BTS SERVICES INFORMATIQUES AUX ORGANISATIONS

E5 : Production et fourniture de services informatiques

SESSION 2017

Durée : 4h00 Coefficient : 5

Cas AHM-23

Ce sujet comporte 18 pages dont un dossier documentaire de 10 pages.

La candidate ou le candidat doit vérifier que le sujet qui lui est remis est complet.

Aucun matériel ni document autorisé.

Dossier documentaire :

Barème :

Mission 1	Évolution de l'architecture logicielle	20 points
Mission 2	Mise en place d'une gestion de stock centralisée	30 points
Mission 3	Finalisation de l'application de saisie des livraisons de bois	20 points
Mission 4	Valorisation des déchets	30 points
	Total	100 points

Conformément aux recommandations du Haut Conseil à l'Égalité entre les femmes et les hommes dans son guide publié en novembre 2015, l'expression du féminin et du masculin s'effectue en utilisant le point, par exemple, employé.e.

Présentation du contexte

L'organisation cliente : l'association AHM-23

Créée en 1960 et reconnue d'utilité publique, l'association d'aide au handicap mental AHM-23 est une association départementale à but non lucratif (association loi 1901) affiliée à l'Unapei (Union nationale des associations de parents, de personnes handicapées mentales et de leurs amis). Elle œuvre en faveur des droits des personnes souffrant de handicap mental et de leurs familles, en favorisant leur insertion sociale et professionnelle.

En effet, les obligations de la société envers la personne handicapée mentale sont :

- de lui donner les moyens adaptés à la nature et au degré de sa déficience qui lui permettent d'exercer ses droits et d'accomplir ses devoirs ;
- de veiller à ce qu'elle soit reconnue et respectée et de lui apporter la protection qui la mette à l'abri de toute exploitation.

L'AHM-23 gère plusieurs foyers d'hébergement et trois établissements et service d'aide par le travail (Esat) répartis sur le département.

Les établissements Esat permettent aux personnes handicapées qui n'ont pas acquis suffisamment d'autonomie pour travailler en milieu ordinaire, d'exercer une activité professionnelle dans un milieu protégé.

L'AHM-23 emploie au total 188 ouvriers d'Esat, encadrés par 25 travailleurs sociaux, répartis sur trois sites différents : l'Esat La Source, l'Esat Les Mésanges et l'Esat La Forêt.

Chaque Esat dispose d'un ou plusieurs ateliers, par exemple :

- un atelier de fabrication de palettes ;
- un service restauration-traiteur ;
- un service espaces verts ;
- etc.

Le contexte

La loi du 2 janvier 2002 a imposé aux Esat (tout comme aux autres établissements sociaux et médico-sociaux) le renforcement de la qualité par la mise en place d'outils et de démarches qui conduisent à davantage de formalisation et de rationalisation.

Dans ce cadre, en 2012, un constat d'audit interne a préconisé la centralisation de la gestion des stocks et la rationalisation des achats par le comptable de l'AHM-23.

Le projet

Les Esat La Forêt et Les Mésanges ont chacun un atelier dont l'activité consiste à fabriquer des palettes sur commande pour des entreprises locales.

Actuellement, le chef d'atelier de chaque site utilise une application tableur permettant la gestion des commandes de palettes pour ses clients ainsi que la gestion du stock des matières premières entrant dans la fabrication des palettes.

Depuis quelques années, la demande de palettes ne cesse de croitre. Le volume mensuel de palettes fabriquées atteint actuellement 9000 exemplaires au total. De ce fait, la gestion des stocks à l'aide du tableur n'est plus concevable.

Afin d'avoir une vision globale des stocks, l'AHM-23 souhaiterait disposer d'une solution *web* accessible par les deux sites permettant de connaître en temps réel l'état des stocks de matières premières, de palettes fabriquées et d'envisager une valorisation des déchets.

Chaque site a ses propres clients. L'AHM-23 dispose d'un service informatique qui ne fait pas de développement d'applications. Aussi l'association décide-t-elle d'avoir recours au prestataire ALM-TIC pour répondre à ces nouvelles exigences. Le comptable sera l'interlocuteur de l'organisation prestataire.

L'organisation prestataire

Spécialisée dans l'édition de logiciels applicatifs spécifiques, la société ALM-TIC, est une entreprise de services du numérique (ESN) qui emploie une vingtaine de salariés.

Son activité principale est le développement de progiciels dans le domaine social, essentiellement pour des PME ou des associations de la région. Elle propose également des prestations de formation ainsi que de maintenance préventive et corrective. Trois salariés, dont deux développeurs et un formateur itinérant, seront amenés à travailler sur le projet de l'AHM-23.

Le cahier des charges prévoit la migration des données du tableur vers une base de données relationnelle et le développement d'une application *web* intégrant les fonctionnalités de l'application tableur et les nouveaux besoins. Cette application exploitera la base de données et sera accessible par les deux Esat La Forêt et Les Mésanges.

Vous assistez M. Laplace développeur de la société ALM-TIC affecté sur le projet.

Mission 1 : Évolution de l'architecture logicielle

Document à utiliser : 1

Le projet d'évolution de l'application tableur vers l'application *web* va nécessiter la mise en place d'une nouvelle architecture matérielle et applicative ainsi que le développement du site *web*.

Le site *web* et la base de données seront hébergés au siège d'AHM-23, en charge des fonctions courantes d'exploitation du système informatique de l'association.

Dans le cadre de la refonte de l'application de gestion des stocks, l'analyse de l'application tableur existante a permis de modéliser la structure des données actuelles. La nouvelle base de données contiendra des informations sur les stocks des deux Esat et devra faire l'objet d'une sécurisation :

- physique par sauvegardes incrémentielles et,
- logique par contrôles d'accès.

