TP C++ : Ecologie

Les Nettoyeurs sont des petits robots dont le but est de recycler les déchets. Avides de propreté, ces petites bêtes se reproduisent rapidement et interagissent avec leur environnement pour en tirer de l'énergie.
Ils vont sillonner les plages, ramasser des objets, les consommer, et utiliser leur énergie pour se multiplier.
Le cycle de vie classique de ces nettoyeurs est :

- Je cherche des objets,
- J'absorbe leur énergie,
- Je communique avec les autres nettoyeurs pour augmenter de catégorie,
- Je crée une Base quand je le peux,
- La Base crée de nouveaux nettoyeurs,
- etc...

Nous allons suivre dans ce TP la vie de deux petits nettoyeurs. : Lilo et Stitch.

<u>Commençons par créer nos nettoyeurs.</u>

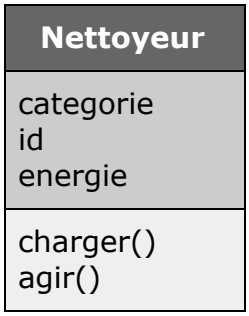
Chaque nettoyeur est représenté par :

- Une catégorie (un entier)
- Un identifiant (chaine)
- Un niveau d'énergie (un entier)

Chaque nettoyeur peut :

- Se charger (augmenter son énergie)
- Agir (diminuer son énergie)

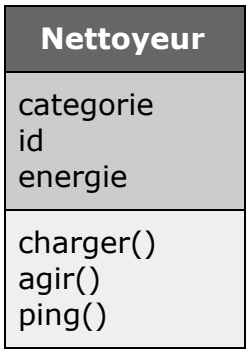
1. Créez la classe Nettoyeur.



★ Attention à faire les bonnes inclusions, les bonnes portées, et à bien séparer vos fichiers.

2. Créez une méthode ping()

❑ Cette méthode permettra à chaque nettoyeur d'afficher son identifiant, sa catégorie, et son énergie.



3. Surchargez le constructeur

❑ Ceci permettra de pouvoir choisir la valeur des attributs lors de la création d'un nettoyeur.

★ Par défaut, les nettoyeurs sont de catégorie 1, ont 100 d'énergie, et un identifiant “Wall-e”.

Bon, assez codé, construisons nos deux vedettes.
Instanciez deux nettoyeurs : Stitch, et Lilo.

★ Attention, passez les bons paramètres à votre constructeur pour créer vos deux nettoyeurs.

❖ Vérifiez par des pings que chacun est bien qui il doit être.

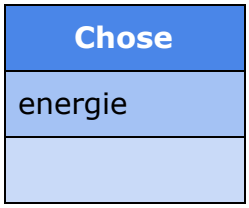
4. Codez les méthodes charger et agir.

❑ Celles-ci agissent sur l’énergie du nettoyeur (respectivement en l’augmentant et en la diminuant).

C’est bien, mais si il suffisait qu’ils se chargent tous seuls dans leur coin, Stitch et Lilo auraient la vie facile. En réalité, ils sont obligés de consommer des choses pour en tirer de l’énergie.

5. Créons donc une classe Chose

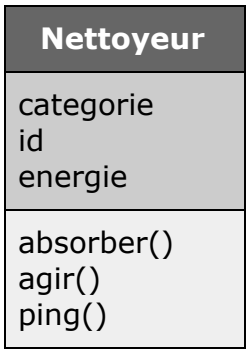
❑ Celle-ci ne sera représentée que par un niveau d’énergie (un entier).



Ok, l'idée maintenant est de faire en sorte que l'énergie de nos nettoyeurs n'augmente qu'en consommant des Choses.

6. Modifions la méthode charger.

- ❏ Renommons la en absorber(), et faisons en sorte qu'elle prenne en paramètre une chose. C'est l'énergie de cette chose qui sera absorbée.



Problème : l'énergie d'une chose est privée. Or, la seule raison d'être d'une Chose est d'avoir une énergie à disposition. Rajoutons donc une méthode getEnergie() publique, qui nous renverra l'énergie de chaque Chose.

- ❏ Créez une chose(énergie au choix) et vérifiez que Stitch et Lilo peuvent l'absorber.

- ★ Oui mais, dans notre cas, un objet absorbé peut toujours l'être. Modifions un peu notre code pour qu'un objet absorbé ne soit plus disponible. Comment? Eh bien réduisons à 0 l'énergie d'un objet quand il est absorbé. (Changeons la méthode getEnergie en syphoner(). Syphoner renverra l'énergie de la chose, et mettra cette énergie à 0.). Attention, la chose existe toujours, mais elle n'a plus d'énergie.

Exemple :
Canette (én.150) est absorbé par Stitch (én.100). Canette a maintenant 0 en énergie, et Stitch 250.

