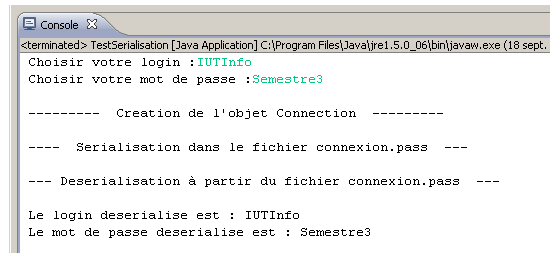


TD JAVA n°5: E/S en JAVA : Sérialisation

Exercice 1 : Sérialisation d'un objet

1. En respectant le principe d'encapsulation, écrire la classe **Connection** qui a deux attributs de type **String**: login et motPasse
2. Transformer cette classe en classe « sérializable ».
3. Ecrire une application **TestSerialisation** qui permet :
 - de créer un objet de type **Connection** à partir des valeurs pour le login et le motPasse saisies comme le montre la copie d'écran ci-dessous.
 - de mettre en œuvre la sérialisation de cet objet dans le fichier **connexion.pass**
 - de mettre en œuvre la désérialisation à partir du fichier **connexion.pass**
 - d'afficher l'objet désérialisé



```
<terminated> TestSerialisation [Java Application] C:\Program Files\Java\jre1.5.0_06\bin\javaw.exe (18 sept.)
Choisir votre login : IUTInfo
Choisir votre mot de passe : Semestre3

----- Creation de l'objet Connection -----
---- Serialisation dans le fichier connexion.pass ---
--- Deserialisation à partir du fichier connexion.pass ---

Le login deserialise est : IUTInfo
Le mot de passe deserialise est : Semestre3
```

Exercice 2 : CabinetMedical : Sérialisation/Désérialisation d'une liste de Personne

Nous souhaitons maintenant sérialiser/désérialiser dans un fichier une liste d'objets de type **Personne**

1. Comment modifier le projet cabinetMedical déjà existant pour permettre la sérialisation/désérialisation ?
2. Notez bien qu'il n'est pas nécessaire de sérialiser les objets les uns après les autres, mais qu'il est possible de *sérialiser directement un objet de type Collection* dans le fichier. En effet, la sérialisation *sauvegarde tout le graphe des objets*. Ainsi, lorsqu'un objet est sérialisé, tous les objets que ses variables d'instance référencent le sont aussi, et tous les objets que ces objets référencent le sont aussi, etc...En sérialisant directement un objet de type **Collection<Personne>**, on sérialise bien sûr tous les objets de type **Personne** qu'il contient (qu'ils soient de type **Patient** ou de type **Professionnel**) ainsi que tous les objets de type **Adresse** qui leur sont associés.

Afin de mettre en place la sérialisation dans le projet cabinetMedical, écrire une classe

PersonneDAOFichier qui possède les deux méthodes suivantes :

- **public static void** storeAllPersonnes (**Collection<Personne>** uneListe)
qui permettra de sérialiser une **Collection<Personne>** dans le fichier **cabMedPersonne.data**
- **public static** **Collection<Personne>** findAllPersonnes ()
qui permettra de désérialiser une **Collection<Personne>** à partir du fichier **cabMedPersonne.data**

Remarque : On souhaite faire en sorte que la méthode **findAllPersonnes()** renvoie toujours une **référence valide sur une Collection**...c-a-d que s'il y a un problème quelconque lors de la **désérialisation**, on ne renverra jamais une référence nulle (**null**), mais bien une référence sur une **Collection** (vide dans cas cas)

3. Mettre en place la Sérialisation/Désérialisation dans le fichier suivant :

```
public class EssaiCabMed_v3 {

    public static void main(String[] args) {
        new EssaiCabMed_v3 ();
    }

    EssaiCabMed_v3 ()
    {

        // Désérialisation---
        // Récupération des données du fichier CabMedPersonne.data
        // dans une liste de Personne
        ..... à vous de coder .....

        // Manipulation de la liste de Personne
        ..... Par exemple le code du TP précédent .....

        // Sérialisation de la liste de personne
        // dans le fichier CabMedPersonne.data
        ..... à vous de coder .....

    }

}
```

Correction TD JAVA n°4: E/S en JAVA : Sérialisation

Exercice 1 : Interface

1. Classe Connection :

```
public class Connection{

    // Attributs
    private String login;
    private String motPasse;

    //Constructeurs
    public Connection()
    { }

    public Connection(String login, String motPasse)
    {
        this.login = login;
        this.motPasse = motPasse;
    }

    // Méthodes
    public String getLogin(){
        return login;
    }

    public String getMotPasse(){
        return motPasse;
    }

    public void setLogin(String newLogin){
        this.login=newLogin;
    }

    public void setMotPasse(String newMotPasse){
        this.motPasse = newMotPasse;
    }

}

2. Classe Connection qui peut être sérialisée :

import java.io.Serializable; // ne pas oublier le import ! ! !

public class Connection implements Serializable{

    private static final long serialVersionUID = 1L;
    // facultatif
    // permet d'éviter le warning sous Eclipse...

    ..... code écrit précédemment .....
}
```

