Gestion des problèmes et du changement

Lorsqu'un revendeur souhaite acheter des sacs, il doit se rendre au siège social à la fois pour voir les produits et pour passer commande. La commerciale ou son assistant se connecte alors à l'application GestSac, choisit dans le menu le choix "Gestion des commandes" puis "Enregistrer une commande". Le fonctionnement de l'application est décrit dans les documents 1A à 1D.

L'application GestSac donne satisfaction auprès des commerciaux. Les incidents de niveau de gravité « bloquant » et « majeur » sont corrigés au fur et à mesure mais certains incidents de niveau de gravité « mineur » sont en attente de traitement pour la prochaine version de l'application. Afin de rendre plus fiable la couche métier, il a été décidé de prendre en charge les incidents de niveau de gravité « mineur » pour lesquels la couche métier est concernée.

Comme les informaticiens en place souhaitent s'appuyer sur la couche métier de l'application existante (développée sous la plateforme .NET) pour le futur site vitrine, ils entreprennent une revue de code des classes composant cette couche métier afin de la fiabiliser. Cette revue de code concerne la vérification du respect des règles de développement et la correction d'incidents mineurs. Les efforts portent actuellement sur la classe Commande dont un extrait est fourni dans le document 1E (appliquer les corrections sur les autres classes de l'appli).

Vous participez à cette revue de code et disposez de la documentation technique suivante :

- Un extrait des règles de développement, document 1D,
- Un extrait de la classe Commande, document 1E,
- Un extrait de la classe de tests unitaires de la classe Commande, document 1F,
- La classe Dictionnary du framework .NET, document 1G,
- La classe Assert du framework .NET, document 1H,
- Fiche d'incident n° GS53 de niveau de gravité « mineur », document 1I.

1.1	**Exposer de manière justifiée les règles de développement qui ne sont pas respectées dans l'extrait de code de la classe Commande.**
1.2	**Procéder à l'analyse du problème exposé dans la fiche d'incident n°GS53 en rédigeant le texte renseignant la rubrique « Cause du problème » de la fiche d'incident sur votre copie.**

Vos collègues vous indiquent que les tests unitaires de la classe Commande n'ont pas permis de détecter plus tôt le problème relaté dans la fiche d'incident n°GS53.

1.3	**Apporter les modifications nécessaires à la classe Commande et à la classe de test associée pour vous assurer que le problème ne se produise plus.**

La perspective de mise en place d'un site vitrine oblige le service informatique à réfléchir d'une part à l'évolution fonctionnelle de l'application GestSac qui servira à définir le cahier des charges de la future application en ligne et d'autre part aux changements organisationnels induits par ce nouveau service.

Les besoins fonctionnels portent sur
- L'enregistrement des revendeurs : l'application doit permettre au moment de la commande la saisie d'un nouveau revendeur.
- La gestion des stocks : un commercial doit être en mesure de signaler à un revendeur que la quantité qu'il souhaite commander n'est pas en stock.

La documentation technique suivante doit vous guider pour circonscrire les besoins d'évolution :

- Le schéma relationnel de la base de données, document 1A,
- La copie d'écran de l'enregistrement d'une commande dans l'application GestSac, document 1B,

- Le cas d'utilisation "Enregistrement d'une commande", document 1C.

1.4	**Justifier en quoi l'application actuelle répond ou non aux besoins fonctionnels énoncés ci-dessus.**

Si l'application GestSac satisfait ses utilisateurs, on peut s'interroger sur son usage avec d'autres utilisateurs que les commerciaux pour lesquels elle a été conçue.

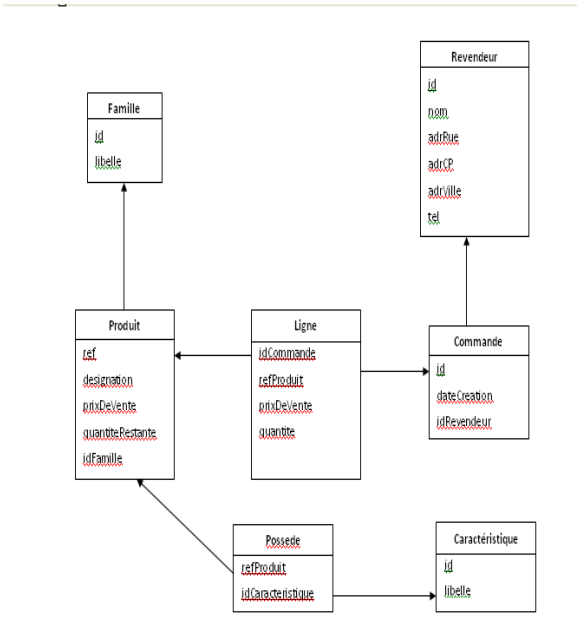
1.5	**Expliquer ce qui dans l'interface de l'application actuelle pourrait être amélioré pour faciliter son usage par de nouveaux utilisateurs.**

Malgré la mise en place du site vitrine, certains revendeurs continueront à se rendre à la salle d'exposition du siège social.

1.6	**Indiquer les modifications techniques, humaines et organisationnelles engendrées par l'arrivée du nouveau canal de distribution.**

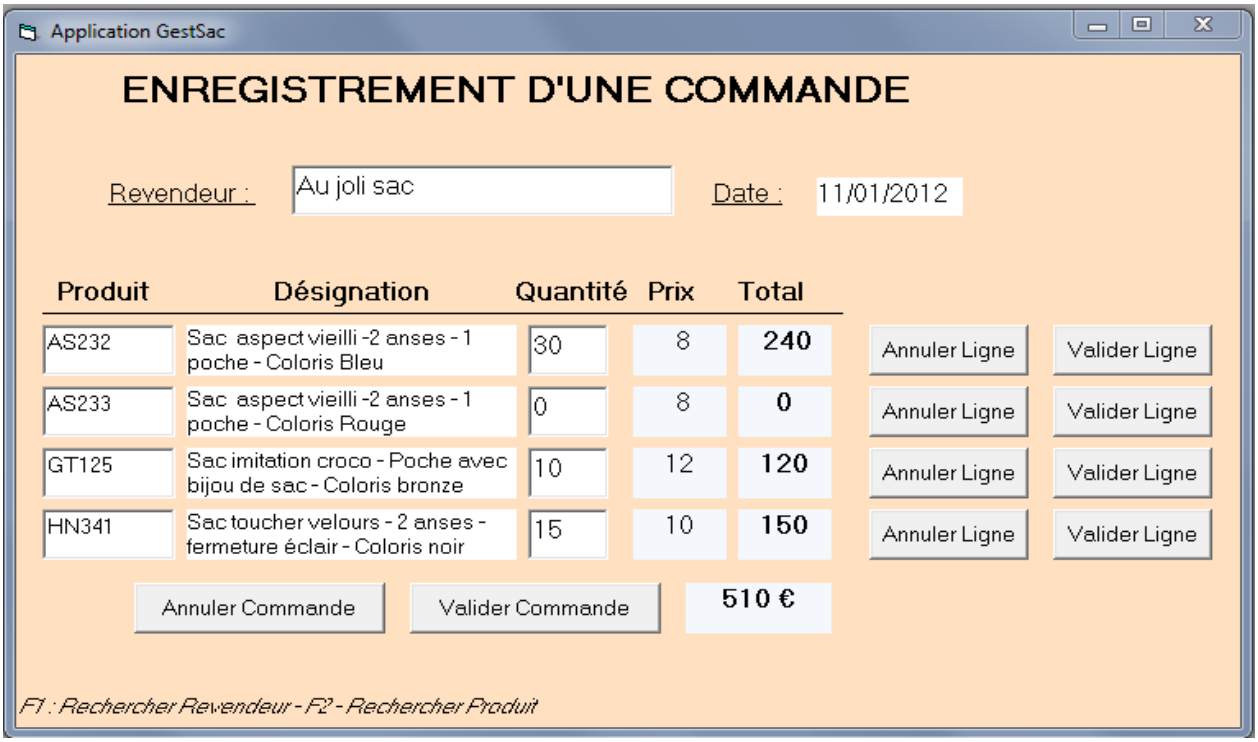
Annexes

DOCUMENT 1A - Schéma relationnel de la base de données

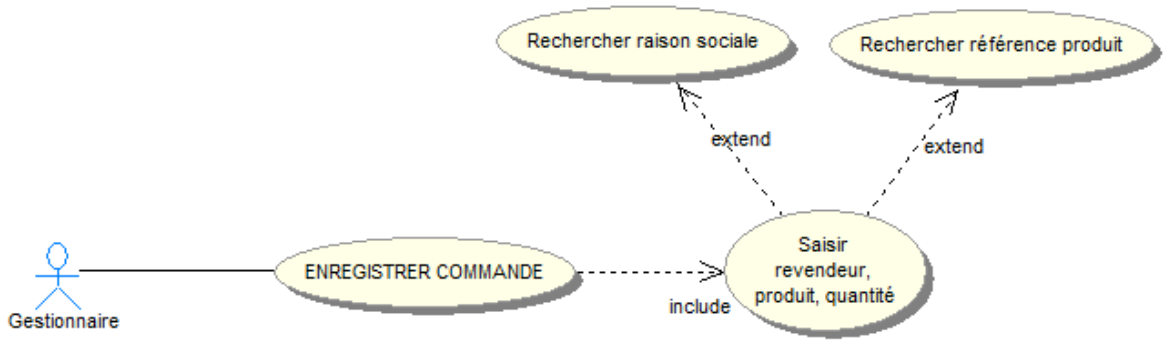


L'attribut prixDeVente dans la relation Ligne permet de conserver le prix de vente au moment de la transaction

DOCUMENT 1B - Copie d'écran de l'enregistrement d'une commande dans l'application GestSac



DOCUMENT 1C - Cas d'utilisation "Enregistrement d'une commande".



Cas d'utilisation **: Enregistrement d'une commande dans l'application GestSac**

Scénario nominal :

1. Le commercial saisit la raison sociale du revendeur.
2. Le système recherche le revendeur à partir de sa raison sociale et l'affiche.
3. Le commercial saisit la référence et la quantité du produit à commander.
4. Le système recherche le produit à partir de la référence saisie, affiche sa désignation et son prix puis vérifie le stock restant. S'il est inférieur à la quantité souhaitée, le système modifie la quantité saisie par la quantité restante. Pour terminer, le système valide la ligne de commande, met à jour les totaux de la ligne et de la commande, et place le curseur à la ligne suivante pour une nouvelle saisie de produit à commander.
5. --- Le *commercial peut recommencer au point 4 pour continuer la saisie d'autres produits* ---
6. Le commercial valide la commande.
7. Le système enregistre les lignes de commandes, la commande, met à jour les stocks de produit selon les quantités commandées et informe l'utilisateur que la commande a bien été enregistrée.

Extensions :

- 2. . Le système ne trouve pas le revendeur :
  - Le système informe l'utilisateur qu'il n'a pas trouvé le revendeur.
  - Retour à l'étape 1.
- 4. Le système ne trouve pas le produit :
  - Le système informe l'utilisateur qu'il n'a pas trouvé le produit.
  - Retour à l'étape 3.

DOCUMENT 1D – Extrait des règles de développement

1. Toute méthode publique d'une classe sera précédée d'une documentation qui comprendra au minimum le résumé, la description des paramètres et du résultat suivant le format ///.

2. Le nom (identificateur) d'une classe respectera la notation Pascal : la première lettre du nom de méthode et la première lettre de chaque mot présent dans l'identificateur sont en majuscules. Par exemple, LigneCommande respecte la notation Pascal.

3.Le nom (identificateur) des attributs d'une classe, paramètres formels et des variables locales respectera la notation Camel : la première lettre du nom est en minuscules et la première lettre de chaque mot présent dans l'identificateur est en majuscules. Par exemple, uneQte respecte la notation Camel.

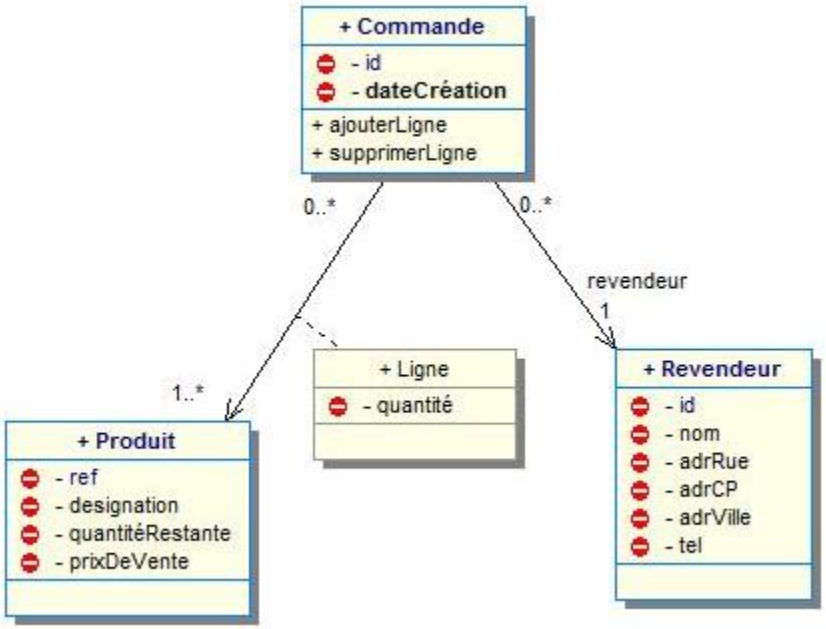
4. Le nom d'une méthode est un verbe, ou un groupe verbal. Il respectera la convention Pascal. Les méthodes qui permettent de lire (resp. écrire) directement une variable privée d'instance sont préfixées par Get (resp. set), suivi du nom de la variable.

5. Le nom des méthodes, paramètres et variables doit être le plus explicite possible et informer de leur rôle. Il faut privilégier la lisibilité à la concision.

6. Le nom des méthodes, paramètres et variables ne contient que des lettres non accentuées ou des chiffres : le tiret bas, trait d'union ou tout autre caractère non alphanumérique sont interdits.

DOCUMENT 1E – Extrait de code de la classe Commande

La classe Commande s'insère dans le diagramme de classes suivant :



Classe Produit – Méthodes publiques (extrait)

```
public class Produit
    public Produit(String uneRef, String uneDesignation, int uneQteRestante,
                   double unPUHT) // constructeur
    public int getQuantiteRestante()          // accesseurs sur la quantité restante
    public void setQuantiteRestante(int value)
```

```java
public class Commande {
	int          Id;
	Date         DateCreation;
	Revendeur    revendeur;
	HashMap<Produit, Integer> lignes;

	/**
	 * Crée une instance de classe Commande.
	 * @param    i      l'id de commande
	 * @param    d      la date de commande
	 * @param    r      le revendeur
	*/
	public Commande(int i, Date d, Revendeur r) {
		Id           = i;
		DateCreation = d;
		lignes       = new HashMap<Produit, Integer>();
		revendeur    = r;
	}

	/**
	 * Fournit la liste des lignes de la commande sous forme de dictionnaire
	 * en clé, l'instance de produit, en valeur, la quantité commandée
	 * @return   liste des lignes de commande
	*/
	public HashMap<Produit, Integer> getLignes() {
		return lignes;
	}

	/**
	 * Ajoute une ligne de commande à l'instance de commande courante
	 * La quantité de la ligne est ajustée à la qté du produit restant en stock.
	 * Si le produit figure déjà dans une ligne de commande, la quantité demandée
	 * s'ajoute à la quantité actuelle.
	 * @param    unProd : instance de classe Produit
	 * @param    uneQte : nombre de produits commandés
	*/

	public void ajouterLigne(Produit unProd, int uneQte){
		int qteRestante = unProd.getQuantiteRestante();
		if ( uneQte > qteRestante ) {
			uneQte = qteRestante;
		}
		if (lignes.containsKey (unProd))
		{
			int ancienneQte = lignes.get (unProd);
			uneQte += ancienneQte ;
			lignes.remove(unProd) ;
		}
		lignes.put(unProd, uneQte);
	}

	public void supprimer_ligne(Produit unProd){
		if (lignes.containsKey(unProd)) {
			lignes.remove(unProd);
		}
	}
}
```

DOCUMENT 1F – Extrait de la classe de tests unitaires de la classe Commande

```
/**
 * Classe de test de la classe Commande
 * Hérite de la classe TestCase issue du framework de tests unitaires Junit
 * Permettra d'enchaîner l'exécution de toutes les méthodes commençant par test
 * L'exécution s'arrête à la première affirmation non respectée, avec le
 * message d'erreur associé qui s'affiche. Rien n'est affiché concernant les
 * affirmations respectées, l'exécution se termine avec le message Test réussi
 * si toutes les affirmations ont été respectées.
*/
public class TestCommande extends TestCase {
    /**
     * Méthode de test pour ajouterLigne
     * Cas de test sur un produit n'ayant pas encore fait l'objet d'une ligne de commande
    */
    public void testAjouterLigneNouveauProduit() {
        // création d'une commande et d'un produit
        Commande commande = new Commande(1, new Date(), new Revendeur(100,"Au joli sac"));
        Produit produit = new Produit("AS232", "Sac aspect vieilli - 2 anses", 10, 8);


        // cas de test d'une ligne de commande d'un produit non encore commandé
        commande.ajouterLigne(produit, 2);
        HashMap<Produit, Integer> lesLignes = commande.getLignes();
        Assert.assertEquals("Une ligne supplémentaire", lesLignes.size(), 1);
        Assert.assertNotNull("Ligne ajoutée accessible", lesLignes.get(produit));
        int qte = lesLignes.get(produit);
        Assert.assertEquals("Quantité de la ligne égale à celle demandée", qte, 2);

        // cas de test d'une nouvelle ligne avec une quantité en stock insuffisante
        Produit autreProduit = new Produit("HN341","Sac toucher velours - 2 anses",5,10);

        commande.ajouterLigne(autreProduit, 10);
        lesLignes = commande.getLignes();
        Assert.assertEquals("Une ligne supplémentaire", lesLignes.size(), 2);
        Assert.assertNotNull("Ligne ajoutée accessible",lesLignes.get(autreProduit));
        qte = lesLignes.get(autreProduit);
        Assert.assertEquals("Quantité égale à celle en stock", qte, 5);
    }
    /**
     * Méthode de test pour ajouterLigne
     * Cas de test d'un produit déjà commandé dans la commande
    */
    public void testAjouterLigneProduitDejaCommande() {
        // création d'une commande, d'un produit et d'une première ligne de commande
        Commande commande = new Commande(1, new Date(), new Revendeur(100,"Au joli sac"));
        Produit produit = new Produit("AS232", "Sac aspect vieilli - 2 anses", 10, 8);

        commande.ajouterLigne(produit, 2);

        // cas de test d'une ligne existante avec une quantité différente
        commande.ajouterLigne(produit, 5);
        HashMap<Produit, Integer> lesLignes = commande.getLignes();
        Assert.assertEquals("Pas de ligne supplémentaire", lesLignes.size(), 1);
        Assert.assertNotNull("Ligne modifiée accessible", lesLignes.get(produit));
        int qte = lesLignes.get(produit);
        Assert.assertEquals("Quantité égale à celle demandée", qte, 5);

        // cas de test d'une ligne existante avec une quantité en stock insuffisante
        commande.ajouterLigne(produit, 15);
        lesLignes = commande.getLignes();
        Assert.assertEquals("Pas de ligne supplémentaire", lesLignes.size(), 1);
        Assert.assertNotNull("Ligne modifiée accessible", lesLignes.get(produit));
        qte = lesLignes.get(produit);
        Assert.assertEquals("Quantité égale à celle en stock", qte, 10);
      }
```

