Devoir de programmation objet
DEUG MIAS 2

Marc Champesme
Marc.Champesme@lipn.univ-paris13.fr
Departement d'Informatique
Institut Galilée

27 avril 2003

Tous les devoirs doivent être remis à votre chargé de TD (en main propre ou dans son casier), **au plus tard une semaine avant le début des partiels de juin 2003**. Tout devoir rendu après cette date sera considéré comme non fait.

1 Objectif du devoir

Le but est de mettre en pratique les concepts de la programmation orientée objet et plus particulièrement :
- bien structurer le code d'une application en classes et méthodes ;
- écrire et commenter un programme afin qu'il puisse être réutilisé ;
- utiliser correctement l'héritage et le polymorphisme ;
- utiliser la programmation par contrat.

2 Présentation du sujet

Il s'agit d'apporter des améliorations à un logiciel de jeu de cartes de type "réussite", tel que ceux fournit en standard avec certains systèmes d'exploitation. Pour réduire la quantité de travail nécessaire, le code source en JAVA d'un logiciel "Jeu du solitaire" vous est fourni (disponible prochainement dans votre dossier partagé). Ce logiciel est le travail rendu par un étudiant de DEUG MIAS 2 en 2002. La version qui est mise à votre disposition est en fait une version légèrement adaptée de ce travail, la principale modification apportée étant une traduction des assertions en JML. En particulier, cette version fonctionne correctement avec les assertions activées.

Cependant, cette version du logiciel possède un certain nombre de défauts :
- il manque les assertions pour de nombreuses méthodes et lorsqu'elle sont présentes elles ne sont pas toujours correctes ou bien elles devraient être complétées ;
- les commentaires sont souvent absents, incomplets ou mal formulés ;
- les identificateurs ne sont pas toujours très explicites ;
- séparation incomplète entre le code gérant l'interface et le reste du code ;
- utilisation non justifiée d'attributs `public` ou `protected` ;
- séparation incomplète (i.e. dans des classes distinctes) entre le code gérant l'interface utilisateur et le reste du code.

3 Travail demandé

Le devoir doit être réalisé individuellement : chaque étudiant devra rendre sa propre version du logiciel sur laquelle il sera évalué. Le travail à réaliser consiste à :
- Modifier le code de l'application fournie, afin qu'il satisfasse les standards de programmations exposés à la fin de ce document. En particulier, les défauts mentionnés précédemment devront être corrigés ;

– Modifier le logiciel afin qu'il soit possible de jouer à un autre jeu de réussite avec le même logiciel : le jeu disponible dans la version mise à disposition est la réussite dite "Klondike" (cf. règles à l'annexe A), votre logiciel devra en plus permettre de jouer à la réussite dite "de l'horloge" dont les règles sont données dans l'annexe B.

Corrections à apporter au logiciel

Vous devrez revoir la structuration du code entre les différentes classes afin de permettre une utilisation simple et directe de certaines classes pour d'autre jeux de cartes. Cette structuration devra utiliser l'héritage de manière privilégiée.

Afin de rendre la gestion de l'interface indépendante de la gestion du jeu, les caractéristiques (attributs ou méthodes) liées à l'affichage doivent être placées dans des classes distinctes de celles assurant le fonctionnement du jeu. Par exemple, dans la classe `PileCartes`, les attributs `x` et `y`, ainsi que les méthodes les utilisant doivent être définies dans des classes distinctes (i.e. en dehors de la classe `PileCartes`).

4 Evaluation

Compte tenu de l'objectif fixé, ce n'est pas la richesse en fonctionnalité, mais plutôt la qualité de la réalisation qui sera évaluée.

La qualité de la réalisation sera évaluée selon les critères suivant :
- architecture de l'application : bonne répartition du code en classes, bon usage de l'héritage, bonne séparation entre le code gérant l'interface utilisateur et le reste du code...
- conception et réalisation favorisant la réutilisabilité du code ;
- utilisation adéquate de la programmation par contrat (dans la définition des assertions *et* l'implémentation des méthodes) ;
- bonne documentation des classes ;
- fonctionnement de l'application : ergonomie (facilité d'utilisation), absence de bug (y compris lorsque l'exécution se fait avec contrôle des assertions activé).

5 Documents à rendre par chaque étudiant

Chaque étudiant rend une disquette et un rapport sur papier contenant :
- Description des modifications apportées à l'application initiale (fichier sur la disquette + version imprimée) ;
- La documentation (produite par `javadoc`) des classes (fichiers html sur la disquette) ;
- Le code de toute l'application (sur la disquette).

