2016/2017

EZ Langage : Les classes

PROFESSEUR : M. JEAN-MICHEL RICHER
REALISATEUR : M. HOSSAM-EDDINE BENHOUD

PLAN

I- LES CONSTRUCTEURS 2
II- LES DESTRUCTEURS 4
III- LA FONCTION __INIT__ 5
IV- LES ACCESSEURS (GETTERS) 7
V- LES MUTATEURS (SETTERS) 9
VI- LA SURCHARGE DES OPERATEURS 12
VII- LA SURCHARGE DES METHODES 13
VIII- LA SURCHARGE DE LA FONCTION D'AFFICHAGE « PRINT » .. 14
IX- LES DONNEES ET FONCTIONS MEMBRES STATIQUES 17
X- PROGRAMME DE DEMONSTRATION 19

I- Les constructeurs

Définition

Un constructeur est ce qui construit un objet et alloue de la mémoire. Dès qu'on crée un objet le constructeur est automatiquement appelé.

En EZ :

- Il n'y pas d'encapsulation : tous les attributs sont accessibles et par défaut publics.
- Le constructeur n'est pas besoin d'être définis dans la classe (il est créé implicitement).

Il y a deux types de constructeurs possibles:

- Par défaut: Un constructeur qui ne prend aucun argument est certainement appelé un constructeur par défaut.
- De copie : Un constructeur de copie, comme son nom l'indique, sert à copier un objet à partir d'un autre lors de la création d'un nouvel objet.

Syntaxe

```
// Déclaration de l'objet avec le constructeur par défaut
c is MaClasse

// Déclaration de l'objet avec le constructeur de copie
c2 is MaClasse(c)
```

Ci-dessous un exemple de déclaration

En EZ :

```
class Person

   nom is string
   prenom is string
   age is integer

end class

// Déclaration de l'objet
p1 is Person

// Initialisation des attributs
p.nom="dupont"
p.prenom="stephanie"
p.age=26

// Construction de l'objet à partir du constructeur de copie
p2 is Person(p1)
```

Traduction en C++ :

```
class Person {
public:
  string nom;
  string prenom;
  int age;

 // Constructeur par défaut
  Person():nom(""),prenom(""),age(0){}

 // Constructeur de copie
  Person(const Person & p)
  :nom(p.nom),prenom(p.prenom),age(p.age){}

}
```

```
// Déclaration de l'objet
Person p1;

// Initialisation des attributs
p1.nom="dupont";
p1.prenom="stephanie";
p1.age=26;

// Construction de l'objet à partir du constructeur de copie
Person p2(p1);
```

II- Les destructeurs

Définition

Un destructeur est ce qui détruit un objet, son rôle principal est la libération de la mémoire allouée via le constructeur, aussi ce qui n'a pas été libéré durant la vie de l'objet.

En EZ :

Il n'est pas nécessaire de définir un destructeur en langage EZ car il doit être créé automatiquement lors de la génération du code C++.

Exemple de destructeur en C++ :

```
class MaClasse {
public:
 ~MaClasse(){
      cout << "destructeur de MaClasse"<<endl;
  }

};
```

III- La fonction __init__

Définition

__init__ est la méthode qui va être appelée automatiquement après qu'un objet ait créé. Ce n'est pas un constructeur mais c'est un initialiseur.

L'utilité :

Le langage EZ interdit la déclaration du constructeur dans la classe, mais dans certains cas les développeurs préfèrent écrire des instructions ou des tests au moment de la construction de l'objet. Par exemple:

- Incrémenter une variable statique qui permet de calculer le nombre d'objet créé.
- Ecrire dans un fichier de journalisation pour garder un historique et enregistrer les opérations du logique métier pendant le fonctionnement d'une application au moment de la construction de l'objet.

