1. Objectifs du TP

- Concepts de base de la programmation orientée objet.
- Environnement Netbeans en mode console.

2. Description générale :

L'objectif du TP est d'écrire un programme permettant de gérer les stocks d'une boutique spécialisée dans les articles culturels d'occasion (livres, manuels, magazines). La boutique récupère des dons de particuliers ou des invendus de la grande distribution, pour les redistribuer à prix modéré.

La boutique propose trois types d'articles : des livres, des manuels scolaires et des magazines.
Chaque article est caractérisé au moins par une description, un prix initial de vente et un nombre d'exemplaires.
Chaque livre possède un numéro ISBN (code à 13 chiffres qui identifie de manière unique les livres, y compris les manuels) et un nombre de pages.
De plus, pour les manuels scolaires sont définis également : une matière (mathématiques, français…) et un niveau scolaire (collège, lycée, licence, etc.).
Tous les livres et manuels bénéficient d'une réduction de leur prix de 50% au mois d'avril (le 23 avril étant la journée mondiale du livre et du droit d'auteur).
Chaque magazine possède un numéro ISSN (code à 8 chiffres qui identifie de manière unique les publications en série), une périodicité (hebdomadaire, mensuel ou trimestriel) et une date de publication.
Un magazine hebdomadaire (resp. mensuel et trimestriel) publié depuis plus de 2 semaines (resp. 2 mois pour un magazine mensuel et 2 trimestres pour un magazine trimestriel) voit son prix réduit de 50%. S'il a été publié depuis plus de 4 semaines (resp. 4 mois et 4 trimestres) il voit son prix réduit de 75%.

Le gérant de la boutique peut ajouter de nouveaux articles et peut augmenter ou réduire le nombre d'exemplaires d'un article déjà existant.

Pour des raisons de suivi, à chaque dépôt d'articles, un bon de dépôt est établi puis, selon le cas, soit l'article est ajouté, soit le nombre d'exemplaires disponibles pour cet article est mis à jour.

Le bon de dépôt doit être automatiquement numéroté et daté.
Le bon de dépôt contient le numéro de téléphone du déposant, la date du dépôt, le nombre d'articles et, pour chaque article, le numéro ISBN ou ISSN de l'article et le nombre d'exemplaires déposés.

Exemples d'articles :

- un livre :

Description	Prix initial de vente	Nombre d'exemplaires	ISBN	Nombre de pages
La maison vide	*25 €*	*10*	*978 2 7073 5674 1*	*752*

- un manuel scolaire :

Description	Prix initial de vente	Nombre d'exemplaires	ISBN	Nombre de pages	Matière	Niveau scolaire
Nouvelle grammaire du collège	*38 €*	*31*	*978 2 2104 4669 4*	*512*	*Français*	*Collège*

- un magazine :

Description	Prix initial de vente	Nombre d'exemplaires	ISSN	Périodicité	Date de publication
Geo	*6,50 €*	*20*	*0020 8245*	*mensuel*	*01/08/25*

3. Etapes de réalisation

1ère partie (TP3) :

Il est demandé de **respecter les noms de classes, attributs et méthodes** indiqués ci-après.
Cela étant, vous pouvez bien entendu, si nécessaire, ajouter des attributs ou méthodes supplémentaires dans votre programme.

Eviter de mélanger saisies, affichages et traitements dans les méthodes.

Tester les méthodes au fur et à mesure de leur écriture, en créant dans le programme principal des instances de vos classes.

1. Créer les différentes classes de base du programme : définir les attributs, les constructeurs et des méthodes *toString*.

   (a). Créer quatre classes d'articles :
   - *Article*. Tout article possède une description, un prix initial de vente et un nombre d'exemplaires en stock.
   - *Livre*. Chaque livre possède aussi un numéro ISBN et un nombre de pages.
   - *Manuel.* Pour chaque manuel on connaît en plus la matière enseignée et le niveau scolaire auquel il correspond.
   - *Magazine*. Pour chaque magazine on connaît aussi son numéro ISSN, sa périodicité (hebdomadaire, mensuel ou trimestriel) et sa date de publication.
     Pour la date de publication, utiliser la classe LocalDate.

   (b). Créer une classe *LigneDepot* : elle représente une ligne d'un bon de dépôt.
   Pour chaque ligne, on connait le numéro ISBN ou ISSN de l'article et le nombre d'exemplaires déposés.

2. Créer une classe *BonDepot* : elle représente un bon de dépôt. Pour chaque bon de dépôt, on connait le numéro de téléphone du déposant, la date d'émission du bon, le nombre d'articles déposés et l'ensemble de ces articles (chaque article est décrit dans une ligne de dépôt). De plus, chaque bon de dépôt doit être numéroté de manière unique.
   On limitera le nombre de lignes de dépôt d'un bon de dépôt et on les stockera dans un tableau.
   Pour la date d'émission du bon, utiliser la classe *LocalDate*.

