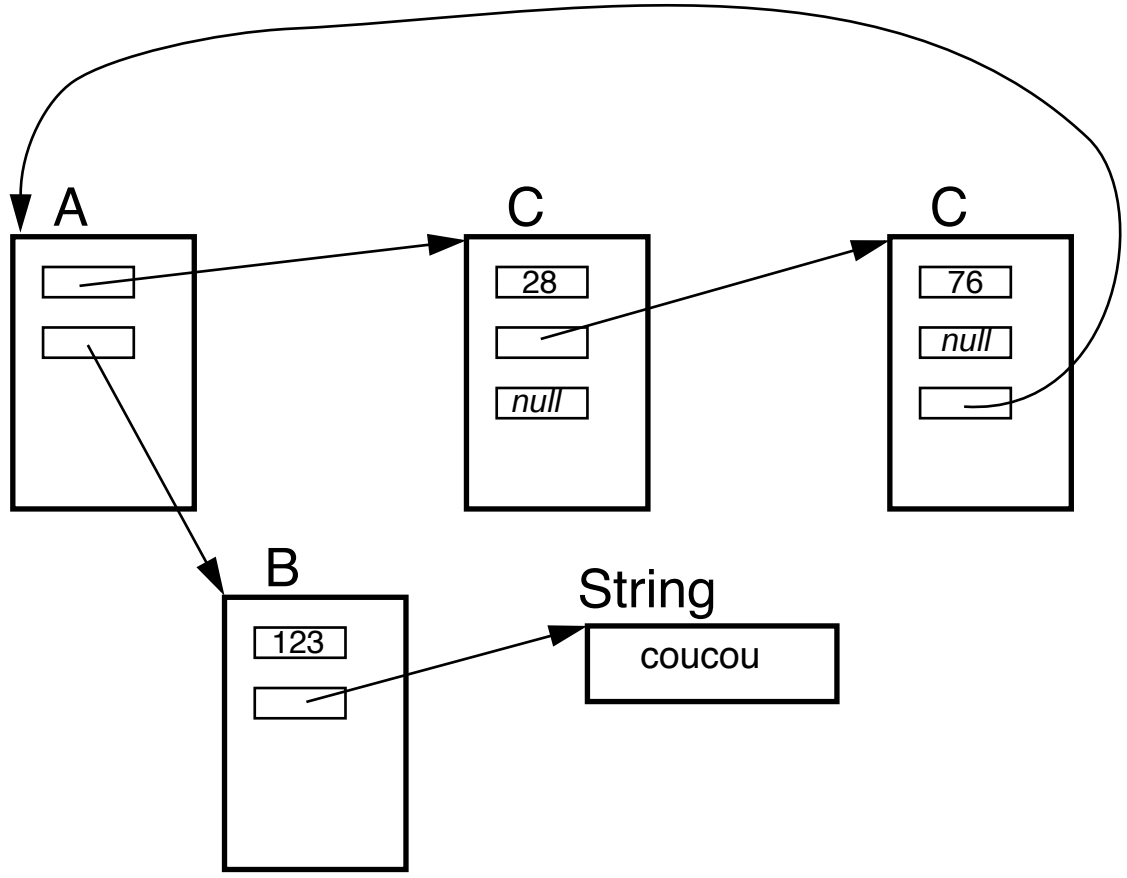
La sérialisation en Java

La sérialisation

La sérialisation d'un graphe d'objets consiste à obtenir une représentation linéaire de ces objets et de leurs relations. Sous cette forme, ils peuvent alors être exportés pour être réutilisés par une autre application, par exemple en les sauvegardant dans un fichier ou en les transmettant par socket à un autre processus.

L'opération inverse de désérialisation reconstruit une forme interne et structurée du graphe d'objet.

La sérialisation peut s'effectuer même en présence de cycles. Dans ce cas, à la première rencontre d'un objet lors du parcours du graphe, il est effectivement intégré à la représentation linéaire ; lors de rencontres ultérieures, seul un marqueur permettant de retrouver l'objet est conservé.



