Support

Développement Orienté Objet

En JAVA

LA PERSISTANCE

SOMMAIRE :

1 - L'INTERFACE SERIALIZABLE ET LES FLUX D'OBJET

La sérialisation d'un objet doit permettre de transporter celui-ci à travers un flux, ou de le stocker dans un fichier afin qu'il soit possible de reconstruire cet objet à l'identique à un moment donné. Pour être sérialisé un objet doit être sérialisable, c'est à dire avec la clause **implements Serializable** codée dans son entête.

Le concepteur d'une classe peut décider que certaines variables d'instance n'auront pas à être sérialisées, parce que ce sont des données transitoires qui ne font pas partie de l'état persistant de l'objet. De telles données, que l'on ne souhaite ni sauvegarder ni restaurer, doivent être déclarées avec le modificateur **transient.** De même toute variable déclarée **static** sera ignorée lors de la sérialisation.

a) Classes de java.io consernées

OBJECT -à INPUTSTREAM -à OBJECTINPUTSTREAM ---(utilise)à FILEINPUTSTREAM

OBJECT -à OUTPUTSTREAM -à OBJECTOUTPUTSTREAM ---(utilise)à FILEOUTPUTSTREAM

b) Exemple Persistance de chaîne ou d'entier à partir des classes String et Integer qui sont sérialisables

```
package tp4;
import java.io.*;
import biputil2.*;

public class Serial1 {
	public static void main (String[] args) throws Exception
	{
		Persiste op=new Persiste();
		String st1=new String("Bonjour");	// instanciation d'un objet st1 de type String
		Integer i2=new Integer(55);		// instanciation d'un objet i2 de type Integer

		System.out.println("\n1 = ecrire une chaine");
		System.out.println("2 = Lire une chaine");
		System.out.println("3 = ecrire un Entier");
		System.out.println("4 = Lire un Entier");

	  int opt = Keyboard.getInt();
	  while (opt != 0)
	  {
		if ( opt == 1 )
			op.Enregistre(st1);
		if ( opt == 2 ) {
			Object stLu = op.Lit();
			op.Affiche(stLu);
		}
		if ( opt == 3 )
			op.Enregistre(i2);
		if ( opt == 4 ){
			Object ILu = op.Lit();
			op.Affiche(ILu);
		}
		System.out.println("\n1 = ecrire une chaine");
```

```
            System.out.println("2 = Lire une chaine");
            System.out.println("3 = ecrire un Entier");
            System.out.println("4 = Lire un Entier");
            opt = Keyboard.getInt();
        }
         System.out.println("Fin Programme");
    }

}

package tp4;

import java.io.*;

class Persiste  {
    static String nomFichier="Serial1.obj";

                // sérialise un objet dans le fichier Serial1.obj

    public void Enregistre (Object o ) throws Exception
    {
        // Création de l'objet fos, flux de sortie de type fichier
          ObjectOutputStream fos= new ObjectOutputStream(new FileOutputStream(nomFichier));

        // sérialisation de l'objet dans le fichier
          fos.writeObject(o);
          fos.close();
    }
    public Object Lit() throws Exception  // réinstancie l'objet qui a été sérialisé
    {
        // Création de l'objet fis, flux d'entrée de type fichier
          ObjectInputStream fis= new ObjectInputStream(new FileInputStream(nomFichier));

        // lecture dans le flux et réinstanciation de l'objet
          Object o = fis.readObject();
        return o;
    }
    public void Affiche( Object o )
    {
          System.out.println("Objet de Classe : " + o.getClass().getName());
          System.out.print("  Valeur : " + o );
    }
}
```

