

Examen terminal écrit

Durée : **2h00** - Tous les documents *personnels*, manuscrits ou imprimés, sont autorisés.

Vous devez composer sur le document joint à ce sujet.

Question de cours

On considère les deux classes publiques suivantes :

```
public class Redef {
    public static void main(String[] a) {
        Y o1= new Y();
        System.out.println(o1.toString());
        System.out.println(o1.toStr());

        Object o2= new Object();
        System.out.println(o2.toString());
        o2= new Y();
        System.out.println(o2.toString());
    }
}

public class Y {
    public String toString() { return "dans Y"; }
    public String toStr()    { return super.toString(); }
}
```

- q1** Qu'est-ce qui s'affiche à l'écran ? Mettez entre < > ce que vous ne pouvez indiquer avec précision car dépendant d'une exécution particulière.
- q2** Justifiez votre réponse.
- q3** Commentez très précisément les deux dernières instructions de *Redef*?

Présentation du jeu simplifié

Le jeu dit de Monopoly, comporte quarante *cases* successives, que les joueurs parcourent en sens croissant, et qui sont organisées cycliquement (lorsqu'on atteint la dernière case, on continue sur la première). Toutes les cases ont un nom. On compte douze cases *de passage*, déclenchant des actions diverses (déplacement, paiement, recevoir, prison), et vingt-huit cases dites *terrains*, pouvant donner lieu à l'achat (ou même la vente) du terrain, ou à un paiement/encaissement de loyer. Ce dernier type de terrain achetable, se subdivise lui-même en vingt-deux *propriétés*, quatre *gares* et deux *compagnies*, la différence provenant, en ce qui nous concerne ici, du mode de calcul du montant de loyer à payer, ainsi que cela sera expliqué un peu plus loin.

Les joueurs progressent sur le plateau du nombre de cases correspondant aux points indiqués par les deux dés qu'ils ont jeté. Si le joueur s'arrête sur une case de passage, il exécute l'action correspondante. Si le

joueur s'arrête sur un terrain, il peut l'acheter s'il n'a pas de propriétaire, il y demeure si c'est lui le propriétaire, et il paie un loyer au propriétaire dans le troisième cas.

On se restreint considérablement par rapport au jeu réel, car on ne s'intéresse grosso modo qu'au problème d'achat de terrain et de règlement des loyers.

Les vingt-huit cases *terrain* sont divisées en dix « groupes », huit de *propriétés* (caractérisés par huit couleurs différentes dans le jeu réel), un pour les *gares*, et le dernier pour les *compagnies*.

Les règles de calcul du montant du loyer que nous retiendrons sont les suivantes :

Propriété	2 * loyer de base	si le joueur possède toutes les propriétés du groupe
	loyer de base	dans l'autre cas
Compagnie	loyer de base * valeur des dés	
Gare	loyer de base * nombre de gares	que possède le propriétaire de cette gare

Toutes les informations décrivant les quarante cases sont stockées dans un fichier texte, une ligne du fichier par case, la première ligne étant particulière car elle ne comporte que le nombre de cases. Voici un extrait de ce fichier :

```

40
pass  Depart          0    0    0
prop  Belleville      6000 200  1    rose
pass  Communaute-2    0    0    2
...
comp  Electricite     15000 400 12    compagnie
prop  Neuilly          14000 1000 13    violet
prop  Paradis          16000 1200 14    violet
gare  De-Lyon           20000 2500 15    gare
...
pass  Taxe-Luxe        0    0    38
prop  De-la-paix      40000 5000 39    indigo

```

N.B. Toutes les sommes du jeu sont en francs sans centimes (entières).

Chaque ligne du fichier comporte successivement les renseignements suivants : type de la case (pass ou bien prop ou bien gare ou bien comp), nom de la case, prix d'achat (0 pour les passages), loyer de base (0 pour les passages), numéro d'ordre de la case (non utilisé), nom du groupe de terrains (non utilisé).

