Programmation Orientée Objet
Classes

Cyril Prissette
prissette@univ-tln.fr

SeaTech

Du C au C++

Ecrire une structure **Vehicule** avec :

- vitesse : entier
- vitesse_max : entier
- poids (en tonnes) : réel, double précision

Ecrire les fonctions :

- initialiser() pour vitesse=0, vitesse_max=130, poids=1
- afficher(), au format (*vitesse*/*vitesse_max*,*poids*)
- accelerer() pour augmenter vitesse de 50 (avec limite)
- ralentir() pour réduire vitesse de 50 (avec limite à 0)

Ecrire un main() pour tester la structure et les fonctions.

★ *Ce symbole indique que c'est à vous de travailler. Ne lisez pas plus loin tant que votre programme ne marche pas.*

Correction - struct et prototypes

```
#include <iostream>
using namespace std;

struct Vehicule
{
int vitesse;
int vitesse_max;
int poids;
};

void initialiser(Vehicule& V);
void afficher(Vehicule V);
void accelerer(Vehicule& V);
void ralentir(Vehicule& V);
```

Correction - fonctions (1/2)

```
void initialiser(Vehicule& V)
{
V.vitesse = 0;
V.vitesse_max = 130;
V.poids = 1;
}

void afficher(Vehicule V)
{
        cout << "(" << V.vitesse << "/" << V.vitesse_max;
        cout << "," << V.poids << ")" << endl;
}
```

Correction - fonction (2/2)

```
void accelerer(Vehicule& V)
{
V.vitesse = V.vitesse + 50;
if (V.vitesse > V.vitesse_max)
        V.vitesse = V.vitesse_max;
}

void ralentir(Vehicule& V)
{
V.vitesse = V.vitesse - 50;
if (V.vitesse < 0)
        V.vitesse = 0;
}
```

Correction - main()

```
int main()
{
Vehicule voiture;

initialiser(voiture);

for (int i=1; i<=5; ++i)
        {
        afficher(voiture);
        accelerer(voiture);
        }
}
```

Du C au C++

En programmation non-objet, on applique des **fonctions** sur des **variables** d'un certain **type**

```
Vehicule voiture;
ralentir(voiture);
```

En programmation orientée objet (POO), on demande aux **objets**, qui appartiennent à des **classes**, de réaliser des actions en executant des **méthodes**.

```
Vehicule voiture;
voiture.ralentir();
```

Première classe - A compléter

```
class Vehicule
{
public :
        int vitesse;
        int vitesse_max;
        int poids;
        void ralentir();
};

void Vehicule::ralentir(void)  // pour voiture.ralentir();
{
vitesse = vitesse - 50;
if (vitesse < 0)
        vitesse = 0;
}
```

Première classe - A compléter

Ecrire les autres méthodes, c'est à dire initialiser(), accelerer() et afficher().
Ecrire un main() qui effectue les mêmes tâches que le main() qui avait été écrit en programmation non-objet (initialiser, puis accelerer 5 fois en affichant l'état du véhicule au fur et à mesure).
★

Encapsulation

```
class Vehicule
{
public :
        int vitesse;
        ...
```

Droit d'accès :

- **public:** ce qui suit est visible par tout le monde
- **protected:** ce qui suit est visible uniquement par les objets

Trouver un moyen d'interdire d'écrire :

```
int main()
{
Vehicule voiture;
voiture.vitesse = -17000;
}
```

★

Encapsulation

La solution repose sur ce que l'on appelle l'encapsulation.
Le mécanisme d'encapsulation consiste à déclarer en **protected** (ou **private**) les attributs pour empécher un accès direct.

L'accès est controlé par des méthodes d'interfaces.