Question 1.1

Énumérer les composants logiciels type devant être installés pour la mise en production de l'application *web*.

Question 1.2

Présenter les étapes nécessaires à la récupération des données depuis l'application tableur pour alimenter et valider le contenu de la nouvelle base de données.

Question 1.3

Proposer une solution à mettre en œuvre pour s'assurer qu'un chef d'atelier puisse modifier les informations de son site et que le comptable puisse consulter l'ensemble des informations des deux Esat.

Le comptable a entendu parler des sauvegardes différentielles et vous interroge à ce propos. Vous lui conseillez de rester sur une sauvegarde incrémentielle.

Question 1.4

Présenter au comptable les arguments en faveur du maintien de cette solution de sauvegarde.

Question 1.5

Indiquer comment utiliser les sauvegardes incrémentielles pour restaurer l'ensemble des données, en cas de perte des données contenues dans la base de données.

Mission 2 : Mise en place d'une gestion de stock centralisée

Documents à utiliser : 1, 2

IMPORTANT : la candidate ou le candidat présentera les évolutions de la structure de la base de données en adoptant le formalisme de son choix (schéma entité-association, diagramme de classes ou encore schéma relationnel).

Les stocks de produits finis représentent une perte de place et un coût de stockage important. Le comptable de AHM-23 souhaite avoir une vision globale du nombre total de palettes fabriquées mais non encore livrées (pas de date de livraison enregistrée) ainsi que le coût de revient total de ce stock basé sur le coût de revient du modèle de palette.

M. Laplace a créé une base de données relationnelle sous *MySQL* dans laquelle il a migré les données de l'application tableur. Cette base de données contient les informations nécessaires pour répondre au besoin urgent du comptable.

Il vous demande de répondre à cette demande par l'intermédiaire d'une vue nommée ***stockPalettes*** contenant les colonnes *nbPalettes* et *coutRevientTotal* et de proposer comment restreindre son utilisation au seul comptable.

Question 2.1

Écrire la vue demandée par le chef de projet permettant de répondre à ce besoin.

Question 2.2

Expliquer en quoi la création d'une vue est une solution pour répondre à ce besoin.

Par ailleurs, les entretiens menés avec le comptable et les chefs d'atelier concernés ont permis d'identifier de nouveaux besoins. Vous travaillez avec M. Laplace pour faire évoluer la structure de la base de données initiale :

- Deux types de modèle de palette sont désormais proposés :
  - des modèles de palette standard, identifiés par une référence catalogue avec des dimensions normalisées ;
  - des modèles de palette personnalisés aux dimensions choisies par l'entreprise cliente, dont on doit connaître la raison sociale et le contact, personne qui a validé le modèle.

  Chaque type de modèle de palette a des dimensions (une longueur et une largeur) exprimées en millimètres.
- Une commande client peut concerner un ou plusieurs modèles de palette, qu'ils soient personnalisés ou standards. Chaque modèle de palette est illustré par un plan (stocké au format PDF) faisant apparaître les composants qui entrent dans la fabrication du modèle de palette, avec pour chacun la quantité à utiliser.

  Par exemple, le modèle de palette personnalisé 1200mmx1200mm de l'entreprise cliente ISOCOUSTIC est composé de 7 planches 1200 mmx75 mm d'épaisseur 15 mm, de 3 traverses 1200 mmx80 mm d'épaisseur 15 mm, de 3 skis 1200 mmx75 mm d'épaisseur 15 mm et de 9 plots 75 mmx80 mm d'épaisseur 100 mm.

- Fabriquer une palette consiste à découper ses composants dans les matières premières, qui sont les éléments de bois achetés et stockés, puis à les assembler. Les composants d'une même palette sont tous issus d'une même essence de bois.

  Exemple de matière première : lamelle de bois de hêtre de longueur 800, largeur 70 et épaisseur 15.

- Un même composant de palette (par exemple un ski de longueur 1200, largeur 75, épaisseur 15) peut être découpé dans différentes matières premières. La quantité découpée sera à mémoriser pour gérer au mieux les stocks de bois. Pour chaque couple composant/matière première, on conservera également les longueurs de perte et de chute.

  Longueur de perte : les pertes sont causées par l'épaisseur des traits de scie.

  Longueur de chute : la longueur du composant à découper peut être inférieure à celle de la matière première utilisée. Il reste donc après la découpe un morceau appelé chute.
  Pour information, les chutes sont valorisées en allume-feux mais cette fabrication annexe ne rentre pas dans le cadre de cette mission.

- Chaque site dispose de sa propre zone de stockage gérée par le chef d'atelier. Pour accéder à la future application, chaque employé.e du site sera authentifié.e par un identifiant et un mot de passe.

- Pour gérer correctement les niveaux de stock des matières premières, il est important de pouvoir mémoriser, pour chaque site et pour chaque matière première, la quantité en stock exprimée en m^3 mais également le niveau de son stock d'alerte qui permettra de déclencher un réapprovisionnement. Les sorties de stock de matières premières seront prises en compte lorsque les palettes d'une ligne de commande seront déclarées fabriquées.

- En raison de l'éloignement des deux sites, chaque entreprise cliente est rattachée à un seul site.

Question 2.3

Proposer une modélisation de l'évolution de la structure de la base de données en intégrant la structure existante et les nouveaux besoins en informations. Seuls les éléments du schéma existant qui sont concernés par l'évolution seront repris dans le schéma proposé.

Mission 3 : Finalisation de l'application de saisie des livraisons de bois

Documents à utiliser : 3, 4, 5 et 6

Les matières premières utilisées pour la fabrication des composants de palettes sont achetées auprès de scieries de la région.