   Ecrire un constructeur et une méthode pour afficher les instances de la classe.

3. Créer une classe *Etablissement.* Dans cette classe sont définis le nom de la boutique, l'ensemble des articles disponibles et l'ensemble des bons de dépôt émis.
   On limitera le nombre d'articles et le nombre de bons de dépôt, et on les stockera dans deux tableaux.

   Ecrire un constructeur et une méthode pour afficher les instances de la classe.

4. Ecrire les méthodes permettant de gérer les articles de l'établissement.

   Le tableau contenant les articles sera classé dans l'ordre lexicographique, selon les numéros ISSN et ISBN.

   (a) Dans la hiérarchie de classes d'articles, écrire les méthodes :
   - *getNumero* retournant le numéro ISBN ou ISSN d'un article,

- *placerApres* permettant de comparer deux articles. La méthode doit retourner *true* si l'article correspondant à l'objet appelant doit être placé après l'autre dans le tableau des articles, *false* sinon.
  Pour comparer les numéros ISBN ou ISSN, utiliser la méthode *compareTo* de la classe String.
- *ajouter* permettant d'augmenter le nombre d'exemplaires d'un article d'une quantité donnée,
- *retirer* permettant de diminuer le nombre d'exemplaires d'un article d'une quantité donnée,
- *calculerPrix* permettant de calculer le prix d'un article en tenant compte des diverses réductions (cf. page 2).

(b) Dans la classe *Etablissement*, écrire les méthodes *ajouter* permettant :
- d'ajouter un livre en indiquant sa description, son prix initial, son nombre d'exemplaires, son numéro ISBN et son nombre de pages,
- d'ajouter un manuel scolaire en indiquant sa description, son prix initial, son nombre d'exemplaires, son numéro ISBN, son nombre de pages, la matière et le niveau scolaire,
- d'ajouter un magazine en indiquant sa description, son prix initial, son nombre d'exemplaires, son numéro ISSN, sa périodicité et sa date de publication.

(c) Dans la classe *Etablissement*, écrire les méthodes permettant de gérer le nombre d'exemplaires des articles :
- *rechercher* permettant de rechercher, dans le tableau des articles, l'article correspondant à un numéro ISBN ou un numéro ISSN donné. La méthode doit retourner l'article si elle l'a trouvé ou *null* sinon,
- *ajouter* permettant d'augmenter le nombre d'exemplaires d'un article en donnant un numéro ISBN ou un numéro ISSN et la quantité d'exemplaires reçue,
- *retirer* permettant de diminuer le nombre d'exemplaires d'un article en donnant un numéro ISBN ou un numéro ISSN et la quantité d'exemplaires vendue,

5. Ecrire les méthodes permettant de gérer les bons de dépôt de l'établissement.

   Le tableau contenant les bons de dépôt sera classé par date d'émission du bon (le plus récent en dernier).

   (a) Dans la classe *BonDepot*, écrire la méthode *ajouterLigne* permettant d'ajouter une ligne de dépôt, en donnant le numéro ISBN ou ISSN de l'article et le nombre d'exemplaires déposés.

   (b) Dans la classe *Etablissement*, écrire une nouvelle méthode *ajouter* permettant d'enregistrer un bon de dépôt.
   Les bons de dépôt doivent être automatiquement datés et numérotés par le programme, de manière continue.

2ème partie (TP4) :