Un autre fichier contient la description des groupes; on n'en tient pas compte ici.

Structure des données

Pour décrire les types de cases, on utilisera les facilités offertes par l'héritage. On aura donc une classe *Case* ayant deux sous-classes (classe filles) *Passage* et *Terrain*.

Suivant que l'on s'arrête sur une propriété, une gare ou une compagnie, le calcul des montants du loyer diffère. On résoudra cela grâce à la redéfinition. La classe *Terrain* devra donc déclarer une méthode *int loyerAPayer()* qui sera redéfinie dans chacune de ses trois sous-classes, *Propriete*, *Gare* et *Compagnie*.

Les quarante cases sont décrites dans un tableau de *Case*, qui est un champ de la classe *Plateau*. Un groupe de terrains est décrit dans un tableau de *Terrain*, qui est un champ de la classe *Groupe*. L'ensemble des groupes est implanté par un tableau de *Groupe*, relatif à la classe *Terrain*. La description d'un joueur est réalisée par la classe *Joueur*.

Très important Toute indentation mal faite ou incorrecte sera notée négativement.

A dessin de la hiérarchie des classes

Dessiner la hiérarchie des classes *Case*, *Passage*, *Terrain*, *Propriete*, *Gare* et *Compagnie*, sous la forme d'un arbre dans lequel vous ne mettez que le nom des classes.

B classe Groupe

Elle a deux champs, un tableau de *Terrain* pour représenter un (un seul) groupe de terrains, et un entier *taille* pour le nombre d'objets *Terrain* du tableau (du groupe).

Donner la définition de la classe *Groupe*, avec un constructeur ayant en paramètre un tableau de *Groupe*, et une méthode publique *getTerrain* fournissant en résultat le *Terrain* correspondant à un index donné.

C classe Joueur

Les champs servent à indiquer son nom, son capital, ainsi que la position actuelle du joueur sur les quarante cases du plateau de jeu. La position de départ est 0, et le capital initial est 150000. Cette classe possède un constructeur servant à initialiser les champs, et deux méthodes permettent de payer un achat ou un loyer (diminution du capital, si possible), ou de recevoir de l'argent (augmentation du capital).

Donner la définition de la classe *Joueur*.

D classe Case (niveau 1)

Donner la définition de la classe *Case*, qui comporte deux champs pour le nom et le numéro de la case, un constructeur qui permet de les initialiser et une redéfinition de la méthode *toString* pour retourner le nom de la case.

E classe Passage (niveau 2)

Tout case de type passage comporte le nom de l'action qui lui est attachée (en plus du nom de la case bien sûr). Les objets de la classe *Passage* comportent donc un champ de plus que ceux de la classe *Case*.

Donner la définition de la classe *Passage*, que vous doterez d'une méthode *toString* pour retourner les noms de la case et de l'action.

F classe Terrain (niveau 2)

Elle comporte tout ce qui est nécessaire et commun pour les terrains, indépendamment des particularités du montant du loyer à payer, qui diffère pour chacun des trois types de terrains. C'est-à-dire, prix d'achat, base du loyer, propriétaire actuel du terrain (une instance de la classe *Joueur*), en ce qui concerne les champs; le constructeur pour initialiser le nom, le numéro, le prix d'achat et la base du loyer; et enfin deux méthodes *int loyerAPayer()* et *void visite(Joueur)*.

f1 Voici la définition de la méthode *visite* de la classe *Terrain*.

```
public void visite(Joueur visiteur) {
    if (proprietaire==null) {
        visiteur.paye(prixAchat);
        proprietaire= visiteur;
    }
    else if (visiteur!=proprietaire) {
        int loyer= loyerAPayer();
        visiteur.paye(loyer);
        proprietaire.recoit(loyer);
    }
}
```

À quel paragraphe de l'énoncé de la règle du jeu correspond cette méthode ?