6 Standards de programmation

Commentaires Les commentaires doivent être rédigés en respectant les règles suivantes :
- Tous les commentaires doivent être en français ;
- Chaque classe, méthode ou champ public doit posséder son propre commentaire au format javadoc.

Choix des identificateurs Tous les identificateurs doivent être francisés. Les seules exceptions admises sont les préfixes `get` et `set` pour les méthodes servant à consulter ou modifier un champ particulier : par exemple, il est admit que l'identificateur de méthode `setCenter` soit traduit (de manière incomplète) en `setCentre`.

Assertions Vous devez définir un invariant pour chaque classe et chaque méthode doit posséder une pré-condition et une post-condition, lorsque cela est possible et justifié.

Vous devez vous efforcer d'écrire les assertions en JML de façon à ce qu'elles puissent être activées à l'exécution. Cependant certaines assertions ne peuvent être décrites de manière satisfaisante en JML, dans ce cas, les assertions pourront être écrites en français.

Visibilité / masquage d'information Toutes les classes et les méthodes doivent être déclarées `public` et tous les attributs doivent être déclarés `private`. Il est néanmoins admis que les attributs `static final` (i.e. les constantes) soient déclarés `public`.

Toute exception à cette règle devra être clairement justifiée dans les commentaires.

Variables de classe L'usage de variables de classes devra être réservé à des cas exceptionnels (par exemple variables `final`).

A La réussite "Klondike"

L'aire de jeu se compose des quatre cellules de destination, de 7 colonnes de cartes et du jeu de cartes. En début de partie, le jeu est en partie distribué dans les 7 colonnes de manière à ce que seule la carte au bas de chaque colonne soit visible.

A.1 But du jeu

- Le but du jeu est de faire passer toutes les cartes dans les cellules de destination, en utilisant les colonnes comme lieu de stockage provisoire.
- Pour gagner, vous devez former dans les cellules de destination quatre piles de cartes correspondant aux quatre couleurs, les cartes étant classées par ordre croissant (en commençant par l'as).

A.2 Déplacements

Pour déplacer une carte, cliquez dessus, puis sur l'endroit où vous voulez la placer. Plusieurs types de déplacements sont autorisés :

- Vous pouvez déplacer vers une cellule de destination une carte située en bas d'une colonne. Ce déplacement vers la cellule de destination ne peut s'appliquer qu'à une carte immédiatement supérieure et de même famille (i.e. cœur, carreau, pique ou trèfle) que celle qui s'y trouve déjà.
- Un as peut toujours être placé dans une cellule de destination libre.
- Vous pouvez déplacer vers le bas d'une colonne une carte située en bas d'une autre colonne. Ce déplacement vers une colonne ne peut s'appliquer qu'à une carte immédiatement inférieure et de couleur (i.e. rouge ou noir) opposée à celle qui s'y trouve déjà.
- Si vous avez plusieurs cartes dans l'ordre dans une colonne, vous pouvez déplacer cette suite en une seule opération vers une autre colonne. Pour déplacer plusieurs cartes d'une suite à la fois, cliquez sur la carte située en bas de la suite à déplacer, puis sur la colonne de destination.

B La réussite "de l'horloge"

L'aire de jeu se compose de douze cellules de destination disposées en forme d'horloge – chaque cellule représentant une heure de l'horloge – et de 8 colonnes de cartes.

B.1 Début de partie

En début de partie (cf. figure 1), chaque cellule de destination contient une carte dont la valeur est fixée en fonction de sa position dans l'horloge :

Position	Carte
1 heure	10 de cœur
2 heures	Valet de pique
3 heures	Dame de carreau
4 heures	Roi de trèfle
5 heures	2 de cœur
6 heures	3 de pique
7 heures	4 de carreau
8 heures	5 de trèfle
9 heures	6 de cœur
10 heures	7 de pique
11 heures	8 de carreau
12 heures	9 de trèfle

Le reste du jeu est complètement distribué dans les 8 colonnes de manière à ce que toutes les cartes soient visibles.

