1. Sea S un conjunto formado por $n > 0$ elementos. Como es conocido, con los elementos de ese conjunto pueden construirse $n!$ *permutaciones*. Por ejemplo, si S es el conjunto $\{1, 2\}$ sus permutaciones son:

1	2
2	1

   Análogamente, si es el conjunto $\{1, 2, 3\}$

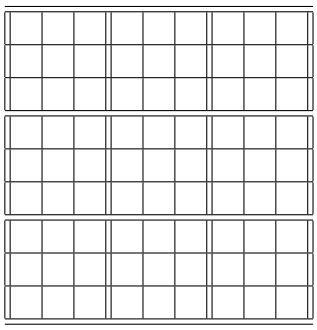
1	2	3
1	3	2
3	1	2
2	1	3
2	3	1
3	2	1

   y, finalmente, si es el conjunto $\{1, 2, 3, 4\}$

1	2	3	4
1	2	4	3
1	4	2	3
4	1	2	3
		⋮	
3	2	1	4
3	2	4	1
3	4	2	1
4	3	2	1

   Si se revisa el problema atentamente, se verá que a partir de las permutaciones del conjunto $\{1\}$, se pueden construir fácilmente las permutaciones del conjunto $\{1, 2\}$. Sabidas las permutaciones del conjunto $\{1, 2\}$, se pueden construir las permutaciones del conjunto $\{1, 2, 3\}$; y así sucesivamente. El programa deberá mostrar las permutaciones formadas por las letras de una palabra (a elegir por el usuario?).

2. Como primer paso para desarrollar un programa para el juego del Sudoku, vd. ha sido encargado de crear una clase `Sudoku`. Los objetos de esa clase representarán *cuadrículas* como la siguiente:



   Cuadrícula: Conjunto de los cuadrados que resultan de cortarse perpendicularmente dos series de rectas paralelas.

   En la versión habitual del juego, cada cuadrado o *casilla* de la cuadrícula puede estar libre u ocupada por un entero entre uno y nueve. Además, las reglas del juego consideran que la cuadrícula está dividida en nueve subcuadrículas iguales, con dimensiones $\sqrt{9} \times \sqrt{9}$. Tras un estudio previo se ha llegado a la conclusión de que interesa lanzar una variante del juego en la cual las dimensiones de la cuadrícula no sean necesariamente 9×9, con las casillas distribuidas en 9 filas

y 9 columnas. En su lugar, esas dimensiones podrán variar de unas cuadrículas a otras, aunque siempre deberán ser iguales al cuadrado de un entero; Así pues, las dimensiones de una cuadrícula podrán ser $3^2 \times 3^2$, $4^2 \times 4^2$, $5^2 \times 5^2$, ... La adaptación de las reglas del juego al nuevo escenario es sencilla. Una cuadrícula con $x^2 \times x^2$ casillas (distribuidas en x^2 filas y x^2 columnas) contiene x^2 subcuadrículas iguales, con dimensiones $x \times x$, y sus casillas deben rellenarse con números entre uno y x^2. Finalmente, la posición de cada cuadrado de una cuadrícula se identificará mediante dos enteros, mayores o iguales que cero, y que corresponderán a su abscisa (el número de la columna en la que está la casilla,) y su ordenada (el número de la columna en la que está la casilla.)

Por diversas razones, vd. debe crear urgentemente un primer prototipo de la clase `Sudoku` de manera que los objetos de esa clase sean capaces de memorizar las dimensiones de la cuadricula, así como el contenido de cada casilla. Esos objetos deben tener funciones para:

- cambiar las dimensiones de la cuadrícula
  después de esta operación, todas las casillas quedarán vacías
- cambiar el contenido de una casilla cualquiera, dejándola vacía
  después de esta operación, la casilla referida quedará vacía
- cambiar el contenido de una casilla cualquiera, almacenando en ella un entero dado
  después de esta operación, en la casilla referida quedará apuntado ese valor, sin tener en cuenta eventuales violaciones de las reglas del juego
- averiguar cuántas casillas tiene la cuadrícula
  el resultado de esta operación será el número total de casillas de la cuadrícula
- averiguar el contenido de una casilla cualquiera
  el resultado de esta operación será -1 si la casilla referida está vacía, o bien si las coordenadas indicadas no corresponden a ninguna casilla

