Modélisation orientée objet en C++

par Thierry Vaira © v.1.00

Ce sujet comprend 19 questions pour un total de 40 points.

Nom : _____

La partie "Modélisation orientée objet en C++" regroupe notamment les savoirs suivants :
- Paradigme objet, concepts : abstraction de données, objets, classes, généralisation, spécialisation, ...
- Interfaces et implémentations, niveaux de protection
- Caractérisation des objets : identité, état, comportement
- Communication entre objets, catégories de messages : constructeurs, destructeurs, sélecteurs, modificateurs, itérateurs
- Synchronisation des messages : synchrone, asynchrone, ...
- Relations entre classes : association, agrégation, composition

A. Modélisation orientée objet en C++

A..1 Concepts

La modélisation orientée objet consiste en la définition et l'interaction de briques logicielles appelées **objets**.

Un objet représente un **concept** comme une voiture, une personne etc ...

Un objet possède une **structure interne**, un **état** et un **comportement**.

Un objet sait **interagir** avec d'autres objets.

A..2 Classe

Une classe **déclare des propriétés communes** à un ensemble d'objets.

Une classe représente une **catégorie d'objets**.

Une classe apparaît comme un **type** à partir duquel il sera possible de créer des objets.

A..3 Objet

Un objet possède une **structure interne** composée :
- de **variables** appelées des attributs ou des membres données
- de **fonctions** appelées des méthodes ou des fonctions membres

Un objet possède un **état** : c'est la valeur de ses attributs.

Un objet possède un **comportement** : ce sont les méthodes qui forment son interface. L'exécution des méthodes d'un objet modifiera son état.

Un objet posséde une **identité** qui permet de distinguer un objet d'un autre objet (son nom, une adresse mémoire).

Un objet est créé à partir d'un **modèle** appelé **classe**.

Chaque objet créé à partir de cette classe est une **instance** de la classe en question.

A..4 Protection

Le C++ permet de préciser le **type d'accès des membres** (**attributs** et **méthodes**) d'un objet.

Cette opération s'effectue au sein des classes de ces objets. On distingue 3 types de visibilité :
- **public** : les membres publics peuvent être utilisés dans et par n'importe quelle partie du programme.
- **privé** (*private*) : les membres privés d'une classe ne sont accessibles que par les objets de cette classe et non par ceux d'une autre classe.
- **protégé** (*protected*) : les membres protégés d'une classe sont accessibles par les objets de cette classe ou d'une classe héritée de cette classe.

L'unité de protection en C++ est la classe. Ceci implique qu'une fonction membre peut accéder à tous les membres de n'importe quel objet de sa classe.

Il est possible d'instancier des **objets constants**. Dans ce cas, il ne sera pas possible de modifier la valeur de ses attributs.

A..5 Encapsulation

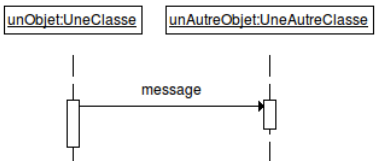
L'encapsulation est l'idée de **protéger les attributs d'un objet** et de ne proposer que des méthodes pour les manipuler.

En respectant ce principe, tous les attributs d'une classe seront donc privées (ou protégées).

Les méthodes publiques qui permettent d'accéder directement à un attribut s'appellent des **accesseurs** :
– l'accesseur `getX()` de l'attribut `x` est un **sélecteur** (accès en lecture)
– l'accesseur `setX()` de l'attribut `x` est un **modificateur** (accès en écriture)

A..6 Messages

Les objets interagissent entre eux en **s'échangeant des messages**.



La réponse à la réception d'un message par un objet est appelée **une méthode**. Une méthode **est donc la mise en oeuvre du message** : elle décrit la réponse qui doit être donnée au message.

L'ensemble des messages forme ce que l'on appelle l'**interface de l'objet**.

On distingue deux types de message :
– **synchrone** (flèche pleine) : l'émetteur se bloque en attendant la réponse du récepteur du message
– **asynchrone** (flèche) : l'émetteur n'attend pas la réponse du récepteur du message et continue son activité

A..7 Constructeur et destructeur

Le constructeur d'une classe est une méthode qui est appelée automatiquement au moment de la création de l'objet.

Le constructeur est chargé d'**initialiser un objet de la classe**.

Le constructeur porte le même nom que la classe et il n'a jamais de type de retour.

