Travail pratique #3

Ce travail doit être fait individuellement

Aucun retard permis

Notions mises en pratique : Respect d'un ensemble de spécifications, les classes et les objets, et les tableaux.

1. Spécifications

Ce travail consiste à implémenter un programme permettant la gestion d'objets perdus, qui pourrait être utilisé, par exemple, au comptoir des objets perdus dans un centre sportif, à l'UQAM, etc. Ce programme permet la consignation d'objets perdus dans le système, et la recherche de ces objets perdus consignés (par mots clés, par dates, et par catégorie) lorsqu'on vient les réclamer au comptoir. La partie 1 de ce TP consiste donc à implémenter la classe `ObjetPerdu`, qui modélise un objet perdu, et qui sera utilisée dans la partie 2 qui consiste à implémenter le programme de gestion des objets perdus, dans une classe nommée `GestionObjetsPerdus`. Vous avez donc deux classes à remettre pour ce TP.

Les **fichiers fournis** pour faire le TP (les classes Java doivent se trouver dans votre projet) :

Fichier	Classe
`Date.java`	Classe `Date` (**À NE PAS MODIFIER**) Cette classe doit être utilisée dans la classe `ObjetPerdu`. C'est le **type** de l'attribut `date` d'un objet perdu. Doit se trouver dans le paquetage par défaut.
`TP3Utils.java`	Classe `TP3Utils` (**À NE PAS MODIFIER**) Cette classe est une classe utilitaire contenant des méthodes utiles pour implémenter le programme dans la classe `GestionObjetsPerdus`. Doit se trouver dans le paquetage par défaut.
`TestsPartielsObjetPerdu.java`	Cette classe contient quelques tests pour tester la classe `ObjetPerdu`. Cette classe est une classe de tests partiels. Vous devrez cependant faire vos propres tests, plus complets, pour tester adéquatement votre classe `ObjetPerdu`.
`toString.txt`	Contient la méthode `toString` à copier-coller dans votre classe `ObjetPerdu`. Cette méthode `toString` fournie **ne doit pas être modifiée**.
`objetsPerdus.txt`	Contient la sauvegarde de quelques objets perdus consignés. Ce fichier sert à vous faciliter la tâche pour tester les options 2 à 6. (Voir section 1.2.1).

ATTENTION : Prenez soin de bien lire les classes `Date` et `TP3Utils` AVANT de commencer le TP pour bien connaître les méthodes qui sont à votre disposition, et les utiliser chaque fois que c'est possible dans votre programme.

1.1 PARTIE 1 : IMPLÉMENTATION DE LA CLASSE `ObjetPerdu`

Vous devez implémenter la classe `ObjetPerdu` contenant les membres suivants.

ATTENTION, VOUS DEVEZ RESPECTER le nom (et le type) des attributs d'instance ou de classe (constants ou non) mentionnés ci-dessous, sinon les tests de votre classe risquent de ne pas compiler, et vous perdrez des points pour non-respect des spécifications.

Constantes de classe publiques

Nom de la constante	Type	Description
CAT_BIJOU	`int`	Cette constante est initialisée à la valeur **0**. Elle représente la catégorie « bijou » d'un objet perdu.
CAT_VETEMENT	`int`	Cette constante est initialisée à la valeur **1.** Elle représente la catégorie « vêtement » d'un objet perdu.

CAT_ARGENT_PORTEFEUILLE	`int`	Cette constante est initialisée à la valeur **2**. Elle représente la catégorie « argent / portefeuille » d'un objet perdu.
CAT_CLE	`int`	Cette constante est initialisée à la valeur **3**. Elle représente la catégorie « clé(s) » d'un objet perdu.
CAT_AUTRE	`int`	Cette constante est initialisée à la valeur **4**. Elle représente la catégorie « autre » d'un objet perdu. Lorsqu'un objet perdu n'entre dans aucune des catégories précédentes, on le place dans cette catégorie.
CATEGORIES	`String[]`	Cette constante est initialisée à la valeur `{"bijou", "vetement", "argent / portefeuille", "cle(s)", "autre"}`. C'est un tableau qui permet d'obtenir le nom de toutes les catégories. Si un objet est de catégorie **CAT_CLE**, par exemple, on peut obtenir le nom de cette catégorie à la position **CAT_CLE** dans ce tableau.

