

## Conception à base de patrons II

### 1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Visitor », « State ». Cette implémentation est effectuée à l'aide du logiciel Visual Studio et le langage C++ sera utilisé tout au long du processus de développement.

### 2 - Patron Visiteur (50 points)

L'application commande du système de PolyScino est constituée d'une ou plusieurs commandes. Les commandes permettent d'orienter, tourner et de prendre des photos, et de répéter un ensemble de commandes dans une boucle. Le patron Visiteur est implémenté pour parcourir les commandes organisées dans un patron composite et produire un résultat correspondant à la fonction spécifique du visiteur.

#### Implémentation

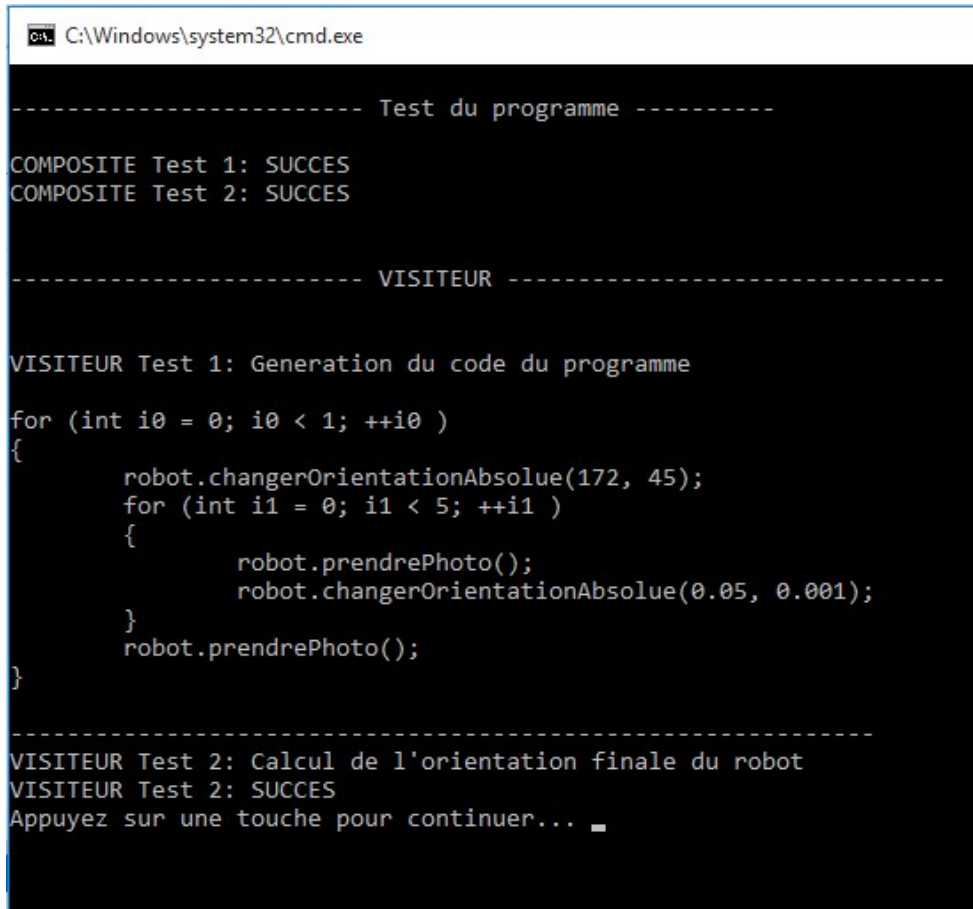
Trois classes ont été ajoutées au projet afin d'implémenter des fonctionnalités sous formes de visiteurs :

1. VisiteurAbs : classe abstraite qui définit une interface de base pour tous les visiteurs concrets qui peuvent être utilisés avec les objets Composites dérivés de la classe abstraite CmdAbs.
2. VisiteurGenCPP: cette classe implémente, sous la forme d'un visiteur, la génération de code C++ correspondant au programme représenté par le composite des commandes exécutées sur le robot.
3. VisiteurCalcOrientation: cette classe implémente, sous la forme d'un visiteur, le résultat du calcul de l'orientation finale du robot sans appliquer les commandes.

