
PROGRAMMATION ORIENTÉE OBJET IG2I, LE3 – 2017-2018

CTP – 18 JANVIER 2018

Diego Cattaruzza, Maxime Ogier

Conditions de l'examen

- **durée** : 4h00 ;
- ordinateurs personnels autorisés ;
- documents et pages internet autorisés ;
- accès à Moodle autorisé ;
- **les moyens de communication** (mails, messageries instantanées, Facebook, Tweeter, téléphones portables, Google Drive, etc.) **sont INTERDITS** ; le fait qu'un moyen de communication ne soit pas listé ci-dessus ne vous donne pas le droit de l'utiliser ;
- si vous utilisez du code que vous n'avez pas développé par vous même lors du CTP, **vous devez citer vos sources (dans les commentaires)** ; ne pas citer ses sources est une fraude.

Rendu

Vous déposez sur Moodle, dans la zone de dépôt **Dépôt du CTP 2017/2018** une archive compressée de votre répertoire de projet. Cette archive doit être nommée **CTP_Nom_Prenom.zip** avec votre nom et votre prénom. **Un nommage différent donnera lieu à deux points de pénalisation. Cette règle est également valide pour toute proposition de nommage de ce CTP : projet, paquetages, classes, méthodes, attributs, ...**

Vérifiez bien que les fichiers sources que vous avez réalisés lors du CTP sont présents dans le dossier que vous déposez sur Moodle (oui, vérifiez bien cela : toutes les années il y a des répertoires vides ou contenant les mauvais fichiers ! Il ne sera pas possible de renvoyer une nouvelle version de votre projet après la date limite de dépôt).

Notation

Cette épreuve de CTP est évaluée sur 20 points. Pour chaque question, la moitié des points est attribuée si le code fonctionne correctement. **L'autre moitié des points est attribuée en fonction de la qualité du code** (respect des principes de la programmation orientée objet, efficacité des algorithmes) et de la présentation du code (indentation correcte, respects des conventions Java dans les noms de classe, attributs, méthodes et variables, commentaires au format Javadoc).

Consignes

Le sujet est composé de **6** sections liées entre elles. Il est fortement conseillé de répondre aux questions dans l'ordre proposé. Le sujet est conçu pour que le code développé pour certaines questions soit réutilisé pour répondre aux questions suivantes.

Vous êtes bien sûr autorisés à rajouter les attributs ou méthodes que vous jugez nécessaires même si ce n'est pas dit explicitement dans le sujet.

Attestation sur l'honneur

Je soussigné(e) certifie que le projet déposé sur moodle est issu d'un travail purement personnel. Je certifie aussi avoir cité toute source de code externe dans les commentaires de mon code.

Fait à

le

signature

Sujet

HappyDays est une entreprise qui produit et distribue des vêtements. Dans ce CTP, nous ne nous intéressons pas aux opérations de production ni à la distribution, mais nous nous focalisons sur la gestion du stock de notre chère entreprise (eh oui, certains parmi vous l'ont déjà rencontrée lors du CTP de l'année dernière).

Le directeur de *HappyDays* vous contacte pour mettre en place un outils de gestion de stock. Dans son entrepôt, *HappyDays* dispose d'un ensemble de produits. Le directeur vous dit que chez *HappyDays*, pour chaque produit, on a défini une quantité minimale et une quantité maximale à garder en stock. La première représente la quantité que l'on souhaite avoir toujours disponible dans l'entrepôt pour être certain de toujours pouvoir satisfaire les commandes des clients de l'entreprise. La quantité maximale est une quantité qu'il n'est pas souhaitable de dépasser pour des raisons d'espace (l'entrepôt n'est pas infini). En tous cas, ce sont des conditions souhaitables, mais ce ne sont pas des contraintes fortes. Il est donc tout à fait possible d'avoir en stock un nombre de pièces inférieur à la quantité minimale ou supérieur à la quantité maximale.

