//-------------------------Lister les fichiers----------------------------------

 **public** **void** listerRepertoire(File repertoire){

 **String** [] listeEspace;

 **int** i;

 System.out.println("Applications linéaire Disponible :");

 listeEspace=repertoire.list();

 for(i=0;i<listeEspace.length;i++){

 if(listeEspace[i].endsWith("\_LinearAppli.ser"))

 System.out.println(listeEspace[i].replaceFirst("\_LinearAppli.ser",""));

 }

 }

//------------------------------------------------------------------------------

//-------------------------SERIALISATION----------------------------------------

 **public** void serialize(linearAppli apl ){

 **try**

 {

 **FileOutputStream** fichier = **new** FileOutputStream(apl.nomapli+"\_LinearAppli.ser");

 **ObjectOutputStream** oos = **new** ObjectOutputStream(fichier);

 oos.writeObject(apl);

 oos.flush();

 oos.close();

 }

 **catch** (java.io.IOException e) {

 e.printStackTrace();

 }

 **public** linearAppli deserialize(String nom, boolean S){

 linearAppli ap = **new** linearAppli();

 **try** {

 **FileInputStream** fichier = **new** FileInputStream(nom+"\_LinearAppli.ser");

 **ObjectInputStream** ois = **new** ObjectInputStream(fichier);

 ap = (linearAppli) ois.readObject();

 }

 **catch** (java.io.IOException e) {

 System.out.println(" -----Erreur!!!-----\n nom de l'application non trouvé\n\n");

 listerRepertoire(ap.get\_File());

 ap.A=false;

 }

 **catch** (ClassNotFoundException e) {

 System.out.println(" -----Erreur!!!-----\n nom de l'application non trouvé\n\n");

 listerRepertoire(ap.get\_File());

 ap.A=false;

 }

 return ap;

 }

}

//------------------------------------------------------------------------------

package calculmathematique;

import java.util.Scanner;

import **java**.**io**.\*;

**public** class **linearAppli** **implements** **java**.**io**.**Serializable**{

 **private** **String** **nomapli**;

 **private** **Matrix** **matrice**;

 **private** **int** **dimension**;

 **private** **Matrix** **vecteur**;

 **private** **Matrix** **result**;

 **private** **boolean** **A**=true;

 **public** **File** **repR** = **new** File(System.getProperty("user.dir"));

//----------------------------CONSTRUCTEUR--------------------------------------

 **public** **linearAppli**(){

 **String** **nomapli**="";

 **int** **dimension**;

 **Matrix** **matrice**= **new** Matrix();

 **boolean** **A**=true;

 **File** **repR** = **new** File(System.getProperty("user.dir"));

 }

//----------------------------GETTER & SETTER-----------------------------------

 **public** **File** get\_File(){

 return repR;

 }

 **public** **boolean** get\_A(){

 return A;

 }

 **public** **int** get\_dimension(){

 return this.dimension;

 }

 **public** **String** get\_nom(){

 return this.nomapli;

 }

//--------------------------------INITIALIZE--------------------------------------------

 **public** **linearAppli** Initialize(){

 **Scanner** saisir= **new** Scanner(System.in);

 **String** lol;

 System.out.println("Saisir le nom de l'appli : \n");

 this.nomapli=saisir.next();

 System.out.println("Saisir la dimension de la matrice associé");

 this.dimension=saisir.nextInt();

 **Matrix** matrice = **new** Matrix();

 this.matrice=matrice.Initialize(matrice,nomapli+"\_Matrix",dimension,dimension);

 return this;

 }

//----------------------------------AFFICHAGE--------------------------------------------

 **public** **void** Show(linearAppli ali){

 System.out.println("Nom de l'application : "+ali.nomapli);

 ali.matrice.show();

 }

 **public** **Matrix** vectImage(Matrix Vect){

 **Matrix** M= **new** Matrix(1,this.get\_dimension());

 M = Vect.multi(this.matrice);

 return M;

 }

 }

Import de tous les éléments
de java.io
déclaration de la classe

Essaie d’exécuter, si erreur, exécute catch

Pour rendre un objet
sérialisable, il faut rajouter
**implements Java.io.serializable**
lors de la création de sa classe

**1** Définition du fichier de sortie
**2** On instancie l’objet de sérialisation
**3** Ecriture de l’objet dans le fichier
**4** On vide la mémoire tampon
**5** Destruction de l’objet oos

**Static** : Les variables d'instance (Static) sont des variables propres à un objet. Il est possible de définir une variable de classe qui est partagée entre toutes les instances d'une même classe : elle n'existe donc qu'une seule fois en mémoire. Une telle variable permet de stocker une constante ou une valeur modifiée tour à tour par les instances de la classe. Elle se définit avec le mot clé static.

static float pi = 3.1416f;

Une méthode static est une méthode qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe. Ces méthodes peuvent être utilisées sans instancier (créer) un objet de la classe. Les méthodes ainsi définies peuvent être appelées avec la notation classe.methode() au lieu de objet.methode() : la première forme est fortement recommandée pour éviter toute confusion. Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable

d'instance à partir d'une méthode de classe statique.