1. Dans la classe *Etablissement*, écrire les méthodes permettant d'afficher les articles du magasin :

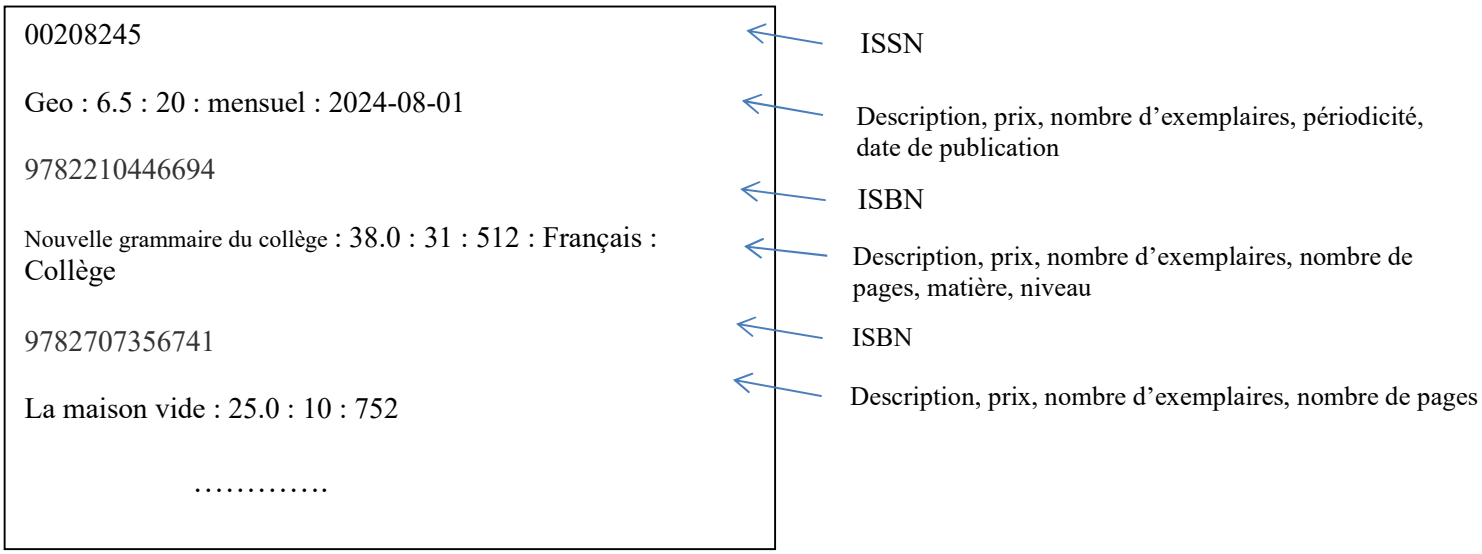
   - une méthode *lister* permettant d'afficher l'ensemble des articles avec leur prix à la date courante, rangés par ordre croissant du nombre d'exemplaires,
   - une méthode *lister* permettant d'afficher l'ensemble des bons de dépôt correspondant à un client donné (les clients sont identifiés par leur numéro de téléphone), rangés par date (la plus récente en dernier),
   - une méthode *lister* permettant d'afficher l'ensemble des bons de dépôt réalisés pour un numéro ISBN ou ISSN précis et une période donnée.

2. On veut que l'ensemble des articles et que l'ensemble des bons de dépôt soient sauvegardés dans des fichiers texte.

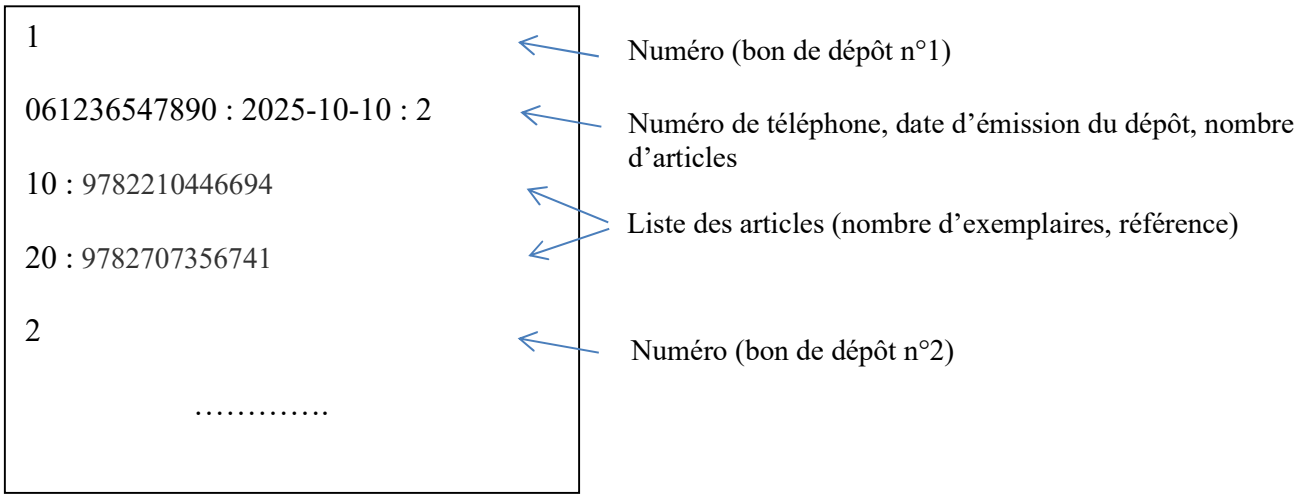
   Ces deux fichiers doivent respecter le format suivant (**sans aucun saut de ligne**) :

   - le fichier des articles :

.



   - le fichier des bons de dépôt :



(a). Dans la hiérarchie de classes des articles, dans la classe *LigneDepot* et dans la classe *BonDepot* écrire une méthode *versFichier* retournant sous forme d'une chaine de caractères les informations à écrire dans le fichier.