Attribut de classe

Nom de l'attribut	Type	Description
sequenceId	`int`	Cet attribut contient le numéro d'identification unique (`id`) qui sera assigné au prochain objet perdu créé. Vous devez vous servir de cet attribut pour assigner automatiquement le numéro d'identification unique de chaque objet perdu créé. Cet attribut est incrémenté de 1 chaque fois qu'un objet perdu est construit.

Attributs d'instance

Nom de l'attribut	Type	Description
id	`int`	Le numéro unique d'identification de cet objet perdu. Ce numéro doit être assigné automatiquement (voir constructeur).
motsCles	`String[]`	Un tableau non `null` contenant tous les mots clés associés à cet objet perdu. À TOUT MOMENT, ce tableau doit toujours être de longueur minimale. C'est-à-dire que s'il y a, par exemple, 0 / 2 / 5 mots clés associé(s) à cet objet perdu, ce tableau doit être de longueur 0 / 2 / 5. De plus, ce tableau ne doit contenir aucune chaine `null` ou vide. Les mots clés servent à décrire l'objet perdu pour pouvoir plus tard effectuer des recherches par mots clés.
categorie	`int`	La catégorie associée à cet objet perdu. Antécédent : doit être une valeur entre 0 et CATEGORIES.length - 1 inclusivement.
date	`Date`	La date de consignation de cet objet perdu. Elle peut être `null`.
localisation	`String`	L'endroit où a été rangé cet objet perdu lors de sa consignation. Permettra de le retrouver rapidement lorsqu'il sera réclamé. Cet attribut peut être `null`.

ATTENTION, VOUS DEVEZ RESPECTER le nom des méthodes mentionnées ci-dessous. AUSSI, les paramètres dans l'entête des constructeurs et méthodes doivent respecter l'ordre d'énumération dans les tableaux décrivant leurs paramètres. Sinon les tests de votre classe risquent de ne pas compiler, et vous perdrez des points pour non-respect des spécifications.

Constructeur

Paramètre	Type	Description
categorie	`int`	La catégorie associée à cet objet perdu. **Antécédent** : doit être une valeur entre `0` et `CATEGORIES.length - 1` inclusivement.
date	`Date`	La date de consignation de cet objet perdu.
localisation	`String`	L'endroit où l'on range l'objet perdu consigné (pour le retrouver plus facilement lorsqu'il sera réclamé)

Note : lorsqu'il y a un **antécédent** sur un paramètre, on considère cette condition vraie lors de l'appel de la méthode (ou du constructeur). Tout antécédent doit être inscrit dans la Javadoc du paramètre (pour toute méthode ou constructeur).

Ce constructeur construit un objet `ObjetPerdu` en initialisant ses attributs `categorie`, `date`, et `localisation` avec les valeurs passées en paramètre. De plus :

- Le constructeur assigne automatiquement à l'attribut `id` un numéro d'identification unique correspondant à la valeur de l'attribut `sequenceId`. L'attribut `sequenceId` est ensuite incrémenté de 1 (et devient ainsi le numéro d'identification du prochain objet perdu qui sera créé).
- Le tableau `motsCles` doit être instancié avec une longueur égale à 0.

Méthodes d'instance publiques – observateurs (*getters*)

Les **4** ***getters*** pour les attributs d'instance (`id`, `categorie`, `date`, et `localisation`) qui ne prennent aucun paramètre, et retournent la valeur de l'attribut d'instance concerné dont voici les noms (que vous devez respecter) :

`getId`, `getCategorie`, `getDate`, et `getLocalisation`.

Méthodes d'instance publiques – modificateurs (*setters*)

Les **3** ***setters*** suivants qui prennent en paramètre la valeur de modification de l'attribut concerné, et qui ne retournent rien (`void`) : `setCategorie`, `setDate`, et `setLocalisation`.

Antécédent sur le paramètre de la méthode `setCategorie` :
- doit être une valeur entre `0` et `CATEGORIES.length - 1` inclusivement.

Autres méthodes d'instance publiques

Nom méthode : `obtenirMotsCles`
Type retour : `String`
Paramètres : aucun

Permet d'obtenir une chaine de caractères contenant tous les mots clés associés à cet objet perdu (stockés dans l'attribut `motsCles`). Plus précisément, la chaine retournée doit contenir les mots clés dans l'ordre où ils apparaissent dans le tableau `motsCles`, et les mots clés doivent être séparés par un espace (PAS d'espace au début ni à la fin de la chaine retournée).