c) Serialisation d'un Vecteur

Une Classe Liste serializable contenant un vecteur est passée en paramètre à une classe Fichier contenant des méthodes Ouvrir , Lire et Enregistrer et Fermer implémentant les méthodes readObject et writeObject

```
package tp4_2;

/**
 * <p>Titre : </p>
 * <p>Description : Classe sérialisable</p>
 * <p>Copyright : Copyright (c) 2002</p>
 * <p>Société : </p>
 * @author non attribué
 * @version 1.0
 */
import java.io.*;
import java.util.*;

public class Liste  implements  Serializable
{
	private Vector listeMot;

	public Liste () // constructeur
	{
		listeMot = new Vector();
	}
	public void ajout (String mot)
	{
		listeMot.addElement (mot);
	}
	public void affiche ()
	{
		System.out.println("Nombre d'elements du vecteur :" + listeMot.size());
		int i;
		for (i = 0; i <  listeMot.size(); i++)
		{
			System.out.println( i + " " + listeMot.elementAt(i).toString());
		}
	}
}
```



```
package tp4_2;

/**
 * <p>Titre : </p>
 * <p>Description : Classe permettant la sérialization et la désérialization (sauvegarde et
restauration)</p>
 * <p>d'une classe sérialisable (Liste) dans un fichier </p>
 * <p>Copyright : Copyright (c) 2002</p>
 * <p>Société : </p>
 * @author non attribué
 * @version 1.0
 */

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

```
public class Fichier
{
	private String nomFichier = "fichier.txt"; // fichier texte sur le disque

	private char mode;
	// = "E" pour ouvrir et écrire le flux,
	// = "L" pour ouvrir et lire

	private ObjectOutputStream oFW; // Flux d'écriture
	private ObjectInputStream iFW;	// Flux de lecture

	public boolean Ouvrir (String o)
	{
		try
		{
		mode = (o.toUpperCase()).charAt(0);
		if (mode == 'E')		// Ouverture du flux de sortie, pour écrire
			oFW= new ObjectOutputStream(new FileOutputStream(nomFichier));
		else				// Ouverture du flux d'entrée, pour lire
			iFW= new ObjectInputStream(new FileInputStream(nomFichier));
		return true;
		}
		catch (IOException e) // Absence du fichier sur disque
		{
			System.out.println("Fichier inexistant, faire la saisie et la sauvegarde");
			return false;
		}
	}
	public void Enregistre (Liste o) throws Exception
	{
		if (o != null)
			oFW.writeObject(o); // Ecriture du flux
	}
	public Object Lit() throws Exception
	{

		Liste o = (Liste) iFW.readObject(); // Lecture du flux,
		// récupération d'un objet o et "casting" ou "typage"
		// avec la classe liste
	   return o;// renvoi de l'objet Liste
	}

	public void Fermer (String f) throws Exception
	{
		mode = (f.toUpperCase()).charAt(0);
		if (mode == 'L')
			iFW.close();	// fermeture du flux d'entrée
		else
			oFW.close();	// fermeture du flux de sortie
	}

}
package tp4_2;

/**
 * <p>Titre : </p>
 * <p>Description : Scénario d'utilisation des classes Fichier et Liste</p>
 * <p>Copyright : Copyright (c) 2002</p>
 * <p>Société : </p>
 * @author non attribué
```

```
 * @version 1.0
 */

import java.io.*;
import java.util.*;
import biputil2.*;

public class Serializ
{
	public static void main (String[] args) throws Exception
	{
		int opt = 0;
		boolean ok;
		String strL;

		Liste l = new Liste();		// instanciation d'un objet l
		Fichier f = new Fichier();	// instanciation d'un objet f

		System.out.println("1 = Saisir une liste de mots");
		System.out.println("2 = Sauvegarde de la liste");
		System.out.println("3 = Resauration de la liste");
		System.out.println("4 = Affichage de la liste");
		System.out.println("5 = FIN");
		opt = Keyboard.getInt();

		while ( opt < 5 )
		{
		if ( opt == 1 )
		{
			System.out.println("SAISIE d'une Liste de mots");
			strL = Keyboard.getString();

			while ( strL.compareTo("")!=0 )  // ajout à la liste
			{
				l.ajout(strL);
				strL=Keyboard.getString();
			}
		}
		if ( opt == 2 )
		{
			f.Ouvrir ("E");
			f.Enregistre(l);
			f.Fermer("E");
		}
		if ( opt == 3 )
		{
			ok = f.Ouvrir ("L");
			if (ok)
			{
				l = (Liste) f.Lit();
				f.Fermer("L");
			}
		}
		if ( opt == 4 )
		{
			l.affiche();
		}
		System.out.println("1 = Saisir une liste de mots");
		System.out.println("2 = Sauvegarde de la liste");
		System.out.println("3 = Resauration de la liste");
```

