Compte-rendu - TP2 - CPP

Lucas Oros

Introduction du TP

L'objectif de ce TP est d'appliquer les principes de l'héritage multiple et de la création d'exceptions.

1 CRÉATION D'UN VÉHICULE

Dans le fichier Vehicule.hpp, nous trouverons la description de la classe Vehicule représentée ci-dessous.

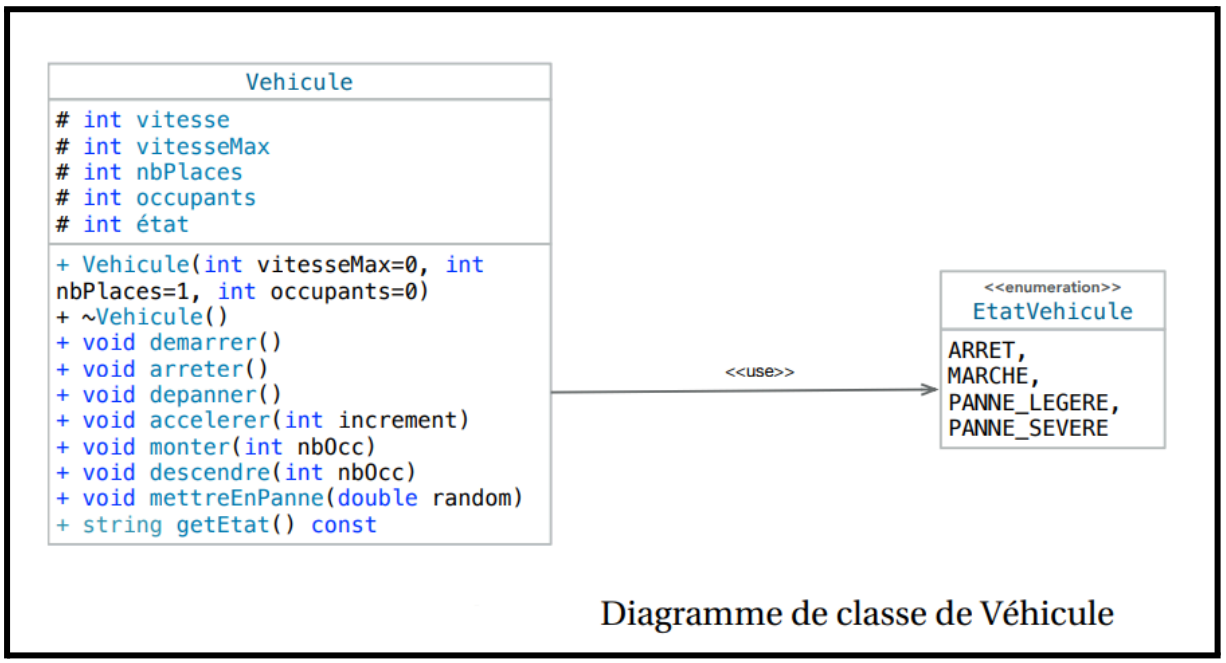


Diagramme de classe de Véhicule

Objectif

L'objectif de cette première partie est de concevoir la classe Vehicule, qui servira de classe de base pour les futures classes Voiture, Bateau et VoitureAmphibie.
Cette classe modélise un véhicule générique et pose les fondations du projet en définissant :
- la déclaration des méthodes virtuelles, qui seront redéfinies plus tard dans les classes dérivées ;
- la gestion des erreurs et incohérences à l'aide d'exceptions C++

À ce stade, les méthodes de la classe ne modifient pas encore l'état interne du véhicule (comme la vitesse ou l'état de marche).
Elles se limitent à vérifier la validité des actions demandées et à signaler les erreurs par des exceptions.

Par exemple, la méthode demarrer() contrôle que le véhicule est dans un état cohérent avant de pouvoir démarrer :

```
void Vehicule::demarrer()
{
    if (etat_==PANNE_SEVERE) throw logic_error("Le véhicule est en panne sévère, il ne peut pas démarrer");
    if (etat_==PANNE_LEGERE) throw logic_error("Le véhicule est en panne légère, il ne peut pas démarrer");
    if (etat_==MARCHE) throw logic_error("Le véhicule est déjà en marche");
    if (occupants_==0) throw logic_error("Le véhicule ne peut pas démarrer sans occupants");
}
```

Cette approche permet d'isoler la logique de validation dans la classe de base, tandis que les comportements réels (changement d'état, vitesse, etc.) seront implémentés dans les classes dérivées (Voiture et Bateau).