Les commandes effectuées auprès de ces fournisseurs sont livrées par ces derniers. Lorsqu'un camion arrive, il est déchargé sur une zone de stockage de bois frais. Toute commande livrée est mesurée en volume par le chef d'atelier, car les quantités livrées sont souvent légèrement supérieures aux quantités commandées.

La gestion des commandes de matières premières auprès des fournisseurs sera intégrée à l'application *web*.

Afin de faciliter le travail des chefs d'atelier, il a été décidé de les doter d'un terminal mobile de type tablette afin que chacun d'eux puisse saisir les quantités mesurées sur la zone de stockage de bois frais de l'atelier dont il s'occupe.

Ces deux zones de stockage du bois livré, assez vastes, ne sont pas couvertes par les réseaux *Wi-Fi* des établissements. Aussi, une application native sous *Android* de saisie des quantités livrées sera développée et installée sur les tablettes, ainsi qu'une base de données locale, réplique de la base de données centralisée. Avant de partir sur la zone de stockage pour effectuer les mesures de commandes livrées, le chef d'atelier devra charger la base de données sur la tablette.

Vous participez, dans cette mission, au développement de cette application codée en partie par M. Laplace.

Question 3.1

Justifier le choix de la société ALM-TIC pour l'application native, par rapport à d'autres types d'applications.

Question 3.2

En vous inspirant du code implémentant le premier écran de l'application (affichage des commandes livrées), compléter à partir de la ligne 5 le code permettant d'alimenter le deuxième écran (affichage des lignes de la commande sélectionnée sur l'écran précédent).

Question 3.3

Compléter, à partir de la ligne 4, le code exécuté au clic du bouton "+" du troisième écran, en suivant les consignes données par M. Laplace.

Mission 4 : Valorisation des déchets

Documents à utiliser : 7 et 8

IMPORTANT : la candidate ou le candidat peut choisir de présenter les éléments de code à l'aide du langage de programmation de son choix ou de pseudo-code algorithmique.

La gestion des déchets fait partie intégrante de la démarche de développement durable de l'AHM-23. Tous les déchets et chutes de bois sont triés et valorisés :

- Les chutes de bois sont commercialisées en allume-feux.
- La sciure et les copeaux de bois sont commercialisés pour diverses utilisations comme le compostage, le jardinage ou encore les toilettes sèches.

La sciure et les copeaux ne nécessitent pas de retraitement : on ramasse et on met en sac, alors que les allume-feux doivent être débités en tronçons identiques aussi bien en longueur qu'en largeur. Les valeurs énergétiques des essences de bois étant différentes, on distingue les allume-feux selon leur bois d'origine. Par exemple, le sapin a une valeur énergétique de 4.4 kWh/kg alors que le hêtre possède une valeur énergétique de 4.2 kWh/kg (valeurs données pour du bois sec de deux ans).

Il est important pour l'AHM-23 d'avoir connaissance de différents indicateurs :

- le chiffre d'affaires sur une période donnée, c'est-à-dire le cumul du montant de ses ventes ;
- le poids total d'une vente afin de mieux gérer ses livraisons ;
- des statistiques sur le volume total vendu par essence de bois sur différentes périodes dans le but d'étudier la valorisation des déchets.

L'AHM-23 demande à la société ALM-TIC de concevoir une application de gestion de ces déchets capable de répondre à ces besoins.

Le diagramme de classes a été produit, vous devez terminer l'écriture des méthodes manquantes.

Question 4.1

Écrire le constructeur de la classe AllumeFeu.

Question 4.2

Écrire la méthode *totalCA* de la classe Période.

Question 4.3

Expliquer ce que retourne la méthode *ventesAllumeFeu* de la classe Période.

Question 4.4

Écrire la méthode *getPoidsTotal* de la classe Commande.