3. Ecrire une application **TestSerialisation** qui permet :

```
import java.io.*;
import java.util.Scanner;

public class TestSerialisation{

    public static void main (String args[]){
        new TestSerialisation();
    }

    public TestSerialisation() {
        System.out.print (" Choisir votre login :");
        Scanner sc = new Scanner(System.in);//Nouvel objet Scanner
        String monLogin = sc.next(); // Récupération de la chaîne de caractère

        System.out.print (" Choisir votre mot de passe :");
        sc = new Scanner(System.in);//Nouvel objet Scanner
        String monMotPasse = sc.next(); // Récupération de la chaîne de caractère

        //Creation de l'objet Connection
        System.out.println (" \n ----- Creation de l'objet Connection -----");
        Connection maConnection = new Connection(monLogin,monMotPasse);

        // Serialisation dans le fichier connexion.pass
        System.out.println (" \n ---- Serialisation dans le fichier connexion.pass ----");
        try
        {
            FileOutputStream fichier = new FileOutputStream("connexion.pass");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(maConnection);
            oos.close();
        }
        catch (IOException e)
        {
            System.out.println("Erreur !!");
        }

        // Désérialisation à partir du fichier connexion.pass
        System.out.println(" \n --- Deserialisation à partir du fichier connexion.pass ---");
        try
        {
            FileInputStream fichier = new FileInputStream("connexion.pass");
            ObjectInputStream ois = new ObjectInputStream(fichier);
            Connection connectRecup = (Connection) ois.readObject();
            ois.close();

            // Affichage dans le try, PAS après ...
            // car il y aura eu garbage collector sur connectRecup...
            // En effet, avec ce code, connectRecup est une variable locale de portée try{...}
            System.out.println("\n Le login deserialise est : " +
                               connectRecup.getLogin());

            System.out.println(" Le mot de passe deserialise est : " +
                               connectRecup.getMotPasse());
        }
        catch (IOException e)
        {
            System.out.println("Erreur !!");
        }
        catch (ClassNotFoundException e)
        {
            System.out.println("Erreur de classe !!");
        }

    } // fin constructeur

} // fin classe
```

Remarque :

Pour faire l'affichage après le bloc `try`, voilà ce qu'il aurait fallu écrire !!!

1^{ère} solution :

```
public class TestSerialisation{

public static void main (String args[]){
    new TestSerialisation();
}

public TestSerialisation() {
    // ... code identique à précédemment ...

// Désérialisation à partir du fichier connexion.pass
System.out.println(" \n --- Deserialisation à partir du fichier connexion.pass ---");
Connection connectRecup = new Connection("", "");
try
{
    FileInputStream fichier = new FileInputStream("connexion.pass");
    ObjectInputStream ois = new ObjectInputStream(fichier);
    connectRecup = (Connection) ois.readObject();
    ois.close();
}
catch (IOException e)
{
    System.out.println("Erreur !!");
}
catch (ClassNotFoundException e)
{
    System.out.println("Erreur de classe !!");
}

// Affichage après le try possible dans ce cas ...
System.out.println("\n Le login deserialise est : " +
                    connectRecup.getLogin());
System.out.println(" Le mot de passe deserialise est : " +
                    connectRecup.getMotPasse());
} // fin constructeur
} // fin classe
```

Remarque :

→ Il faut absolument écrire : `Connection connectRecup = new Connection("", "");` ;
→ Mais surtout pas juste : `Connection connectRecup ;` => il n'y aurait pas de compilation

parce que l'initialisation de `connectRecup` se fait dans un bloc `try` qui est susceptible de ne pas être exécuté entièrement.

- 2^{ème} solution :

Il faut déclarer `ConnectionRecup` comme attribut (variable globale) et il est impératif de passer par un constructeur ...

On n'aurait pas pu écrire directement le code dans le : `public static void main (String args[])`

La variable `ConnectionRecup` (globale) n'aurait pas été accessible dans une méthode statique ...

Remarque : *pour éviter la fuite de ressources* (fichiers tronqués), on pourrait **fermer le flux dans un bloc `finally`** (c'est ce qu'on fera avec `JDBC`, on fermera les flux dans un bloc `finally`)...

Bien sûr cela complexifie un peu le code ... Les étudiants pourront essayer cette solution en TP ...

