Université de Nice-Sophia Antipolis — POLYTECH
PeiP2 — 2017–2018

Examen de Algorithmique et Programmation JAVA

Durée : 2h
Aucun document autorisé
Mobiles interdits

Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.

1 Tableaux

Dans cet exercice, on se propose de créer un ticket de caisse listant des articles achetés par un client. Plus précisément, on trouvera sur un ticket : le nom des articles, leur prix, ainsi que le montant total des achats. Chaque ticket possédera un numéro qui sera défini automatiquement : le premier ticket créé aura le numéro 1, le suivant aura le numéro 2 et ainsi de suite. Écrivez le code de la classe `Ticket` qui possède :

1. les variables d'instance privées :
   — `numTicket` (de type **int**),
   — `articles` (de type tableau de `String`),
   — `prix` (de type tableau de **double**).
2. la variable de classe privée `TicketsCréés` (de type **int**, ayant pour valeur initiale `1`).
   Remarque : cette variable va compter le nombre total de tickets créés.
3. un constructeur qui prend en paramètre un entier `n` et
   — initialise les tableaux `articles` et `prix` à `n` éléments
   Remarque : les deux tableaux ont la même taille puisqu'à chaque article correspond un prix (le prix d'un article à l'indice j du tableau `articles` se trouve à l'indice j du tableau `prix`),
   — donne la valeur de la variable `TicketsCréés` à l'attribut `numTicket`,
   — incrémente la variable de classe `TicketsCréés`.
4. un accesseur `getNumTicket` qui renvoie le numéro du ticket courant.
5. un mutateur `setNumTicket` qui prend en paramètre un entier `n` et l'affecte au numéro du ticket courant.
6. une méthode `acheter` qui permet d'ajouter un article et son prix au ticket. Elle prend deux paramètres : `a` (de type `String`) et `p` (de type **double**) pour les ajouter respectivement aux tableaux `articles` et `prix`.
   Remarque : pour ajouter un article au tableau dans la première case vide, vous pouvez créer une autre variable d'instance qui compte le nombre de cases déjà occupées.
7. une méthode `calculerTotal` qui renvoie le montant total du ticket, c'est-à-dire la somme des prix des articles achetés. Elle ne possède aucun paramètre.
8. une méthode `comparer` qui compare le ticket courant avec un autre ticket passé en paramètre. Cette méthode renvoie le numéro du ticket dont le montant total est le plus élevé.
   Remarque : vous pouvez utiliser la méthode `calculerTotal` même si vous ne l'avez pas implémentée.
9. une méthode `toString` qui permet l'affichage du ticket au format suivant :


```
Numéro du ticket: ...
  - article_1 : prixArticle_1
  - article_2 : prixArticle_2
  ...
  - article_n : prixArticle_n
Montant total: ...
```

```
/**
 * La classe Ticket représente un ticket de caisse caractérisé par un
 * numéro ainsi que par les articles achetes et leur prix
 */
public class Ticket {
    //Variables d'instance
    //numéro du ticket (1 pour le premier créé, 2 pour le second etc.)
    private int numTicket;
    //articles achetés (tableau de String)
    private String articles[];
    //prix des articles achetés (tableau de double -
    //de même taille que le tableau précédent)
    private double prix[];
    //l'indice i permet de savoir où on en est dans le remplissage
    //des tableaux (nombre d'articles dans le ticket courant)
    private int i = 0;

    //Variable de classe
    //nombres de tickets de caisse générés
    private static int nbTickets = 1;

    //Constructeur
    public Ticket(int n){
        this.articles = new String[n];
        this.prix = new double[n];
        this.numTicket = nbTickets; //le numéro du ticket dépend
                                    //du nombre de tickets generes
        nbTickets++; //un nouveau ticket vient d'être genere
                     //(le prochain aura le numéro suivant)
    }

    //Accesseur
    public int getNumTicket() {
        return this.numTicket;
    }

    //Mutateur
    public void setNumTicket(int n){
        this.numTicket = n;
    }

    //Méthode acheter (permet d'ajouter un article au ticket ainsi que son prix
    //- à l'indice i des 2 tableaux)
    public void acheter(String a, double p) {
        if (this.i >= this.articles.length)
            //si le ticket est plein
            System.err.println("Impossible d'ajouter un article, le ticket est plein !");
        else {
            this.articles[this.i] = a;
            this.prix[this.i] = p;
```

```
            this.i ++; //un article a été ajouté à l'indice i donc
                       //le prochain devra etre ajoute a l'indice i+1
        }
    }

    //Méthode calculerTotal (calcul le montant total du ticket)
    public double calculerTotal() {
        double total = 0;
        for (int j = 0 ; j < this.i ; j ++)
            total += this.prix[j];
        return total;
    }

    //Méthode comparer (compare 2 tickets de caisse et renvoie le
    //numéro du ticket dont le prix total est le plus élevé)
    public int comparer(Ticket t){
        return this.calculerTotal()-t.calculerTotal() > 0 ?
            this.numTicket : t.numTicket;
    }

    //méthode toString
    public String toString(){
        String ticket = "\nNuméro de ticket: " + this.numTicket + "\n";
        for (int j = 0 ; j<this.i ; j++)
            ticket = ticket + "- " + this.articles[j] + " : " + this.prix[j] + "\n";
        return ticket + "Total: " + this.calculerTotal();
    }

}
```