Syntaxe

```
__init__(arg1,arg2,..)
 // Instruction
end

// ou

__init__()
 // Instruction
end init

// Remarque: On peut terminer par "end"  ou "end init"
```

En EZ :

```
class Person

   // L'initialisation de la variable statique est obligatoire
   compteur is shared integer=0
   nom is string
   prenom is string

__init__()
    Person.compteur++
 end init

end class
```

Traduction en C++ :

```
class Person
{
public:

  string nom;
  string prenom;
  static  int compteur;

  Person():nom(""),prenom(""){
      __init__();
  }

  void __init__(){
      compteur++;
  }

};
int Person::compteur=0;
```

IV- Les accesseurs (Getters)

Définition

Un accesseur est une fonction membre permettant de récupérer le contenu d'une donnée.

En EZ il n y a pas de notion d'encapsulation, ainsi les accesseurs sont très utile pour rajouter des instructions ou des tests supplémentaires aux attributs de la classe.

Exemple

En EZ :

```
class Person

    nom is string
    prenom is string

        function get_nom() return string
            return nom.toUpperCase()
        end function

         function get_prenom() return string
            return nom.toLowerCase()
         end function

end class

//classe main en ez

program main

procedure main()

        p is Person
        p.nom=”dupont”
        p.prenom=”andre”
```

```
    print "nom : "   ,  p.get_nom(), ”\n”
    print "prenom :" , p.get_prenom() , “\n”

end procedure

// output
nom: DUPONT
prenom : ANDRE
```

Traduction en C++ :

```
class Person {


public:

  string nom;
  string prenom;

  Person():nom(""),prenom(""){}


  string get_nom(){

       string s=nom;
       transform(s.begin(), s.end(), s.begin(), ::toupper);
       return s;
  }


  string get_prenom(){

       string s=prenom;
       transform(s.begin(), s.end(), s.begin(), ::toupper);
       return s;
  }

};
```

```
int main()
{
  Person p;
  p.nom="dupont";
  p.prenom="andre";

  cout<<"nom: " <<p.nom<<endl;
  cout<<"prenom: "<<p.prenom<<endl;
}

// output
nom: DUPONT
prenom : ANDRE
```

V- Les mutateurs (Setters)

Définition

Un mutateur est une fonction membre permettant de modifier le contenu d'une donnée membre protégée.

En EZ il n'y a pas de notion d'encapsulation, ainsi les mutateurs sont très utile pour rajouter des conditions avant de muter une variable de la classe.

Exemples

En EZ :

```
class Person
    nom is string
    prenom is string
    age is integer

        procedure set_nom(_nom is string)
            nom= _nom.toLowerCase()
```

```
        end procedure


        procedure set_prenom(_prenom is string)
            prenom= _prenom.toLowerCase()
        end procedure

        procedure set_age(_age is integer)

          if _age <=0  or _age>100 then
              print “l’age n’est pas correct ! “
          else
              age = _age
          end if
        end procedure

end class

//classe main en ez
program main

procedure main()

        p is Person
        p.nom=”duPOnt”
        p.prenom=”anDRe”
        p.age=26

       print "nom  "   ,  p.nom, ”\n”
    print "prenom " , p.prenom , “\n”
       print "age " , p.age , “\n”

end procedure


// output
nom  dupont
prenom  andre
age 26
```

Traduction en C++ :

```
class Person {
public:
  string nom;
  string prenom;
  int age;

  Person():nom(""),prenom(""),age(0){}

  void set_nom(string _nom){
   transform(_nom.begin(), _nom.end(), _nom.begin(),
::tolower);
       nom=_nom;
  }

 void set_prenom(string _prenom){
     transform(_prenom.begin(), _prenom.end(), _prenom.begin
::tolower);
     prenom=_prenom;
  }

 void set_age(int _age){

      if(_age <= 0 || _age >100){
         cout<<"l'age n'est pas correct ! "<<endl;
      }else{
         age=_age;
      }
  }

};


int  main(){

  Person p;
  p.set_nom("duPOnt");
  p.set_prenom("anDRe");
  p.set_age(26);
```

```
  cout<<"nom " <<p.nom<<endl;
  cout<<"preniom "<<p.prenom<<endl;
  cout<<"age “<<p.age<<endl;
}


// output
nom  dupont
prenom  andre
age 26
```

VI- La surcharge des opérateurs

Définition

La redéfinition d’un opérateur se fait en déclarant et définissant une méthode ayant pour nom « operator » suivi de l’opérateur.