Exemples :
- Si le tableau `motsCles` contient `{"t-shirt", "rouge", "blanc", "rayure"}` alors la chaine retournée par cette méthode sera `"t-shirt rouge blanc rayure"`.
- Si le tableau `motsCles` est vide, la chaine retournée par cette méthode sera la chaine vide.

Nom méthode : `ajouterMotCle`
Type retour : `boolean`

Paramètre	Type	Description
motCle	`String`	Un mot clé à associer à cet objet perdu.

Permet d'ajouter le `motCle` donné en paramètre dans le tableau `motsCles` de cet objet perdu. Le `motCle` n'est PAS ajouté si :
- `motCle` est `null`
- `motCle` est une chaine vide
- `motCle` existe déjà (**peu importe la casse**) dans le tableau `motsCles`.

Si le `motCle` peut être ajouté au tableau `motsCles` de cet objet perdu, on ajoute une case à la fin du tableau `motsCles`, et le `motCle` est ajouté dans cette nouvelle case. Après l'ajout, la longueur du tableau `motsCles` aura augmenté de un. La méthode retourne `true` si le `motCle` a bien été ajouté, `false` sinon.

Nom méthode : `estAssocieACeMotCle`
Type retour : `boolean`

Paramètre	Type	Description
motCle	`String`	Le mot clé dont on veut savoir s'il est associé à cet objet perdu.

Cette méthode retourne `true` si le `motCle` donné en paramètre est présent (sans tenir compte de la casse) dans le tableau `motsCles` de cet objet perdu, retourne `false` sinon.

Nom méthode : `supprimerMotCle`
Type retour : `boolean`

Paramètre	Type	Description
motCle	`String`	Le mot clé dont on veut retirer l'association à cet objet perdu.

Permet de supprimer le `motCle` donné en paramètre du tableau `motsCles` de cet objet perdu. Notez que la recherche du mot clé à supprimer dans le tableau doit se faire sans tenir compte de la casse. Si `motCle` n'est pas dans le tableau `motsCles`, cette méthode ne supprime rien, et le tableau `motsCles` demeure inchangé. Si `motCle` est trouvé dans le tableau `motsCles` (sans tenir compte de la casse), celui-ci est supprimé du tableau, et la longueur du tableau `motsCles` aura diminué de un. L'ordre des mots clés restants dans le tableau ne doit pas être modifié.

Voici quelques exemples de suppressions :

Mot clé à supprimer	Tableau motsCles avant la suppression	Tableau motsCles après la suppression
`"A"`	`[ "C", "K", "A", "B", "E", "P" ]`	`[ "C", "K", "B", "E", "P" ]`
`"C"`	`[ "C", "K", "A", "B", "E", "P" ]`	`[ "K", "A", "B", "E", "P" ]`
`"P"`	`[ "C", "K", "A", "B", "E", "P" ]`	`[ "C", "K", "A", "B", "E" ]`
`"A"`	`[ "A" ]`	`[] //tableau de longueur 0`

Cette méthode retourne `true` si le `motCle` donné en paramètre a été retiré du tableau `motsCles` de cet objet perdu, retourne `false` sinon.

Nom méthode : `toString`
Type retour : `String`
Paramètres : aucun

Retourne une représentation de cet `ObjetPerdu` sous forme d'une chaine de caractères.

CETTE MÉTHODE VOUS EST FOURNIE dans le fichier **toString.txt**. Vous devez simplement la copier-coller dans votre classe.

Méthodes de classe publiques

Nom méthode : `getSequenceId`
Type retour : `int`
Paramètre : aucun.

Retourne la valeur de l'attribut de classe `sequenceId`.

Nom méthode : `setSequenceId`
Type retour : `void`

Paramètre	Type	Description
valeur	`int`	La valeur à assigner à la variable de classe `sequenceId`.

Permet de modifier la valeur de l'attribut de classe `sequenceId` par la `valeur` donnée en paramètre. La nouvelle valeur de `sequenceId` devient ainsi l'id à assigner au prochain objet perdu qui sera créé.

