```
/* APOO : TD4, Graphe, implementation par ensemble de successeurs*/

import java.util.Collection ;
import java.util.* ;

/* On implemente le graphe par une collection de Sommet,
 Chaque Sommet contenant la collection de ses successeurs.
 Un sommet contient un attribut, fourni par l'utilisateur, qui respecte l
'interface InfoSommet
 */

interface InfoSommet
{
    String toString() ;
}

class Sommet {
        // Un sommet ici contient une Collection de successeurs.
        // On peut ensuite choisir 2 implementations
        // - Un Set : voir le constructeur ci dessous : on ne peut pas
representer des arcs multiples;
        // - Une liste : On peut avoir plusieurs fois le meme sommet
successeur et donc plusieurs arcs
        //      entre les 2 memes sommets (attention ce n'est plus un
vrai graphe, mais peut etre
        //      interessant pour representer par exemple un reseau de
fibres optiques):
    private Collection< Sommet> succ ;

    // Le sommet a une InfoSommet  qui, si elle existe, identifie le
sommet
    // Sinon un sommet est identfiÈ par sa liste de successeurs.
    private InfoSommet  info ;

    public String toString() { return ( info == null) ? "" : info.
toString() ; }

    public Sommet()
    {
    // Ici, on choisit un Set, donc un seul arc entre 2 sommets avec succ
= new HashSet< Sommet > () ;
    // Sinon succ = new LinkedList<Sommet>();
    // Et c'est la seule chose a changer dans le code qui suit, car on
utilise des methodes communes à List et Set
    succ = new HashSet< Sommet > () ;
    /* succ = new LinkedList<Sommet>(); */ // A decommenter selon le
choix;
    info = null ;
    }

    public Sommet(  InfoSommet  userinfo )
    {
    succ = new HashSet< Sommet > () ;
    /* succ = new LinkedList<Sommet>(); */ // A decommenter selon le
choix;
    info = userinfo ;
    }
```



```
      // equals et hashCode doivent etre coherents :  celles par defaut
sont correctes.
      // Sinon, on pourrait eventuellement dire que deux sommets qui sont
associÈs ‡ la mÍme Info sont Ègaux.
      // Ca permettrait d'introduire instanceof et les conversions
      public boolean equals ( Object o )
      {
    if (info == null) return (o instanceof Sommet) && ( succ.hashCode() =
= ((Sommet) o).succ.hashCode() ) ;
    return (o instanceof Sommet) && (info.equals (((Sommet) o).info) ) ;
      }

      public int hashCode () { return (info == null) ? succ.hashCode() :
info.hashCode (); }

        // Avec un Set, on pourrait tester si l'arc, donc le sommet s est
deja présent dans les successeurs et remonter une exception
    public void addSucc ( Sommet v ) { succ.add( v );}
        // La methode remove ne supprime qu'un seul element de la liste
si plusieurs sont presents.
        // Donc, une boucle est necessaire pour une implementation
alternative liste/set
    public void removeSucc ( Sommet v ) { while (succ.remove( v )); }
    public void removeAll ( ) { succ.clear() ; }

        // Suite a la question : peut on faire retourner un objet
constant comme en C++, la reponse
        // est oui, en utilisant Collections.unmodifiableCollection.
        // Avec un accesseur en lecture, c'est plutot mieux
        // Attention cependant, elle n'est pas synchronisee.
        // Remarque ; getSucc retourne une collection, ce qui ne presage
pas de l'implantation par une liste ou un Set
    Collection< Sommet >  getSucc() { return Collections.
unmodifiableCollection(succ) ; }

        // utilise par la version 2 de existe chemin : marquage explicite
    private boolean marque = false;
    boolean marque () { return marque; }
    void demarquer () { marque = false; }
    void marquer () { marque = true; }
}


class GrapheLinked {
        // Le graphe est implante par un ensemble de Sommet.
    private Set< Sommet > sommets ;

    public GrapheLinked() { sommets = new HashSet< Sommet >  () ; }

    public Sommet addSommet() { Sommet s = new Sommet() ; sommets.add(s)
; return s ; }
    public Sommet addSommet(InfoSommet inf) { Sommet s = new Sommet(inf)
; sommets.add(s) ; return s ; }

    public boolean removeSommet(Sommet k)   {
      if (! sommets.contains( k ) ) return false ;
```

```
        // En detruisant le sommet, il faut detruire la liste de ses
successeurs.