Définir un objet comme sérialisable

1) Pour être sérialisable, un objet doit appartenir à une classe implantant l'interface ***java.io.Serializable*** (ou héritant d'une classe qui implante cette interface). Il est alors possible d'utiliser directement le mécanisme de sérialisation par défaut.
   Le mécanisme par défaut consiste à écrire tous les champs non statiques et non ***transients***. Les champs de types primitifs (***int, boolean***…) sont directement sauvés, et les champs objets sont sérialisés.
   Pour ne pas rencontrer d'erreur à la sérialisation, il est donc nécessaire que tous les champs de l'objet soient eux-mêmes sérialisables (et ainsi récursivement pour tous les objets accessibles indirectement).
2) Un objet dont la classe implante ***java.io.Serializable*** et fournit les méthodes :

```
private void writeObject(java.io.ObjectOutputStream out) throws IOException;
private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;
```

   est sérialisable. Ces méthodes peuvent contenir un code utilisateur arbitraire. Elles sont appelées pour sérialiser/désérialiser un objet de cette classe.
3) un objet qui implante l'interface ***Serializable*** avec une méthode ***writeObject*** qui lève l'exception ***java.io.NotSerializableException*** n'est pas sérialisable, ce qui permet en particulier d'annuler la possibilité de sérialisation pour une sous-classe d'une classe spécifiée comme sérialisable.

Les versions

Lors qu'un objet est sérialisé, le flux de données produit comporte, outre l'état sérialisable proprement dit, une série de métadonnées comme le nom de la classe, le type et le nom des champs, et le numéro de « version série » (***serialVersionUID***). Ce numéro permet d'autoriser (éventuellement) la lecture d'un objet dont la classe a évolué depuis que l'objet a été chargé :

- Pour indiquer qu'une évolution de la classe est compatible avec sa version précédente, du point de vue de la sérialisation, il faut ajouter à la classe un attribut **static final long** ***serialVersionUID***. Dans ce cas, la (dé)sérialisation est possible, si la valeur définie pour ***serialVersionUID*** est la même pour les deux versions.
- Si l'attribut ***serialVersionUID*** n'est pas défini par le programmeur, celui-ci est calculé automatiquement, par hachage à partir de métadonnées de la classe, lors de la sérialisation, puis lors de la désérialisation. Une exception ***ClassNotFoundException*** est alors levée si les numéros ne concordent pas (parce que la classe a changé, ou parce que le résultat du hachage a changé). Les versions récentes de la plateforme Java recommandent (vivement) de spécifier explicitement le ***serialVersionUID***.

Les flots d'objets

Un objet de classe ***java.io.ObjectOutputStream*** permet de produire un flot d'objets sérialisés. Inversement, un objet de classe ***java.io.ObjectInputStream*** permet d'obtenir un flot d'objets sérialisés. La ressource support du flot peut être un fichier, un tube ou un socket.

La classe ObjectOutputStream

Constructeur :

```
public ObjectOutputStream(OutputStream out) throws IOException;
```

Méthodes d'écriture d'un (graphe d')objet ou d'un type primitif:

```
public void writeObject(Object obj) throws IOException ;
public void writeInt(int data) throws IOException ;
public void writeDouble(double data) throws IOException;
...
```

Autres méthodes utiles:

```
public void close() throws IOException;
public void reset() throws IOException ;
```

La méthode ***reset*** permet de purger la mémoire qui conserve trace des objets déjà transmis, et qui sert en particulier à transmettre des graphes avec cycles. Il peut être nécessaire de remettre à zéro cette mémoire lorsque l'on souhaite transmettre une deuxième fois un objet dont l'état a changé. Si on ne réinitialise pas la mémoire du flot, seul un marqueur identifiant l'objet est transmis, et la désérialisation fournit l'objet dans l'état de la première transmission.