2 Fichiers

Un fichier est composé d'une suite de trames. Le nombre de trames est quelconque et peut-être égal à 0. Chaque trame est formée d'un entier (type **int** $\geqslant 0$), suivi de réels (type **double**). Le nombre de réels dans la trame est égal à la valeur de l'entier, et on appellera longueur d'une trame ce nombre.

▶ 1. Écrivez la fonction `nbTrames` qui renvoie le nombre de trames de longueur lg contenues dans un fichier de trames. Le nom du fichier de trames, et la longueur des trames recherchées sont donnés en paramètre. On considérera que les trames sont correctement formées. Vous pourrez utilisez les classes `DataInputStream`, `FileInputStream`, `EOFException`, `FileNotFoundException` et `IOException`. L'en-tête de la méthode est le suivant :

```
static int nbTrames(String f, int lg)
```

```
/**
  * Antécédent : f nom du fichier de trames
  *              lg>=0 longueur des trames recherchées
  *
  * Rôle : renvoie le nombre de trames de longueur lg
  *        contenues dans le fichier
  */
static int nbTrames(String f, int lg) {
  int n = 0;
  try (DataInputStream dis = new DataInputStream(new FileInputStream(f)))
```



```
  {
      while (true) {
         //lire l'entier
         int x = dis.readInt();
         if (x==lg)
            //on a une nouvelle trame de longueur lg
            n++;
         //lire les x réels
         for (int i=0; i<x; i++) dis.readDouble();
      }
  }
  catch(EOFException e) {}
  catch(FileNotFoundException e) { /* traiter l'erreur */ }
  catch(IOException e) { /*  traiter l'erreur */ }
  finally { return n; }
}
```

3 Héritage

On désire modéliser, ici en Java, une banque, qui est opérée par des employés, dont un directeur. Un directeur sera aussi un employé, et l'un comme l'autre seront des personnes (par héritage, mais l'application ne manipulera jamais des objets de la classe `Personne` directement). Une banque sera au moins constituée du directeur, donc d'un employé.

Le directeur a un salaire calculé différemment de celui des autres employés. En fonction du nombre d'années d'expérience, et d'un salaire de base, le directeur perçoit mensuellement 5% en plus de ce salaire de base par année d'expérience, le nombre d'années étant initialisé au moment de son embauche.

