TDTP
Vente d'équipements sportifs

1. Objectifs du TP :

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

2. Description générale :

L'objectif du TP est d'écrire un programme permettant de gérer les stocks d'un magasin spécialisé dans la vente d'équipements sportifs aux collectivités (établissements scolaires, associations sportives, etc.).

Le magasin propose plusieurs types d'équipements : des équipements pour le terrain (cage de buts, poteaux de rugby, paniers de basketball, …) et des équipements pour les joueurs (maillots, shorts, chaussures, …), parmi lesquels des équipements de protection (casques, gants, …).

Différentes informations sont disponibles sur ces équipements, en fonction de leur type.

Exemples d'équipements :

- Équipements pour le terrain :

référence	sport	désignation	prix (en euros)	nombre d'exemplaires	hauteur (en m)	largeur (en m)	poids (en kg)
TR005	rugby	paire de poteaux	2850	78	11	5,6	178
TR112	football	cage de buts	1600	64	2,44	7,32	67,6

- Équipements pour les joueurs :

référence	sport	désignation	prix (en euros)	nombre d'exemplaires	taille	coloris
JO092	rugby	short	17	1587	M	bleu
JO123	basket	maillot	29	1823	L	bleu

- Équipements pour la protection des joueurs :

référence	sport	désignation	prix (en euros)	nombre d'exemplaires	taille	coloris	niveau de pratique

PR034	rugby	protège-dents	4	1895	S	incolore	débutant
PR906	rugby	casque	39	1458	M	noir	expert

Le programme doit permettre de gérer les équipements vendus dans le magasin et les commandes effectuées par les collectivités.

De manière générale, le délai maximal de livraison d'un équipement est de 5 jours pour les équipements de terrain très lourds (poids supérieur à 100 kg) et de 3 jours pour tous les autres équipements. Cependant, si le nombre d'exemplaires en stock dans le magasin est insuffisant, un délai d'approvisionnement de 30 jours doit être ajouté.

3. Etapes de réalisation :

1ère partie :

Il est demandé de respecter les noms de classes, attributs et méthodes indiqués ci-dessous.
Cela étant, vous pouvez bien entendu, si nécessaire, ajouter des attributs ou des 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'équipements :

- *Equipement.* Quel que soit l'équipement, on connait sa référence (référence unique de 5 caractères interne au magasin), le sport, la désignation, le prix et le nombre d'exemplaires de l'équipement,
- *Terrain*. Quel que soit l'équipement pour le terrain, on connait le poids, la hauteur et la largeur de l'équipement.
- *Joueurs.* Quel que soit l'équipement pour les joueurs, on connait la taille (XS, S, M, etc.) et le coloris de l'équipement.
- *ProtectionJoueurs.* Quel que soit l'équipement dédié à la protection des joueurs, on connait le *niveau* de pratique auquel il est destiné (joueur débutant, régulier ou expert).

On doit pouvoir créer un équipement de terrain de deux façons différentes :
- soit en spécifiant une hauteur et une largeur,
- soit en ne spécifiant ni l'une ni l'autre, lorsqu'elles ne sont pas renseignées.

(b) Créer une classe *LigneCommande.* Pour chaque ligne d'un bon de commande, on connait la référence de l'équipement commandé, le nombre d'exemplaires et le prix unitaire.

(c) Créer une classe *Commande.* Pour chaque bon de commande, on connait le numéro de la commande (numéro unique à 8 chiffres), l'email de la collectivité qui a passé la commande, la date d'émission de la commande, la date de livraison prévue, le total de la commande et un tableau contenant les différentes lignes de la commande (chaque équipement commandé est décrit par une ligne de commande).

Pour les attributs correspondant à des dates, utiliser la classe *LocalDate* (cf. annexe).

2. Créer une classe *Magasin.* Dans cette classe seront définis le nom du magasin, le nombre et l'ensemble des équipements vendus dans le magasin, le nombre et l'ensemble des commandes passées au magasin.
Pour simplifier, on limitera le nombre d'équipements et le nombre de commandes, et on les stockera les équipements et les commandes dans des tableaux.

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

3. Ecrire les méthodes permettant de gérer les équipements en vente dans le magasin.

Le tableau contenant les équipements doit être classé par ordre lexicographique de la référence.

(a) Dans la classe *Equipement*, écrire la méthode :

- *placeApres* permettant de comparer deux équipements : elle doit retourner *true* si l'équipement correspondant à l'objet appelant doit être placé après l'autre équipement dans le tableau des équipements du magasin, *false* sinon.

(b) Dans la classe *Magasin*, écrire des méthodes *ajout* permettant :

- d'ajouter un équipement pour le terrain en donnant ~~sa référence,~~ **le sport, la désignation, le prix, le nombre d'exemplaires et le poids de l'équipement**,
- d'ajouter un équipement pour le terrain en donnant, en plus, sa hauteur et sa largeur,
- d'ajouter un équipement pour les joueurs en donnant ~~sa référence,~~ **le sport, la désignation, le prix, le nombre d'exemplaires, la taille et le coloris de l'équipement**,
- d'ajouter un équipement pour la protection des joueurs en donnant, en plus, le niveau de pratique auquel est destiné l'équipement.