Exemple 1

En EZ :

```
operator== (c is C)  return  bool
end
```

Traduction en C++ :

```
bool operator==(C const & c) const{
}
```

Exemple 2

En EZ :

```
operator< (c is C)  return  bool
end
```

Traduction en C++ :

```
bool operator<(C const & c) const{
}
```

Exemple 3

En EZ :

```
operator= (c is C)  return  C
end
```

Traduction en C++ :

```
C const & operator=(C const & c){
}
```

VII- La surcharge des méthodes

Il est possible de déclarer et définir plusieurs fonctions ayant le même nom, à condition que leurs arguments soient différents.

Exemple

```
//Méthode 1

function add() return integer
      v is integer
      v = 1 + 2
      return v
end function


//Méthode 2
function add(a,b are integer) return integer
      v is integer
      v = a + b
      return v
end function


//Méthode 3
function add(a,b,c are integer) return integer
      v is integer
      v = a + b + c
      return v
end function
```

VIII- La surcharge de la fonction d'affichage « print »

Définition

Le mot-clé print permet de faire une sortie d'affichage. Il est toutefois possible de le redéfinir dans une classe selon la sortie souhaitée.

Exemple

En EZ :

```
class Person
   nom is string
   prenom is string
   age is integer

   procedure print()
      print “nom: ” , nom,  “  prenom:”, prenom, “  age: ”, age
   end procedure

end class


//main
program main

procedure main()

// Déclaration de l’objet
p is Person
p.nom="dupont"
p.prenom="laurent"
p.age=25

print p

end procedure

//ouput
nom: dupont  prenom: laurent  age: 25
```

Traduction en c++:

```
class Person {

 public:
  string nom;
  string prenom;
  int age;

  Person():nom(""),prenom(""),age(0){
  }

  friend ostream & operator<<(ostream & os, Person const &
p){
      os <<"nom: "<<p.nom<<" prenom: "<<p.prenom<<
" age : "<<p.age<<endl ;

  return os ;
  }

} ;


int main() {

   Person p ;
   p.nom="laurent";
   p.prenom="andre";
   p.age=25 ;

   cout<<p<<endl ;


   return 0;
}
//ouput
nom: dupont  prenom: laurent  age: 25
```

IX- Les données et fonctions membres statiques

Définition

Une fonction membre déclarée static a la particularité de pouvoir être appelée sans devoir instancier la classe.
Elle ne peut utiliser que des variables et des fonctions membres statiques.

Exemple 1

En EZ :

```
class MaClasse
        // L'initialisation de la variable statique est obligatoire
        a is shared integer = 0
end class
```

Traduction en C++ :

```
class Exemple {
public:
  static int a;
};

// dans le fichier .cpp
int Exemple::a = 0;
```

Exemple 2

En EZ :

```
class A
  // non static
   var1 is integer
   procedure f1()
   end procedure

  //static
var2 is shared integer = 0

shared procedure f2()
end procedure

end class

// function main
procedure main()

  // non static
   a is A
   a.var1 = 1
   a.f1()
  //static
   A.var2 = 1;
   A.f2();

end procedure
```

En C++ :

```
class A
{
public:
  // non static
```

```
  int var1;
  void f1() {};
  //static
  static int var2;
  static void f2() {};
};
// initialisation de la variable static
int A::var2 = 0;
int main()
{
  // non static
  A a;
  a.var1 = 1;
  a.f1();

  //static

  A::var2 = 1;
  A::f2();
}
```