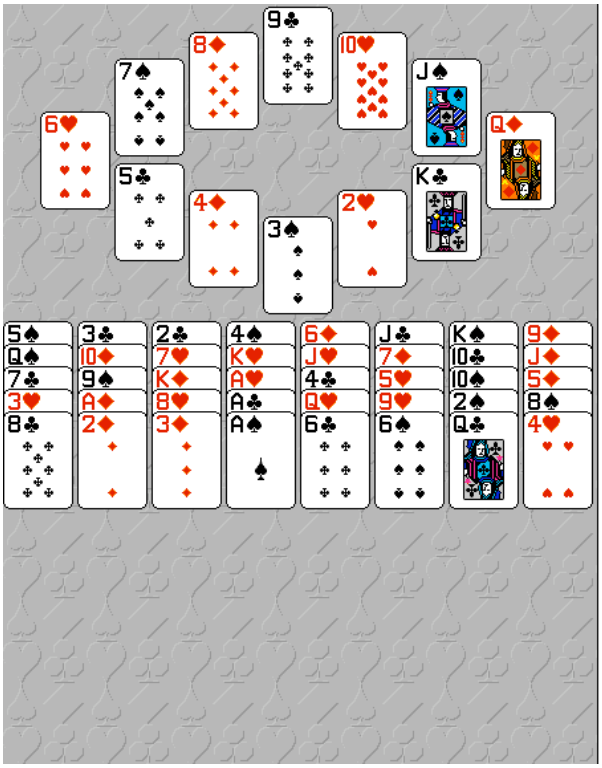

FIG. 1 – Configuration du jeu en début de partie.

B.2 Fin de partie

En fin de partie (cf. figure 2), chaque cellule de destination contient une carte dont le rang correspond à sa position dans l'horloge :

Position	Carte
1 heure	As de cœur
2 heures	2 de pique
3 heures	3 de carreau
4 heures	4 de trèfle
5 heures	5 de cœur
6 heures	6 de pique
7 heures	7 de carreau
8 heures	8 de trèfle
9 heures	9 de cœur
10 heures	10 de pique
11 heures	Valet de carreau
12 heures	Dame de trèfle

Les 8 colonnes sont vides.

B.3 Déplacements

Plusieurs types de déplacements sont autorisés :

- Vous pouvez déplacer vers une cellule de destination une carte située en bas d'une colonne. Ce déplacement vers la cellule de destination ne peut s'appliquer qu'à une carte immédiatement supérieure et de même famille (i.e. cœur, carreau, pique ou trèfle) que celle qui s'y trouve déjà. Par exemple, seul le 2 cœur peut-être placé sur l'As de cœur.
- Vous pouvez déplacer vers le bas d'une colonne une carte située en bas d'une autre colonne. La carte déplacée doit être de rang immédiatement inférieure à la carte réceptrice (par exemple : déplacement

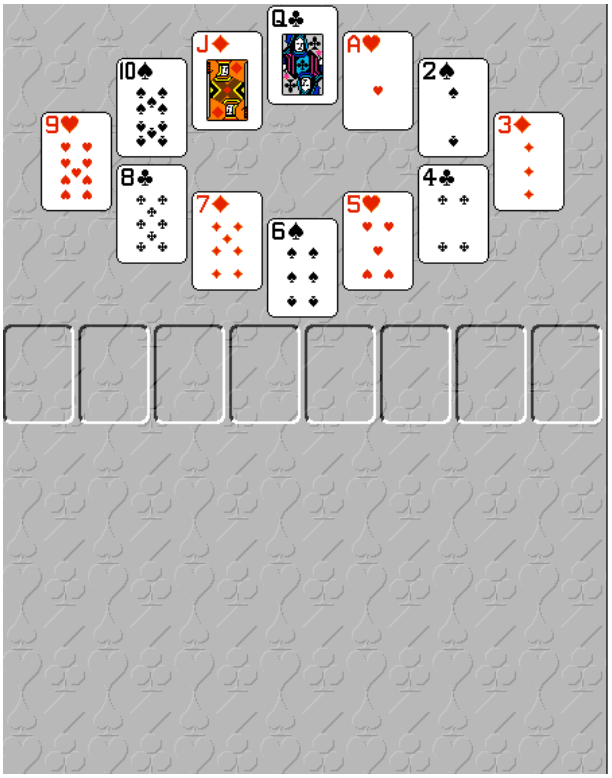

FIG. 2 – Configuration du jeu en fin de partie.

du 2 de pique sur le 3 de trèfle) et quelques soient les couleurs ou familles des cartes. Un As peut-être déplacé sur un 2 et un Roi sur un As.