Partiel “analyse objet / programmation C++”

M1 finance – P. Laroque

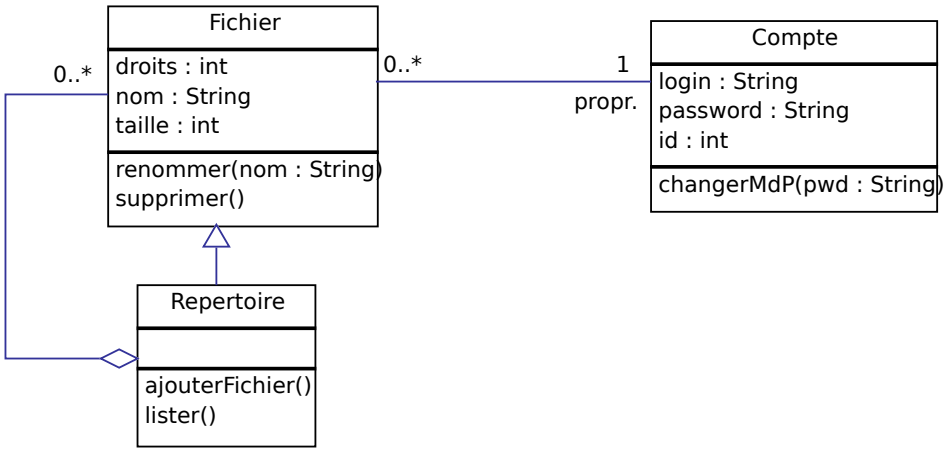
mai 2012

durée 2h - tous documents autorisés

1 Modélisation UML statique (env. 7 points)

Représenter à l’aide d’un schéma UML la structure d’un système de fichiers sur un ordinateur. Les notions de fichier, dossier (ou répertoire), droits d’accès, propriétaire etc. - ainsi que les associations et opérations courantes sur ces objets - devront figurer dans votre modèle.

Correction indicative



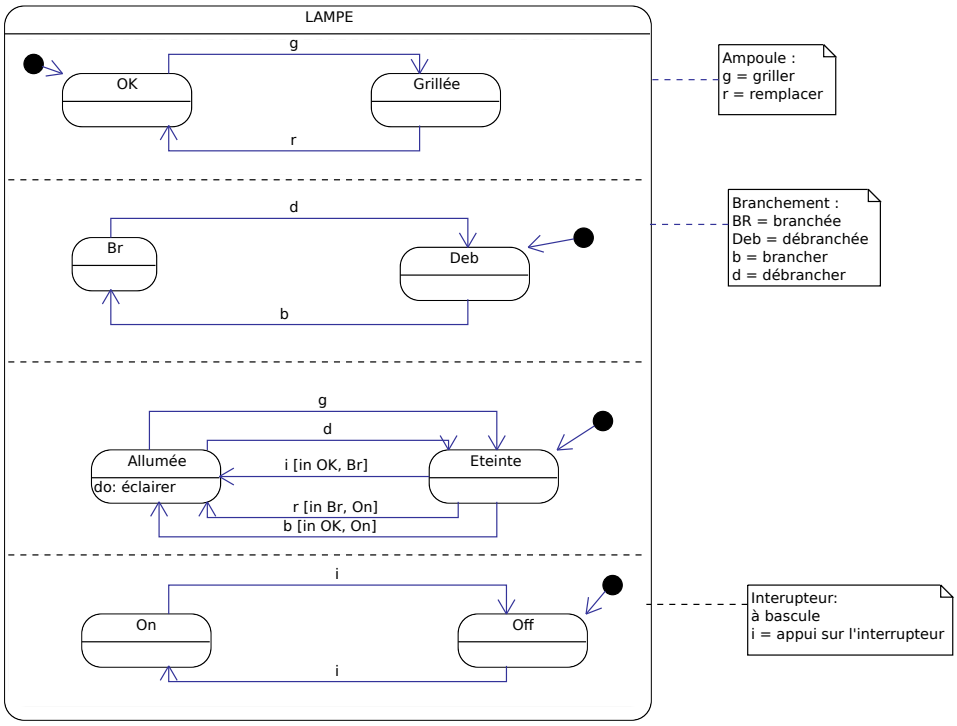
2 Modélisation UML dynamique (env. 7 points)

Reprendre le statechart de la lampe évoqué en cours, en tenant compte des propriétés suivantes :

- la lampe peut être branchée ou non ;
- l’ampoule peut “griller”, elle peut alors être remplacée.

On supposera un bouton de type bascule : si la lampe est allumée, une pression l’éteint, et réciproquement.