A continuación se muestra un posible ejemplo de uso de esa clase:

```
package ehu.student;

public class UsoSudoku {

    public static void main(String[] args) {
        int dim = 4*4;

        Sudoku tablero = new Sudoku();

        tablero.redimensionar(dim);

        int totalCasillas = tablero.casillas();
        if (totalCasillas != dim){
            System.out.println("Error! hay: " + totalCasillas + " casillas !!!");
        }

        int v = tablero.queIntEn(0, 0);
        if (v == -1){
            System.out.println("Casilla(0, 0) vacia");
        } else {
            System.out.println("Error! Casilla(0, 0): " + v + "!!!");
        }

        tablero.ponerEn(0, 0, 1);
        int w = tablero.queIntEn(0, 0);
        if (w == 1){
            System.out.println("Casilla(0, 0): " + w);
        } else {
            System.out.println("Error! Casilla(0, 0): " + w + "!!!");
        }
    }
}
```

3. Tras haber cumplido exitosamente los objetivos del apartado anterior, vd. recibe el encargo de modificar la clase `Sudoku` de manera que el intento de colocar un valor en una casilla vacía deje la cuadrícula inalterada, si esa acción violar las restricciones del juego. Además, los objetos de esa clase tendrán funciones para:

   - averiguar cuáles son todos los valores que podrían colocarse en una casilla vacía sin violar las restricciones del juego
     el resultado de esta operación será un conjunto con todos los valores que cumplen los requisitos indicados; o bien el valor especial `null`, si la casilla referida no está vacía, o las coordenadas indicadas no corresponden a ninguna casilla. Si alguna de las casillas contiene un valor que viola las restricciones del juego, el resultado de esta operación está indefinido.

   A continuación se muestra un posible ejemplo de uso de la nueva versión:

```java
package ehu.student;

import java.util.TreeSet;

public class UsoSudoku {

    public static void main(String[] args) {
        int dim = 3*3;

        Sudoku tablero = new Sudoku();
        tablero.redimensionar(dim);
        tablero.ponerEn(0, 0, 1);
        tablero.ponerEn(1, 1, 2);
        tablero.ponerEn(1, 1, 3);

        TreeSet<Integer> conjto_0_0 = tablero.quePosibilidadesEn(0, 0);
        if (conjto_2_0 == null){
           System.out.println("Error! resultado no es null!!!");
        }

        TreeSet<Integer> conjto_2_0 = tablero.quePosibilidadesEn(2, 0);
        /*
         * El conjunto de valores posibles en (2, 0) debiera ser:
         *       4-5-6-7-8-9
         */

        TreeSet<Integer> conjto_8_8 = tablero.quePosibilidadesEn(8, 8);
        /*
         * El conjunto de valores posibles en (8, 8) debiera ser:
         *       1-2-3-4-5-6-7-8-9
         */
    }
}
```

4. Supóngase que un número de archivos de datos contienen valores enteros y que, además, los valores contenidos en cada archivo, no necesariamente distintos entre sí, están ordenados crecientemente. Vd. ha recibido el encargo de escribir un programa para copiar en un archivo *todos* los valores contenidos en los referidos archivos de datos, de tal manera que los valores contenidos en el archivo resultante estén también ordenados crecientemente. A manera de ejemplo, a continuación se muestra el contenido de tres hipotéticos de datos y el contenido del archivo con la copia referida.

archivo 1	*archivo 2*	*archivo 3*	*archivo resultante*
127	1	1	1
127	1024	255	1
255			127
255			127
			255
			255
			255
			1024

Por diversas razones, está prohibido almacenar todo el contenido de uno o varios de los archivos referidos en objetos `ArrayList<Integer>` o similares; además, Vd. debe completar la clase siguiente:

```
package ehu.student;

import java.util.ArrayList;
import java.io.Scanner;
import java.io.PrinStream;

public class Mezclador {

    public void procesar(ArrayList<Scanner> datos, PrinStream resultado) {
    }
}
```

A continuación se esboza un posible ejemplo de uso de esa clase:

```
package ehu.student;

public class UsoMezclador {

    public static void main(String[] args) {
        Scanner sc_0 = null;
        Scanner sc_1 = null;
        Scanner sc_2 = null;
        PrintStream ps = null;
        /*
         * Creacion de objetos y asignacion a sc_0, sc_1, sc_2, ps
         */

        ArrayList<Scanner> lst = new ArrayList<Scanner>();
        lst.add(sc_0);
        lst.add(sc_1);
        lst.add(sc_2);

        Mezclador opr = new Mezclador();
        opr.procesar(lst, ps);
    }

}
```