Dossier documentaire

1. Présentation de la base de données

Schéma conceptuel des données et diagramme de classes

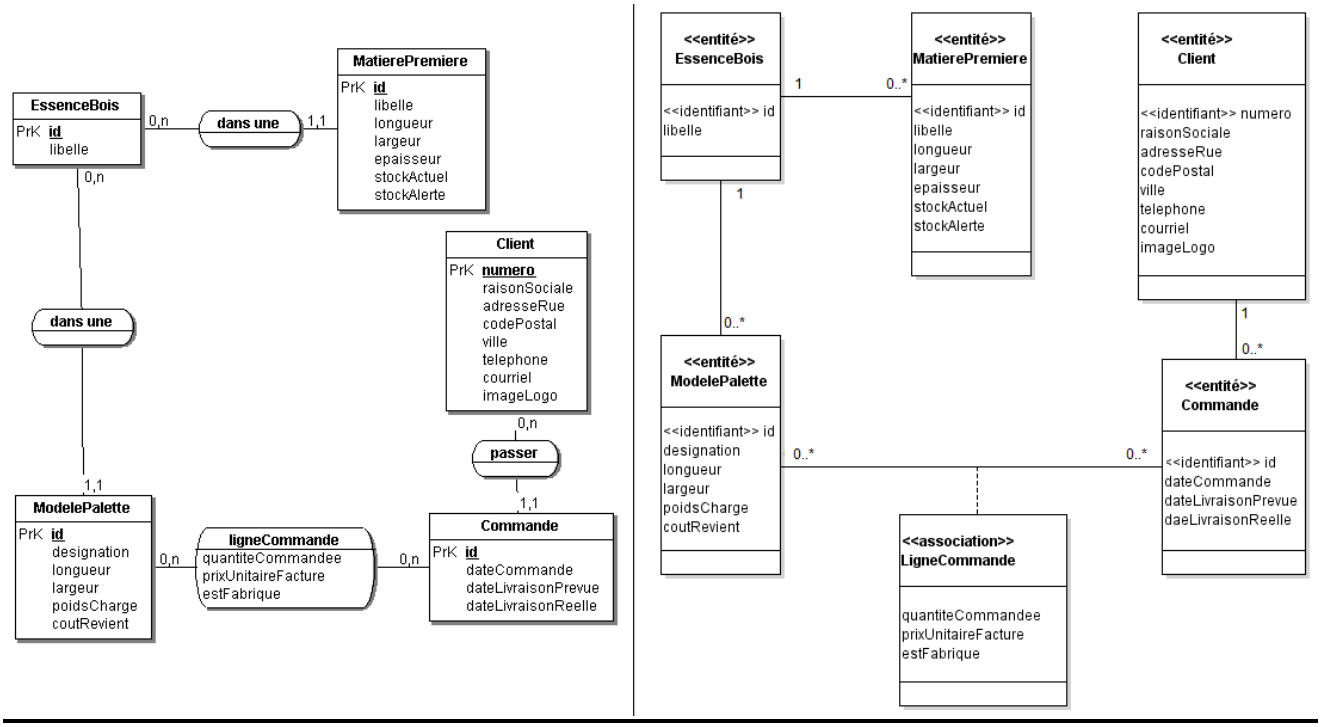


Schéma relationnel

Client (numero, raisonSociale, adresseRue, codePostal, ville, telephone, courriel, imageLogo)
Clé primaire : numero

Commande (id, dateCommande, dateLivraisonPrevue, dateLivraisonReelle, numeroClient)
Clé primaire : id
Clé étrangère : numeroClient en référence à numero de Client
Le champ dateLivraisonReelle est valorisé à null lors de la création de la commande et renseigné ultérieurement lors de la livraison.

EssenceBois (id, libelle)
Clé primaire : id

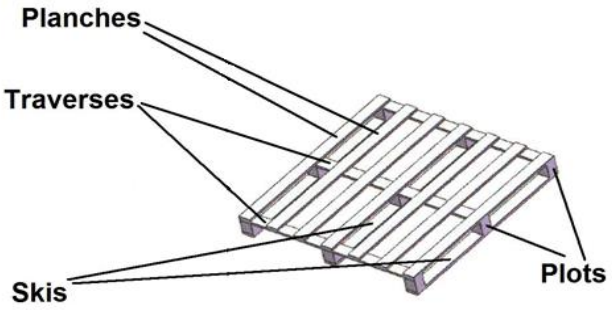
ModelePalette (id, designation, longueur, largeur, poidsCharge, coutRevient, idEssence)
Clé primaire : id
Clé étrangère : idEssence en référence à id de EssenceBois
Le champ coutRevient est le coût de revient unitaire du modèle de palette.

LigneCommande (idCommande, idModele, quantiteCommandee, prixUnitaireFacture, estFabrique)
Clé primaire : idCommande, idModele
Clés étrangères : idCommande en référence à id de Commande
idModele en référence à id de ModelePalette
Le champ estFabrique est initialisé à 0 lors de la création de la ligne de commande et prend la valeur 1 lorsque les palettes sont fabriquées.

MatierePremiere (id, libelle, longueur, largeur, epaisseur, stockActuel, stockAlerte, idEssence)
Clé primaire : id
Clé étrangère : idEssence en référence à id de EssenceBois

2. Nomenclature et plan des modèles de palette

Les composants de palette :



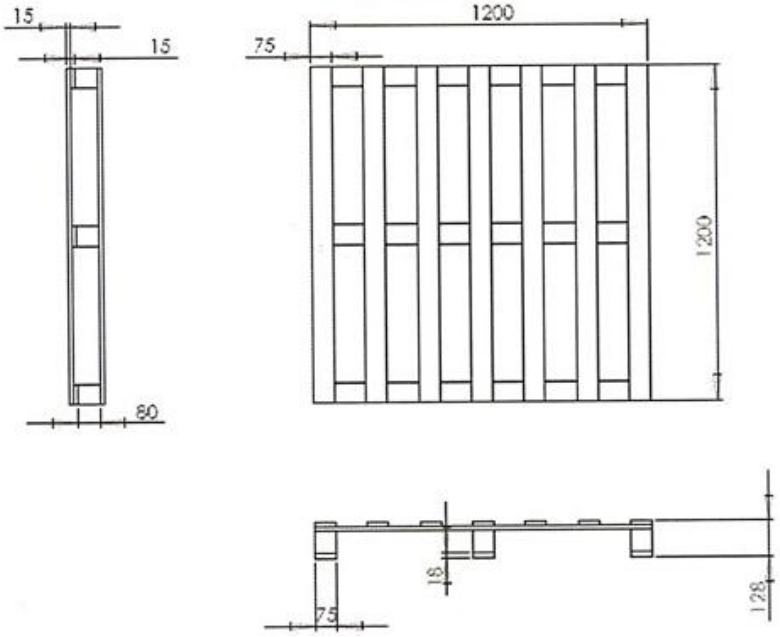
Exemple de plan de palette personnalisée :

IsoCoustic Client émetteur : ISOCOUSTIC Contact : Paul NIDGRON	**SPECIFICATION D'ACHAT** Palette bois personnalisée en résineux **Palettes 1200 x 1200 à plots**	**Date de création :** 02/10/2016 **Date d'application** : 04/01/2017

Composants :
- 7 planches 1200x75 ép. 15 ;
- 3 traverses 1200x80 ép. 15 ;
- 3 skis de 1200x75 ép. 18 ;
- 9 plots de 75x80 ép. 100.

Assemblage :
- 2 pointes par point assemblage ;
- Clous torsadés, crantés ou annelés ;
- Les clous ne dépassent pas du bois écartés d'au moins 25 mm, ils se situent au minimum à 15 mm du bord de la planche.