1.2 PARTIE 2 : IMPLÉMENTATION DE LA CLASSE `GestionObjetsPerdus`

Il s'agit d'écrire un programme qui permet la gestion des objets perdus, et qui pourrait éventuellement être utilisé dans un comptoir des objets perdus. Votre programme doit se trouver dans une classe d'application nommée `GestionObjetsPerdus.java` qui contiendra, entre autres, la méthode `main`. Dans ce TP, les objets perdus consignés doivent être stockés en mémoire dans un tableau de type `ObjetPerdu`. De plus, les objets perdus stockés dans ce tableau seront sauvegardés dans un fichier texte lors de la fermeture de l'application (menu Quitter). Lors du démarrage de l'application, les objets perdus stockés dans le fichier texte seront remontés en mémoire, de sorte que vous ne perdrez plus vos données lorsque l'application se termine. Ces méthodes de sauvegarde et récupération des données vous sont données dans la classe `TP3Utils`.

1.2.1 Lancement du programme

Au démarrage du programme, celui-ci affiche le titre de l'application suivi d'un menu principal de 7 options. Vous devez valider le choix au menu. ATTENTION, les SEULS choix valides sont exactement 1, 2, 3, 4, 5, 6 ou 7. Par exemple, si l'utilisateur ne fait que taper ENTRÉE, ou s'il saisit "11", ce n'est pas valide. Notez aussi que la saisie du choix au menu ne doit jamais faire planter le programme.

Note sur l'implémentation :

Au tout début de votre méthode `main`, vous devez créer le tableau `objetsPerdus` qui contient tous les objets perdus déjà consignés à partir des informations enregistrées dans le fichier texte objetsPerdus.txt (qui se trouve à la racine de votre projet, s'il existe). Pour ce faire, appelez la méthode `recupererDonnees` de la classe `TP3Utils`. Celle-ci vous retournera un tableau **de longueur minimale** contenant tous les objets perdus (non `null`) ayant été sauvegardés dans le fichier objetsPerdus.txt lors de la dernière fermeture de l'application. Au premier démarrage du programme, ce fichier n'existe pas encore. Dans ce cas, le tableau retourné par la méthode `recupererDonnees` sera de longueur 0. Ce sera seulement au moment de la fermeture du programme (au choix de l'option Quitter) que votre programme sauvegardera les données en mémoire dans le fichier objetsPerdus.txt. Ce fichier sera donc créé au moment où vous quitterez l'application. **La première instruction de votre méthode `main` doit donc être l'initialisation du tableau des objets perdus** à l'aide de la méthode `recupererDonnees` de la classe `TP3Utils`, comme ceci :

```
ObjetPerdu[] objetsPerdus = TP3Utils.recupererDonnees();
```

Note : Lors de vos tests, vous pouvez copier-coller le fichier objetsPerdus.txt (fourni avec l'énoncé de ce TP) à la racine de votre projet pour démarrer votre programme avec un tableau des objets perdus non vide (qui contient déjà quelques objets consignés). Ceci peut vous faire gagner du temps pour tester les options de recherche, par exemple, ou l'option pour rendre un objet réclamé, ou afficher tous les objets. Cela vous évitera d'avoir à consigner plusieurs objets avec l'option 1 avant de pouvoir faire des tests pour les autres options.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier exempleExec_validationMenu.txt).

1.2.2 Option 1 - Consigner un objet perdu

Au choix de cette option, le programme demande d'abord de **saisir la catégorie de l'objet perdu** à consigner. Vous devez valider la catégorie saisie entre 1 et 5 inclusivement. ATTENTION, les SEULS choix valides sont exactement 1, 2, 3, 4, et 5. Par exemple, si l'utilisateur ne fait que taper ENTRÉE, ou s'il saisit "11", ce n'est pas valide. Notez aussi que la saisie de la catégorie ne doit jamais faire planter le programme.

Ensuite, le programme demande de **saisir la date de consignation**. Il demande d'abord d'entrer le jour de la date. Vous devez valider que le jour saisi est un entier strictement positif. Il demande ensuite de saisir le mois de la date. Vous devez valider que le mois saisi est un entier strictement positif. Il demande finalement de saisir l'année de la date. Vous devez valider que l'année saisie est un entier strictement positif. Notez aussi que la saisie du jour, du mois, ou de l'année ne doit jamais faire planter le programme. Une fois que le jour, le mois, et l'année ont été saisis, le programme valide que le jour, le mois, et l'année saisis forment une date valide. Si ce n'est pas le cas, il affiche un message d'erreur et recommence la saisie du jour, du mois, et de l'année (et ce, tant que le jour, le mois, et l'année ne forment pas une date valide). Notez que pour qu'une date soit considérée comme valide, son année doit être supérieure ou égale à `Date.ANNEE_MIN`, et son jour, son mois et son année doivent former une date valide. De plus, dans le cadre de cette application, une date valide ne doit pas être postérieure à la date du jour (la date au moment de la consignation de l'objet perdu), ce qui ne ferait pas de sens.