La méthode *loyerAPayer* retourne 0 au niveau de la classe *Terrain*. Le calcul du loyer étant différent suivant les différentes sous-classes de terrain, quelle solution proposez-vous pour le calcul du loyer ?

- f2** Donner la définition de la classe *Terrain* correspondant à ce qui vient d'être expliqué, en considérant que la méthode *loyerAPayer* retourne la valeur 0. Mettre l'entête de la méthode *visite*, mais il n'est pas nécessaire de recopier ses instructions, trois points suffiront, (...).

G classe *Compagnie* (niveau 3)

Elle permet de définir les instructions de calcul du loyer pour une compagnie (terrain particulier). Le loyer à payer est *coeff* fois le loyer de base. La valeur de ce coefficient est connue par un appel à la méthode *static int valeurDes()* de la classe *Partie* (classe qui n'est pas étudiée dans ce texte).

Donner la définition de la classe *Compagnie*.

H classe *Terrain* (suite) (niveau 2)

En cas de difficulté de compréhension, passez à la question I en considérant la question H résolue.

On a vu que tout terrain appartient à un et un seul groupe de terrains (cf. explications sur le mode de calcul des loyers à payer). La description complète des groupes de terrains est implantée par un tableau de *Groupe*, le premier groupe correspondant à l'index 0, le deuxième à l'index 1, ... et le dernier à l'index 9. Ce tableau doit être commun à tous les objets de la classe *Terrain*, et non pas particulier à chacun d'eux. Le problème de l'initialisation de ce tableau à partir du fichier de description des groupes ne sera pas abordé dans ce sujet.

- h1** Quel champ faut-il rajouter pour indiquer que la classe *Terrain* comporte une référence sur un tel tableau de description des groupes de terrains.

- h2** On considère la méthode, déclarée dans la classe *Terrain*, suivante

```
public String strGrp() {
    String res= "";
    for (int g=0; g<lesGroupes.length; g++) {
        Groupe grp= lesGroupes[g];
        for (int el=0; el<grp.taille; el++)
            res+= (grp.getTerrain(el)).nom+" ";
        res+= "\n";
    }
    return res;
}
```

Expliquer en français le rôle de l'instruction `res+= (grp.getTerrain(el)).nom+" "`

À quoi peut servir cette méthode ?

I classe *Propriete* (hiérarchie-6)

Elle permet de définir les instructions de calcul du loyer pour une propriété (terrain particulier). Selon les règles, c'est une ou deux fois le loyer de base, selon que le propriétaire du terrain (qui est un *Joueur*) possède seulement une partie ou la totalité du groupe de terrains. Une méthode *boolean possedeLeGroupe()*, permettra de savoir si c'est le cas ou non; elle parcourt tous les terrains du groupe et renvoie *false* dès qu'un terrain du groupe a un autre propriétaire; si ce n'est pas le cas, elle renvoie *true*.

Donner la définition de la classe *Propriete*.

La classe *Gare* ayant un comportement plus ou moins similaire à la classe *Propriete* ne sera pas abordée.

J méthode *initCases* de la classe *Plateau*

On considère la classe *Test*, et la classe *Plateau* qui commence ainsi :

```
import java.io.*;
import java.util.*;

class Plateau {
    private Case[] lesCases;

    Plateau() {}
    Plateau(String nomFichDsc, String nomFichGrp) {
        initCases(nomFichDsc);
        initGroup(nomFichGrp);
    }
    ...
    private void initCases(String nomFichier) {
        ... // instructions : question j2
    }
    ...
}

class Test {
    public static void main(String [] a) {
        Plateau monopoly= new Plateau(a[0],a[1]);
        System.out.println(monopoly.toString());
        System.out.println(Terrain.strGrp());
    }
}
```

j1 À quoi correspondent les paramètres effectifs *a[0]* et *a[1]*, et les paramètres formels *nomFichDsc* et *nomFichGrp* ?

