TD-TP : *POLYMORPHISME*

1. Une classe et une sous-classe – instances simples

Etant donnée la classe `Sportif` avec un `nom` et un `prénom` du type `string`, plus une `anneeNaissance` de type entier, avec les méthodes `get&set`, `toString()` et `getAge()`

Etant donnée la classe `SportifMenteur`, une sous-classe de `Sportif` pour laquelle la méthode `getAge()` retourne 18 ans si l'âge réel est inférieur à 18 ans ☹.

Note : Pour cela vous utiliserez la classe `Outils` dans laquelle vous disposerez de la méthode de classe :

```
static short int  anneeActuelle ()
```

qui retourne l'année en cours.

Outils	
ATTRIBUTS	*-non-*
CONSTRUCTEUR	*-non-*
ENCAPSULATION	*-non-*
METHODES USUELLES	*-non-*
METHODES SPECIFIQUES static short int **anneeActuelle ()**	

Travail à faire : créer un projet **`Polymorphisme`** dans **`VSCode`**

1.1. Classe `Outils`

Ajoutez au projet la classe **`Outils`** avec le fichier **`Outil.h`** fourni en **Annexe**, puis écrivez un **`main`** qui affiche l'année actuelle.

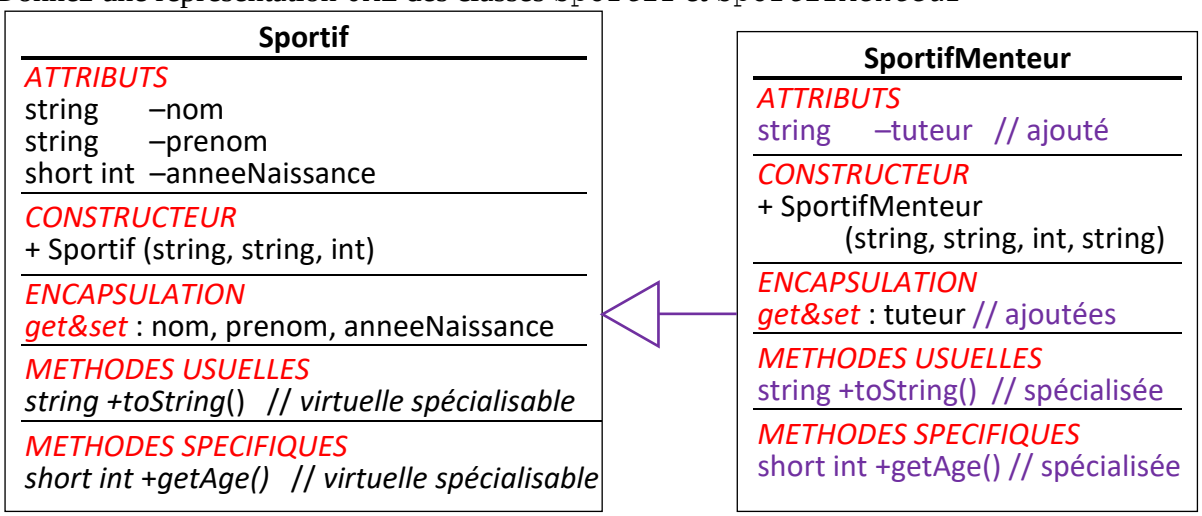
```
#include <iostream>
#include "Outils.h"

using namespace std;

int main() {
    cout << Outils::anneeActuelle() << endl;
    return 0;
}
```

1.2. Représentation `UML`

Donnez une représentation `UML` des classes `Sportif` et `SportifMenteur`



1.3. Codez les classes `Sportif` et `SportifMenteur`

Sur la base de `Individu` (super classe) et `Etudiant` (sous classe) présents sur **eLearn**, donnez le code des classes `Sportif` et `SportifMenteur`.
Attention aux méthodes `Sportif::getAge()` et `SportifMenteur::getAge()`

```
#include "SportifMenteur.h"

SportifMenteur::SportifMenteur // Constructeur
(string nom, string prenom, short int annee, string tuteur)
 : Sportif (nom, prenom, annee)   // Délègue la construction/init
                                  // des attributs à la super-classe
{
    setTuteur (tuteur);
}

string SportifMenteur::toString() { // Spécialise toString()
    string message;
    // Invoque le toString() de la super-classe Sportif
    message = this-> Sportif::toString();
    message += "  -  tuteur : " + getTuteur();  // Complète le message
    return message;
}

short int SportifMenteur::getAge() { // Spécialise getAge()
    short int age;
    // Invoque le getAge() de la super-classe Sportif
    age = this->Sportif::getAge(); // Obtient l'âge réel
    return (age<18 ? 18 : age);
}
```

