Architectures Avancées des Systèmes d'Information

Durée : 1h 30

Exercice 1

Dans les exemples suivants, quel Design Pattern peut-on utiliser ?

1. On souhaite développer un éditeur HTML de texte (String). Les types de formatages envisagés sont : italic (…), bold (<u>…</u>), boldItalic (…), etc.

➔ **Decorator**

2. Dans une interface graphique (swing par exemple), on souhaite repérer deux types de boutons fréquemment utilisé dans toutes les fenêtres :

A. Un bouton « Exit » avec son comportement permettant de fermer une fenêtre.

B. Un bouton « Valider » avec son comportement permettant de valider un traitement en se basant sur des informations supplémentaires passées en paramètres.

➔ **Flyweight**

C. Chacun des membres de la compagnie reçoit un salaire.
• À tout moment, il doit être possible de demander le coût d'un employé.
• Le coût d'un employé est calculé par :
– Le coût d'un individu est son salaire.
– Le coût d'un responsable est son salaire plus celui de ses subordonnés.

➔ **Composite**

Exercice 2

Nous disposons d'un service qui représente un contrôleur de température TemperatureSensor qui offre les opérations de gestion de la température d'un système industriel. Les méthodes offertes par ce service sont les suivantes :

```
void augmenterTemp (double tempVal) { }
void diminuerTemp (double tempVal) { }
double lire_temp () { }
```

1. Quelles sont les étapes nécessaires pour le développement d'une application distribuée avec RMI ?

➔ **Réponse : 4 étapes essentielles: interface, implémentation de l'interface, serveur, client**

2. On souhaite rendre chacune de ces méthodes accessibles à distance de manière à ce qu'elles définissent l'interface TemperatureSensorInterface entre le client et le serveur. Ecrire cette interface.

➔ **interface TemperatureSensorInterface**

```
public interface TemperatureSensorInterface extends Remote {
        void augmenterTemp (double tempVal) throws java.rmi.RemoteException;
        void diminuerTemp (double tempVal) throws java.rmi.RemoteException;
        double lire_temp () throws java.rmi.RemoteException;
};
```

3. Déduire la classe TemperatureSensor qui matérialise le service qui offre les opérations augmenterTemp(), diminuerTemp() et lire_temp().

➔ **classe TemperatureSensor**

```
public class TemperatureSensor extends UnicastRemoteObject implements
TemperatureSensorInterface {
	private double temp;
	public TemperatureSensor (double t) throws java.rmi.RemoteException {
		super();
		temp=t;
	}
	public void augmenterTemp(double tempVal) throws java.rmi.RemoteException {
		temp = temp+tempVal;
	}
	public void diminuerTemp (double tempVal) throws java.rmi.RemoteException {
		temp=temp-tempVal;
	}
	public double lire_temp() throws java.rmi.RemoteException {
		return temp;
	}
}
```

4. Donner le programme du serveur Serveur.java qui doit être installé sur la machine services.isi.tn sachant que le service de noms doit être activé sur le port numéro 2003.

➔ **classe Serveur**

```
public class Serveur {
	public static void main(String[] args) {
	try {
		System.out.println("Serveur : Construction de l'implémentation");
		TemperatureSensor ts= new TemperatureSensor(15.50);
		System.out.println("Objet TemperatureSensor enregistré dans
		RMIregistry");
		Naming.rebind("rmi:// services.isi.tn:2003/SensorCourant",ts);
		System.out.println("Attente des invocations des clients ");
	}
	catch (Exception e) {
		System.out.println(e.getMessage());
	}
	}
}
```

Exercice 3

Design Pattern à utiliser est Strategy

Code : Même Logique que l'exercice 2 en utilisant protocole RMI