, le canevas a donc 5 couches, 5 formes, une aire de 72953,98 pixels$^2$ et la couche active est la couche 0.

**Partie couche :** Les informations dans cette partie sont des informations sur la couche active. Dans l'ordre, elles sont l'état (et), le nombre de formes (nf), l'aire totale (ai) et la forme active (fa). À la figure B.1, la couche active est à l'état a (active), elle est composée de 2 formes, elle a une aire de 60000 pixels$^2$ et la forme active est -1 (pas de forme active).

**Partie forme :** Les informations dans cette partie sont des informations sur la forme active. Dans l'ordre ce sont les coordonnées (xy), l'aire (ai) et les informations complémentaires (info). À la figure B.1, il n'y a pas de forme active, donc un tiret (-) est affiché. Plus de détails sur les informations complémentaires sont donnés dans la section B.4.

**Partie boutons :** Il y a trois boutons dans cette zone. Le bouton H pour activer et désactiver le surlignage, le bouton P pour activer et désactiver le mode pile et le bouton I pour activer et désactiver l'affichage des les zones d'informations (canevas, couche et forme). À la figure B.1 seul le mode surlignage est actif. Le mode surlignage est complètement géré par GraphicusGUI. Si le surlignage est actif, les formes de la couche active sont dessinées avec un fond à (petits) pois et avec un contour gris foncé. À la figure B.1, on peut voir que la couche active, la couche 0, est surlignée. Lorsque le mode pile est activé, l'ordre des couches doit être inversé dans le canevas et l'affichage de formes doit être rafraichi. Lorsque le mode pile est désactivé, l'ordre des couches doit être réinversé dans le canevas et l'affichage de formes doit être rafraichi. Il est à noter que seules les couches sont inversées et non pas les formes. Le bouton I permet de montrer ou de cacher les parties canevas, couche et forme et il est complètement géré par GraphicusGUI.

Les informations dans les parties canevas, couche et forme ne sont pas ajustées automatiquement. Pour rafraichir les données dans ces parties, le programmeur doit invoquer la méthode associée à la zone d'information avec les bonnes données chaque fois qu'il le juge approprié.

## B.2 Format texte

Un format a été défini pour interagir avec la zone de graphisme de GraphicusGUI. C'est aussi ce format qui est attendu lors de la sauvegarde dans un fichier (menu Sauvegarder et Sauvegarder sous...) ou lorsqu'un fichier est chargé (menu Ouvrir). Le format est le suivant :

**`L e` :** Une ligne qui débute par un `L` indique le début d'une nouvelle couche. Le caractère qui suit (`e`) indique l'état de la couche. Les valeurs possibles de `e` sont `a` pour active, `i` pour initialisée et `x` pour inactive.

**`R x y l h` :** Une ligne qui débute par un `R` indique une forme de type rectangle. Les quatre valeurs qui suivent sont des valeurs entières qui représentent respectivement, la coordonnée en x, la coordonnée en y, la largeur et la hauteur du rectangle.

**`K x y c` :** Une ligne qui débute par un `K` indique une forme de type carré. Les trois valeurs qui suivent sont des valeurs entières qui représentent respectivement, la coordonnée en x, la coordonnée en y et la longueur du côté du carré.

**`C x y r` :** Une ligne qui débute par un `C` indique une forme de type cercle. Les trois valeurs qui suivent sont des valeurs entières qui représentent respectivement, la coordonnée en x, la coordonnée en y et le rayon du cercle.

Dans l'exemple de la figure B.2, on peut remarquer qu'il y a 5 couches. Les couches 2 et 4 sont vides et inactives. La couche 0 est inactive et elle a deux formes, un rectangle et un carré. La couche 1 est active et elle contient un cercle. La couche 3 est inactive et elle contient deux formes, un rectangle et un carré.

```
L a
R 10  10 100 200
K 50 100 200
L x
C 10 200 50
L i
L x
R 300 50 10 20
K 275 75 70
L i
```

FIGURE B.2 – Exemple de canevas en format texte.

