Réseaux Avancés
M1 GL 2023/2024 - TP N° 1 (pré requis Sockets)

Sommaire :

Exercice 1 : Modificateur transient pour la sérialisation 1
Exercice 2 : Sérialisation & transient 3
Exercice 3 : isReachable 3
Exercice 4 : Scanner et HashMap 4
Exercice 5 : Gestion de flots avec FileReader et FileWriter 5
Exercice 6 : Gestion de flots avec FileWriter 6

Utiliser l'IDE NETBEANS pour les TPs de ce module.

Exercice 1 : Modificateur transient pour la sérialisation

Objectif : parfois, nous cherchons à sérialiser une partie de l'objet et pas l'objet tout entier (tous ses attributs). Ex. Marquer un champ tel que le mot de passe comme non inclus lors de la sérialisation des objets de type Person est crucial pour la sécurité. Les données sensibles comme les mots de passe ne doivent pas être exposées lors de la sérialisation et de la désérialisation d'objets, car cela pourrait compromettre la sécurité du système.

Remarque : utiliser le chemin "./src/TP1/employee.txt" si package TP1 au lieu du default package.

Soit la classe java suivante :

```
import java.io.Serializable;
public class Employee implements Serializable {
        public String nom;
        static String prenom;
        transient String adresse;
        transient static String affiliation;
}
```

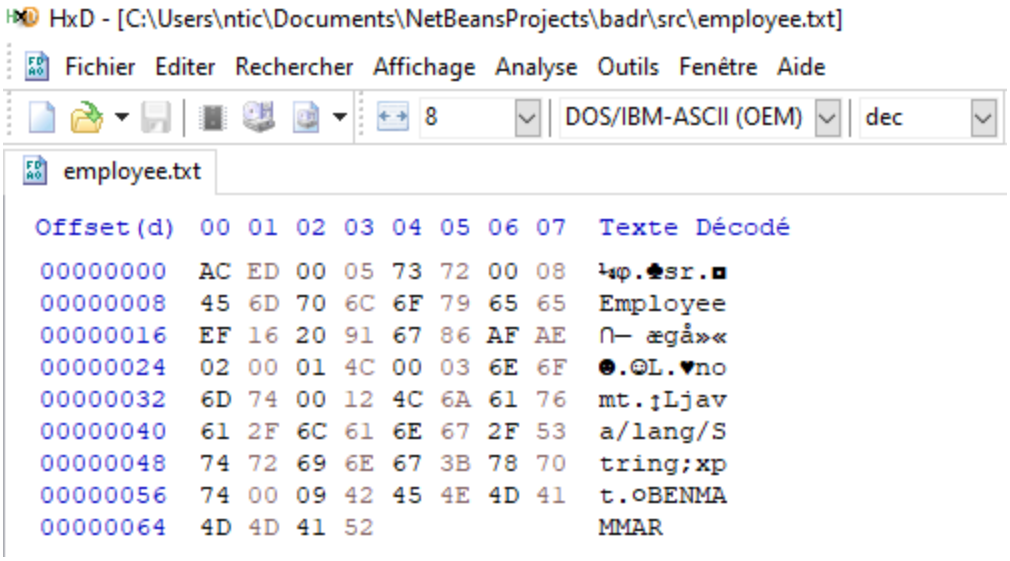
La classe java suivante est utilisée pour la sérialisation.

```
import java.io.*;
public class Serialization {
        public static void main(String[] args) {
                Employee emp = new Employee();
                emp.nom = "BENMAMMAR";
                emp.prenom = "BADR";
                emp.adresse = "TLEMCEN";
                emp.affiliation = "UABT";
                try {    FileOutputStream fileOut = new FileOutputStream("./src/employee.txt");
                        ObjectOutputStream out = new ObjectOutputStream(fileOut);
                        out.writeObject(emp);
                        out.close();
                        fileOut.close();
                        System.out.printf("Serialisation dans ./employee.txt");
                } catch (IOException i) {i.printStackTrace();}
        }
}
```

La classe java suivante est utilisée pour la désérialisation.

```
import java.io.*;
public class Deserialization {
        public static void main(String[] args) {
                Employee emp=null;
                try {    FileInputStream fileIn = new FileInputStream("./src/employee.txt");
                         ObjectInputStream in = new ObjectInputStream(fileIn);
                         emp = (Employee) in.readObject();
                         in.close();fileIn.close();
                } catch (IOException i) {i.printStackTrace();
                } catch (ClassNotFoundException c) {c.printStackTrace();}
                System.out.println("Deserialisation...");
                System.out.println("Nom: " + emp.nom);
                System.out.println("Prenom: " + emp.prenom);
                System.out.println("Adresse: "+emp.adresse);
                System.out.println("Affiliation: "+emp.affiliation);
        }
}
```

- Visualiser le fichier **employee.txt** avec **HxD Hex Editor.** Normalement, vous devrez avoir un affichage comme suit :



- Exécuter **Serialization.java** et **Deserialization.java** et examiner le résultat.
- Quel est l'effet de **static** et de **transient** sur la sérialisation ?
- Modifier la déclaration des attributs dans la classe **Employee** comme suit :

```
public String nom;
static String prenom="BBB";
transient String adresse="CCC";
transient static String affiliation="DDD";
```

- Relancer l'exécution.
- **Constat :** la sérialisation ne marche toujours pas pour l'attribut **transient** (malgré l'affectation), par contre, ça marche pour les attributs **static** dont nous avons fait une affectation (dont **transient static**).