Les références des équipements doivent être générées automatiquement par le programme. Elles sont composées de 2 lettres correspondant au type d'équipement (« JO » pour joueurs, « PR » pour protection des joueurs, « TR » pour terrain) suivies de 3 chiffres correspondant à un compteur spécifique à chaque type d'équipement.

(c) Dans la classe *Magasin*, écrire les méthodes suivantes :

- *recherche* qui, étant donnée une référence, retourne l'équipement correspondant,
- *affichage* qui, étant donnés un type (terrain, joueurs ou protection des joueurs) et un sport, permet d'afficher tous les équipements correspondants,

- *choixEquip* qui permet à une collectivité de choisir les équipements qu'elle souhaite acheter. Pour chaque équipement, le client spécifie le type d'équipement et le sport recherchés, le programme lui affiche les équipements potentiels, le client choisit une référence et le nombre d'exemplaires puis le programme remplit la ligne de commande correspondante.
  A la fin, la méthode retourne un tableau contenant toutes les lignes de commande.

2ème partie :

1. Ecrire les méthodes permettant gérer les commandes passées par les collectivités au magasin.

Le tableau contenant les commandes doit être classé par email puis par numéro de commande.

(a) Dans la hiérarchie d'équipements, ajouter les méthodes suivantes :

- *majDispo* qui, étant donnée une quantité, permet de l'ajouter ou de la retirer au nombre d'exemplaires de l'équipement,
- *verifDispo* qui, étant donnée une quantité, retourne *true* si le nombre d'exemplaires en stock dans le magasin est supérieur à cette quantité, *false* sinon,
- *calculDelai* qui, étant donnée une quantité, calcule et retourne le délai maximal de livraison (cf. page 2).

(b) Dans la classe *Commande*, ajouter la méthode suivante :

- *placeApres* permettant de comparer deux commandes : elle doit retourner *true* si la commande correspondant à l'objet appelant doit être placée après l'autre commande dans le tableau des commandes du magasin, *false* sinon.

(c) Dans la classe *Magasin*, écrire une nouvelle méthode *ajout* permettant d'enregistrer une commande.
Lorsqu'une collectivité souhaite passer une commande, le programme lui demande son email puis lui fait choisir les équipements qu'elle veut acheter. Il calcule ensuite le total et la date de livraison de la commande, puis met à jour le stock des équipements commandés.
Pour simplifier, on ne tiendra pas compte des délais supplémentaires imputables aux jours non ouvrables et aux jours fériés.

Les commandes doivent être numérotées automatiquement par le programme de manière continue : les 4 premiers chiffres du numéro correspondent à l'année en cours, les 4 derniers chiffres correspondent à un compteur incrémenté à chaque commande et remis à 0 au début de chaque année.

(d) Dans la classe *Magasin*, ajouter de nouvelles méthodes *affichage* permettant :

- d'afficher toutes les commandes effectuées par une collectivité donnée,
- d'afficher toutes les commandes devant être livrées après une date donnée.

2. On veut que l'ensemble des équipements et des commandes soient sauvegardés dans des fichiers texte.

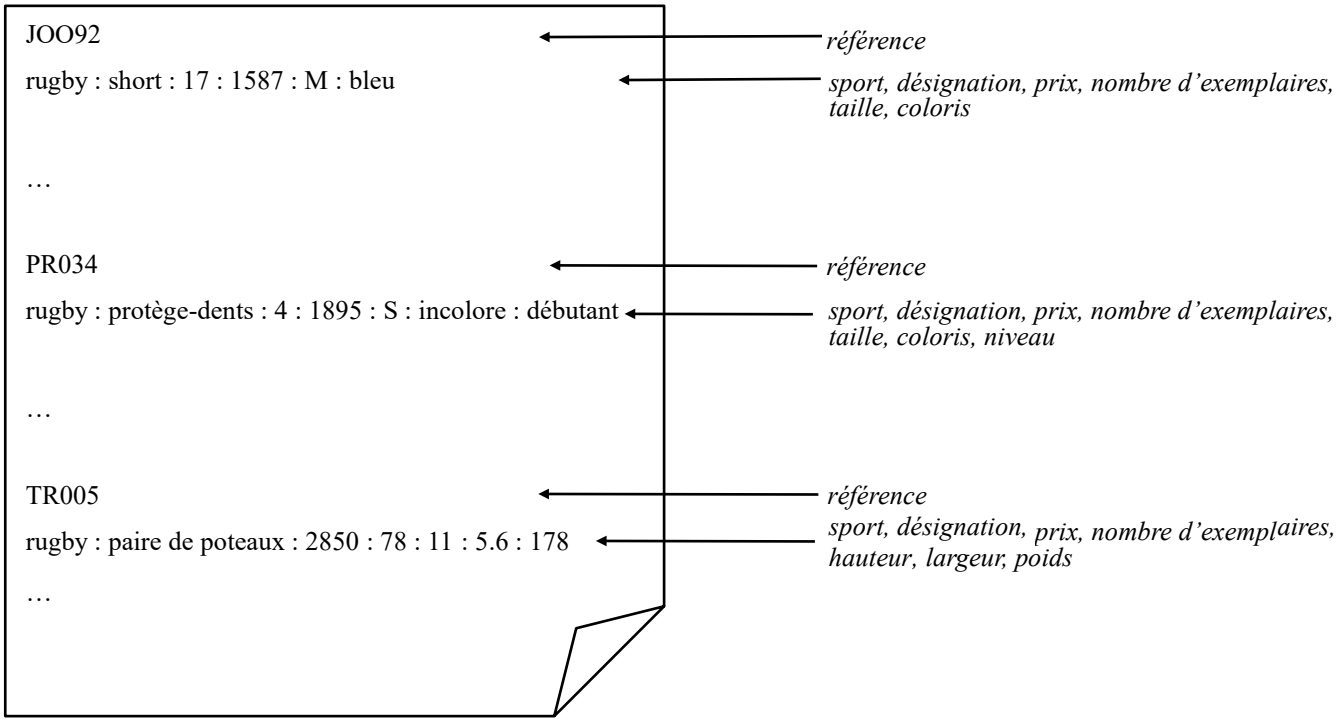
(a) Dans la hiérarchie d'équipements, dans la classe *LigneCommande* et dans la classe *Commande*, écrire une méthode *versFichier* retournant sous forme d'une chaine de caractères les informations à écrire dans le fichier.

