Algorithmique et Programmation 2 : Initiation à la Programmation Orientée Objet

Algorithmique et Programmation 2 : Les Classes

Objectifs

- Initiation à la **Programmation Orientée Objet** (Classe, héritage, polymorphisme, agrégation, classe abstraite)
- Algorithmique : manipulation de **Types Abstraits de Données** (Pile, File, Liste, Arbre)

Programmation Impérative Exemple

```
struct Personne {
  string prenom;
  string nom;
  int age;
} ;
```

Structure Personne

```
void initialise ( Personne & p, string pr, string n, int a );
void quiSuisJe ( Personne p );
void anniversaire ( Personne & p );
```

Fichier.h

```
#include "Personne.h"

void initialise( Personne & p, string pr, string n, int a ) {
  p.prenom = pr;
  p.nom = n;
  p.age = a;
}

void quiSuisJe( Personne p ) {
  cout << "je suis " << p.prenom << " " << p.nom << endl;
  cout << "j'ai " << p.age << " ans" << endl;
}

void anniversaire( Personne & p ) {
  p.age++;
}
```

Fichier.cc

Programmation Impérative Exemple

```
struct Personne {
  string prenom;
  string nom;
  int age;
} ;
```

Structure Personne

```
void main() {
  Personne jpp;
  initialise( jpp, "Jean-Pierre","Papin", 50 );
  quiSuisJe( jpp );
  anniversaire( jpp );

  // ...

  Personne zz;
  zz.prenom = "Zinedine";
  zz.nom = "Zidane";
  zz.age = 40;
  zz.age++;
}
```

ProgPrincipal.cc

Programmation Impérative Problèmes

- **Variables non encapsulées** : on peut lire ou modifier ces variables partout
- **Variables mal initialisées :** inconsistances de certaines valeurs
- **Traitements séparés :** connaissance imprécise de toutes les opérations possibles sur le type ; définition de traitements incohérents avec la sémantique du type

Classe : Définitions

- Une ***Classe*** est un type de donnés dont le rôle est de rassembler sous un même nom à la fois ***données*** et ***traitements***
- Les données sont appelées ***attributs***
- Les traitements sont appelés ***méthodes***

Classe : Définitions

- Une ***variable*** dont le type est une classe est appelée ***instance*** ou ***objet***.

Remarques

- Les méthodes sont définies de façon ***génériques*** (pour toutes les instances possibles de la classe) mais à l'exécution, une méthode est reliée à ***une seule instance*** (un objet courant).
- Une méthode est ***toujours*** appelée sur une instance de la classe.

Classes en C++

- En c++, on sépare l'***interface*** d'une classe et son ***corps***
- L'***interface*** (***fichier entête***) rassemble sous le nom de la classe ses attributs et les prototypes de ses méthodes. Généralement, l'interface se trouve dans le fichier *Ma_classe.h* (ou .hpp, .H, .h++).
- Le ***Corps*** de la classe contient les corps des méthodes définies dans l'interface. Généralement, le corps de la classe se trouve dans *Ma_classe.cc* (ou .cpp, .C, .c++, .cxx).