## B.3 Compilation

GraphicusGUI est basée sur la librairie graphique Qt. Afin de facilement obtenir un projet Visual Studio , un fichier `pro` est utilisé avec l'utilitaire qmake. L'utilitaire qmake fait partie de la distribution de Qt et utilise l'information contenue dans le fichier `pro` pour générer le projet Visual Studio , un peu comme make avec un makefile. Avec qmake, il est plus facile d'éviter les ennuis de configuration et d'avoir un bon fichier de projet Visual Studio qui utilise Qt.

Ce qui suit présume que vous avez installé Visual Studio et que vous validé votre installation de Qt et que qmake est accessible sur la ligne de commande. Si vous n'avez pas fait et validé ces étapes, consultez la section 2 et consultez aussi la section *Manuels, logiciels* sur le site web de session pour des directives et conseils d'installation de Visual Studio et Qt.

Un exemple d'utilisation de GraphicusGUI est distribué sur la page web de l'unité. Il vous est suggéré de le compiler et de l'exécuter afin de bien comprendre le fonctionnement de cette librairie. S'il le faut, modifiez cet exemple afin de vous assurer d'avoir bien compris.

Voici la procédure pour générer un projet pour Visual Studio .

— Ajuster le contenu d'un fichier `pro` existant ou encore créer un fichier `pro` de zéro. Entre autres, assurez-vous que les répertoires sont les bons. L'exemple d'utilisation de GraphicusGUI est distribué avec un exemple de fichier `pro`.
— Démarrer l'invite de commandes pour les outils de Visual Studio . Typiquement, c'est dans la section de Visual Studio du menu *Démarrer* de Windows. L'invite de commandes à utiliser est appelée ***x64 Native Tools Command Prompt for VS 2022***. Choisissez la bonne ! Il y en a typiquement plusieurs dont les noms sont très semblables, comme le montre la figure B.3.
— Dans l'invite de commandes, aller dans le répertoire de votre fichier `pro`.
— Tapez `qmake` suivi du nom de votre fichier `pro`, par exemple :
`qmake graphicus-03.pro`
Si tout s'est bien passé, un projet pour Visual Studio a été généré, c'est un fichier avec l'extension `vcxproj`.

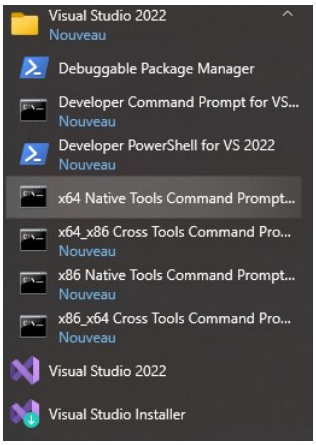


FIGURE B.3 – Choix de l'invite de commande.

Le projet ainsi créé peut être doublecliqué pour démarrer automatiquement Visual Studio et ensuite lancer la compilation et l'exécution. Le projet peut aussi être ajouté dans une solution Visual Studio existante, ce qui est très pratique dans le cas où vous voulez gérer plusieurs projets.

## B.4 Interface avec la classe

Cette section donne les détails de l'interface avec GraphicusGUI incluant les méthodes associées aux actions et à la mise à jour des zones d'informations et de graphisme. Une description de chacune des actions est donnée dans la section B.1.

### Constructeurs

**`GraphicusGUI::GraphicusGUI(const char* titre);`**

Ce constructeur permet de créer une instance de GraphicusGUI avec le titre de la fenêtre spécifié par la chaine de caractères passée en paramètre. Si aucun titre n'est fourni, alors le titre par défaut est `"GraphicusGUI"`.

### Actions de fichiers

Les méthodes suivantes sont invoquées lorsque l'usager sélectionne l'action correspondante dans l'interface. Ces méthodes doivent être redéfinies afin de leur donner le comportement désiré.

**`bool GraphicusGUI::ouvrirFichier(const char* nom);`**

Cette méthode est invoquée lorsque le menu Ouvrir est sélectionné et qu'un fichier a été choisi avec succès par l'usager. Le nom du fichier choisi est passé en paramètre dans la chaine de caractères `nom`.

**`bool GraphicusGUI::sauvegarderFichier(const char* nom);`**

Cette méthode est invoquée lorsque le menu Sauvegarder ou Sauvegarder sous est sélectionné et qu'un nom de fichier a été choisi avec succès par l'usager. Le nom du fichier choisi est passé en paramètre dans la chaine de caractères `nom`.