        // Quelqu'un a suggéré d'avoir le lien inverse pour ne pas
rechercher sequentiellement
        // On detruit toutes les occurences de v comme successeur d'un
autre noeud
      for (Sommet s : sommets ) s.removeSucc(k) ;
      sommets.remove( k ) ;
      return true ;
    }

    public void addArc(Sommet s, Sommet d  ) {
      if ( sommets.contains( s ) && sommets.contains( d ) )  s.addSucc( d
) ;
    }

    public void removeArc(Sommet s, Sommet d  ) {
      if ( sommets.contains( s ) && sommets.contains( d ) ) s.removeSucc(
d ) ;
    }

    boolean isEmpty() { return sommets.isEmpty() ; }

    public void clear() { sommets.clear() ;}

    public String toString() {  String res  = "[{" ;
           // Affiche les sommets
      for (Sommet s: sommets ) res += s + " " ;
      res += "},{" ;
           // Affiche les transitions
      for (Sommet s: sommets ) {
        for (Sommet d: s.getSucc() ) res += "(" + s + "->"  + d + ")" ;
      }
      return res + "}]" ;
    }

        // existence de chemin : version avec HashSet contenant les
sommets deja examines
        // La methode recursive est privee et non visible
    public boolean existeChemin (Sommet s, Sommet d) {
        HashSet< Sommet> DejaVisites =  new HashSet< Sommet >()  ;
        boolean res = exploreRec (s, d, DejaVisites ) ;
        DejaVisites = null; // On supprime le tableau internmediaire
        return res ;
    }
    private boolean exploreRec (Sommet s, Sommet d, Collection< Sommet >
dejaVus ) {
      if (s.equals( d ))  return true ;
      dejaVus.add ( s ) ;
      for (Sommet succ: s.getSucc() )
        if (!dejaVus.contains(succ) && exploreRec( succ, d, dejaVus ))
return true ;
      return false ;
    }

        // existence de chemin : version avec marquage explicite
        // Les methodes auxiliaires sont privees et non visibles
```



```
        // Par rapport a la version precedente, doit etre Synchronized
(effet de bord sur le boolean marque)
    synchronized boolean existeChemin2 (Sommet s1, Sommet s2) {
        if (!sommets.contains (s1) || !sommets.contains (s2))
            return false;
        else {
                for(Sommet s : sommets) s.demarquer ();
            return existeC (s1, s2);
        }
        }
        private boolean existeC (Sommet s1, Sommet s2) {
        if (s1.equals (s2)) return true;
        s1.marquer ();
        for( Sommet s : s1.getSucc())
            if (!s.marque () && existeC (s, s2)) return true;
        return false;
        }

}

////////////////////////////////////////////////////////
// Un bout de test, bien maigre.

class MyInfo implements InfoSommet
{
  String nom;
  MyInfo( String name ) { nom = name ; }
  public String toString() { return nom ; }
}

public class GrapheLinkedMDJL
{
    public static void main(String args[]) {
        GrapheLinked G = new GrapheLinked() ;
                // Creation des sommets
        Sommet v1 = G.addSommet(new MyInfo( new String("A1") ) )  ;
        Sommet v2 = G.addSommet(new MyInfo( "A2" ) )  ;
        Sommet v3 = G.addSommet(new MyInfo( "A3" ) )  ;
        Sommet v4 = G.addSommet(new MyInfo( "A4" ) )  ;
        Sommet v5 = G.addSommet(new MyInfo( "A5" ) )  ;
        Sommet v6 = G.addSommet(new MyInfo( "A6" ) )  ;
            // Creation des arcs
        G.addArc(v1, v2) ;      G.addArc(v2, v3) ;      G.addArc( v1, v3
);      G.addArc(v3, v4) ;
        G.addArc(v4, v3) ;      G.addArc(v4, v5) ;      G.addArc(v6, v4)
;       G.addArc(v5, v1) ;
            // Affichage du graphe
        System.out.println( "Devrait imprimer: [{1 2 3 4 5 6 }, {(1->3)(1
->2)(2->3)(3->4)(4->3)(4->5)(5->1)(6->4)}]");
        System.out.println( "G = " + G ) ;
            // Test des ExisteCheminssss
        System.out.println( "ExisteChemin de 1 a 5 (en theorie OUI): " +
G.existeChemin( v1, v5 ) ) ;
        System.out.println( "ExisteChemin2 de 1 a 5 (en theorie OUI): " +
G.existeChemin2( v1, v5 ) ) ;
        System.out.println( "ExisteChemin de 1 a 6 (en theorie NON): " +
G.existeChemin( v1, v6 ) ) ;
```

```
        System.out.println( "ExisteChemin2 de 1 a 6 (en theorie NON): " +
G.existeChemin2( v1, v6 ) ) ;

            // Tentative de mettre 2 fois le meme arc !!
            // Deux arcs A2 A3 si on travaille avec une liste, un seul
sinon
        G.addArc(v2, v3) ;
        System.out.println("On ajoute 2 fois l'arc 2->3 : possible avec
une liste, impossible avec un Set");
        System.out.println( "G = " + G ) ;

            // Suppression du sommet A3 : verifier qu'aucun successeur ne
contient A3
        G.removeSommet(v3) ;
        System.out.println( "... Apres Suppression de 3:" ) ;
        System.out.println( "Devrait imprimer: [{1 2 4 5 6 }, {(1->2)(4->
5)(5->1)(6->4)}]" ) ;
        System.out.println( "G = " + G ) ;
            // Il n'existe plus de chemin entre A1 et A5
        System.out.println( "ExisteChemin de 1 a 5 (en theorie NON): " +
G.existeChemin( v1, v5 ) ) ;
        System.out.println( "ExisteChemin de 1 a 6 (en theorie NON): " +
G.existeChemin( v1, v6 ) ) ;

        G.removeArc(v5, v1) ;
        System.out.println( "... Apres Suppression de l'arc (5,1) " ) ;
        System.out.println( "Devrait imprimer: [{1 2 4 5 6 }, {(1->2)(4->
5)(6->4)}]" ) ;
        System.out.println( "G = " + G ) ;
        System.out.println( "ExisteChemin de 1 a 5 (en theorie NON): " +
G.existeChemin( v1, v5 ) ) ;
        System.out.println( "ExisteChemin de 1 a 6 (en theorie NON): " +
G.existeChemin( v1, v6 ) ) ;

            // Tentative de suppression d'un opu de plusieurs successeurs
de v1 à travers l'accesseurs getSucc de la classe sommets
            // Exception java.lang.UnsupportedOperationException à
'execution
            // Attention : ca n'empeche cependant pas de muter les
contenus des objets de la liste
        System.out.print("Suppresion des successeus de v1   ");
        System.out.println(": devrait afficher une exception de type java
.lang.UnsupportedOperationException");
        v1.getSucc().clear(); // Va planter car on a pris une
unmodifiableCollection
        System.out.println(v1);
    }
}
```