# TD – Constructeurs

Commenter le code suivant :

class Vehicule

{

protected :

string immat ;

public:

Vehicule(string i) :

immat (i) {}

virtual void advancer() = 0;

};class Voiture : public Vehicule

{

protected:

int couleur;

public:

Voiture(string i, int c) {

immat=i;

couleur=c;

}

};

* Les classes embarquent une méthode virtuelle, il faut donc définir un destructeur virtuel.
* Le constructeur de Voiture fait un appel implicite au constructeur par défaut de Véhicule. Il faut donc soit le fournir, soit appeler explicitement le constructeur, ce que je propose :

Voiture(string i, int c) : Vehicule(i), couleur(c) {}

Comme on fournit un constructeur avec paramètres, le compilateur estime qu’il n’a plus à fournir le constructeur par défaut.

On déclare maintenant un tableau dans la fonction main() par exemple :

Voiture tab1[10] ;

Erreur du compilateur car on essaie d’instancier dix objets Voiture avec le constructeur par défaut qui n’est pas fourni. Deux alternatives :

* On fournit les constructeurs par défaut (de toute façon, il faut le faire pour respecter la forme de Coplien)
* On utilise un tableau de pointeurs, ce qui shunte cette construction implicite d’objets.

On ajoute les constructeurs par défaut aux classes Vehicule et Voiture(en utilisant éventuellement des valeurs par défaut pour les paramètres)

Considérons maintenant :

Vehicule tab2[10] ;

Erreur de compilation car la classe Vehicule est abstraite. Toute classe qui définit une méthode virtuelle pure (abstraite) est abstraite.

## Complément/retour sur la méthode abstraite :

Dans le fichier hpp

class Vehicule {

virtual void avancer() = 0 ;

} ;

Dans le .cpp

void Vehicule ::avancer() {

cout << immat   
 << " avance " << endl ;

}

Très utile pour éviter la duplication de code mais aussi pour bien préciser qu’une méthode est virtuelle pure donc abstraite que l’on utilise dans la classe fille comme cela :

void Voiture ::avancer() {

Vehicule ::avancer() ;

}