## Actions de canevas, de couches et de formes

Les méthodes suivantes sont invoquées lorsque l'usager sélectionne l'action correspondante dans l'interface. Ces méthodes doivent être redéfinies afin de leur donner le comportement désiré.

```
void GraphicusGUI::reinitialiserCanevas();
```
Cette méthode est invoquée lorsque le canevas doit être réinitialisé.

```
void GraphicusGUI::coucheAjouter();
```
Cette méthode est invoquée lorsqu'une couche doit être ajoutée dans le canevas.

```
void GraphicusGUI::coucheRetirer();
```
Cette méthode est invoquée lorsque la couche active du canevas doit être retirée.

```
void GraphicusGUI::coucheTranslater(int deltaX, int deltaY);
```
Cette méthode est invoquée lorsque la couche active du canevas doit être translatée. Les valeurs de translation en $x$ et $y$ sont passées en paramètres.

```
void GraphicusGUI::ajouterCercle(int x, int y, int rayon);
```
Cette méthode est invoquée lorsqu'un cercle doit être ajouté à la couche active. Les valeurs spécifiques du cercle à créer sont passées en paramètres.

```
void GraphicusGUI::ajouterRectangle(int x,int y,int long_x,int long_y);
```
Cette méthode est invoquée lorsqu'un rectangle doit être ajouté à la couche active. Les valeurs spécifiques du rectangle à créer sont passées en paramètres.

```
void GraphicusGUI::ajouterCarre(int x, int y, int cote);
```
Cette méthode est invoquée lorsqu'un carré doit être ajouté à la couche active. Les valeurs spécifiques du carré à créer sont passées en paramètres.

```
void GraphicusGUI::retirerForme();
```
Cette méthode est invoquée lorsque la forme courante de la couche active doit être retirée de la couche.

```
void GraphicusGUI::modePileChange(bool mode);
```
Cette méthode est invoquée lorsque le mode pile change. Le paramètre `mode` indique si le mode pile a été activé (`true`) ou désactivé (`false`).

## Actions de navigation

Les méthodes suivantes sont invoquées lorsque l'usager sélectionne l'action correspondante dans l'interface. Ces méthodes doivent être redéfinies afin de leur donner le comportement désiré.

```
void GraphicusGUI::couchePremiere();
```
Cette méthode est invoquée lorsque l'usager désire se déplacer à la première couche du canevas et la rendre active.

```
void GraphicusGUI::coucheSuivante();
```
Cette méthode est invoquée lorsque l'usager désire se déplacer à la couche précédente du canevas et la rendre active.

```
void GraphicusGUI::coucheSuivante();
```
Cette méthode est invoquée lorsque l'usager désire se déplacer à la couche suivante du canevas et la rendre active.

**`void GraphicusGUI::coucheDerniere();`**
Cette méthode est invoquée lorsque l'usager désire se déplacer à la dernière couche du canevas et la rendre active.

**`void GraphicusGUI::formePremiere();`**
Cette méthode est invoquée lorsque l'usager désire se déplacer à la première forme de la couche active du canevas.

**`void GraphicusGUI::formePrecedente();`**
Cette méthode est invoquée lorsque l'usager désire se déplacer à la forme précédente de la couche active du canevas.

**`void GraphicusGUI::formeSuivante();`**
Cette méthode est invoquée lorsque l'usager désire se déplacer à la forme suivante de la couche active du canevas.

**`void GraphicusGUI::formeDerniere();`**
Cette méthode est invoquée lorsque l'usager désire se déplacer à la dernière forme de la couche active du canevas.

### Mise à jour de la zone de graphisme

Les méthodes de cette section peuvent être invoquées par votre code au moment opportun afin que le comportement de l'application soit celui désiré. Ces méthodes ne doivent pas être redéfinies ; elles ne sont d'ailleurs pas virtuelles.