X- Programme de démonstration

En EZ :

```
/* LA CLASSE Employee  */

class Employee

        counter is shared integer = 0
_id is integer
        _nom is string
_prenom is string
        _age is integer
_salaire is real
        _anciennete is integer
```

```
/* la méthode qui va être appelée automatiquement
après qu'un objet ait créé. Ce n'est pas un
constructeur mais c'est un initialiseur */
__init__()

Employee.counter++
_id = Employee.counter

end init

/*cette procedure permet de faire une sortie
d'affichage. Il est toutefois possible de le
redéfinir dans une classe selon la sortie
souhaitée.*/
procedure print()

print "  Id:"_id,"  Nom:",_nom , "Prenom:" ,
_prenom , "Age:",_age,"Anciennete:",_anciennete,"
Salaire:",_salaire

end procedure


end class


/* LA CLASSE Employees  */

class Employees

/* vecteur des employées */
  _list is vector of Employee

/* cette procedure permet d'ajouter un employée*/
procedure ajouter(emp is Employee )
_list.put_first(emp)
end procedure


/*cette procedure permet de modifier un
employée*/
```

```
procedure modifier(emp is Employee)

for i in _list.range()
if _list[i]._id == emp._id then
_list[i]=emp
end if
end for
end procedure


/*Cette procedure permet de supprimer un
employée*/
procedure supprimer_par_employee(emp is Employee)

for i in _list.range()

if _list[i]._id == emp._id then

_list.remove(i)
end if
end for
end procedure

/*cette procedure permet de supprimer un employée
par id*/
procedure supprimer_par_id(id is integer)

for i in _list.range()

if _list[i]._id == id then

_list.remove(i)
end if
end for
end procedure

/*cette fonction permet de retourner l'employée
le plus ancien*/
```

```
function employee_plus_ancien() return Employee
if _list.size() > 0 then
        Employee emp=_list[0]
for i in _list.range()
if _list[i]._anciennete > emp._anciennete then
emp = _list[i]
end if
end for
return emp
end if
end function


/*cette fonction permet de retourner l'employée
le moins ancien*/
function  employee_moins_ancien() return Employee
if _list.size()>0 then
        Employee emp=_list[0]
for i in _list.range()
if _list[i]._anciennete < emp._anciennete then
emp = _list[i]
end if
end for
return emp
```

```
end if

end function


/*cette procedure permet de trier les employées
par age*/
procedure trier_employee_par_age()

_list.sort(_age)

end procedure

/*cette procedure permet de trier les employées
par salaire*/
procedure trier_employee_par_salaire()

_list.sort(_salaire)
end procedure


/*cette procedure permet de faire une sortie
d'affichage. Il est toutefois possible de le
redéfinir dans une classe selon la sortie
souhaitée.*/
procedure print()

for i in _list.range()
print _list[i]
end for

end procedure


end class
```

```
/* LA CLASSE MAIN  */

        program main

procedure main()


// déclaration des employées
emp1 is Employee
        emp1._nom="dupont"
emp1._prenom="andré"
emp1._age=23
emp1._anciennete=5
emp1._salaire=1800

emp2 is Employee
        emp2._nom="zuckerberg"
emp2._prenom="mark"
emp2._age=25
emp2._anciennete=10
emp2._salaire=2800.5

emp3 is Employee
        emp3._nom="deperois"
emp3._prenom="david"
emp3._age=23
emp3._anciennete=7
emp3._salaire=3600.5


// Objet employees
employees is Employees

// ajout des employées
        employees.ajouter(emp1)
employees.ajouter(emp2)
employees.ajouter(emp3)


// afficher tous les employées
employees.print()
```

```
//afficher l'employée le plus ancien

print "L'employée le plus
ancien:",employees.employee_plus_ancien()
print "L'employée le moins
ancien:",employees.employee_moins_ancien()


//trier les employées par ages
print "\nTrier les employées par age"
employees.trier_employee_par_age()

// afficher tous les employées
employees.print()


//trier les employées par salaire
print "\nTrier les employées par salaire"
employees.trier_employee_par_salaire()

// afficher tous les employées
employees.print()

//supprimer par id
id is integer = 3
print "\nSupprimer par id = ",id
        employees.supprimer_par_id(id)

// afficher tous les employées
employees.print()

//supprimer par employée
print "\nSupprimer l'employée 1"
employees.supprimer_par_employee(emp1)


// afficher tous les employées
employees.print()

end procedure
```

Traduction en c++:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


class Employee{

public:

	static int counter;
	int _id;
	string _nom;
	string _prenom;
	int _age;
	float _salaire;
	int _anciennete;


