






# TP 15 – Contrôle 1

 L'objectif de ce TP est de construire un ensemble de classe de gestion d'angles.

## Exercice 0 Mise en place du TP

-  Téléchargez le fichier **VEtudiant.zip** sur l'ENT et décompressez-le. Il contient la solution Visual Studio vous permettant de construire les classes.
-  Le projet contient le fichier **main.cpp** qui décrit un programme de test. Tous les tests sont préalablement écrits, **VOUS NE DEVEZ SOUS AUCUN PRÉTEXTE MODIFIER LES TESTS**.
-  Vous devrez décommenter au fur et à mesure ce fichier au cours de ce TP pour appliquer la méthodologie TDD. Pour aller plus vite, vous pouvez décommenter les tests par bloc. Les blocs sont séparés par des lignes vides.

 Les diagrammes UML proposés dans les exercices suivants ne proposent que le minimum nécessaire. Vous pouvez, si besoin, ajouter des données et/ou des fonctions membres.

## Exercice 1 Implémentation de la classe CDoubleApprox

En calcul numérique, des erreurs d'arrondis peuvent parfois apparaître. Par exemple, le calcul mathématique de  $\sqrt{2} * \sqrt{2}$  vaut naturellement 2. Mais l'exécution du code équivalent **sqrt(2.0)\*sqrt(2.0)** donne la valeur 2.0000000000000004. Cette petite imprécision fait que le test « **sqrt(2.0)\*sqrt(2.0) == 2.0** » retourne « **false** », là où nous aurions probablement préféré « **true** ».

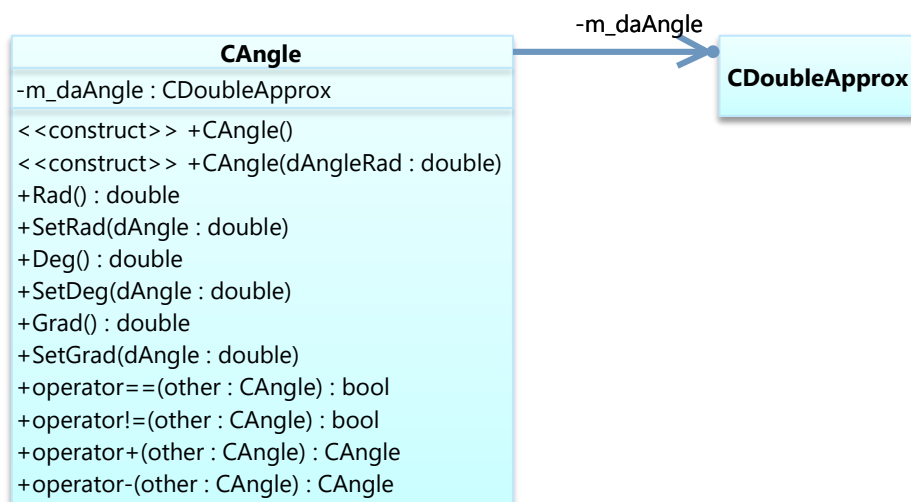
Pour pallier ce problème, l'idée de ce premier exercice est d'implémenter une classe **CDoubleApprox** qui représentera une valeur réelle « **double** » mais dont les opérateurs de comparaison seront vrais à une tolérance près. Nous fixerons cette tolérance à  $10^{-10}$ . Le diagramme UML de la classe **CDoubleApprox** est représenté sur la figure ci-contre.

<b>CDoubleApprox</b>
-m_dVal : double
<<construct>> +CDoubleApprox(dVal : double)
+operator==(other : CDoubleApprox) : bool
+operator!=(other : CDoubleApprox) : bool
+operator>(other : CDoubleApprox) : bool
+operator<(other : CDoubleApprox) : bool
+operator>=(other : CDoubleApprox) : bool
+operator<=(other : CDoubleApprox) : bool
+operator-() : CDoubleApprox
+operator+() : CDoubleApprox
+operator*(other : CDoubleApprox) : CDoubleApprox
+operator+(other : CDoubleApprox) : CDoubleApprox
+operator-(other : CDoubleApprox) : CDoubleApprox
+operator/(other : CDoubleApprox) : CDoubleApprox
+operator*=(other : CDoubleApprox) : CDoubleApprox&
+operator+=(other : CDoubleApprox) : CDoubleApprox&
+operator-=(other : CDoubleApprox) : CDoubleApprox&
+operator/=(other : CDoubleApprox) : CDoubleApprox&
+to_double() : double

- ➡ Données membres :
    - ➡ **m\_dVal** : la valeur du double
  - ➡ Fonctions membres :
    - ➡ **Constructeur avec un paramètre** : Crée l'objet correspondant à la valeur du double passé en paramètre.
    - ➡ **to\_double** : Fonction de conversion vers un **double**.
  - ➡ Opérateurs
    - ➡ **operator==** : retourne **true** si les **CDoubleApprox** sont identiques à  $10^{-10}$  près. C'est-à-dire que la différence entre les deux valeurs (en valeur absolue) doit être inférieure ou égale à  $10^{-10}$ . Vous pouvez vous aider de la fonction **std::fabs()** de la bibliothèque standard (incluse dans le fichier **<cmath>**)
    - ➡ **operator!=** : opérateurs complémentaires de l'opérateur **==**.
    - ➡ **operator<**, **operator>**, **operator<=** et **operator>=** : opérateurs de comparaison usuels, respectant les règles imposées aux opérateurs **==** et **!=**.
    - ➡ **-** et **+** unaires, **\***, **+**, **-**, **/**, **\*=**, **+=**, **-=**, **/=** : opérateurs mathématiques usuels.
  - ➡ Fonctions non-membres mais en relation avec la classe **CDoubleApprox** :
    - ➡ Tous les opérateurs binaires équivalents à ceux qui sont membres de la classe mais dont l'opérande de gauche est un **double** et l'opérande de droite un **CDoubleApprox**.
- 📁 Implémentez la classe **CDoubleApprox** selon la méthodologie TDD en suivant le plan de tests proposé dans le fichier **main.cpp**.