2 VOITURE ET BATEAU

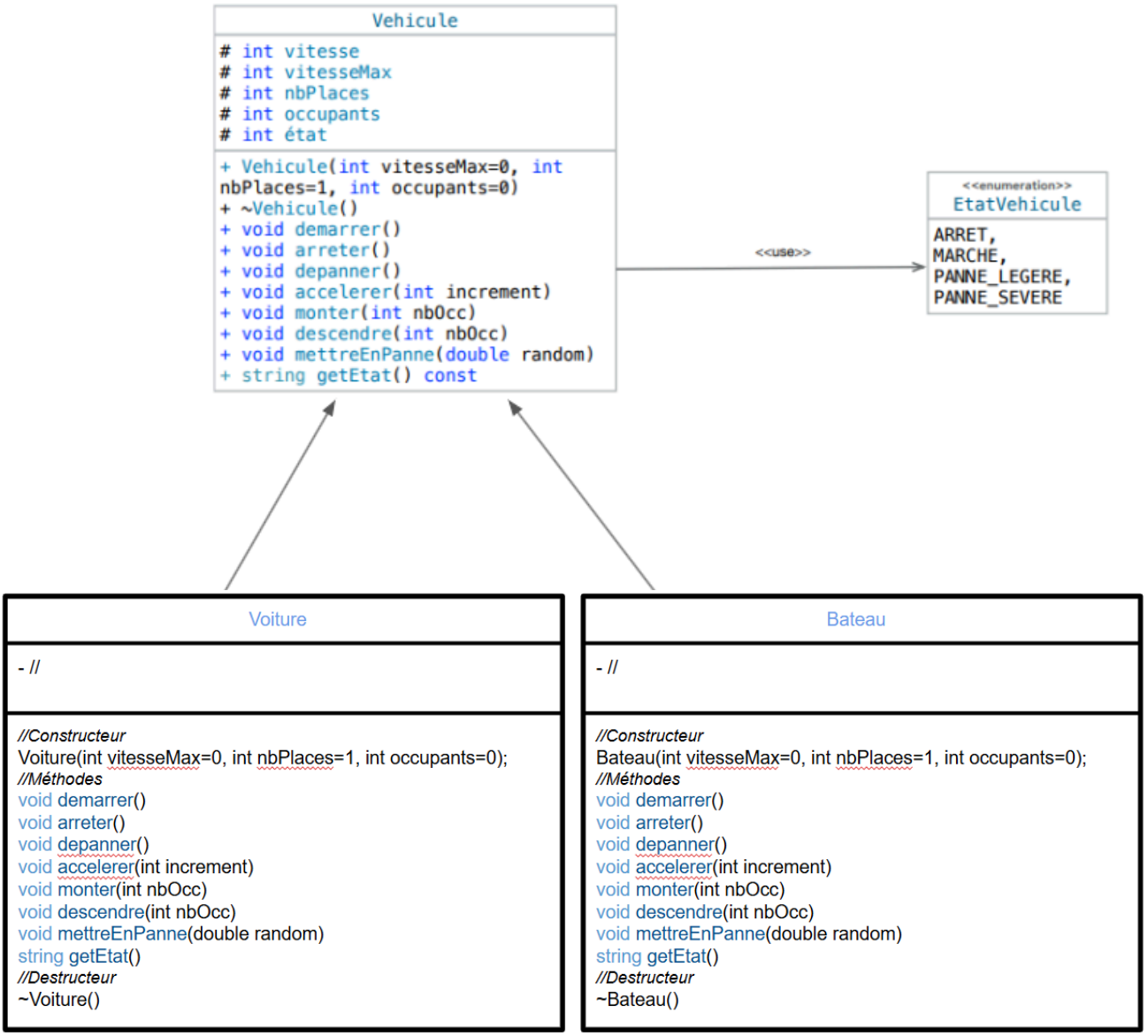
Dans cette deuxième partie, on met en place les classes Voiture et Bateau, qui héritent toutes deux de la classe de base Vehicule.
Elles reprennent les mêmes méthodes que Vehicule, mais ne sont plus déclarées comme virtuelles, puisque leur comportement devient concret.

L'objectif est de spécialiser la classe de base pour modéliser deux types de véhicules distincts, tout en réutilisant les vérifications d'erreurs déjà définies dans Vehicule.

