TD6 – Découverte du diagramme d'états

I Processus dans un système d'exploitation LINUX

Un processus est une occurrence d'un programme (fichier exécutable) en exécution.

Un processus peut être lancé en arrière-plan (background) en ajoutant un « et commercial » (&) à la fin de l'appel du programme. Sans ce symbole, le processus est lancé au premier plan.

Un processus au premier plan peut être suspendu (arrêt provisoire) par la commande « <CTRL> Z », un processus en arrière-plan peut être suspendu par la commande « stop %n» (n étant le numéro de « job » associé au processus).

La commande « job » affiche la liste des processus en cours d'exécution avec leur numéro d'exécution.

Un processus au premier plan peut être tué par la commande « <CTRL> C » ; un processus en arrière-plan peut être tué par la commande « kill –9 %n ».

Un processus suspendu peut être relancé au premier plan par la commande « fg %n » ou en arrière-plan par la commande « bg %n ».

Question

1- Etablissez le diagramme d'états de la classe Processus.

II Question de Cours

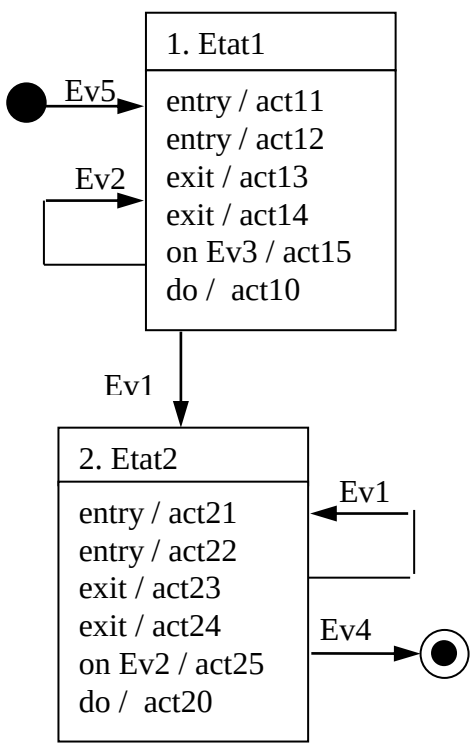


Diagramme d'états de la classe «Classe Bidule »

1. Que fait un objet de classe « Classe Bidule » dans Etat1 lors de l'arrivée de l'événement « Ev1 » ?
2. Que fait un objet de classe « Classe Bidule » dans Etat1 lors de l'arrivée de l'événement « Ev2 » ?
3. Que fait un objet de classe « Classe Bidule » dans Etat1 lors de l'arrivée de l'événement « Ev3 » ?
4. Que fait un objet de classe « Classe Bidule » dans Etat1 lors de l'arrivée de l'événement « Ev4 » ?
5. Que fait un objet de classe « Classe Bidule » dans Etat2 lors de l'arrivée de l'événement « Ev1 » ?
6. Que fait un objet de classe « Classe Bidule » dans Etat2 lors de l'arrivée de l'événement « Ev2 » ?
7. Que fait un objet de classe « Classe Bidule » dans Etat2 lors de l'arrivée de l'événement « Ev3 » ?
8. Que fait un objet de classe « Classe Bidule » dans Etat2 lors de l'arrivée de l'événement « Ev4 » ?
9. Que se passe-t-il lors de l'arrivée de l'événement « Ev5 »

III Gestion d'un jeu de pions à deux joueurs

Voici une présentation partielle et simplifiée des règles du fonctionnement d'un jeu de pions à deux joueurs à plusieurs coups par main (par exemple Tzaar) :

- deux joueurs s'affrontent, l'un ayant les pièces de couleur blanche (c.-à-d. Blanc) et l'autre celles de couleur noire (c.-à-d. Noir) ;
- au début de la partie, le joueur « Blanc» à la main (c.-à-d. qu'il commence) ;
- à chaque fois qu'un joueur à la main, il joue un premier coup consistant à prendre une pièce adverse (si cela lui est possible) et un second coup consistant soit à prendre de nouveau une pièce adverse soit à passer son tour sans rien faire ; ensuite, c'est le joueur adverse qui a la main ;
- la partie se termine lorsque le joueur qui a la main ne peut pas jouer son premier coup : c'est le perdant.

