Programmation orientée objet

LIV 2024-2025

Travaux Pratiques - Polymorphisme

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`

Exercice 1. *Véhicule*

1. Construire une hérarchie de classe de `Vehicule` , `Voiture`, `Moto`. Chaque véhicule doit possèder le nom du constructeur (Honda,Peugeot ...). La classe `Vehicule` doit posséder la méthode virtuelle `presenteToi` qui affiche un message indiquant si c'est une moto ou une voiture et son constructeur, (Exemple : ≪Je suis une voiture construite par .. ≫).
2. Remplir un tableau par des adresses d'objet de type `Vehicule` (soit `Moto` soit `Voiture`) appeler sur chacun la méthode `presenteToi`.

Exercice 2. *Pion*

On souhaite construire deux types de pion, des soldats qui peuvent attaquer d'autres pions en leur enlevant leur points de vie (6 points de vie) et des soigneurs qui peuvent soigner un autre pion en leur redonnant des points de vie (5 points de vie). Chaque pion possède 30 points de vie à sa construction. Utiliser l'héritage pour concevoir vos classes. Exemple :

```
Soldat g1; // 30 points de vie
Soigneur s1; // 30 points de vie
g1.vie(); // renvoie 30
g1.attaque(s1) ; // enlève 6 points de vie
s1.vie(); // renvoie 24
s1.soigne(s1); // regagne 5 points de vie
s1.vie(); // renvoie 29
//s1.attaque(s1); // erreur s1 n'est pas un soldat

Soldat g2;
g1.attaque(g2);
g2.vie(); // renvoie 24
s1.soigne(g2);
g2.vie(); // renvoie 29
```

Exercice 3. *Nombre*

On peut représenter un nombre, soit avec une fraction (deux entiers), soit avec un type `long` (donc forcément entier), soit avec un type `double`.

On souhaite écrire les classes `Entier, Float, Fraction` qui héritent de la classe `Nombre`. Implanter ce qu'il faut pour trier un tableau contenant des pointeurs sur des objets de type `Nombre` que ces objets soient `Entier` `Float` ou `Fraction`.

Exemple :

```
Nombre* T[4];
T[0]=new Entier{6};
T[1]=new Fraction{3,2}; // fraction 3/2   (= 1.5)
T[2]=new Float{345.556};
T[3]=new Fraction{2,3}; // fraction 2/3    (~ 0.6666)

void triNombre(3,T);

// ici dans l'ordre il doit y avoir Fraction{2,3} Fraction{3,2} Entier{6} Float{345.556}
// dans le tableau T
```