Chaque classe dérivée (Voiture et Bateau) redéfinit les principales méthodes, notamment demarrer() et arreter().
Au début de chaque méthode, on appelle la version de la classe mère pour conserver la logique de validation :

```
void Voiture::demarrer(){
    Vehicule::demarrer(); // Appel de la méthode de la classe de base
    cout << "Démarrage d'une voiture" << endl;
    etat_=MARCHE;
    vitesse_=10; // Vitesse initiale lors du démarrage (arbitraire)
}
```

Ce principe garantit que :
- les tests d'erreurs définis dans Vehicule sont exécutés avant toute action ;
- les comportements spécifiques (affichage, mise à jour de la vitesse, état, etc.) sont ensuite appliqués dans les classes dérivées.



3 VOITURE AMPHIBIE

Dans cette dernière partie, nous avons conçu la classe VoitureAmphibie, qui hérite à la fois de Voiture et de Bateau.
L'objectif est de créer un véhicule capable de fonctionner à la fois comme une voiture et comme un bateau, avec des vitesses maximales différentes selon le mode.

3.1 Gestion du diamant d'héritage

Lorsqu'une classe hérite de deux classes qui elles-mêmes héritent d'une même classe de base (Vehicule ici), on se retrouve avec une duplication des attributs hérités.
Chaque branche (Voiture et Bateau) aurait sa propre copie des attributs vitesse_, etat_, nbPlaces_, etc., ce qui provoquerait des incohérences.

Pour éviter cela, nous avons modifié la déclaration des deux classes intermédiaires comme suit :

class Voiture : public virtual Vehicule { ... };
class Bateau : public virtual Vehicule { ... };

Le mot-clé virtual indique que Vehicule doit être hérité une seule fois, même si plusieurs classes en héritent.
Ainsi, la classe VoitureAmphibie possède une seule instance de Vehicule, partagée entre Voiture et Bateau.

3.2 Redéfinition et usage de override

Dans les classes Voiture et Bateau, nous avons ajouté le mot-clé override dans les signatures des méthodes redéfinies, par exemple :
- void demarrer() override;
- void arreter() override;

Ce mot-clé permet au compilateur de vérifier que la méthode redéfinit bien une méthode virtuelle de la classe de base (Vehicule).

3.3 Structure de la classe VoitureAmphibie

La classe VoitureAmphibie hérite donc des deux classes précédentes :

```
class VoitureAmphibie : public Voiture, public Bateau
{
protected:
    bool mode_; // false = mode voiture, true = mode bateau
    int vitesseMaxVoiture_;
    int vitesseMaxBateau_;

public:
```