Poids de la charge :
500 Kg

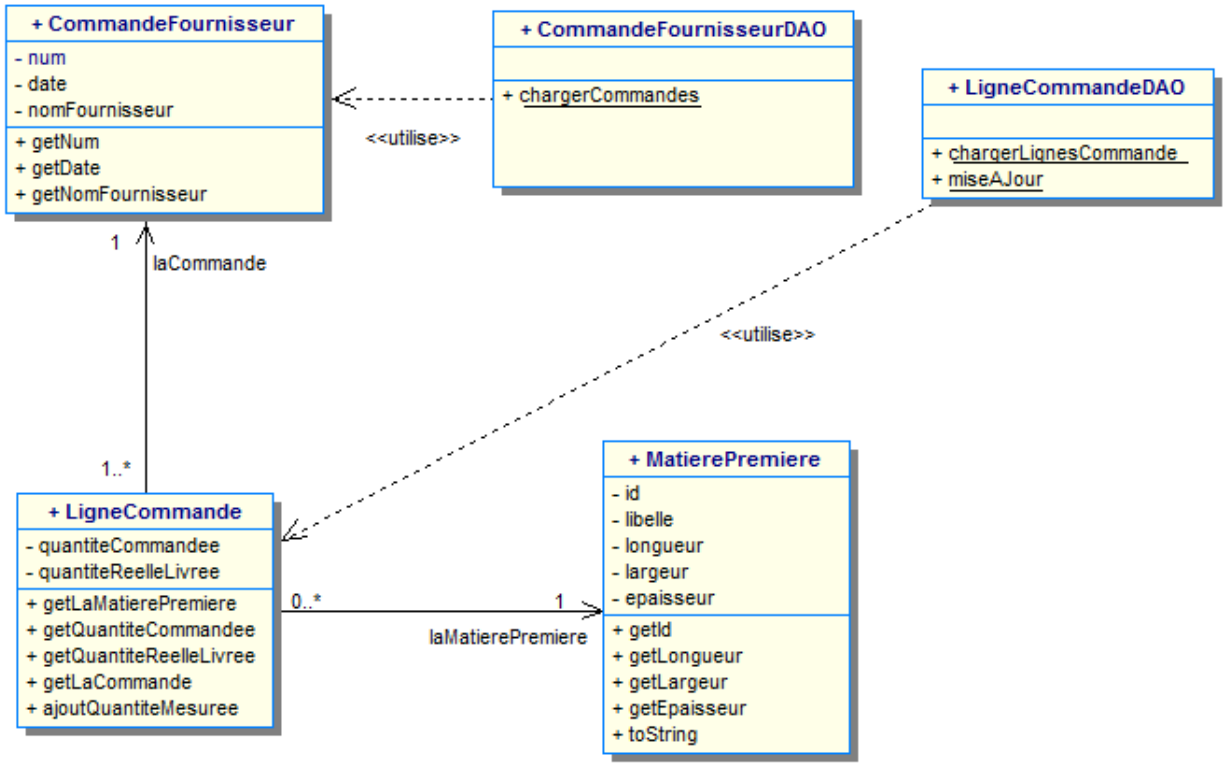


Format : 1200 x 1200			Produit : Palette produit fini		Echelle 1/15	Format : A4H
					Essence : Résineux	

Remarque :

- *Les informations relatives à l'assemblage, l'échelle et les formats ne sont pas à prendre en compte dans le cadre de cette étude.*

3. Description partielle des classes créées pour l'application de saisie des livraisons de bois



Les classes DAO contiennent les méthodes accédant à la base de données implantée sur la tablette mobile.

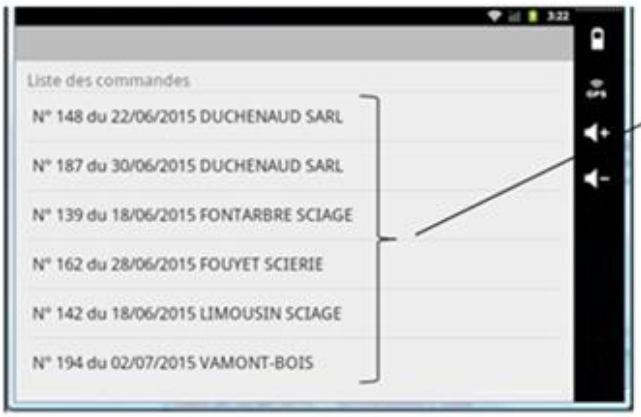
Cette base de données, qui contient les commandes venant d'être livrées, est chargée à partir de la base de données centralisée de l'application *web* de gestion du stock de bois.

```
Class CommandeFournisseur {
    //attributs privés
        private String num;
        private String date;
        private String nomFournisseur;

    //accesseurs
        public String getNum( )  { return this.num; }
        public String getDate ( )  { return this.date; }
        public String getNomFournisseur( )  { return this.nomFournisseur; }
}
```

```
Class CommandeFournisseurDAO {
    // Méthode retournant une collection d'objets "CommandeFournisseur" remplie avec
    // toutes les commandes figurant dans la base de données locale
        public static ArrayList<CommandeFournisseur> chargerCommandes() {
            - - - - - - - - - - - - - - -
        }
}

Class MatierePremiere {
    //attributs privés
        private int id;
        private String libelle;
        private String longueur;
        private String largeur;
        private String epaisseur;

    //accesseurs
        public int getId () { return this.id ; }
        public String getLongueur ( )  { return this.longueur; }
        public String getLargeur ( )  { return this.largeur; }
        public String getEpaisseur ( )  { return this.epaisseur; }
        @override
        public String toString ( )  { … }
        // retourne une chaîne sous la forme libelle longueur x largeur x epaisseur
        // par exemple : Lamelle en hêtre 800x70x15
}

Class LigneCommande {
    //attributs privés
        private CommandeFournisseur laCommande;
        private MatierePremiere laMatierePremiere;
        private float quantiteCommandee;
        private float quantiteReelleLivree;

    //accesseurs
        public MatierePremiere getLaMatierePremiere ( ) { return this.laMatierePremiere;}
        public float getQuantiteCommandee ( )  { return this.quantiteCommandee; }
        public float getQuantiteReelleLivree ( )  { return this.quantiteReelleLivree; }
        public CommandeFournisseur  getLaCommande() {…}

    //méthode cumulant à la quantité réelle livrée une quantité mesurée reçue en paramètre
        public void ajoutQuantiteMesuree (float uneQuantiteMesuree) {…}
}

Class LigneCommandeDAO {
    // Méthode retournant une collection d'objets "LigneCommande" remplie à partir de la
    // base de données locale, avec toutes les lignes de la commande dont le numéro est passé en
    paramètre
        public static ArrayList<LigneCommande> chargerLignesCommande(int unNumero) {
            - - - - - - - - - - - - - - -
        }

    // Méthode mettant à jour une ligne de commande de la base de données locale à partir des
    valeurs contenues  dans l'objet "LigneCommande" reçu en paramètre
        public static void miseAJour(LigneCommande uneLigne) {
            - - - - - - - - - - - - - - -
        }
}
```