La « conception » actuelle se résume au code en Java de la classe `GestionGlobalJeu` réalisé par un programmeur un peu paresseux.

```
public class GestionGlobalJeu {
      private int c;
      private boolean partieFinie;
      private int g, joueurCourant;
      // 0 == Blanc, 1 == Noir, -1 == inconnu

      public GestionGlobalJeu() {
            c = 1; joueurCourant = 0; partieFinie = false; g = -1;  }

      private void suivant(){
            if (c == 1) {c = 2; return;}
            if (joueurCourant == 0) joueurCourant = 1;
            else joueurCourant = 0;
            c = 1;   }

      private void pasPrise(int j){
            if ((c == 1) && (joueurCourant == j)) {
            partieFinie = true; g = (j+1)%2;  }

      public void prendre(int j){
            if ((j == joueurCourant) && (!partieFinie)) suivant();  }

      public void passer(int j){
            if ((j == joueurCourant) && (!partieFinie) && (c==2))
                  suivant();  }
      public void pasPriseNoir() { pasPrise(1); }
      public void pasPriseBlanc() { pasPrise(0); }
}
```

Questions

1. Réalisez le diagramme de classes `GestionGlobalJeu`.
2. Proposez un diagramme d'états pour la classe `GestionGlobalJeu`, avec les actions.
3. Caractérisez les états de la classe `GestionGlobalJeu`.

Voici des compléments ou des modifications du fonctionnement présenté précédemment :

- au tout début de la partie (et donc une seule fois durant toute la partie), le joueur « Blanc » ne joue que le premier coup.
- à chaque fois qu'un joueur à la main, il joue un premier coup consistant à prendre une pièce adverse (si cela lui est possible) et **3 autres coups** consistant soit à prendre, soit à passer son tour sans rien faire ; ensuite, c'est le joueur adverse qui a la main.

Questions

4. Modifiez le diagramme de classes en prenant en compte les éléments d'informations supplémentaires.
5. En prenant en compte la nouvelle spécification, écrivez le code complet de la classe `GestionGlobalJeu`.
6. En prenant en compte la nouvelle spécification, dessinez un nouveau diagramme d'états pour la classe `GestionGlobalJeu`, avec les actions et caractérisez les états.

IV Contrôle d'une porte palière d'ascenseur

Le contrôle d'une porte palière d'ascenseur est géré par un logiciel. L'objectif est d'ouvrir la porte palière uniquement lorsque l'ascenseur est présent à l'étage pour éviter tout accident.
La porte est ouverte à l'arrivée de l'ascenseur. La porte reste ouverte 30 secondes pour permettre aux personnes à l'étage d'entrer dans l'ascenseur.

