Programmation orientée objet
TP1 Rappels

L3 MIAGE

07 septembre 2023

1 Quelques rappels

Note : les exercices commencent à la section 2. Nous vous recommandons chaudement de lire cette section avant de commencer les exercices.

Éditeurs de texte et IDE

Vous êtes libre d'utiliser l'éditeur de texte ou l'environnement de développement intégré (IDE) de votre choix. Parmi tous les choix disponibles, nous retenons les trois suivants

- Visual studio code
- Eclipse
- IntelliJ

Ces trois outils sont disponibles sur les ordinateurs de l'Université Paris-Nanterre au deuxième étage.

Recommandation : utiliser de préférence un éditeur de texte plutôt qu'un IDE. Notons que les IDE sont lourds puisqu'ils contiennent, entre autres, de nombreux outils pour vous faciliter l'écriture de programme. Notamment, ils permettent de générer du code à votre place (méthodes, constructeurs, getters, setters, etc.). Ceci est très pratique pour un informaticien expérimenté mais ne facilite pas l'apprentissage de la programmation. Il est important que vous sachiez reproduire le code, sans aide, qu'une machine est capable de générer automatiquement. Vous aurez ensuite toute votre carrière pour utiliser ces puissants outils.

Visual studio code : si vous choisissez d'utiliser cet éditeur de texte, vous pouvez installer *the Extension Pack for Java* qui contient des outils, entre autres, des outils de coloration syntaxique, un debugger et vous donne la possibilité d'exécuter un programme sans passer par un terminal.

Programme principal (compilation et exécution)

Voici une classe Main qui pourra vous servir tout au long de ce TP pour exécuter les programmes que vous avez réalisés. Vous êtes libre d'en modifier son contenu.

```java
// Fichier Main.java
public class Main {
   public static void main(String[] args) {
      System.out.println("Voici un programme simple !");
   }
}
```

Pour compiler et exécuter votre programme en ligne de commande, vous pouvez utiliser les instructions suivantes dans le répertoire courant de votre fichier *Main.java*

```
javac Main.java
java Main
/* Resultat attendu:
Voici un programme simple !
```

Convention d'écriture (simple)

En java, il existe de nombreuses conventions d'écritures, mais nous vous demandons de respecter absolument les deux suivantes :

- suivre la syntaxe **camelCase** pour nommer les classes, interfaces, méthodes et variables. Si le nom est composé d'au moins deux mots, alors la première lettre du deuxième mot est une majuscule. Notons que la première lettre du nom d'une classe ou d'une interface doit toujours commencer par une majuscule tandis que la première lettre d'une méthode ou d'une variable doit toujours commencer par une minuscule.
- pour l'**indentation**, on utilise **4 espaces** (ou une tabulation équivalente à 4 espaces).

```java
// fichier BienIndenter.java
class BienIndenter {
   // Le premier caractere de chaque champ commence par une
      minuscule. A chaque nouveau mot, le premier caractere
      commence par une majuscule. On evite d'utiliser le caractere
      '_' ou '-' pour separer les mots.
   int variable;
   String varLong;
   String varTresTresLong;

   // Le constructeur a toujours le meme nom que la classe
   BienIndenter(int v, String vL, String vTTL) {
      this.variable = v;
      this.varLong = vL;
      this.varTresTresLong = vTTL;
   }

   int calculerQuelqueChose() {
      return variable * 42;
   }
}
```

Champs et méthodes de classe

En java, il est possible de définir des champs associés à une classe C et non à un objet du type de la classe C. Vous pouvez voir ces champs comme des données globales

qui n'existent qu'en un exemplaire. De la même manière, nous pouvons définir des méthodes de classes qui peuvent être appelées sans utiliser un objet de la classe. Voici quelques exemples :

```
class Main {
   // champs de classe ou champs statiques
   static in a = 55;
   static int b = 33;
   static int c = 44;

   // methode de classe ou methode statique
   static int somme(int x, int y) {
      return x + y;
   }

   // methode de classe ou methode statique
   static int produit(int x, int y) {
      return x * y;
   }

   // notons que la methode main est TOUJOURS une methode de classe
   public static void main(String[] args) {
      // Afin d'appeler une methode de classe f appartenant a la
         classe C, il faut utiliser l'instruction est: C.f(...)
      Main.somme(4181,999);

      // Afin d'acceder a un champ de classe x appartenant a la
         classe C, il faut utiliser l'instruction C.x
      Main.produit(Main.a, Main.b);
   }
}
```

