Programmation Objet, Concepts Avancés
Université Paris Diderot – Paris 7 – Master 2
Examen

Durée : 1 heure 50

Tout document autorisé. Matériel électronique interdit.

18 décembre 2009

Résolution d'appel de méthodes en C++ (10 min)

Voici un programme écrit en C++ :

```
#include <iostream>
class A {
public:
  virtual void m1 () = 0;
  virtual void m3 (int x) = 0;
  void m4 () { std::cout << "A::m4()" << std::endl; }
};

class B : public virtual A {
public:
  virtual void m1 () { std::cout << "B::m1()" << std::endl; }
  virtual void m3 () { std::cout << "B::m3()" << std::endl; }
  void m4 () { std::cout << "B::m4()" << std::endl; }
};

class C : public virtual A {
public:
  virtual void m1 () { std::cout << "C::m1()" << std::endl; }
  virtual void m2 () { std::cout << "C::m2()" << std::endl; }
};

class D : public B, public C {
public:
  virtual void m1 () { std::cout << "D::m1()" << std::endl; }
  virtual void m3 (int x) { std::cout << "D::m3()" << std::endl; }
  void m4 () { std::cout << "D::m4()" << std::endl; }
};
```

Pour chacune des définitions suivantes de la fonction `show`, indiquez si le compilateur accepte le programme et le cas échéant, ce qui est affiché sur la sortie standard. Si vous pensez que le compilateur rejette le programme, expliquez pourquoi.

1. `void show () { D d; d.m1 (); }`
2. `void show () { D d; d.m2 (); }`
3. `void show () { D d; d.m3 (); }`
4. `void show () { D d; A* a = &d; a→m4 (); }`

Une interface de dictionnaire en Scala (20 min)

Voici un programme écrit en SCALA :

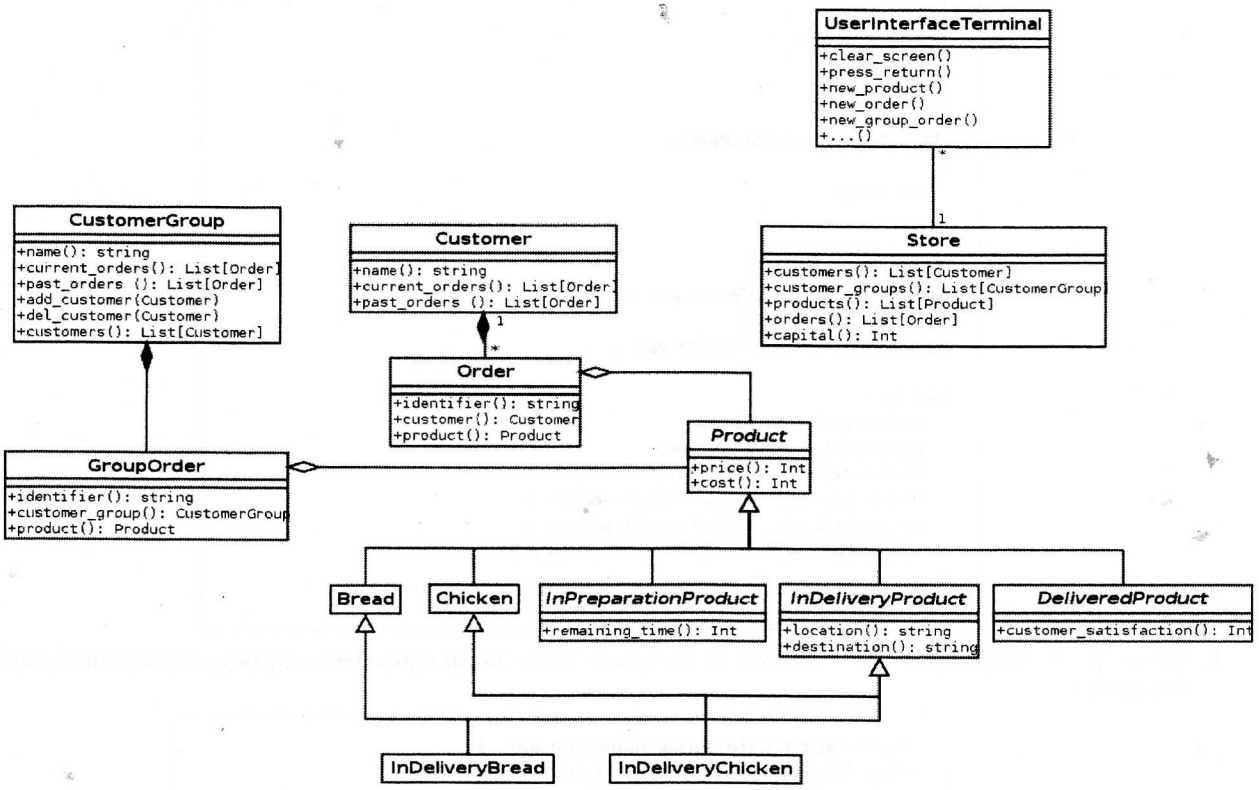
```
abstract class Dict {
    type Key
    type Value
    // Precondition :mem (k) == false
    // Postcondition :assoc (k) == v /\ mem (k) == true
    def add (k :Key, v :Value)
    // Precondition :mem (k) == true
    // Postcondition :true
    def assoc (k :Key) :Value
    // Precondition :true
    // Postcondition :result == true ⇒ exists v, assoc (k) == v
    def mem (k :Key) :Boolean
}
```

