**ITI 1520 – Devoir 5**

Disponible : samedi, 19 novembre  
Date de remise : **samedi, 3 décembre**, avant 21:00

### Notez bien

* Vous devez faire ce travail en **ÉQUIPE DE DEUX PERSONNES** et suivre toutes les consignes décrites dans le manuel de laboratoire quant à la soumission de devoirs par le Campus Virtuel. Des points seront déduits si vous ne suivez pas ces consignes.
* Assurez-vous de mettre les **noms et numéros des deux étudiants** dans chaque programme.
* Pour ce devoir, des fichiers de test avec les résultats attendus (**D5Qx.java** et **D5Qx.txt**, où **x** est le numéro de la question) et des tests JUnit vous sont offerts. Assurez-vous d'obtenir les mêmes résultats! Vous pouvez ajouter vos propre tests (autre méthode **main()** ou tests JUnit), mais vous n'avez pas à les remettre.
* Utilisez **EXACTEMENT** les signatures de méthodes données (dans les descriptions ou les tests), car elles seront corrigées avec JUnit.
* Veuillez remettre les 4 fichiers (4 classes, commentées) qui sont demandés.

### Barème

* Consignes et standards: 10 points
* Question 1: 20 points
* Question 2: 20 points
* Question 3: 25 points
* Question 4: 25 points

## 

## Question 1 (20 Points)

On vous demande de développer une classe **Client** pour emmagasiner l'information sur un client d’une entreprise en ligne qui se spécialise en Livres, DVD et Jeux Video.

### Attributs

Vos objets **Client** devront contenir l'information suivante pour chaque client:

* Le nom du client.
* L'adresse courriel du client.
* Un mot de passe utilisé par le client.
* Un montant d'argent crédité lorsqu'un nouveau client utilise cette entreprise. Ce montant (en dollars entier, pas de fraction) pourra être utilisé pour un achat.

### Constructeur

Quand un objet **Client** est créé, des valeurs doivent être fournies pour son nom, son courriel, et son mot de passe initial. Un montant de 10$ est automatiquement crédité pour ce nouveau client à ce moment.

**public** Client (String nomClient, String courrielClient, String mdpClient)

### Accesseurs et modificateurs

Vous devez fournir des méthodes d'accès publiques pour tous les champs sauf, bien sûr, le mot de passe. Vous devez aussi fournir une méthode de modification publique pour l'adresse courriel (qui peut changer). Il est interdit de changer le nom d'un client.

**public** String getNom ()  
 **public** String getCourriel ()  
 **public** **int** getMontantCrédité ()  
 **public** **void** setCourriel (String nouvCourriel)

### Autres méthodes

Trois autres méthodes sont nécessaires. La première compare un mot de passe fourni par le client et le compare au mot de passe emmagasiné dans l'objet **Client**. Cette méthode retourne true si le client a fourni le bon mot de passe, et false sinon.

**public** **boolean** vérifieMotDePasse (String mdp)

La deuxième méthode permet de changer le mot de passe du client. Un mot de passe ne peut être changé que si le mot de passe actuel est connu. Ainsi, cette méthode contient deux paramètres: l'ancien mot de passe et le nouveau. Si l'ancien mot de passe correspond au mot de passe connu, alors le mot de passe de ce client est remplacé par le nouveau mot de passe suggéré et la méthode retourne true. Sinon, la méthode retourne false et le mot de passe demeure inchangé. Il faut faire attention à ne pas révéler le mot de passe emmagasiné dans l'objet!

**public** **boolean** changeMotDePasse (String ancienMDP, String nouveauMDP)

Enfin, comme il sera possible au client d'utiliser le montant d'argent crédité (en partie ou en totalité) pour acheter un item, la méthode suivante devra soustraire le montant passé en paramètre du montant crédité. Elle retourne  true si le montant déduit ne dépasse pas le montant crédité, sinon elle retourne false et le montant crédité demeure inchangé.

**public boolean** réduitMontantCrédité (**int** montantDéduit)

### Test

Testez votre classe **Client** à l'aide de la classe de test JUnit [ClientTest.java](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/ClientTest.java)).  Vous pouvez tester d'autres situations à l'aide de vos propres classes de test ou de tests JUnit, mais assurez-vous que les tests dans ClientTest.java (non modifiée) fonctionnent correctement.

## Question 2 (20 Points)

On vous demande de développer une classe **Item** pour emmagasiner l'information sur un item achetable chez dans cette entreprise.   
  