Classes en C++ : exemple

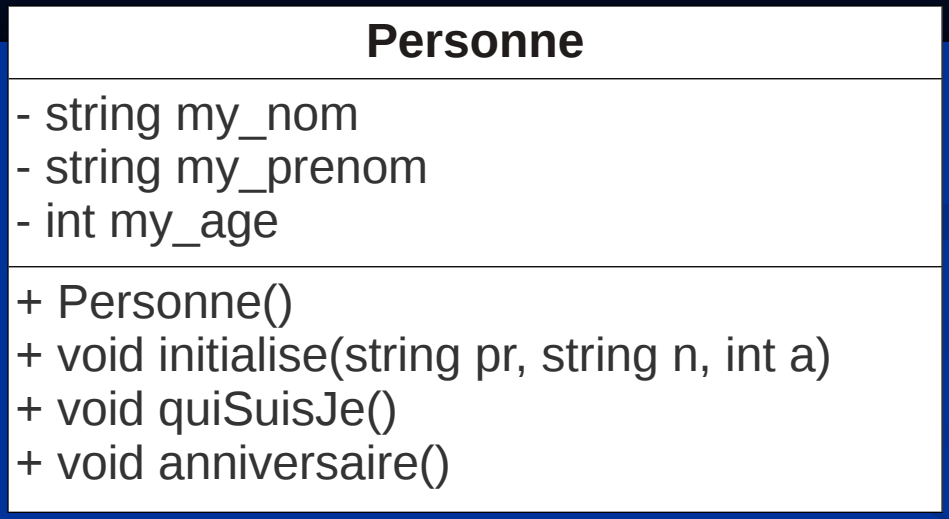

Diagramme de classe UML

Classes en C++ : exemple

Attributs

Méthodes

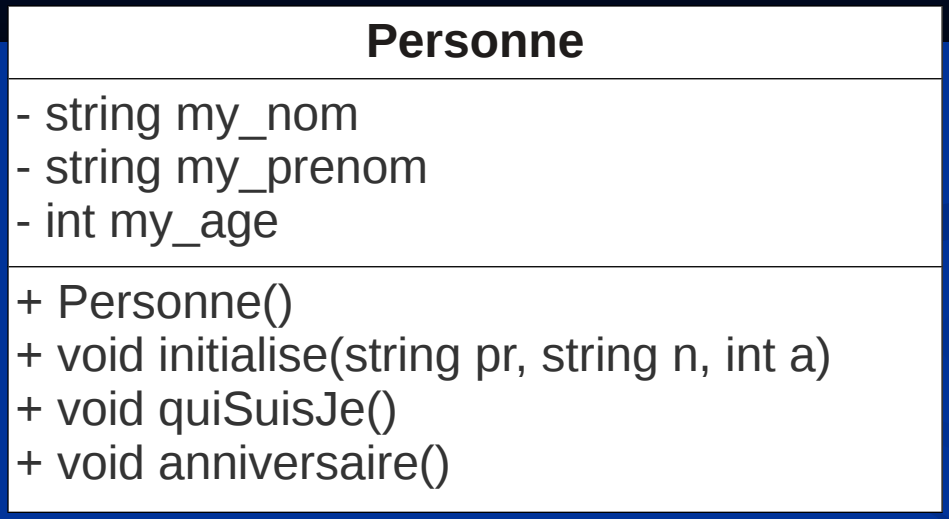

Diagramme de classe UML

Classes en C++ : exemple

```
#ifndef PERSONNE_H
#define PERSONNE_H

class Personne {
 private:
  string my_nom;
  string my_prenom;
  int my_age;

 public:
  // constructeur
  Personne();

  //méthode ou fonctions membres
  void initialise(string nom,
                  string prenom,
                  int age);
  void quiSuisJe();
  void anniversaire();
};

#endif
```

Interface : .h

attributs / prototypes
des méthodes

```
#include "Personne.h"

Personne::Personne(){
  my_nom = my_prenom = "";
  my_age = -1;
}

void Personne::initialise(string nom,
                          string prenom,
                          int age){
  my_nom = nom;
  my_prenom = prenom;
  my_age = age;
}

void Personne::quiSuisJe(){
  cout << "Je suis " << my_prenom << " " <<
    my_nom << endl;
  cout << "j'ai " << my_age << " ans" << endl;
}

void Personne::anniversaire(){
  my_age += 1;
}
```

Corps : fichier source ***.cc***
(ou .cpp, .C, .cxx)

Classes en C++ : exercice 1

- Recopier l'interface et le corps de cette classe Personne
- Créer une instance de personne et
  - Initialisez ses informations
  - Affichez ses informations
  - Modifiez son age
  - Affichez de nouveau ses informations

Programmation Impérative Exemple

- Comment compiler?
  - g++ -c Personne.cc
  - g++ -c main.cc
  - g++ main.o Personne.o -o test

Programmation Impérative Exemple

Contenu du makefile

```
CC=g++
OBJECTS= main.o Personne.o
main : $(OBJECTS)
	$(CC) $(OBJECTS) -o main
Personne.o : Personne.cc Personne.h
	$(CC) -c Personne.cc
clean :
	rm -f *~ *.o test
```

```
>make
> g++    -c -o main.o main.cpp
>g++  -c Personne.cc
>g++  main.o Personne.o -o main
```

Que fait >make clean ?

