

Titre :
(LgP6Linf.EPS)
Auteur :
Adobe Illustrator(TM) 3.2
Aperçu :
Cette image EPS n'a pas été enregistrée
avec un aperçu intégré.
Commentaires :

Examen terminal écrit : un programme

Présentation du jeu simplifié

On considère un jeu de cartes de hasard pour enfants, présenté ici dans une version considérablement simplifiée et modifiée, que nous appellerons le « jeu des mille points ». Ses 73 cartes se répartissent en 46 cartes *point* et 27 cartes *action*. Chacune des 46 cartes point porte une valeur, 50 ou 100, soit 30 cartes à 50 points et 16 cartes à 100 points. Les 27 cartes action servent à entraver la marche de l'adversaire ou à lever une entrave de l'adversaire; elles se répartissent en trois groupes d'attaque/parade comportant chacun 3 exemplaires de la carte *attaque* et 6 exemplaires de la carte *parade* correspondante. Au total on a donc $73 = (30+16) + (3 * (3+6))$.

Le principe du jeu consiste à cumuler des points, jusqu'à ce qu'un des deux joueurs, atteigne mille points, auquel cas ce joueur est déclaré gagnant, la marche aux mille points pouvant parfois être entravée par des actions de l'adversaire. Si arrivé au bout des 73 cartes, aucun des joueurs n'a atteint mille points, le gagnant est celui qui a le plus de points.

Au départ, on mélange le paquet des 73 cartes, puis on distribue six cartes à chacun des deux joueurs. À un moment donné du déroulement de la partie, chaque joueur dispose, avant de jouer, de ses six cartes (sa main) et d'un tas de cartes posées sur le tapis de jeu comportant entre autres toutes les cartes point déjà déposées par ce joueur. Bien entendu, en plus des deux *tas particuliers* des deux joueurs, il y a sur le tapis de jeu deux autres tas, communs aux deux joueurs, correspondant à la *pioche* des cartes à tirer et à la *défausse* des cartes n'ayant pu être mises dans un des tas des joueurs (cartes non utilisées).

Lorsque son tour arrive, le joueur dispose de ses six cartes, sa main. Il prend une septième sur le dessus de la pioche et l'ajoute à sa main. Pour revenir à six cartes à la fin du tour, il doit obligatoirement déposer une des sept cartes. Il applique pour cela un des quatre cas suivants :

- ou bien il pose une carte point dans son propre tas pour augmenter son cumul de points, à condition de ne pas être présentement bloqué par une attaque de l'adversaire,
- ou bien il pose une carte attaque dans le tas de l'adversaire, pour empêcher celui-ci de déposer des cartes point dans son tas lorsque viendra son tour,
- ou bien il pose une carte parade dans son propre tas pour ne plus être bloqué et contrer l'attaque de l'adversaire et pouvoir par la suite continuer à déposer des cartes point sur son tas,
- ou bien il pose une quelconque de ces sept cartes dans la défausse.

On précise qu'une attaque d'un groupe ne peut être contrée que par une carte parade du même groupe.

Dans ce qui suit on s'intéresse principalement à la description du jeu de cartes, en vue malgré tout de son utilisation dans le cadre d'une partie, certaines méthodes demandées y faisant explicitement référence.

Structure des données

Pour décrire les trois types de cartes, on utilisera les facilités offertes par l'héritage. On aura donc une classe *Carte* ayant deux sous-classes (classe filles) *CarteAvance* et *CarteAction*, cette dernière ayant à son tour deux sous-classes *CarteAttaque* et *CarteParade*.

Pour pouvoir jouer, on a besoin de connaître le type de carte : la classe *Carte* comportera une méthode *getMode()*, qui sera redéfinie dans chacune des classes correspondant effectivement à la définition des objets pour chacun des trois types de cartes du jeu.

Les soixante-treize cartes sont décrites dans un tableau de *Carte*, qui est un champ de la classe *LesCartes*. La composition du jeu de cartes est décrite dans un interface *IConst* qui est utilisé dans *LesCartes* (cf. question J).

Choix des autorisations d'accès

Sauf indication dans la question, il est de votre responsabilité de bien choisir l'autorisation d'accès (*private*, *protected*, *public*, *abstract*) que vous donnez pour un champ, une méthode ou une classe, le principe général étant qu'il convient de verrouiller au maximum l'accès aux données tout en autorisant l'utilisation nécessaire des données. N'oubliez pas non plus le principe de l'encapsulation des données.

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

A dessin de la hiérarchie des classes

Dessiner la hiérarchie des classes *Carte*, *CarteAvance*, *CarteAction*, *CarteAttaque*, *CarteParade*, et *LesCartes*, sous la forme d'un arbre ayant pour racine la classe *Object*, dans lequel vous ne mettez que le nom des classes.



