# TD 1 - Classes et Objets

Programmation Orientée Objet

## Objectif

- Comprendre ce qu'est une classe ;
- Comprendre ce qu'est un objet ;
- Comprendre la différence entre une classe et un objet ;
- Manipuler un programme objet.

## 1 Classe HelloWorld

1) Ecrivez un programme C++ affichant "Hello World !".

2) Ecrivez une classe `HelloWorld` dont une méthode affiche le message "Hello World !". Donnez le code du programme principal appelant la méthode de la classe `HelloWorld`.

```
#include<iostream>
#include<cstring>

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


class HelloWorld{
        public:
                void affiche() const;
};

void HelloWorld::affiche() const{
        cout << "Hello World" << endl;
}

////////////////////////////////////////////////////////////////////////////////////////////////
// Programme principal
////////////////////////////////////////////////////////////////////////////////////////////////

int main()
{
        cout << "Hello World" << endl;

        HelloWorld hw;
        hw.affiche();

        return (EXIT_SUCCESS);
}
```

## 2 Classe Personne - Premiers pas

1) Créez une classe Personne. `Cette` classe comportera les informations suivantes stockées sous la forme de chaînes de caractères : le `nom` et le `prénom`. Cette classe aura comme opération : `afficher`, `saisir` et `raz`.

2) Ecrire un petit programme d'essai qui affecte tout d'abord les valeurs aux différents champs d'une telle classe, les affiche avant de leur appliquer la fonction `raz`.

```
#include<iostream>
#include<cstring>

using namespace std;
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe HelloWorld
////////////////////////////////////////////////////////////////////////////////////////////////////

const int TAILLE=20;

class Personne{
        char sNom[TAILLE];
        char sPrenom[TAILLE];

        public:
                void affiche() const;
                void saisir(const char *, const char *);
                void raz();
};

void Personne::affiche() const{
        cout << " " << sNom << " " << sPrenom << endl;
}

void Personne::saisir(const char *Nom, const char *Prenom){
        strcpy(sNom,Nom);
        strcpy(sPrenom,Prenom);
}

void Personne::raz(){
        strcpy(sNom,"");
        strcpy(sPrenom,"");
}

////////////////////////////////////////////////////////////////////////////////////////////////////
// Programme principal
////////////////////////////////////////////////////////////////////////////////////////////////////

int main()
{
        Personne p;
        char sBuffer1 [TAILLE];
        char sBuffer2 [TAILLE];

        cout << "Entrez le nom : " ;
        cin >> sBuffer1;

        cout << "Entrez le prénom : " ;
        cin >> sBuffer2;

        // Saisie du nom et du prénom
        p.saisir(sBuffer1, sBuffer2);

        // Affichage pour verification
        p.affiche();

        // remise à zero
        p.raz();

        // Affichage pour verification
        p.affiche();

        return (EXIT_SUCCESS);
}
```

2) Ecrire un constructeur et un destructeur pour cette classe. Le constructeur devra renseigner les champs de `personne`. Ecrire un programme principal créant un objet instance de la classe `Personne`.

```
#include<iostream>
#include<cstring>

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

const int TAILLE=20;

class Personne{
        char sNom[TAILLE];
        char sPrenom[TAILLE];

        public:
//              Personne();
                Personne(const char *, const char *);
```

```
                ~Personne();
                void affiche() const;
                void saisir(const char *, const char *);
                void raz();
};

/*Personne::Personne(){
        #ifdef DEBUG
        cout << " constructeur sans parametre\n";
        #endif
}
*/
Personne::Personne(const char * sBuffer1, const char * sBuffer2){
        #ifdef DEBUG
        cout << " constructeur avec parametres\n";
        #endif
        // Saisie du nom et du prénom
        saisir(sBuffer1, sBuffer2);
}

Personne::~Personne(){
        #ifdef DEBUG
        cout << " destructeur" << endl;
        #endif
}

void Personne::affiche() const{
        cout << " " << sNom << " " << sPrenom << endl;
}

void Personne::saisir(const char *Nom, const char *Prenom){
        strcpy(sNom,Nom);
        strcpy(sPrenom,Prenom);
}

void Personne::raz(){
        strcpy(sNom,"");
        strcpy(sPrenom,"");
}

////////////////////////////////////////////////////////////////////////////////////////////////////
// Programme principal
////////////////////////////////////////////////////////////////////////////////////////////////////

int main()
{
cout << " Constructeur simple : ";
        Personne p ("","");
        // On est obligé de préciser que le constructeur
        // a des parametres maintenant, même si ces parametres sont vides
cout << " Constructeur avec parametres : ";
        Personne p2 ("Belloir", "Nicolas") ;

        // affectation
        p = p2;

        return (EXIT_SUCCESS);
}
```