❖ Testez votre code avec quelques appels à ces méthodes.

Ok, on peut maintenant se nourrir de choses.

7. Un nettoyeur est une chose !

★ Attention, plus complexe : On peut considérer que les nettoyeurs sont des choses. Un nettoyeur doit donc pouvoir absorber un autre nettoyeur.

❑ Ce qu'il faut se dire, c'est qu'un nettoyeur est un type particulier de chose, et peut donc hériter de Chose.

★ Attention ! Il va falloir adapter les constructeurs, les portées, et les attributs en conséquence ! (recherchez "liste d'initialisation" pour vous aider avec les constructeurs)

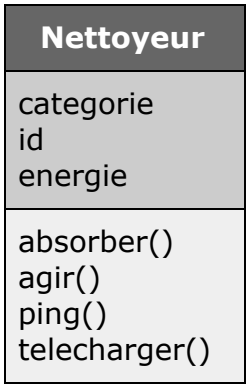
Après cette étape, Stitch devrait pouvoir absorber Lilo.

Mais bon, les nettoyeurs ont cet avantage de réfléchir au bien commun avant le leur, ils vont donc se focaliser sur d'autres choses plutôt que sur des nettoyeurs.

Nos nettoyeurs peuvent donc se nourrir. Ils vont maintenant devoir apprendre les uns des autres pour pouvoir évoluer.

8. Codons la méthode telecharger

- ❑ Cette méthode permet à un nettoyeur de monter de catégorie en téléchargeant les données d'un autre nettoyeur.



- ★ Attention, il y a certaines règles à suivre:

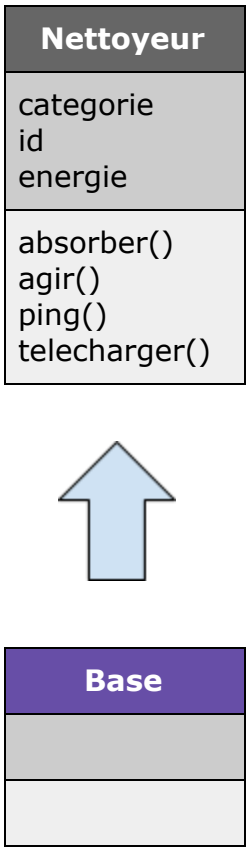
- ★ Télécharger coûte de l'énergie. Le nettoyeur qui télécharge doit dépenser la moitié de l'énergie de sa cible * sa catégorie.
  Exemple : Si Lilo (cat 3 energie 200) veut absorber Stitch (cat 2 energie 100), elle devra dépenser 2*50=100 énergie. Lilo passera donc en catégorie 4, et aura 100 d'énergie restante).

- ★ Télécharger ne modifie pas le nettoyeur ciblé.

❏ N’oublions pas de tester !

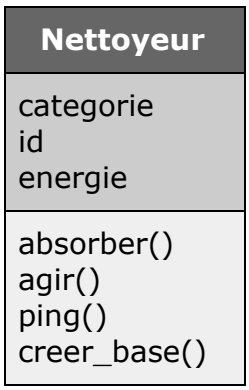
Nos nettoyeurs ne sont toujours pas en mesure de se répliquer…. Il est temps d’y remédier.
Lorsqu’un nettoyeur a accumulé assez de données, il peut construire une Base, qui sera en charge de créer de multiples nettoyeurs.

9. Construisons notre classe Base



★ Attention ! Une Base est bien sûr un nettoyeur ! (conseil : forward dec dans Nettoyeur.h de la classe Base, et un include dans Nettoyeur .cpp)

10. Permettons à un nettoyeur de pouvoir créer une Base.



- ★ Attention ! Seul un nettoyeur de catégorie 5 ou plus qui n'a pas de base peut construire une Base.
- ★ De plus, les Nettoyeurs ne peuvent avoir qu'une seule Base. Il va donc falloir que chaque nettoyeur sache qui est sa base. Par défaut, les nettoyeurs n'ont pas de base (Stitch et Lilo n'en ont donc pas. Celà dit, une fois qu'un nettoyeur crée une base, elle devient sa base, et chaque nettoyeur créé par cette base lui sera affilié. Il faut donc rajouter à chaque nettoyeur cette information. (Un pointeur serait le bienvenu).
- ★ Enfin, créer une base est coûteux en énergie (1000/categorie). Vérifiez bien qu'un nettoyeur peut créer sa base avant de la créer….

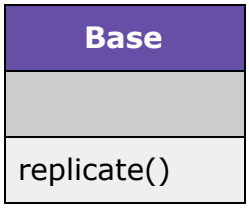
11. Ajoutons un constructeur à notre base

- ❑ Ceci permettra que son id contienne celle de son créateur (une base créée par Lilo s'appellera BaseLilo).
- ★ Note: Une base n'a pas de base.