Il existe un **constructeur par défaut** qui initialise une instance lorsqu'aucun autre constructeur fourni n'est applicable.

A..8 Destructeur

Le destructeur est une méthode qui appelée automatiquement lorsqu'une instance (objet) de classe cesse d'exister en mémoire.

Le destructeur est chargé de **libérer toutes les ressources** qui ont été acquises par cet objet (typiquement libérer la mémoire qui a été allouée dynamiquement).

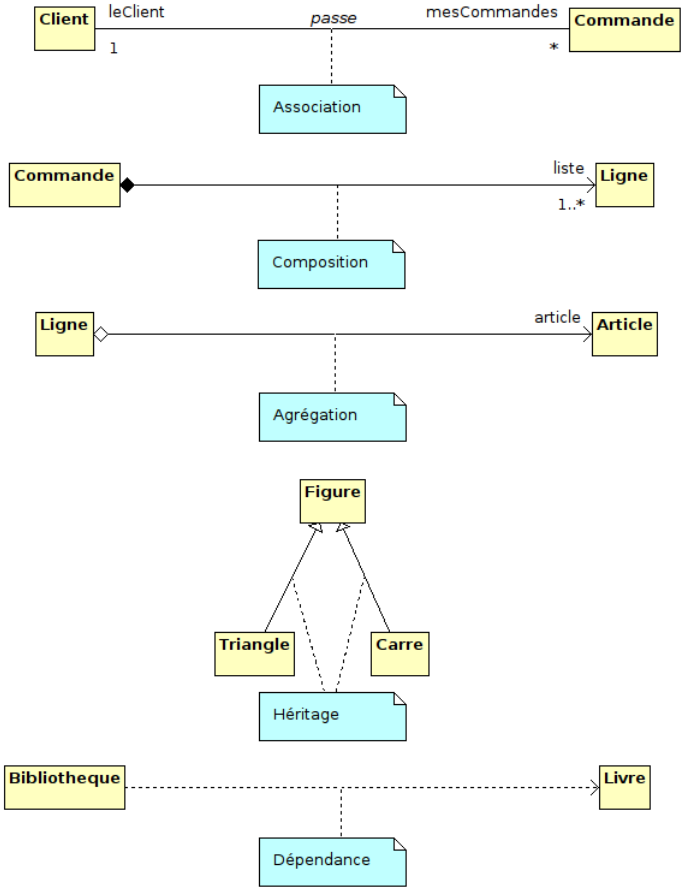
Un destructeur est une méthode qui porte toujours le même nom que la classe, précédé de "~".

Un destructeur est unique. Il ne possède aucun paramètre et n'a jamais de type de retour.

A..9 Relations

Les **relations** entre les classes permettent aux objets d'interagir entre eux.

On distingue cinq différents types de **relations** de base entre les classes :
- les relations de type « avoir » (au sens « a », « contient », « possède ») :
  - l'**association** (trait plein avec ou sans flèche)
  - la **composition** (trait plein avec ou sans flèche et un losange plein)
  - l'**agrégation** (trait plein avec ou sans flèche et un losange vide)
  - la **dépendance** (flèche pointillée)
- les relations de type « être » (au sens « est un ») :
  - la relation de **généralisation** ou d'**héritage** (flèche fermée vide)



La flèche permet de préciser le sens de la navigabilité (« qui **connaît** qui »). Lorsqu'il n'y a pas de flèche, la relation est bidirectionnelle.

Il existe :
- des relations **durables** : association, composition, agrégation, héritage et
- des relations **temporaires** : dépendance.

Aux extrémités d'une association, agrégation ou composition, il est possible d'y indiquer une **multiplicité** (ou **cardinalité**) : c'est pour préciser le nombre d'instances (objets) qui participent à la relation. Une multiplicité peut s'écrire : `n` (exactement `n`, un entier positif), `n..m` (`n` à `m`), `n..*` (`n` ou plus) ou `*` (plusieurs).

