Maîtrise d'Informatique
Module POD

Programmation Objet et Distribution
Documents autorisés

Université Pierre et Marie Curie
Année 1997–1998
Durée 3 heures

Examen du 11 septembre 1998

Exercice 1 : Redéfinition et Surcharge de méthodes

Cet exercice a pour but de bien différencier ces 2 notions sur une série d'exemples.
Soient les définitions, en pseudo-Java, des classes suivantes :

```
classe C
  void M1 (int x)  {...}
  void M2 (int x,int y) { ... this.M1(y) ...}

classe SC sous-classe de C
  void M1(int x) { ...}
  void M1(int x, String s) { ...}

classe S_SC sous-classe de SC
  void M1(int x){ ...}
  void M2(int x, int y){ ... this.M1(y) ... }
```

1. Écrire la classe C et ses sous-classes SC et `S_SC` en Java. Le corps des méthodes affichera en entrée le nom de la classe, le nom de la méthode et les arguments. La méthode `M2` de `C` envoie un message `M1` sur *this*.
2. Écrire la classe `C` en O'Caml.
3. Pourquoi ne peut-on pas écrire la classe `SC`?
4. Qu'affichera le programme Java suivant :

```
C c1 = new C();
c1.M1(3);
c1.M2(1,2);
SC sc1 = new SC();
sc1.M1(4);
sc1.M1(5,"truc");
sc1.M2(3,4);
C c2 = sc1;
c2.M1(7);
c2.M2(9,3);
S_SC s_sc1 = new S_SC();
s_sc1.M1(10);
s_sc1.M2(11,12);
C c3 = s_sc1;
c3.M1(14);
c3.M2(15,16);
```

Exercice 2 : clients et serveur pour le Web

Cet exercice consiste en la réalisation d'un client de recherche d'images sur une page HTML (Hyper Text Markup Language : le langage de description de documents du Web) réalisé en O'Caml et d'un mini-serveur pour le protocole HTTP (Hyper Text Transfert Protocol: le protocole utilisé pour le Web) réalisé en Java.

---

coté clients en O'Caml

1. Écrire une classe `client_http` en O'Caml "générique" qui se connecte à un serveur HTTP : `machinename:80`, envoie une requête `GET filename\n\n` et récupère dans une chaîne de caractères toute la page HTML transférée par le serveur. Cette chaîne est un des champs de données de cette classe.

2. Écrire une méthode `sauve` qui écrit cette chaîne dans un fichier dont le nom est donné en paramètre.

3. En héritant de `client_http`, écrire une sous-classe `client_images` qui va construire une liste des noms des fichiers images contenus dans le texte HTML.
   On rappelle, en le simplifiant, qu'un nom d'image est codé de la manière suivante en HTML :

   ```
   <IMG SRC="path1/path2/filename">
   ```

   Il existe d'autres paramètres possibles pour le "tag" `<IMG ..>`, mais on n'en tiendra pas compte. De même on supposera les "tags" toujours en majuscules. On cherche à extraire la chaîne de caractères `"path1/path2/filename"` du tag `IMG`.
   Écrire une méthode `extraire` qui construit une liste des noms d'images qui sera conservée dans un champs de données de la classe. On utilisera les fonctions sur les `string` d'O'Caml.

4. Ajouter une méthode `simplify` qui ne conserve que des occurrences uniques dans la liste des noms d'images.

5. Écrire une méthode `charger_image` qui à partir d'un nom d'image, ouvre une connexion vers le serveur HTTP avec la requête `GET nom_image`.

6. Écrire une méthode `cest` (pour `charge_et_sauve_tout`) qui à partir de cette liste de noms d'images, les charge et les sauve une à une dans différents fichiers. On utilisera les fonctions de base des entrées/sorties d'O'Caml pour l'écriture dans un canal.

---

coté serveur en Java

7. Écrire un mini-serveur HTTP en Java. Il ne saura répondre qu'à la requête :`GET filename`, sachant que `filename` peut contenir un chemin d'accès. Le port par défaut sera le port 80. À une requête le serveur enverra le contenu du fichier indiqué dans `filename` et fermera la connexion.

Exercice 3 : Producteur/Consommateur en RMI

Cet exercice consiste à implanter un Producteur/Consommateur. Voici une version avec threads sans RMI :

```
public class ProdCon {

  public static void main (String [] args) {
     Shop myShop = new Shop(12);

     Producer p = new Producer(myShop);
     p. start();

     Consumer c1 = new Consumer(myShop,"C11");
     c1.start();
     Consumer c2 = new Consumer(myShop,"C12");
     c2.start();
     Consumer c3 = new Consumer(myShop,"C13");
     c3.start();
  }
}  // end class ProdCon

class Product {

  String name;

  Product () {name = "Standard";}
  Product (String n) {name = n;}

  String getName() {return name;}
```

```
}  // end class Product

class Shop {
  protected int size=8;
  protected Product [] buffer;
  protected int ip = 0;
  protected int ic = 0;


  Shop () { emptyShop();}

  Shop(int n){size=n;emptyShop();}

  void emptyShop() {
    buffer = new Product[size];
    for (int i=0; i< size; i++) {
      buffer[i]=new Product("Empty");
    }
  }

  void display1() {
    System.out.println("enter product :  [("+(ip % size)+")] "+buffer[ip % size].getName());
  }

  void display2() {
    System.out.println("exit product :  [("+(ic % size)+")] "+buffer[ic % size].getName());
  }

  synchronized void put(Product p) {
    while (ip-ic+1 > buffer.length) {
      try{wait();}
      catch(Exception e) {}
    }
    buffer[ip % size] = p;
    display1();
    ip++;
    notify();
  }


  synchronized Product get() {
    while (ip == ic) {
      try{wait();} catch(Exception e) {}
    }
    display2();
    notify();
    return buffer[ic++%size];
  }


}  // end class Shop


class Consumer extends Thread {

  Shop aShop;
  String name;

  Consumer(Shop s, String sn){aShop=s; name = sn;}
```

```
  public void run() {
    while(true) {
        Product p = aShop.get();
        display(p);
        try {sleep((int)(1000*Math.random()));}
        catch (Exception e0) {}
    }
  }

  void display(Product p) {
        System.out.println("Get :  "+p.getName()+ " from "+name);
  }

}  // end class Consumer

class Producer extends Thread {

  Shop aShop;
  int num = 0;

  Producer(Shop s) {aShop=s;}

  public void run() {
    while(true) {
      Product p = makeProduct();
      aShop.put(p);
      display(p);
      try {sleep((int)(333*Math.random()));}
      catch (Exception e0){}
    }
  }

  void display(Product p) {
        System.out.println("Put :"+p.getName());
  }

  Product makeProduct() {
    Product p = new Product("Product"+num);
    num++;
    return p;
  }

}  //end class Producer
```

1. Donner les différentes classes de ce programme et leurs possibles liens.
2. Décrire ce que fait ce programme, et indiquer le début possible des affichages.
3. On désire avoir des instances de `Shop` accessibles par le réseau en utilisant le mécanisme de RMI. Les consommateurs et les producteurs pourront invoquer des méthodes distantes d'instances de `Shop` du serveur. Indiquer de manière générale comment vous adapteriez ce programme pour cela?
4. Décrire l'interface RMI de ces nouvelles `Shop`.
5. Indiquer les modifications à apporter aux classes `Consumer` et `Producer`.
6. Écrire le serveur.
7. Quelles remarques vous inspirent ce programme en RMI?