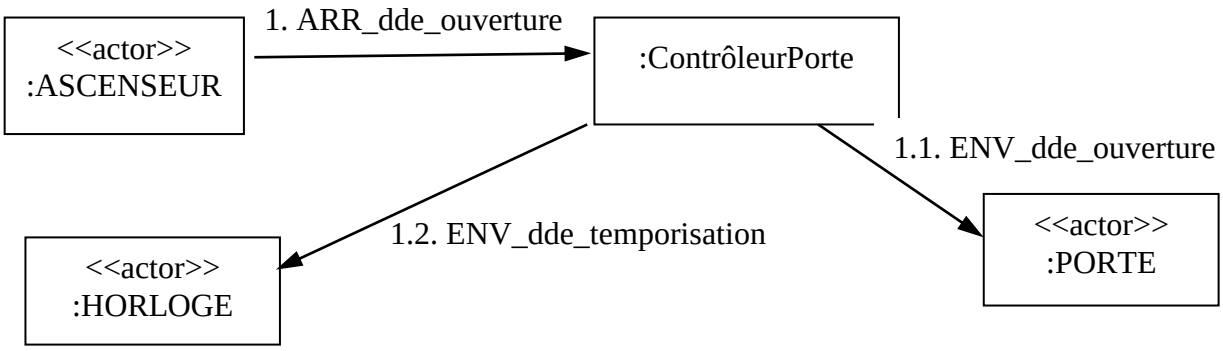


Diagramme de communication : ouverture porte palière

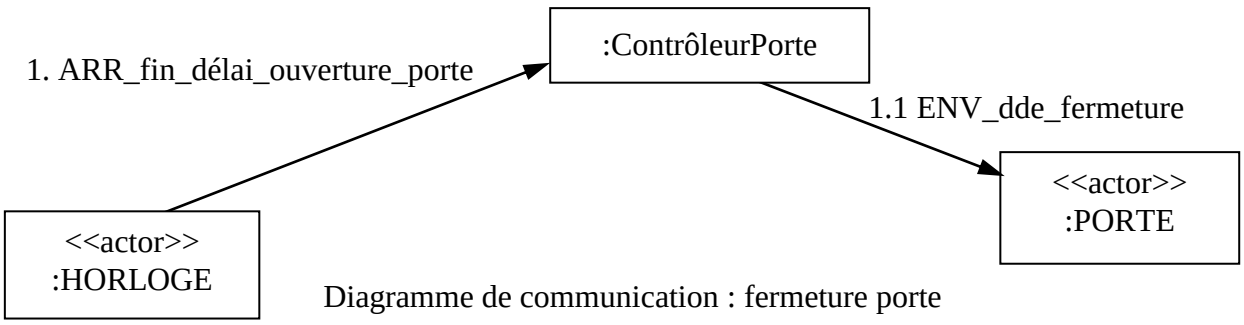


Diagramme de communication : fermeture porte

Questions

1. Etablissez le diagramme d'états de la classe `ContrôleurPorte` à l'aide des diagrammes de communication ci-dessus.

Complément d'information :

Pour garantir que l'ascenseur ne se déplace que lorsque la porte palière est effectivement fermée, l'ascenseur est bloqué jusqu'au moment où le contrôleur de la porte palière lui indique que la porte physique est effectivement fermée.
Lors de la demande de fermeture de la porte, un capteur de fermeture de porte est mis en marche par le contrôleur. Dès que la porte est effectivement fermée, le capteur transmet un signal au contrôleur de la porte qui débloque l'ascenseur (l'ascenseur peut à nouveau se déplacer).

Questions

2. Modifiez les diagrammes de communication précédents.
3. Construisez les diagrammes de communication du scénario suivants « déblocage de l'ascenseur ».
4. Complétez le diagramme d'états de la classe « Contrôleur Porte », avec les actions.
5. Donnez la classe « Contrôleur Porte ».

V Le Bintz

Le Bintz est un petit animal sympathique mais quelque peu exigeant.
À sa naissance, le Bintz est en état normal.
Un Bintz en état normal n'a pas faim pendant un certain temps (appelé temps d'autonomie). Au bout de ce temps, le Bintz a faim et il pleure.
Pour lui donner à manger, le propriétaire du Bintz le met à table et le Bintz s'arrête de pleurer.
À table, un Bintz met son bavoir et mange pendant un certain temps (appelé temps de restauration). Au bout de ce temps, il retire son bavoir et se remet à pleurer.
Il pleure jusqu'à ce que son propriétaire le sorte de table.
Quand il sort de table, le Bintz revient dans l'état normal... et ainsi de suite tant que le Bintz ne meurt pas.
Si le Bintz pleure plus de 5 minutes d'affilée, il meurt.
Un Bintz en état normal sourit lorsqu'on lui chante une chanson.

Questions

1. Proposez un diagramme d'états modélisant le comportement d'un Bintz.
2. Réalisez le diagramme de classes « Bintz ».

VI Validation de la distribution des cartes

Votre objectif est de concevoir les tests de conformité de la méthode `distribuer` de la classe `MainCartes` cette méthode vérifie la correction de la distribution des cartes pour diverses situations (présentées ci-dessous).

- Dans le cas d'une partie de Tarot à 5 joueurs, chaque joueur reçoit 15 cartes distribués trois par trois.
- Dans le cas d'une partie de Tarot à 4 joueurs chaque joueur reçoit 18 cartes distribuées trois par trois.
- Dans le cas d'une partie de Tarot à 3 joueurs, chaque joueur reçoit 24 cartes, distribuées quatre par quatre.
- Dans le cas d'une partie de Bridge (4 joueurs), chaque joueur reçoit 13 cartes, distribuées une par une.
- Dans le cas d'une partie de Belote (4 joueurs), chaque joueur reçoit 12 cartes, distribuées deux par deux.