**Final** : Le mot clé final s'applique aux variables de classe ou d'instance ou locales, aux méthodes, aux paramètres d'une méthode et aux classes. Il permet de rendre l'entité sur laquelle il s'applique non modifiable une fois qu'elle est déclarée pour une méthode ou une classe et initialisée pour une variable. Une variable qualifiée de final signifie que la valeur de la variable ne peut plus être modifiée une fois que celle-ci est initialisée.

public static final float PI = 3.141f;

**Abstract :** Le mot clé abstract s'applique aux méthodes et aux classes. Abstract indique que la classe ne pourra être instanciée telle quelle. Toutes les méthodes de la classe abstract ne sont pas implémentées et devront être redéfinies par des méthodes complètes dans ses sous classes. Abstract permet de créer une classe qui sera une sorte de moule. Les classes dérivées pourront profiter des méthodes héritées et n'auront à implémenter que les méthodes déclarées abstract.

**abstract** **class ClasseAbstraite** {

 ClasseAbstraite() { ... //code du constructeur }

 void methode() { ... // code partagé par tous les descendants }

 **abstract void methodeAbstraite();** }

**class** ClasseComplete **extends ClasseAbstraite** {

 ClasseComplete() { super(); ... }

 **void methodeAbstraite()** { ... // code de la méthode }

 // void methode est héritée }

**Exemple : sérialisation et dé sérialisation d'un objet et enregistrement sur le disque dur**

**import java.io.\*;**

public class SerializerPersonne {

public static void main(String argv[]) {

Personne personne = new Personne("Dupond","Jean",175);

 try {

 FileOutputStream fichier = new FileOutputStream("personne.ser");

 ObjectOutputStream oos = new ObjectOutputStream(fichier);

 oos.writeObject(personne);

 oos.flush();

 oos.close();

 }

 catch (java.io.IOException e) {

 e.printStackTrace();

 }

}

}

Déclaration des tableaux
int tab[]; // déclaration

tab = new int[50]; //allocation
float tableau[][]=new float[8][8];

**ENUMERATION**

public class TestEnum2 {
 public enum Jour {

 LUNDI, MARDI, MERCREDI, JEUDI,

 VENDREDI, SAMEDI, DIMANCHE

 } }

Les énumérations permettent de définir un ensemble fini de constantes, chacune d'entre-elles est séparée des autres par une virgule. Comme ces champs sont constants, leur nom est en majuscule par convention.

**import java.io.\*;**

public class DeSerializerPersonne {

public static void main(String argv[]) {

 try {

 FileInputStream fichier = new FileInputStream("personne.ser");

 ObjectInputStream ois = new ObjectInputStream(fichier);

 Personne personne = (Personne) ois.readObject();

 System.out.println("Personne : ");

 System.out.println("nom : "+personne.getNom());

 System.out.println("prenom : "+personne.getPrenom());

 System.out.println("taille : "+personne.getTaille());

 }

 catch (java.io.IOException e) {

 e.printStackTrace();

 }

 catch (ClassNotFoundException e) {

 e.printStackTrace();

 }

Mots Clés

L'opérateur **instanceof** permet de déterminer la classe de l'objet qui lui est passé en paramètre. La syntaxe est **objet** **instanceof** **classe**

**void** testClasse(Object o) {

**if** (**o** **instanceof MaClasse** )

*System.out.println*(" o est une instance de la classe MaClasse ");

**else** *System.out.println*(" o n'est pas un objet de la classe MaClasse "); }

Syntaxe de déclaration d'une classe:
 **extends** et **implements** sont facultatifs

**modificateurs** **class** ***nom\_de\_classe*** **extends** ***classe\_mère*** **implements** ***interfaces*** { ... }

Modificateur Rôle

Abstract La classe contient une ou des méthodes abstraites, qui n'ont pas de
 définition explicite. Une classe déclarée abstract ne peut pas être
 instanciée : il faut définir une classe qui hérite de cette classe et qui
 implémente les méthodes nécessaires pour ne plus être abstraite.

Final La classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage
 est interdite. Les classes déclarées final ne peuvent donc pas avoir de
 classes filles.

private La classe n'est accessible qu'à partir du fichier où elle est définie

public La classe est accessible partout