```
import java.util.*;
import java.lang.*;

public class L {
        Object head;
        L tail;
        public static L nil=null;
        public static boolean isempty( L l) { return null==l; }
        public static Object hd( L l) { return l.head; }
        public static L tl( L l) { return l.tail; }
        public static L cons( Object o, L q) { L l=new L(); l.head=o; l.tail=q;
return l;}
        public static L map( L l, F f)
        {       if (nil== l) return l;
                return cons( f.eval( hd(l)), map( tl(l), f));
        }
        public static Object fold_left( Operation op, Object sivide, L l)
        {       if (nil==l) return sivide;
                return fold_left( op, op.eval2( sivide, hd(l)), tl(l));
        }

        public static L iaj( int i, int j)
        {       if( i==j) return cons( new Integer(i), nil);
                else return  cons( new Integer(i), iaj( 1+i, j));
        }

        public static void printL( L l)
        {       System.out.print("[");
                for( L tmp=l; null != tmp ; tmp=tl(tmp)) System.out.print( hd(tm
p) +";");
                System.out.println("]");
        }

        public static L filter( L l, F pred)
        {
                if (nil==l) return nil;
                Boolean boo = (Boolean) pred.eval( hd(l));
                if (boo.booleanValue())
                        return cons( hd(l), filter( tl(l), pred));
                else return filter( tl(l), pred);
        }
        public static L fusion( L a, L b, final Operation comp)
        {
                if (null==a) return b;
                if (null==b) return a;
                Integer comparaison= (Integer) comp.eval2( hd( a), hd( b));
                int cas= comparaison.intValue();
                if (cas < 0) return cons( hd( a), fusion( tl(a), b, comp));
                return cons( hd( b), fusion( a, tl(b), comp));
        }
        public static L halflist( L l) // un element sur 2
        {       if (null==l) return nil;
                if (null==tl(l)) return cons( hd(l), nil);
                return cons( hd(l), halflist( tl( tl( l))));
        }
        public static L trifusion( L l, final Operation comp)
        {
                if (null==l || null==tl(l)) return l;
                return fusion( trifusion( halflist( l), comp),
                                trifusion( halflist( tl( l)), comp), comp);
        }
        public static L qksort( L l, final Operation comp)
        {
                if (null==l) return l;
                final Object  pivot = hd( l);
                F pluspetit= new F() { public Object eval( Object o)
                        { Integer oo= (Integer) comp.eval2( o, pivot);
                          int cas = oo.intValue();
                          return new Boolean( 0 > cas);
```

```
                         }} ;
                F plusgrand= new F() { public Object eval( Object o)
                        { Integer oo= (Integer) comp.eval2( o, pivot);
                          int cas = oo.intValue();
                          return new Boolean( 0 < cas);
                         }} ;
                F egal = new F() { public Object eval( Object o)
                        { Integer oo=  (Integer) comp.eval2( o, pivot);
                          int cas = oo.intValue();
                          return new Boolean( 0 ==cas);
                         }} ;
                L petits = filter( tl(l), pluspetit);
                L grands = filter( tl(l), plusgrand);
                L egaux  = filter( l, egal);
                return concat( qksort( petits, comp),
                                concat( egaux, qksort( grands, comp)));
        }
        public static L concat( L a, L b)
        {       if (null==a) return b;
                if (null==b) return a;
                return cons( hd(a), concat( tl(a), b));
        }

        public static int alea( int n)
        {       double nn= n;
                double hasard = Math.random() * nn;
                int a= (int) hasard;
                return a;
        }
public static void main (String[] args)
{
        L lalea=null;
        for( int i=0; i<30; i++)        lalea= cons( alea(10000), lalea);
        printL( lalea);

// pour trier dans ordre croissant
        Operation compare= new Operation() {
                public Object eval2( Object a, Object b)
        { Integer aa= (Integer) a;
          Integer bb= (Integer) b;