j2 Le principe de la méthode *initCases* de la classe *Plateau* est simple :

```
private void initCases(String nomFichier) {
    < instructions permettant d' « ouvrir » le fichier >

    int nbLig= < récupérer le nombre de cases a partir du fichier >
    < créer l'objet référencé par lesCases >
    String code= "",nom= ""; int prix= 0,loyer= 0;
    for (int l= 0; l<nbLig; l++) {
        code= < recuperer le type de la case a partir de la ligne du fichier>
        nom= < recuperer le nom de la case a partir de la ligne du fichier>
        prix= < recuperer le prix d'achat de la case a partir de la ligne du fichier>
        loyer= < recuperer le prix d'achat de la case a partir de la ligne du fichier>
        lesCases[l]= creerCase(code,nom,prix,loyer);
    }
    < instructions permettant de « fermer » le fichier >
}
```

On vous demande de donner la définition de la méthode *creerCase* qui, à partir code de la case (pass prop comp gare), crée et retourne un objet de la classe correspondante.

```

class Groupe {
    private Terrain[] leGroupe;
    public int taille;

    Groupe() { super(); }
    Groupe(Terrain[] tab) { leGroupe= tab; length= tab.taille; }

    public Terrain getTerrain(int index) { return leGroupe[index]; }
}

class Joueur {
    private String nom;
    private int capital, position;

    Joueur() {}
    Joueur(String s, int montant) {
        nom= s;
        capital= montant;
    }

    public boolean paye(int montant) {
        if (montant>capital) { capital-= montant; return true; }
        else return false;
    }

    public void recoit(int montant) { capital+= montant; }

    public String toString() { return nom+" (" +capital+")"; }
}

class Case {
    protected String nom;

    Case() {}
    Case(String s) { nom=s; }

    public String toString() { return nom; }
}

class Passage extends Case{
    private String action;

    Passage(String n, String act) {
        super(n);
        action= act;
    }

    public String toString() { return super.toString()+" : "+action; }
}

```

```

class Terrain extends Case{
    public static Groupe[] lesGroupes;

    private    int prixAchat;
    protected int baseLoyer;
    protected int codeGroupe;      // protected Groupe group;
    protected Joueur proprietaire;

    Terrain() { super(); }
    Terrain(String nom, int prix, int loyer) {
        super(nom);
        prixAchat= prix;
        baseLoyer= loyer;
    }

    public void setProprio(Joueur j) { proprietaire= j; }
    public void setCode(int code)    { codeGroupe= code; }
    // public void setGroupe(Groupe g) { group= g; }
    public Joueur getProprio()       { return proprietaire; }

    public void visite(Joueur visiteur) {
        if (proprietaire==null) { visiteur.paye(prixAchat); proprietaire= visiteur; }
        else if (visiteur!=proprietaire) {
            int loyer= loyerAPayer();
            visiteur.paye(loyer); proprietaire.recoit(loyer);
        }
    }

    int loyerAPayer() { return 0; }    // ou    abstract int loy...

    public String toString() {
        return super.toString()+" (" +prixAchat+" "+baseLoyer+" "
            +codeGroupe+" - "+proprietaire+")";
    }

    public static String strGrp() {
        String res= "";
        for (int g=0; g<lesGroupes.length; g++) {
            Groupe grp= lesGroupes[g];
            for (int el=0; el<grp.length; el++)
                res+= (grp.getTerrain(el)).nom+" ";
            res+= "\n";
        }
        return res;
    }
}

class Compagnie extends Terrain{

    Compagnie() {}
    Compagnie(String nom, int prix, int loyer) {
        super(nom,prix,loyer);
    }

    int loyerAPayer() { return Partie.getCoeff()*baseLoyer; }
}

