INSTITUT SUPERIEUR INFORMATIQUE ISI المعهد العالي للإعلامية		
Devoir surveillé ■	Examen ☐	Session : Principale ■ de Rattrapage ☐
Niveau d'étude : 3ème Licence SIL **Matière :** Développement d'Applications Réparties **Nombre de pages :** 2 **Chargés de cours :** Amen Ben Hadj Ali		**Semestre :** 1 **Date :** 18/11/2011 **Durée :** 1h30 **Documents :** autorisés ☐ Non autorisés ■

Exercice 1 :

Nous disposons de la classe **Livre** qui permet de représenter un livre électronique accessible par plusieurs lecteurs. La structure de cette classe est décrite comme suit :

```
Class Livre {
    void Lire(String nomLecteur, String paragraphe, Thread t){
    System.out.println("livre demandé par lecteur "+ nomLecteur);
    System.out.println(nomLecteur + ″ lit : ″ + paragraphe );
}
```

1. Donner la structure et le code de la classe **Lecteur** qui représente un lecteur du livre. Un lecteur est défini par son **nom**, le **Livre** dans lequel il va lire et le **passage** à lire (une chaine de caractère qui représente le paragraphe lu par le lecteur).
2. On suppose que lorsqu'un lecteur utilise le livre, ce dernier n'est pas disponible pour un autre lecteur. Modifier la méthode Lire() pour qu'un Livre soit utilisable par un seul lecteur à la fois c'est à dire que chaque lecteur doit attendre la fin de lecture du lecteur qui utilise le livre (on suppose que le temps de lecture d'un paragraphe est égal à 1 minute).
3. Ecrivez en java un programme qui utilise trois lecteurs, leur permet de lire dans un livre en alternance et d'afficher le nom du lecteur qui prend la parole. Le programme doit attendre la fin de lecture de tous les lecteurs avant de se terminer.
4. Peut-on gérer l'ordre d'exécution des threads? par qui les threads sont gérés dans la machine?

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.

Réponse : La structure de la classe interface **TemperatureSensorInterface** est la suivante. Tous les champs en gras sont obligatoires.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface TemperatureSensorInterface extends java.rmi.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().

Réponse : La structure de la classe d'implémentation **TemperatureSensor** est la suivante. Tous les champs en gras sont obligatoires.

```
import java.rmi.*;
import java.rmi.server.*;
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.

Réponse : La classe Serveur est la suivante :

```
import java.rmi.*;
import java.rmi.server.*;
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("Erreur de liaison de l'objet TemperatureSensor ");
System.out.println(e.toString());
}
}
}
```

4. Compléter le programme du client **Client.java** qui doit être lancé à partir d'une autre machine.

```
import java.io.*;
import java.rmi.*;
class Client
{
public static void main (String [] argv) throws IOException
{
if(argv.length != 2){
        System.out.println("Usage : java Client <nombre><operation>");
        System.exit(1);
        }
// operation = 1: augmenter, 2: diminuer
double valeur = Double.parseDouble(argv[0]);
int operation = Integer.parseInt(argv[1]);
try {
        // à compléter …

        // afficher la température courante …
   }
catch (Exception e) {
System.out.println("Erreur d'accès à un objet distant");
System.out.println(e.toString());}}}
```

Réponse:

```
import java.io.*;
import java.rmi.*;
class Client
{
public static void main (String [] argv) throws IOException
{
if(argv.length != 2){
System.out.println("Usage : java Client <nombre>
<operation>");
System.exit(1);
}
// operation = 1: augmenter, 2: diminuer
System.setSecurityManager(new RMISecurityManager());
double valeur = Double.parseDouble(argv[0]);
int operation = Integer.parseInt(argv[1]);
```

```
try {
TemperatureSensorInterface ts= (TemperatureSensorInterface)
Naming.lookup ("rmi://services.isi.tn:2003/SensorCourant");
if (operation==1) ts.augmenter(valeur);
if (operation ==2) ts.diminuer(valeur);
System.out.println ("La température courante = " +
ts.lire_temp() + " degrés");
}catch (Exception e) {
System.out.println("Erreur d'accès a un objet distant");
System.out.println(e.toString());
}
}
}
```

5. **Question Bonus**: Schématiser l'architecture logique de l'application développée en spécifiant le rôle de chaque partie.

(voir cours)