- Modifier à nouveau la déclaration des attributs dans la classe **Employee** comme suit :

```
public String nom;
static String prenom;
transient final String adresse="CCC";  // une constante
transient static String affiliation;
```

- Bien évidemment, il faut mettre la ligne **emp.adresse = "TLEMCEN";** (dans la classe **Serialization**) en commentaire. Relancer l'exécution. Quel est l'effet de **final** sur **transient** pour la sérialisation ?

Résumé : La variable globale **(static)** est sérialisée avec sa valeur d'affectation (ou valeur par défaut : **null** pour **String**). **transient** empêche la sérialisation sauf si elle est utilisée en combinaison avec **final (constante).**

Exercice 2 : Sérialisation & transient

- Ecrire une classe **Personne** contenant le constructeur, la méthode **toString()** ainsi que 3 attributs : **nom (String), prénom (String) et age (int)**.
- Ecrire une classe **EcrirePersonne** qui permet de sauvegarder une liste de personnes dans un fichier.
- Ecrire une classe **LirePersonne** pour restaurer cette liste et afficher les caractéristiques des personnes.
- Modifier la classe **Personne** afin de sérialiser que l'attribut **nom.** Les deux autres attributs **(prénom et age)** ne seront pas sérialisés (utiliser **transient**).

Exercice 3 : isReachable

Avant de commencer, examiner la méthode **isReachable** dans la documentation ORACLE de la classe **InetAddress.** https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html
La signature à considérer est la suivante : **public boolean isReachable (int timeout) throws IOException**
Voir le lien suivant : https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html#isReachable(int)
Le code suivant utilise la méthode **isReachable.**

```
import java.net.*;
public class DD {
        public static void main(String [ ] args) {
             String host = "localhost";
            // String host = "www.facebook.com"; si vous êtes connectés à internet
             mystere (host);
        }
        private static void mystere (String host) {
                try {    InetAddress b = InetAddress.getByName(host);
                         boolean var = b.isReachable(2000);
                          if (var)  System.out.println("OK : " + host);
                          else System.out.println("Not OK : " + host);
                    } catch (Exception e) {e.printStackTrace();}
        }
}
```

1. Que fait le code précédent ? Quelle est la commande réseau équivalente au comportement de la méthode **isReachable ?**

2. Que fait le code suivant ?

```
import java.io.*;
import java.net.*;
public class RSD {
public static void main(String[ ] args) throws IOException {
String host = "www.facebook.com"; // ou n'importe quelle autre machine (localhost en local par ex.)
InetAddress ip = InetAddress.getByName(host);
int debut = (int) System.currentTimeMillis();
boolean var = ip.isReachable(5000);
if (var) System.out.println("OK dans "+ ((int) System.currentTimeMillis() - debut)+" ms");
else    System.out.println("Not OK");
 }
}
```

- **isReachable** possède une deuxième signature. Voir le lien suivant : https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html#isReachable(java.net.NetworkInterface,%20int,%20int)
  public boolean isReachable (NetworkInterface netif, int ttl, int timeout) throws IOException
  3. Quelle est la signification des deux autres paramètres : **netif et ttl ?** Pour **ttl,** voir le lien suivant : https://www.frameip.com/entete-ip/
  4. Quel est le scénario dans lequel on doit préciser le paramètre **netif ?** c'est-à-dire, quelle est la particularité de la machine à partir du quelle, nous avons lancé la méthode **isReachable ?**

Exercice 4 : Scanner et HashMap

Avant de commencer, examiner le fichier : **symptomes.txt.** Il s'agit d'un contenu représentant 100 symptômes d'un certain nombre de patients. Un patient peut avoir un ou plusieurs symptômes.