La classe ObjectInputStream

Constructeur :

```
public ObjectInputStream(InputStream in) throws IOException, StreamCorruptedException;
```

Méthodes de lecture d'un (graphe d')objet ou d'un type primitif :

```
public Object readObject() throws ClassNotFoundException, IOException;
public int readInt() throws IOException;
public double readDouble() throws IOException;
...
```

Exemple 1 : sérialisation automatique

Le code suivant construit et sauvegarde la structure présentée en introduction.

```
import java.io.ObjectOutputStream ;
import java.io.FileOutputStream ;
import java.io.IOException ;
import java.io.Serializable ;
class A implements Serializable {
    public B b ;
    public C c ;
}
```

```
class B implements Serializable {
    transient private int x = 3 ;
    private String s = "un" ;
    public B (int x, String s) { this.x = x ; this.s = s ; }
    public String toString() { return s + "/" + x ; }
}
class C implements Serializable {
    public int i ;
    public A a ;
    public C suiv ;
    public C (int i, A a, C suiv) { this.i = i ; this.a = a ; this.suiv = suiv ; }
}
public class Ecrivain {
    public static void main (String args[ ]) throws IOException {
        A a = new A() ;
        a.b = new B(123, "coucou" ) ;
        a.c = new C (28, null, new C (76, a, null)) ;
        System.out.println ( "B = "+ a.b) ;
        System.out.println ( "C1.i = "+ a.c.i) ;
        System.out.println ( "C2.i = "+ a.c.suiv.i) ;
        System.out.println ( "C3.i = "+ a.c.suiv.a.c.i) ;
        ObjectOutputStream flot = new ObjectOutputStream(new FileOutputStream( "/tmp/sauve")) ;
        flot.writeObject (a) ;
        flot.close() ;
    }
}
```

Voici le lecteur :

```
import java.io.FileInputStream ;
import java.io.ObjectInputStream ;
import java.io.IOException ;
public class Lecteur {
    public static void main (String[ ] args) throws IOException, ClassNotFoundException {
        ObjectInputStream flot = new ObjectInputStream(new FileInputStream("/tmp/sauve")) ;
        A a = (A) flot.readObject() ;
        System.out.println ( "B = "+ a.b) ;
        System.out.println ( "C1.i = "+ a.c.i) ;
        System.out.println ( "C2.i = "+ a.c.suiv.i) ;
        System.out.println ( "C3.i = "+ a.c.suiv.a.c.i) ;
    }
}
```

Déterminer ce qu'affichent à l'écran l'écrivain et le lecteur.

Exemple 2 : sérialisation partiellement manuelle

Un objet de classe ***B*** contient maintenant une référence à un objet de classe ***D,*** et la classe ***B*** réalise elle-même sa sérialisation.

```
import java.io.ObjectOutputStream ;
import java.io.ObjectInputStream ;
import java.io.FileOutputStream ;
import java.io.IOException ;
import java.io.Serializable ;
class A2 implements Serializable {
    public B2 b ;
    public C2 c ;
}
class B2 implements Serializable {
    private D2 d ;
    private int x = 3 ;
    private String s = "un" ;
    public B2 (int x, String s) {
        this.x = x ; this.s = s ;
        this.d = new D2() ; this.d.di = 345 ;
    }
    public String toString() { return s + "/" + x + "/"+ d.di ; }
    private void writeObject (ObjectOutputStream flot) throws IOException {
        flot.writeObject (this.s) ;
    }
    private void readObject (ObjectInputStream flot) throws IOException, ClassNotFoundException {
        this.d = new D2() ;
        this.s = (String) flot.readObject() ;
    }
}
class C2 implements Serializable {
    public int i ;
    public A2 a ;
    public C2 suiv ;
    public C2 (int i, A2 a, C2 suiv) { this.i = i ; this.a = a ; this.suiv = suiv ; }
}
class D2 /* doesn't implement Serializable */ {
    public int di = 31415 ;
}
```

```
public class Ecrivain2 {
    public static void main (String args[ ]) throws IOException {
        A2 a = new A2() ;
        a.b = new B2 (123, "coucou" ) ;
        a.c = new C2 (28, null, new C2 (76, a, null)) ;
        System.out.println ( "B = "+ a.b) ;
        System.out.println ( "C1.i = "+ a.c.i) ;
        System.out.println ( "C2.i = "+ a.c.suiv.i) ;
        System.out.println ( "C3.i = "+ a.c.suiv.a.c.i) ;
        ObjectOutputStream flot = new ObjectOutputStream(new FileOutputStream("/tmp/sauve")) ;
        flot.writeObject (a) ;
        flot.close() ;
    }
}
```

```
import java.io.FileInputStream ;
import java.io.ObjectInputStream ;
import java.io.IOException ;
public class Lecteur2 {
    public static void main (String[ ] args) throws IOException, ClassNotFoundException {
        ObjectInputStream flot = new ObjectInputStream(new FileInputStream("/tmp/sauve")) ;
        A2 a = (A2) flot.readObject() ;
        System.out.println ( "B = "+ a.b) ;
        System.out.println ( "C1.i = "+ a.c.i) ;
        System.out.println ( "C2.i = "+ a.c.suiv.i) ;
        System.out.println ( "C3.i = "+ a.c.suiv.a.c.i) ;
    }
}
```