Exemple avec la sérialisation :

// Serialisation dans le fichier connexion.pass

```
ObjectOutputStream oos=null; // déclaration et initialisation de l'objet oos
try
{
    FileOutputStream fichier = new FileOutputStream("connexion.pass");
    oos = new ObjectOutputStream(fichier);
    oos.writeObject(maConnection);
}
catch (IOException e)
{
    System.out.println("Erreur !!");
}
finally{
    // bloc finally... dans lequel on trouve
    if (oos!=null) // ... dans lequel on trouve ...
        try {
            // bloc try...catch à cause de l'exception lancée par close
            oos.close();
        } catch (IOException e) {e.printStackTrace();}
}
```

Exercice 2 : CabinetMedical : Sérialisation/Deérialisation d'une liste de Personne

1. Modifications à apporter : Les objets métier doivent devenir Serializable ...

```
import java.io.Serializable;
public abstract class Personne implements Serializable{ ...}
```

et

```
public class Adresse implements Serializable{ ...}
```

Remarque :

➤ Du moment qu'on implémente Serializable dans la classe Personne les classes filles :

Patient et Professionnel sont automatiquement Serializable ...

implements Serializable est donc facultatif pour les classes Patient et Professionnel ...

➤ ATTENTION :!!!

On ne peut pas se contenter d'implémenter seulement en Serializable Patient et Professionnel :

```
public class Patient extends Personne implements Serializable{ ...}
```

```
public class Professionnel extends Personne implements Serializable{ ...}
```

Comme Patient et Professionnel dérive de Personne :

⇒ !!! Personne doit OBLIGATOIREMENT ETRE IMLEMENTE Serializable !!!

2. Serialisation et Desérialisation de la collection

```
import java.util.ArrayList;
import java.util.Collection;
import java.io.*;
```

```
public class PersonneDAOFichier {
```

```
////////////////////////////////////
// Méthode permettant la SERIALISATION
public static void storeAllPersonnes (Collection<Personne> uneListe)
{
    try
    {
        FileOutputStream fichier = new FileOutputStream("cabMedPersonne.data");
        ObjectOutputStream oos = new ObjectOutputStream(fichier);
        oos.writeObject(uneListe);
        oos.close();
    }
    catch (IOException e)
    { System.out.println("Erreur lors de la sérialisation !" +
e.getMessage());
    }
} // fin storeAllPersonnes
```

```
////////////////////////////////////
// Méthode permettant la DESERIALISATION
public static Collection<Personne> findAllPersonnes ()
{
    Collection<Personne> maListe=null;
    try
    {
        FileInputStream fichier = new FileInputStream("cabMedPersonne.data");
        ObjectInputStream ois = new ObjectInputStream(fichier);
        maListe =(Collection<Personne>) ois.readObject();
        ois.close();
    }
    catch (IOException e)
    {
        System.out.println("Erreur !!");
    }
    catch (ClassNotFoundException e)
    {
        System.out.println("Erreur de classe !!");
    }

    if (maListe==null) // ... prévoir le cas où il pourrait y avoir
                    // un pb lors de la désérialisation
                    // => fichier vide ou fichier inexistant
    {
        maListe = new ArrayList<Personne>();
    }
    //Instanciation indispensable pour renvoyer une référence valide (!=null)
    return maListe;
} // fin findAllPersonnes
} //fin classe
```

Remarque : on instancie à ArrayList pour renvoyer une référence sur une Collection vide et non une référence à null

... peu importe la classe choisie pour instancier la collection on aurait pu choisir LinkedList...

Le but est de renvoyer une référence valide dans tous les cas

Ainsi dans un programme appelant on manipulera directement des collections sans se soucier de leur implémentation

⇒ séparation du code et des compétences ...

Les laisser chercher, il testeront en TP ...

... Comme on renvoie toujours une référence valide, on peut manipuler directement des collections ...

```
package com.iut.cabinet.essai;

import java.util.ArrayList;
import java.util.List;
import com.iut.cabinet.metier.*;

public class EssaiCabMed_v3 {

    public static void main(String[] args) {
        new EssaiCabMed_v3 ();
    }

    public EssaiCabMed_v3 ()
    {

        // Désérialisation---
        // Récupération des données du fichier CabMedPersonne.data
        // dans une liste de Personne
        List<Personne> =(List<Personne>)PersonneDAOFichier.findAllPersonnes();
        // Appel méthode statique...ne pas oublier le cast...
        // ... comme on renvoie toujours une référence valide,
        // pas besoin de tester par rapport à null ! ! !
        // 1 seule ligne suffit
        // Attention, il s'agit bien de List et non Collection...
        // à cause du sort dans la suite du code ...


        // Manipulation de la liste de Personne
        ..... Par exemple le code du TP précédent .....


        // Sérialisation de la liste de personne
        // dans le fichier CabMedPersonne.data
        PersonneDAOFichier.storeAllPersonnes(maListe); // Appel méthode statique

    }

}
```

Remarque : en théorie, il est possible d'écrire :

```
Collection<Personne> maListe = new ArrayList<Personne>();
... mais pas avec le pgm du TP précédent car les comparateurs font appel à la méthode sort et premier
paramètre de type List (donc on peut passer List, ArrayList, mais pas Collection)
```