CAS SENOIX

SENOIX 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).

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

SENOIX 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 SENOIX 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, SENOIX 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 développer ce service *web*.

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

Principe de fonctionnement

- Le distributeur devra avoir obtenu auprès de SENOIX 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.senoix.fr/services/etatCommandes.php?distributeur=carr15432

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

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

- La classe *PersistanceSQL* 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 D*.

Chargement des données depuis la base

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

- Elle utilise la classe *PersistanceSQL* 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	
1	Écrire la méthode *GetDistributeur()* de la classe *GestionCommandes*.
2	Écrire la méthode *EnCours()* de la classe *Commande*.
3	Écrire la méthode *GetCommandesEnCours()* de la classe *Distributeur*.
4	Écrire la méthode *XmlCommande()* de la classe *Commande*.

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 A. Les données se trouvent dans la base de données de nom "baseSenoix" sur le serveur d'adresse IP 192.168.10.240 écoutant le port 2207.

Classe Test

```
Public
        Procédure AfficheCommandesCarreClerc ( )
                persist : PersistanceSQL
                // à compléter…
        Fin procédure
```

Fin classe

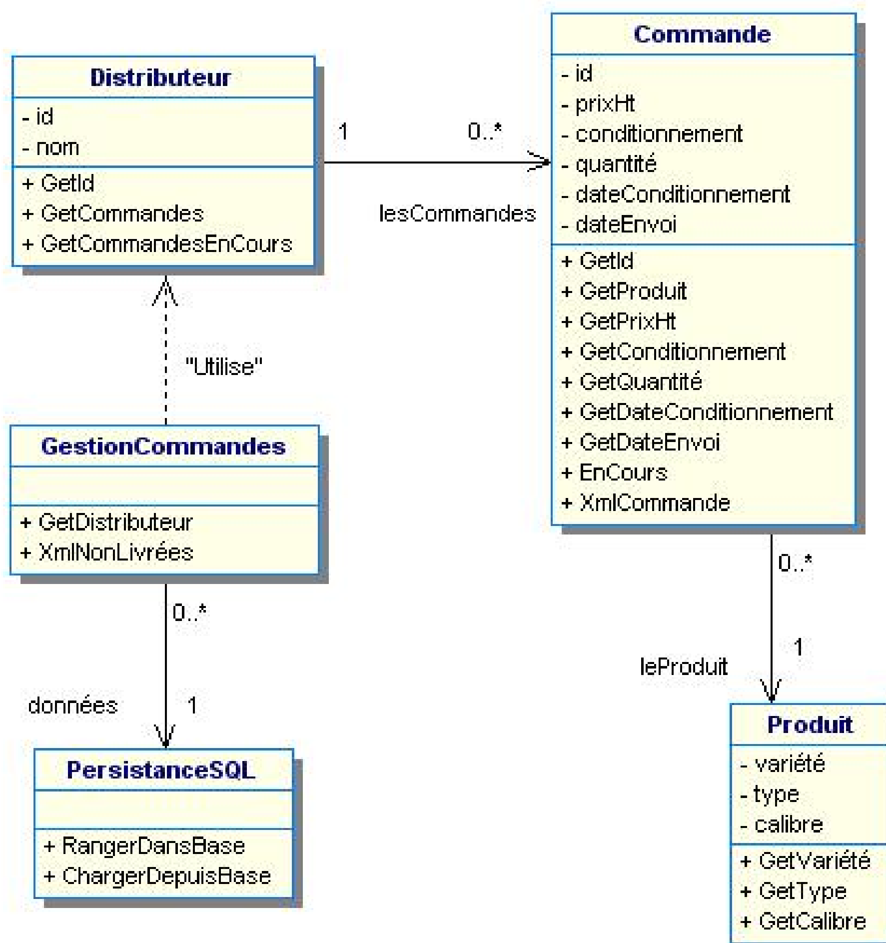
Travail à faire	
5	Écrire la méthode *AfficheCommandesCarreClerc()* de la classe *Test*.

Annexe A - 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 B - Diagramme de classes partiel



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

Annexe C - Classes métier

Classe PersistanceSQL

Public
    // Constructeur
    PersistanceSQL (ipBase : chaîne, port : entier, nomBaseDonnee : chaîne)
    // Construit un objet *PersistanceSql.* 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 PersistanceSQL
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. Á 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 **A***).
// 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 : PersistanceSQL
// Attribut qui permet de rendre les objets métiers accessibles.
```

Public

```
//Constructeur
GestionCommandes (lesDonnees : PersistanceSQL)
// 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 A.
{
    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 D - 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