1.4. Codez le `main()` qui

- Crée un Sportif `sportif1` né en 2007, un SportifMenteur `sportif2` né en 2000 et un SportifMenteur `sportif3` né en 2010.

```
Sportif          sportif1 ("DUPONT", "Pierre",  2007);
SportifMenteur   sportif2 ("MARTIN", "Jean",    2000, "ZIDANE");
SportifMenteur   sportif3 ("DURAND", "Jacques", 2010, "DESCHAMPS");
```

- Affiche l'année en cours, puis pour chaque individu affiche sa date de naissance ainsi que l'âge qu'il dit avoir.

```
cout << Outils::anneeActuelle() << endl;

cout << sportif1.getAnneeNaissance() << "-" <<  sportif1.getAge() << "ans\n";
cout << sportif2.getAnneeNaissance() << "-" <<  sportif2.getAge() << "ans\n";
cout << sportif3.getAnneeNaissance() << "-" <<  sportif3.getAge() << "ans\n";
```

2. Polymorphisme = `virtual` + par référence ou par adresse

Pour que dans une méthode, fonction ou procédure, un objet récupéré en paramètre exécute la méthode de la classe dont il est instance (ex. `getAge()`), le programmeur doit s'assurer de deux précautions :

(1) Le paramètre récupéré ne doit pas être un paramètre par valeur mais un paramètre par référence ou par adresse.

(2) La méthode spécialisée dans la classe dont il dépend, doit être déclarée du genre `virtual` dans la super-classe.

Travail à faire : adapter le projet `Plymorphisme`

2.1. Si ce n'est déjà fait, préfixer la déclaration de la méthode `Sportif::getAge()` du mot clef `virtual`.

```
virtual short int getAge ();
```

2.2. Ecrire dans le fichier **`main.ccp`** trois fonctions globales :

```
void afficheAgeSportifParValeur    (Sportif  sportif);
void afficheAgeSportifParReference (Sportif& sportif);
void afficheAgeSportifParAdresse   (Sportif* sportif);
```

telles que chacune affiche :

- l'année en cours,
- la date de naissance du Sportif considéré,
- l'age du Sportif considéré.

```
void afficheAgeSportifParValeur (Sportif sportif) {
    cout << "Ne en " << sportif.getAnneeNaissance ()
         << " dit avoir " << sportif.getAge () << " ans\n";
};

void afficheAgeSportifParReference (Sportif& sportif) {
    cout << "Ne en " << sportif.getAnneeNaissance ()
         << " dit avoir " << sportif.getAge () << " ans\n";
};

void afficheAgeSportifParReference()ParAdresse (Sportif* sportif) {
    cout << "Ne en " << sportif->getAnneeNaissance ()
         << " dit avoir " << sportif->getAge () << " ans\n";
};
```

2.3. Ecrire un `main` qui appelle ces trois fonctions pour chacun des individus `sportif1`, `sportif2` et `sportif3` créés en **1.4**

```
  // Usage d'une fonction par valeur. Le polymorphisme ne marche pas
  cout << "Par valeur\n";
  afficheAgeSportifParValeur (sportif1);
  afficheAgeSportifParValeur (sportif2);
  afficheAgeSportifParValeur (sportif3);

  // Usage d'une fonction par référence. Le polymorphisme marche
  cout << "Par reference\n";
  afficheAgeSportifParReference (sportif1);
  afficheAgeSportifParReference (sportif2);
  afficheAgeSportifParReference (sportif3);

  // Usage d'une fonction par adresse. Le polymorphisme marche
  cout << "Par adresse\n";
  afficheAgeSportifParAdresse (&sportif1);
  afficheAgeSportifParAdresse (&sportif2);
  afficheAgeSportifParAdresse (&sportif3);
```

ANNEXE Outils.h

```
#ifndef OUTILS_H
#define OUTILS_H

#include <time.h>
class Outils {
    /// ATTRIBUTS -non-

    /// METHODES
    public:
        // CONSTRUCTEUR      -non-
        // DESTRUCTEUR       -non-
        // ENCAPSULATION     -non-
        // METHODES USUELLES -non-

        // METHODE SPECIFIQUE : anneeActuelle
        // Méthode de classe qui retourne l'année actuelle
        static short int anneeActuelle () {
            time_t     timer;     // stocke l'heure  actuelle
            struct tm  *newTime;  // pointe sur une structure

            // demande l'heure que l'on récupère à l'adresse de timer
            time (&timer);
            newTime = localtime(&timer); // décompose timer dans 1 structure
            return (newTime->tm_year + 1900); // retourne l'heure actuelle
        }
    protected: // -non-
    private:   // -non-
};
#endif
```