Partiel “analyse objet / programmation C++”

M1 finance – P. Laroque

Semestre 2 - session 1 - mai 2011

durée 2h - tous documents autorisés

1 Modélisation UML statique (env. 6 points)

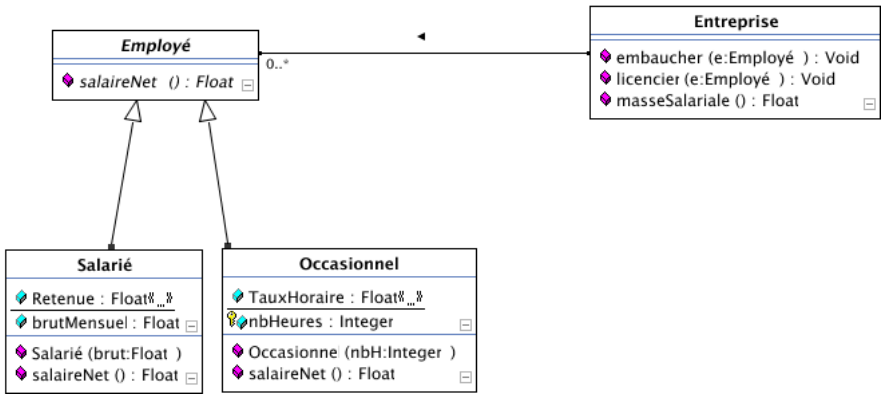
L’entreprise “La mouette rieuse”, spécialisée dans les aliments pour chats, souhaite refaire son système de paie.
Elle emploie deux types de personnes :

1. Les salariés ordinaires, dont le salaire net est calculé après retenue de 23% sur le brut mensuel figurant sur leur contrat.
2. Les employés occasionnels, dont le salaire net est calculé à partir du nombre d’heures effectuées dans le mois et d’une base horaire forfaitaire de 12 euros.

On demande d’écrire un modèle UML simple mais aussi précis que possible (classes, relations, méthodes, attributs) permettant au moins

- l’inscription d’un nouvel employé (salarié ou occasionnel)
- le calcul de la masse salariale nette (la somme de tous les salaires nets) de l’entreprise en fin de mois

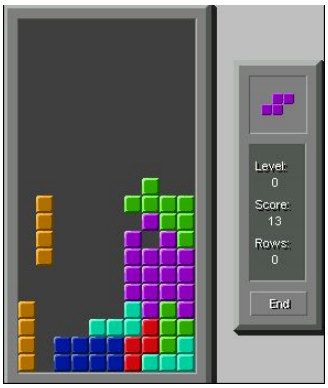
1.1 Corrigé indicatif



2 Modélisation dynamique (env. 6 pts)

On souhaite modéliser dans un statechart la dynamique d’un jeu de type “tetris”. On dispose de deux boutons de direction (un pour diriger la brique qui descend vers la gauche, l’autre vers la droite), ainsi que d’un bouton de démarrage d’une partie. On suppose données les fonctions et variables suivantes :

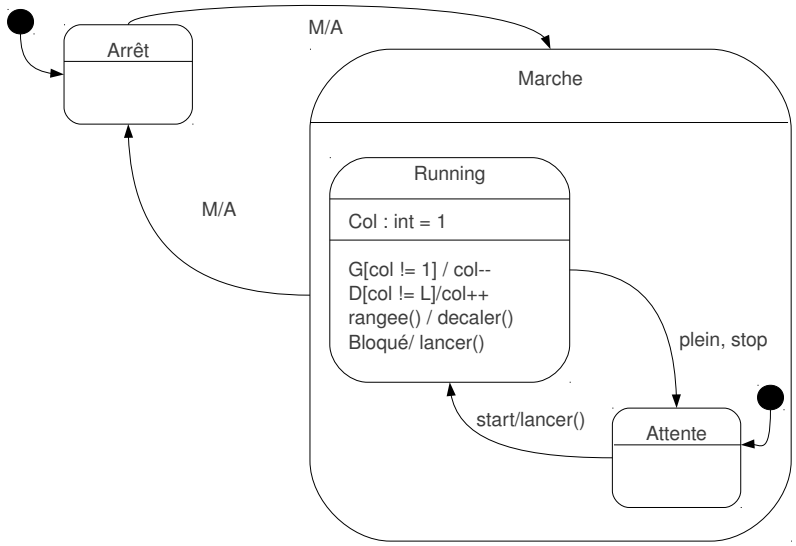
- `rangée()` : prédicat indiquant qu’une ligne est complète (pleine de briques)
- `décaler()` : provoque la disparition d’une ligne complète et le décalage des briques situées au-dessus
- `Ncol` : le nombre de colonnes du jeu



On demande, après avoir défini les événements pertinents et les variables / fonctions supplémentaires qui vous sembleraient nécessaires, de dessiner le statechart de ce jeu.

2.1 Corrigé indicatif

Ajouts / contexte initial :
- Une variable `col`, qui indique le numéro de colonne courant du curseur (entre 1 et N)
- Une action `lancer()` pour commander l'apparition d'un bloc en haut de l'écran
- Un événement `M/A` qui dénote l'appui sur le bouton de marche / arrêt du jeu
- Un bouton de début / fin de partie, qui généère les événements `start` et `stop`
- Un événement `plein` lorsque plus aucun bloc ne peut être lancé



3 Programmation C++ (env. 8 points)

On reprend le contexte de l'exercice 1 pour l'implémentation.

1. Ecrire le code C++ complet de la classe représentant les employés occasionnels.
2. Ecrire la déclaration C++ de la classe représentant l'entreprise
3. Ecrire le code de la méthode de calcul de la masse salariale

3.1 Corrigé indicatif

3.1.1 Code de la classe `Occasionnel`

Fichier `Occasionnel.h`

```
// Occasionnel.h
#ifndef __Occasionnel_h
#define __Occasionnel_h

#include ''Employé.h''

class Occasionnel: public Employé {

 protected:
  int _nbHeures;

 public:
  static float TauxHoraire;
  Occasionnel(int nbH);
  float salaireNet();
};

#endif
```

Fichier `Occasionnel.C`

```
// Occasionnel.C

#include ‘‘Occasionnel.h’’

float Occasionnel :: TauxHoraire = 12;

Occasionnel :: Occasionnel(int nbH) : Employe() {
  _nbHeures = nbH;
}

float Occasionnel :: salaireNet() {
  return _nbHeures * TauxHoraire;
}
```

3.1.2 La classe `Entreprise`

Fichier `Entreprise.h`

Ici, on peut choisir – comme on l'a vu en cours – de représenter l'association unidirectionnelle Entreprise - Employé par un tableau d'employés dans la classe `Entreprise` :

```
// Entreprise.h

#ifndef __Entreprise_h
#define __Entreprise_h

#include ‘‘Employé.h’’

class Entreprise {

 protected:
  Employé ** _employés; // tableau de pointeurs
  int nbEmployés;
  int nbMaxEmployés; // taille du tableau

 public:
  Entreprise();
  void embaucher(Employé& e);
  void licencier(Employé& e);
  float masseSalariale();
};

#endif
```

3.1.3 Méthode de calcul de la masse salariale

```
// Entreprise.C

// ... // autres méthodes

float Entreprise :: masseSalariale() {
  float res = 0;
  for (int i = 0; i < nbEmployés; i++)
    res += employés[i] -> salaireNet();
  return res;
}
```