Par ailleurs, chaque fin d'année, la paie du mois de décembre est majorée d'une prime qui pour les employés correspond à 50% du salaire. La méthode ci-dessous de la classe `Banque` permet d'établir ces fiches de paie.

```
public String établirFichesPaie(String date) {
    String fiches="";
    for (int i=0; i<this.nb_employés; i++) {
        fiches+="Fiche de paie en date du "+date + "\t" +
            employés[i].initialiseFiche() + "\t";

        if (date.startsWith("Décembre"))
            fiches+=employés[i].getSalaire()*1.5; //prime de fin d'année
        else
            fiches+=employés[i].getSalaire();
        fiches+="\n";
    }
    return fiches;
}
```

Il vous sera demandé de fournir les classes nécessaires à l'exécution d'une méthode `main` telle que donnée ci-dessous :

```
public static void main(String[] s){
   //directrice, salaire base 3000
   //et 4 années expérience; salaire de base augmenté de 5% par année d'expérience
   Directeur d1=new Directeur("Mme Baude", 3000,4);
   Employé e1=new Employé("Léa ", 1500);
   Employé e2=new Employé("Franck",2000);
   Banque b=new Banque(d1);
```

```
    b.ajouterEmployé(e1);
    if (!b.ajouterEmployé(e2))
         System.err.println("Il n'y a pas possibilité d'avoir un employé supplémentaire");

     System.out.println(b.établirFichesPaie("Novembre 2017"));
     System.out.println(b.établirFichesPaie("Décembre 2017"));
     System.out.println(b.établirFichesPaie("Janvier 2018"));
}
```

et dont le résultat affiché sur la sortie standard est le suivant **pour l'instant** :

```
Fiche de paie en date du Novembre 2017 Employé de nom Mme Baude  Fonction de directeur    3600.0
Fiche de paie en date du Novembre 2017 Employé de nom Léa  1500.0
Fiche de paie en date du Novembre 2017 Employé de nom Franck 2000.0

Fiche de paie en date du Décembre 2017 Employé de nom Mme Baude  Fonction de directeur    5400.0
Fiche de paie en date du Décembre 2017 Employé de nom Léa  2250.0
Fiche de paie en date du Décembre 2017 Employé de nom Franck 3000.0

Fiche de paie en date du Janvier 2018 Employé de nom Mme Baude  Fonction de directeur    3600.0
Fiche de paie en date du Janvier 2018 Employé de nom Léa  1500.0
Fiche de paie en date du Janvier 2018 Employé de nom Franck 2000.0
```

Nous allons devoir améliorer le calcul du salaire du directeur : il doit être pris en compte que son salaire du mois de janvier suivant doit mécaniquement être augmenté considérant le fait qu'il a une année d'expérience en plus. Pour cela, on devra disposer d'une méthode dans la classe `Directeur`, utilisée ainsi dans `main` :

```
//au moment de la paie de janvier, le directeur est crédité
//d'une année d'exp. de plus
d1.incrementeAnnéesExpérience();
System.out.println(b.établirFichesPaie("Janvier 2018"));
```

Et donc, le résultat final sur la sortie standard devra être celui-ci :

```
Fiche de paie en date du Novembre 2017 Employé de nom Mme Baude  Fonction de directeur    3600.0
Fiche de paie en date du Novembre 2017 Employé de nom Léa  1500.0
Fiche de paie en date du Novembre 2017 Employé de nom Franck 2000.0

Fiche de paie en date du Décembre 2017 Employé de nom Mme Baude  Fonction de directeur    3600.0
Fiche de paie en date du Décembre 2017 Employé de nom Léa  2250.0
Fiche de paie en date du Décembre 2017 Employé de nom Franck 3000.0

Fiche de paie en date du Janvier 2018 Employé de nom Mme Baude  Fonction de directeur    3750.0
Fiche de paie en date du Janvier 2018 Employé de nom Léa  1500.0
Fiche de paie en date du Janvier 2018 Employé de nom Franck 2000.0
```

Question 1

Écrivez les en-têtes des classes `Personne`, `Employé` et `Directeur` ainsi que les attributs nécessaires.