Sortie

Voici quelques instructions contenant des méthodes d'affichage et des paramètres qui pourront vous être utile :

```
System.out.print("toto"); // Affiche le message toto sans retour a
   la ligne.

System.out.println("toto"); // Affiche le message toto avec un retour
a la ligne.

int rep = 42;
System.out.printf("La reponse est %d", rep); // La methode printf
   est similaire a la fonction printf en C. L'execution des
   intructions precedentes affichera le message "la reponse est 42".

System.out.print("1\n2\n3\n4"); // '\n' est un retour chariot c-a-d
   qu'il permet de retourner a la ligne. Ainsi, chaque chiffre sera
   affiche sur une ligne differente.
```

Les tableaux

Voici les instructions pour initialiser un tableau d'entiers et initialiser son contenu avec une boucle *for* :

```
int[] tab = new int[10];
for (int i = 0; i < 10; i++) {
   tab[i] = i + 1;
}
// Le tableau contient les valeurs 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Pour accéder à la taille d'un tableau, il existe l'instruction suivante ***tab.length***.

Chaîne de caractères

En java, vous pouvez représenter une chaîne de caractères par un tableau de caractères mais aussi en utilisant la classe String. Voici un exemple :

```
char[] msgTab = {'t', 'o', 't', 'o'};
String msgString = "toto";

// - pour un unique caractere on utilise les appostrophes ''
// - pour une sequence de caracteres (chaine de caracteres), on
   utilise les guillemets ""
```

Manipuler les entrées

En java, il est possible de récupérer les entrées saisies au clavier par l'intermédiaire de la classe *Scanner*. Voici quelques exemples :

```
import java.util.Scanner;

class Main {
   public static void main(String[] args) {
      Scanner entree = new Scanner(System.in);

      System.out.println("Saisir une chaine de caractere:");
      String s = entree.nextLine();
      System.out.println("Saisir un entier:");
      int a = entree.nextInt();
   }
}
```

Concaténation de chaîne de caractères

Pour la concaténation de chaîne de caractères en java, il est possible d'utiliser l'opérateur $+$. Voici quelques exemples :

```
System.out.println("Voici " + "mon" + " message !");

String s1 = "Ph'nglui mglw'nafh Cthulhu";
String s2 = "R'lyeh wgah'nagl fhtagn";
String s3 = s1 + " " + s2;
System.out.println(s3);
// Affichage attendu: Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl
   fhtagn
```

Si vous avez besoin de faire de nombreuses concaténations successives, par exemple au sein d'une boucle (*for*, *while*, ou autre), nous vous conseillons d'utiliser la classe *StringBuilder* qui est plus efficiente. Voici un exemple :

```
StringBuilder sb = new StringBuilder();
for (int i = 1; i < 10; i++) {
   sb.append(i);
}
System.out.println(sb);
// Affichage attendu: 123456789
```

2 Maîtriser les bases

Nous vous conseillons de récupérer le code de la sous-section **programme principal (compilation et exécution)** et de le placer dans un fichier *Main.java*. Vous pourrez utiliser ce code comme base pour les exercices 2.* suivants. Dans cette section, toutes les méthodes que vous devez implémenter sont statiques (voir les rappels un peu plus haut).

2.1 Affectation d'un tableau (avec saisie)

Écrire la méthode *saisieTableau* qui étant donné un entier n passé en paramètre, retourne un tableau d'entiers de n entiers contenant des valeurs saisies au clavier par l'utilisateur. Pour cela, utiliser la classe *Scanner* qui est présentée dans la section rappel.

```
static int[] saisieTableau(int n);
```

Appeler ensuite la méthode *saisieTableau* depuis votre méthode main et afficher la valeur contenue dans le tableau à la position 2.
Question : que se passe-t-il si l'utilisateur n'a saisie que deux entiers ? Pourquoi ?

2.2 Affectation d'un tableau (sans saisie)

Écrire la méthode *initTab* qui étant donné un entier n passé en paramètre, retourne un tableau d'entiers de n entiers contenant les valeurs 0, 10, 20, 30, 40, . . . , (n-1)*10.

```
static int[] initTab(int n);
```

Appeler ensuite la méthode *initTab* depuis votre méthode main et afficher la valeur contenue dans le tableau à la position 4.

2.3 Afficher les tous

Écrire la méthode *afficherTab* qui étant donné un tableau d'entiers *tab* passé en paramètre, affiche toutes les valeurs contenues dans ce tableau.