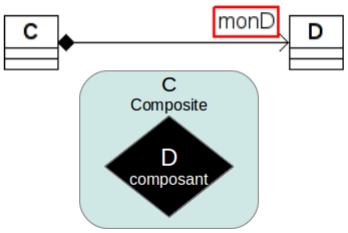
UML

Association

C++

```
class E
{
   private:
      F * monF;
};

class F
{
   private:
      E * monE;
};
```

UML

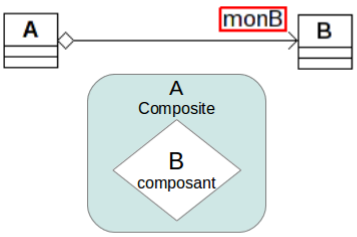


Composition

C++

```
class C
{
   private:
      D monD;
};

class D
{

};
```

UML



Agrégation

C++

```
class A
{
   private:
      B * monB;
};

class B
{

};
```

UML

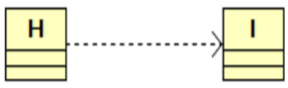
Héritage

C++

```
class A
{

};

class B : public A
{

};

B::B() : A()
{
}
```

Appel du constructeur parent

UML



Dépendance

Généralement, les dépendances ne sont pas montrées dans un diagramme de classes car ce sont des relations temporaires.

C++

```
class H
{
   void traiter(I i)
   {
      I i;
      // ...
   }
};

class I
{

};
```

Objet temporaire car il n'existe que pendant la durée de l'exécution de la méthode.

A..10 Synthèse

Déclaration d'une classe

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

public:
   Point();
   ~Point();
   void Afficher();
};
```

attributs
constructeur
destructeur
méthode

Point.h

Définition d'une classe

```
Point::Point()
{
   x = 0.;
   y = 0.;
}

Point::~Point()
{
}

void Point::Afficher()
{
    std::cout << x << y;
}
```

Initialisation des données membres de la classe Point

Définition d'une fonction membre de la classe Point

Point.cpp

Instanciation d'objets

```
Point point1;

point1.Afficher();

Point *point2 = new Point;

point2->Afficher();

delete point2;
```

Allocation statique d'un objet Point

L'opérateur d'accès aux membres d'un objet est le **.**

Allocation dynamique d'un objet Point

L'opérateur d'accès aux membres d'un objet à partir d'une adresse est la **->**

Libération mémoire de l'objet Point

main.cpp

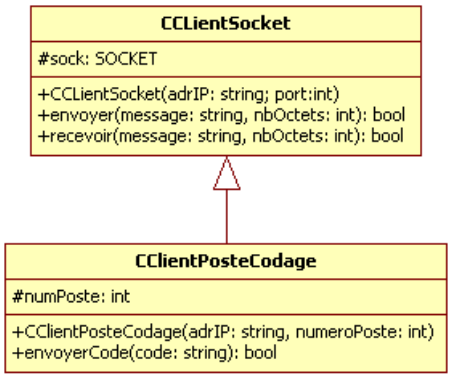
A..11 Questions

A..11.1 ESI 2014

Sur une chaîne de tri d'objets postaux, on s'interesse à l'envoi du code postal entré par l'opérateur au serveur de supervision « WCS » (*Warhouse Control System*). Ce serveur va pouvoir commander le système pour transmettre l'objet postal jusqu'au conteneur de sortie.

Le poste de codage, après avoir vérifié que le code postal est correct, doit envoyer cette information au serveur « WCS », cette communication se fait par réseau et en particulier grâce à un logiciel client sur le poste de codage.

Le diagramme de classes partiel du client est le suivant :



Question 1 (1 point)
Est-ce que la classe `CClientPosteCodage` possède un constructeur par défaut ?

Question 2 (1 point)
Quel est le type de relation entre la classe `CClientSocket` et la classe `CClientPosteCodage` ?

Question 3 (2 points)
Donner les autres types de relation possible dans un diagramme de classes UML.

Un numéro de poste est attribué à chaque poste de codage. Ce numéro est transmis au serveur « WCS » avec le code postal saisi par l'opérateur de tri.

Question 4 (3 points)
Écrire la déclaration de la classe `CClientPosteCodage` en C++.

Le port pour la classe `CClientPosteCodage` est `2345`.

Question 5 (2 points)
Écrire la définition du constructeur de la classe CClientPosteCodage en C++.

L'adresse IP du serveur « WCS » est 192.168.107.15.

Question 6 (2 points)
Instancier un objet de type `CClientPosteCodage` pour le numéro de poste `3`.

Soit la définition partielle de la méthode `envoyer()` de la classe `CClientSocket` :