1. Que signifient ici les déclarations "type Key" et "type Value" ?
2. Décrivez (en SCALA ou en français) ce que pourrait être une sous-classe concrète de Dict.
3. On veut définir une classe "FlexibleDict", similaire à la classe "Dict" mais autorisant des multiples appels à la méthode "add" avec la même clé k. Lors d'un appel de méthode de la forme "add k v", si il existe déjà dans le dictionnaire une association entre k et une valeur v' alors elle est masquée par une nouvelle association entre k et v. Doit-on faire hériter cette nouvelle classe de la classe "Dict" définie plus haut ? Pour répondre à cette question, vous commencerez par définir cette classe "FlexibleDict" et vous écrirez sa spécification.
4. On veut maintenant étendre la classe "Dict" à l'aide d'une méthode "upper" qui attend une clé k et renvoie la liste des valeurs associées aux clés du dictionnaire qui sont plus grandes que k. Comment doit-on écrire cette classe ? En particulier, comment exprimer l'existence d'une relation d'ordre sur les clés ?

Modélisation objet d'un commerce (50 min)

On souhaite développer un logiciel de gestion des commandes d'un petit commerce (boulanger, boucher, épicier, ...). Ce logiciel doit permettre la gestion du cycle de vie des commandes. Les commandes sont initiées par des clients ou des groupes de clients et portent sur des produits (pain, poulet, ...). Le cycle de vie d'une commande commence par la préparation, se poursuit par la livraison et enfin, se termine par le service après-vente (comprenant par exemple une étude de satisfaction du client).

Une (mauvaise) étude de conception a mené au diagramme statique de classes suivant :



1. Évaluez la qualité de cette modélisation (en structurant votre réponse). Pour vous guider, voici une liste (non exhaustive) de questions auxquelles vous pouvez confronter ce système :
   - Est-ce que les concepts utilisés vous semblent avoir été précisément définis ?
   - Quelle classe doit-on instancier pour représenter un poulet en cours de préparation ?
   - Quelles sont les classes à écrire pour prendre en compte un nouveau produit dans le système ?
   - La duplication des méthodes de la classe `Order` dans la classe `GroupOrder` était-elle inévitable ?
   - Peut-on facilement intégrer une notion de stock (ensemble de produits en réserve) dans le logiciel ?
   - Peut-on facilement étendre le logiciel avec un système de notifications, indiquant à l'utilisateur le passage d'étape d'une commande (passage de la préparation à la livraison par exemple) ?
   - Peut-on facilement transformer ce logiciel de gestion de commerce en un site de commerce électronique (qui propose une interface utilisateur *via* un navigateur WEB) ?
2. Proposez un diagramme de classes corrigeant les défauts exhibés par votre réponse à la question précédente. Argumentez vos choix. Si vous utilisez des patrons de conception vus en cours, expliquez leurs pertinences dans votre modélisation.

Multi-méthodes et le patron de conception "Visiteur" (20 min)

Le langage de programmation NICE est un langage de programmation orienté objet, s'appuyant sur des classes et des **multi-méthodes**. La résolution d'appel d'une multi-méthode, appelée délégation multiple (*multiple dispatch*), prend en compte le type dynamique de plusieurs objets à la fois et non pas le type d'un seul objet comme dans le cas des méthodes classiques, dites à délégation simple (*single dispatch*).

Voici un exemple de programme écrit en NICE :

```
class Person
{
  String name;
  int age;
}

class Worker extends Person
{
  int salary;
}

// equals : version 1.
equals (Person p1, Person p2) = ..
// equals : version 2.
equals (Worker w1, Worker w2) = ...

test () {
  Person p = new Person;
  Person w1 = new Worker;
  Person w2 = new Worker;
  equals (p, w1); // Call equals version 1.
  equals (w1, p); // Call equals version 1.
  equals (w1, w2); // Call equals version 2.
}
```

1. Est-ce que le programme C++ suivant se comporte d'une façon équivalente au programme NICE précédent ? Pourquoi ?

```
struct Person { std::string name; int age; };
struct Worker : public Person { int salary };
bool equals (Person* p1, Person* p2) { ... }
bool equals (Worker* w1, Worker* w2) { ... }
void test () {
  Person* p = new Person ();
  Person* w1 = new Worker ();
  Person* w2 = new Worker ();
  equals (p, w1);
  equals (w1, p);
  equals (w1, w2);
}
```

2. Montrez comment le patron de conception de "Visiteur" permet de simuler la délégation multiple en C++. (Indice : n'hésitez pas à définir plusieurs visiteurs.)