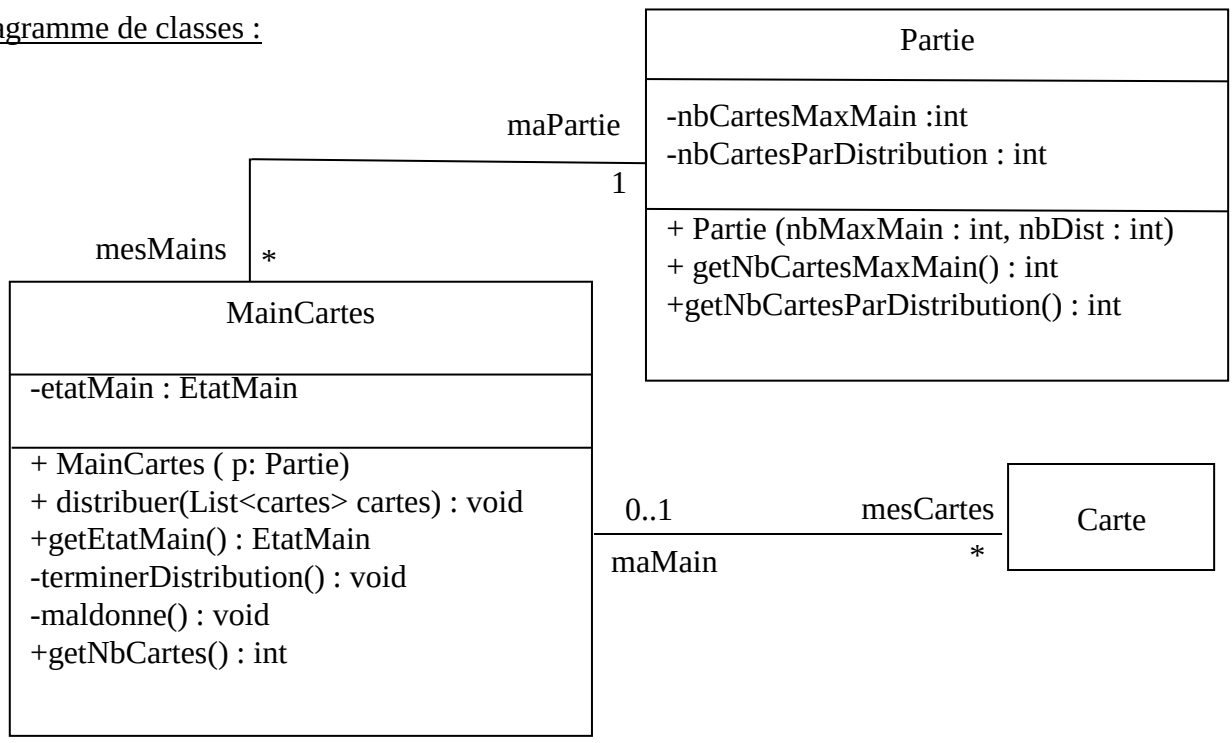
Tout écart par rapport à ces règles constitue une « maldonne ».

Hypothèse simplificatrice : l'ordre des joueurs dans les distributions est toujours correct.

Reste à vérifier que chaque joueur (main) obtienne le bon nombre de cartes et qu'à chaque distribution, le nombre de cartes distribuées est corrects (1, 2, 3, ou 4).

```
public enum EtatMain {
            DISTRIBUTION_TERMINEE, DISTRIBUTION_EN_COURS, MALDONNE;    }
```