HappyDays gère régulièrement des *mouvements* des produits. Un mouvement est toujours associé à un seul produit et peut représenter :

- la commande d'un client pour un produit ; ce mouvement entraîne la sortie d'un nombre de pièces du produit et il est donc associé à une quantité négative (car le stock diminue) ;
- la commande de *HappyDays* pour se ravitailler d'un produit ; ce mouvement entraîne l'entrée d'un nombre de pièces du produit et il est donc associé à une quantité positive (car le stock augmente).

Le modèle conceptuel de données ainsi que le modèle logique de données de notre application sont présentés dans les Figures 1 et 2 respectivement.



Figure 1: Modèle conceptuel de données pour l'application du *HappyDays*.

Le modèle conceptuel de données contient deux classes : **Produit** et **Mouvement**. Un *produit* représente très simplement un produit que l'on peut retrouver dans l'entrepôt de *HappyDays*. Il est défini par son nom, le nombre minimal de pièces que l'on souhaite toujours avoir en stock, et le nombre maximal de pièces qu'il n'est pas souhaitable de dépasser pur des raisons d'espace. Un *mouvement*

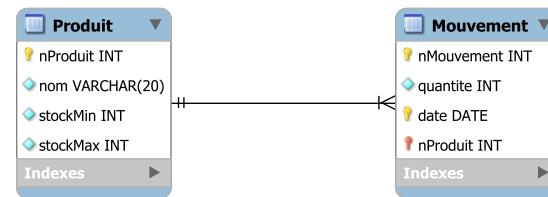


Figure 2: Modèle logique de données pour l'application du *HappyDays*.

représente un mouvement dans le stock : un ajout de nouvelles pièces, ou une sortie de certaines pièces (pour satisfaire la commande d'un client). Un mouvement est donc défini par le produit concerné, la quantité (positive pour un ajout, négative pour une sortie) et la date à laquelle il est effectué.

1 Base de données et connexion avec l'application (1 point)

Votre base de donnée doit **impérativement** s'appeler **gestionStock**. Le nom d'utilisateur et le mot de passe sont identiques et doivent **impérativement** être **stock**. Chaque nommage différent donnera lieu à **2 points** de **pénalisation**.

Question 1. Mettez en place la base de données pour l'application du *HappyDays*, en utilisant les scripts SQL disponibles sur Moodle, dans l'ordre suivant :

- 01_derbyGestionStock.sql
- 02_derbyStockMovements.sql
- 03_derbyStockMovements.sql
- 04_derbyStockMovements.sql
- 05_derbyStockMovements.sql
- 06_derbyStockMovements.sql

Les scripts sont prévus pour un SGBD Derby, donc le plus simple est de créer votre base de données sur Netbeans, comme vu dans le TP2. À la fin, vous devez obtenir 30 produits et 24314 mouvements.

Question 2. Créez un nouveau projet nommé **CTP_Nom_Prenom** avec votre nom et votre prénom. Ajoutez un paquetage **metier** et une classe **RequeteGestionStock**. Dans cette classe, ajoutez un attribut de type **Connexion** et faites en sorte que la connexion à la base de donnée soit établie lors de l'appel au constructeur par défaut. Mettez en place le *design pattern* singleton

de telle sorte qu'il n'y ait pas plus d'une connexion à la base de données. Testez le bon établissement de la connexion à la base de données.

Remarque : n'oubliez pas d'importer le pilote JDBC dans les bibliothèques du projet.

2 Mise en place du modèle objet (1 point)

À présent, nous souhaitons ajouter les classes **Produit** et **Mouvement** pour le modèle objet. Vous pouvez vous référer à la Figure 1 qui présente le modèle conceptuel de données.

Question 3. Ajoutez à votre projet un paquetage **modele**. Ajoutez dans ce paquetage une classe **Produit**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString()**. Vérifiez le bon fonctionnement de cette classe.

Question 4. Ajoutez dans le paquetage **modele** une classe **Mouvement**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString()**. Vérifiez le bon fonctionnement de cette classe.