3) On désire ajouter des accesseurs aux membres privés de la classe. En quoi cela consiste-t-il ? Quel est l'intérêt de cela ? Implémentez des accesseurs sur les membres privées de `Personne`.

```
#include<iostream>
#include<cstring>

using namespace std;

const int TAILLE=20;

class Personne{
        char sNom[TAILLE];
        char sPrenom[TAILLE];

        public:
                Personne(const char *, const char *);
                ~Personne();
                char * getNom() ;
                void setNom(const char *);
                char * getPrenom();
                void setPrenom(const char *);
                void affiche() const;
```

```
                void saisir(const char *, const char *);
                void raz();
};

Personne::Personne(const char *sBuffer1, const char *sBuffer2){
        #ifdef DEBUG
        cout << " constructeur avec parametres\n";
        #endif
        // Saisie du nom et du prénom
        setNom(sBuffer1);
        setPrenom(sBuffer2);
}

Personne::~Personne(){
        #ifdef DEBUG
        cout << " destructeur" << endl;
        #endif
}

char * Personne::getNom() {
        return(sNom);
}

void Personne::setNom(const char * Nom){
        strcpy(sNom, Nom);
}

char * Personne::getPrenom() {
        return(sPrenom);
}

void Personne::setPrenom(const char * Prenom){
        strcpy(sPrenom, Prenom);
}

void Personne::affiche() const{
        cout << "Personne : " << sNom << " " << sPrenom << endl;
}

void Personne::saisir(const char *Nom, const char *Prenom){
        strcpy(sNom,Nom);
        strcpy(sPrenom,Prenom);
}

void Personne::raz(){
        strcpy(sNom,"");
        strcpy(sPrenom,"");
}

////////////////////////////////////////////////////////////////////////////////
// Programme principal
////////////////////////////////////////////////////////////////////////////////

int main()
{
        const char sNom[]="Belloir";
        const char sPrenom[]="Nicolas";

        Personne p(sNom, sPrenom) ;

        p.affiche();

        // Test accessors
        cout << "Test accessors : \n";
        cout << p.getNom() << endl;
        cout << p.getPrenom() << endl;


        return (EXIT_SUCCESS);
}
```

# 3 Classe Voiture - Approche composants

Jusqu'à présent, nous avons réalisé les classes dans un seul fichier source commun avec le programme principal. Cette approche est contraire à l'esprit de la POO. On y préfère implémenter une classe sous la forme d'un composant logiciel, c'est à dire en décrivant la structure de la classe dans un fichier d'entête (`.h`) et le code source

des fonctions membres dans un fichier de source(`.cpp`). Enfin le programme principal sera implémenté dans un programme source à part. Faire cela pour la classe `Voiture`, en réalisant les fichiers `Voiture.h`, `Voiture.c` `MainVoiture.c`. Attention au problème de la double inclusion !

On considère qu'une voiture est caractérisée par sa marque, son modèle. De plus, elle peut contenir un lien vers ses passagers. Ceux-ci sont au maximum au nombre de 5. Il convient donc de créer également une fonction d'ajout de passagers à la voiture ainsi qu'une méthode `afficherPassager()` affichant le nom de chaque passager.