```

```

class Propriete extends Terrain{

    Propriete(String nom, int prix, int loyer) {
        super(nom,prix,loyer);
    }

    int loyerAPayer() { return baseLoyer*(possedeLeGroupe() ? 1 : 2); }

    private boolean possedeLeGroupe() {
        Groupe g= lesGroupes[codeGroupe];
        for (int el= 0; el<g.length; el++)
            if (g.getTerrain(el).getProprio()!=proprietaire) return false;
        return true;
    }
}

```

```

class Gare extends Terrain{

    Gare() {}
    Gare(String nom, int prix, int loyer) {
        super(nom,prix,loyer);
    }

    int loyerAPayer() { return nbGares()*baseLoyer; }

    private int nbGares() {
        Groupe g= lesGroupes[codeGroupe];
        int res= 0;
        for (int el= 0; el<g.length; el++)
            if (g.getTerrain(el).getProprio()==proprietaire) res++;
        return res;
    }
}

```

```

class Partie {
    ...

    public static int getCoeff() {
    }
}

```

```

import java.io.*;
import java.util.*;

```

```

class Plateau {
    private Case[] lesCases;

    Plateau() {}
    Plateau(String nomFichDsc, String nomFichGrp) {
        initCases(nomFichDsc);
        initGroup(nomFichGrp);
    }

    private void initCases(String nomf) {
        BufferedReader bf= null;
        try { bf= new BufferedReader(new FileReader(nomf)); }
        catch (IOException e) { System.out.println("inconnu"); System.exit(1); }

        try {
            int nbLig= Integer.parseInt(bf.readLine());
            lesCases= new Case[nbLig];
            StringTokenizer st;
            String code= "", nom= ""; int prix= 0, loyer= 0;
            for (int l= 0; l<nbLig; l++) {
                st= new StringTokenizer(bf.readLine());
                code= st.nextToken(); nom= st.nextToken();
                prix= Integer.parseInt(st.nextToken());
                loyer= Integer.parseInt(st.nextToken());
                if (code.equals("prop"))
                    lesCases[l]= new Propriete(nom, prix, loyer);
                else if (code.equals("gare"))
                    lesCases[l]= new Gare(nom, prix, loyer);
                else if (code.equals("comp"))
                    lesCases[l]= new Compagnie(nom, prix, loyer);
                else lesCases[l]= new Passage(nom, "");
            }
            bf.close();
        }
        catch(IOException e) { System.exit(2); }
    }

    private void initGroup(String nom) {
        FileReader f= null; BufferedReader bf= null;
        try { bf= new BufferedReader(new FileReader(nom)); }
        catch (IOException e) { System.out.println("inconnu"); System.exit(1); }

        try {
            int nbLig= Integer.parseInt(bf.readLine());
            Terrain.lesGroupes= new Groupe[nbLig];
            for (int gr= 0; gr<nbLig; gr++) {
                StringTokenizer st= new StringTokenizer(bf.readLine());
                int nbEl= Integer.parseInt(st.nextToken());
                Terrain[] t= new Terrain[nbEl];
                for (int el=0; el<nbEl; el++) {
                    t[el]= (Terrain) lesCases[Integer.parseInt(st.nextToken())];
                    t[el].setCode(gr); // t[el].setGroupe(t);
                }
                Terrain.lesGroupes[gr]= new Groupe(t);
            }
            bf.close();
        }
        catch(IOException e) { System.exit(2); }
    }

    public String toString() {

```

```

String res="";
for (int el=0; el<lesCases.length; el++)
    res+= lesCases[el].toString()+"\n";
return res;
}
}

class Test {
    public static void main(String [] a) {
        a= new String[2];
        a[0]= "DscCases.txt";
        a[1]= "DscGrps.txt";
        Plateau monopoly= new Plateau(a[0],a[1]);
        System.out.println(monopoly.toString());
        System.out.println(Terrain.strGrp());
    }
}