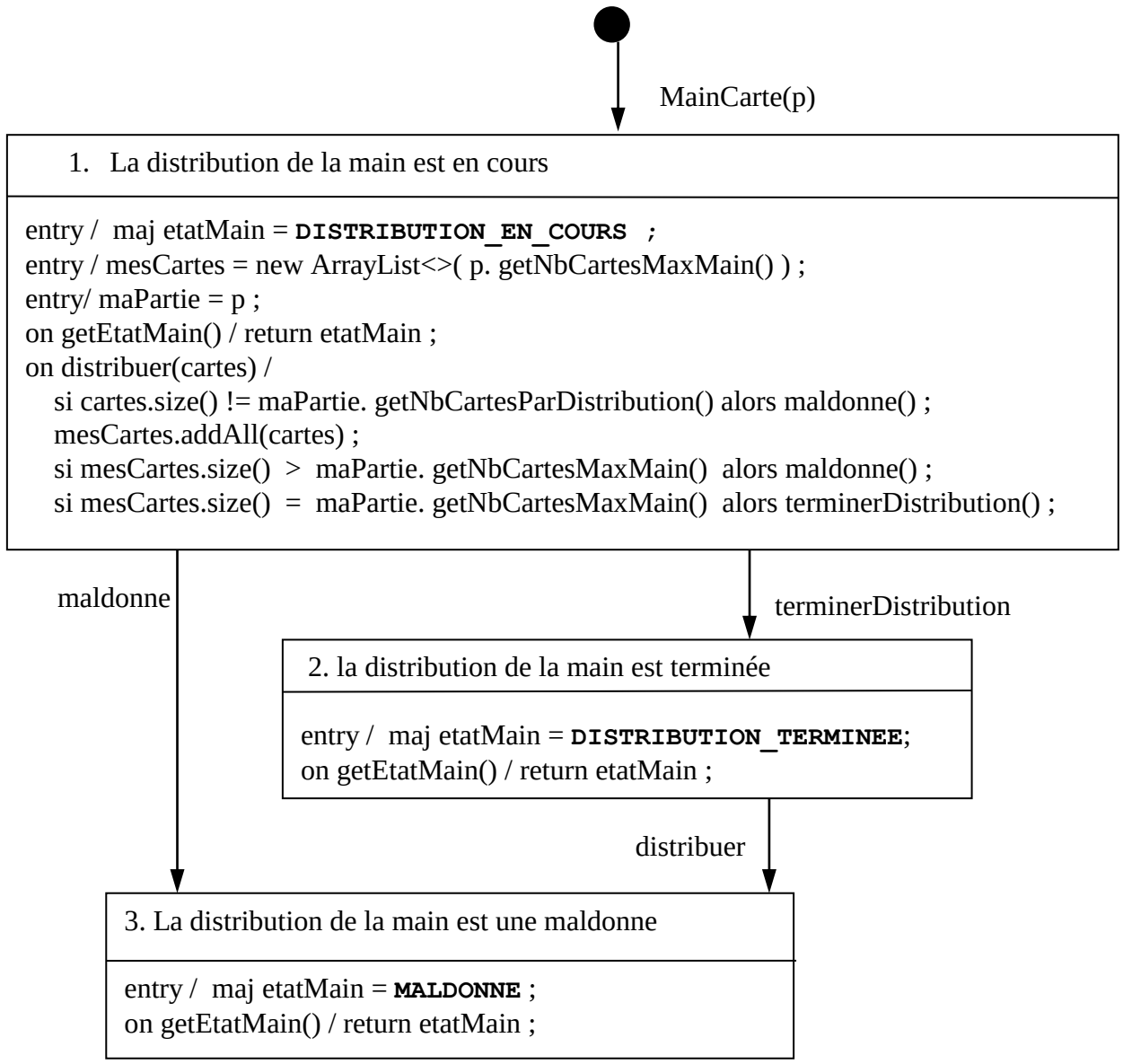
Diagramme de classes :



L'objectif de la méthode `distribuer` est de vérifier la distribution réalisée par le joueur pas d'aider le joueur à éviter une maldonne. La méthode `distribuer` :

- ajoute aux cartes de la main (mesCartes) les cartes de la liste « cartes » (paramètre de la méthode) si cela est conforme aux règles du jeu et si l'état de la main le permet;
- elle met à jours l'état de la main.

Diagramme d'états de la classe MainCartes



Questions –

1. Ecrivez le code de la méthode `distribuer`.
2. Pour proposer un jeu de tests complet pour la méthode `distribuer` de la classe `MainCartes`, il s'agit d'écrire les algorithmes (pseudo-code) du jeu de tests.
    a. Listez les cas d'usage à tester pour la méthode `distribuer`, et pour chaque cas d'usage, listez les valeurs à tester.
    b. Proposez les scénarios de tests et réaliser la méthode `distribuerTest` qui permet de tester la méthode `distribuer`.