```
            System.out.println("4 = Affichage de la liste");
            System.out.println("5 = FIN");
            opt = Keyboard.getInt();
          }
          System.out.println("Au revoir ");
        }

}
```

d) cas salarie

Il s'agit de rendre persistant les salaries en y implemantant l'interface Serializable pour pouvoir utiliser les méthode Read et writeObject

VOIR à SalariePersistSol.doc

E) Rappels sur le Dictionnaire : HASHTABLE

Permet de stocker en mémoire une collection d'objets avec des clefs

```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```


Pour retrouver un nombre :

```
Integer n = (Integer)numbers.get("two");
if (n != null)
{
        System.out.println("two = " + n);
}
```

2/ LES FLUX DE FICHIERS ASCII

a) les Classes du package Java.io

1- Classe File = fichier ASCII – Déclaration du fichier

```
File fichier = new File("c:\fichier.txt") ;
If ( !fichier.exists() )
  System.out.printll(« \nFichier inconnu) ;
```

2 -Classe FileReader = Lecteur de Fichier

```
// constructeur : FileReader(File)   : à partir d'un fichier
        FileReader fr = new FileReader(fichier) ;
// autre manière,  avec le chemin du fichier
        FileReader fr = new FileReader("c:\fichier.txt "));
```

3-Classe BufferedReader = Buffer de lecture Fichier


```
// c'est avec cet objet que l'on peut réellement lire le fichier
// constructeur : BufferdReader (FileReader) : à partir d'un lecteur de fichier
        BufferdReader br = new BufferdReader( fr ) ;

// autre maniere :
        BufferedReader br   = new BufferedReader(new FileReader("fichier.txt"));
```

Méthode :

```
public void close() throws IOException ; // fermeture du flux
public int read() throws IOException : //   Read a single character.
public String readLine() throws IOException ;
        //  Read a line of text. A line is considered to be terminated by any one of a line   feed ('\n'), a
        carriage return ('\r'), or a carriage return followed immediately by    a linefeed.

Exemple : String strLigne = new String() ;
                Try
                {
                        SrtLigne= br.readLine() ;
                }
                Catch  ( IOException e )
                {
                        System.out.println(« \nErreurt lecture «  + e ) ;
                 }
                  // OK
                 If ( strLigne==null)
                        System.out.println(« \n Fin de fichier ») ;
                  else
                         System.out.println(« \n » + strLigne ) ;
```

4-Classe FileWriter = Ecriveur de Fichier

```
// constructeur : FileWriter(File)   : a partir d'un fichier
        FileWriter fwr = new FileWriter(fichier) ;
// autre manière
        FileWriter fwr = new FileWriter("c:\fichier.txt "));  // avec le chemin du fichier
```

5- Classe BufferedWriter = Buffer d'écriture Fichier

```
// c'est avec cet objet que l'on peut réellement lire le fichier
// constructeur : BufferdWriter (FileWriter) : a partir d'un Ecriveur de fichier
        BufferedWriter bwr = new BufferdWriter( fwr ) ;
// autre maniere :
        BufferedWriter bwr   = new BufferedWriter(new FileWriter("fichier.txt"));
```

Methode :

close() : Close the stream.
newLine() : Write a line separator.
write(char[], int, int) : Write a portion of an array of characters.
write(int) : Write a single character.
write(String, int, int) : Write a portion of a String.