Puis, le programme demande de **saisir les mots clés** qu'on désire associer à cet objet perdu. Ces mots clés servent de description de l'objet consigné (et permettra plus tard une recherche par mots clés). Vous devez valider la chaine entrée. Une chaine valide doit contenir entre 2 et 50 caractères inclusivement. Les mots clés entrés doivent être séparés par un (ou plusieurs espaces). **Note sur l'implémentation** : le programme s'occupe de séparer les mots clés au(x) espace(s), et de les ajouter au tableau des mots clés de cet objet perdu (sans espace au début ou à la fin de chaque mot clé). Par exemple, si les mots clés saisis sont : `"t-shirt rouge blanc rayure"`, le tableau des mots clés contiendra les 4 mots clés suivants ; `"t-shirt"`, `"rouge"`, `"blanc"`, et `"rayure"` (stockés dans le tableau dans le même ordre que la saisie). Vous pouvez utiliser la méthode `split` de la classe `String` pour séparer les mots clés aux espaces. L'appel de `motsCles.split("\\s+")` retourne un tableau contenant tous les mots clés qui ont été séparés aux espaces.

Finalement, le programme demande de **saisir la localisation** de cet objet perdu (l'endroit où il est rangé physiquement dans l'attente d'une réclamation). Vous devez valider la chaine entrée. Une chaine valide doit contenir entre 5 et 50 caractères inclusivement.

Note sur l'implémentation : Lorsque toutes les informations ont été saisies, un nouvel objet perdu est créé avec ces informations, puis est ajouté au tableau des objets perdus (`objetsPerdus`). Si le tableau `objetsPerdus` n'a plus de place avant l'ajout, il faut l'agrandir de 4 cases avant d'ajouter le nouvel objet perdu.

Lorsque l'objet a été consigné, le programme affiche de nouveau le menu dans l'attente d'un choix de l'utilisateur.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier exempleExec_options1_6.txt).

1.2.3 Option 2 - Rendre un objet réclamé

Cette option permet de rendre un objet réclamé, et donc de le supprimer du tableau des objets perdus. Si le tableau des objets perdus ne contient aucun objet perdu, le programme affiche un message à cet effet, et demande de taper sur ENTRÉE pour revenir au menu.

Si le tableau des objets perdus contient au moins un objet perdu, le programme demande de saisir le numéro d'identification de l'objet à rendre. Vous devez valider le numéro saisi. Un numéro d'identification valide est un entier strictement positif. Si le numéro valide entré ne correspond à aucun objet perdu dans le tableau des objets perdus consignés, le programme affiche un message à cet effet, et demande de taper sur ENTRÉE pour revenir au menu.

Si le numéro valide entré correspond à un objet dans le tableau des objets perdus, le programme demande alors une confirmation pour la remise. Vous devez valider la réponse saisie. Une réponse valide est soit la chaine `"oui"` (pour confirmer la remise) soit la chaine `"non"` (pour annuler la remise) et ce, **peu importe la casse**.

Si la réponse valide est négative, le programme affiche un message confirmant l'annulation, et demande de taper ENTRÉE pour revenir au menu. Dans ce cas, l'objet perdu n'est pas supprimé du tableau des objets perdus. Si la réponse valide est affirmative, le programme affiche un message de confirmation de la remise, et demande de taper ENTRÉE pour revenir au menu. Dans ce cas, le programme supprime l'objet perdu du tableau des objets perdus.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier exempleExec_options2_6.txt).

1.2.4 Option 3 - Rechercher un objet par mots clés

Cette option permet de rechercher un objet perdu par mot(s) clé(s). Si, au choix de cette option, il n'y a aucun objet perdu consigné, le programme affiche un message à cet effet, et demande de taper ENTRÉE pour revenir au menu.

S'il y a au moins un objet de consigné, le programme demande d'entrer le(s) mot(s) clé(s) pour la recherche. Vous devez valider la chaine entrée. Une chaine valide doit contenir entre 2 et 50 caractères inclusivement. Les mots clés entrés doivent être séparés par un (ou plusieurs espaces). **Note sur l'implémentation** : le programme s'occupe de séparer les mots clés au(x) espace(s).

