TP
Programmation orientée objets

Yannick Loiseau yannick.loiseau@uca.fr

Licence 2me année
version du 30 janvier 2023

TP 1

Échauffement

1.1 Introduction

Exercice 1 (Premiers pas en Java)

Question 1 (La base) *:* Créez un fichier Hello.java contenant un programme Java qui affiche `Hello World !` à l'écran.

Pour afficher, utilisez la méthode `println` [a] de `System.out` [b].

a. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/PrintStream.html#println(java.lang.Object)

b. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#out

Question 2 (Compilation) *:* Compilez à l'aide de la commande `javac Hello.java`. Quel est le nom du fichier généré ? Que contient-il ?

Question 3 (Exécution) *:* Exécutez ce programme à l'aide de la commande `java Hello`.

Question 4 (Organisation) *:* Créez un répertoire `src` et déplacez-y votre fichier source. Créez un repertoire `build` et essayez à nouveau de compiler, avec `javac -d build src/*.java`. Quel est l'intérêt de l'option `-d`.

Ajoutez la ligne `package poo.tp.premierspas ;` au début de votre programme et recommencez. Que ce passe-t-il ?

Essayer de l'exécuter à nouveau. Vous devrez peut-être utiliser l'option `-cp` de `java`. Regardez son utilité.

Question 5 (Paramètres) *:* Modifiez le programme pour qu'il accepte le nom de la personne à saluer en argument de la ligne de commande. Si aucun nom n'est fourni, le programme doit échouer avec un message d'erreur adapté.

Question 6 (Erreurs) *:* Introduisez délibérément des erreurs dans votre programme, et tentez de l'exécuter, afin de vous familiariser avec les messages d'erreur du compilateur Java.

Exercice 2 (Nombre mystère)

On se propose ici de réaliser un jeu simple de question–réponse. Le but du programme est de faire deviner à l'utilisateur un nombre dans un intervalle donné. Le programme devra donc lire dans ses options de ligne de commande les deux valeurs minimales et maximales des nombres à deviner (et déclencher une erreur si le 2^{e} est inférieur au 1er).

Il devra ensuite s'exécuter en mode interactif, demandant successivement à l'utilisateur de donner un nombre, et lui indiquant si le nombre à deviner est plus grand ou plus petit que la proposition.

On peut lire une chaine saisie au clavier avec `System.console().readLine()` [a]. Du texte peut être convertie en entier avec `Integer.parseInt(String)` [b]. On peut générer des nombres aléatoires avec la classe `java.util.Random`, notamment sa méthode `nextInt(int)` [c]. Pour afficher un message d'erreur et terminer le programme, on peut utiliser respectivement la méthode `printf` [d] sur `System.err` [e] et `System.exit(int)` [f] avec une valeur positive décrivant l'erreur.

a. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Console.html#readLine()

b. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Integer.html#parseInt(java.lang.String)

c. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html#nextInt(int)

d. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/PrintStream.html#printf(java.lang.String,java.lang.Object...)

e. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#err

f. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#exit(int)

Question 1 *:* Dans un premier temps, réalisez ce programme comme une unique méthode `public static void main(String[] args)`. Pensez à signaler si la proposition est à l'extérieur de l'intervalle demandé.

Question 2 *:* Le programme est maintenant une instance de la classe. Déplacer les variables locale (min, max, et nombre à deviner) dans des attributs. Découpez la méthode `main` en sous méthodes d'instances. La méthode `main` se contente de créer une instance, de l'initialiser et d'appeler sa méthode `run`.

1.2 Familles

Exercice 3 (Dates) Une date est constituée d'une année (supérieure ou égale à 0), d'un mois (compris entre 1 et 12) et d'un jour (compris entre 1 et 31).

Question 1 *:* Créez la classe `Date` permettant de créer des dates et de les afficher. On ne vérifiera pas que le nombre de jours est correct par rapport au mois. Interdisez aux utilisateurs de modifier une date créée. La classe `Date` doit obligatoirement se trouver dans un fichier nommé `Date.java`

Si l'un des paramètres passé au constructeur est erroné, vous pouvez utiliser `throw new IllegalArgumentException();` pour indiquer une erreur.