3 Requêtes sur la base de données (4 points)

Question 5. Dans la classe **RequeteGestionStock**, ajoutez une méthode **public List<Produit> ensProduits() throws SQLException** qui renvoie la liste des produits ordonnés par ordre croissant du nom. Testez le bon fonctionnement de cette méthode.

Question 6. Dans la classe **RequeteGestionStock**, ajoutez une méthode **public List<Produit> ensProduits(String nom) throws SQLException** qui renvoie la liste des produits dont le nom commence par la chaîne de caractères **nom** (pas de sensibilité à la casse de la chaîne de caractères donnée en paramètre). Testez le bon fonctionnement de cette méthode.

Question 7. On souhaite à présent ajouter à un produit l'ensemble des mouvements qui lui sont associés. Dans la classe **RequeteGestionStock**, ajoutez une méthode **public void addAllMouvementsToProduit (Produit produit) throws SQLException** qui ajoute au produit **produit** l'ensemble des ses mouvements. Bien évidemment, vous pouvez ajouter toutes les modifications que vous jugez nécessaires dans le modèle objet. Par ailleurs, faites en sorte que, dans le modèle objet, les mouvements soient triés du plus récent au plus ancien. Testez le bon fonctionnement de cette méthode.

Question 8. Nous souhaitons à présent pouvoir enregistrer un nouveau mouvement. Ajoutez une méthode **public boolean ajouteMouvement(Produit produit, int quantite) throws SQLException** qui :

- ajoute dans la base de données un nouveau mouvement associé au produit **produit**, avec une quantité **quantite**, à la date courante ;
- si l'ajout dans la base a bien été réalisé, alors le modèle de données est mis à jour : une nouvelle instance de **Mouvement** est créée, et ce nouveau mouvement (qui est le plus récent) est ajouté aux mouvements du produit **produit**.

Cette méthode renvoie un booléen qui vaut **true** si le mouvement a bien pu être ajouté en base (et donc également dans le modèle objet), et **false** sinon.

4 Affichage de la page d'accueil (4 points)

On souhaite à présent ajouter à notre application une première fenêtre qui sera l'accueil de notre application. Sur cette fenêtre, on placera :

- une **JList** qui contient les noms des produits ;
- un **JLabel** avec le texte "*Recherche* : " ;
- un **JTextField** dans lequel le texte rentré permettra de filtrer les produits affichés dans la liste : leur nom devra commencer par le texte entré ;
- un **JButton** qui permettra par la suite d'accéder aux prévisions de rupture de stock ;
- un **JSpinner** qui permettra par la suite de contrôler la quantité d'un nouveau mouvement ;
- un **JButton** qui permettra par la suite d'enregistrer un mouvement.

Un exemple de cette fenêtre est donné dans la Figure 3.

Question 9. Ajoutez à votre projet un paquetage **vuecontrôle**. Ajoutez dans ce paquetage une fenêtre **Accueil** qui sera similaire à celle présentée dans la Figure 3. Cette fenêtre est l'accueil de votre application et devra impérativement pouvoir être lancée si l'on exécute la classe principale de votre projet (clic sur la flèche verte sur NetBeans). À l'ouverture de cette fenêtre, l'ensemble des produits sera affiché dans la liste, toujours triés par ordre alphabétique. Ajoutez la logique de contrôle de telle sorte que lorsque le texte dans la zone de texte est modifié, la liste des produits est mise à jour (seulement les produits ayant le nom commençant par ce texte sont affichés). Vérifiez le bon fonctionnement de cette fenêtre.