Classes en C++ : exercice

- Ecrire l'interface et le corps d'une classe Entier, qui peut être construite à partir d'un int.
- Cette classe doit permettre les opérations de base sur les entiers (+,-,*,/)

Classes en C++ : Visibilité des attributs/méthodes

Visibilité des membres d'une classe sont définies dans l'interface de la classe:

- ***public*** : (+ en UML) autorise l'accès à tous
- ***private*** : (- en UML) restreint l'accès aux seules méthodes de la classe
- ***protected*** : (# en UML) comme private sauf que l'accès est aussi autorisé aux corps des classes qui ***héritent*** de cette classe (cf Héritage)

Classes en C++ : Visibilité des attributs/méthodes

Exercice

- Essayez (depuis le main) de mettre à jour directement l'attribut « my_valeur ». Compilez. Que se passe-t-il?
- Passez l'attribut « my_valeur » en public, Compilez. Que se passe-t-il?

Classes en C++ : Accès aux membres d'un objet

Etant donnée une instance d'un objet, on accède à ses attributs et méthodes grâce à la notation :

- ***Pointée « . »*** dans le cas d'objet alloué statiquement

  ex : Personne jpp;

  ```
  ...
  jpp.quiSuisJe();
  ```

- ***Fléchée « -> »*** dans le cas d'objet alloué dynamiquement

  ex : Personne * jpp = new Personne();

  ```
  ....
  jpp->quiSuisJe()
  ```

Classes en C++ : Visibilité des attributs/méthodes

Exercice

- Instanciez statiquement un objet Entier (sans paramètre). Affichez la valeur du paramètre
- Instanciez dynamiquement un objet Entier (sans paramètre). Additionnez lui 5 et affichez la valeur du paramètre
- Instanciez dynamiquement un objet Entier (avec un paramètre). Soustrayez lui 10 et affichez la valeur du paramètre

Classes en C++ : Constructeurs/Destructeur

Constructeurs : définition de comportements particuliers lors de l'instanciation d'une classe (notamment pour initialiser un nouvel objet)

- Les ***constructeurs*** portent tous le ***nom de la classe :***
  - méthode **Personne** () pour la classe **Personne**
  - **Méthode Entier() pour la classe Entier**
- ***Plusieurs constructeurs*** différenciés par la liste et les types des paramètres

Classes en C++ : Constructeurs/Destructeur

Exemple en remplaçant la méthode initialise(...)

Personne.h

```
class Personne {
private:
   // attributs ou données membres
   string my_prenom;
   string my_nom;
   int my_age;
public:
   ...
   // Constructeur
   Personne( string pr, string n, int a );
   ...
} ;
```

Personne.cc

```
// Personne.cc
#include "Personne.h"

Personne::Personne( string pr , string n, int a ) {
   my_prenom = pr;
   my_nom = n;
   my_age = a;
}
```

Classes en C++ : Constructeurs/Destructeur

Différent constructeurs :

- par ***défaut*** ou constructeur sans paramètre
- avec ***paramètres***
- par ***copie***

Classes en C++ : Constructeurs/Destructeur

Différent constructeurs :

- par ***défaut*** ou constructeur sans paramètre :
  - appelé lors de l'instanciation d'un objet sans arguments d'appel
  - appelé lors de l'instanciation d'un tableau d'objets sur chacune des positions du tableau.
- avec ***paramètres***
- par ***copie***

Classes en C++ : Constructeurs/Destructeur

Différent constructeurs :

- par ***défaut*** ou constructeur sans paramètre :
- avec ***paramètres***
- par ***copie***
  - appelé lors de l'instanciation d'un objet avec en argument d'appel un autre objet de même type
  - appelé lors de l'appel d'une fonction avec en argument un objet *passé par valeur* ou lors du retour d'une fonction qui retourne un objet de ce type

```
Personne(const Personne &p) // dans le .h
Personne::Personne(const Personne &p){//dans le .cc
  my_nom = p.my_nom; //il faut recopier TOUS les attributs de p dans l'instance
}
```

Classes en C++ : Constructeurs/Destructeur

Différent constructeurs : Exemple

```
// CONSTRUCTEUR PAR DEFAUT
Personne pers;
Personne * pers2 = new Personne  ;
Personne groupe[5]; // appel du constructeur pour chaque personne
                    // du tableau

// CONSTRUCTEUR AVEC PARAMETRES
Personne jpp ("Papin", "Jean-Pierre", 40);

// CONSTRUCTEUR PAR COPIE
Personne zidane (jpp);
champion(zidane); // void champion(Personne pers)
Personne clone = getNewPersonne(); // Personne getNewPersonne()
```

Classes en C++ : Constructeurs/Destructeur

Le destructeur : méthode particulière définie implicitement pour toutes les classes

- Son nom : ***~nom_classe***
- un destructeur ***unique*** par classe
  - appelé lors de la destruction/désallocation de l'objet
  - nécessaire quand on fait appel à l'allocation dynamique...

