

**Examen de rattrapage  
du Vendredi 2er février 2007  
Durée : 2h  
Le barème sur 40 est indicatif**

*Remarque : Les deux problèmes sont totalement indépendants.*

**PROBLÈME 1 (20 POINTS)**

On veut modéliser une Agence de voyage qui affrète ses propres vols. Pour cela on va définir quatre classes, Agence, Vol, Client et Voyage.

Chaque Agence est caractérisée par un numéro, elle a à sa disposition un ensemble de Vols et connaît un ensemble de Clients. Elle peut :

- Ajouter un Vol
- Ajouter un Client .
- Rechercher un Vol connaissant la date du vol, la ville de départ, la ville d'arrivée;
- Réserver une place sur un Vol
- Créer un Voyage.

Un Vol est caractérisé par sa capacité (nombre de passagers), sa ville de départ, sa ville d'arrivée, la date du vol (on utilisera une chaîne de caractères pour représenter cette date), le nombre de places libres et les passagers qui ont déjà réservé. Un Vol peut ajouter un passager et faire la liste des passagers.

Un Client est modélisée par son nom et l'ensemble des voyages qu'il a acheté. Il peut lister ses différents voyages.

Un Voyage est l'association entre un Client et un Vol, il peut s'afficher.

*N.B. On ne vous demandera pas d'écrire les classes dans leur ensemble, certaines méthodes seront utilisables mais ne devront pas être écrites. Si vous avez besoin d'accesseurs, vous pouvez les créer en disant dans quelle classe ils sont et ce qu'ils font.*

**Question 1 (1 point)**

Classe Client

Définir les attributs de cette classe, un constructeur avec un paramètre de type chaîne de caractères et un constructeur sans paramètre qui initialise le nom à "xxx".

Réponse :

```
public class Client {
    private final int MAX_VOY=10;
    private String nom;
    private Voyage[] voyages;
    private int nbVoyages;

    /** Creates a new instance of Client */
    public Client() {
        this ("nemo");
    }
    public Client(String s) {
        nom=s;
        voyages = new Voyage[MAX_VOY];
    }
}
```

```
nbVoyages=0;
}
```

## Question 2 (5 points)

Classe Vol

a) Définir les attributs de la classe Vol et son ou ses constructeurs.

2 points

```
public class Vol {
    private String depart;
    private String arrivee;
    private String date;
    private int capa;
    private int nbLibre;
    private Client[] passagers;

    /** Creates a new instance of Vol */
    public Vol() {
        this(300, "170606", "Paris", "Berlin");
    }
    public Vol (int t, String date, String depart, String arrivee){
        capa=t;
        this.date=date;
        this.depart=depart;
        this.arrivee=arrivee;
        nbLibre=capa;
        passagers=new Client[capa];
    }
}
```

b) Définir la méthode listePassagers qui rend une chaîne de caractères contenant les noms des différents passagers.

1 point

Corrigé :

```
public String listePassagers (){
    String s="";
    for(int i=0; i<capa-nbLibre; i++){
        s=s+passagers[i].getNom()+" ";
    }
    s=s+'\n';
    return s;
}
```

c) Redéfinir la méthode toString qui rend une chaîne de caractères contenant la date du vol, la ville de départ et la ville d'arrivée.

1 point

Corrigé :

```
public String toString(){
    return "Le "+date +"départ de "+depart+ " arrive à "+arrivee +"\n";
}
```

d) Définir une méthode caracVol qui retourne une chaîne de caractères donnant les caractéristiques du Vol et la liste des passagers.

1 point

Corrigé :

```
public String caracVol(){
    return ( "Vol de capacite "+capa +" places, il reste "+nbLibre+
    " places \n "+toString()+listePassagers());
}
```

## Question 3 (3 points)

Classe Agence :

a) Définir les attributs de cette classe et un constructeur sans paramètres, les nombres maximums de Vols et de Clients seront définis comme des constantes.

```
2 points
public class Agence {
    private static final int NB_MAX_Vol=10;
    private static final int NB_MAX_CLIENTS=5;
    private static int genAgence=1;
    private int numAgence;
    private int nbVols;
    private int nbClients;
    private Vol [] flotte;
    private Client [] clients;
    /** Creates a new instance of Agence */
    public Agence() {
        numAgence=genAgence++;
        flotte = new Vol [NB_MAX_Vol];
        nbVols=0;
        clients=new Client [NB_MAX_CLIENTS];
        nbClients=0;
    }
}
```

b) On a à notre disposition les méthodes d'en-tête : `String lesVols ()` et `String lesClients ()` qui rendent chacune une chaîne de caractères correspondant respectivement aux vols et aux clients de l'agence. **ELLES NE SONT PAS À ÉCRIRE**. Redéfinir la méthode `toString` qui rend une chaîne de caractères contenant le numéro de l'agence et la liste de ses avions et de ses clients.

```
1 point
Corrigé :
public String toString(){
String s= "Agence numéro "+ numAgence+" dont les Vols sont :\n"
+lesVols()+" et les clients :\n"+ lesClients();
    return s;
}
```