Étant donné un produit, avec l'ensemble des mouvements associés à ce produit, on souhaite connaître la quantité actuellement en stock pour ce produit. On suppose qu'à l'origine la quantité en stock était nulle. Il suffit ensuite de faire la somme des quantités associées aux mouvements du produit pour connaître la

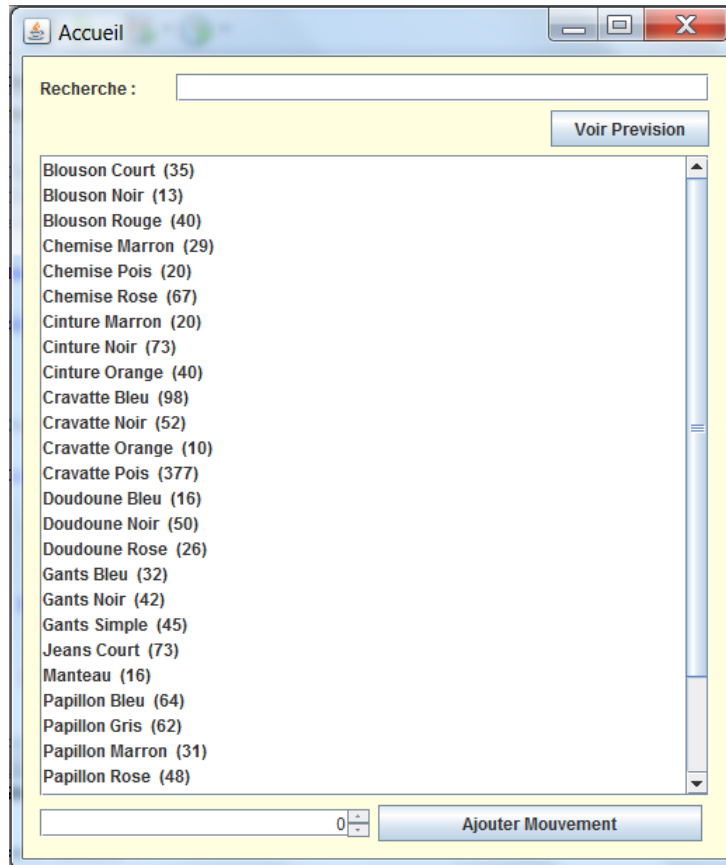


Figure 3: Exemple d'affichage de la page d'accueil.

quantité actuellement en stock. Par exemple, si un produit p a 3 mouvements avec les quantités $+10$; -2 et -3 ; alors la quantité actuellement en stock est $10 - 2 - 3 = 5$.

Question 10. Modifiez votre code pour pouvoir afficher, dans la fenêtre d'accueil, à côté de chaque produit, la quantité actuellement en stock pour le produit. Testez le bon fonctionnement du calcul des quantités actuellement en stock.

Question 11. Modifiez votre code pour que les valeurs dans le *spinner* soient associées au produit sélectionné dans la **JList** de votre fenêtre et puissent varier entre :

- moins la quantité disponible du produit sélectionné ;
- la valeur maximale d'un entier.

(**Suggestion** : comme les **JTable**, les **JSpinner** sont initialisés à partir d'un modèle. Vous disposez de l'interface **SpinnerModel** avec l'implémentation **SpinnerNumberModel**. Lorsqu'un **SpinnerNumberModel** a été créé avec le bon constructeur, vous pouvez l'associer à votre *spinner*).

5 Ajout d'un mouvement (4 points)

Nous souhaitons ajouter un mouvement dans notre base de données. On suppose que tout nouveau mouvement est associé à la date à laquelle le mouvement est enregistré dans la base de données.

Question 12. Modifiez votre code pour que quand vous cliquez sur le bouton "Ajouter Mouvement" :

