Programmation orientée objet

LIV 2025-2026

Travaux pratiques 5

Site du cours : `https://defelice.up8.site/poo.html`
Les exercices marqués de (@) sont optionnels.
Un fichier écrit en langage C++ se termine conventionnellement par .cpp.
Une commande de compilation est `g++ fichier_source1.cpp fichier_source2.cpp fichier_source3.cpp`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-Wextra` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-std=c++20` pour compiler selon la norme c++20
- `-g -fsanitize=address` pour que l'exécutable puisse detecter certaines erreurs de mémoire (adresse inacessible, fuite mémoire) (l'inonveniant et que l'executable est plus lent, -g pour avoir plus de précision).
- `-g` pour ajouter des informations de débugage, utile avec debug (executable plus lent).

Exemple : `g++ -Wall -Wextra fichier1.cpp -o monprogramme`

Dans tous les exercices de ce TP (et des tp suivants) il faut si possible penser à déclarer les méthodes constantes.

Exercice 1. *Figure*

Fabriquer la classe `Figure` qui modèlise des figures du plan. Ici une figure du plan est simplement une liste ordonnée de `Point2D` (de l'exercice séparation du TP précédent).La classe `Figure` possède deux méthodes `affiche` et `rotate` qui utilisent les méthodes `affiche` et `rotate` de la classe `Point2D`. Il ne doit pas y avoir de limite maximum de points à la taille d'une figure. Il faut écrire la classe `figure` dans deux fichiers `figure.hpp` et `figure.cpp`. Pensez à bien gérer la mémoire, pour vérifiez utiliser l'option `-fsanitize=address`.

```
#include"figure.hpp"
#include"point2d.hpp"
int main(){
   Figure o; // figure vide
   o.ajouter(Point2D{0.,1.});
   o.ajouter(Point2D{1.,2.});
   o.rotate(); // rotation de 90 d à chaque point de la figure o
   o.affiche(); // affiche (textuellement) la liste des points
}
```

Exercice 2. *Figure et Point2D amélioration*

Cet exercice consiste simplement à améliorer (en parallèle) les classes `Figure` et `Point2D` afin de permettre aux figures d'effectuer d'autres opérations. Améliorations demandées :

- ajouter à `Figure` la méthode d'objet `translater(Point2D a)` qui translate chaque point selon le vecteur $\overrightarrow{O\texttt{a}}$.
  Où O est le point de coordonnées $(0, 0)$
- ajouter à `Figure` la méthode d'objet `dilater(double d)` qui multiplie les coordonnées de chaque point par `d`.
- ajouter à la classe `Figure` la méthode d'objet `tourner(double a)` qui effectue une rotation de `a` degrés à chaque point de la figure autour du point de coordonnées $(0, 0)$.

Exercice 3. *Compte appels*

Construire la classe `CptAppel` possédant un constructeur sans paramètre et une méthode (publique) de classe `static int compterAppel(void)` renvoyant le nombre d'utilisations du constructeur. Écrire la classe dans deux fichiers `cptappel.hpp` et `cptappel.cpp`.

Exemple :

```
int main(){
   CptAppel a; // premier appel
   {
      CptAppel b; // second appel
      CptAppel c; // troisième appel
      int n=a.compterAppel(); // n contient 3 (car 3 appels jusqu'à maintenant)
      CptAppel d; // quatrième appel
   }
   CptAppel e; // cinquième appel
   CptAppel::compterAppel(); // renvoie 5
}
```

Exercice 4. *Tableau*

Implanter une classe `Tableau6` qui implante des tableaux d'entiers de 6 cases. Voici un exemple d'utilisation de la classe.

```
Tableau6 tab1;// creer un tableau de taille  6
              // initialise chaque case avec la valeur 0

tab1[4]=2; // place 2 dans la cinquième case du tableau
tab1[1]; // renvoie 0

const Tableau6 tabc{tab1};  // tableau constant recopié à partir de tab1

tabc[4];   // renvoie 2
// tabc[4]=3; // erreur! tabc est constant

tabc==tab1; // renvoie 1 (ou vrai)

Tableau6 tab2;
tabc==tab2; // renvoie 0 (ou faux)
tab2=tabc;
tabc!=tab2; // renvoie 0 (ou faux)
// tabc=tab1; // erreur ! tabc est constant
```

Sans ajouter de nouveau constructeur à la classe, construire (dans `main`) un objet constant `c` de la classe `Tableau6` qui contient `2,3,5,7,11,13`. Sachant qu'on ne pourra pas utiliser une instruction comme `c[5]=3`.

Exercice 5. *dilatation*

Écrire une classe `VecteurPlan` qui permet les opérations suivantes (Il est inutile de définir plus d'un constructeur).

```
VecteurPlan p{2.2,-5.1}; // créer un couple de float
VecteurPlan p2=p*1.1; //  p2 est le couple (2.2*1.1,-5.1*1.1)
VecteurPlan p3=1.5*p; //  p3 est le couple (2.2*1.5,-5.1*1.5)
```

Exercice 6. *Somme*

Écrire une classe `Relies` qui possède au moins les méthodes suivantes.

- un constructeur qui produit un objet mémorisant une valeur entière.
- `set(int a)` qui modifie cette valeur.
- `int get()` qui renvoie la valeur.
- `int somme()` qui renvoie la somme de toutes les valeurs entières mémorisées dans les objets actuels.

Indice : La méthode `somme` est-elle forcément liée à un objet ?

Précision :

- A priori la méthode somme() renvoie une valeur différente avant ou après qu'un objet soit détruit.
- Pensez à prendre en compte les objets construits (ou non construit) à travers des opération de : construction par copie, destruction, affectation.