



Ce document a été numérisé par le CRDP de Bordeaux pour la Base Nationale des Sujets d'Examens de l'enseignement professionnel.

Campagne 2010

Ce fichier numérique ne peut être reproduit, représenté, adapté ou traduit sans autorisation.

E4D : ÉTUDE DE CAS

Durée : 5 heures

Coefficient : 5

CAS NOIXCOOP

Ce sujet comporte 15 pages dont 7 pages d'annexes.

Il est composé de quatre dossiers indépendants.

Le candidat est invité à vérifier qu'il est en possession d'un sujet complet.

Matériels et documents autorisés

- Lexique SQL sans commentaire ni exemple d'utilisation des instructions.

Tous les types de calculatrice sont INTERDITS pour cette épreuve.

Liste des annexes :

- Annexe 1 : 1A Les calibres de noix
1B Exemple de livraison*
- Annexe 2 : 2A Éléments de coûts
2B Exemple d'utilisation du service web
2C Diagramme de classes partiel
2D Classes métier
2E Classe Collection*
- Annexe 3 : Schéma du réseau de la coopérative*

Barème :

Dossier 1 : Distribution de noix	32 points
Dossier 2 : Commercialisation de plants de noyers	16 points
Dossier 3 : Suivi des commandes	40 points
Dossier 4 : Mise en production du service	12 points

Total : 100 points

CODE ÉPREUVE : ISE4D	EXAMEN : BREVET DE TECHNICIEN SUPERIEUR	SPÉCIALITÉ : INFORMATIQUE DE GESTION Option Développeur d'applications
SUJET	ÉPREUVE : ÉTUDE DE CAS	
Coefficient : 5	Code sujet : 10DA02N	Page : 1/15

Présentation du contexte

NOIXCOOP est une coopérative agricole spécialisée dans la collecte, la transformation et le conditionnement de la noix qualifiée "noix de Grenoble". Les membres de la coopérative sont des producteurs situés dans la vallée de l'Isère (affluent du Rhône).

Dossier 1	Distribution de noix
------------------	-----------------------------

Documents à utiliser : annexes 1A et 1B.

NOIXCOOP est engagée dans une démarche qualité et souhaite améliorer la traçabilité de ses produits. Elle envisage donc de réorganiser la partie de son système d'information relative à la gestion des approvisionnements et des ventes.

Les producteurs

Les membres de la coopérative sont des producteurs de noix, appelés nuciculteurs. Chaque année, la coopérative accepte, dans une certaine proportion, la production de producteurs non adhérents.

Les producteurs qui sont engagés dans la lutte pour le respect de l'environnement obtiennent des certifications garantissant leur engagement dans ce domaine (par exemple les labels Agriculture Biologique et GLOBALGAP). Pour les producteurs adhérents, on conserve une trace des différentes certifications qu'ils possèdent (certification et date d'obtention).

La coopérative souhaite conserver dans son système d'information certaines caractéristiques des producteurs avec lesquels elle travaille : nom et adresse de la société, nom et prénom du responsable de production. Pour les producteurs adhérents, il faut également conserver la date de leur adhésion.

Les vergers

Un producteur peut exploiter plusieurs vergers. Parmi les caractéristiques d'un verger, on trouve la variété de la noix, la superficie plantée et le nombre d'arbres à l'hectare.

Certains vergers permettent au producteur d'obtenir les labels AOC (appellation d'origine contrôlée), avec l'appui du Comité Interprofessionnel de la Noix de Grenoble.

Pour avoir droit à l'appellation d'origine contrôlée (AOC), les productions de noix doivent remplir deux conditions :

- provenir de vergers situés dans les communes référencées par la réglementation. Ces communes sont toutes situées le long de la vallée de l'Isère,
- correspondre aux variétés de noix Franquette, Mayette ou Parisienne.

La production

Chaque année la coopérative réceptionne les récoltes de ses producteurs. Cette activité de l'entreprise, comme toutes les autres, doit être inscrite dans une démarche qualité assurant la traçabilité des produits finis.

Afin d'assurer cette traçabilité, la coopérative identifie chaque livraison de noix réalisée par un producteur. Une livraison est caractérisée par le verger dont proviennent les noix, sa date, le type de produit (entière fraîche ou entière sèche) et la quantité livrée.

