Java

Conteneur

- Créez une classe GroupeEtudiant qui possède la méthode void ajout(Etudiant e). Les étudiants sont stockés dans un ArrayList.
- Dans le main d'une classe ClientEtudiant, ajoutez des objets de la classe Etudiant à la classe GroupeEtudiant. Au moins un de ces objets aura comme nom *Toto*.
- Dans la classe ClientEtudiant, rechercher *Toto*, en appelant la méthode List recherche(String nom) codée dans GroupeEtudiant.

```
public class Etudiant {
    private Integer numero;
    private String prenom;
    private String nom;
    public Etudiant(Integer numero, String prenom, String nom) {
    this.numero = numero;
    this.prenom = prenom;
    this.nom = nom; }
    public boolean aCeNom(String unNom) {return (nom.equals(unNom)); }
    @Override
    public String toString() {
    return super.toString() + "[" + nom + " " + prenom + ", n° " + numero + "]"; }}
```

```
public class GroupeEtudiant{
    private ArrayList<Etudiant> listEtudiant = new ArrayList<Etudiant>();
    public void ajout(Etudiant e){listEtudiant.add(e);}
    public ArrayList<Etudiant> recherche(String nom){
        ArrayList<Etudiant> result = new ArrayList<Etudiant>();
        for ( Etudiant e : listEtudiant ) {
            if (e.aCeNom(nom)){
                result.add(e);
            }
        }
        return result;}}
```

- Créez une classe MultiMap<K,V> qui associe à une clef de type K plusieurs valeurs de type V stockées dans une liste. Elle devra implémenter l'interface Map<K,List > et posséder une méthode void putOneValue(K,V) qui s'occupe de l'insertion d'une nouvelle valeur et boolean containsOneValue(V) qui retourne vraie si la valeur donnée se trouve dans le conteneur, faux sinon.

```
public class MultiMap<K,V> extends HashMap<K,List<V>> implements Map<K,List<V>>{
    public void  putOneValue(K key, V value){
        List<V> list;
        if(containsKey(key)){list = get(key);}
        else {list = new ArrayList<V>();}
        list.add(value);
        put(key, list); }
    public boolean containsOneValue(V value){
        for (Map.Entry<K, List<V>> e: entrySet()) {
            List<V> list = e.getValue();
            if (list.contains(value)) return true;
        }
        return false;}}
```

Flux

- Ecrivez une classe exécutable *Copy* qui réalise la copie d'un fichier dans un autre octet par octet, en respectant la convention suivante:
    - java Copy source cible pour copier source vers cible.
    - java Copy source pour écrire sur la sortie standard.
    - java Copy pour lire puis écrire sur la sortie standard.

```
public class CopyMessage {
    public static void main(String[] args){
        InputStream fin;
        OutputStream fout;
        try {
            switch (args.length){
                case 0:
                    fin = System.in;
                    fout = System.out;
                    Copy(fin, fout);
                    break;
                case 1:
                    fin = new FileInputStream(new File(args[0]));
                    fout = System.out;
                    Copy(fin, fout);
                    break;
                case 2:
                    fin = new FileInputStream(new File(args[0]));
                    fout = new FileOutputStream(new File(args[1]));
                    Copy(fin, fout);
```

```
                    break;
                default:
                    fin = new FileInputStream(new File(args[0]));
                    fout = new FileOutputStream(new File(args[1]));
                    break;}
        } catch(Error e){{…}}
    private static void Copy(InputStream is, OutputStream os) throws IOException {
        int val = 0;
        while (val != -1) {
            val = is.read();
            os.write(val);}
        is.close();
        os.close();}}
```

- Compilez et tapez la commande java EcrireMessage msg.ser. Que se passe-t-il ? -> écrit "toto date() salut" dans le un ficher msg.ser
- Ecrire une classe exécutable appelée *LireMessage* qui, en utilisant la méthode *readObject* de *ObjectInputStream*, lit le message stocké dans le fichier *msg.ser*, puis l'affiche sur la sortie standard.

```
public static void main(String args[]){
        if (args.length > 1){
            try {
                File src = new File(args[0]);
                FileInputStream fls = new FileInputStream(src);
                ObjectInputStream ols = new ObjectInputStream(fls);

                Object message = ols.readObject();
                System.out.print(message);
                ols.close();
                fls.close();
            }
            catch(Error e){…}}}}
```

- Ecrivez une classe *Serveur* écrivant sur la sortie standard les lignes de textes envoyées par le client. Comme vous devez procéder ligne par ligne, vous allez envelopper le flux d'entrée dans un *BufferedReader* comme dans l'exemple précédent.
- Testez votre serveur avec telnet: telnet localhost 8080

```
public class Server {
    public static void main (String args[]){
        int numeroPort = 8089;
        try{
            ServerSocket connection = new ServerSocket(numeroPort);
            Socket socket = connection.accept();
            System.out.println(« connection to »+ socket.getInetAddress());
            BufferedReader reader  = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String line = reader.readLine();
            while (line != null){
                System.out.println(line);
                line = reader.readLine();}
            connection.close();
        }catch (IOException e){…}}}
```

- Ecrivez une classe *Client* qui envoie des lignes de texte lues sur l'entrée standard au serveur. Comme vous devez procéder ligne par ligne, vous allez envelopper le flux de sortie dans un *PrintWriter* comme dans l'exemple précédent. Cependant, activez le *flush* automatique avec l'argument supplémentaire *true*

```
public class Client {
    public static void main(String[] args) throws IOException {
        int numeroPort = Integer.parseInt(args[0]);
        Socket socket = new Socket(« localhost », numeroPort);
        OutputStream os = socket.getOutputStream();
        int val = 0;
        while (val != -1) {os.write(val);}
        socket.close();}}
```

Réflexité

Dans XRayClass on retrouve par exemple

```
    // ? super T autorize toutes les class parents de T
    public <T> ArrayList<Class<? super T> >
        getClassesFromClass(Class<T> c) {
        ArrayList<Class<? super T> > cList = new ArrayList<Class<? super T> >();
        Class<? super T>  cTemp;
        cTemp = c;
        while ((cTemp = cTemp.getSuperclass()) != null) {
            cList.add(cTemp);}// Add all parent
        return cList;}
```