          int aaa= aa.intValue();
          int bbb= bb.intValue();
          if (aaa<bbb) return new Integer( -1);
          if (aaa==bbb) return new Integer( 0);
          return new Integer( 1);
        }
        };
        L laleatriee= qksort( lalea, compare);
        printL( laleatriee);

// pour trier dans ordre decroissant
        Operation compare2= new Operation() {
                public Object eval2( Object a, Object b)
        { Integer aa= (Integer) a;
          Integer bb= (Integer) b;
          int aaa= aa.intValue();
          int bbb= bb.intValue();
          if (aaa<bbb) return new Integer( 1);
          if (aaa==bbb) return new Integer( 0);
          return new Integer( -1);
        }
        };
        L laleatriee2= qksort( lalea, compare2);
        printL( laleatriee2);


// tri par fusion et ordre croissant
        L ltrieefusion = trifusion( lalea, compare);
        printL( ltrieefusion);
        L liste= iaj( 1, 9);
        printL( liste);
        F pair = new F() { public Object eval( Object o)
                        {       Integer oo=(Integer) o;
                                return new Boolean( 0== ((oo.intValue()) % 2));
                        } };
        L pairs = filter( liste, pair);
        printL( pairs);
        F impair = new F() { public Object eval( Object o)
                        {       Integer oo=(Integer) o;
                                return new Boolean( 1== ((oo.intValue()) % 2));
                        } };
        L impairs = filter( liste, impair);
        printL( impairs);
        F2 f2= new F2();
        L l2 = map( liste, f2);
        printL( l2);
        Fact fa= new Fact() ;
        L l = map( liste, fa);
        printL( l);
        final int K= 100; // declare final car java le demande...
        F fonc= new F() { public Object eval( Object o)
                                {       Integer oo=(Integer) o;
                                        int i = oo.intValue();
                                        return new Integer( K*i);
                                }
                        };
        l = map( liste, fonc);
        printL( l);

        Object sum= fold_left( new Plus(), new Integer(0), iaj( 1, 10));
        System.out.println( "somme de 1 Ã  10 = " + sum.toString());

{
        System.out.println( "erreur Ã  l’exÃ©cution: javac ne peut pas detecter l’erreur Ã  la compilatio
n, car il n’y a pas d’infÃ©rence de types en java (Le langage OCaml et d’autres langages fonctionnels fournissent l’inf
Ã©rence de types). Il faudrait utiliser la programmation gÃ©nÃ©rique Java pour que l’erreur soit (peut−Ãªtre) dÃ©te
ctÃ©e Ã  la compilation. Mais ni la programmation gÃ©nÃ©rique ni la programmation fonctionnelle (en java 8) ne pe
rmettent des programmes simples et courts... ");
        L bug=qksort( lalea, new Operation() {
                public Object eval2( Object a, Object b)
        { Double aa= (Double) a;
          Double bb= (Double) b;
          double aaa= aa.doubleValue();
          double bbb= bb.doubleValue();
          if (aaa<bbb) return new Integer( 1);
          if (aaa==bbb) return new Integer( 0);
          return new Integer( −1);
        }});
}
}
}

class F { public //static
                Object
                eval( Object o) {return o; }}

class F2 extends F { public Object eval( Object o) {
                return new Integer( (((Integer)o).intValue()) * 2);
        }}

class Fact extends F
{       public Object eval( Object o)
        {
        System.out.println( "I AM CALLED");
        Integer oo= (Integer) o;
        int n = oo.intValue();
        int i=1; int fi=1;
```

```
        for (; i<= n; fi *= i, i++);
        return new Integer( fi);
        }
}

class Operation { public Object eval2( Object a, Object b) {return a; }}
class Plus extends Operation { public Object eval2( Object a, Object b)
        { Integer aa= (Integer) a;
          Integer bb= (Integer) b;
          return new Integer( (aa.intValue()) + (bb.intValue()));
        }}
class Mult extends Operation { public Object eval2( Object a, Object b)
        { Integer aa= (Integer) a;
          Integer bb= (Integer) b;
          return new Integer( (aa.intValue()) * (bb.intValue()));
        }}
class Mod extends Operation { public Object eval2( Object a, Object b)
        { Integer aa= (Integer) a;
          Integer bb= (Integer) b;
          return new Integer( (aa.intValue()) % (bb.intValue()));
        }
}
```