Après réception, les noix sont triées en fonction de leur taille (voir *annexe 1A*). La coopérative utilise pour cela une calibreuse destinée à répartir la livraison en différents lots de production. Un lot de production est donc constitué de l'ensemble des noix d'une livraison conformes à un calibre particulier. L'*annexe 1B* présente un exemple de livraison répartie en différents lots de production.

Les campagnes de ventes

Les noix sont vendues à des clients qui sont pour l'essentiel des grossistes, des entreprises de la grande distribution ou des professionnels des métiers de la bouche (pâtisseries, restaurateurs).

Une commande est caractérisée par un numéro de commande, une date de commande et un client. Chaque commande porte sur un seul lot de production déterminé en fonction de la demande du client (variété de noix, type de produit et calibre), ce qui permet d'assurer la traçabilité des produits vendus.

Les informations concernant le client (nom, adresse, nom du responsable des achats) doivent être enregistrées.

Pour chaque commande, il faut également mémoriser les types de conditionnement et les quantités commandés. Les différents types de conditionnement proposés par NOIXCOOP sont : le sachet (de 250 g, 500 g et 1 kg), le filet (de 1 kg, 5 kg, 10 kg et 25 kg) et le carton de 10 kg.

Une commande peut par exemple comporter 800 sachets de 1 kg et 300 filets de 10 kg pour un poids total de 3 800 kg.

Travail à faire	
1.1	À partir des informations fournies dans ce dossier et des <i>annexes 1A et 1B</i> , établir un schéma entité association des données concernant les approvisionnements et les ventes de noix.

À la demande de ses adhérents NOIXCOOP développe une nouvelle activité de culture de plants de noyers compatibles avec la charte qualité associée au label AOC. Elle cultive et commercialise des plants de noyers de différentes variétés. Pour cela elle utilise une base de données, dont voici un extrait :

TypeUsageArbre(id, libellé)

id : clé primaire

Limité à : (1, 'Fruitier'), (2, 'Huile'), (3, 'Greffon')

Il s'agit du type d'usage principal d'une variété d'arbres

EspèceArbre(id, libellé)

id : clé primaire

Par exemple : (1, 'noyer commun'), (2, 'noyer cendré'), (3, 'noyer noir')

VariétéArbre(id, idEspèce, idTypeUsage, libellé)

id : clé primaire

idEspèce : clé étrangère en référence à id de EspèceArbre

idTypeUsage : clé étrangère en référence à id de TypeUsageArbre

*Par exemple : (1, 1, 1, 'franquette'), (2, 1, 1, 'parisienne'), (3, 1, 1, 'mayette'),
(4, 3, 3, 'soligo')*

Dimension(id, min, max)

id : clé primaire

min, max : plage de hauteur approximative de l'arbre, en mètre : [min .. max]

Par exemple : (1, 0.15, 0.30), (2, 0.30, 0.50), (3, 0.4, 0.6), (4, 2.5, 3.0)

PrixVenteArbre(idVariété, âge, idDimension, prix)

idVariété, âge, idDimension : clé primaire

idVariété : clé étrangère en référence à id de VariétéArbre

idDimension : clé étrangère en référence à id de Dimension

âge : valeur entière en année

prix : prix de vente hors taxes à l'unité, en euros

Par exemple : (1, 2, 2, 9), (1, 2, 5, 12.5)

Travail à faire	
2.1	En prenant en compte les exemples de données ci-dessus, expliquer l'intention de cette requête : <pre>INSERT INTO PrixVenteArbre VALUES (1, 1, 1, 8)</pre>
2.2	Expliquer le résultat que permet d'obtenir la requête ci-dessous puis en proposer une version simplifiée. <pre>SELECT EspèceArbre.libellé AS espèce, VariétéArbre.libellé AS variété FROM VariétéArbre, TypeUsageArbre, EspèceArbre WHERE VariétéArbre.idTypeUsage = TypeUsageArbre.id AND VariétéArbre.idEspèce = EspèceArbre.id ORDER BY EspèceArbre.libellé</pre>
2.3	Écrire la requête SQL permettant à un utilisateur de connaître le nombre de variétés d'arbres proposées par type d'usage (id, libellé).
2.4	Écrire la requête SQL permettant de connaître les variétés d'arbres (libellé) ne faisant l'objet d'aucun prix de vente.