Classes en C++ : Accesseurs

```
void affTab (Personne tab[], int n) {
  int i;
  for (i=0 ; i<n ; i++)
    tab[i].quiSuisJe();
}
```

Comment n'afficher que les personnes ayant plus de 18 ans ?

Classes en C++ : Accesseurs

```
void affTab (Personne tab[], int n) {
  int i;
  for (i=0 ; i<n ; i++)
    tab[i].quiSuisJe();
}
```

Comment n'afficher que les personnes ayant plus de 18 ans ?

Impossible : l'accès aux attributs (my_age,...) est interdit (***private***)

Classes en C++ : Accesseurs

```
void affTab (Personne tab[], int n) {
  int i;
  for (i=0 ; i<n ; i++)
    tab[i].quiSuisJe();
}
```

Comment n'afficher que les personnes ayant plus de 18 ans ?

Impossible : l'accès aux attributs (my_age,...) est interdit (***private***)

Les ***accesseurs*** permettent de retourner les valeurs des attributs privés nécessaires à une fonction **non membre** de la classe

Classes en C++ : Accesseurs

```
class Personne {
 private:
  string my_nom;
  string my_prenom;
  int my_age;

 public:
  ...

  //accesseur en ecriture
  void setAge(int age);
  void setPrenom(string prenom);
  void setNom(string nom);

  //accesseur en lecture
  int getAge();
  string getPrenom();
  string getNom();
```

Personne.h

Classes en C++ : Exercices

Exercice: Ecrire l'implémentation et le corps de la classe Personne (respectant les spécificités ci-dessous)

Personne
- string my_nom - string my_prenom - int my_age
+ Personne() + Personne(string nom, string prenom, int age) + Personne (const Personne &p) + ~Personne() + void setAge(int age) + void setPrenom(string prenom) + void setNom(string nom) + int getAge() + string getPrenom() + void quiSuisJe()

Faites attention à la consistance des données!

Classes en C++ : Exercices

Tests à réaliser:

- Instancier une personne pers1 sans paramètres, puis utiliser les setters pour mettre à jour ses informations
- Instancier une personne pers2 à partir de pers1
- Instancier une personne pers3 avec paramètres

Classes en C++ : Exercices

Tests à réaliser:

- Que va-t-il se passer?

```
Personne tabPers [3];
  tabPers[0]=pers1;
  tabPers[0].quiSuisJe();
  tabPers[1]=pers2;
  tabPers[1].quiSuisJe();
  tabPers[2]=pers3;
  tabPers[2].quiSuisJe();
  tabPers[0].setAge(41);
  tabPers[0].quiSuisJe();
  pers1.quiSuisJe();
  pers3.setAge(61);
  pers3.quiSuisJe();
  tabPers[2].quiSuisJe();


  Personne tabPersCopie [3]={pers1,pers2,pers3};
  tabPers[0].quiSuisJe();
  tabPers[1].quiSuisJe();
  tabPers[2].quiSuisJe();
```

Classes en C++ : Exercices

Tests à réaliser:

- Si l'attribut int my_age devient « int * my_age », Qu'est ce que cela change?

Classes en C++ : Exercices

Exercice: Ecrire l'implémentation et le corps de la classe Date (respectant les spécificités ci-dessous)

Date
- int *jour - int * mois - int *annee
+ Date() + Date(int j, int m, int a) + Date (const Date &d) + ~Date() + void setDate(int j, int m, int a) + void afficheDate() + int getJour() + int getMois() + int getAnnee()

Classes en C++ : Exercices

Tests à réaliser:

- Instancier une Date date_init au 1er janvier 2009
- Instancier une Date date_copie utilisant le constructeur par copie (et copiant date_init)
- Modifiez la date de date_init au 31 décembre 2010
- Affichez à l'écran la date de Date_init
- Affichez à l'écran la date de Date_copie (qu'observez-vous?)