B classe *Carte*

Elle sera utilisée lors de la définition du tableau d'objets servant à représenter les 73 cartes du jeu. En conséquence, elle comporte une méthode `getMode()`, qui sera redéfinie dans les classes filles, en vue de pouvoir connaître le type d'objet effectivement repéré par un élément du tableau.

Chaque objet comporte un numéro, correspondant au rang de la carte dans le jeu, ce numéro étant attribué par le constructeur par défaut à partir d'une variable de classe. En ce qui nous concerne, ce numéro sera utilisé lors de l'affichage des cartes du jeu.

On ne redéfinit pas `toString`.

Donner la définition de la classe *Carte*.

```
abstract class Carte {
    private static int rang= 1;
    protected int numero;

    Carte() { numero= rang++; }

    abstract public char getMode();
}
```

C classe *CarteAvance*

Elle réalise concrètement *Carte* pour les cartes point.

Elle comporte donc un champ pour le nombre de points de la carte, initialisé par un constructeur adéquat, et disposant d'un accesseur.

Le « mode » de la carte (son type) sera représenté par le caractère '+'. On redéfinit `toString` pour sortir un affichage du type `5:+(50)`, où 5 est le rang de la carte, 50 le nombre de points de la carte et '+' le type de la carte (carte point).

Donner la définition de la classe *CarteAvance*.

```
class CarteAvance extends Carte{
    private int nbPoints;
```

```

CarteAvance(int np) {
    super();
    nbPoints= np;
}

public char getMode(){ return '+'; }
public int getKm() { return nbPoints; }

public String toString() { return "+"+nbPoints+" "; }
}

```

D classe *CarteAction*

Les attaques/parades sont réparties en trois groupes. Il convient d'ajouter un champ pour le code du groupe, qui sera 1, 2 ou 3. C'est là le seul objectif de cette classe.

Elle comporte donc un champ, ainsi que deux constructeurs, le constructeur par défaut permettant d'initialiser à -1 le groupe par appel de l'autre constructeur.

Si vous avez programmé la classe *Carte* conformément aux normes de Java, vous devriez normalement avoir une erreur de compilation. Justifiez ce fait.

Donner la définition de la classe *CarteAction*.

```

abstract class CarteAction extends Carte{
    protected int groupe;

    CarteAction(int g) { super(); groupe=g; }
    CarteAction()      { this(-1); }
}

```

E classes *CarteAttaque* et *CarteParade*

Elles servent à réaliser concrètement *Carte* pour les cartes attaque et parade (y compris donc la notion de groupe). Les modes respectifs seront 'A' et 'P'. L'affichage sera du même type que celui de *CarteAvance*, en remplaçant le nombre de points par le code du groupe.

Donner les définitions des classes *CarteAttaque* et *CarteParade*.

```

class CarteAttaque extends CarteAction {
    CarteAttaque(int g) { super(g); }

    public char getMode(){ return 'a'; }
    public String toString() { return "A("+groupe+" "; }
}

class CarteParade extends CarteAction{
    CarteParade(int g) { super(g); }

    public char getMode(){ return 'p'; }

    public String toString() { return "P("+groupe+" "; }
}

```

F classe *LesCartes* - champs

Les 73 cartes du jeu y sont représentées grâce à un tableau d'objets de nom *paquet*. À un moment donné d'une partie, la valeur de l'indice *dessus* permet de délimiter la pioche parmi ces 73 cartes : elle est constituée de tous les éléments du tableau à partir de l'indice *dessus* jusqu'à la fin du tableau, le dessus de la pioche correspondant à l'indice *dessus*.

Donner la définition de ces deux champs, sachant qu'ils ne sont pas initialisés au niveau de la définition mais par le constructeur vu en question K.

```

interface IConst {
    final int nbCartes= 73;
    final int[] nbAttaques= {3,3,3};
    final int[] nbParades = {6,6,6};
}

```

```

    final int[] nbValeurs = {30,16};
    final int[] valeurPoints= {50,100};
}

class LesCartes implements IConst{
    private int dessus;
    private Carte[] paquet;

    LesCartes() {
        paquet= new Carte[nbCartes];
        description();
        melanger();
        dessus= 0;
    }
}

```

G classe *LesCartes* – mélanger le paquet

Avant des pouvoir distribuer les cartes aux deux joueurs, il convient de mélanger le paquet de cartes.

Donner en français ou en pseudo-français le principe de l’algorithme que vous allez utiliser pour cela.