```
bool CClientSocket::envoyer(const string &message, int nbOctets)
{
}
```

Question 7 (1 point)
Que signifient le caractère `&` et le mot clé `const` dans l'argument de la méthode `envoyer()` ?

Question 8 (1 point)
À quoi correspond le symbole `#` devant l'identifiant *sock* dans la classe `CClientSocket` ? Quelle est sa signification ?

A..11.2 ESI 2015

L'acier est issu d'un mélange de minerai de fer et de coke chauffé dans des hauts fourneaux. Après décapage, il est souvent fourni sou forme de bobines appelées *coils*.

Le laminage est un procédé de fabrication par déformation plastique. Il concerne différents matériaux comme le métal ou tout autre matériau sous forme pâteuse tels que le papier ou les pâtes alimentaires. Cette déformation est obtenue par compression continue au passage entre deux cylindres contrarotatifs appelés laminoir. Un laminoir est une installation industrielle ayant pour but la réduction d'épaisseur d'un matériau (généralement du métal) Il permet également la production de barres profilés (produits longs).

Les bobines sont ensuite acheminées vers des transformateurs et destinées soit :
– au découpage en feuilles (cisaillage)
– au refendage (découpe dans le sens de la longueur et reformation de bobines de plus faible largeur),
– à la fabrication de profilés reconstitués (poutrelles cornières, tés, tubes etc.)
– à l'emboutissage.

Le processus de Cisaillage utilise une balance permettant de peser le tôles découpées.

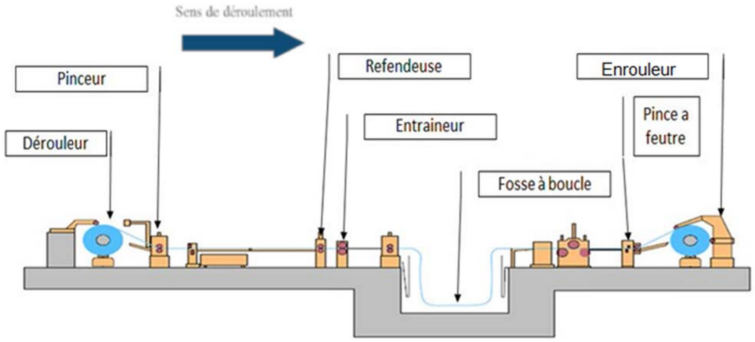


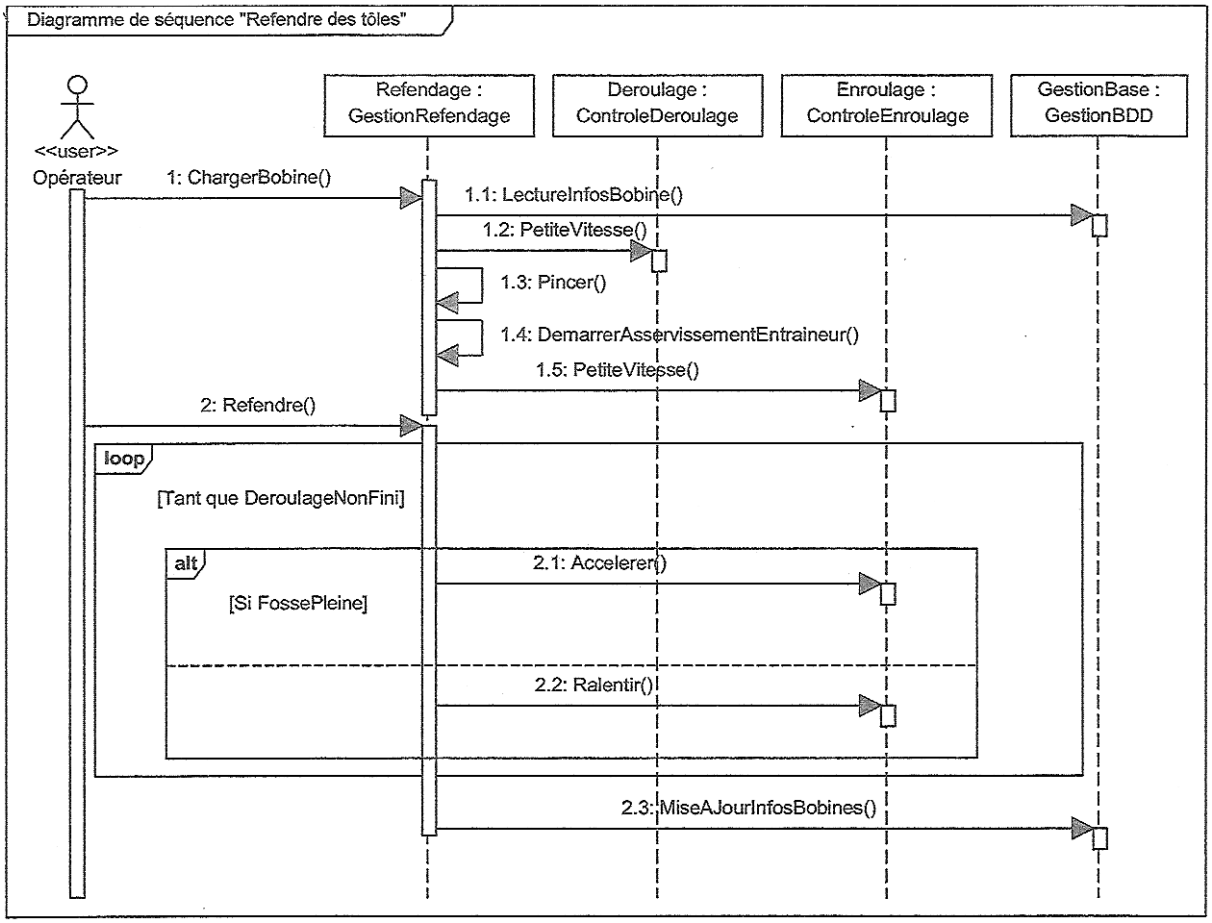
Figure 6 Schéma de la ligne de refendage.

Le refendage consiste à dérouler une bobine, la recouper dans le sens de la longueur et à la rebobiner. On peut voir, figure 6 du sujet, le schéma de la ligne de refendage.

Le scénario de refendage est le suivant :
1) une bobine à refendre est montée sur le dérouleur ;
