public static String josephus( int n ) {

roundTable rt = new roundTable( n ); // configurazione iniziale

while (rt.getSize() >2){

rt = rt.serveNeighbour();

rt = rt.passJug();

}

return rt.servingKnights().ListToString(); }

public static void main( String args[] ) {

System.out.println( josephus(1500)); }

package com.company;

public class roundTable {

// VARIABILI DI ISTANZA

private final int size; // numero di cavalieri totale

private final int jug; // id del cavaliere con la brocca

private final IntSchemeList head; //lista dei cavalieri successivi a quello con la brocca

private final IntSchemeList tail; // lista dei cavalieri che hanno già passato un giro

public roundTable(int n){

size = n;

jug = 1;

head = IntSchemeList.creaListaRange(2,n);

tail = IntSchemeList.NULL\_INTLIST;

}

private roundTable(int n, int j, IntSchemeList h, IntSchemeList t){

size = n;

jug = j;

head = h;

tail = t;

}

public int getJug() {return jug; }

public int getSize() { return size; }

public roundTable serveNeighbour() { // serve il commensale vicino a sinistra:

// il commensale servito lascia la tavola

if (size <= 2) return this;

else if (head.listLength() < 2){

roundTable rt = new roundTable(size, jug, head.append(tail.reverse()), IntSchemeList.NULL\_INTLIST);

return rt.serveNeighbour();

} return new roundTable(size-1,jug, head.cdr().cdr(), tail.cons(jug).cons(head.car()));

}

public IntSchemeList servingKnights(){

if (tail.isNull()) return head.cons(jug);

return tail.reverse();

}

public roundTable passJug() { // passa la brocca al commensale vicino

return (size <= 2)?

this :

new roundTable(size, head.car(),head.cdr().append(tail.reverse()),IntSchemeList.NULL\_INTLIST);

}

public String getTail() { return tail.reverse().ListToString(); }

public String getHead() { return head.ListToString(); }

}