Au début de cette année, la coopérative NOIXCOOP désire augmenter de 3% le prix de vente de tous ses noyers âgés d'au moins 3 ans, toute variété confondue.

Travail à faire	
2.5	Écrire la requête SQL effectuant cette augmentation.

Dossier 3	Suivi des commandes
------------------	----------------------------

Documents à utiliser : annexes 2A, 2B, 2C, 2D et 2E.

Les distributeurs de noix sont des partenaires privilégiés de NOIXCOOP. Ils représentent en effet 80% des débouchés de la production de NOIXCOOP.

NOIXCOOP réalise 10% de son chiffre d'affaires avec l'un de ces distributeurs (SUPERMARKET). Ce distributeur vient d'actualiser son système d'information et impose à ses fournisseurs une interconnexion des systèmes d'information au moyen d'une architecture orientée "Web Service" (architecture WS).

Chez NOIXCOOP les commandes sont gérées par téléphone tant pour la prise de commande que pour le suivi. Afin d'améliorer le suivi des commandes et de répondre à la demande de SUPERMARKET, NOIXCOOP décide donc de mettre à la disposition de ses clients un service *web* qui leur permettra d'obtenir un document XML présentant la liste de leurs commandes en cours.

Votre rôle consiste à étudier la rentabilité de ce service *web*, puis à le développer.

Les éléments de coûts liés à la mise en place de cette architecture sont présentés en *annexe 2A*.

Travail à faire	
3.1	En utilisant les données fournies en <i>annexe 2A</i> , évaluer le coût annuel de l'activité commerciale après la mise en place de l'architecture WS.

Le coût annuel supplémentaire lié à l'exploitation de l'architecture WS proprement dite est finalement estimé à 3 000 €.

Travail à faire	
3.2	Calculer le taux de rentabilité annuel de l'activité de NOIXCOOP avec le distributeur SUPERMARKET dans le contexte de l'architecture WS.
3.3	Indiquer les avantages apportés aux distributeurs par le futur service par rapport au système actuel.
3.4	Donner trois arguments montrant l'intérêt de la mise en place de cette solution pour NOIXCOOP.

Un exemple d'utilisation du futur service *web* est présenté en *annexe 2B*.

Principe de fonctionnement

- Le distributeur devra avoir obtenu auprès de NOIXCOOP un identifiant de connexion au service (exemple carr15432 pour le distributeur *carreclerc*).
- À l'aide de cet identifiant de connexion, une application cliente pourra interroger le service *web* à l'aide d'une requête HTTP GET :

<http://ws.noixcoop.fr/services/etatCommandes.php?distributeur=carr15432>

- Il obtiendra en retour la liste de ses commandes en souffrance (non expédiées) au format XML (*annexe 2B*).

Un ensemble de classes présentées en *annexes 2C et 2D* est en cours de développement.

- La classe *PersistenceSQL* permet d'accéder à la base de données. Elle permet de charger et/ou d'enregistrer les données dans la base.
- La classe *GestionCommandes* permet d'orchestrer les traitements liés au service Web.
- La dépendance *Utilise* représente simplement le fait que la classe *GestionCommandes* doit avoir connaissance de la classe *Distributeur* puisqu'elle possède une méthode retournant un objet de la classe *Distributeur*.

Vous disposez également de la classe *Collection* présentée en *annexe 2E*.

Chargement des données depuis la base

Le fonctionnement de la méthode *GetDistributeur()* de la classe *GestionCommandes* est le suivant :

- Elle utilise la classe *PersistenceSQL* pour charger les données concernant un distributeur depuis la base (l'identifiant de ce distributeur étant passé en paramètre).
- Ce chargement provoque de manière automatique le chargement des données concernant l'ensemble des commandes de ce distributeur, ainsi que les produits associés.
- Elle retourne l'objet *Distributeur* ainsi créé, cet objet possédant la collection des commandes du distributeur.

