

# Examen

Lundi 9 Janvier 2012

Tout document papier est autorisé. Les téléphones portables, ordinateurs, comme tout autre moyen de communication vers vos voisins ou l'extérieur sont strictement interdits. Le temps à disposition est de 3 heures. Justifiez correctement vos réponses (relisez-vous!). Les exercices sont donnés dans le désordre; leur place respective ne reflète pas leur degré de difficulté. Lisez donc le sujet dans son intégralité avant de commencer

## 1 Exercice

```
class JeuDeSociété {
    protected:
        int nombreDeJoueurs;
        virtual void initialiserLeJeu()=0;
        virtual void faireJouer(int joueur)=0;
        virtual bool partieTerminée()=0;
        virtual void proclamerLeVainqueur()=0;
    public:
        void jouerUnePartie(int nombreDeJoueurs) {
            this->nombreDeJoueurs = nombreDeJoueurs;
            initialiserLeJeu();
            int j = 0;
            while( ! partieTerminée() ) {
                faireJouer(j);
                j = (j+1) % nombreDeJoueurs;
            }
            proclamerLeVainqueur();
        }
};
```

Une telle classe est typique d'une implémentation du patron de conception « patron de méthode » (Template method pattern).

1. Peut-on supprimer le mot-clé **virtual**?
2. À quoi sert le `=0` derrière les méthodes marquées **virtual**?
3. Pourquoi la méthode `jouerUnePartie(...)` n'est elle pas déclarée virtuelle?
4. Pourquoi le terme patron/template est-il employé pour nommer cette technique?
5. Puisqu'en C++ les templates sont disponibles, remplacez la définition de la classe abstraite ci-dessus par la définition d'un template de fonction

C++ qui permettrait de réaliser la même chose (i.e. une fonction de prototype void jouer(Jeu &jeu)).

## 2 Exercice

```
class A {
private: int x;
public:
    A(int t=0) {
        x = t;
        cout << "A(" << t << ")" << endl;
    }
    A(const A &a) {
        x = a.x;
        cout << "A(" << a << ")" << endl;
    }
    const A &operator=(const A &a) {
        x = a.x;
        cout << "A=" << a << ")" << endl ;
        return *this;
    }
    friend ostream& operator<<(ostream& os, const A& a) {
        return os << "A.x=" << a.x;
    }
};

class B : public A {
private: A a; int y;
public:
    B (int t=1, int u=2) : a(t), y(u) {
        cout << "B(" << t << ", " << u << ")" << endl;
    }
    B (const A& t, int u=0) : a(t), y(u) {
        cout << "B(" << t << ", " << u << ")" << endl;
    }
    friend ostream& operator<<(ostream& os, const B& b) {
        return os << "(B.a=" << b.a << ", B.y=" << b.y << ")";
    }
};

class C {
private: B b;
public:
    C() {
        cout << "C()" << endl;
    }
    C (int t) : b(t) {
        cout << "C(" << t << ")" << endl;
    }
};
```

```

    }
    C (B t) : b(t,1) {
        cout << "C(" << t << ")" << endl;
    }
};

int main() {
    cout << "AAAAAAA" << endl;
    A a1; A a2 = 1; A a3 = a2;
    cout << "BBBBBBBB" << endl;
    B b1; B b2 (2, 3); B b3 (a2, 4); b2 = b3;
    cout << "CCCCCCCC" << endl;
    C c1; C c2 = 5; C c3 = b2; c3 = c1;
}

```

1. Qu'affiche ce programme?

### 3 Exercice

```

class Gray {
private:
    double level;
    void setLevel(double level) { this->level = level; }
public:
    Gray(double level) { setLevel(level); }
    double getLevel() { return level; }
    static Gray White;
    static Gray Black;
};

int main() {
    cout << Gray::White << endl;
    cout << Gray::Black << endl;
}

```

1. Compléter ce code de sorte que son exécution produise :
 

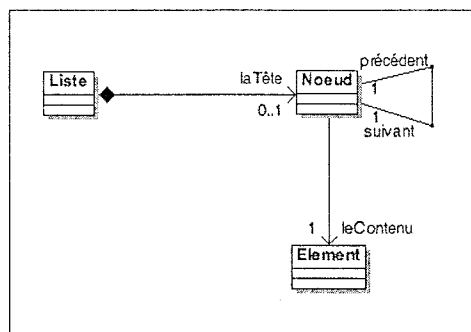
```

1
0

```
2. Qu'est ce qui fait que cette classe est dite immuable (aide : abus de langage car en fait on parle des instances) ?
3. Puisque cette classe est immuable, l'optimisation consistant à assurer que pour un niveau de gris choisi il n'existe jamais plus d'une instance de cette classe représentant le niveau en question est possible. Transformez la classe pour assurer cette propriété.

## 4 Exercice

Soit la définition UML suivante :



1. Implémentez en C++ les classes correspondantes (constructeurs et destructeurs compris). Il est interdit d'utiliser une quelconque structure de donnée prédéfinie (par exemple dans la STL type **vector** ou autre)
2. Implémentez dans la classe **Liste** une méthode `bool estVide()` ; permettant de savoir si une liste contient des éléments ou non.
3. On souhaite s'abstraire du type **Element** :
  - (a) Modifier votre code de sorte que l'on puisse construire une liste de n'importe quoi
  - (b) Modifier votre code de sorte que l'on ne puisse construire que des listes d'objets du même type

## 5 Exercice

1. Établir un diagramme UML permettant d'exprimer le fait que l'on désire manipuler des matrices d'entiers à deux dimensions de sorte qu'il soit possible : de modifier la taille d'une matrice, de manipuler les éléments individuellement et repérés par leurs coordonnées et d'afficher une matrice. Dans cette question on ne se préoccupe pas du langage de programmation qui sera utilisé...
2. Écrire une implémentation C++ du diagramme précédent en justifiant les choix d'implémentation (polymorphisme, accesseurs, etc).
3. On souhaite pouvoir écrire en C++ des choses telles que  
Matrice A, B(10,20), C(10,20); A = B+2\*C;  
Qu'elles sont les modifications à apporter au code pour que cela puisse fonctionner ? Répondre en fournissant le code C++.