L'héritage

Un moyen d'organisation

Adieu les grosses classes !

- Les classes énormes sont difficiles à gérer
- Elles contiennent quantité d'outils inutiles (pour un besoin précis)
- La modification d'une spécification pour un utilisateur X oblige tous les utilisateurs à recompiler, même s'ils n'utilisaient pas la fonctionnalité modifiée

Modulariser les classes

- Depuis longtemps, pour de nombreuses raisons, les programmes sont modulaires
- Les classes doivent l'être aussi, ce qui leur permet d'être :
  - plus petites,
  - assemblables selon les besoins,
  - réutilisables séparément,
  - individuellement corrigibles.

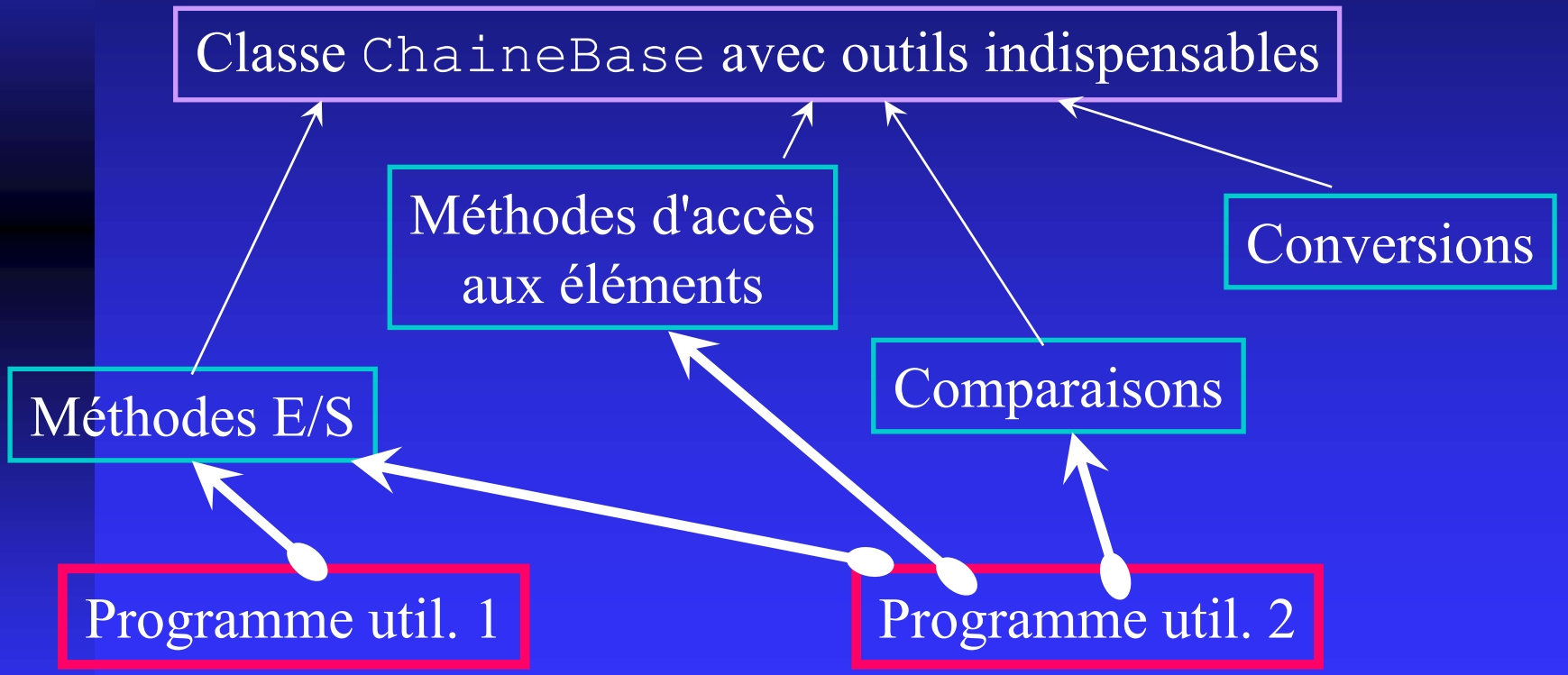
Un exemple : la classe chaîne de caractères

- a besoin de méthodes pour :
  - l'initialisation, la terminaison,
  - les entrées, les sorties,
  - gérer la taille (agrandir, diminuer, interroger,...)
  - accéder aux caractères (opérateurs [], insertion,...)
  - comparer (`<`, `<=`, `==`, `...`)
  - convertir
  - corriger, etc.

Aucun programme n'aura besoin de toutes ces fonctionnalités

- il sera toujours possible de changer le corps des méthodes, à condition qu'elles n'aient pas toutes été stockées dans le même `.C`
- si on souhaite ajouter un nouvel outil de conversion, par exemple, ce serait dommage d'avoir à recompiler tous les programmes qui utiliseraient la classe **`chaîne`**, même s'ils ne font aucune conversion !