### Question 4 (3 points)

Classe Voyage

a) Définir les attributs de cette classe, et son ou ses constructeurs

```
1 point
Corrigé :
public class Voyage {
    Vol vol;
    Client passager;
    public Voyage(Vol a, Client p) {
        vol=a;
        passager=p;
    }
}
```

b) Redéfinir la méthode `toString` qui définit le voyage. La chaîne de caractères résultat contiendra seulement le nom du passager, la date du vol, la ville de départ et celle d'arrivée.

```
1 point
Corrigé :
public String toString(){
    return "Passager \t"+ passager.getNom()+"\n Vol "+vol.toString();
}
```

c) Ecrire la méthode `emettreBillet` qui affiche le billet

```
1 point
Corrigé :
public void emettreBillet(){
    System.out.println(toString());
}
```

### Question 5 (2 points)

La classe Agence est enrichie par les méthodes ajouterVol et ajouterClient qui permettent respectivement d'ajouter un Avion ou un Client à l'ensemble correspondant de la classe.

Ecrire la méthode ajouterVol qui a pour en-tête : void ajouterVol (Vol).

**NE PAS ÉCRIRE LA MÉTHODE ajouterClient!!!**

```
2 points
Corrigé :
public void ajouterVol(Vol v){
    if (nbVols== flotte.length)
System.out.println("Ce vol "+v+" n'a pu être ajouté, trop de vols");
    else {
flotte[nbVols]=v;
        nbVols++;
    }
}
```

### Question 6 (3 points)

Définir la méthode rechercherVol de la classe Agence qui à partir d'une date, d'une ville de départ et d'une ville d'arrivée, rend un Vol s'il existe, null sinon.

```
3 points
Corrigé :
public Vol rechercherVol (String date, String depart, String arrivee){
    int i =0;
    while (i<nbVols &&
        (date != flotte[i].getDate() ||
depart != flotte[i].getDepart() ||
arrivee!=flotte[i].getArrivee())) {
        i++;
    }
    if (i==nbVols) return null;
    else return flotte[i];
}
```

### Question 7 (3 points)

On dispose dans la classe Vol de la méthode boolean ajoutPassager(Client). Elle ajoute un passager à un Vol, et renvoie vrai si c'est possible, faux sinon (plus de place dans l'avion).

On dispose dans la classe Client de la méthode boolean ajoutVoyage (Voyage) qui ajoute un Voyage dans la liste des voyages d'un Client et renvoie vrai si on a pu faire l'ajout et faux sinon.

**VOUS NE DEVEZ PAS ÉCRIRE CES DEUX MÉTHODES, VOUS AUREZ SEULEMENT À LES UTILISER.**

Définir la méthode reserverVol de la classe Agence, qui va pour un Client donné, une date, une ville de départ et une ville d'arrivée données, rechercher si il existe un Vol à la bonne date, partant de la bonne ville, pour la bonne destination. Si ce Vol existe, la méthode va, si cela est possible, créer le Voyage correspondant (associer Client et Vol), ajouter le client à la liste des passagers du Vol, et ajouter le Voyage à la liste des voyages du client. La méthode reserverVol rend le booléen vrai si la réservation complète a pu être faite, et faux sinon.

```
3 points
Corrigé :
public boolean reserver(String date, String depart, String arrivee, Client
p){
    Vol v= rechercher(date, depart, arrivee);
    if (v==null) return false;
    else {
        Voyage b=new Voyage(v,p);
        else {
            boolean b1 = v.ajoutPassager(p);
            boolean b2 = p.ajoutVoyage(b);
        }
    }
}
```

```

        if (b1 && b2)    b.emettreBillet();
            return b1 && b2;
        }
    }
}

```

## PROBLÈME 2 (20 POINTS)

Dans un pays imaginaire, le mandat du président est arrivé à échéance et de nouvelles élections s'organisent. On veut écrire un programme Java qui permette de gérer les bureaux de vote et le vote des électeurs.