Par exemple, si les mots clés saisis sont :
`"t-shirt rouge blanc rayure"`,
les mots clés à utiliser pour la recherche sont :
`"t-shirt"`, `"rouge"`, `"blanc"`, et `"rayure"`.

Le programme affiche ensuite tous les objets perdus consignés qui sont associés à TOUS les mots clés saisis (peu importe la casse). La recherche effectuée est une recherche exacte. Par exemple, si le mot clé saisi pour la recherche est "noir", on ne retournera pas un objet associé au mot clé "noir**e**" plutôt que "noir", si le mot clé pour la recherche est "soulier", on ne retournera pas un objet ayant le mot clé "soulier**s**" plutôt que "soulier", etc. Il faut que les mots clés correspondent exactement (sans tenir compte de la casse). De plus, les objets trouvés (qui contiennent tous les mots clés saisis pour la recherche) doivent être affichés **en ordre croissant de leur id (numéro d'identification)**.

Si aucun objet n'est trouvé, le programme affiche un message à cet effet. Dans tous les cas, le programme demande ensuite de taper ENTRÉE pour revenir au menu.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP
(fichiers exempleExec_options3_4_5.txt, exempleExec_options3.txt).

1.2.5 Option 4 - Rechercher un objet par date(s)

Cette option permet de rechercher un objet perdu par date ou par période. Si, au choix de cette option, il n'y a aucun objet perdu consigné, le programme affiche un message à cet effet, et demande de taper ENTRÉE pour revenir au menu.

S'il y a au moins un objet de consigné, le programme demande si l'utilisateur veut spécifier une date ou bien une période pour faire la recherche. Vous devez valider le choix de l'utilisateur. Les deux seules réponses valides sont "d" pour date ou "p" pour période, peu importe la casse. Aucune autre réponse n'est valide.

Lorsque la réponse valide saisie est "d" :

Le programme demande alors de **saisir la date pour la recherche**. Il demande d'abord d'entrer le jour de la date. Vous devez valider que le jour saisi est un entier strictement positif. Il demande ensuite de saisir le mois de la date. Vous devez valider que le mois saisi est un entier strictement positif. Il demande finalement de saisir l'année de la date. Vous devez valider que l'année saisie est un entier strictement positif. Notez aussi que la saisie du jour, du mois, ou de l'année ne doit jamais faire planter le programme. Une fois que le jour, le mois, et l'année ont été saisis, le programme valide que le jour, le mois, et l'année saisis forment une date valide. Si ce n'est pas le cas, il

affiche un message d'erreur et recommence la saisie du jour, du mois, et de l'année (et ce, tant que le jour, le mois, et l'année ne forment pas une date valide). Notez que pour qu'une date soit considérée comme valide, son année doit être supérieure ou égale à `Date.ANNEE_MIN`, et son jour, son mois et son année doivent former une date valide. De plus, dans le cadre de cette application, une date valide ne doit pas être postérieure à la date du jour (la date au moment d'effectuer la recherche).

Lorsque la date entrée est valide, le programme affiche tous les objets perdus dont la date de consignation est égale à la date saisie pour la recherche, **en ordre croissant des numéros d'identification (id)** des objets trouvés. Si aucun objet n'est trouvé, le programme affiche un message à cet effet. Dans tous les cas, le programme demande ensuite de taper ENTRÉE pour revenir au menu.

Lorsque la réponse valide saisie est "p" :

Le programme demande alors de **saisir la date de début de la période** à spécifier pour la recherche. Vous devez valider la date de début comme expliqué ci-dessus (lorsque la réponse est "d"). Le programme demande ensuite de **saisir la date de fin de la période** à spécifier pour la recherche. Vous devez valider la date de fin de la même manière que la date de début.

Lorsque les deux dates entrées sont valides, le programme affiche alors tous les objets perdus dont la date de consignation se trouve entre la date de début saisie, et la date de fin saisie inclusivement. Les objets doivent être affichés en ordre croissant de leur date de consignation. Notez que si la date de fin précède la date de début, la recherche ne retourne aucun objet. Si la date de début est égale à la date de fin, ça revient à faire une recherche par date avec la date donnée. Si aucun objet n'est trouvé, le programme affiche un message à cet effet. Dans tous les cas, le programme demande ensuite de taper ENTRÉE pour revenir au menu.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP
(fichiers exempleExec_options3_4_5.txt, exempleExec_options4d.txt, exempleExec_options4p.txt).