(b). Dans la classe *Etablissement*, écrire les méthodes pour gérer le fichier des clients :

- une méthode *versFichierArticles* permettant de sauvegarder dans un fichier texte les informations contenues dans le tableau des articles,
- une méthode *depuisFichierArticles* permettant de charger l'ensemble des articles à partir d'un fichier texte.

Pour écrire dans un fichier texte, utiliser la classe *FileWriter* (cf. annexe).
Pour lire dans un fichier texte, utiliser les classes *FileReader* et *BufferedReader* (cf. annexe).

(c). Dans la classe *Etablissement*, écrire les méthodes pour gérer le fichier des dépôts :

- une méthode *versFichierDepots* permettant de sauvegarder dans un fichier texte les informations contenues dans le tableau des bons de dépôt,
- une méthode *depuisFichierDepots* permettant de charger l'ensemble des bons de dépôt à partir d'un fichier texte.

3. Dans la classe principale du projet :

- créer une instance de la classe *Etablissement,*
- appeler les méthodes de chargement des fichiers texte,
- ajouter un menu comportant les différentes fonctionnalités du programme (cf. méthodes de la classe *Etablissement*),
- appeler les méthodes de sauvegarde dans des fichiers texte, avant de quitter le programme.

Annexe

Classe LocalDate :

- *static LocalDate of (int y, int m, int d)* : retourne une instance de la classe LocalDate correspondant à l'année, mois et jour du mois *y, m* et *d* donnés.
- *static LocalDate now ( )* : retourne une instance de la classe LocalDate correspondant à la date du jour.
- *static LocalDate parse (String ch) :* retourne une instance de la classe LocalDate correspondant à la conversion de la chaîne de caractères *ch* donnée. Par défaut, le format utilisé est : " yyyy-MM-dd ".
- int getMonthValue ( ) : retourne le mois, sous forme d'un entier entre 1 et 12, de la date correspondant à l'objet appelant.
- *boolean isAfter (LocalDate d),* resp. *isBefore* : vérifie si la date correspondant à l'objet appelant est postérieure, resp. antérieure, à la date *d* donnée.
- *LocalDate plusWeeks (long n)* : retourne la date correspondant à l'ajout du nombre de semaines *n* donné à la date correspondant à l'objet appelant.
- *LocalDate plusMonths (long n)* : retourne la date correspondant à l'ajout du nombre de mois *n* donné à la date correspondant à l'objet appelant.
- *long until (LocalDate d2, DAYS), resp. WEEKS* ou *MONTHS :* retourne la durée écoulée en jours, resp. semaines ou mois, entre la date correspondant à l'objet appelant et la date *d2* donnée (accepter d'importer « java.time.temporal.ChronoUnit.DAYS», resp. « java.time.temporal.ChronoUnit.WEEKS » ou « java.time.temporal.ChronoUnit.MONTHS »).

Classe FileWriter :

- *FileWriter* (String *nomF*) : crée une instance de la classe *FileWriter*, *nomF* est le nom du fichier dans lequel on veut écrire (accepter d'ajouter les clauses que Netbeans vous propose).
- *FileWriter* (String *nomF,* boolean *ajout*) : crée une instance de la classe *FileWriter*, *nomF* est le nom du fichier dans lequel on veut écrire et *ajout* indique s'il faut écrire à la fin du fichier (cas où *ajout* est égal à *true*) ou s'il faut écraser le fichier existant (cas où *ajout* est égal à *false*).
- *void write (*String *lig)* : écrit la ligne *lig* donnée dans le fichier texte.
- *void close ( )* : ferme le fichier.

Classe System :

- *static String lineSeparator ( )* : retourne la chaine de caractères correspondant au passage à la ligne (ce caractère dépend du système d'exploitation).

Classe FileReader :

- *FileReader* (String *nomF*) : crée une instance de la classe *FileReader*, *nomF* est le nom du fichier dans lequel on veut lire.
- *void close ( )* : ferme le fichier.

Classe BufferedReader :

- *BufferedReader (FileReader fich)* : crée une instance de la classe BufferedReader, *fich* est l'instance de la classe FileReader correspondant au fichier texte que l'on veut lire (accepter d'ajouter les clauses que Netbeans vous propose).
- *String readLine ( )* : lit et retourne une ligne du fichier.

Classe String :

- *String [] split (String exp)* : retourne un tableau contenant les sous-chaînes de la chaîne de caractères correspondant à l'objet appelant, en la découpant selon l'expression *exp* donnée.
- *int compareTo (String ch2)* : retourne 0 si les deux chaines sont égales, un entier strictement positif si la chaine correspondant à l'objet appelant est supérieure (selon l'ordre lexicographique) à la chaine *ch2*, un entier strictement négatif sinon.