Travail à faire	
3.5	Écrire la méthode <i>GetDistributeur()</i> de la classe <i>GestionCommandes</i> .
3.6	Écrire la méthode <i>EnCours()</i> de la classe <i>Commande</i> .
3.7	Écrire la méthode <i>GetCommandesEnCours()</i> de la classe <i>Distributeur</i> .
3.8	Écrire la méthode <i>XmlCommande()</i> de la classe <i>Commande</i> .

Une classe *Test* a été créée dans le but de tester votre travail. Sa méthode *AfficheCommandesCarreClerc()* ébauchée ci-dessous doit afficher les commandes en souffrance du distributeur *Carreclerc* d'identifiant "carr15432", c'est-à-dire une chaîne de caractères conforme à l'annexe 2B. Les données se trouvent dans la base de données de nom "baseNoixCoop" sur le serveur d'adresse IP 192.168.10.240 écoutant le port 2207.

Classe Test

```
Public
    Procédure AfficheCommandesCarreClerc ( )
        persist : PersistenceSQL
        // à compléter...
    fin procédure
```

Fin classe

Travail à faire	
3.9	Écrire la méthode <i>AfficheCommandesCarreClerc()</i> de la classe <i>Test</i> .

L'application de gestion du service *web* de suivi des commandes est terminée ; elle est actuellement installée sur votre poste en local (poste dédié au développement).

Elle paraît dans cette configuration conforme aux exigences définies au départ par NOIXCOOP. Vous devez maintenant la déployer sur le serveur *web* (serveur HTTP) de la coopérative pour tester la mise en production (voir *annexe 3* : schéma du réseau).

Vous devez transférer le service *web* sur le serveur HTTP à partir de votre poste. Or le poste de développement ne dispose actuellement que d'un éditeur de texte avec reconnaissance syntaxique et d'un serveur *web* de test accessible seulement en local (*localhost*).

Travail à faire	
4.1	Proposer une solution pour transférer facilement vos scripts sur le serveur.
4.2	Écrire la commande système permettant de vérifier la capacité du poste de développement à communiquer (par datagrammes IP) avec le serveur <i>web</i> .
4.3	Donner la passerelle par défaut du poste de développement et indiquer de quelle autre adresse IP celui-ci aura besoin pour pouvoir naviguer sur Internet.

Vous configurez un nouveau poste à l'exploitation. L'administrateur du réseau vous a fourni l'adresse IP à utiliser : 192.168.1.14.

Travail à faire	
4.4	Préciser le masque de sous-réseau.

Vous assistez l'administrateur pour changer le routeur R2. Il n'a inscrit pour l'instant qu'une ligne (la route par défaut) dans la table de routage :

Destination réseau	Masque réseau	Adresse passerelle	Adresse IP de l'interface du routeur
0.0.0.0	0.0.0.0	81.58.110.10	81.58.110.13

Travail à faire	
4.5	Présenter la ligne ouvrant l'accès au serveur HTTP depuis Internet.
4.6	Donner la ligne de la route par défaut du routeur R1.

Les routes sont maintenant toutes correctement configurées, mais un client depuis internet qui envoie un paquet vers le service d'adresse 81.58.110.13:80 n'obtient pas de réponse.

Travail à faire	
4.7	Préciser l'ordinateur qui devrait recevoir ce paquet et indiquer ce qu'il faut vérifier sur la configuration du routeur R2.

Annexe 1A - Les calibres de noix

Identifiant	libellé
1	inférieur à 24 mm
2	24 à 28 mm
3	28 à 30 mm
4	30 à 32 mm
5	32 à 34 mm
6	supérieur à 34 mm

Annexe 1B - Exemple de livraison

Code Livraison : L0512

Date de livraison : 18/04/2010

Producteur :

Verger :

Les noix de Polienas
52, route de Grenoble
38 210 POLIENAS

V058 - Verger du moulin

Variété : Franquette

Quantité : 12 520 Kg

Type de produit : Entière fraîche

Lots de production après calibrage :

Numéro de lot	Calibre	Quantité
1	24 à 28 mm	4 200 Kg
2	28 à 30 mm	3 820 Kg
3	30 à 32 mm	1 580 Kg
4	32 à 34 mm	2 920 Kg