1.2.6 Option 5 - Rechercher un objet par catégorie

Cette option permet de rechercher un objet perdu par catégorie. Si, au choix de cette option, il n'y a aucun objet perdu consigné, le programme affiche un message à cet effet, et demande de taper ENTRÉE pour revenir au menu.

S'il y a au moins un objet de consigné, le programme demande d'entrer le numéro de catégorie pour la recherche. Vous devez valider la catégorie saisie entre 1 et 5 inclusivement. ATTENTION, les SEULS choix valides sont exactement 1, 2, 3, 4, et 5. Par exemple, si l'utilisateur ne fait que taper ENTRÉE, ou s'il saisit "11", ce n'est pas valide.

Le programme affiche ensuite tous les objets perdus consignés qui sont dans la catégorie saisie pour la recherche. De plus, les objets trouvés doivent être affichés **en ordre croissant de leur id (numéro d'identification)**. Si aucun objet n'est trouvé, le programme affiche un message à cet effet. Dans tous les cas, le programme demande ensuite de taper ENTRÉE pour revenir au menu.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP
(fichiers exempleExec_options3_4_5.txt, exempleExec_options5.txt).

1.2.7 Option 6 - Afficher tous les objets consignés

Au choix de cette option, le programme affiche tous les objets perdus consignés (se trouvant dans le tableau `objetsPerdus`), ordonnés (ordre croissant) par id (numéro d'identification des objets perdus). S'il n'y a aucun objet consigné, le programme affiche un message à cet effet. Dans tous les cas, le programme demande de taper ENTRÉE pour revenir au menu.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier exempleExec_options1_6.txt, par exemple).

1.2.8 Option 7 - Quitter

Au choix de cette option, le programme se termine, et affiche un message à cet effet.

Note sur l'implémentation :

Au choix de cette option, le programme doit sauvegarder tous les objets perdus consignés dans le tableau `objetsPerdus` dans le fichier texte objetsPerdus.txt. Pour ce faire, vous devez appeler la méthode `sauvegarder` de la classe `TP3Utils`, comme ceci : `TP3Utils.sauvegarder(objetsPerdus);`

L'appel de cette méthode va créer (ou mettre à jour) le fichier objetsPerdus.txt (à la racine de votre projet). Notez que vous pouvez supprimer ce fichier (à la racine de votre projet), si vous voulez démarrer votre programme sans qu'il contienne d'objets perdus consignés (comme au premier démarrage).

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier exempleExec_options1_6.txt, par exemple).

2. Précisions

Pour la classe `ObjetPerdu`

- Vous devez respecter le principe d'encapsulation des données.
- **Les seules variables globales permises sont les attributs (de classe et d'instance) énumérés à la section 1.1**. Vous pouvez cependant ajouter d'autres constantes de classe si vous le jugez pertinent.
- Les méthodes énumérées dans la section 1.1 **sont les seules méthodes publiques dans la classe `ObjetPerdu`**. Vous pouvez, ET DEVRIEZ ajouter des méthodes privées pour bien découper vos méthodes publiques et éviter la répétition du code.

Pour la classe `GestionObjetsPerdus`

- Vous classe `GestionObjetsPerdus` doit contenir la méthode `main` et toutes les autres méthodes de classe que vous aurez conçues pour bien découper votre code.
- Prenez soin de bien découper votre code à l'aide de méthodes. Voici quelques lignes directrices pour vous guider dans la conception de vos méthodes :
  - Chaque fois que vous vous apercevez que différentes parties de votre code se ressemblent énormément, essayer d'en faire une méthode bien paramétrée (si possible).
  - Faites une méthode pour chaque petite tâche spécifique à accomplir dans la résolution du problème.
  - Si une méthode semble longue, posez-vous la question à savoir si elle pourrait être découpée en plusieurs méthodes plus spécialisées.
- Plusieurs méthodes utiles pour la validation, par exemple, ou pour le tri des objets perdus se trouvent dans la classe `TP3Utils`. Utilisez-les !
- **Toutes vos méthodes doivent être `public static`.**
- Lorsqu'on vous demande de valider une valeur saisie par l'utilisateur, cela sous-entend une **BOUCLE** de validation qui sollicite la valeur, lit la valeur, et lorsque la valeur saisie est invalide, affiche un message d'erreur, sollicite et lit de nouveau la valeur, et ce tant que la valeur saisie n'est pas valide.
- **Vous DEVEZ utiliser la classe `Clavier.java` pour effectuer toutes les saisies.**
- Vous DEVEZ respecter à la lettre les **informations, les messages**, et le **format** de l'affichage qui sont montrés dans les exemples d'exécution fournis avec l'énoncé du TP. Vous devez considérer les exemples d'exécution comme des spécifications complémentaires à celles décrites dans cet énoncé.
- Votre programme NE DOIT JAMAIS PLANTER lors d'une saisie faite par l'utilisateur.
- Toute **VARIABLE GLOBALE** est **INTERDITE : toutes les variables doivent être déclarées à l'intérieur (et au début) d'une méthode** (sauf pour les boucles `for` où vous pouvez déclarer la variable de contrôle dans l'entête de la boucle).