Il ne reste plus qu'à pouvoir créer des nettoyeurs, et ça c'est le boulot de la base.

12. Créons la méthode replicate

- ❏ Celle-ci renvoie un Nettoyeur tout neuf.
- ★ Attention, créer un nettoyeur est coûteux. (100 d'énergie).



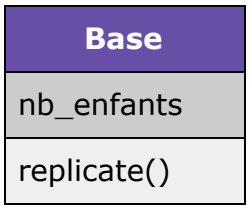
Il va donc falloir que la base absorbe quelques choses si besoin.

- ★ Le nom de chaque nouveau nettoyeur sera le nom de sa base suivi par un numéro de série. Le numéro de série est donc le nombre de nettoyeur créés par cette base.

Exemple : BaseLilo crée trois nettoyeurs, ils se nommeront BaseLilo1,BaseLilo2 et BaseLilo3.

13. Ajoutons un numéro de série incrémental

- ❏ Cell-ci nous permettra d'identifier les nouveaux nettoyeurs.



- ★ Le numéro est donc propre à chaque base, ce qui veut dire qu'il ne peut pas être static.
- ★ Il doit donc être implémenté comme attribut de Base, et bien s'incrémenter à chaque fois qu'elle crée un nettoyeur.

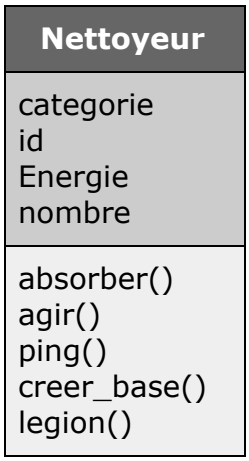
Note : la fonction to_string existe, mais suivant l'IDE et le compilateur que vous utilisez, elle peut bugger (sur codeblocks par exemple…) au cas où, voici une réécriture de la fonction, à mettre dans votre .cpp:

```
#include <sstream>
template <typename T>
std::string to_string(T value)
{
	std::ostringstream os ;
	os << value ;
	return os.str() ;
}
```

Enfin, on aimerait connaitre le nombre total de nettoyeurs créés depuis le début.

14. Créons donc un attribut static

- Celui-ci sera dans notre classe Nettoyeur, et nous renseignera sur le nombre total de nettoyeurs.

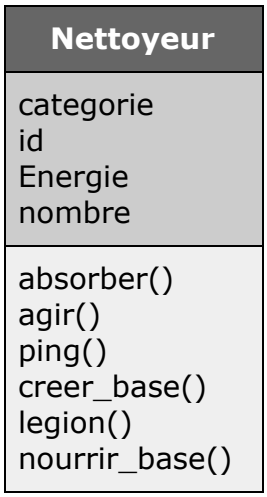


- Cette information peut être obtenue en demandant à n'importe quel nettoyeur leur nombre (méthode int legion()). Attention aux constructeurs et destructeurs !

La base est cependant peu mobile, elle a besoin de ses nettoyeurs pour se nourrir. Il faut donc permettre aux nettoyeurs de pouvoir donner de leur énergie à leur base.

16. Codons la méthode nourrir_base()

Cette méthode de chaque nettoyeur lui permet de transférer une partie de son énergie à sa base.



Pour les plus avancés, on va maintenant créer un monde (une succession de choses.)

17. Créez un vector de choses

- ❑ Ce vector sera à remplir de choses. (voir vector, push_back(), pop_back()...)

- ★ A chaque fois qu'un nettoyeur va vouloir absorber une chose, on la retire des choses disponibles sur le monde. Attention aux cas où le monde est vide !

Enfin, chaque base doit pouvoir savoir quels nettoyeurs elle a créé.

18. Ajoutons un attribut vector

- ❏ Ce vector de pointeurs sur nettoyeurs, appelé cohorte, contiendra l'adresse de tous les nettoyeurs créés par cette base.

19. Ajoutons une méthode ping_cohorte()

Cette méthode nous permettra de pinger tous les nettoyeurs créés par la base.

Pour aller plus loin :

Nous avons la base de nos nettoyeurs, ils peuvent consommer et se reproduire. Libre à vous d'implémenter un monde autour de ça.

Idées :

Faire une carte avec un visuel des nettoyeurs. On commence avec un nettoyeur et beaucoup d'objets. Le nettoyeur doit aller vers les objets pour les consommer. Faire cela lui coûte de l'énergie. Une fois qu'il en a accumulé suffisamment, il peut créer une base immobile. Celle-ci va utiliser l'énergie que ses nettoyeurs lui apportent pour continuer à produire d'autres nettoyeurs. Ajouter une IA de recherche des objets sur le monde, et observer les nettoyeurs se multiplier et se débarrasser des déchets.

- ★ PS: Cela constitue une base tout à fait acceptable pour un éventuel projet C++...