  
Il y a des valeurs qui sont applicables à tous les objets **Item**:

* Un code numérique indiquant si l'item est un livre (constante LIVRE valant 0), un DVD (constante DVD valant 1) ou un jeu vidéo (constante JEU valant 2).

Les attributs pour chaque objet **Item** individuel sont les suivantes:

* Le type de l'item (LIVRE, DVD, ou JEU).
* Le titre de l'item.
* Le prix de vente de l'item (en dollars seulement, pas de cents).
* Le nombre de copies présentement disponibles.

Lorsqu'un objet **Item** est créé, une valeur doit être fournie pour chaque variable d'instance.  
  
Vous devez inclure des accesseurs pour chaque variable d'instance, ainsi qu'un modificateur permettant de mettre à jour le champ prix

### Méthodes supplémentaires

* La méthode décrémenteCopies() doit décrémenter le nombre de copies disponibles d'une unité. Cette méthode retourne true s'il est possible de décrémenter, sinon elle retourne false et affiche un message d'erreur approprié (Erreur: Aucune copie de '*titre de l'item*' disponible.).
* La méthode incrémenteCopies() doit incrémenter le nombre de copies disponibles d'une unité.
* Enfin, la dernière méthode (typeToString(...)) retourne une chaîne de caractères décrivant un type d'item. Les trois chaînes possiblement retournées sont: "Livre", "DVD" ou "Jeu vidéo".

### Test

Testez votre classe **Item** à l'aide de la classe [D5Q2.java](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q2.java) ci-jointe (et produisant la sortie [D5Q2.txt](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q2.txt)).  Vous pouvez tester d'autres situations à l'aide de vos propres classes de test ou de tests JUnit, mais assurez-vous que la classe D5Q2.java (non modifiée) fonctionne correctement.

## Question 3 (20 Points)

Dans cette question, vous devrez créer une classe **Achat** permettant d'emmagasiner l'information sur des instances spécifiques d'achats d'items par des clients. Quand un client achète un item, cette classe représentera l'association entre un objet **Client** et un ou plusieurs objets **Item** achetés lors de cette transaction.

### Attributs et méthodes de base

Les objets individuels **Achat** doivent inclure l'information suivante (et privée) pour chaque transaction:

* Un numéro de facture qui identifie de façon unique la transaction d'achat.
* Une référence vers le **Client** acheteur.
* Un tableau de références, où les éléments font référence à chacun des objets **Item** achetés lors de cette transaction
  + *Hypothèse*: pas plus d'un exemplaire de chaque item par achat
* Le numéro du jour où l'achat a été effectué (les dates seront simplement représentées par des entiers dans ce devoir, et non par des dates complètes).
* Le coût total de l'achat (nombre entier), incluant les frais de transport.

Lorsqu'un nouvel objet **Achat** est créé, un numéro de facture lui est automatiquement assigné. Le numéro de la première facture d'achat est 1000, et les numéros de factures sont incrémentés à tous les achats subséquents.  
  
Pour créer un objet **Achat**, il faut fournir la référence vers le **Client**, le tableau de références vers les objets **Item**, de même que le jour de la transaction. Le coût total de l'achat devrait être déterminé automatiquement en calculant la somme des prix des items, et en ajoutant ensuite les frais de transport (2$ de manutention, plus 1$ par item livré). Si l'acheteur a un montant crédité, il sera utilisé pour réduire (en partie ou en totalité) le coût de l'achat. Si aucun des items n'est disponible, il n'y aura évidemment aucun frais de transport.  
  
Le nombre de copies de chaque item devrait évidemment être décrémenté lors d'un achat. Notez qu'il se pourrait qu'il n'y ait plus de copie d'un item demandé. La transaction d'achat aura alors quand même lieu, mais le ou les items manquants ne seront pas comptabilisés dans le coût total.  
  
Votre classe devrait inclure des méthodes accesseur pour le numéro de la facture, le coût total d'achat, et la date de l'achat.

