**Note Prépa Contrôle :**

**Héritage, méthode virtuelle & polymorphisme**

***Prof :*** *Les classe qui hérite d’une classe sont des classes dérivées*

Une méthode virtuelle est une méthode susceptible d’être redéfinie ou spécialisée dans une sous-classe dérivée d’une classe initiant l’existence de cette méthode

Une méthode virtuelle est dite polymorphe si celle-ci a pour objectif d’être redéfinis dans une ou plusieurs classes dérivées.

Une méthode virtuelle pure est une méthode virtuelle qui en plus déclarée = 0 éventuellement dans le fichier h

**Exemple :** virtual double getVolume() = 0;

***Note :*** *si la définition de la méthode est déportée dans un fichier .cpp, il n’est pas nécessaire de repréciser que la méthode est virtuelle dans le fichier .cpp. (cad de rajouter la syntaxe virtual) car elle est par défaut maintenant virtuelle et même pour toutes les sous-classe dérivées*

Une classe abstraite est une classe ayant au moins une méthode virtuelle pure. Elle n’est pas instanciable.

Elle est utile pour permettre aux classes dérivées de cette classe d’implémenter la méthode virtuelle pure pour la rendre opérationnelle lors de l’instanciation des objets des sous-classe L’appelant syntaxiquement.

***Prof :*** *Aucun objet d’une classe abstraite peut exécuter les méthodes virtuelles pures car ces méthodes ne disposent pas de fonctionnalités ou d’opérations susceptible d’exécuter une tache*

Remarque : Une classe abstraite définit les méthodes que devront implémenter les sous-classe (elle peut donc être généralement la classe mère)

Une classe concrète est donc une classe dont toutes les méthodes sont implémentées (cad aucune méthode pourrait être virtuelle pure)

**Autres :**

**Résolution statique :** C’est le paramètre passé à l’intérieur de la fonction qui prime

**Résolution dynamique :** Ici c’est le type de l’objet qui appel la méthode qui prime et pour ce fait il faut :

Que la méthode appelé soit déclarée comme virtuelle

Que cet objet soit une référence ou une adresse

**Les Templates :** sont des éléments permettant de définir des modèles de structures de données, de classes et de fonctions, sur des objets dont le type est paramétrable.

Prof : A la différence d’un tableau, une list est dynamique et peut contenir n’importe quel type d’objet

Dans une list on stocke des objets et des éléments des objets

Rmq : l’iterator pointe sur une instance de la classe et non sur une classe

Chaque pointeur de type iterator connais l’adresse du suivant. C’est pourquoi on parle de list chaînée

**Conseil tp1 :**

Vous devez faire la différence entre surcharge et redéfinition

Les attributs privés doivent commencer par \_.

static : Veut dire qu’elle est visible au niveau de la classe et qu’elle est partagée dans toute la classe

**Conseil tp2 :**

**Prof :** Le but de ce tp est de connaitre les relation UML entre les différentes classes.

Dans ce tp, nous visualisons bien l’association bilatéral définis entre Individu et Voiture

**Prof :** On peut remplacer getMarque par : marque,this->marque,this->getMarque . La différence se trouve dans l’espace de recherche du compilateur

**Prof :** Privilégier les getters et les setters ; à l’intérieur duquel on manipule les attributs en question

**Prof :** On utilise la notion -> lorsqu’on est en face de l’adresse d’un objet

Relation entre classe -> clé étrangère en déclarant la classe X dans le fichier de la classeY afin que les la classeY puisse utiliser les membres de la classeX (éventuellement déclarer un pointeur de type la classe X dans Y qui sert de clé étrangère)

Savoir utiliser le nullptr

Rmq : nullptr veut dire nulle pointeur (c’est la valeur nulle du pounteur comme 0 pour l’algebre de bool)

**Conseil tp3**

Cas ou on a les cle(conteneur associatif)

La paire pair<iteroator,bool> permet de vérifier si l’insertion c’est bien réaliser(donc dans la plus part des cas des conteneurs qui n’accepte pas de doublon).

Son itérateur est différent de l’iterator de recherche

typedef Map :: iteraotor i ; // Iterator pour indiquer ou l’insertion s’est réaliser et non pour parcourir (iterateur de la pair)

pair<i,bool> testeInsertion ;

//Pour tester si l’insertion s’est bien realiser

If(testInsertion.second == true ){cout << « Insertion bien realiser »<< endl ;}else{cout<<””}

i i2=leConteneur.begin(); // Iterotor pour parcourir les éléments du conteneur (son second ne designe pas le booleen i2->second mais le second élément du conteneur)

while(i2 !=leConteneurnstancier.end()){

cout << i2->first << endl ; // S’il y avait un seul élément dans le conteneur. On ferra

// cout << \*i2 << endl ;

}

On utilise la méthode find\_if dans une liste <list> et la méthode find lorsqu’on a un map

find recherche un élément à partir de la clé d’un map

find\_if recherche un élément entre un intervalle de valeur [first, last[d’un conteneur séquentiel ou associatif (Si conteneur séquentiel liste de produit par exemple, il va retourner le produit s’il trouve sinon il retourne le dernier élément) à nous d’afficher l’élément du produit qu’on veut récupérer par une condition if suite à la recherche

//Exemple cas conteneur associatif

existe=find\_if(uneListeDeProduits.begin(),uneListeDeProduits.end(),memeReference);

if(existe==uneListeDeProduits.end()){

cout << referenceCherchee+"n'existe pas " << endl;

}

else{

cout << existe->getReference() << endl;

}

Ou même référence est le prédicat ; dès que le prédicat retourne false, la recherche n’est plus fructueuse

Il faut donc que le prédicat retourne true

La méthode for\_each (first, last, fct) a appliqué la fonction fct à chacun des éléments d’un conteneur qui sont entre les itérateurs [first, last [où fac est une procédure toujours

**Conseil tp4**

Une classe canonique est une classe constituée d’un constructeur avec paramètre, d’un constructeur de recopie, d’un destructeur et d’un operateur d’affectation.

|  |
| --- |
| MaClasse |
| MaClasse(parametres) ;  MaClasse(const MaClasse&) ;  ~MaClasse  MaClasse& operator=(const MaClasse&) |

**Contrôle 2022**

**Cours**

1. Définition d’une classe abstraite
2. Comment obtenir les objets a partir d’une classe abstraite (rappel de ce que le prof a dit)
3. Définition d’un Template
4. Comment peut-on instancier une classe Template
5. Définition d’une classe canonique
6. Que permet de faire l’héritage qui ne permet pas de faire un Template et réciproquement

**Exercice2**

Héritage-Syntaxe, Relation UML C++, maitriser les bool operator de la classe Produit par exemple

**Exercice 3**

Conteneur set , Probleme : Produit,Magasin,Panier,Achat

->Savoir utiliser un set (par exemple avec la methode insert)!

->Boucle for lorsqu’on declare un iterator…

**Td1**

Principe de virtualité important à savoir

**Td2**

***N.B : Maitriser les tds et tps , venir avec les sujets imprimer des tds et tps***

***Remarque : Dans les controles du prof, le td1 vient en question de cours***