Question 2 *:* Compilez votre programme au moyen de la commande `javac -Xlint :all Date.java`. Pouvez-vous l'exécuter ?

Question 3 *:* La classe `Object` (dont hérite la classe `Date`) possède une méthode `public String toString()` permettant de retourner une chaîne de caractères associée à un objet. Redéfinissez cette méthode.

L'opérateur permettant de concaténer des chaînes de caractère est `+`. Ainsi, `"abc" + 3 + "def" + "ghi"` génère la chaîne de caractère `"abc3defghi"`.
On peut aussi utiliser la méthode statique `String.format` [a].

a. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html#format(java.lang.String,java.lang.Object...)

Question 4 *:* Écrivez une méthode de comparaison permettant de savoir si la date actuelle est antérieure, égale ou postérieure à une date passée en paramètre (voir l'interface `Comparable<T>` [1]).

Exercice 4 (Personnes) Une personne possède un nom, un prénom et une date de naissance.

Question 1 *:* Créez une classe `Personne` permettant de manipuler les informations concernant une personne. Notamment, une personne doit pouvoir changer son nom.

Question 2 *:* Une personne peut être soit célibataire, soit mariée. Lorsqu'elle est mariée, il est important de connaître la date du mariage. Programmez une méthode permettant à deux personne de se marier. On vérifiera que la date du mariage est postérieure à la date de naissance de chaque personne.

Programmez aussi une méthode permettant de savoir si une personne est mariée ou non.

Exercice 5 (Liens de parenté)

1. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Comparable.html#

Question 1 *:* À présent, on ajoute à chaque personne deux parents. Ces parents sont optionnels, dans le sens où ils peuvent être inconnus.

Question 2 *:* Programmez une méthode permettant de savoir si la personne en question est le frère ou la sœur d'une autre personne passée en paramètre (un seul parent en commun est considéré suffisant).

Question 3 *:* On considère maintenant que les personnes sont soit des hommes, soit des femmes et que les hommes ne peuvent pas changer leur nom de famille (le système est assez ancien). Faites les modifications appropriées.

TP 2

Exercice 1 (Animaux)

Question 1 *:* Écrivez une classe abstraite `Animal`. On supposera qu'un animal possède toujours un nom, mais ne peux pas en changer. Redéfinissez la méthode `toString`.

Écrivez aussi l'accesseur permettant d'obtenir le nom de l'animal.

Question 2 *:* Écrivez deux sous-classes `Dog` et `Cat`. Redéfinissez la méthode `toString` pour chacune.

Question 3 *:* Dans la fonction main, créez les objets `medor` et `felix` de la manière suivante :

```
Dog medor = new Dog("Medor");
Animal felix = new Cat("Felix");
```

Affichez les objets `medor` et `felix`. Quelle est la méthode `toString` appellée ?

Question 4 *:* Dans la classe `Dog`, créez une méthode `public void woof()` affichant un message. Pouvez-vous faire aboyer l'objet `medor` ?

Question 5 *:* Dans la classe `Chat`, créez une méthode `public void meow()` affichant un message. Pouvez-vous faire miauler l'objet `felix` ?

Question 6 *:* L'objet `felix` ne peut pas miauler car il n'est pas considéré comme un `Chat` mais comme un `Animal` au moment de la compilation. Réessayez en appelant un chat un chat : `((Cat)felix).meow();`

Question 7 *:* Que se passe-t-il si vous essayez de faire miauler `medor` ?

Exercice 2 (Mots) La classe `String`[1] du paquetage `java.lang` permet de manipuler des chaînes de caractère.

1. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html#

Question 1 *:* Écrire une classe `Mot` qui permet de stocker une chaîne de caractères lue au clavier.

On peut lire une chaine saisie au clavier avec `System.console().readLine()`.

Question 2 *:* Ajoutez la méthode `toString` pour votre classe `Mot`.

Question 3 *:* Écrivez une méthode `afficheVoyelles` qui affiche les voyelles du mot, dans l'ordre où elles apparaissent. Par exemple, cette fonction affiche `ioaiue` à partir du mot `informatique`.

Pour obtenir la longueur d'une chaîne de caractères, la classe `String` met à disposition la méthode `public int length()`. Pour obtenir le caractère en position *index*, la classe `String` expose la méthode `public char charAt(int index)`.