Créer les classes et les méthodes permettant la gestion de cette classe `Voiture`.

```
#ifndef PERSONNE_H
#define PERSONNE
#include<iostream>
#include<cstring>

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

const int TAILLE=20;

class Personne{
        string sNom;
        string sPrenom;

        public:
                Personne(const string &, const string &);
                Personne () {};
                ~Personne();
                string & getNom() ;
                void setNom(const string &);
                string & getPrenom()  ;
                void setPrenom(const string&);
                void affiche() const{
                                cout << "Personne : " << sNom << " " << sPrenom << endl;
                }
                void saisir(const string &, const string &);
};

#endif
```

```
#include "Personne.h"

Personne::Personne(const string & sBuffer1, const string & sBuffer2){
        #ifdef DEBUG
        cout << " constructeur avec parametres\n";
        #endif
        // Saisie du nom et du prénom
        setNom(sBuffer1);
        setPrenom(sBuffer2);
}

Personne::~Personne(){
        #ifdef DEBUG
        cout << " destructeur" << endl;
        #endif
}

string & Personne::getNom() {
        return(sNom);
}

void Personne::setNom(const string & Nom){
        sNom= Nom;
}

string & Personne::getPrenom() {
        return(sPrenom);
}

void Personne::setPrenom(const string & Prenom){
        sPrenom= Prenom;
}

void Personne::saisir(const string &Nom, const string &Prenom){
        sNom = Nom;
        sPrenom = Prenom;
}
```

```
#ifndef VOITURE_H
#define VOITURE
#include<iostream>
#include<string>

#include "Personne.h"

using namespace std;

const int NB_PASSAGERS = 5;
////////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe VOITURE
////////////////////////////////////////////////////////////////////////////////////////////////////

class Voiture{
        string sMarque;
        string sModele;
        Personne * tabBassager[NB_PASSAGERS];
        int iNbPassagers;

        private :
                int getNbPassagers() const;
        public:
                Voiture(const string &, const string &);
                Voiture ();
                ~Voiture();
                string getMarque() const;
                void setMarque(const string &);
                string getModele() const ;
                void setModele(const string &);
                int AjoutePassager(Personne *);
                void affiche() const;
                void afficheListePassagers() const;
                void saisir(const string &, const string &);
};

#endif
```

```
#include "Voiture.h"
//*************************************************************//
Voiture::Voiture(const string & marque, const string & modele){
        sMarque = marque;
        sModele = modele;
        iNbPassagers=0;
        for(int i=0; i< NB_PASSAGERS; i++){
                tabBassager[i]=NULL;
        }
}

//*************************************************************//
//** REMARQUE                                                **//
//*************************************************************//
//** On est obliege d avoir un constructeur vierge pour pouvoir //
//** continuer dans le code Ã  declarer des objets comme :       //
//**      Voiture v;                                           //
//** Dans le cas contraire  il faudrait :                      //
//**      Voiture v("REnault", "Kangoo");                      //
//** On peut aussi allouer dynamiquement                       //
//**              Voiture * v3 = new Voiture("Renault", "Kangoo");        //
//*************************************************************//
Voiture::Voiture () {
        sMarque = "";
        sModele = "";
        iNbPassagers=0;
        for(int i=0; i< NB_PASSAGERS; i++){
                tabBassager[i]=NULL;
        }

}

Voiture::~Voiture(){
        for (int i=0; i< getNbPassagers(); i++)
                delete tabBassager[i];

//      delete [] tabBassager; plante si cases vides
}

//*************************************************************//
//** REMARQUE                                                //
//*************************************************************//
//** On peut Ã¤tre tente de retourner une refernce sur le string //
```

```
//** Cependant, si on le fait, cela veut dire qu on travaille   //
//** directement sur l'objet string de la voiture. Ainsi, on    //
//** viole l'encapsulation d une part et d'autre part c est     //
//** incoherent avec le const appliquÃ© a la fonction qui indique//
//** qu on ne modifie pas l objet. D ou levee d une erreur de   //
//** de compilation. Il faut donc faire :                       //
//** string Voiture::getMarque() const                  ou     //
//** string & Voiture::getMarque()                              //
//************************************************************//
string Voiture::getMarque() const{
        return (sMarque);
}

//************************************************************//
void Voiture::setMarque(const string & marque){
        sMarque = marque;
}

//************************************************************//
string Voiture::getModele() const {
        return (sModele);
}

//************************************************************//
void Voiture::setModele(const string & modele){
        sModele = modele;
}

//************************************************************//
int Voiture::AjoutePassager(Personne *p){
        if (iNbPassagers<NB_PASSAGERS){
                tabBassager[iNbPassagers++] = p;
        }
        else
                return -1;
        return 0;
}

//************************************************************//
int Voiture::getNbPassagers() const{
        return (iNbPassagers);
}

//************************************************************//
void Voiture::affiche() const{
                cout << "Voiture : " << sMarque << " " << sModele << endl;
}

//************************************************************//
void Voiture::afficheListePassagers() const{
        for(int i=0; i< getNbPassagers(); i++){
                //** On utilise le pointeur et non le point !!! **//
                cout << tabBassager[i]->getNom() << " ";
        }
        cout << endl;
}

//************************************************************//
void Voiture::saisir(const string & marque, const string & modele){
        setMarque(marque);
        setModele(modele);
}
```

```
#include "Voiture.h"

int main (){

        Voiture v1;
        Voiture v2("Renault", "Kangoo");

        Voiture * v3 = new Voiture("Renault", "Kangoo");

        Personne * p = new Personne("Belloir", "Nicolas");
//      v2.AjoutePassager(new Personne("Belloir", "Nicolas"));
        v2.AjoutePassager(p);
        v2.affiche();
        v2.afficheListePassagers();

        delete v3;

        return 0;
```

```
}
```