Employee():_nom(""),_prenom(""),_age(0),_salaire(
0),_anciennete(0){

	   __init__();

	}

	void __init__(){
	   Employee::counter++;
	   _id=Employee::counter;
	}


friend std::ostream& operator<<(std::ostream& os,
const Employee& emp){
 os<<"  Id:"<< emp._id<<"  Nom:"<<emp._nom<<"
Prenom:"<<emp._prenom<<"  Age:"<<emp._age<<"
Anciennete:"<<emp._anciennete<< "
Salaire:"<<emp._salaire<<endl;
```

```
 return os;
}

};

int Employee::counter=0;

class Employees{

public:

      vector<Employee> _list;



      void ajouter(Employee emp){
          _list.push_back(emp);
      }



      void modifier(Employee emp){

         for(int i=0;i<_list.size();i++){

            if(_list[i]._id == emp._id){

                _list[i]=emp;
            }
         }
      }

      void supprimer_par_employee(Employee emp){

         for(int i=0;i<_list.size();i++){

            if(_list[i]._id == emp._id){

                  _list.erase(_list.begin()+i);
              }
         }
      }
```

```
      void supprimer_par_id(int  id){

            for(int i=0;i<_list.size();i++){

               if(_list[i]._id == id){


_list.erase(_list.begin()+i);
                     }
            }
      }


      Employee employee_plus_ancien(){


       if(_list.size()>0){

          Employee emp=_list[0];

          for(Employee tmp: _list){

             if(tmp._anciennete>emp._anciennete){
                emp=tmp;
             }

          }
          return emp;

       }
           return Employee();

      }

      Employee employee_moins_ancien(){

           if(_list.size()>0){

             Employee emp=_list[0];

             for(Employee tmp: _list){


if(tmp._anciennete<emp._anciennete){
```

```
                    emp=tmp;
                }

            }
            return emp;

        }

        return Employee();

    }

    void trier_employee_par_age(){

        sort(_list.begin(),_list.end(),[](const
Employee  & emp1, const Employee & emp2){return
emp1._age < emp2._age;});
    }

    void trier_employee_par_salaire(){
        sort(_list.begin(),_list.end(),[](const
Employee  & emp1, const Employee & emp2){return
emp1._salaire < emp2._salaire;});

    }

    friend std::ostream&
operator<<(std::ostream& os, const Employees&
obj)
    {
        for(Employee emp : obj._list){
            os<<emp;
        }
        return os;
    }
};


int main() {


   // déclaration des employées
   Employee  emp1;
   emp1._nom="dupont";
```

```
    emp1._prenom="andré";
    emp1._age=23;
    emp1._anciennete=5;
    emp1._salaire=1800;

    Employee emp2;
    emp2._nom="zuckerberg";
    emp2._prenom="mark";
    emp2._age=25;
    emp2._anciennete=10;
    emp2._salaire=2800.5;

    Employee emp3;
    emp3._nom="deperois";
    emp3._prenom="david";
    emp3._age=23;
    emp3._anciennete=7;
    emp3._salaire=3600.5;



    // Objet employees
    Employees employees;

    // ajout des employées
    employees.ajouter(emp1);
    employees.ajouter(emp2);
    employees.ajouter(emp3);


    // afficher tous les employées
    cout<<employees<<endl;

    //afficher l'employée le plus ancien

    cout<<"L'employée le plus
ancien:"<<employees.employee_plus_ancien();
    cout<<"L'employée le moins
ancien:"<<employees.employee_moins_ancien();


    //trier les employées par ages
       cout<<"\nTrier les employées par
age"<<endl;
```

```
        employees.trier_employee_par_age();
    // afficher tous les employées
        cout<<employees<<endl;



    //trier les employées par salaire
        cout<<"\nTrier les employées par
salaire"<<endl;
        employees.trier_employee_par_salaire();

    // afficher tous les employées
        cout<<employees<<endl;

    //supprimer par id
        cout<<"\nSupprimer par id=3"<<endl;
        int id=3;
        employees.supprimer_par_id(id);

    // afficher tous les employées
    cout<<employees<<endl;

    //supprimer par employée
    cout<<"\nSupprimer l'employée 1"<<endl;
    employees.supprimer_par_employee(emp1);


    // afficher tous les employées
    cout<<employees<<endl;


    return 0;
}
```