```

fichier DscCases.txt

```

40
pass   Depart           0    0  0
prop   Belleville       6000 200  1   rose
pass   Communaute-2     0    0  2
prop   Lecourbe         6000 400  3   rose
pass   Impot            0    0  4
gare   Montparnasse     20000 2500 5   gare
prop   Vaugirard        10000 600  6   pervenche
pass   Chance-7         0    0  7
prop   Courcelles      10000 600  8   pervenche
prop   Republique      12000 800  9   pervenche
pass   Prison           0    0 10
prop   La-Villette     14000 1000 11  violet
comp   Electricite     15000 400 12  compagnie
prop   Neuilly         14000 1000 13  violet
prop   Paradis         16000 1200 14  violet
gare   De-Lyon          20000 2500 15  gare
prop   Mozart          18000 1400 16  orange
pass   Communaute-17   0    0 17
prop   St-Michel        18000 1400 18  orange
prop   Pigalle          20000 1600 19  orange
pass   ParcGratuit     0    0 20
prop   Matignon        22000 1800 21  rouge
pass   Chance-22       0    0 22
prop   Malesherbes     22000 1800 23  rouge
prop   Henri-Martin    24000 2000 24  rouge
gare   Du-Nord          20000 2500 25  gare
prop   St-Honore        26000 2200 26  jaune
prop   La-Bourse        26000 2200 27  jaune
comp   Eaux             15000 400 28  compagnie
prop   LaFayette       28000 2400 29  jaune
pass   Police           0    0 30
prop   Breteuil        30000 2600 31  vert
prop   Foch            30000 2600 32  vert
pass   Communaute-33   0    0 33
prop   Capucines       32000 2800 34  vert
prop   St-Lazare        20000 2500 35  gare
pass   Chance-36       0    0 36
prop   ChampsElysees  35000 3500 37  indigo
pass   Taxe-Luxe        0    0 38
prop   De-la-paix      40000 5000 39  indigo

```

fichier DscGrps.txt

```
10
2   1  3
3   6  8  9
3  11 13 14
3  16 18 19
3  21 23 24
3  26 27 29
3  31 32 34
2  37 39
4   5 15 25 35
2  12 28
```

Exemple d'affichage à l'écran

```
Depart :
Belleville (6000 200 0 - null)
Communaute-2 :
Lecourbe (6000 400 0 - null)
Impot :
Montparnasse (20000 2500 8 - null)
Vaugirard (10000 600 1 - null)
Chance-7 :
Courcelles (10000 600 1 - null)
Republique (12000 800 1 - null)
Prison :
La-Villette (14000 1000 2 - null)
Electricite (15000 400 9 - null)
Neuilly (14000 1000 2 - null)
Paradis (16000 1200 2 - null)
De-Lyon (20000 2500 8 - null)
Mozart (18000 1400 3 - null)
Communaute-17 :
St-Michel (18000 1400 3 - null)
Pigalle (20000 1600 3 - null)
ParcGratuit :
Matignon (22000 1800 4 - null)
Chance-22 :
Malesherbes (22000 1800 4 - null)
Henri-Martin (24000 2000 4 - null)
Du-Nord (20000 2500 8 - null)
St-Honore (26000 2200 5 - null)
La-Bourse (26000 2200 5 - null)
Eaux (15000 400 9 - null)
LaFayette (28000 2400 5 - null)
Police :
Breteuil (30000 2600 6 - null)
Foch (30000 2600 6 - null)
Communaute-33 :
Capucines (32000 2800 6 - null)
St-Lazare (20000 2500 8 - null)
Chance-36 :
ChampsElysees (35000 3500 7 - null)
Taxe-Luxe :
De-la-paix (40000 5000 7 - null)
```

```
Belleville Lecourbe
Vaugirard Courcelles Republique
La-Villette Neuilly Paradis
Mozart St-Michel Pigalle
Matignon Malesherbes Henri-Martin
St-Honore La-Bourse LaFayette
Breteuil Foch Capucines
ChampsElysees De-la-paix
Montparnasse De-Lyon Du-Nord St-Lazare
Electricite Eaux
```