# 4 Ensemble

Définir une classe `Ensemble` pour manipuler des ensembles d' "elements" (`#define elements int` dans un premier temps). On utilisera l'allocation dynamique pour manipuler des ensembles de taille quelconque.

1) Donnez la définition de la classe dans un premier temps sans son implémentation. On considère pour le moment que la classe Ensemble ne contient que les opérations `appartient`, `card` et `inserer`.

2) L'utilisation de l'allocation dynamique implique d'initialiser les données membres de la classe lors de sa création. Pour cela, il vous faut modifier le constructeur par défaut, ainsi que le destructeur.

3) Ajouter une méthode `affiche()` pour la classe.

4) Surchargez la méthode `affiche()` de manière à ce qu'elle puisse prendre comme paramètre le rang d'un élément et afficher cet élément.

5) Ajouter un programme principal utilisant la classe `Ensemble`. Créez deux éléments. Donnez leur une valeur. Ajouter les à l'ensemble. Affichez.

6) Modifier l'ensemble des programmes afin de manipuler un ensemble de personnes.

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

#define element int

class Ensemble{
        element * tab;
        int taille;
  public :
        int card() const;
        bool appartient (const element e) const ;
        void inserer (const element e) ;
        void affiche () const;
        void affiche (const int iRang) const;
        Ensemble();
        ~Ensemble();
};

/* Redefinition du constructeur par defaut afin de lui permettre
d'initialiser les donnees membres */
Ensemble::Ensemble(){
  taille=0;
  tab=NULL;
}

/* Redefinition du destructeur par defaut afin de lui permettre
de liberer les donnees allouees dynamiquement */
Ensemble::~Ensemble(){
  taille=0;
  delete [] tab;
  tab=NULL;
}

/* L'operation est specifiee comme const car elle ne modifie pas
l'ensemble */
int Ensemble::card() const{
  return taille;
}

/* Le parametre e1 est specifie comme const car il n'est pas modifie par l'operation */
bool Ensemble::appartient(const element el) const{
  int i=0;

        if (tab!=NULL)
                for (int i=0; i< taille; i++)
                        if (tab[i]==el)
                                return true;
         return (false);
}

/* L'operation n'est pas specifiee comme const car elle modifie l'ensemble */
void Ensemble::inserer (const element e){
    if (!appartient(e)){
                //tab=(element*)realloc(tab,(++taille)*sizeof(element));
                // le new ne permet pas de faire de la reallocation memoire
```

```
                // il faut donc allouer un nouveau bloc pour la nouvelle taille
                // et recopier la totalite du precedent tableau dans le nouveau
                // et enfin liberer l'ancien
                // ou alors utiliser le template vector
        element * tab1 = new element [++taille];  // allocation tab temporaire
            // recopie des elements de tab dans tab1
            for (int i=0; i<taille-1;i++){
                tab1[i] = tab[i];
            }
            //affectation du nouvel element
        tab1[taille-1]=e;
            // liberation de la memoire reservee par l ancien tableau
            delete [] tab;
            // affectation du nouveau tableau
            tab=tab1;
    }
}

void Ensemble::affiche () const{
        for (int i=0; i<taille; i++)
                cout << tab[i] << ' ';
        cout << endl;
}

void Ensemble::affiche (const int iRang) const{
        cout << tab[iRang] << endl;
}

main(){

        Ensemble ens1;
        element e1, e2;
        const element e3=10;

        e1 = 3;
        e2 = 7;

        ens1.inserer(e1);
        ens1.inserer(e2);
        ens1.inserer(e3);

        ens1.affiche();
        ens1.affiche(2);
}
```