Annexe 2A - Éléments de coûts

Données comptables actuelles : extrait de la balance annuelle au 31/12

Numéro de compte	Libellé du compte	Solde débiteur	Solde créditeur
626100	Téléphonie	12 100,00 €	
626200	Liaison Internet	4 200,00 €	
707000	Vente		10 000 000,00 €

Analyse des coûts actuels

Coûts	Total	Répartition par centre d'activité		
		Commercial	Administratif	Production
Téléphonie	12 100,00 €	10 000,00 €	2 000,00 €	100,00 €
Abonnement Internet	4 200,00 €	1 400,00 €	1 400,00 €	1 400,00 €
<i>Total</i>	<i>16 300,00 €</i>	<i>11 400,00 €</i>	<i>3 400,00 €</i>	<i>1 500,00 €</i>

Éléments de devis en vue de la mise en place de l'architecture WS

- Hébergements externes dédiés à l'architecture WS 5 300 € par an
- Liaison internet haut débit 525 € par mois

Extraits du procès verbal de la réunion de travail sur le projet de mise en place d'une solution **Web Service** pour tenir compte du cahier des charges du client **SUPERMARKET**

"On estime à 30%, le gain sur les coûts de téléphonie dédiés à l'activité commerciale. Il ne devrait pas y avoir d'incidence significative sur les centres d'activité Administratif et Production.

La liaison Internet haut débit remplacera l'abonnement Internet existant. La répartition des coûts de l'abonnement Internet sera la même qu'actuellement car le haut débit bénéficiera à l'ensemble des centres.

L'hébergement ne concerne que l'activité commerciale, donc son coût sera entièrement imputé sur ce centre d'activité".

"La part de notre chiffre d'affaires avec SUPERMARKET est de 10% et notre taux de rentabilité est actuellement, sans l'architecture WS, de l'ordre de 4 %. C'est plutôt mieux qu'avec les autres clients pour lesquels ce taux se situe entre 3,5 et 3,8%".

"Ce service pourra bénéficier à d'autres clients, voire être étendu à d'autres processus métiers comme la prise de commande".

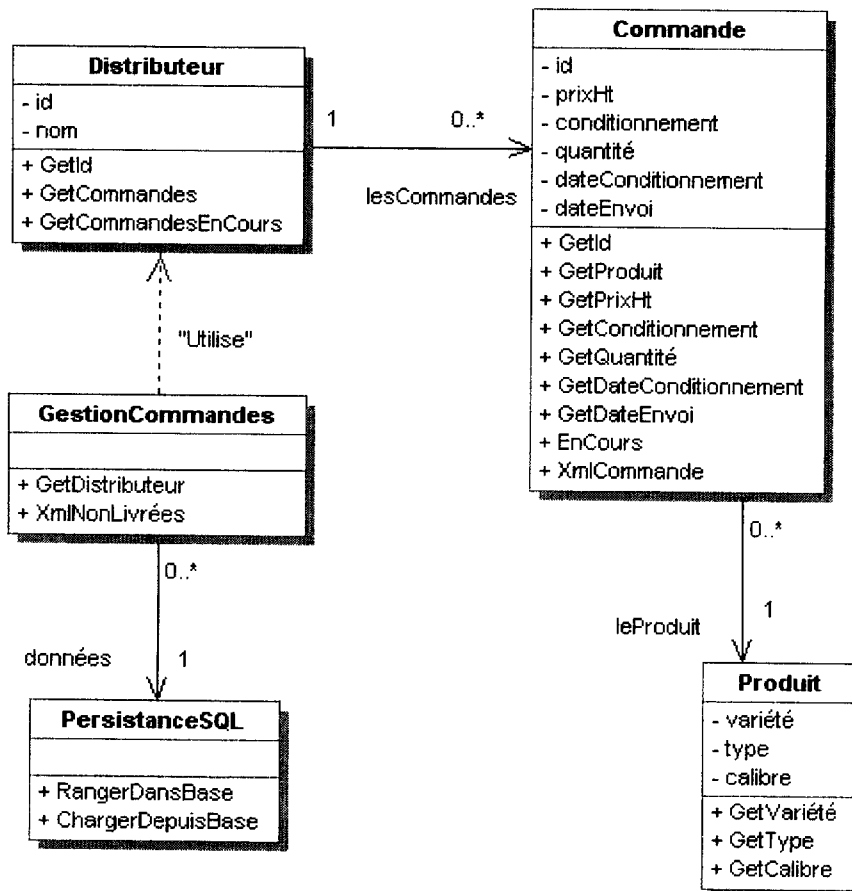
Remarque : taux de rentabilité = résultat / chiffre d'affaires.