### Question 1 (4 points)

Une personne a un nom et habite dans une ville. Un candidat est une personne qui en plus appartient à un parti (de type chaîne de caractères).

On suppose qu'on dispose dans la classe `Personne` des deux méthodes suivantes (**DONT ON N'ÉCRIRA PAS LE CODE**) :

`int alea(int a, int b)` retourne un entier compris entre `a` et `b` inclus.

`String chaineAlea()` retourne une chaîne de caractères aléatoire.

**Dans toute la suite on supposera pour simplifier que `chaineAlea()` retourne aussi bien un nom, qu'une ville, qu'un parti, etc...**

a) Ces deux méthodes doivent-elles être plutôt déclarées d'instance ou de classe? Justifiez.

1 point

Réponse : Elles pourraient à la limite être déclarées d'instance (mauvais) mais il est plus logique de les déclarer static car elle ne dépendent pas d'un objet particulier.

b) Ecrire la classe `Personne` avec un constructeur par défaut et un constructeur avec paramètres, ainsi que la méthode `toString()` qui rend la chaîne "`<nom> de <ville>`", par exemple "Paul de Lyon".

```

1 point
Réponse :
abstract class Personne {
    private String nom, ville;

    Personne(String nom, String ville) {
        this.nom=nom;
        this.ville=ville;
    }
    Personne() {
        this(chaineAlea(),chaineAlea());
    }
    public String toString() {
        return nom+" de "+ville+"\n";
    }
}

```

c) ) Ecrire la classe `Candidat` avec un constructeur par défaut et un constructeur avec paramètres, ainsi que la méthode `toString()` qui rend la chaîne "<nom> de <ville> du parti <parti>", par exemple "Paul de Lyon du parti des Mauves".

2 points

```
class Candidat extends Personne{
    private String parti;

    Candidat(String n,String v,String p) {
        super(n,v); parti=p;
    }

    Candidat() {
        this(chaineAlea(),chaineAlea(),chaineAlea());
    }

    public String toString() {
        return super.toString() +" du parti " + parti + "\n";
    }
}
```

### Question 2 (3 points)

Un électeur est une personne qui a en plus un numéro d'électeur (attribué dans l'ordre à sa création), un indicateur qui exprime s'il a voté ou non, et le numéro de son bureau de vote (de 1 à 30).

- a) Ecrire la classe `Electeur` (attributs, les constructeurs, la méthode `toString()`). La méthode `toString()` rend la chaîne "Electeur no <noElecteur>", par exemple "Electeur no 15".

2 points

```
class Electeur extends Personne {
    private static int nbElecteurs=0;
    private int noElecteur;
    private boolean aVote=false;
    private int bureau=alea(1,30);
    Electeur(String n,String v, int b){
        super(n,v);bureau=b;
        nbElecteurs++; noElecteur=nbElecteurs;
    }
    Electeur() {
        super();
        nbElecteurs++; noElecteur=nbElecteurs;
    }
    public String toString() {
        return "Electeur no "+noElecteur+" ";
    }
}
```

- b) Écrire la méthode `voter()` qui fait fait voter cet électeur et affiche (par exemple pour l'électeur numéro 15) : "Electeur no 15 a voté".

1 point

```
void voter() {
    aVote=true;
    System.out.print("\n"+this +" a vote ");
}
```

### Question 3 (6 points)

Un bureau de vote a un numéro, un nombre d'électeurs inscrits (au maximum 2000), la liste des électeurs (qu'on gèrera au choix par tableau ou Vector).

- a) Définir une classe `ExceptionBureauPlein` (qu'on va lancer dans la question suivante).

```
1 point
class ExceptionBureauPlein extends Exception {
    ExceptionBureauPlein(String s) {super(s);}
}
```

- b) Ecrire la classe `BureauVote` - **SANS LES CONSTRUCTEURS NI LA MÉTHODE `TOSTRING`** - mais avec une méthode `inscrire(Electeur e)` qui ajoute un électeur à la liste des inscrits à ce bureau de vote, et lève l'exception définie en a) si la liste est pleine.

```
3 points
class BureauVote {
    private static final int max=2000;
    private int bureau;
    private int nbInscrits=0;
    private Electeur[] listeElecteurs=new Electeur[max];
}
```

Ne pas tenir compte dans la note que cette initialisation ait été mise ou non car cela peut être fait dans les constructeurs qu'on leur dit de ne pas écrire! De plus cela a déjà été demandé dans le 1<sup>er</sup> problème.

```
static int getMax(){return max;}
void inscrire(Electeur e) throws ExceptionBureauPlein {
    if (nbInscrits==max)
        throw new ExceptionBureauPlein("Bureau plein");
    listeElecteurs[nbInscrits]=e;
    nbInscrits++;
}
```