**`void GraphicusGUI::effacer();`**
Cette méthode permet d'effacer le contenu de la zone de graphisme.

**`void GraphicusGUI::dessiner(const char * texte);`**
Cette méthode permet de dessiner dans la zone de graphisme. Ce qui était affiché est effacé et ce qui sera dessiné dépend du contenu de la chaine de caractères qui est passé en paramètre. Le contenu de la chaine de caractères doit respecter le format qui est décrit à la section B.2. L'usage de la classe standard `ostringstream` peut être avantageux ici, si l'opérateur d'insertion (`<<`) est surchargé pour la classe de canevas.

### Mise à jour de la zone d'informations

Les informations dans la zone d'informations ne sont pas mises à jour automatiquement. Lorsque les données affichées doivent être rafraichies, les méthodes de cette section peuvent être invoquées au moment opportun dans votre code afin que le comportement de l'application soit celui désiré. Ces méthodes ne doivent pas être redéfinies ; elles ne sont d'ailleurs pas virtuelles.

**`void GraphicusGUI::effacerInformations();`**
Cette méthode permet de réinitialiser la zone d'informations.

**`void GraphicusGUI::setInformations(Informations info);`**
Cette méthode permet d'ajuster les informations contenues dans la zone d'informations. Les informations sont fournies à GraphicusGUI avec le paramètre `info` en donnant des valeurs aux membres en accord avec l'état actuel du canevas. La structure `Informations` est décrite ci-dessous.

```
struct Informations
{
  // Canevas
```

```
  int nbCouches;       // Nombre total de couches
  int nbFormesCanevas; // Nombre total de formes
  int coucheActive;    // la couche active, si < 0, pas de couche active
  double aireCanevas;  // Aire totale
  // Pour la couche active du canevas, s'il y en a une
  int nbFormesCouche;  // Nombre de formes de la couche
  char etatCouche[20]; // État de la couche en format libre
  double aireCouche;   // Aire totale de la couche
  int formeActive;     // la forme active, si < 0, pas de forme active
  // Pour la forme active de la couche active, s'il y en a une
  int coordX, coordY;  // Coordonnées de la forme
  double aireForme;    // Aire de la forme
  char informationForme[50]; // Informations format libre: rayon, etc.
};
```

## Utilitaires

Les méthodes de cette section peuvent être utilisées, selon les besoins de l'application, au moment opportun dans votre code afin que le comportement de l'application soit celui désiré. Ces méthodes ne doivent pas être redéfinies ; elles ne sont d'ailleurs pas virtuelles.

**`bool GraphicusGUI::getModePile();`**
Cette méthode permet d'obtenir l'état actuel du mode pile. La valeur `true` est retournée quand le mode pile est activé, sinon elle retourne `false`.

**`bool GraphicusGUI::getSurlignage();`**
Cette méthode permet d'obtenir l'état actuel du surlignage. La valeur `true` est retournée quand le surlignage est activé, sinon elle retourne `false`.

**`void GraphicusGUI::message(const char *);`**
Cette méthode permet d'afficher un message informatif à l'usager dans la zone d'informations. Le message reste affiché pendant trois secondes.

**`void GraphicusGUI::messageErreur(const char *);`**
Cette méthode permet d'afficher un message d'erreur à l'usager dans la zone d'informations. Le message d'erreur est de couleur rouge et il reste affiché pendant trois secondes.

**`void GraphicusGUI::afficher(ostream& os);`**
Cette méthode écrit dans le flot spécifié en paramètre le contenu de la zone de graphisme dans le format de la section B.2. Cette méthode est essentiellement utilisée à des fins de débogage.

# C COMPILATION AVEC MODÈLES

La compilation avec les modèles (*templates*) en C++ peut s'avérer difficile les premières fois. La plupart des compilateurs utilisent la *technique d'inclusion*, plutôt qu'une *technique séparée* pour les modèles. Si les compilateurs utilisaient tous la technique séparée, les choses seraient plus simples pour les programmeurs. Malheureusement, souvent, c'est la technique d'inclusion qui est utilisée.

Les pages qui suivent font un survol rapide de la manière de structurer votre code C++ utilisant les modèles avec la technique d'inclusion. Tous les détails ne sont pas expliqués, seulement ceux nécessaires pour structurer correctement votre code sont abordés. Si vous désirez plus de détails sur ces différentes techniques de compilation des modèles en C++, consultez le web.