### Méthodes supplémentaires

* getNomClient() retourne le *nom* du **Client** acheteur.
* itemIci(...): détermine si l'**Item** passé en paramètre fait partie de la liste des items achetés dans cette transaction. Retourne true si c'est le cas, et false sinon. Assurez-vous d'avoir une recherche efficace.
* toString(): retourne un objet String contenant l'information sur cet **Achat**, un peu à la façon de ce que vous verriez sur une facture:
  + Le numéro de la facture et la date d'achat
  + Le nom du client et son courriel
  + Les titres et types des items achetés (incluant ceux qui n'ont pas été livrés)
  + Le montant total payé
    - L'avantage de créer une méthode toString() est qu'elle sera utilisable directement lors de l'affichage. Par exemple, si monAchat est un achat:
      * System.out.println(monAchat);
    - Notez que la constante suivante permet d'ajouter un changement de ligne à une chaîne de caractère String  
                final String newLine = System.getProperty("line.separator");
    - exemple: String résultat = "Bonjour!" + newLine;

Vous pouvez ajouter d’autres methods privées ou de variables, si nécessaire.

NOTE: Assurez-vous d'utiliser les signatures de méthodes et les messages affichés qui sont utilisés dans la classe de test <D5Q3.java> et ses résultats!

### Test

Testez votre classe **Achat** à l'aide de la classe [D5Q3.java](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q3.java) ci-jointe (et produisant la sortie [D5Q3.txt](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q3.txt)). Il faut auparavant que les classes **Client** et **Item** fonctionnent bien. Vous pouvez tester d'autres situations à l'aide de vos propres classes de test ou de tests JUnit, mais assurez-vous que la classe D5Q3.java (non modifiée) fonctionne correctement.

## Question 4 (25 Points)

Votre entreprise a besoin de maintenir une liste d'**Achat**s, et de pouvoir y faire certaines requêtes d'information. C'est ce que vous devez faire dans la classe **Journal**, qui doit contenir l'information suivante:

* Une liste d'achats sous forme de tableau de références vers des objets **Achat**.
* Le nombre d'achats actuellement contenues dans le journal (entier)
* La taille maximale de la liste d'achats (entier)

Comme un seul journal sera utilisé globalement, nous ne créerons pas d'objets à partir de cette classe. Ainsi, tous ses attributs/méthodes devront être déclarés "static".  
  
Quatre méthodes devront y être ajoutées:

* initialise(...): prend en entrée un entier qui deviendra la taille maximale de la liste d'achats. Cette méthode devra aussi créer le tableau d'achats (de taille maximale) et initialiser le nombre d'achats à 0.
* ajoute(...): ajoute un nouvel **Achat** à la liste d’achats. Si le tableau d'achats est déjà plein, alors le tableau reste inchangé et un message d'erreur approprié est affiché. En sortie, cette méthode retourne une valeur indiquant si l'ajout a été effectué avec succès. La signature de cette méthode est:   
        public static boolean ajoute (**Achat unAchat**)
* achatsPourClient(...): retourne un tableau (de bonne taille) de coûts totaux pour les achats faits par un  **Client** passé en paramètre. La signature de cette méthode est:   
        public static int[]  achatsPourClient (Client client)
* afficheAchatsItem(...): affiche les noms de tous les clients qui ont tenté d'acheter l'item passé en paramètre, de même que les dates d'achat correspondantes.
* calculeTotalAchats(...): retourne la somme de tous les achats enregistrés dans le journal.

Plusieurs autres types de requêtes pourraient être ainsi supportées par des méthodes Java additionnelles. À nouveau, assurez-vous d'utiliser les signatures de méthodes et les messages affichés qui sont utilisés dans la classe de test <D5Q4.java> et ses résultats.

### Test

Testez votre classe **Journal** à l'aide de la classe [D5Q4.java](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q4.java) ci-jointe (et produisant la sortie [D5Q4.txt](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/D5Q4.txt)). Il faut auparavant que la classe **Achat** fonctionne bien. Vous pouvez tester d'autres situations à l'aide de vos propres classes de test ou de tests JUnit, mais assurez-vous que la classe D5Q4.java (non modifiée) fonctionne correctement.

### Aperçu supplémentaire

Nous vous fournissons des diagrammes de classe UML afin de vous montrer les attributs et méthodes (publiques/privées, de classe ou d'instance) pour chaque classe. De plus, nous vous donnons un   
diagramme de classe UML qui vous offre un aperçu global des classes et de leurs associations et dépendances (sans les détails des attributs et des opérations). Ce genre de modèle facilite la conception de solutions orientées-objet pour systèmes complexes. Tous ces diagrammes UML se trouvent dans le fichier [devoir-5-UML.ppt](http://www.site.uottawa.ca/~nat/Courses/ITI1520_2011/devoir-5-UML.ppt) ci joint.