Question 4 *:* Ajoutez une méthode `estPalindrome` qui permet de savoir si le mot est un *palindrome*. Un palindrome est un mot qui peut être lu dans les deux sens.

Question 5 *:* Écrivez une méthode `estContenu` qui indique si un `Mot` donné en paramètre contient ou non le mot en cours.

Question 6 *:* Avec des boucles `for` et sans tableau, triez les lettres d'un `Mot`. Par exemple, pour le mot `informatique`, cela donne `aefiimnoqrtu`.

TP 3

Exercice 1 (Le jeu de la bataille (simplifiée)) Nous allons programmer une version simplifiée du jeu de la bataille. Dans ce jeu de cartes, chaque joueur joue la carte de son choix (de son paquet) à chaque tour. La carte ayant la plus forte valeur remporte, ou la plus grande couleur si les valeurs sont égales. Le joueur ayant mis la carte la plus forte ramasse les deux cartes posées et le jeu recommence, jusqu'à ce que l'un des deux joueurs n'ait plus de carte [1]

Le diagramme de classes en figure 3.1 présente une version simplifiée de l'architecture de ce programme.

Question 1 *:* Programmez la classe `Carte`. La méthode `compareTo` retourne une valeur négative (strictement) si la carte actuelle est plus faible que l'autre carte, 0 si elles sont égales, et une valeur positive (strictement) sinon. Elle est définie dans l'interface `Comparable<T>`. Votre classe `Carte` devra donc l'implémenter.

Question 2 *:* Programmez la classe `Paquet`. Pour le moment, ne programmez pas le contenu de la méthode `public void melanger()`.

Question 3 *:* Programmez la classe abstraite `Joueur`.

Question 4 *:* Programmez la classe `Ordinateur`. Pour jouer, l'ordinateur choisi une carte aléatoirement dans son paquet.

Question 5 *:* Programmez finalement la classe `Jeu`.

Question 6 *:* Programmez la méthode `melanger` de la classe `Paquet`. Pour mélanger le jeu, on échangera x fois deux cartes choisies aléatoirement. La variable x aura pour valeur deux fois la taille du paquet.

1. Ce jeu n'est certe pas très distrayant, dans le sens où le joueur ayant la carte la plus forte initialement est sûr de ne pas pouvoir perdre (et même de gagner s'il joue cette carte à chaque tour). Qu'à cela ne tienne, c'est la programmation du jeu qui sera distrayante.

FIGURE 3.1 – Diagramme de classes du jeu de la bataille

Question 7 : Programmez une classe `Humain` héritant de `Joueur` et permettant à un joueur humain de jouer, en affichant son jeu à l'écran et demandant quelle carte il décide de jouer.

Question 8 : Ajouter enfin la méthode `main` à la classe `Jeu`. Elle initialise le jeu avec un joueur humain et un ordinateur, distribue les cartes et joue la partie. Lorsque la partie est terminée, le gagnant est affiché et le programme se termine.

TP 4

Listes, tableaux et comparaison de structures de données

Il est possible de stocker des données dans plusieurs structures. Le choix d'une structure de donnée adaptée peut avoir un impact sur la performance du programme. Dans ce TP, nous allons étudier différentes structures de données et comparer leur efficacité sur un exemple simple.

Exercice 1 (Tableaux)

Question 1 *:* Implémentez une classe `Matrice` qui permet de manipuler des matrices à deux dimensions.

Question 2 *:* Créez une méthode `public Matrice ajouter(Matrice autreMatrice)` retournant la matrice obtenue par la somme de la matrice actuelle avec une autre matrice.

Question 3 *:* Écrivez une méthode permettant de créer une structure de données triangulaire de la forme :

$$\begin{bmatrix} X & \varnothing & \varnothing \\ X & X & \varnothing \\ X & X & X \end{bmatrix}$$

Il est possible d'accéder à la case $(3, 3)$, mais pas à la case $(2, 3)$. Faites en sorte de ne pas allouer des cases inutilement.

Exercice 2 (Listes)

La classe utilisée pour manipuler les listes ici est la classe `java.util.ArrayList`.
Parmi les méthodes les plus utilisées d'une liste, on trouve :

- `boolean add(E elt)` ;
- `boolean contains(Object o)` ;
- `boolean remove(Object o)` ;

— `int size()`.