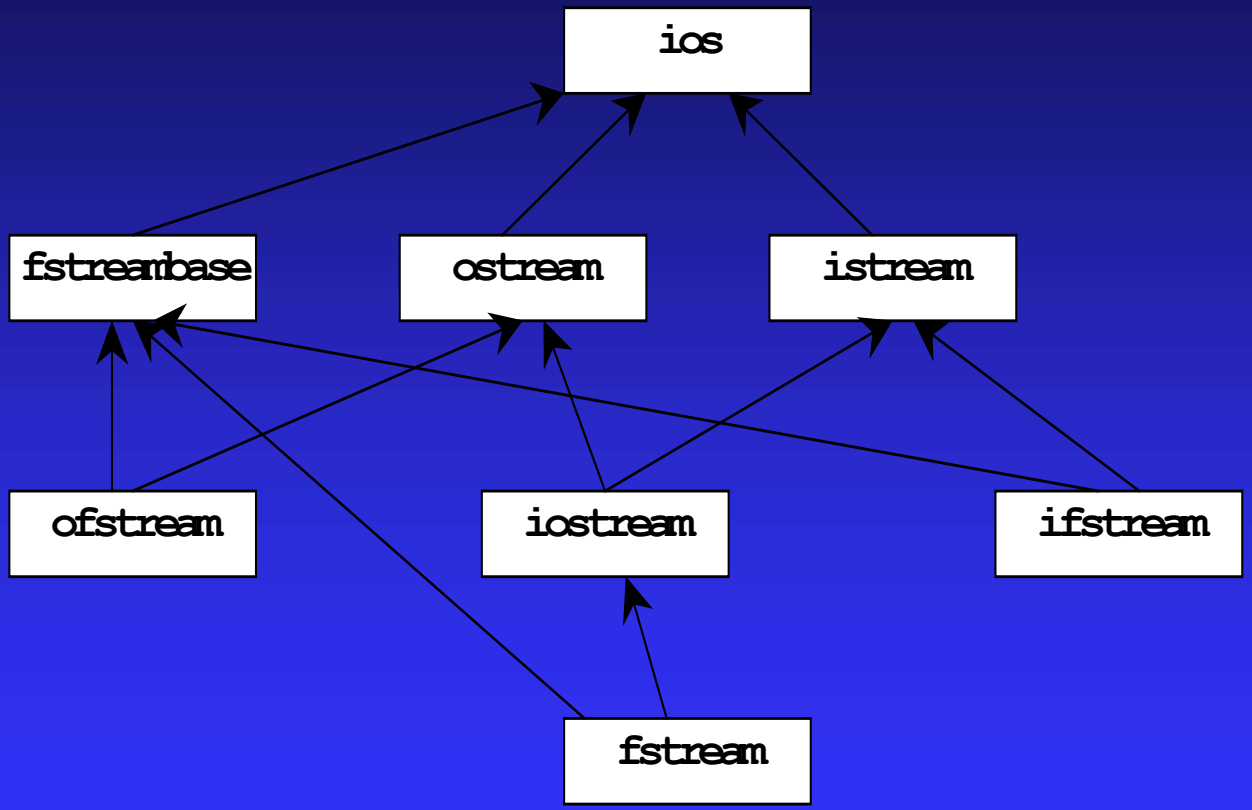
Organisation de classes dérivées, une par groupe de méthodes.



Résultat

- Un travail beaucoup plus professionnel
- La possibilité de modifier la spécification d'une méthode en ne recompilant que les programmes des utilisateurs concernés
- fichiers nombreux et risque de dispersion
- éventuels besoins de communication entre les classes dérivées et leurs classes de base

Exemple : iostream



Les classes abstraites

- Ce sont des classes qui ne serviront pas à produire des objets, mais uniquement à servir de classes de base pour des classes concrètes
- On ne pourra pas déclarer d'objets de cette classe, ni de tableaux d'objets de cette classe
- On pourra déclarer des pointeurs.

Définir des interfaces

- Lorsque les méthodes que l'on réalise sont destinées à être appelées par un outil extérieur figé, leur nom est standardisé
  - Exemple C++: une classe propose une méthode indispensable et des méthodes virtuelles pures => pour disposer de la méthode, il est nécessaire de créer une classe dérivée et il est obligatoire de surcharger les méthodes pures sous peine de rester abstrait…
  - Exemple Java: **`prepareImage`** et **`imageUpdate`** (interface **`ImageObserver`**), **`sort`** et **`compareTo`**(interface **`Comparable`**)

Utilisation de classe abstraite

- Si **`ClassExemple`** est une classe abstraite :
  - **`ClassExemple x;`** est impossible
  - **`ClassExemple t[10];`** est impossible
  - **`ClassExemple * p;`** est possible
- il n'existera pas d'objets du type **`ClassExemple`**, mais **`p`** pourra pointer sur de objets de classes dérivées. (polymorphisme)

Une classe abstraite

- contient au moins une méthode virtuelle pure, par exemple :

```
virtual int methode(...)= 0;
```

- Cette méthode devra être réalisée dans les classes dérivées.
- Toute classe dérivée d'une classe abstraite qui laisserait au moins une méthode virtuelle pure serait aussi abstraite.

Les méthodes virtuelles

- permettent de déclencher, à partir d'un pointeur sur la classe de base, une méthode de classe dérivée
- le corps d'une méthode virtuelle impure constitue une sorte de "traitement par défaut"