Annexe 2B - Exemple d'utilisation du service web

Exemple : le distributeur *carreclerc* désire obtenir l'état de ses commandes restant à livrer. Une commande qui reste à livrer est une commande dont la date d'envoi n'est pas renseignée (sa valeur est NULL). Il sollicite le service web "etatCommandes" et obtient le fichier XML ci-dessous.

```
<?xml version="1.0" encoding="UTF-8"?>
<commandes idDistributeur="carr15432" xmlns:xlink="http://www.w3.org/1999/xlink">
  <commande id="00213">
    <produit variete="Mayette" type="Fraiche Entière" calibre="2" />
    <conditionnement type="filet 1kg" />
    <quantite>50</quantite>
    <date_conditionnement>12-05-09</date_conditionnement>
    <date_envoi>null</date_envoi>
  </commande>
  <commande id="00215">
    <produit variete="Parisienne" type="Cerneaux" calibre="1" />
    <conditionnement type="filet 5kg" />
    <quantite>100</quantite>
    <date_conditionnement>08-05-09</date_conditionnement>
    <date_envoi>null</date_envoi>
  </commande>
</commandes>
```

Annexe 2C - Diagramme de classes partiel



Par souci de simplification, on considère ici qu'il n'y a qu'une seul conditionnement possible par commande.

Annexe 2D - Classes métier

Classe PersistenceSQL

Public

// Constructeur

PersistenceSQL (ipBase : chaîne, port : entier, nomBaseDonnee : chaîne)

// Construit un objet *PersistenceSql*. Cet objet permettra de charger les données depuis une base
// de données ou de les sauvegarder dans la base.

procédure RangerDansBase (unObjet : Objet)

// Stocke les données de l'objet dans la base de données.

fonction ChargerDepuisBase (id : chaîne, nomClasse : chaîne) : Objet de la classe *NomClasse*.

// Retourne l'objet de la classe *NomClasse* dont l'identifiant est "id". Cet objet est chargé

// depuis la base de données, ainsi que l'ensemble de ses objets liés (voir l'exemple d'utilisation

// ci-dessous). Retourne NULL si aucun objet de cette classe ne possède cet identifiant.

Fin classe

Exemple d'utilisation :

// *persist* est une instance de PersistenceSQL

Distributeur leDistributeur ← *persist.chargerDepuisBase* ("2", Distributeur)

// *leDistributeur* est l'instance de la classe Distributeur dont l'identifiant est 2.

Toutes les commandes du distributeur sont automatiquement chargées dans la collection *leDistributeur.lesCommandes*. Chaque produit commandé est également chargé, et se trouve donc référencé par le champ *leProduit* de l'objet Commande correspondante.

Classe Distributeur

Privé

id : chaîne

nom : chaîne

lesCommandes : Collection de <Commande> // Toutes les commandes du distributeur.

Public

// Constructeur

Distributeur (unId : chaîne , unNom : chaîne)

// Construit un objet Distributeur. À ce stade, il n'est pas stocké dans la base de données.

fonction GetId () : chaîne

// Retourne l'identifiant du distributeur.

fonction GetCommandes () : Collection de <Commande>

// Retourne l'ensemble des commandes passées par ce distributeur.

fonction GetCommandesEnCours () : Collection de <Commande>

// Retourne une collection constituée des commandes en cours (non expédiées) du distributeur.

Fin classe

Classe Commande

Privé

id : entier // Identifiant de la commande.
leProduit : Produit // Le produit commandé (catégorie de noix).
prixHt : réel // Prix unitaire du produit négocié avec le client.
conditionnement : chaîne // Type de conditionnement.
quantité : entier // Quantité de produits conditionnés commandée.
dateConditionnement : Date // Date de conditionnement de la commande.
dateEnvoi : Date // Date d'envoi de la commande.