Exemple 3 : accès successifs à un même objet

L'exemple suivant montre la mise en œuvre du mécanisme de sérialisation via un flot de données entre deux processus lourds communiquant par sockets.

L'objet sérialisable est un simple entier qui est régulièrement incrémenté par le processus émetteur du flot d'objets. Le processus récepteur se contente de lire les objets qu'il reçoit.

Classe d'objets sérialisables échangés

```
import java.io .* ;
/*   L'objet entier sérialisable est créé via le constructeur par défaut
     La méthode incr permet d'incrémenter sa valeur courante
     La méthode val permet de lire sa valeur courante
*/
public class EntierSerialisable implements Serializable {
     private int i = 0 ;
     public void inc() { i++; }
     public int val() { return i ; }
     public String toString() { return "INT =" + Integer.toString(i) : }
}
```

Le programme du processus récepteur

```
import java.io .* ;
import java.net.*;
/* Programme récepteur : écoute l'arrivée des objets via un socket sur le port 8080, et  affiche chaque objet reçu. */
public class Recepteur {
static final int port = 8080;
public static void main (String []args) {
     try { // Enregistrement du récepteur via le socket
          ServerSocket s = new ServerSocket (port);
          //  Attente de connexion de l'émetteur
          Socket socket = s.accept();
          // Création du flot d'entrée
          ObjectInputStream Entree = new ObjectInputStream(socket.getInputStream());
          //Boucle d'écoute
          whlle (true) {
               EntierSerialisable objLu = (EntierSerialisable) Entree.readObject();
               System.out.println (objLu);
          }
     } catch (Exception e) { System.out.println(e) ; }
}
}
```

Le programme du processus émetteur

```
import java.io .* ;
import java.net.*;
/*   Programme émetteur produt un flot continu d'objets à partir d'un objet unique, modifié entre chaque production.
     On suppose que l'appel du programme comporte l'adresse de la machine cible en paramètre
*/
public class Emetteur {
 static final int port = 8080;
 public static void main (String []args) {
     try { // Connexion par socket et création du flot de sortie
         InetAddress addr = InetAddress.getByName (args[0]);
         Socket socket = new Socket (addr, port);
         ObjectOutputStream sortie = new ObjectOutputStream(socket.getOutputStream());
         // Création de l'objet sérialisable
         EntierSerialisable objEmis = new EntierSerialisable();
         // Boucle d'envoi du même objet modifié
         while (true) {
             objEmis.inc() ;
             System.out.println (objEmis);
             sortie.reset () ;
             sortie.writeObject (objEmis);
         }
     } catch (Exception e) { System.out.println(e) ; }
 }
}
```

Exercices

1) Que se passe-t-il si l'on supprime l'appel ***sortie.reset()*** dans le programme Emetteur ?
2) Vérifier si lors de la réception d'un objet, il y a toujours création d'un nouvel objet sur le site récepteur, même si l'objet envoyé est le même.