```
static void afficherTab(int[] tab);
```

Pour un tableau contenant les valeurs 0, 10, 20, l'affichage produit devra être le suivant :

```
/* Affichage attendu:
   tab[0] = 0
   tab[1] = 10
   tab[2] = 20
```

Enfin, appeler votre méthode *afficherTab* depuis la méthode *main* avec comme paramètre un tableau de 10 entiers qui a été généré par la méthode *initTab* du précédent exercice.

2.4 Entiers pairs

Écrire la méthode *sommePair* qui étant donné un tableau d'entiers *tab* passé en paramètre, retourne la somme des entiers pairs contenus dans ce tableau.

```
static int sommePair(int[] tab);
```

Appeler votre méthode *sommePair* depuis la méthode *main* avec comme paramètre le tableau suivant :

```
int tab[] = {8, 7, 12, 13, 15, 17, 20, 13, 5, 9};
```

Afficher ensuite la somme obtenue, celle-ci vaut normalement 40.

2.5 Échange dans un tableau

Écrire la méthode *echanger* qui étant donné un tableau d'entiers *tab* passé en paramètre, un entier i et un entier j, retourne le même tableau d'entiers où les valeurs aux positions i et j ont été échangées.

```
static int[] echanger(int[] tab, int i, int j);
```

Exemple :

```
int[] tab = {44, 5, 12, 22};
tab = echanger(tab, 0, 3);
afficherTab(tab);
/* Affichage attendu:
tab[0] = 22
tab[1] = 5
tab[2] = 12
tab[3] = 44
```

Note : si la position i ou j n'existe pas dans le tableau, alors afficher un message d'erreur en précisant la/les position(s) qui ne sont pas correctes. On retournera dans ce cas le tableau initial.

3 Classes et objets

3.1 Une première classe

Implémenter une classe *Film* qui doit permettre de créer des objets film contenant les informations suivantes :

- le titre (String)
- le réalisateur (String)
- la liste des acteurs principaux (String [ ])
- le pays de production (String)
- l'année de sortie (int)
- la durée en minutes (int)

Note : pour le moment, il n'est pas nécessaire de mettre d'accesseur (*public* ou *private*).

Nous souhaitons pouvoir créer un objet *Film* même si nous ne disposons pas de toutes les données. Il faudrait donc un constructeur pour chacune des situations suivantes :

- créer un objet *Film* à partir d'un titre et d'un réalisateur ;
- créer un objet *Film* à partir d'un titre, d'un réalisateur, de l'année de sortie et de la durée ;

— créer un objet *Film* à partir de toutes les données.

Question de cours : la classe *Film* dispose de trois constructeurs. À quelle concept de la programmation orientée objet cela fait-il référence ?

Bonus (prérequis : maîtriser les exceptions) : apporter les modifications nécessaires afin qu'aucun film ayant une date de sortie antérieure à 1891 ou supérieure à 2023 ne puisse être créé. Pour information, le premier film de l'histoire du cinéma serait *Dickson Greeting* et serait sortie en 1891.

3.2 Créer des objets

Depuis la méthode *Main*, initialiser des objets *Film* contenant les données suivantes :

Film 1 :

```
Titre: Dickson Greeting, realisateur: William Kennedy Laurie
    Dickson, annees de sortie: 1891, duree: 2 secondes
```

Film 2 :

```
Titre: Night is Short, Walk on Girl, realisateur: Masaaki Yuasa
```

Film 3 :

```
Titre: What s Eating Gilbert Grape, realisateur: Lasse Hallstrom,
acteurs principaux: Johnny Depp, Leonardo Dicaprio, Juliette Lewis,
pays: Etats-Unis, annee de sortie: 1993, duree: 118 minutes
```

Film 4 :

```
Titre: Sonatine melodie mortelle, realisateur: Takeshi Kitano, annee
    de sortie: 1995, duree: 94 minutes
```

Question : aurait-on pu initialiser les objets *Film* sans avoir implémenté un constructeur ?

3.3 Afficher des objets

Nous souhaitons pouvoir afficher le contenu de chaque objet *Film* dans la console comme suit :

```
System.out.print(filmGilbert); // ou filmGilbert est un objet Film
/* Affichage attendu:
Titre: What's Eating Gilbert Grape
Realisateur: Lasse Hallstrom,
Acteurs principaux: Johnny Depp, Leonardo Dicaprio, Juliette Lewis,
Pays: Etats-Unis,
Annee de sortie: 1993,
Duree: 118 minutes
*/
```

Pour cela, nous devons redéfinir le comportement de la méthode *toString* qui est définie dans la classe *Object*. Cette méthode retourne une chaîne de caractères et voici son prototype :