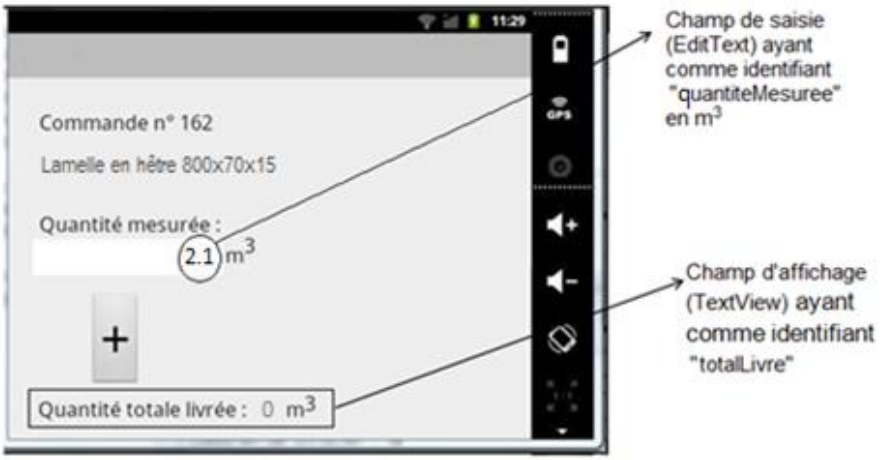
4. Maquette de l'interface graphique de l'application de saisie des livraisons de bois

Maquette de l'écran 1 :
Affichage des commandes livrées



Maquette de l'écran 2 :
Affichage des lignes de la commande sélectionnée par l'utilisateur sur l'écran précédent

Maquette de l'écran 3 :
Saisie de la quantité mesurée pour la ligne de commande sélectionnée sur l'écran précédent



Explications concernant l'écran n° 3 :

Les matières premières livrées sont conditionnées en bottes pouvant être de volumes différents. Si, pour la commande 162, il a été commandé 8 m^3 de lamelles 800x70x15, ces 8 m^3 seront par exemple livrés en :

- une botte de 2,1 m^3
- une botte de 3,1 m^3
- une botte de 3 m^3

Ce qui fera au total une livraison de 8,2 m^3, légèrement supérieure à la quantité commandée de 8 m^3.

Pour chaque botte, le chef d'atelier, après mesure de celle-ci, saisira son volume et cliquera sur le bouton "+", ce qui aura pour effet de cumuler le volume saisi dans la quantité totale livrée, affichée au bas de l'écran.

5. Extraits du code de l'application de saisie des livraisons de bois

Code implémentant le premier écran (affichage des commandes livrées) :

```
// Récupération de toutes les commandes fournisseurs présentes dans la base de données
// et stockage dans une collection de nom "lesCommandes"
ArrayList<CommandeFournisseur> lesCommandes ;
lesCommandes = CommandeFournisseurDAO.chargerCommandes();

// Déclaration d'une collection de chaînes de caractères, destinée à contenir les descriptions des
// commandes (numéro, date commande et nom du fournisseur) et à remplir la ListView.
ArrayList<String> lesDescriptions = new ArrayList<String>();

// Parcours de la collection "lesCommandes".  Pour chaque objet CommandeFournisseur : stockage de sa
// description (numéro, date commande et nom du fournisseur) dans la collection "lesDescriptions"
for (CommandeFournisseur  uneCommande : lesCommandes){
      lesDescriptions.add( "N° " + uneCommande.getNum()
      + " du " + uneCommande.getDate() + " "
      + uneCommande.getNomFournisseur());
}

// Création d'un objet "Adaptateur de données" qui va préparer les données de la collection des
// descriptions de commandes pour une ListView (paramètre "android.r.layout.simple_list_item_1")
ArrayAdapter<String> monAdapteur = new ArrayAdapter<String>
          (this,android.R.layout.simple_list_item_1, lesDescriptions);

ListView listeViewCommandes = (ListView) findViewById(R.id.listeCommandes);
TextView textView = new TextView(this);
// Affectation à la ListView d'une entête contenant le texte "Liste des commandes"
textView.setText("Liste des commandes");
listeViewCommandes.addHeaderView(textView);
listeViewCommandes.setAdapter(monAdapteur);  // Affectation de la liste des commandes
```