2) les informations de la bobine sont extraites de la base de donnée ;
3) le dérouleur démarre en petite vitesse ;
4) le refendeur pince la tôle ;
5) le système d'asservissement de l'entraineur est mis en marche ;
6) l'enrouleur démarre en petite vitesse ;
7) la feuille d'acier déroulée est maintenue tendue en position horizontale entre le pinceur et l'entraineur, la refendeuse taille la feuille d'acier à la largeur souhaitée, la feuille retaillée passe par la fosse à boucle (dans cette zone tampon la feuille n'est plus sous tension). Dans la zone tampon, tant que le déroulage n'est pas terminé :
   - le dérouleur se met en grande vitesse ;
   - l'enrouleur accélère si la fosse est pleine et ralentit si la fosse n'est pas pleine.
8) A la fin de l'enroulage de la bobine, les informations sont mises à jour dans la base de données.

Remarque : Au niveau de l'entraineur, la feuille a une vitesse linéaire supérieure à celle en sortie du dérouleur afin de compenser le retard dû à l'arrêt pour le cisaillage. Les fosses à boucle servent de tampon.

Le diagramme de séquence du cas d'utilisation « Refendre des tôles » :



Document à consulter : « Annexe 3 : ».

Le système regroupant de multiples classes, elles ont été ordonnées à l'aide de packages. Les classes issues des diagrammes de séquences des processus de refendage et de cisaillage sont regroupées dans un package nommé « *Découpage* ».

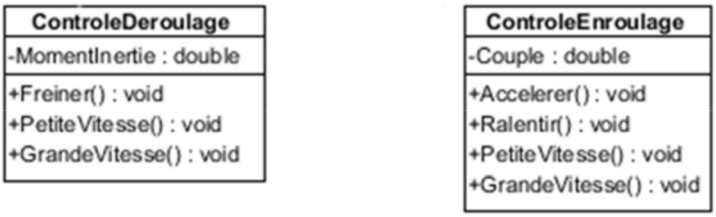


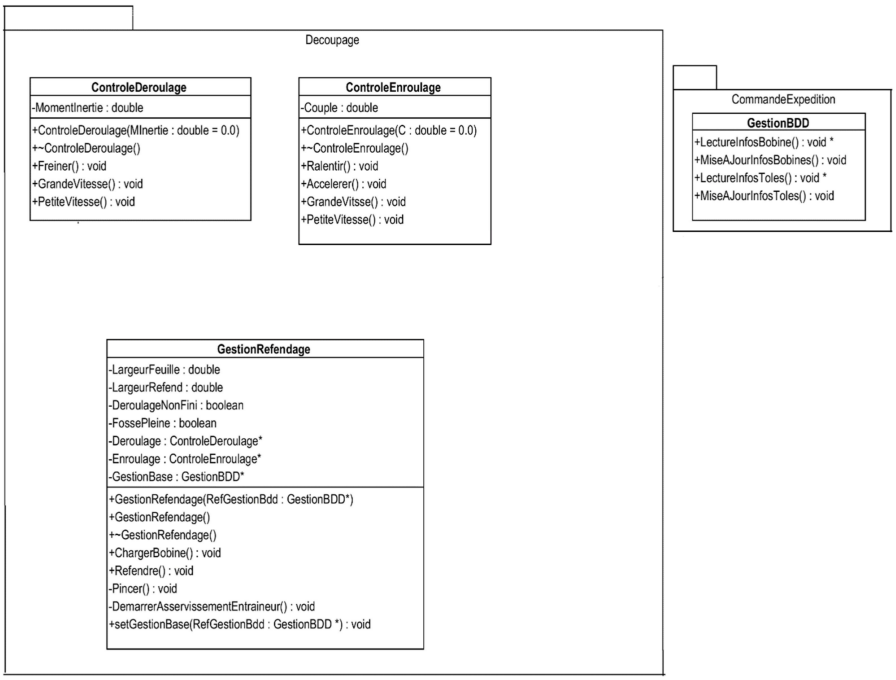
Figure 11 Extrait du package « Découpage »

Question 9 (2 points)
Dans le package « Découpage » du diagramme de classes (annexe 3 et figure 11), on remarque que les classes `ControleEnroulage` et `ControleDeroulage` ont deux méthodes communes. Proposer une solution utilisée en P.O.O pour éviter cette redondance.

Le constructeur de la classe `GestionRendage` est le suivant :