```
public String toString();
```

Une fois que la méthode *toString* a été redéfinie, afficher le contenu des quatre films créés dans l'exercice précédent.

3.4 Visibilité des champs et accesseurs (*getters*)

Par défaut, nous n'avons pas modifié la visibilité des champs de la classe *Film*. Afin de respecter le principe d'encapsulation des données, nous souhaitons restreindre l'accès de ces champs en dehors des objets *Film*. Pour cela, passer tous les champs de *Film* en privée avec le mot clef *private*.
Nous souhaitons ensuite ajouter des accesseurs (*getters*) qui permettent de lire les données à n'importe quel endroit du programme. Pour cela, implémenter des méthodes *getX* pour chaque champ *X* de la classe. Un accesseur doit retourner le champ *X*. Voici un exemple pour le champ titre :

```
// Accesseur de titre
public String getTitre() {
   return this.titre;
}
```

Ajouter ensuite un accesseur pour chaque champ.
Question : est-il nécessaire d'utiliser les accesseurs (*getters*) dans la méthode *toString* de la classe *Film* avec d'accéder aux champs ? Pourquoi ?

3.5 Mutateurs (*setters*)

Nous souhaitons pouvoir modifier l'année de sortie d'un film en cas d'erreur de saisie. Pour cela, nous vous proposons d'implémenter un mutateur (*setter*) du champ *sortie*. Un mutateur d'un champ *X* est une méthode nommer *setX* qui reçoit en argument un paramètre du même type que *X*. Son rôle est ensuite de modifier la valeur du champ *X* dans l'objet. Voici un exemple avec un mutateur du champ *titre* :

```
public void setTitre(String titre) {
   this.titre = titre;
}
```

Modifier votre mutateur *setSortie* afin qu'il ne soit pas possible de modifier l'année de sortie pour une année antérieur à 1891 ou supérieure à 2023.

Question : Dans le cas où le champ *sortie* est publique, aurait-on pu limiter sa modification ? Y a-t-il donc un intérêt de privatiser un champ et de mettre à disposition son mutateur (*setter*) ?

Ajouter ensuite un accesseur pour chaque champ de la classe *Film*.

3.6 Compteurs d'objets

Nous souhaitons disposer d'un compteur sur le nombre d'objets créé de la classe *Film*. Pour le moment, la classe *Film* et tous les objets *Film* doivent être les seuls à y avoir accès. Proposer une solution et implémenter là
Nous vous conseillons fortement de relire les rappels si vous n'avez pas d'idées.

3.7 Méthode de classe

Implémenter une méthode publique *afficherNbFilms* qui affiche le nombre d'objets *Film* actuellement instancié. Pour cela, utiliser une méthode de classe qui n'est pas liée aux objets *Film*. Voici son prototype :

```
public static void afficherNbFilms();
```

Question : depuis la méthode *afficherNbFilms*, peut-on accéder aux champs titre et duree ? Pourquoi ?

3.8 Tableau de films

Ajouter un champ de classe privé *tabFilms* qui contiendra, sous forme de tableau, tous les films instanciés depuis la classe *Film*. Mettez à jour votre code afin que tous les tableaux instanciés dans la classe *Film* soient contenus dans le tableau *tabFilms*.

3.9 Afficher tous les films

Implémenter une méthode publique *afficherFilms* qui affiche informations sur tous les films instanciés depuis la classe *Film*. Pour cela, utiliser une méthode de classe. Voici son prototype :

```
public static void afficherFilms();
```

3.10 Comparer deux films

Voici les instructions suivantes :

```
Film f1 = new Film("Un singe en hiver", "Henri Verneuil");
Film f2 = new Film("Un singe en hiver", "Henri Verneuil");

if (f1 == f2) {
   System.out.print("f1 est egale a f2 !");
} else {
   System.out.print("f1 n'est pas egale a f2 !");
}
```

Question : sans exécuter les instructions ci-dessus, donner le message qui sera afficher dans la console à l'exécution du programme.

Nous vous proposons maintenant de redéfinir la méthode de comparaison entre des objets de type *Film*. Pour cela, il est possible de redéfinir le comportement de la méthode *equals* qui est défini dans la classe *Object*. Par défaut, la méthode *equals* est défini comme suit :

```
public boolean equals(Object obj) {
   return this == obj;
}
```

Donc par défaut, elle ne fait que comparer la référence de l'objet courant (*this*) avec celle de l'objet passé en paramètre (*object*). Nous souhaitons modifier son comportement afin de comparer la valeur de chaque champ des deux objets et, s'ils sont tous égales, retourner vraie. Pour cela, nous vous donnons une portion du code nécessaire (avec quelques explications) et nous vous laissons compléter.

