TD 2 - Amitié et surcharge d'opérateurs

Programmation Orientée Objet

Objectif

- Comprendre le principe des fonctionnalités amies
- Comprendre ce qu'est la surcharge des opérateurs ;

1 Fonctions amies

Créer un classe `Avion` ayant pour données privées le fabriquant, le type, l'altitude et le cap de l'avion. Créer une fonction *non membre* `afficheFiche`, amie de la classe `Avion`, permettant d'afficher les informations caractéristiques d'un avion sous le format ci-dessous :

```
**** AVION ****
* Airbus
* A320
* Alt : 10000
* Cap : 345
******************
```

```
#ifndef AVION_H
#define AVION_H
#include<iostream>
#include<string>

using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Avion
////////////////////////////////////////////////////////////////////////////////////////////////////

class Avion{
        string sFabriquant;
        string sType;
        int iAlt;
        int iCap;

        public:
                // Constructeur et destructeur
                Avion(const string &, const string &, int, int);
                ~Avion();

                //Accesseurs
                string getFabriquant() const;
                void setFabriquant(const string &);
                string getType() const;
                void setType(const string &);
                int getAlt() const;
                void setAlt(const int);
                int getCap() const;
                void setCap(const int);

                // fonctions utilitaires
                void affiche();

                // fonctions amies
                friend void afficheFiche(const Avion);

};


#endif
```

```
#include "Avion.h"

//*****************************************************************//
Avion::Avion(const string & f, const string & t , int a, int c){
        setFabriquant(f);
        setType(t);
        setAlt(a);
        setCap(c);
}

//*****************************************************************//
Avion::~Avion(){
}

//*****************************************************************//
string Avion::getFabriquant() const{
        return sFabriquant;
}

//*****************************************************************//
void Avion::setFabriquant(const string &f){
                sFabriquant = f;
}

//*****************************************************************//
string Avion::getType() const{
        return sType;
}

//*****************************************************************//
void Avion::setType(const string & t){
        sType = t;
}

//*****************************************************************//
int Avion::getAlt() const{
        return iAlt;
}

//*****************************************************************//
void Avion::setAlt(const int a){
        iAlt = a;
}

//*****************************************************************//
int Avion::getCap() const{
        return iCap;
}

//*****************************************************************//
void Avion::setCap(const int c){
        iCap = c;
}

//*****************************************************************//
void Avion::affiche(){
                cout << "Avion : " << getFabriquant() << " " << getType() << " " << getAlt() << " " << getCap(
}
```

```
#include "Avion.h"

// Implémentation de la fonction amie afficheFiche
// Celle-ci ne recevant pas d'argument implicite comme
// le ferait une fonction membre. Il faut donc passer un
// argument de type Personne
void afficheFiche(const Avion a)
{
        cout << "**** AVION ****\n* " << a.sFabriquant << "\n* " << a.sType  << "\n* Alt : " << a.iAlt << "\n*
}


// ******************************
// Prog principal
// ******************************

int main(){
        Avion A("Airbus", "A320", 10000, 345);

        afficheFiche(A);

        return (EXIT_SUCCESS);
```

```
}
```

2 Classes amies et surcharge

Créez une classe `TourDeControle`. Cette classe contiendra un tableau d'`Avion` de taille fixe, le nombre d'avions enregistrées dans ce tableau. Cette classe sera une classe amie de la classe `Avion`.

1) Que faut-il faire pour rendre TourDeControle amie de la classe Avion ?

Il faut la déclarer amie dans la classe avion

3) Créez une fonction membre `AjouterTabAvionNormal` ajoutant un avion au tableau en utilisant les accesseurs de la classe `Avion`.

4) Créez une fonction membre `AjouterAvionViaAmie` ajoutant une personne au tableau en utilisant les accès directs à la classe `Avion`.

5) Comment faudrait-il faire pour ajouter un avion sans passer par les accesseurs ni par un accès direct ? Proposez une solution.

Il faut la déclarer surcharger l'opérateur d'égalité

6) Créez un programme principal manipulant la classe `TourDeControle`. Pour cela créez 3 avions différents, ajoutez les en utilisant les différentes méthodes, et affichez les avions du tableau à l'aide de la méthode `affiche()`.