1. Que fait le code suivant ?

```
import java.io.*;
import java.util.*;
public class A {
  static Integer var = 1;
  public static void main(String args[]) throws IOException {
    Map<String, Integer> map = new HashMap<>();
    Scanner a = new Scanner(new File("./src/symptomes.txt"));
      while(a.hasNextLine()) traiter (a.nextLine(), map);
    System.out.println(map);
  }
  static void traiter(String ligne, Map hash) {
    StringTokenizer st = new StringTokenizer(ligne);
    while (st.hasMoreTokens())  Ajouter (hash, st.nextToken());
  }
  static void Ajouter (Map map, String mot) {
    Object k = map.get(mot); // le mot existe ou pas encore dans le map ?
    if (k == null) map.put(mot,var); // insérer le mot pour la première fois
     else { // le mot existe déja
      int nb = ((Integer) k) + 1;
      map.put(mot, nb); // mettre à jour le nombre de fois
    }
  }
}
```

- Pour Hashtable voir : https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
- Pour Map voir : https://docs.oracle.com/javase/8/docs/api/java/util/Map.html
- Pour Scanner voir : https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
- Pour StringTokenizer voir : https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html

Modifier la classe A comme suit :

```
import java.io.*;
import java.util.*;
public class A {
  public static void main(String args[]) throws IOException {
    HashMap<String, Integer> map = new HashMap<>();
      try (Scanner a = new Scanner(new File("./src/symptomes.txt"))) {
          while (a.hasNextLine()) ajouter(map, a.nextLine());
      }
  }
  static void ajouter(Map<String, Integer> map, String symptome) {
    // compute() prend en paramètre une clé (symptome dans ce cas),
     // ainsi qu'une fonction à appliquer sur la clé et la valeur correspondante
     // (la fonction lambda (key, value) -> (value == null) ? 1 : value + 1) dans ce cas)
   map.compute(symptome, (key, value) -> (value == null) ? 1 : value + 1);
  }
}
```

2. En examinant la doc de l'interface **Map**, expliquer le comportement de la méthode **compute.** Pour les expressions lambda, vous pouvez consulter le lien suivant : https://www.javatpoint.com/java-lambda-expressions.

3. Que fait donc la classe **A** ?

Exercice 5 : Gestion de flots avec FileReader et FileWriter

Soit la classe java suivante (**test1** contient le mot BONJOUR, **test2** contient le mot ABCD) :

```
import java.io.*;
public class A {
 public static void main(String[] args) throws IOException {
 File inputFile = new File("./src/test1.txt");
 File outputFile = new File("./src/test2.txt");
 FileReader in = new FileReader(inputFile);
 FileWriter out = new FileWriter(outputFile);
 int c;
 while ((c = in.read()) != -1) out.write(c);
 in.close(); out.close();
 }
}
```

1. Exécuter le code précédent. Que fait cette classe ?
2. A la main, modifier le contenu de **test2** à ABCD. Modifier maintenant votre code afin de ne pas écraser le contenu de **test2.** Le nouveau contenu de **test2** doit être : ABCDBONJOUR (ajout à la fin).

Soit la classe java suivante :

```
import java.io.*;
public class B {
public static void main(String[ ] args)  {
    File fichier= new File("./src/test1.txt");
      try {
        FileWriter flotEcriture = new FileWriter(fichier);
         for (int i=1; i<=9;i++) flotEcriture.write(i);                    // ligne 7
        flotEcriture.close();
      } catch (IOException e) { }
      }
}
```

3. Exécuter cette classe et visualiser le contenu du fichier **test1** (pour visualiser **test1**, utiliser **Notepad++** dans le cas ou son contenu ne sera pas visible à partir de **l'IDE Netbeans**).
4. Modifier la ligne 7 par : **for (int i=65; i<=74;i++) flotEcriture.write(i);** Exécuter à nouveau la **classe B** et visualiser le fichier **test1.** Que représente l'entier passé comme paramètre à la méthode **write.**
5. Reprendre **la ligne 7** : **for (int i=1; i<=9;i++) flotEcriture.write(i);** Modifier votre code **(avec trois manières différentes)** afin d'écrire les entiers de 1 à 9 dans le fichier **test1**.

Exercice 6 : Gestion de flots avec FileWriter

Soit la classe A suivante :

```
import java.io.*;
public class A {
        public static void main(String [ ] args)  {
        File fichier= new File("./src/test3.txt");
        try {
        FileWriter flotEcriture = new FileWriter(fichier);
                for (int i=1; i<=7;i++){
                        flotEcriture.write("mirsdgl",0,8-i);
                        flotEcriture.write("\n");
                    }
        flotEcriture.close();
        } catch (IOException e) { }
        }
}
```

1. Exécuter cette classe et visualiser le contenu du fichier **test3.**
2. Expliquer l'instruction **flotEcriture.write("mirsdgl",0,8-i);** C'est quoi la signification des 3 paramètres ?
3. Modifier la classe A afin d'avoir le contenu suivant dans le fichier **test3.**

mirsdgl
irsdg
rsd
s
rsd
irsdg
mirsdgl

- Même question pour le contenu suivant :

```
m      l
mi    gl
mir dgl
mirsdgl
mir dgl
mi    gl
m      l
```