## Exercice 2 Implémentation de la classe CAngle

La classe **CAngle** gère... un angle. Elle propose une conversion automatique des angles dans les différents systèmes d'unité : radians, degrés et gradians. Par ailleurs, elle assure que l'angle soit exprimé dans la plage  $[-\pi, \pi[$  et ses correspondances dans les autres unités. Afin de permettre un changement d'unité simplifié, en interne de la classe, l'angle est stocké dans la plage  $[-1, 1[$ . Avec cette astuce, exprimer l'angle en radian revient simplement à multiplier cette valeur interne par  $\pi$ , l'exprimer en degré ou gradian consiste à multiplier la valeur interne respectivement par 180 ou 200. Voici le diagramme UML de la classe **CAngle** :

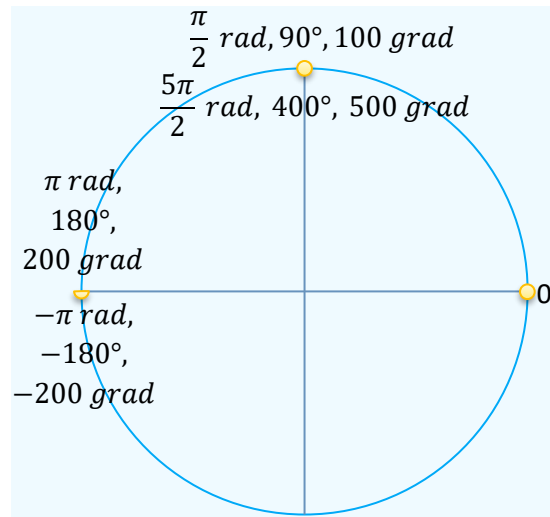


Données membres :

- ➔ **m\_daAngle** : Valeur stockant l'angle compris dans la page  $[-1,1[$ .

Fonctions membres :

- ➔ **Constructeur par défaut** : Crée un angle nul.
- ➔ **Constructeur avec un paramètre** : Crée un angle dont la valeur passée en paramètre est l'angle exprimé en radian.
- ➔ **Rad** : Accesseur en lecture de l'angle exprimé en radian (une valeur interne de  $-1$  doit retourner  $-\pi$ ).
- ➔ **SetRad** : Accesseur en écriture de l'angle exprimé en radian (une valeur définie à  $-\pi$  doit être stockée comme  $-1$ ). Un angle passé en dehors de la plage  $[-\pi, \pi[$  doit être rapporté à cette plage. Par exemple, l'angle  $\frac{5\pi}{2}$ , équivalent à  $\frac{\pi}{2}$  doit être stockée avec la valeur  $\frac{1}{2}$ .
- ➔ **Deg** : Accesseur en lecture de l'angle exprimé en degré (une valeur interne de  $-1$  doit retourner  $-180$ ).
- ➔ **SetDeg** : Accesseur en écriture de l'angle exprimé en degré (une valeur définie à  $-180$  doit être stockée comme  $-1$ ). Un angle passé en dehors de la plage  $[-180, 180[$  doit être rapporté à cette plage. Par exemple, l'angle  $400^\circ$ , équivalent à  $90^\circ$  doit être stockée avec la valeur  $\frac{1}{2}$ .
- ➔ **Grad** : Accesseur en lecture de l'angle exprimé en gradian (une valeur interne de  $-1$  doit retourner  $-200$ ).
- ➔ **SetGrad** : Accesseur en écriture de l'angle exprimé en degré (une valeur définie à  $-200$  doit être stockée comme  $-1$ ). Un angle passé en dehors de la plage  $[-200, 200[$  doit être rapporté à cette plage. Par exemple, l'angle  $500$  grad, équivalent à  $100$  grad doit être stockée avec la valeur  $\frac{1}{2}$ .



Opérateurs

- ➔ **==, !=** : opérateurs usuels de comparaison entre deux angles.
- ➔ **+, -** : opérateurs mathématiques usuels entre deux angles.

Fonctions non-membres mais en relation avec la classe **CAngle** :

- ➔ Tous les opérateurs binaires équivalents à ceux qui sont membres de la classe mais dont l'opérande de gauche est un **double** et l'opérande de droite un **CDoubleApprox**.
- ➔ **Opérateur de sortie vers flux** : doit permettre d'afficher l'angle par la syntaxe `std::cout << angle;` si `angle` est un objet de type **CAngle**. La sortie doit être « *angle\_rad (angle\_deg° / angle\_grad grad)* » où *angle\_rad*, *angle\_deg* et *angle\_grad* sont respectivement la valeur de l'angle exprimée en radian, degré et gradian.

Implémentez la classe **CAngle** selon la méthodologie TDD en suivant le plan de tests proposé dans le fichier **main.cpp**.