Début du code chargé d'alimenter le deuxième écran (affichage des lignes de la commande sélectionnée sur le premier écran) :

```
// Récupération, dans une variable "numeroCommande", du numéro de commande correspondant à
// la ligne sélectionnée par l'utilisateur sur le premier écran.
// Ce numéro a été transmis depuis le premier écran, par l'intermédiaire d'une donnée
// nommée "numero" que l'on récupère ici avec la méthode getExtras().
1. int numeroCommande = thisIntent.getExtras().getInt("numero");

// Récupération à partir de la base de données, des lignes de la commande concernée
// et stockage dans une collection de nom "lesLignes"
2. ArrayList<LigneCommande> lesLignes ;
3. lesLignes = LigneCommandeDAO.chargerLignesCommande(numeroCommande);

// Déclaration d'une collection de chaînes de caractères, destinée à contenir les descriptions des lignes
// de commandes (type de la matière première et dimensions). Cette collection servira à remplir la ListView.
4. ArrayList<String> lesDescriptions = new ArrayList<String>();
5. . .// . À COMPLÉTER SUR VOTRE COPIE . . .
```

Début du code exécuté au clic du bouton "+" du troisième écran :

```
1. EditText edQuantiteMesuree;
2. edQuantiteMesuree = (EditText) findViewById(R.id.quantiteMesuree);
3. float quantiteMesureeSaisie =
   Float.parseFloat(edQuantiteMesuree.getText().toString());
4.  .// . . À COMPLÉTER SUR VOTRE COPIE. . .
```

6. Consignes données par M. Laplace

M. Laplace ayant codé en partie l'application de saisie des livraisons de bois vous demande de terminer l'écriture du code s'exécutant au clic du bouton "+" de l'écran 3.

Pour cela, il vous laisse les consignes suivantes.

a) *L'objet **uneLigneCommande** de la classe LigneCommande contient la ligne de commande sur laquelle travaille l'utilisateur et est correctement instancié.*

b) *Les trois actions que vous devez coder sont :*
   1. *l'ajout de la quantité mesurée saisie à la quantité réelle livrée de cet objet ;*
   2. *l'affichage de la nouvelle quantité totale livrée au bas de l'écran ;*
   3. *la mise à jour de la base de données à partir des valeurs enregistrées dans cet objet.*

c) *Pour récupérer une valeur saisie dans un champ de saisie (EditText)*

Exemple avec un EditText ayant comme identifiant "salaireNet" :

```
EditText edSalaire;
edSalaire = (EditText) findViewById(R.id.salaireNet);
float salaire=Float.parseFloat(edSalaire.getText().toString());
```

d) *Pour renseigner un champ d'affichage (TextView)*

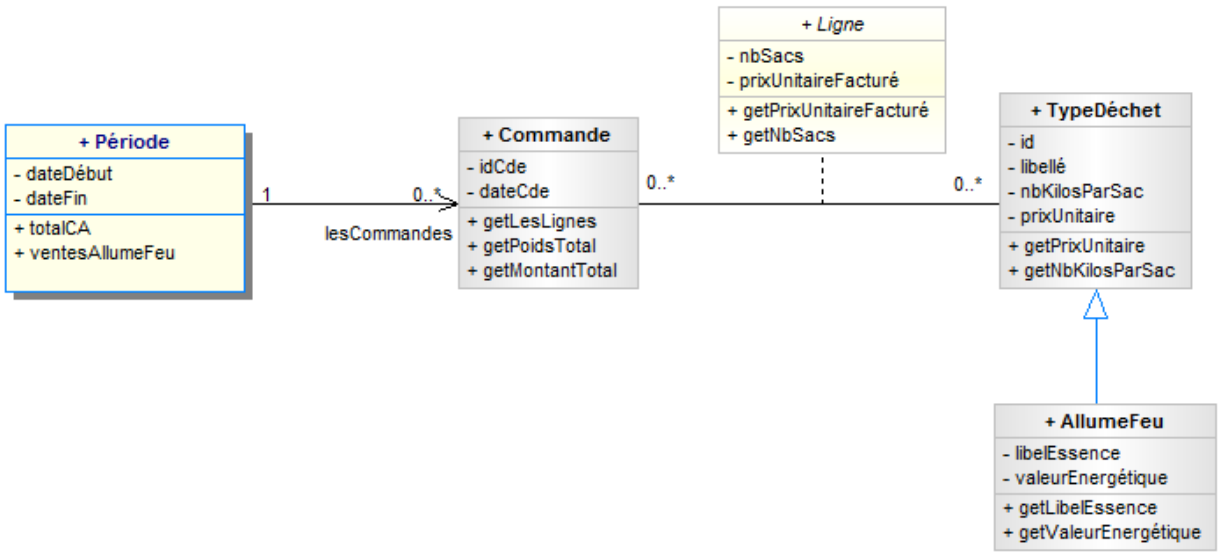
Exemple 1 avec un TextView ayant comme identifiant "prix" avec pour valeur 45.80 :

```
TextView tvPrix;
tvPrix = (TextView) findViewById(R.id.prix);
tvPrix.setText(String.valueOf(45.80));
```

Exemple 2 avec un TextView ayant comme identifiant "message" :

```
TextView tvMessage;
tvMessage = (TextView) findViewById(R.id.message);
tvMessage.setText("Produit actuellement indisponible");
```

7. Description des classes métiers créées pour la valorisation des déchets (extrait)



Classe Période

Attributs privés :

dateDébut : Date *// date de début de la période à prendre en compte*

dateFin : Date *// date fin de la période*

// collection des commandes pour la période

lesCommandes : Collection de <Commande>

Constructeur :

Période(uneDateDébut : Date, uneDateFin : Date)

Méthodes publiques :

Fonction totalCA() : réel *// retourne le chiffre d'affaires réalisé sur la période*
// soit le cumul du montant des ventes de la période