On vous demande de compléter le code des méthodes correspondant aux visiteurs concrets, dans les fichiers sources `VisiteurGenCPP.cpp` et

VisiteurCalcOrientation.cpp afin que les tests programmés dans la méthode `Test_TP5::executeVisitorTest()` s'exécutent avec succès.

Le résultat des tests des patrons visiteurs est comme suit :



```
C:\Windows\system32\cmd.exe

----- Test du programme -----

COMPOSITE Test 1: SUCCES
COMPOSITE Test 2: SUCCES

----- VISITEUR -----

VISITEUR Test 1: Generation du code du programme
for (int i0 = 0; i0 < 1; ++i0 )
{
    robot.changerOrientationAbsolue(172, 45);
    for (int i1 = 0; i1 < 5; ++i1 )
    {
        robot.prendrePhoto();
        robot.changerOrientationAbsolue(0.05, 0.001);
    }
    robot.prendrePhoto();
}

-----
VISITEUR Test 2: Calcul de l'orientation finale du robot
VISITEUR Test 2: SUCCES
Appuyez sur une touche pour continuer... _
```

## Questions à répondre

- 1) Identifiez la structure des classes réelles qui participent aux patrons « Visiteur » ainsi que leurs rôles (faite un diagramme de classes avec Enterprise Architect pour chaque visiteur, ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).
- 2) Quels changements doit-on réaliser si on ajoute les nouvelles commandes « activerFlash » et « desactiverFlash » ?

### 3 - Patron State (50 points)

Le robot de notre système PolyScino offre des comportements différents en fonction de son état. Lorsque le robot est en état Normal, les méthodes d'exécution du programme devraient fonctionner normalement. Lorsque le robot est en état erreur, les méthodes d'exécution du programme ne devraient pas fonctionner.

#### Implémentation

Trois classes ont été ajoutées au projet afin d'implémenter les fonctionnalités liées à l'état du robot :

- « EtatRobotAbs »: cette classe abstraite définit une interface de base pour tous les états que peut avoir le robot.
- « EtatRobotNormal » et
- « EtatRobotErreur » les classes qui implémentent les états du robot.

**IMPORTANT** : chaque méthode qui peut modifier le robot devrait appeler la méthode "verifierEtat" pour s'assurer de l'état du robot et le changer au besoin.

On vous demande de compléter les fichiers `EtatRobotNormal.cpp` et `EtatRobotErreur.cpp` afin que les tests programmés dans la méthode `Test_TP5::executeProxyTest()`, `Test_TP5::executeCompositeTest()` et `Test_TP5::executeTemplateMethodTest()` s'exécutent avec succès.

#### Questions à répondre

- 1) Identifiez la structure des classes réelles qui participent au patron ainsi que leurs rôles (faite un diagramme de classes avec Enterprise Architect, ajouter des notes en UML pour indiquer les rôles, et exportez le tout en pdf).
- 2) Pourquoi a-t-on ajouté la méthode protégée « `changerLongitudeLatitude` » dans la classe abstraite « `EtatRobotAbs` » ?

## 4 – À remettre

- 1) Une archive LOG2410\_TP5\_matricule1\_matricule2.zip qui contient les éléments suivants :
  - a) Le fichier `ReponsesAuxQuestions.pdf` avec la réponse aux questions 2.2, et 3.2.
  - b) Le fichier `DiagrammeDeClasses_Visitor.pdf` pour le diagramme de classes des deux patrons composite de la question 2.1.
  - c) Le fichier `DiagrammeDeClasses_State.pdf` pour le diagramme de classes des deux patrons composite de la question 3.1.
  - d) Les fichiers C++ que vous avez modifiés, c'est-à-dire,  
*VisiteurGenCPP.cpp,*  
*VisiteurCalcOrientation.cpp,*  
*EtatRobotNormal.cpp,*  
et *EtatRobotErreur.cpp,*

Vous ne pouvez pas modifier les autres fichiers `.h` et `.cpp`. Le correcteur va insérer vos fichiers dans le code, et ça doit compiler et s'exécuter.