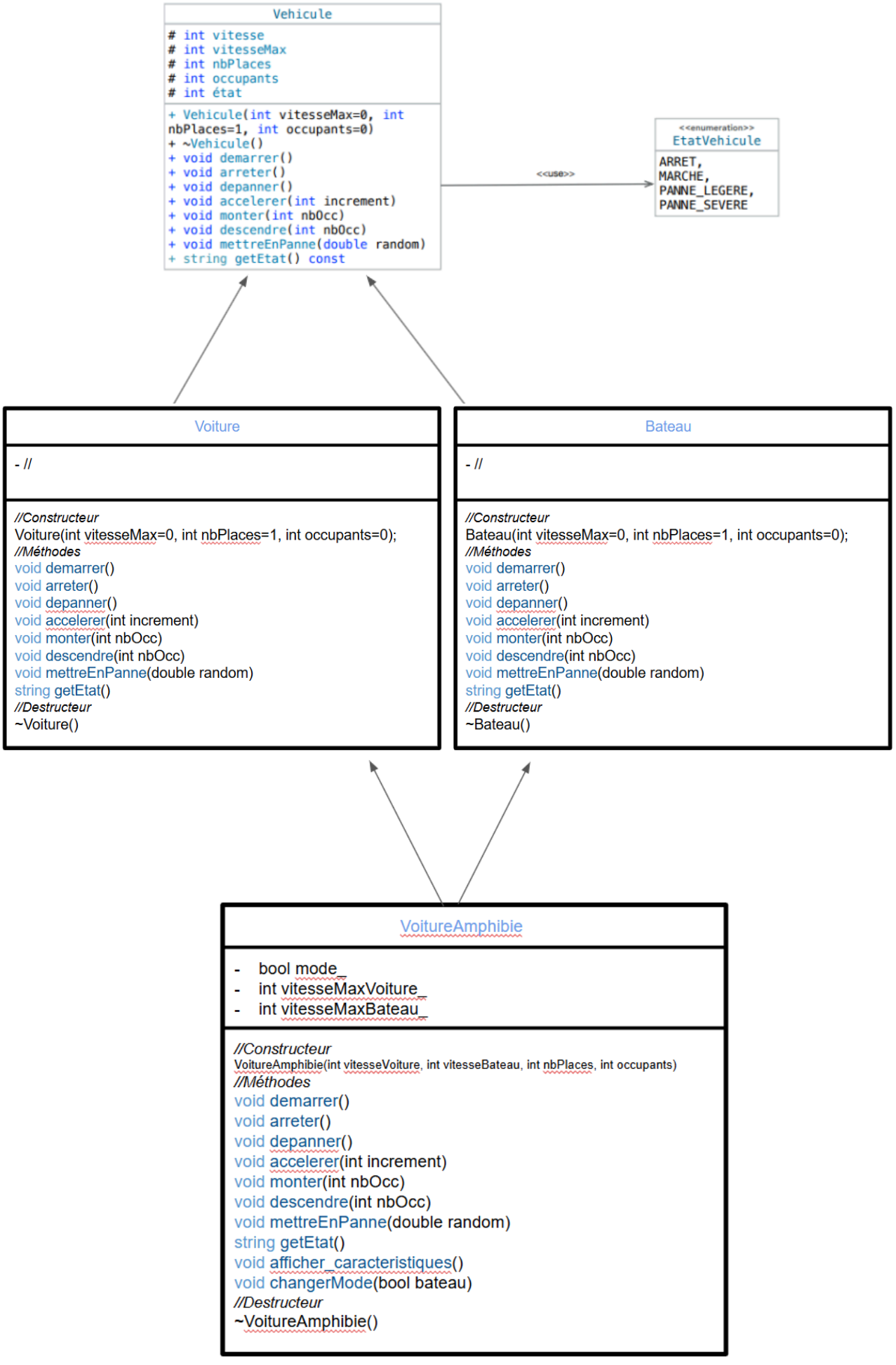
On introduit ici un attribut mode_ qui permet de savoir dans quel mode se trouve le véhicule (sur route ou sur mer).
Cette information est indispensable, car les vitesses maximales diffèrent selon le mode d'utilisation (exemple: pour accélérer (mise à jour de la vitesse)).

3.4 Constructeur de VoitureAmphibie

Le constructeur de la classe est défini comme suit :

```
VoitureAmphibie::VoitureAmphibie(int vitesseVoiture, int vitesseBateau, int nbPlaces, int occupants)
    : Vehicule(vitesseVoiture, nbPlaces, occupants),   // on démarre en mode voiture
      Voiture(vitesseVoiture, nbPlaces, occupants),
      Bateau(vitesseBateau, nbPlaces, occupants),
      vitesseMaxVoiture_(vitesseVoiture),
      vitesseMaxBateau_(vitesseBateau),
      mode_(false)
{
    cout << "Constructeur de la classe VoitureAmphibie" << endl;
}
```

Explication :
- Le constructeur de Vehicule est appelé explicitement une seule fois (héritage virtuel oblige).
- Les constructeurs de Voiture et Bateau sont également appelés pour initialiser leurs parties respectives.
- vitesseMaxVoiture_ et vitesseMaxBateau_ stockent les deux vitesses maximales distinctes.
- Le mode est initialisé à false, c'est-à-dire en mode voiture par défaut.