Corrigé indicatif



3 Programmation C++ (env. 6 points)

On désire représenter la notion de nombre complexe dans des programmes. On doit donc écrire une classe C++ qui représente cette notion, avec les opérations suivantes :

- le module d'un nombre complexe ;
- l'argument d'un complexe ;
- sa partie réelle, sa partie imaginaire ;
- la somme, le produit de deux nombres complexes.
- la construction d'un nombre complexe à partir de deux nombres réels. On donnera la possibilité de construire un complexe à partir de deux réels (parties réelle et imaginaire), d'un seul réel (partie imaginaire nulle par défaut) ou de rien du tout (ce qui crée un complexe nul).

1. Ecrire le fichier `Complexe.h` correspondant
2. Ecrire le fichier `Complexe.C` qui contient l'implémentation de ces opérations
3. Si vous en avez le temps, imaginez un petit programme de test (`main()`)

Corrigé indicatif

Fichier en-tête Complexe.h

```
/*************************************************************************
 *
 * $Id: Complexe.h 1434 2008-04-29 15:05:09Z phil $
 *
 *************************************************************************/

#ifndef __COMPLEXE_H
#define __COMPLEXE_H

/******************************
 * Partiel M1F 2008: la classe Complexe
 ******************************/

#define PI 3.141592535897

class Complexe {

 protected:
  double _re, _im;
 public:
  Complexe(double re=0, double im=0);
  double module();
  double argument();
  double partieReelle();
  double partieImaginaire();
  Complexe somme(Complexe z);
  Complexe produit(Complexe z);
};

#endif

/*************************************************************************/
```

Fichier source Complexe.C

```
/*************************************************************************
 *
 * $Id: Complexe.C 1434 2008-04-29 15:05:09Z phil $
 *
 *************************************************************************/

static char rcsId[] = "@(#) $Id: Complexe.C 1434 2008-04-29 15:05:09Z phil $";

#include <math.h>
using namespace std;

#include "Complexe.h"

/*************************************************************************
 * Constructeur: 0, 1 ou 2 parametres
```

```
 ************************************************************************/
Complexe :: Complexe (double re, double im) { _re = re; _im = im; }

/************************************************************************
 * calcul du module du complexe
 ************************************************************************/
double Complexe :: module() { return sqrt(_re*_re + _im*_im); }

/************************************************************************
 * calcul de l'argument du complexe
 ************************************************************************/
double Complexe :: argument() { return _re==0? PI/2 : atan(_im/_re); }

/************************************************************************
 * partie reelle du complexe
 ************************************************************************/
double Complexe :: partieReelle() { return _re; }

/************************************************************************
 * partie imaginaire du complexe
 ************************************************************************/
double Complexe :: partieImaginaire() { return _im; }

/************************************************************************
 * calcul de la somme de deux complexes.
 * ATTENTION: operation binaire => un seul parametre (z.somme(z'))
 ************************************************************************/
Complexe Complexe :: somme (Complexe z) {
  Complexe res = z;
  res._re += _re;
  res._im += _im;
  return res;
}

/************************************************************************
 * calcul du produit de deux complexes.
 * ATTENTION: operation binaire => un seul parametre (z.produit(z'))
 ************************************************************************/
Complexe Complexe :: produit(Complexe z) {
  Complexe res;
  res._re = _re*z._re - _im*z._im;
  res._im = _re*z._im + _im*z._re;
  return res;
}

/************************************************************************/
```

Fichier de test tComplexe.C

```
/************************************************************************
```

```
 *
 * $Id: tComplexe.C 1434 2008-04-29 15:05:09Z phil $
 *
 ***************************************************************************/

static char rcsId[] = "@(#) $Id: tComplexe.C 1434 2008-04-29 15:05:09Z phil $";

#include <iostream>
using namespace std;

#include "Complexe.h"

/***************************************************************************
 * fonction simple d'affichage d'un complexe
 ***************************************************************************/
void printComplexe(Complexe z) {
  cout << '('
       << z.partieReelle()
       <<" +i "
       <<z.partieImaginaire()
       <<")\n";
}

/***************************************************************************
 * tests unitaires des methodes de la classe Complexe
 ***************************************************************************/
int main() {
  Complexe z1;                    // = 0
  Complexe z2(2);                 // = 2 + 0i
  Complexe z3(1,1);               // = 1 + i

  printComplexe(z1);
  printComplexe(z2);
  printComplexe(z3);

  cout << "module = " << z3.module() << endl;
  cout << "argument = " << z3.argument() << endl;
  cout << "somme = "; printComplexe(z3.somme(z3));
  cout << "produit = "; printComplexe(z3.produit(z3));
  return 0;
}
/***************************************************************************/
```