Exemple : une fonction d'écriture fichier

```
    public boolean Ecrire(String strLine)
        {
        try        // bwr = buffer d'ecriture
        {
         bwr.write(strLine,0,strLine.length());
         bwr.newLine();
        }
         catch (Exception e)
        {
                System.err.println("Erreur Ecriture Séquentiel " + e);
                 return false;
        }
      return true;
    }
```

6 - Classe RandomAccessFile : Fichier Direct

//constructeur :

RandomAccessFile(File file, String mode) throws IOException

Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.

The mode argument must either be equal to "r" or to "rw", indicating either to open the file for input, or for both input and output, respectively.

Parameters:

file - the file object.

mode - the access mode.

Throws: IllegalArgumentException if the mode argument is not equal to "r" or to "rw".

Throws: IOException if an I/O error occurs.

Throws: SecurityException : si ecriture sans rw

//quelques methodes (il y en a envir on 30) :

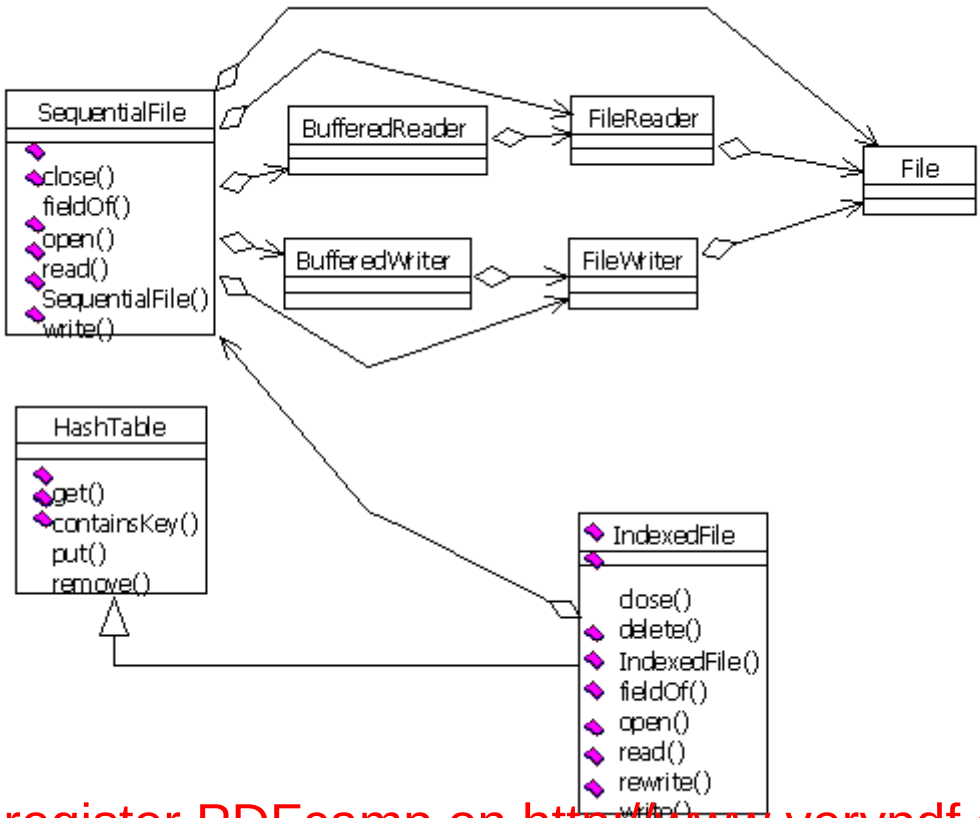
close()	Closes this random access file stream and releases any system resources
length()	Returns the length of this file.
read()	Reads a byte of data from this file.
readChar()	Reads a Unicode character from this file.
!readInt()	Reads a signed 32-bit integer from this file.
readLine()	Reads the next line of text from this file.
seek(long)	Sets the offset from the beginning of this file at which the next read or write occurs.
write(int)	Writes the specified byte to this file.
writeChar(int)	Writes a char to the file as a 2-byte value, high byte first.
writeChars(String)	Writes a string to the file as a sequence of characters.
writeInt(int)	Writes an int to the file as four bytes, high byte first.