7) Surchargez la méthode cout pour afficher directement les éléments d'un avion.

```
#ifndef AVION_H
#define AVION_H
#include<iostream>
#include<string>

using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Avion
////////////////////////////////////////////////////////////////////////////////////////////////

class Avion{
        string sFabriquant;
        string sType;
        int iAlt;
        int iCap;

        public:
                // Constructeur et destructeur
                Avion ();
                Avion(const string &, const string &, int, int);
                ~Avion();

                //Accesseurs
                string getFabriquant() const;
                void setFabriquant(const string &);
                string getType() const;
                void setType(const string &);
                int getAlt() const;
                void setAlt(const int);
                int getCap() const;
                void setCap(const int);

                // fonctions utilitaires
                void affiche();

                // surcharge opertor affectation
                 Avion operator= (const Avion);

                 friend ostream & operator<< (ostream & , const Avion & );

                // classe amie
                friend class TourDeControle;

};


#endif
```

```
#include "Avion.h"

//*****************************************************************//
Avion::Avion(){
}

//*****************************************************************//
Avion::Avion(const string & f, const string & t , int a, int c){
        setFabriquant(f);
        setType(t);
        setAlt(a);
        setCap(c);
}

//*****************************************************************//
Avion::~Avion(){
}

//*****************************************************************//
string Avion::getFabriquant() const{
        return sFabriquant;
}

//*****************************************************************//
void Avion::setFabriquant(const string &f){
                sFabriquant = f;
}

//*****************************************************************//
string Avion::getType() const{
        return sType;
}

//*****************************************************************//
void Avion::setType(const string & t){
        sType = t;
}

//*****************************************************************//
int Avion::getAlt() const{
        return iAlt;
}

//*****************************************************************//
void Avion::setAlt(const int a){
        iAlt = a;
}

//*****************************************************************//
int Avion::getCap() const{
        return iCap;
}

//*****************************************************************//
void Avion::setCap(const int c){
        iCap = c;
}

//*****************************************************************//
void Avion::affiche(){
                cout << "Avion : " << getFabriquant() << " " << getType() << " " << getAlt() << " " << getCap(
}

//*****************************************************************//
Avion Avion::operator= (const Avion a){
        this->sFabriquant = a.sFabriquant;
        this->sType = a.sType;
        this->iAlt = a.iAlt;
        this->iCap = a.iCap;

        cout << "**** Operateur d affectation ***" << endl;

 return a;
}

//*****************************************************************//
ostream & operator<< (ostream & os, const Avion & av){
     os << "Avion : " << av.getFabriquant() << " " << av.getType() << " " << av.getAlt() << " " << av.getCap()
     return os;
 }
```

```
#ifndef TOURDECONTROLE_H
#define TOURDECONTROLE_H
#include<iostream>
#include<string>
#include<vector>

#include "Avion.h"

using namespace std;

const int TAILLEMAX = 3;

////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe TourDeControle
////////////////////////////////////////////////////////////////////////////////////////////////

class TourDeControle{

        Avion Tab[TAILLEMAX];

        int iNbAvion;

        public:
                // Constructeur et destructeur
                TourDeControle();
                ~TourDeControle();

                void AjouterAvionViaAmie(Avion&);
                void AjouterAvionViaNormal(Avion&);
                void AjouterAvionViaSurcharge(Avion&);
                void AfficherListeAvionNormal();
                void AfficherListeAvionViaSurcharge();
};


#endif
```

```
#include "TourDeControle.h"
//*****************************************************************//
TourDeControle::TourDeControle(){
        // Le new ne peut fonctionner que sur des classes.
        // Donc il faut utiliser un malloc
        // pour un tableau defini comme suit : Avion * tab[];
        // MAUVAISE Tab = new (Avion *)[iNb];
        /// BON : Tab = (Avion*) malloc (sizeof(Avion*)*iNb);

        iNbAvion = 0;
}

//*****************************************************************//
TourDeControle::~TourDeControle(){
}
//*****************************************************************//
void TourDeControle::AjouterAvionViaAmie(Avion &pA){
        if (iNbAvion < TAILLEMAX){
                Tab[iNbAvion].sFabriquant=pA.sFabriquant;
                Tab[iNbAvion].sType=pA.sType;
                Tab[iNbAvion].iAlt=pA.iAlt;
                Tab[iNbAvion].iCap=pA.iCap;

                iNbAvion++;
        }
}
//*****************************************************************//
void TourDeControle::AjouterAvionViaNormal(Avion &pA){
        if (iNbAvion < TAILLEMAX){
                Tab[iNbAvion].sFabriquant=pA.getFabriquant();
                Tab[iNbAvion].sType=pA.getType();
                Tab[iNbAvion].iAlt=pA.getAlt();
                Tab[iNbAvion].iCap=pA.getCap();

                iNbAvion++;
        }
}
//*****************************************************************//
void TourDeControle::AjouterAvionViaSurcharge(Avion &pA){
        if (iNbAvion < TAILLEMAX){
                Tab[iNbAvion]=pA;

                iNbAvion++;
        }
```

```
}
//*****************************************************************//
void TourDeControle::AfficherListeAvionNormal(){
        for (int i=0; i<iNbAvion;i++)
                Tab[i].affiche();
}
//*****************************************************************//
void TourDeControle::AfficherListeAvionViaSurcharge(){
        for (int i=0; i<iNbAvion; i++)
                cout << Tab[i];
        }
```

```
#include "Avion.h"
#include "TourDeControle.h"


// ******************************
// Prog principal
// ******************************

int main(){
        TourDeControle tour;

        Avion av1 ("Airbus", "A320", 10000, 340);
        Avion av2 ("Boeing", "747", 2000, 90);
        Avion av3 ("ATR", "42", 30000, 255);

        tour.AjouterAvionViaAmie(av1);
        tour.AjouterAvionViaNormal(av2);
        tour.AjouterAvionViaSurcharge(av3);

        tour.AfficherListeAvionNormal();
        tour.AfficherListeAvionViaSurcharge();

        return (EXIT_SUCCESS);
}
```

```
all: main

main: Main.cpp Personne.o DRH.o
        g++ Personne.o DRH.o Main.cpp -o main
Personne.o: Personne.cpp
        g++ -c Personne.cpp
DRH.o: DRH.cpp Personne.h
        g++ -c DRH.cpp
```