Fonction ventesAllumeFeu() : Dictionnaire<chaîne, entier>

```
Variables
        lesVentes : Dictionnaire<chaîne, entier>
        uneCde : Commande
        lesClés : Collection de <TypeDéchet>
        uneClé : TypeDéchet
Début
        lesVentes ← new Dictionnaire<chaîne, entier>()
        Pour chaque uneCde dans lesCommandes
                // parcours du dictionnaire lesLignes
                lesClés ← uneCde.getLesLignes.donnerLesClés()
                Pour chaque uneClé dans lesClés
                        Si uneClé.getType()= "AllumeFeu"  Alors
                                Si lesVentes.existe(uneClé.getLibelEssence()) Alors
                                        lesVentes.ajouter(uneClé.getLibelEssence(),
                                  lesVentes.donnerValeur(uneClé.getLibelEssence()) +
                                  uneCde.getLesLignes().donnerValeur(uneClé).getNbSacs())
                                Sinon
                                   lesVentes.ajouter(uneClé.getLibelEssence(),
                                  uneCde.getLesLignes().donnerValeur(uneClé).getNbSacs())
                                Finsi
                        Finsi
                FinPour
        FinPour
        Retourner lesVentes
Fin
```

Fin Classe

Classe Ligne

Attributs privés :

nbSacs : entier

prixUnitaireFacturé : réel *// prix de vente du sac*

Constructeur :

Ligne(unNbSacs : entier, unPrixUnitaireFacturé : réel)

Méthodes publiques :

Fonction getPrixUnitaireFacturé() : réel

Fonction getNbSacs() : entier

FinClasse

Classe Commande

Attributs privés :

idCde : entier *// identifiant de la commande*

dateCde : Date *// date de la commande*

lesLignes : Dictionnaire<TypeDéchet, Ligne>

// lesLignes est un dictionnaire : clé= un objet de la classe TypeDéchet

// valeur associée = un objet de la classe Ligne

Constructeur :

Commande(unIdCde :entier, uneDateCde : Date)

Méthodes publiques :

Fonction getLesLignes() : Dictionnaire<TypeDéchet, Ligne>

Fonction getPoidsTotal() : réel *// calcule et retourne le poids total de la commande*

// tout type de déchets confondus

Fonction getMontantTotal() : réel *// retourne le montant total de la commande*

FinClasse

Classe TypeDéchet

Attributs privés :

id : entier *// identifiant du type de déchet*

libellé : chaîne *// libellé du type de déchet (sciure, copeaux ou Allume-feux)*

nbKilosParSac : entier *// nombre de kilos par sac*

prixUnitaire : réel *// prix de vente du sac*

Constructeur :

TypeDéchet(unId :entier, unLibellé : chaîne, unNbKilosParSac : entier,
unPrixUnitaire : réel)

Méthodes publiques :

Fonction getPrixUnitaire() : réel

Fonction getNbKilosParSac() : entier *//retourne le nombre de kilos contenus dans un sac*

FinClasse

Classe AllumeFeu : hérite de TypeDéchet

Attributs privés :

libelEssence : chaîne *// libellé de l'essence (sapin, hêtre….)*

valeurEnergétique : réel *// valeur énergétique (en kWh/kg) de l'essence*

Constructeur :

AllumeFeu (unId :entier, unLibellé : chaîne, unNbKilosParSac : entier,
unPrixUnitaire : réel, unLibelEssence : chaîne , uneValeurEnergétique : réel)

Méthodes publiques :

Fonction getLibelEssence() : chaîne

Fonction getValeurEnergétique() : réel

FinClasse

8. Description des classes techniques (extrait)

Classe Collection de <TypeElément>

Méthodes publiques :

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 0.
procédure ajouter (unObjet : TypeElément) *// Ajoute l'objet unObjet à la collection.*

Fin Classe

Exemple d'utilisation de la collection :

```
lesClients : Collection de <Client>
unClient : Client
lesClients ← new Collection de <Client> ()
…
lesClients.ajouter (unClient)
…
leClient : Client
Pour chaque leClient dans lesClients      // Parcours de la collection :
    …
FinPour
```

Classe Objet *// toute classe dérive implicitement de la classe Objet*

Méthodes publiques :

fonction getType () : chaîne
// retourne dans une chaîne de caractères le nom de classe de l'objet courant
// par exemple : Client unClient ← new Client(…)
// unClient.getType() retournera la chaîne de caractères ''Client''

FinClasse

Classe Dictionnaire <*TypeClé, TypeValeur*>

// ensemble d'éléments (clé, valeur) permettant d'extraire une valeur (de type TypeValeur) à partir
// de sa clé (de type TypeClé) ; à une clé présente dans le dictionnaire correspond
// une et une seule valeur
Méthodes publiques :

fonction donnerValeur(clé : TypeClé) : TypeValeur
// accède à la valeur dont la clé est passée en paramètre.
fonction donnerLesClés() : Collection de <TypeClé>
// retourne une collection contenant toutes les clés du dictionnaire.
fonction existe(clé : TypeClé) : booléen
// retourne vrai si l'élément dont la clé est passée en paramètre
// existe dans le dictionnaire
procédure ajouter(clé : TypeClé, uneValeur : TypeValeur)
// ajoute un nouvel élément (clé, valeur) au dictionnaire
// ou remplace par uneValeur, la valeur de l'élément identifié par la clé.

Fin classe

Exemple de parcours d'un dictionnaire :

```
unDico : Dictionnaire de <TypeClé, TypeValeur>
uneValeur : TypeValeur
uneClé : TypeClé
lesClés : Collection de <TypeClé>
unDico ←new Dictionnaire de <TypeClé , TypeValeur>
lesClés ← unDico.donnerLesClés()
        Pour chaque uneClé dans lesClés
                uneValeur ← unDico.donnerValeur(uneClé)
        FinPour
```