```
GestionRendage::GestionRefendage(GestionBDD *RefGestionBDD)
{
  GestionBase = RefGestionBDD;
  Deroulage = new ControleDeroulage();
  Enroulage = new ControleEnroulage();
}
```

Question 10 (6 points)
Compléter l'extrait du diagramme de classes en faisant apparaître les liaison et les cardinalités entre la classe `GestionRefendage` et les classes `ControleDeroulage` et `ControleEnroulage` ainsi que la liaison et les cardinalités entre la classe `GestionRefendage` et la classe `GestionBDD`.



Question 11 (6 points)
A partir du diagramme de séquence du processus de refendage traité à la question ci-dessus, proposer la définition en C++ de la méthode `ChargerBobine()` de la classe `GestionRefendage` du package « *Découpage* » (`gestionrefendage.cpp`).

Question 12 (1 point)
A partir du diagramme de séquence du processus de refendage traité à la question ci-dessus, préciser si les messages échangés sont synchrones ou asynchrones.

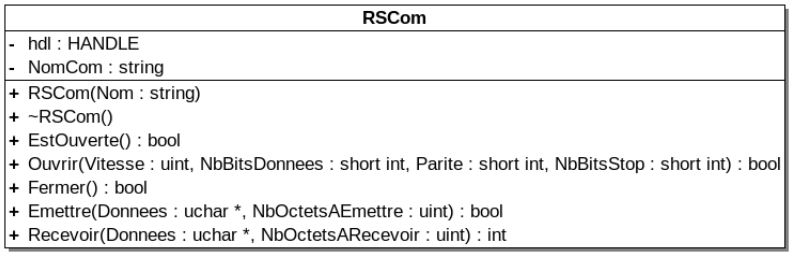
A..11.3 ESI 2011

La société GSI a été sollicitée par un groupement d'organisations sportives pour qu'elle leur développe un système informatisé de gestion de bout en bout de leurs manifestations sportives : des courses à pied. Ce système a été nommé **CAP-Chrono**.

Chaque coureur est muni d'un transpondeur électronique sans contact, transpondeur RFID, couramment nommé « puce ». La mise en oeuvre d'une zone de détection-lecture au niveau de la ligne d'arrivée de la course permet l'identification informatisée du coureur et, en conséquence, la mesure de sa performance (temps et classement) de façon automatique et instantanée.

Les lecteurs RFID disposent d'une liaison série RS232 ou RS485/422 pour communiquer avec un ordinateur. Afin de les relier facilement à un même PC de chronométrage (ou PC « Arrivées ») il a été décidé d'utiliser un concentrateur RS232 ←→ USB qui permet une communication en USB, coté PC. Au bilan, chaque lecteur est « vu » par le PC comme une liaison série.

L'accès aux ports série pour la communication avec les lecteurs de transpondeurs est réalisé à partir de la classe `RSCom` dont le diagramme de classe UML est donné ci-dessous.

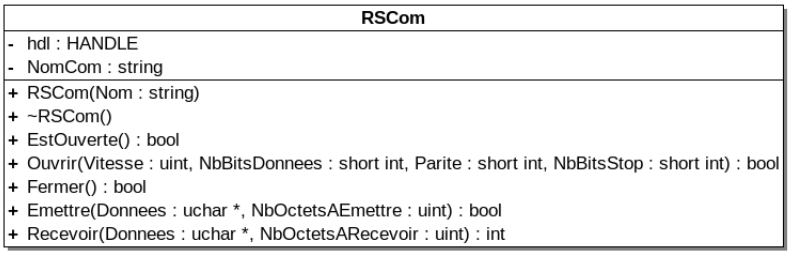


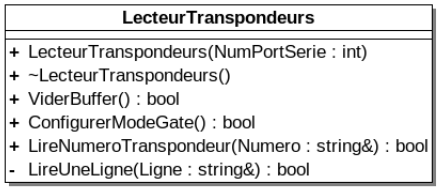
La déclaration en C++ de la classe `LecteurTranspondeurs` est donnée ci-dessous.