(b) Dans la classe *Magasin*, écrire les méthodes *versFichierEquipements* et *versFichierCommandes* permettant de sauvegarder dans des fichiers texte les informations contenues dans le tableau des équipements et dans le tableau des commandes.

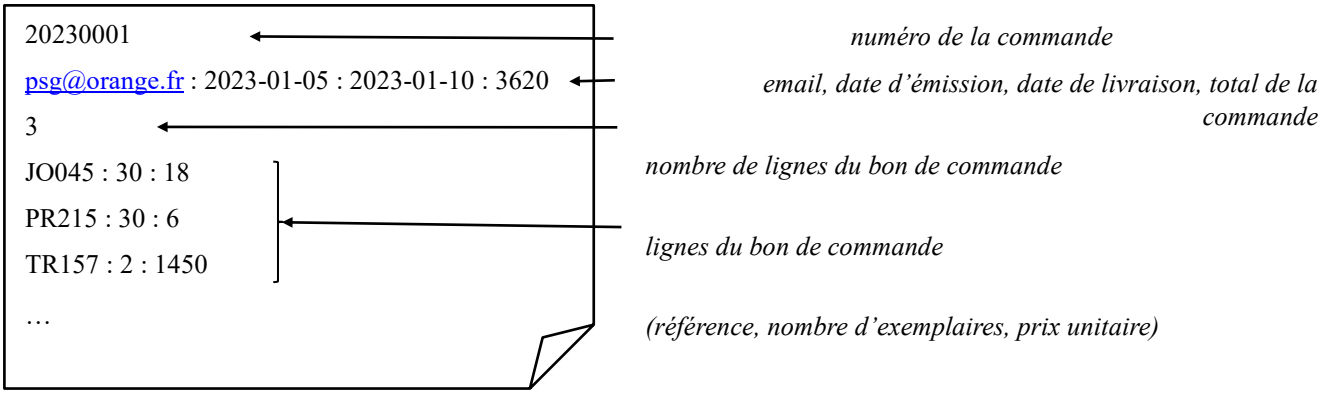
Pour écrire dans un fichier texte, utiliser la classe *FileWriter* (cf. annexe).

Les deux fichiers doivent respecter le format ci-dessous (sans aucune ligne vide) :

- le fichier des équipements :



- le fichier des commandes :



3. Dans la classe *Magasin*, écrire les méthodes *depuisFichierEquipements* et *depuisFichierCommandes* permettant de charger l'ensemble des équipements et l'ensemble des commandes à partir d'un fichier texte.

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

4. Dans la classe principale du projet :

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

Annexe

Classe LocalDate :

- *static LocalDate of (int annee, int mois, int jour)* : retourne une instance de la classe LocalDate correspondant au *jour*, *mois* et *annee* donnés.
- *static LocalDate now ( )* : retourne une instance de la classe LocalDate correspondant à la date du jour.
- *LocalDate plusDays (long nbre)* : retourne une copie de la date correspondant à l'objet appelant à laquelle a été ajoutée le nombre de jours *nbre* donné.
- *boolean isAfter (LocalDate d)* : vérifie si la date correspondant à l'objet appelant est postérieure à la date *d* donnée.
- *long until (LocalDate d, YEARS)* : retourne le nombre d'années écoulé entre la date correspondant à l'objet appelant et la date *d* donnée (accepter d'importer « java.time.temporal.ChronoUnit.YEARS »).

Classe FileWriter :

- *FileWriter* (String *nomFichier*) : crée une instance de la classe *FileWriter*, *nomFichier* est le nom du fichier dans lequel on veut écrire.
- *void write (*String *ligne)* : écrit la ligne 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 *nomFichier*) : crée une instance de la classe *FileReader*, *nomFichier* 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.
- *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.