- c) Y ajouter la méthode `faireVoter()`, qui pour chaque électeur de la liste électorale de ce bureau, le fait voter avec une probabilité de 70% (c'est à dire qu'il y a en fait 3 chances sur 10 pour qu'il ne vote pas - utiliser la méthode `Math.random()`).

```
2 points
void faireVoter() {
    for (int i=0;i<nbInscrits;i++) {
        if (Math.random() <0.7) listeElecteurs[i].voter();
    }
}
```

#### Question 4 (2 points)

Donnez les instructions du main qui crée un bureau de vote, y inscrit 2001 électeurs, et les fait voter. (utilisez les constructeurs dont vous avez besoin **EN SUPPOSANT QU'ILS EXISTENT** si ce n'est pas le cas). On pensera à pouvoir attraper l'exception qui ne manquera pas d'être levée à la 2001<sup>ème</sup> inscription (cf Question 3).

```
2 points
BureauVote b=new BureauVote(1);
//création de max+1 électeurs et inscription de ceux-ci dans le bureau:
try {
    for (int i=0;i<BureauVote.max+1;i++) {
        Electeur e=new Electeur();
        b.inscrire(e);
    }
}
catch(ExceptionBureauPlein e) {
    System.out.println(e.getMessage());
}
System.out.println("le bureau vote : ");
b.faireVoter();
```

#### Question 5 (4 points)

On veut maintenant spécialiser la classe `Electeur` en considérant les électeurs résidant sur une longue période à l'étranger (qui votent par correspondance), ainsi que les électeurs handicapés ne pouvant se déplacer seuls pour aller voter (qui doivent être accompagnés jusqu'au bureau de vote).

- a) Enrichir le schéma hiérarchique des classes en y ajoutant les classes `ElecteurAletranger` et `ElecteurHandicape` et les attributs `pays` (de type `String`), `modeDeVote` (de valeur "normal", "par correspondance", ou "handicapé") et `degreHandicap` (pourcentage de type `int`) en face de la classe où il est le plus logique de les mettre.

**ON N'ÉCRIRA PAS LE CODE DE CES CLASSES!!!!!!!.**

```
1 point
pays dans ElecteurAletranger extends Electeur
modeDeVote dans Electeur
degreHandicap dans Handicape extends Electeur
```

On s'intéresse maintenant dans le main aux instructions suivantes (on suppose que les constructeurs utilisés ici existent):

```
//electeur du bureau 1, résidant en Allemagne, autres attributs aléatoires:
    Electeur e1=new ElecteurAletranger(1, "Allemagne");
//electeur du bureau 1, handicapé a 50%,autres attributs aléatoires :
    Electeur e2=new ElecteurHandicape(1, 50);
    e1.voter(); e2.voter();
```

- b) En supposant que `e1` et `e2` ont respectivement les numéros d'électeur 12 et 13, quel affichage provoquent ces instructions?

```
1 point
Electeur no 12 a vote
Electeur no 13 a vote
```

- c) Expliquez brièvement comment faire pour que le vote de `e1` affiche :

```
"Electeur no 12 a voté de l'étranger" et que le vote de e2 affiche :
"Electeur no 13 a voté accompagné"
```

1 point

Il suffit de redéfinir la méthode `voter()` dans les classes `ElecteurAletranger` et `ElecteurHandicape`. Ces deux méthodes masqueront la méthode `voter()` de la superclasse

- d) Faites-le (écrivez le code correspondant).

```
1 point
Dans ElecteurAletranger :
void voter() {
super.voter();
    System.out.println("de l'etranger");
}
Dans ElecteurHandicape :
void voter() {
    super.voter();
    System.out.println(" accompagné");
}
```

### Question 6 (1 point)

Que répondriez-vous à la question suivante :

"Mais un candidat est aussi un électeur, comment gérer cette double appartenance, l'héritage multiple n'existant pas en java?"

Vous pouvez proposer plusieurs solutions.

1 point

La solution adoptée dans l'énoncé est de dupliquer le candidat (une occurrence dans Candidat et une occurrence dans Electeur). Mais on peut aussi créer une troisième sous-classe de Personne : ElecteurCandidat. Dans ce cas les trois sous-classes sont disjointes. On accepte que les étudiants ne donnent qu'une seule solution.