```
class LecteurTranspondeurs
{
   private:
      RSCom PortSerie;
      bool LireUneLigne(string& Ligne);

   public:
      LecteurTranspondeurs(int NumPortSerie);
      ~LecteurTranspondeurs();
      bool ViderBuffer();
      bool ConfigurerModeGate();
      bool LireNumeroTranspondeur(string& Numero);
};
```

Question 13 (2 points)
Donner le nom du type de relation (association, composition ou agrégation) qui existe entre les classes `LecteurTranspondeurs` et `RSCom`.

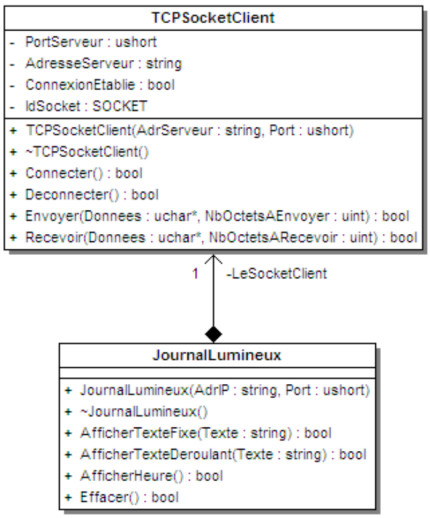
Question 14 (4 points)
Compléter le diagramme de classes en dessinant la relation existant entre les classes `LecteurTranspondeurs` et `RSCom`. Faire apparaître le nom de rôle, la navigabilité et la cardinalité de la relation.





Une évolution du système permettra de diffuser de l'information à destination du public à partir d'un journal lumineux. Cet afficheur alphanumérique possède une interface Ethernet et peut être piloté en lui envoyant des commandes. Dans la configuration utilisée, le journal lumineux est serveur et l'application le pilotant est cliente ; le protocole de communication est basé sur le protocole TCP/IP.

Pour faciliter le pilotage de ce journal lumineux, l'architecture logicielle (partielle) suivante est mise en place :



Les méthodes de la classe `TCPSocketClient` retournent `true` en cas de réussite ou `false` sinon. Elles mettent à jour l'attribut `ConnexionEtablie` pour que celui-ci reflète en permanence l'état de la connexion avec le serveur.

Question 15 (1 point)
L'attribut `ConnexionEtablie` est-il accessible à partir de la classe `JournalLumineux` ? Expliquer pourquoi.

Une méthode permettant d'accéder en lecture à l'attribut `ConnexionEtablie` à partir de la classe `JournalLumineux` doit être ajoutée à la classe `TCPSocketClient`.

Question 16 (1 point)
Proposer en C++ le prototype de cette méthode.

Question 17 (1 point)
Justifier la visibilité de cette méthode.

Question 18 (1 point)
Coder cette méthode en C++.

Question 19 (2 points)
Faudrait-il ajouter une méthode publique permettant d'accéder en écriture à l'attribut `ConnexionEtablie` de la classe `TCPSocketClient` à partir de la classe `JournalLumineux` ?

A..12 Annexe 3

Annexe 3 : Diagramme de classes partiel du système