EZ Langage : Les classes

PLAN

1. 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);
```

2. 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;
   }

};
```

3. 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 de la logique métier pendant le fonctionnement d'une application au moment de la construction de l'objet.

Syntaxe

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

 // or
__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;
```

4. 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
```

5. 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
```

6. 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.

Exemples

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){

}
```

7. 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
```

8. 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.

Exemples

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()

    p is Person(“dupont”,”laurent”,25)
    print p

end procedure

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

9. 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 static.

Exemples

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 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();
}
```

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
```