## C.1 Quelques explications

Lorsque vient le temps de compiler avec les modèles, la plupart des compilateurs, comme g++ et Visual Studio , dérogent du modèle de compilation séparée comme elle est faite habituellement avec des fichiers d'entête (.h) et de code (.cpp). C'est une des conséquences de la technique d'inclusion.

Pour compiler avec les modèles, la règle à suivre est simple : il faut inclure le code d'une classe (tout le code !) ou d'une fonction qui utilise les modèles dans le fichier source où elle est utilisée. Il ne faut pas faire la compilation séparée avec cette classe (ou fonction), comme on le fait habituellement. Le fichier de code n'est pas compilé séparément ; il ne se retrouve donc pas, par exemple, explicitement dans un projet Visual Studio .

Il y a plusieurs manières de respecter cette règle. Plutôt que de l'expliquer en plusieurs paragraphes, les deux exemples qui suivent montrent comment l'appliquer.

## C.2 Méthode 1

Voici un exemple qui illustre une première façon de structurer votre code avec modèles afin de compiler avec le modèle d'inclusion.

**Fichier principal : main.cpp**

```
#include "test.h"

int main()
{
   Test< int > tmp(1);
   return 0;
}
```

**Fichier d'entête : test.h**

```
template <class T>
class Test
{
public:
   Test( T d );
private:
   T donnee;
};

template < class T >
Test< T >::Test( T d )
{
   donnee = d;
}
```

À remarquer que la définition de la classe et le code de ses méthodes sont tous dans le fichier d'entête (`test.h`) et qu'il n'y a pas de fichier de code (`test.cpp`) correspondant. Tout est dans le fichier d'entête !

Le fichier d'entête étant inclus dans le fichier principal (`main.cpp`), tout le code de la classe avec modèle est donc inclus et compilé en compilant le fichier principal. La compilation de cet exemple se fait simplement en compilant le fichier principal (`main.cpp`). Remarquez que le fichier `test.cpp` n'est pas compilé, car il n'existe pas !

## C.3 Méthode 2

Cette méthode est une variante de la méthode 1. Elle tente de respecter un peu plus la compilation séparée, en ce qui concerne la classe avec modèle, en ayant quand même un fichier d'entête (.h) et de code (.cpp) pour cette classe.

On peut préférer cette deuxième méthode, car avec très peu de modifications on peut passer de la technique d'inclusion à la technique séparée. Ceci peut survenir s'il y a un changement de compilateur ou encore lorsque l'on amène du code sur une autre plateforme.

**Fichier principal : main.cpp**

```
#include "test.h"

int main()
{
   Test< int > tmp(1);
   return 0;
}
```

**Fichier d'entête : test.h**

```
template <class T>
class Test
{
public:
   Test( T d );
private:
   T donnee;
};

#include "test.cpp"
```

**Fichier de code : test.cpp**

```
template < class T >
Test< T >::Test( T d )
{
   donnee = d;
}
```

Remarquez les choses suivantes :

- Le fichier principal n'a pas changé par rapport à la méthode 1.
- À la dernière ligne du fichier d'entête (`test.h`), le fichier de code (`test.cpp`) est inclus. Ceci est logiquement équivalent au fichier d'entête de la méthode 1, sauf que le code des méthodes de la classe est dans le fichier de code (`test.cpp`).
- Le fichier `test.h` n'est pas inclus dans `test.cpp`, comme c'est le cas habituellement. À l'inverse de ce qui est fait habituellement, c'est le fichier d'entête qui inclut le fichier de code !

La compilation a lieu de la même manière qu'avec la méthode 1 et pour les mêmes raisons. On compile uniquement le fichier principal (`main.cpp`). Le fichier de code `test.cpp` n'est pas compilé séparément ; il ne se retrouve donc pas, par exemple, explicitement dans un projet Visual Studio .

# LISTE DES RÉFÉRENCES

[1] B. Charroux, A. Osmani, et Y. Thierry-Mieg, *Langage UML 2 : Pratique de la modélisation*, 3e édition, séries Collection Synthex. Pearson Education, 2010.

[2] S. Rao, *Sams Teach Yourself C++ in One Hour a Day*, 8e édition. Sams publishing, Pearson Education, 2017.