°°°

B) Exemple : Package BipUtil

Deux classes pour encapsuler ces classes techniques :
SequentialFile : un fichier sequentiel
IndexedFile : un fichier texte s'appuyant sur un fichier séquentiel et une HashTable pour simuler l'index

1 - MODELE STATIQUE OBJET :



2 - Code JAVA Extrait de biputil

```
class SequentialFile  {
   private boolean m_ouvert;
   private char m_mode;
   private File m_f;
   private FileReader m_fr;
   private BufferedReader m_br;
   private FileWriter m_fw;
   private BufferedWriter m_bw;
   public boolean EndOfFile;

public SequentialFile(){  // constructeur
       m_ouvert=false;
        EndOfFile=false;
   }

public boolean open(String strFile,char cRW){
try{
   EndOfFile=false;
   if ( !(m_ouvert))      {
      m_ouvert=true;
      m_f = new File(strFile);
      m_fr = new FileReader(m_f);
switch (cRW) {
case 'r' :
   m_fr = new FileReader(m_f);
```

```
   m_br = new BufferedReader(m_fr);
   break;
}
case 'R' :
   m_fr = new FileReader(m_f);
   m_br = new BufferedReader(m_fr);
   break;
}
  case 'w' :
{  m_fw = new FileWriter(m_f);
   m_bw = new BufferedWriter(m_fw);
}
m_mode=cRW;
return true;
}
else   {
       System.err.println("Erreur Mode Ouverture ");
      return false;
   }
}
catch (Exception e) {
System.err.println("Erreur Ouverture Sequentiel " + e);
return false;
   }
}

public String read() {
   String strLine = new String();
   try  {
     if (m_mode=='r'||m_mode=='R')
         strLine= m_br.readLine();
    else
        System.err.println("Erreur Lecture Sequentiel");
   }
catch (Exception e)  {
    System.err.println("Erreur Lecture Sequentiel " + e);
  }
if (strLine==null)
EndOfFile=true;
return strLine;
 }

public boolean write(String strLine)    {
try{
   if (m_mode=='w'||m_mode=='W')   {
          m_bw.write(strLine,0,strLine.length());
          m_bw.newLine();
    }
else
       System.err.println("Erreur Ecriture Sequentiel");
 }
catch (Exception e)      {
     System.err.println("Erreur Ecriture Sequentiel " + e);
     return false;
 }
return true;
  }

public void close()   {
try  {
```

```
    if (m_ouvert)     {
       m_ouvert=false;
        if (m_br != null ) m_br.close();
        if (m_fr != null ) m_fr.close();
        if (m_bw != null ) m_bw.close();
      if (m_fw != null ) m_fw.close();
       // if (m_f != null ) m_f.close();
    }
   else
         System.err.println("Erreur Fermeture Sequentiel " );
 }
catch (Exception e)     {
     System.err.println("Erreur Fermeture Sequentiel " + e);
  }
}
```

3 / LES SGBD

a - le package java.sql et la norme JDBC

Java et les SGBD

Java s'est penché très tôt sur le problème de la connexion aux bases de données relationnelles car elles sont au coeur des applications de l'entreprise. Java dispose d'un package dédié à cette connexion : **java.sql** ou **JDBC** pour **J**ava **D**ata**B**ase **C**onnectivity. JDBC est un ensemble de classes et d'interfaces permettant de réaliser des connexions vers des bases de données, d'effectuer des requêtes, et quelques autres fonctionnalités comme la conversion de types Java en SQL et inversement. De manière générale, un programme utilisant JDBC fonctionne de la façon suivante :

1 - Connexion avec la base
2 - Emission de requêtes
3 - Récupération des données
4 - Fermeture de la connexion.

La première étape est prise en charge par un composant externe au programme, un pilote logiciel. C'est le **driver JDBC de la base de données**. La seconde fait appel à des **connaissances en SQL**. On va tout simplement émettre des requêtes SQL à la base. Le troisième point va essentiellement consister à **convertir les types SQL en types Java**. Le dernier point contient de façon sous-jacente la notion de **transaction**. L'API JDBC supporte les principaux modes de transaction et les fonctions de "commit" et "rollback". Par contre, leur implémentation est confiée au driver de la base.