En particulier, factorisez le nom au plus haut niveau et munissez `Personne` du constructeur approprié pour initialiser ce nom. Pour autant, on ne doit pas permettre l'instantiation d'une `Personne`.

Définissez un attribut `salaire`, au niveau de la classe `Employé`, considéré comme étant le salaire de base lorsque on est dans le cas particulier du `Directeur`, et du salaire hors prime quand on n'est pas directeur. Que faut-il en plus dans la classe `Directeur` ?


```
public abstract class Personne {
    protected String nom;
    protected Personne(String nom) {
        this.nom=nom;
    }
}

public class Employé extends Personne {
    protected double  salaire;
}

public class Directeur extends Employé {
    protected int nb_années_exp; //expérience passée comptée en années
}
```

.....

Question 2

Munissez les classes `Directeur` et `Employé` d'un constructeur qui permet de réutiliser le maximum de code des classes mères.

Plus précisément, le `Directeur` sera instancié par le constructeur ci-dessous qui devra réutiliser le constructeur de `Employé(String nom, double salaire)` pour initialiser son salaire à l'aide uniquement du salaire de base :

```
Directeur(String nom, double salaireBase, int nb_années_exp_initial)
```

```
public class Directeur extends Employé {
    protected int nb_années_exp; //expérience passée comptée en années
    Directeur(String nom, double salaireBase, int exp) {
        super(nom,salaireBase);
        this.nb_années_exp=exp;
    }

}

public class Employé extends Personne {
    protected double  salaire;
    public Employé(String nom, double salaire) {
        super(nom);
        this.salaire=salaire;
    }
}
```

.....

Question 3

Munissez la classe `Employé` d'une méthode publique `getSalaire()`. Redéfinissez cette méthode dans la classe `Directeur` afin de prendre en compte le nombre d'années d'expérience (Rappel : 5% de plus du salaire de base par année d'expérience)

---

dans la classe `Employé` :

```
public double getSalaire() {
    return salaire;
}
```

et dans la classe `Directeur`

```
public double getSalaire(){
  return salaire+(salaire*0.05)*nb_années_exp;
}
```

.....

Question 4

Utilisez cet exemple du calcul du salaire qui diffère selon qu'on est Directeur ou non, pour expliquer le principe de liaison dynamique et de polymorphisme, qui est utilisé dans la méthode `établirFichePaie`.

Question 5

Écrivez une classe publique `Banque`, avec un constructeur qui doit permettre l'exécution de la méthode `main` donnée précédemment, c'est-à-dire, qui doit permettre de créer une banque, en passant en paramètre un directeur qui sera mémorisé dans une variable d'instance de type `Directeur`. Par ailleurs, toute instance de `Banque` ne devra jamais avoir plus d'un nombre fixe et identique d'employés pour toute instance. Définissez ce nombre comme une constante de classe ayant la valeur 10. Les employés, le directeur y compris, seront mémorisés dans un tableau d'`Employé`.

```
private static final int NB_MAX_EMPL=10; //incluant le directeur
private int nb_employés;
public Banque(Directeur d){
    employés=new Employé[NB_MAX_EMPL];
    this.d=d;
    employés[0]=d;
    nb_employés=1;
}
```

.....

Question 6

Ajoutez ce qu'il faut au(x) classe(s) pour que l'utilisation de la méthode nommée `initialiserFiche()` dans la méthode `établirFichePaie` donne le résultat voulu écrit sur la sortie standard. Maximimisez la factorisation du code (réutilisez au maximum le code déjà écrit présent dans les classes ancêtres).

Dans `Employé` :

```
public String initialiseFiche() {
    return "Employé␣de␣nom␣"+ nom;
}
```

Dans `Directeur` :

```
public String initialiseFiche(){
  return  super.initialiseFiche()+"\t␣Fonction␣de␣directeur␣\t␣";
}
```

.....