Public

// Constructeur
Commande (...)
// Construit un objet Commande, la liste des paramètres est sans importance.
// Le champ dateEnvoi est initialisé à NULL.

fonction GetId () : entier
// Retourne l'identifiant de la commande.

fonction GetProduit () : Produit
// Retourne le produit commandé.

fonction GetPrixHt () : réel
// Retourne le prix unitaire négocié avec le client.

fonction GetConditionnement () : chaîne
// Retourne le type de conditionnement des produits de cette commande.

fonction GetQuantité () : entier
// Retourne la quantité commandée.

fonction GetDateConditionnement () : Date
// Retourne la date de conditionnement de la commande.

fonction GetDateEnvoi () : Date
// Retourne la date d'envoi de la commande.

fonction EnCours () : booléen
// Renvoie vrai si la commande n'est pas encore expédiée, faux sinon.
// Une commande n'est pas expédiée si sa date d'envoi contient NULL.

fonction XmlCommande () : chaîne
// Retourne la chaîne correspondant au code XML représentant la commande (voir annexe 3B).
// Cette fonction est appelée par la méthode *XmlNonLivrées()* de la classe
// *GestionCommandes* décrite ci-après.

Fin classe

Classe Produit

Privé

```
variété : chaîne // Variété de noix, exemple : "Mayette".
type : chaîne // Type de noix, exemple : "fraîche entière".
calibre : entier // Calibre des noix, exemple : 2.
```

Public

```
// Constructeur
Produit ( ... )
// Construit un objet Produit, la liste des paramètres est sans importance.
```

```
fonction GetVariété () : chaîne
// Renvoie la variété du produit.
```

```
fonction GetType () : chaîne
// Renvoie le type du produit.
```

```
fonction GetCalibre () : entier
// Renvoie le calibre du produit.
```

Fin classe

Classe GestionCommandes

Privé

```
données : PersistenceSQL
// Attribut qui permet de rendre les objets métiers accessibles.
```

Public

```
//Constructeur
GestionCommandes (lesDonnees : PersistenceSQL)
// Construit un objet GestionCommandes avec un modèle de persistance associé.
```

fonction GetDistributeur (idDistributeur : chaîne) : Distributeur

```
// Retourne l'objet Distributeur qui possède l'identifiant idDistributeur passé en paramètre,
// retourne null si aucun Distributeur ne possède cet identifiant.
```

```
fonction XmlNonLivrées (unDistributeur : Distributeur) : chaîne
// Retourne une chaîne de caractères qui représente le document XML de la liste des commandes
// non livrées du distributeur passé en paramètre comme le montre l'exemple de l'annexe 3B.
```

```
{
  xml : chaîne
  xml ← ' <?xml version="1.0" encoding="UTF-8"?> '
  xml ← xml + ' <commandes idDistributeur=' + unDistributeur.GetId() + ' '
  xml ← xml + ' xmlns:xlink="http://www.w3.org/1999/xlink"> '
  enCours : Collection de <Commande>
  enCours ← unDistributeur.GetCommandesEnCours()
  laCommande : Commande
  Pour chaque laCommande dans enCours faire
    xml ← xml + laCommande.XmlCommande()
  FinPour
  xml ← xml + "</commandes>"
  retourner xml
}
```

Fin classe

Annexe 2E - Classe Collection

Classe Collection de <TypeElément>

// TypeElément peut être un type simple ou une classe.

Public

//Constructeur

Collection de <TypeElément> () // Construit une collection d'objets "TypeElement".

...

fonction NbEléments () : entier // Retourne le nombre d'éléments de la collection.

fonction GetElément (unIndex : entier) : TypeElément // Retourne l'objet d'index *unIndex*.

// Le premier élément est à l'index 1.

procédure Ajouter (unObjet : TypeElément)

// Ajoute l'objet *unObjet* à la collection.

Fin Classe Collection

Exemple d'utilisation de la collection :

lesClients : Collection de <Client>

unClient : Client

lesClients ← new Collection de <Client> ()

...

lesClients.ajouter (unClient)

...

// Parcours de la collection :

leClient : Client

Pour chaque leClient dans lesClients faire

 Si leClient.GetVille () = "Paris" alors

 Afficher ("Ce client habite Paris")

 FinSi

FinPour

Annexe 3 - Schéma du réseau de la coopérative