Donner la définition de la méthode privée `melanger` permettant de programmer cet algorithme.

```

private void melanger() {
    // melanger le paquet (toute methode qui le fait est bonne)
    for (int i=0, nbFois= paquet.length*3/2; i<nbFois; i++) {
        int pos1= (int) (Math.random()*paquet.length);
        int pos2= (int) (Math.random()*paquet.length);
        Carte temp= paquet[pos1]; paquet[pos1]= paquet[pos2]; paquet[pos2]=
temp;
    }
}

```

H classe *LesCartes* – prendre la carte du dessus de la pioche

Relisez tout d’abord les indications de la question F.

Donner la définition de la méthode `carteSuiivante`, qui permet à tout instant de la partie de fournir en résultat la carte du dessus de la pioche. Cette méthode vérifie grâce à la valeur de `dessus` si l’opération est effectivement possible, et renvoie la carte dans ce cas et `null` sinon.

Pour gérer cette méthode à l’aide du mécanisme des exceptions on peut utiliser la classe *ArrayIndexOutOfBoundsException* qui hérite de la classe *Exception* (et donc de la classe *Throwable*). Donner la définition de cette deuxième version d’écriture de la méthode.

La remarque entre parenthèses n’est nullement anodine. Expliquez pourquoi.

```

public Carte carteSuiivante() {
    // if (dessus<paquet.length)
    //     return paquet[dessus ++];
    // else return null;
    try { return paquet[dessus ++]; }
    catch (ArrayIndexOutOfBoundsException e) { return null; }
}

```

I classe *LesCartes* – distribuer n cartes

Au début du jeu, on doit distribuer six cartes à chacun des deux joueurs. On généralise cela en disant que la méthode `distribuer(int n)` prend les n cartes du dessus de la pioche et fournit en résultat une main de n cartes.

Donner la définition de la méthode `distribuer`.

```

public Carte[] distribuer(int n) {
    Carte[] res= new Carte[n];
    for (int c=0; c<n; c++) res[c]= carteSuiivante();
    return res;
}

```

J classe *LesCartes* – pour finir ...

On considère qu'il existe dans la définition de la classe *LesCartes* la méthode `description()` qui permet de décrire chacune des 73 cartes. En voici des extraits :

```
private void description() {
// description de toutes les cartes a partir de IConst
int c= 0;
// boucles pour les cartes attaque
    paquet[c++]= new CarteAttaque(v);
// boucles pour les cartes parade
    paquet[c++]= new CarteParade(v);
// boucles pour les cartes point
    paquet[c++]= new CarteAvance(valeurPoints[v]);
}
```

Cette méthode utilise directement des constantes de l'interface `IConst`. Donner l'entête complet de la définition de la classe *LesCartes*.

Donner alors la définition du constructeur par défaut de *LesCartes*, qui utilise certaines des méthodes définies pour initialiser le jeu de cartes et le rendre disponible pour la distribution initiale.

La redéfinition de la méthode `toString` permettra d'afficher toutes les cartes du paquet mélangé. Donner la définition de la méthode `toString`.

Donner la définition de la classe `Test` permettant de tester ces classes.

```
private void description() {
// description de toutes les cartes a partir de IConst
int c= 0;
for (int v=0; v<nbAttaques.length; v++)
    for (int i=0; i<nbAttaques[v]; i++)
        paquet[c++]= new CarteAttaque(v);
for (int v=0; v<nbParades.length; v++)
    for (int i=0; i<nbParades[v]; i++)
        paquet[c++]= new CarteParade(v);
for (int v=0; v<nbValeurs.length; v++)
    for (int i=0; i<nbValeurs[v]; i++)
        paquet[c++]= new CarteAvance(nomsPoints[v]);
}

public String toString() {
    String res= "";
    for (int c=0; c<paquet.length; c++) { res += paquet[c].toString(); }
    return res;
}

class Test {
    public static void main(String[] a) {
        LesCartes cartes= new LesCartes();
        System.out.println(cartes);
    }
}
```

Exercice indépendant

Cet exercice réfère malgré tout au jeu des mille points expliqué ci-dessus.

Construire une classe *PanneauCarte* héritant de *Jpanel* et permettant de dessiner une carte de type *CarteAvance*. Visuellement, cette carte comporte deux rectangles l'un sur l'autre, le supérieur, qui a une hauteur du quart de la taille de la carte et est rouge, et l'inférieur, qui a une hauteur des trois quarts de la taille de la carte et est blanc; au centre du rectangle inférieur blanc on affichera le nombre de points suivi du mot « points ».

Écrire un programme de test qui affichera une carte dans une fenêtre, la valeur de la carte étant tirée aléatoirement parmi les deux valeurs possibles.

Écrire un programme de test qui affichera une main de 3 cartes (un ensemble de 3 cartes) les unes à côté des autres.