L'interface java.sql.driver est l'abstraction d'un pilote JDBC. Dans la plupart des cas, un pilote de base de données doit simplement fournir une implémentation des classes abstraites de l'API JDBC. Côté client, on aura donc besoin d'une librairie de classes pour effectuer la connexion à la base.

Différents types de pilotes JDBC

Javasoft propose de classer les pilotes JDBC en quatre catégories :

- **Type 1** : Pont entre JDBC et ODBC. Nécessite l'emploi d'une librairie native.
- **Type 2** : Accès en Java aux librairies natives du moteur de base de données. Nécessite l'emploi d'une librairie native.
- **Type 3** : Pilote tout Java. Utilisation d'un protocole (propriétaire, défini par le fournisseur du pilote) entre un client Java et la base de données. Il n'est pas nécessaire d'utiliser des librairies natives.
- **Type 4** : Comme le précédent, mais utilisation du protocole du moteur de la base de données.

Connexion avec une base de données

Pour établir une connexion avec une source de données, il faut connaître **son nom** et lui **associer un pilote**. La convention de « nommage » retenue pour **JDBC** est dérivée de celle utilisée pour les **URL** sur Internet. On retrouve ainsi un protocole, un nom d'hôte et une partie qui elle est variable. La forme générale est la suivante :

jdbc:<sous-protocole>:<complément>

Pour une source de données locale avec un pilote de type 1, l'URL pourraît être de la forme :

jdbc:odbc:Annuaire

si Annuaire est une source ODBC connue du système.
Pour une source de données distante, l'URL pourra être de la forme :

jdbc:oracle://serveur:port/base;paramètre=valeur...

Dans tous les cas il faut vérifier avec la documentation fournie la syntaxe exacte supportée par le pilote.

Les différentes étapes pour utiliser JDBC :

1 - Enregistrement du pilote à l'aide de la méthode statique **registerDriver** du **DriverManager**
2 - Création de la connexion à l'aide de la méthode statique **getConnection** du DriverManager avec l'URL de la base passée en paramètre
3 - Récupération d'un objet **Statement** qui possède les méthodes nécessaires pour exécuter des requêtes sur la base
4 - Exécution de requêtes à l'aide de la méthode **executeQuery** de Statement
5 - Récupération des résultats à l'aide de **ResultSet**

Exemple : liste des salariés avec passerelle jdbc-odbc

```
package javabd;


/**
 * <p>Titre : </p>
 * <p>Description : Connection Java Base de Donnée par JDBC Type 1</p>
 * <p>Copyright : Copyright (c) 2002</p>
 * <p>Société : </p>
 * @author non attribué
 * @version 1.0
 */

import java.sql.*;
import java.io.*;
```

```
import biputil2.* ;

public class SalarieBd {

public static void main(String args[]) {

  String url = "jdbc:odbc:Salarie";   // Source ODBC Win32
  Connection con = null;
  try
   {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      //  On se connecte via la passerelle jdbcOdbc implanté dans la librairie jdbcodbc.dll
      con = DriverManager.getConnection(url,"","");
      Statement requete = con.createStatement();

      // on fabrique une vue resultat
      ResultSet resultat = requete.executeQuery ("select * from salarie");
      System.out.println("Numero  Nom des Salariés ");

      while ( resultat.next())
      {
      //  Booleen faux si EOF , lecture suivante
        System.out.println(resultat.getInt(1) + "  " + resultat.getString(2));
      }
    }
    catch(Exception e)
    {
      System.out.println("Exception");
    }
```



```
    finally
    {
    try
    {
      con.close();
    }
    //catch(SQLException e) {e.printStackTrace();
    catch(Exception e)
    {
      e.printStackTrace();
      char cC=Keyboard.getChar();
    }
    }
}
}
```