Vehicule
int vitesse
int vitesseMax
int nbPlaces
int occupants
int état
+ Vehicule(int vitesseMax=0, int nbPlaces=1, int occupants=0)
+ ~Vehicule()
+ void demarrer()
+ void arreter()
+ void depanner()
+ void accelerer(int increment)
+ void monter(int nbOcc)
+ void descendre(int nbOcc)
+ void mettreEnPanne(double random)
+ string getEtat() const
<<use>>
<<enumeration>>
EtatVehicule
ARRET,
MARCHE,
PANNE_LEGERE,
PANNE_SEVERE
Voiture
- //
//Constructeur
Voiture(int vitesseMax=0, int nbPlaces=1, int occupants=0);
//Méthodes
void demarrer()
void arreter()
void depanner()
void accelerer(int increment)
void monter(int nbOcc)
void descendre(int nbOcc)
void mettreEnPanne(double random)
string getEtat()
//Destructeur
~Voiture()
Bateau
- //
//Constructeur
Bateau(int vitesseMax=0, int nbPlaces=1, int occupants=0);
//Méthodes
void demarrer()
void arreter()
void depanner()
void accelerer(int increment)
void monter(int nbOcc)
void descendre(int nbOcc)
void mettreEnPanne(double random)
string getEtat()
//Destructeur
~Bateau()
VoitureAmphibie
- bool mode_
- int vitesseMaxVoiture_
- int vitesseMaxBateau_
//Constructeur
VoitureAmphibie(int vitesseVoiture, int vitesseBateau, int nbPlaces, int occupants)
//Méthodes
void demarrer()
void arreter()
void depanner()
void accelerer(int increment)
void monter(int nbOcc)
void descendre(int nbOcc)
void mettreEnPanne(double random)
string getEtat()
void afficher_caracteristiques()
void changerMode(bool bateau)
//Destructeur
~VoitureAmphibie()

4 Tests

Pour valider le bon fonctionnement des classes développées, plusieurs fonctions de test ont été écrites dans le programme principal (main.cpp).
Ces tests permettent de provoquer volontairement des situations incohérentes afin de vérifier la levée correcte des exceptions définies dans la classe Vehicule.

Dans le main on fait des fonctions permettant de lever volontairement ces erreurs :
— si la vitesse dépasse la vitesse maximum
— si on cherche à démarrer un véhicule déjà en marche ou en panne
— si on cherche à ajouter un occupant mais que le véhicule est déjà plein
— si on cherche à faire descendre plus d'occupants qu'il y en avait dans le véhicule

Tests fonctionnels

En complément, trois fonctions de test distinctes ont été réalisées :
Voiture_test() pour la classe Voiture,
Bateau_test() pour la classe Bateau,
VoitureAmphibie_test() pour la classe VoitureAmphibie.

Chaque fonction simule un scénario d'utilisation typique :
- création du véhicule,
- montée et descente d'occupants,
- démarrage, accélération, panne, dépannage, et nouvel essai de démarrage.

Voici le test de la VoitureAmphibie dans le main ainsi que le résultat ci dessous :

```
void voitureAmphibie_test(void){
    VoitureAmphibie va(150, 50, 4 ,1);
    va.afficher_caracteristiques();

    cout << "\n-- Mode voiture --" << endl;
    va.monter(2);
    va.demarrer();
    va.accelerer(100);
    cout << va;
    double random = (double)rand() / RAND_MAX;
    va.mettreEnPanne(random);
    va.afficher_caracteristiques();
    va.descendre(1);
    va.depanner();

    cout << "\n-- Changement en mode bateau --" << endl;
    va.changerMode(true);
    va.demarrer();
    va.accelerer(30);
    va.afficher_caracteristiques();

}
```

```
------------------------
Constructeur de la classe VoitureAmphibie
=== VoitureAmphibie ===
Vitesse max en mode voiture : 150
Vitesse max en mode bateau  : 50
Nombre de places : 4
Occupants : 1
Etat : ARRET

-- Mode voiture --
Montée de 2 occupants dans une voiture amphibie
Démarrage d'une voiture amphibie
Démarrage d'une voiture
Accélération d'une voiture amphibie de 100
Vitesse : 110 / Vitesse max : 150 / Nombre de places : 4 / Occupants : 3 / Etat : MARCHE
Mise en panne d'une voiture amphibie
=== VoitureAmphibie ===
Vitesse max en mode voiture : 150
Vitesse max en mode bateau  : 50
Nombre de places : 4
Occupants : 3
Etat : PANNE_SEVERE
Descente de 1 occupants d'une voiture amphibie
Dépannage d'une voiture amphibie

-- Changement en mode bateau --
Passage en mode BATEAU
Démarrage d'une voiture amphibie
Démarrage d'une voiture
Accélération d'une voiture amphibie de 30
=== VoitureAmphibie ===
Vitesse max en mode voiture : 150
Vitesse max en mode bateau  : 50
Nombre de places : 4
Occupants : 2
Etat : MARCHE
Destructeur de la classe VoitureAmphibie
Destructeur de la classe Bateau
Destructeur de la classe Voiture
```