AsciiDoc et Asciidoctor : Exemple en C++

Sommaire

1. Asciidoc . . . 1
2. Exemple C++ . . . 1
2.1. Création d'une classe . . . 1
2.2. Instanciation . . . 3

1. Asciidoc

AsciiDoc est un langage de balisage léger proche du langage *Markdown*, proposant une richesse sémantique plus importante.

Syntaxe : http://asciidoctor.org/docs/asciidoc-syntax-quick-reference/

Markdown vs Asciidoc

Comme Markdown, les fichiers `.adoc` (extension par défaut pour des fichiers Asciidoc) utilisent une syntaxe simple à écrire et à lire. Voici un comparatif entre les deux formats : http://asciidoctor.org/docs/user-manual/#compared-to-markdown

Lien : https://asciidoc.org/

2. Exemple C++

Comme Markdown, le format AsciiDoc est très apprécié et utilisé par les développeurs pour rédiger notamment des documentations.

L'exemple ci-dessous montre une utilisation d'AsciiDoc dans le cadre d'un développement en **C++**.
Le fichier en Asciidoc : exemple-asciidoc.adoc

Une syntaxe est directement dédiée pour intégrer du code source : https://asciidoctor.org/docs/asciidoc-syntax-quick-reference/#source-code

2.1. Création d'une classe

Déclaration d'une classe `Point` en C++

Le code C++ peut être directement intégré dans le fichier AsciiDoc.

Point.h

```
class Point
{
   private:
      double x;
      double y;

   public:
      Point(); ①
      Point(double x, double y);

      // Accesseurs
      double getX() const;
      void setX(double x);
      double getY() const;
      void setY(double y);
      // Service(s)
      void affiche() const;
};
```

① Constructeur par défaut

Constructeur par défaut

Le constructeur qui est appelé lorsqu'aucun paramètre est fourni lors de l'instanciation.

Définition d'une classe `Point` en C++

Le code peut être directement intégré à partir d'un fichier C++.

Point.cpp

```
#include "Point.h"
#include <iostream>

using namespace std;

// Constructeurs
Point::Point() : x(0.), y(0.)
{
    cout << "Je suis le constructeur de la classe Point ! " << this << endl;
}

Point::Point(double x, double y) : x(x), y(y)
{
    // ou :
    /*
    this->x = x;
    this->y = y;
```

```
    */
    cout << "Je suis le constructeur de la classe Point ! " << this << endl;
}

// Destructeur
Point::~Point()
{
    cout << "Je suis le destructeur de la classe Point ! " << this << endl;
}

// Accesseurs et mutateurs
double Point::getX() const
{
   return x;
}

void Point::setX(double x)
{
   this->x = x;
}

double Point::getY() const
{
   return y;
}

void Point::setY(double y)
{
   this->y = y;
}

// Services
void Point::affiche() const
{
   cout << "<" << x << "," << y << ">" << endl;
}
```

2.2. Instanciation

En utilisant des *tags*, il est possible d'intégrer des fragments de code. Voir l'utilisation des *tags* dans le fichier source testPoint.cpp.

Exemple d'utilisation de la classe `Point` :

Constructeur par défaut

```
Point p0;
```

Constructeur d'initialisation

```
Point p1(1., 2.);
Point p2(4., 0.);
```

Accesseurs et mutateurs (getter/setter)

```
cout << " L'abscisse de p0 est " << p0.getX() << endl;
cout << " L'abscisse de p1 est " << p1.getX() << endl;
cout << " L'abscisse de p1 est " << p2.getX() << endl;
cout << endl;

cout << " L'ordonnée de p0 est " << p0.getY() << endl;
cout << " L'ordonnée de p1 est " << p1.getY() << endl;
cout << " L'ordonnée de p1 est " << p2.getY() << endl;
cout << endl;

p0.setX(5.5);
cout << " L'abscisse de p0 est maintenant " << p0.getX() << endl;
p0.setY(6.5);
cout << " L'ordonnée de p0 est maintenant " << p0.getY() << endl;
cout << endl;
```

Tableau d'objets

```
Point tableauDe10Points[10]; // typiquement : les cases d'un tableau de Point

cout << "Un tableau de 10 Point : " << endl;
for(int i = 0; i < 10; i++)
{
   cout << " P" << i << " = <" << tableauDe10Points[i].getX() << "," <<
tableauDe10Points[i].getY() << ">\n";
}
cout << endl;
```

Les services d'une classe

```
    cout << "p0 = ";
    p0.affiche();
    cout << "p1 = ";
    p1.affiche();
    cout << "p2 = ";
    p2.affiche();
    cout << endl;

    cout << "Allocation dynamique" << endl;
    Point *p3; // je suis pointeur non initialisé sur un objet de type Point
    p3 = new Point(2. , 2.); // j'alloue dynamiquement un objet de type Point
    //Point *p3 = new Point(2. , 2.); // cette écriture est conseillée

    // Comme p3 est une adresse, je dois utiliser l'opérateur -> pour accéder aux
membres de cet objet
    cout << " L'abscisse de p3 est " << p3->getX() << endl;
    p3->setY(0); // je modifie la valeur de l'attribut y de p3
    (*p3).setY(1); // cette écriture est possible : je pointe l'objet puis j'appelle sa
méthode
    cout << " L'ordonnée de p3 est " << p3->getY() << endl;
    cout << " L'adresse de p3 est " << p3 << endl;
    delete p3; // ne pas oublier de libérer la mémoire allouée pour cet objet
    p3 = nullptr;
    cout << endl;

    return 0;
}
```

Allocation dynamique d'objet

```
Point *p3; // je suis pointeur non initialisé sur un objet de type Point
p3 = new Point(2. , 2.); // j'alloue dynamiquement un objet de type Point
//Point *p3 = new Point(2. , 2.); // cette écriture est conseillée

// Comme p3 est une adresse, je dois utiliser l'opérateur -> pour accéder aux membres
de cet objet
cout << " L'abscisse de p3 est " << p3->getX() << endl;
p3->setY(0); // je modifie la valeur de l'attribut y de p3
(*p3).setY(1); // cette écriture est possible : je pointe l'objet puis j'appelle sa
méthode
cout << " L'ordonnée de p3 est " << p3->getY() << endl;
cout << " L'adresse de p3 est " << p3 << endl;
delete p3; // ne pas oublier de libérer la mémoire allouée pour cet objet
p3 = nullptr;
cout << endl;
```

Site : tvaira.free.fr