Il existe plusieurs classes implémentant l'interface `List` dans le JDK. Ces implémentations alternatives présentent des propriétés différentes. Il faut donc choisir la plus adaptée au problème visé.

Avant la version 1.2, on ne disposait que de la classe `Vector`, non générique. Depuis l'ajout de l'ensemble des collections dans la version 1.2, la classe `ArrayList` doit être utilisée de préférence, sauf dans quelques cas précis.

Comme toutes ces classes, `ArrayList` est une classe *générique*.

Question 1 *:* Comment peut-on ajouter les entiers 1, 2 et 3 dans une `ArrayList` ? Comment récupérer les valeurs entières ?

Question 2 *:* Implémentez vos classes permettant de manipuler des listes comme le fait `ArrayList`.

Question 3 *:* Implémentez une méthode `public void reverse()` qui renverse la liste. Ainsi, la liste (a, b, c, d) deviendrait (d, c, b, a).

Exercice 3 (Tableaux associatifs) La classe utilisée pour manipuler les tableaux associatifs (ou table de hashage), c'est-à-dire les tableaux indexés par un objet et non pas par un entier, est la classe `java.util.HashMap`. Une `HashMap` est implémentée au moyen d'un tableau de listes. Un couple $(clef, valeur)$ se trouve dans la liste de la case `clef.hashCode()` du tableau. Remarquons que la méthode `int hashCode()` est une méthode de la classe `Object`.

Comme pour les listes, plusieurs classes du JDK implémentent l'interface `Map`. Avant l'ajout dans 1.2 des interfaces de collection, la classe de référence était `Hashtable`, qui comme `Vector`, est désormais déconseillée au profis de `HashMap`.

Comme toutes ces classes, `HashMap` est une classe *générique*.

Parmi les fonctions les plus utilisées de la `HashMap`, on trouve :
— `V put(K key, V value)` ;
— `boolean containsKey(Object key)` et `boolean containsValue(Object val)` ;
— `V get(Object key)`.

Question 1 *:* Implémentez vos propres classes permettant de gérer des tableaux associatifs comme le fait `HashMap`. Vous pouvez utiliser la classe `ArrayList`.

Question 2 *:* Est-il possible de modifier votre classe de manière à avoir une méthode `void put(K key, int value)` ?

Pouvez-vous faire la différence entre un `int` et un `Integer` qui seraient ajoutés à votre tableau associatif ?

Est-il possible de modifier votre classe de manière à avoir aussi une méthode `int get(Object key)` lorsque la valeur entrée était un `int` ?

Exercice 4 (Comparaison de structures de données) Nous allons considérer un ensemble de 100 étudiants, ayant chacun un numéro d'étudiant et une note allant de 0 à 20. Nous allons comparer trois structures de données disponibles en Java : les tableaux, les listes (`ArrayList`) et les tables de hashage (`HashMap`)

Question 1 (Insertion des éléments) *:*

- Créez les classes dont vous aurez besoin. Ajoutez les constructeurs et accesseurs.
- Insérez 100 étudiants (générés aléatoirement) dans les structures de données.
- Cherchez dans la documentation de Java sur la classe `java.lang.System` la méthode permettant d'obtenir l'heure actuelle (à la milliseconde près).
- Mesurez le temps que prennent n insertions de 100 étudiants dans chacune de vos structures de données, avec n choisi de sorte que le temps d'exécution des n insertions de 100 étudiants prenne quelques dizaines de secondes.

Question 2 (Parcours des éléments) *:*

- Écrivez une méthode permettant de calculer la moyenne des étudiants, pour chacune des structures de données.
- Réorganisez vos classes de manière à ce que l'on puisse mesurer le temps de n insertions ou de n calcul de moyennes, de manière objet.

Question 3 (Recherche d'un élément) *:* Écrivez une méthode permettant de rechercher un étudiant n'existant pas, pour chacune des structures de données.

Question 4 (Récapitulatif) *:* Présenter la sortie du programme de manière à ce que la structure de données la plus efficace pour chacun des critère soit présentée clairement.