- si aucun produit n'est sélectionné, rien ne doit se passer ;
- si la quantité dans le *spinner* est nulle, un message (à travers une boîte de dialogue) est affiché et aucun enregistrement n'est effectué dans la base de données ;
- si le mouvement est associé à une quantité négative suffisante à dépasser la quantité de stock minimale, alors une alerte est affichée pour rappeler à l'utilisateur de passer une commande pour le produit considéré ; et le mouvement est quand-même enregistré dans la base de données ;
- si le mouvement est associé à une quantité positive suffisante à dépasser la quantité de stock maximale, alors une confirmation est demandée à l'utilisateur (avec une boîte de dialogue de confirmation) avant de procéder à l'enregistrement du mouvement : si l'utilisateur ne confirme pas, l'enregistrement est annulé ;
- si le mouvement est associé à une quantité telle que le stock sera entre le stock minimale et le stock maximal, alors le mouvement est juste enregistré dans la base de données ;
- si le mouvement est enregistré dans la base de données, alors la quantité en stock pour le produit considéré doit être mise à jour ; et l'affichage de la liste des produits doit donc être mis à jour en conséquence ;
- dans tous les cas, la valeur dans le *spinner* est remise à zéro.

6 Pr vision rupture de stock (6 points)

Nous souhaitons   pr sent pouvoir afficher les produits tri s selon la date de pr vision   laquelle le stock sera en rupture (la quantit  en stock sera nulle).

6.1 Calcul de la consommation journali re moyenne et date de rupture de stock d'un produit (3 points)

Nous souhaitons d'abord calculer la consommation journali re d'un produit. Pour faire cela, il faut calculer la quantit  totale du produit consomm  par les clients, puis diviser cette valeur par le nombre total de jours depuis sa premi re apparition dans le magasin (premier mouvement avec une quantit  positive). Par exemple supposons d'avoir, pour un produit p , les mouvements suivants :

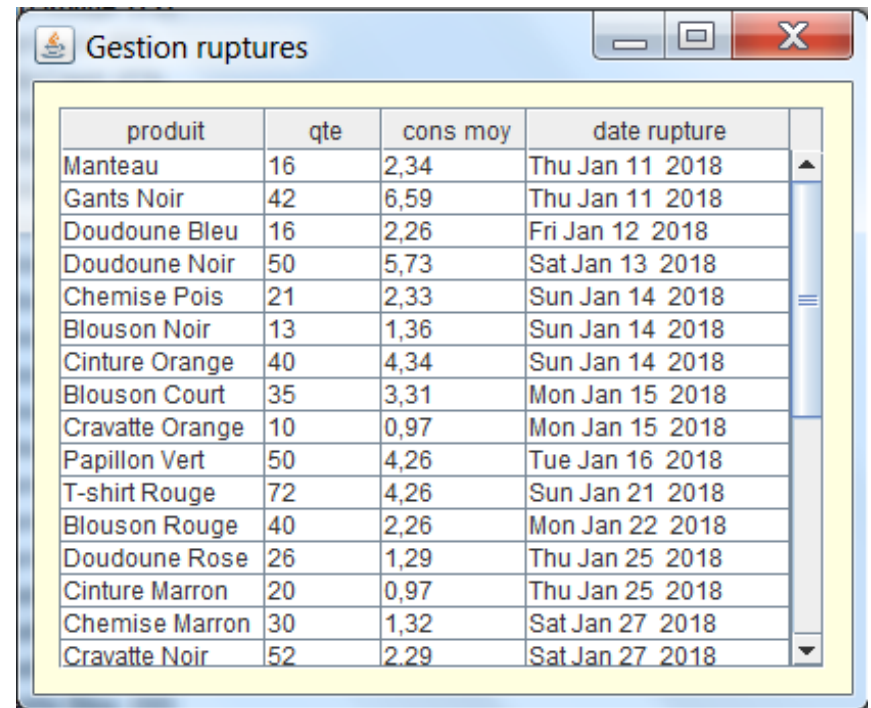
- +10   la date t_0 ;
- -4   la date t_1 ;
- +15   la date t_2 ;
- -8   la date t_3 ;
- -6   la date t_4 .

On suppose les dates ordonn es telles que $t_0 < t_1 < t_2 < t_3 < t_4 < t_{now}$ o  t_{now} est la date actuelle. Alors la quantit  totale consomm e $q_{total} = 4 + 8 + 6 = 18$. Le nombre total de jours depuis la premi re apparition est $nbJours = t_{now} - t_0$ (  exprimer en jours). Et donc la consommation journali re est

$$conso = \frac{q_{total}}{nbJours}.$$

Connaissant la consommation journali re moyenne d'un produit et la quantit  actuellement disponible en stock, vous pouvez ensuite calculer la date pr visionnelle de rupture de stock (en supposant, bien s r, qu'aucun mouvement avec une quantit  positive ne sera effectu  d'ici l ). Si on reprend l'exemple pr c dent, alors la quantit  actuellement en stock est $q = 10 - 4 + 15 - 8 - 6 = 7$. Si la consommation moyenne journali re est $conso = 2$, alors la rupture de stock interviendra dans $n = 7/2 = 3,5$ jours.

Question 13. Modifiez votre code pour pouvoir associer   chaque produit sa consommation journali re moyenne et sa date pr visionnelle de rupture de stock. Vous pouvez bien  videmment modifier toutes les classes que vous jugez n cessaire. Testez le bon fonctionnement de votre code en affichant   la console les deux nouvelles quantit s.



| produit | qte | cons moy | date rupture |
|-----------------|-----|----------|-----------------|
| Manteau | 16 | 2,34 | Thu Jan 11 2018 |
| Gants Noir | 42 | 6,59 | Thu Jan 11 2018 |
| Doudoune Bleu | 16 | 2,26 | Fri Jan 12 2018 |
| Doudoune Noir | 50 | 5,73 | Sat Jan 13 2018 |
| Chemise Pois | 21 | 2,33 | Sun Jan 14 2018 |
| Blouson Noir | 13 | 1,36 | Sun Jan 14 2018 |
| Cinture Orange | 40 | 4,34 | Sun Jan 14 2018 |
| Blouson Court | 35 | 3,31 | Mon Jan 15 2018 |
| Cravatte Orange | 10 | 0,97 | Mon Jan 15 2018 |
| Papillon Vert | 50 | 4,26 | Tue Jan 16 2018 |
| T-shirt Rouge | 72 | 4,26 | Sun Jan 21 2018 |
| Blouson Rouge | 40 | 2,26 | Mon Jan 22 2018 |
| Doudoune Rose | 26 | 1,29 | Thu Jan 25 2018 |
| Cinture Marron | 20 | 0,97 | Thu Jan 25 2018 |
| Chemise Marron | 30 | 1,32 | Sat Jan 27 2018 |
| Cravatte Noir | 52 | 2,29 | Sat Jan 27 2018 |

Figure 4: Exemple d'affichage de la fen tre Prevision.

6.2 Cr ation de la fen tre Prevision (3 points)

Nous souhaitons cr er une fen tre **Prevision**. Cette fen tre est ouverte lors du clic sur le bouton "Voir pr vision". Elle affiche les produits tri s selon leur date de rupture de stock dans un objet de type **JTable**. Pour chaque produit, on affiche son nom, la quantit  restante, la consommation journali re moyenne, et la date de pr vision de rupture de stock. Un exemple de fen tre **Prevision** est donn  en Figure 4 (vous devez obtenir des valeurs similaires pour les consommation journali res, mais pas n cessairement pour les dates de rupture).

Question 14. Dans le paquetage **vuecontr le**, ajoutez une fen tre et appelez-l  **Prevision**. Ajoutez-y un objet de type **JTable**. Puis, ajoutez tous les  l ments n cessaires (attributs, m thodes, ...) au bon fonctionnement de la fen tre. Pour le moment, on ne s'int resse pas   avoir un beau rendu graphique. Pour le mod le de la **JTable**, voici quelques mani res possibles de proc der :

- créer une classe qui hérite de **AbstractTableModel** ;
- créer une classe qui hérite de **DefaultTableModel** ;
- directement utiliser un objet de type **DefaultTableModel**.

Question 15. Pour terminer, il ne vous reste plus qu'à faire un peu de mise en forme pour avoir un rendu agréable. Vous pouvez vous référer à la Figure 4 par exemple.