```
@Override
public boolean equals(Object o) {
```

```
    // Nous commencons par verifier que l'objet passe en argument est
       bien de type Film (cela est indispensable pour pouvoir les
       comparer).
    if (o instanceof Film) {
       Film filmAComparer = (Film) o; // Ici, comme l'objet O est de
          type Film, nous le convertissons dans ce type. Auparavent,
          il etait dans un type generique qui est le type Object.
          Cette technique, que nous verrons plus en details dans le
          cours, est appelee 'declassement' ou 'downcasting' en
          anglais.

       // A completer: comparer les films
       // ...

       // Rappel: avec le mot-clef this, on accede a l'objet courant
          de type Film
 }
```

Maintenant, il est possible d'utiliser la méthode *equals* pour comparer des objets comme suit :

```
Film f1 = new Film("Un singe en hiver", "Henri Verneuil");
Film f2 = new Film("Un singe en hiver", "Henri Verneuil");

if (f1.equals(f2)) {
   System.out.print("f1 est egale a f2 !");
} else {
   System.out.print("f1 n'est pas egale a f2 !");
}
```

4 Exercices bonus

4.1 Maximum

Écrire la méthode *maximum* qui étant donné un tableau d'entiers *tab*, retourne un tableau d'entiers de taille 2 contenant l'entier le plus grand de *tab* ainsi que sa position dans *tab*.

```
static int[] maximum(int[] tab);
```

Exemple :

```
int[] t = {1002, 33, 348, 92, 77, 5775, 20, 15};
int[] tgrand = maximum(t);
System.out.printf("Maximum: t[%d] = %d\n", tgrand[1], tgrand[0]);
/* Resultat attendu:
Maximum: t[5] = 5775
```

4.2 Compter

Écrire la méthode *compteSup* qui étant donné un tableau d'entiers *tab* et un entier naturel v, retourne le nombre d'entiers supérieurs à v dans *tab*.

```
static int compteSup(int[] tab, int v);
```

Exemple :

```
int[] t = {1002, 33, 348, 92, 77, 5775, 20, 15};
System.out.printf("Nombre d'entiers >= a %d: %d", 77, compteSup(t,
    77));
/* Resultat attendu:
Nombre d'entiers >= a 77: 5
```

4.3 Rechercher

Écrire la méthode *rechercher* qui étant donné un tableau d'entiers *tab* et un entier naturel v, retourne la première position de *tab* contenant v. Si *tab* ne contient pas v, alors la méthode retournera -1.

```
static int rechercher(int[] tab, int v);
```

Exemple :

```
int[] t = {1002, 33, 348, 92, 77, 5775, 20, 15};
int pos = rechercher(t, 20);
if (pos >= 0) {
    System.out.printf("20 a ete trouve en %dieme position du
        tableau\n", pos);
} else {
    System.out.printf("20 n'est pas dans le tableau");
}
/* Resultat attendu:
20 a ete trouve en 6ieme position du tableau
```

4.4 Tableau trié ?

Écrire la méthode *estTrie* qui étant donné un tableau d'entiers *tab*, retourne *true* si le *tab* est trié par ordre croissant des valeurs et *false* sinon.

```
static boolean estTrie(int[] tab);
```

Exemple :

```
int[] t = {1, 2, 3, 4, 5, 6, 7, 5};
if (estTrie(t)) {
    System.out.println("Le tableau est trie :) !");
} else {
    System.out.println("Le tableau n'est pas trie :( !");
}
/* Resultat attendu:
Le tableau n'est pas trie :( !
```

4.5 Concaténer deux tableaux

Écrire la méthode *fusion* qui étant donné un tableau d'entiers *tab1*, et un autre tableau d'entiers *tab2*, retourne un tableau contenant les éléments de *tab*1 et les éléments de *tab*2

```
static int[] fusion(int[] tab1, int[] tab2);
```

Exemple :

```
int[] t1 = {44, 38, 29, 16, 39};
int[] t2 = {22, 17, 18, -5, 16};
int[] t = fusion(t1, t2);
afficher(t, true);
/* Resultat attendu:
tab[0] = 44
tab[1] = 38
tab[2] = 29
tab[3] = 16
tab[4] = 39
tab[5] = 22
tab[6] = 17
tab[7] = 18
tab[8] = -5
tab[9] = 16
```