DOCUMENT 1G – Présentation de la classe HashMap du framework Java

```
/**
La classe HashMap permet de mémoriser un dictionnaire d'éléments (clé, valeur). Toute valeur (de
type ValueType) peut être extraite à partir de sa clé (de type KeyType)  à une clé présente dans
le dictionnaire correspond une et une seule valeur
*/
```

***Class* HashMap <KeyType, ValueType>**

```
public void put (KeyType key, ValueType value)
      // ajoute un élément (key, value). Remplace la valeur existante si
      // l'élément existe déjà pour la clé spécifiée.

public ValueType get (KeyType key)
      // retourne la valeur correspondant à la clé spécifiée, ou null si la clé
      // spécifiée est inexistante.

public ValueType remove (KeyType key)
      // retire du dictionnaire l'élément correspondant à la clé spécifiée et
      // retourne la valeur qui y était associée. Ne fait rien et retourne null
      // si la clé était inexistante.

public Boolean containsKey (KeyType key)
      // retourne true si l'élément dont la clé est passée en paramètre est
      // présent dans le dictionnaire, false sinon

public Set<KeyType> keySet ()
      // retourne un ensemble des clés présentes dans le dictionnaire d'éléments
```

DOCUMENT 1H– Présentation de la classe Assert du framework Java

```
/**
La classe Assert réunit un ensemble de méthodes statiques d'assertion utiles pour l'écriture des
tests. Une méthode d'assertion a pour rôle de vérifier si une affirmation est ou non respectée.
Dans le cas où elle ne l'est pas, la méthode déclenche une exception. Si non, elle ne fait rien.
*/
```

***Class* Assert**

```
public static void AssertEquals(String message, int expected, int actual)
      // vérifie si les valeurs entières expected et actual sont égales.
      // Si elles ne le sont pas, déclenche une exception de type
      // AssertException comportant le message d'erreur spécifié.

public static void AssertEquals(String message, Object expected, Object actual)
      // vérifie si les deux objects expected et actual sont égaux.
      // S'ils ne le sont pas, déclenche une exception de type
      // AssertException comportant le message d'erreur spécifié.

public static void AssertNull(String message, Object obj)
      // vérifie si l'objet obj est une référence nulle. S'il ne l'est pas,
      // déclenche une exception de type AssertException comportant le message
      // d'erreur spécifié.

public static void AssertNotNull(String message, Object obj)
      // vérifie si l'objet obj n'est pas une référence nulle. S'il l'est,
      // déclenche une exception de type AssertException comportant le message
      // d'erreur spécifié.
```

Document 1I - Fiche d'incident n° GS53

Description incident

Date signalement :	Rédacteur :	Niveau de gravité :
25/02/2012	*Assistant commercial Arnaud Belloir*	☐Bloquant ☐Majeur ☐Mineur
Environnement concerné	Matériel :	Logiciel:
	Poste Caisse n°2	*GestSac*

Description du problème avec éventuelles captures d'écran, messages d'erreurs :

J'ai constaté que lorque l'on commandait plusieurs fois le même produit, la quantité totale peut dépasser la quantité en stock : sur l'exemple ci-dessous, j'avais commencé par commander 20 sacs référence AS232, puis 10. L'application m'affiche 30 alors qu'en stock il n'y en a que 25.

N°commande	Référence produit	Quantité commandée	Prix de vente
345126	AS232	30	8
345126	AS233	0	8
345126	GT125	10	12

Cause du problème

Date analyse : Rédacteur :

Description de la cause :

Actions réalisées

Date réalisation : Rédacteur : Version de correction :