- **accès en lecture** : accesseur ou "getter"
  Par convention, son nom commence souvent par get
  int Vehicule::get_vitesse()
- **accès en ecriture** : mutateur ou "setter"
  Son nom commence souvent par set
  void Vehicule::set_vitesse(int v)

Encapsulation

```
class Vehicule
{
protected :
        int vitesse;
        int vitesse_max;

public :
        int get_vitesse();
        void set_vitesse(int);
        //...
        void initialiser();
}
```

Encapsulation

```
int Vehicule::get_vitesse()
{
return vitesse;
}

void Vehicule::set_vitesse(int v)
{
vitesse=v;
}

void Vehicule::initialiser()
{
vitesse=0; // ou set_vitesse(0);
vitesse_max=130; // ou set_vitesse_max(130);
}
```

Encapsulation

ALERTE!

Le chef de projet demande un changement de structure interne de la classe Vehicule, utilisant un tableau de 2 entiers nommé vitesses (au pluriel) pour stocker les vitesses, au lieu des variables vitesse et vitesse_max.
Il exige que les programmes main() existants continuent à fonctionner sans nécessiter de changement.

★

Comme le chef de projet est un peu luniatique, il va sans doute revenir sur cette nouvelle lubie d'ici la fin de la journée.

Travaillez sur une copie de votre fichier, au cas où...

Encapsulation

```
class Vehicule
{
protected:
        int vitesses[2];
public:
        void initialiser();
        int get_vitesse();
        void set_vitesse(int);
        ...
};
```

Encapsulation

```
int Vehicule::get_vitesse()
{
return vitesses[0]; // au lieu de return vitesse;
}

void Vehicule::set_vitesse(int v)
{
vitesses[0]=v; // au lieu de vitesse=v;
}

// idem pour get_vitesse_max() et set_vitesse_max()
```

Encapsulation

2 possibilité pour les autres méthodes

```
// version 1 : on remplace ici aussi vitesse par vitesse[0]
void Vehicule::initialiser()
{
vitesses[0] = 0;    // au lieu de vitesse=0;
vitesses[1] = 130; // au lieu de vitesse_max=130;
}
```

OU

```
// version 2 : si on a utilise des setters, rien ne change
void Vehicule::initialiser()
{
set_vitesse(0);
set_vitesse_max(130);
}
```

Encapsulation

Le mécanisme d'encapsulation a l'avantage de masquer les détails de l'implémentation interne de la classe.

Celui qui utilisera la classe Vehicule aura juste besoin de connaitre les fonctions accelerer(), ralentir(), get_vitesse(), etc.

L'encapsulation est une "bonne pratique". Sauf indication contraire, toutes vos classes doivent être encapsulées.

L'utilisation des setters/getters à l'intérieur des méthodes de la classe n'est pas nécessaire, sauf si on veut marquer explicitement une opération dangeureuses pour laquelle on souhaite activer les contrôles.

Le chef de projet lunatique veut qu'on utilise à nouveau vitesse et vitesse_max.

Heureusement que vous lisez les petites lignes en bas des slides.

Surcharge d'opérateurs

Ecrire une méthode attacher(), qui permet de réunir 2 véhicules.

```
Vehicule voiture;
Vehicule caravane;
Vehicule attelage;

attelage = voiture.attacher(caravane);
attelage.afficher();
```

Quand on attache ainsi deux véhicules, la vitesse et la vitesse_max sont celles du vehicule qui réalise l'action.
Le poids est la somme des poids des deux véhicules.
★

Surcharge d'opérateurs

```
Vehicule Vehicule::attacher(Vehicule remorque)
{
Vehicule resultat;

resultat.vitesse = vitesse;
resultat.vitesse_max = vitesse_max;
resultat.poids = poids + remorque.poids;

return resultat;
}
```

Surcharge d'opérateurs

Dupliquer la méthode attacher() et renommer sa copie comme étant **operator+**, puis tester :

```
Vehicule voiture;
Vehicule caravane;
Vehicule attelage;

attelage = voiture+caravane;
attelage.afficher();
```

★

Surcharge d'opérateurs

```
Vehicule Vehicule::operator+(Vehicule remorque)
{
Vehicule resultat;

resultat.vitesse = vitesse;
resultat.vitesse_max = vitesse_max;
resultat.poids = poids + remorque.poids;

return resultat;
}
```