- Utilisez des constantes (`final`) autant que possible : les messages affichés, les bornes de validation, etc.
- Les constantes doivent être déclarées et initialisées au niveau de la classe (constantes globales) : `public static final` (ou `public final static`)…
- L'affichage des résultats doit se faire à la console.
- Utilisez des variables réelles uniquement lorsque requis. Ex. : un compteur ne doit pas être `float` ou `double`.

Pour les deux classes à remettre :

- Documentez correctement toutes vos méthodes (sauf `main`) à l'aide des commentaires de documentation Java.
- Vos classes à remettre doivent se trouver dans le paquetage par défaut (les classes fournies vont aussi dans le paquetage par défaut).
- Vous DEVEZ respecter le style Java, et les bonnes pratiques de programmation vues en classe.
- Vos fichiers de code source doivent être encodés en UTF-8.
- Votre code doit compiler avec le **JDK 7**. Si vous n'utilisez que ce qu'on a vu en classe, ce sera le cas.
- N'oubliez pas d'écrire (entre autres) votre nom complet, et votre code permanent dans l'entête de vos classes à remettre.
- **Les seules CLASSES PERMISES sont** `String`, `Math`, `Character`, `Clavier`, et `System` (en plus des classes fournies pour ce TP).
- **UTILISATIONS INTERDITES :**
  - L'instruction `break` (sauf comme dernière instruction d'un `case` dans un `switch`).
  - L'instruction `continue`.
  - L'instruction `return` dans une méthode dont l'entête mentionne qu'elle ne retourne rien (`void`).
  - La modification de la variable de contrôle à l'intérieur d'une boucle `for`.
  - L'instruction `System.exit(…)`.

  ATTENTION : Une pénalité de **5 points sur 100** (jusqu'à concurrence de 30 points) sera attribuée pour chaque utilisation interdite. Par exemple, si vous utilisez 3 fois un `break` interdit, vous perdrez 15 points.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé du TP (et les exemples d'exécution donnés avec l'énoncé du TP) est susceptible d'engendrer une perte de points.

Note : *Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.*

3. Détails sur la correction

3.1 LA QUALITÉ DU CODE (40 POINTS)

Concernant les critères de correction du code, lisez attentivement le document "**Critères généraux de correction du code Java"**. De plus, votre code sera noté sur le respect des conventions de style Java vues en classe (dont un résumé se trouve dans le document **"Quelques conventions de style Java"**. Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours.

Note : Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, sur le respect des spécifications/consignes mentionnées dans ce document, et sur le respect des bonnes pratiques de programmation.

3.2 L'EXÉCUTION (60 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Note : *Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible.*

4. Date et modalité de remise

4.1 REMISE

Date de remise : AU PLUS TARD le 21 avril 2023 à 23h59. **AUCUN RETARD PERMIS.**

Les deux fichiers à remettre : `ObjetPerdu.java` et `GestionObjetsPerdus.java` (les remettre séparément, PAS DANS UNE ARCHIVE ZIP, RAR…)

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boite de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE)**.

Vérifiez deux fois plutôt qu'une que vous avez remis les BONS fichiers.

4.2 POLITIQUE CONCERNANT LES RETARDS

AUCUN RETARD NE SERA ACCEPTÉ.

4.3 REMARQUES GÉNÉRALES

- **Aucun travail reçu par courriel ne sera accepté.** Plus précisément, un travail envoyé par courriel sera considéré comme un travail non remis.
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire votre nom complet et votre code permanent (entre autres) dans l'entête des classes à remettre.**
- **N'attendez pas à la dernière minute pour commencer le travail**, vous allez fort probablement rencontrer des problèmes inattendus!

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !