DATOS PERSONALES		FIRMA
Nombre: DNI:		
Apellidos:		
ESTUDIO	**ASIGNATURA**	**CONVOCATORIA**
MÁSTER UNIVERSITARIO EN INGENIERÍA MATEMÁTICA Y COMPUTACIÓN (PLAN 2016)	4391010005.- MÉTODOS AVANZADOS DE PROGRAMACIÓN CIENTÍFICA Y COMPUTACIÓN	Extraordinaria
FECHA	**MODELO**	**CIUDAD DEL EXAMEN**
10-12/09/2021	Modelo - D	
Etiqueta identificativa		

INSTRUCCIONES GENERALES

1. La duración del examen es de **2 horas**.
2. Escribe únicamente con **bolígrafo/esfero azul o negro**.
3. No está permitido utilizar más hojas de las que te facilita la UNIR (puedes utilizar folios para hacerte esquemas u organizarte pero **se entregarán junto al examen**).
4. **El examen** PRESENCIAL **supone el 60%** de la calificación final de la asignatura. Es necesario aprobar el examen, para tener en cuenta la evaluación continua, aunque esta última sí se guardará para la siguiente convocatoria en caso de no aprobar.
5. No olvides **rellenar EN TODAS LAS HOJAS los datos del cuadro** que hay en la parte superior con tus datos personales.
6. El **DNI/NIE/PASAPORTE debe estar sobre la mesa** y disponible para su posible verificación.
7. **Apaga y retira del alcance los teléfonos móviles**.
8. **Retirar del alcance y visibilidad el smartwatch**.
9. Las preguntas se contestarán en **CASTELLANO**.
10. El profesor tendrá muy en cuenta las **faltas de ortografía** en la calificación final.
11. Los gráficos y el código los puede realizar a mano e insertar en el documento una foto de lo realizado. En este caso, también debe ceñirse al espacio indicado para cada pregunta, no pudiendo reducirse la foto para que parezca que se ha ocupado menos espacio.

Puntuación

PREGUNTAS TEORICO PRÁCTICAS

- Puntuación máxima 3.00 puntos

PROGRAMACIÓN Y DISEÑO ORIENTADO A OBJETOS

- Puntuación máxima 3.00 puntos

PROGRAMACIÓN CONCURRENTE

- Puntuación máxima 4.00 puntos

- **Cada una de las tres preguntas debe ser desarrollada como máximo en una cara.**
- **Cada pregunta vale 1 punto.**

1. Explique el mecanismo de excepciones en Java y los elementos que se utilizan. Ponga un ejemplo sencillo de un manejador de una excepción definida por el programador.

(1 punto)

(Responder en 1 caras)

2. Defina monitor en Java y sus características, desde el punto de vista de la concurrencia. Explique los métodos que se deben definir y ponga un ejemplo significativo.

(1 punto)

(Responder en 1 caras)

3.

Explique e implemente el patrón Factory.

Construya el diagrama de clases correspondiente, ponga un ejemplo de uso e impleméntelo .

(1 punto)

(Responder en 1 caras)

1. Se necesita modelar un sistema de un gabinete jurídico. Las características que debe cumplir son:

Código de examen: 162955

a. Los clientes tienen como información asociada mínima el nombre, apellido, y/o DNI. La búsqueda de clientes se puede realizar introduciendo cualquiera de estos datos.

b. Los expedientes que gestiona el gabinete, están asignados a uno o más clientes (puede ocurrir que dos o más personas pueden estar involucradas en el mismo caso). Así mismo, un cliente puede tener más de un expediente abierto.

c. Los expedientes pueden ser de varios tipos: derecho civil, derecho laboral y derecho inmobiliario. Los clientes pueden tener expedientes abiertos de varios tipos.

d. Los empleados del gabinete gestionan varios expedientes aunque cada empleado está especializado en un tipo.

e. Los expedientes cuentan con fecha de inicio del expediente, estado y el juzgado donde se tramita.

f. Además, es preciso mantener los jueces pertenecientes a cada juzgado, los secretarios, y un teléfono de contacto de cada juzgado.

1. Realice el diseño de clases en UML e identifique, al menos, dos atributos y dos métodos para cada clase. Implemente las clases. Los métodos no es necesario implementarlos, es suficiente con la cabecera.
2. Identifique las relaciones y su cardinalidad. Impleméntelas y explique y justifique la implementación.
3. Haga un esbozo de la implementación de la función obtener_expedientes de un cliente con todos los datos necesarios para conocer su estado, el juez que lo lleva y el empleado que lo gestiona. Implemente el programa principal que la usa.

(3 puntos)

(Responder en 2 caras)

1. Se quiere implementar un sistema de mensajería de un club social mediante un tablón virtual compartido.

a. Los socios podrán leer simultáneamente el mensaje colgado en el tablón compartido.

b. Como máximo dos socios pueden escribir simultáneamente en el tablón.

c. Las acciones de lectura solo están disponibles cuando no haya socios escribiendo en el tablón.

Implemente en Java las clases necesarias para permitir, desde el punto de vista de la concurrencia, la gestión del tablón de mensajes respetando las condiciones anteriores.

(4 puntos)

(Responder en 3 caras)

Pregunta 1:

El mecanismo de excepciones de java es util para poder crear “interrupciones” cuando se genera un error, o cuando necesitamos hacer un cambio. Java normalmente cuenta con muchos tipos de excepciones para operaciones de entradas o salidas en el programa.

E luso principal de las excepciones en Java es para determinar los errores en tiempo de ejecucion, y esto ocurre cuando se produce algo no esperado en alguna de las instrucciones de Java programadas(por ejemplo un error de desbordamiento o un error de division entre cero).

Cuando se da una excepcion, se cambia por completo el puntero de ejecucion del codigo, y se va hacia uns ector de codigo predefinido para ello, por lo cual, cuando se termina de ejecutar ese codigo, se termina la ejecucion del programa.

Cuando en java se produce una excepcion al crear un objeto de alguna clase especifica, se mantendra la informacion sobre el error producido, y mostrara los metodos necesarios para obtener dicha informacion. Normalmente estas excepciones tienen como clase padre Throwable.

Hay excepciones sobre las cuales podemos tener control, que son programadas por el usuario. Estas se conocen como Try, Catch y Finally. Try se refiere a la seccion de codigo que se ejecutara cuando no exista ninguna excepcion, Catch es quien contiene la seccion de codigo necesaria cuando se ejecute una excepcion(en cualquier tipo), y finally, es quien contiene el trozo de codigo quer se ejecutara luego de finalizar cualquiera de las 2 anteriores.

```
try{
}
catch(TypeException ex){
}
finally{
}
```

Un ejemplo de excepcion definida por el programador es que no se trunque un programa cuando se divide entre cero. Simplemente se genera la excepcion en ese trozo de codigo.

Pregunta 2:

Defina monitor en Java y sus características, desde el punto de vista de la concurrencia. Explique los métodos que se deben definir y ponga un ejemplo significativo.

Normalmente en concurrencia, los monitores son estructuras de datos abstractas para hilos que se estan ejecutando en un tiempo definido. Su principal caracteristica es que sus hilos son ejecutados normalmente con exclusion mutua.

Por decir de otra manera, un monitor mantiene en un conjunto los servicios de los hilos por medio de metodos de acceso, al igual que los tipos de sus variables, locales o globales. Un monitor pretende ayudar a evitar los riesgos a los que se presenta un codigo con errores de programacion, proporcionando asi constructores de programacion mas adecuados, y con mayor abstraccion. Cabe mencionar que los monitores se utilizan en programacion orientada a objetos.

En java, para construir un monitor, se debe utilizar el modificador Synchronized, tienen una sola variable de condicion anonima, y todas las invocaciones a wait() y notify() se refieren automaticamente a esa unica variable anonima.

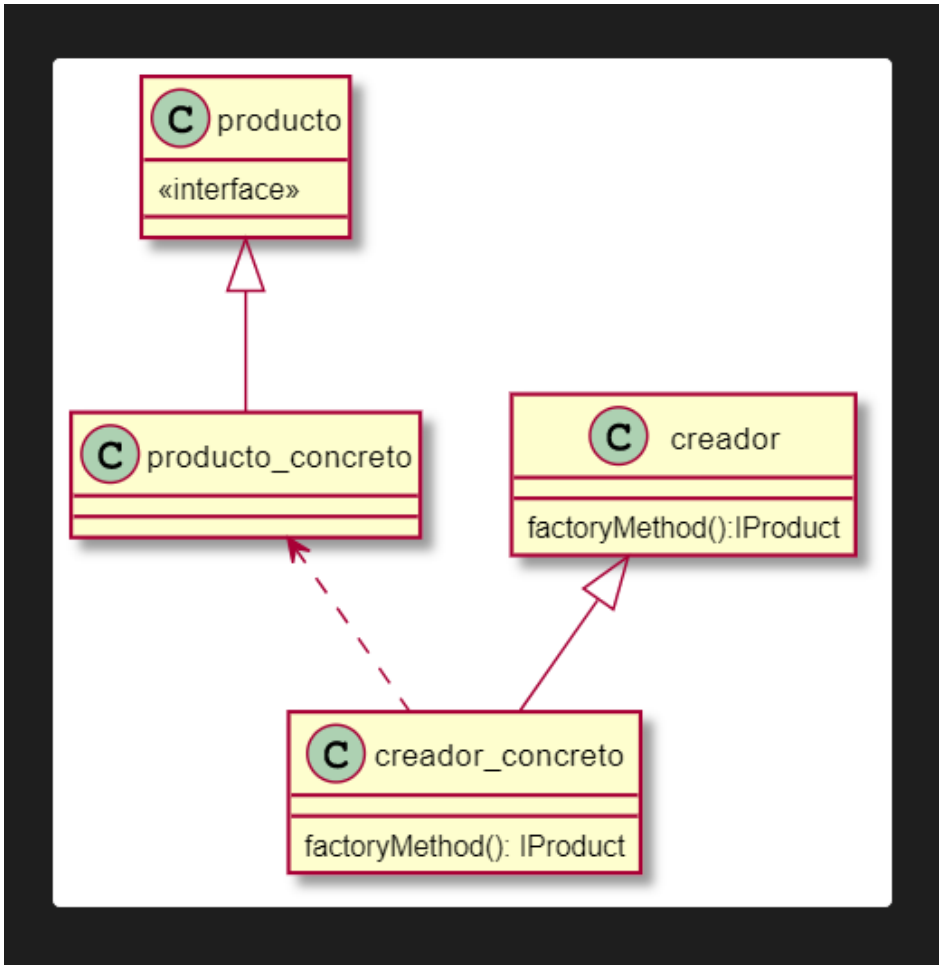
Pregunta 3:

Patron Factory:
Patron de disenio creacional. Proporciona una interfaz de codigo para crear objetos de una manera mas eficiente y ordenada en una superclase, mientras que permite que las subclases puedan alterar el tipo de objetos que crearan.

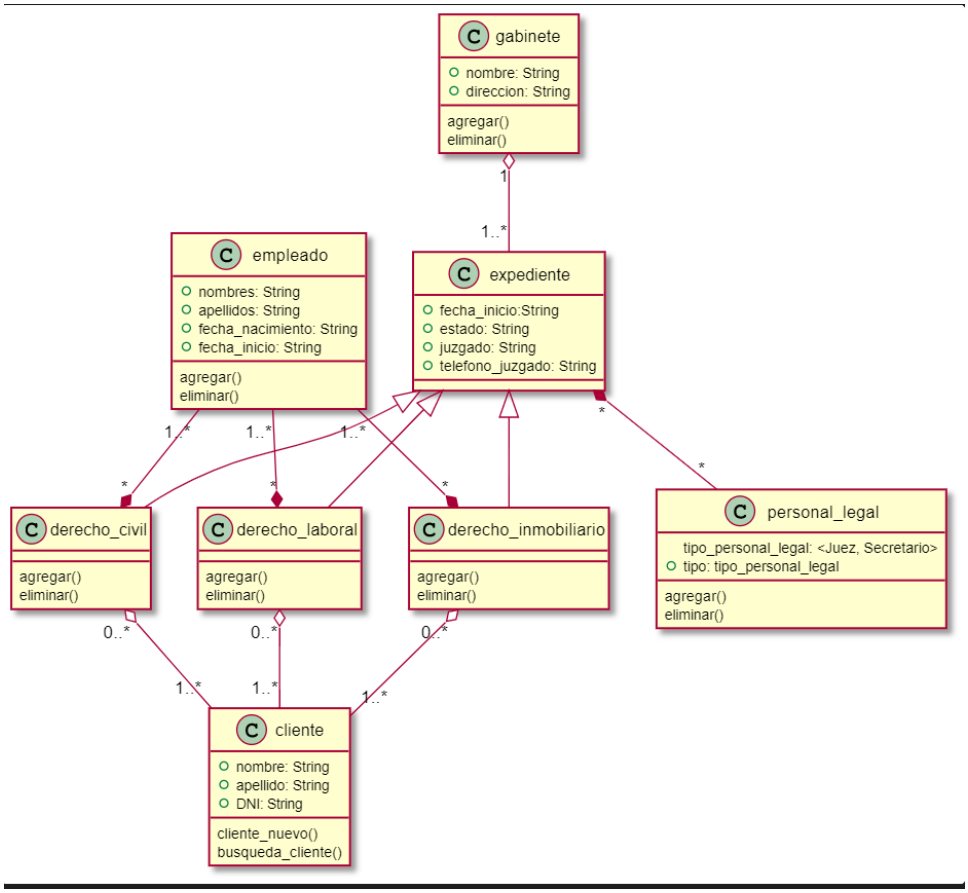
Este patron creracional sugiere que, en lugar de crear un objeto con new, se invoque a un metodo especifico del patron creacional, los cuales se conocen como productos.

Codigo en PlantUML

```
@startuml
    class producto{
        <<interface>>
    }
    class producto_concreto{

    }
    class creador{
        factoryMethod():IProduct
    }
    class creador_concreto{
        factoryMethod(): IProduct
    }
    creador <|-- creador_concreto
    producto_concreto <.. creador_concreto
    producto <|-- producto_concreto
@enduml
```


producto
«interface»
producto_concreto
creador
factoryMethod():IProduct
creador_concreto
factoryMethod(): IProduct

Problema 1:



Codigo en PlantUML:

```
@startuml
class cliente{
    + nombre: String
    + apellido: String
    + DNI: String
    cliente_nuevo()
    busqueda_cliente()
}

class empleado{
    + nombres: String
    + apellidos: String
    + fecha_nacimiento: String
    + fecha_inicio: String
    agregar()
    eliminar()
}

class gabinete{
    + nombre: String
    + direccion: String
    agregar()
    eliminar()
}

class expediente{
```

Código de examen: 162955

```
    + fecha_inicio:String
    + estado: String
    + juzgado: String
    + telefono_juzgado: String
}

class personal_legal{
    tipo_personal_legal: <Juez, Secretario>
    + tipo: tipo_personal_legal
    agregar()
    eliminar()
}

class derecho_civil{
    agregar()
    eliminar()
}

class derecho_laboral{
    agregar()
    eliminar()
}

class derecho_inmobiliario{
    agregar()
    eliminar()
}

gabinete "1" o-- "1..*" expediente
expediente <|-- derecho_inmobiliario
expediente <|-- derecho_laboral
expediente <|-- derecho_civil
expediente "*" *-- "*" personal_legal
derecho_civil "0..*" o-- "1..*" cliente
derecho_inmobiliario "0..*" o-- "1..*" cliente
derecho_laboral "0..*" o-- "1..*" cliente
empleado "1..*" --* "*" derecho_laboral
empleado "1..*" --* "*" derecho_civil
empleado "1..*" --* "*" derecho_inmobiliario
@enduml
```

Problema 2:

La solucion en Java consta de 3 clases, una clase Tablon, otra clase Socio, y otra clase ClubSocialVirtual.

Clase Tablon:

```java
package examen;

public class Tablon {
    private String nombre;
    public int libre = 0;
    private static int limite_escritura=2;
    public boolean escribiendo = false;

    public Tablon(String nombre) {
        this.nombre = nombre;
    }

    public synchronized boolean escribirTablero(int id_socio) throws InterruptedException{
        if(libre==limite_escritura)
            return false;

        System.out.println("== El socio "+id_socio+" va a escribir en el tablon ==");
        libre +=1;
        System.out.println("== Socios escribiendo actualmente ["+this.libre+"] ==");

        return true;
    }

    public synchronized void liberarTablero(int id_socio) throws InterruptedException{
        libre -=1;
        System.out.println("==== El socio "+ id_socio +" libera el tablon ====");
        System.out.println("== Socios escribiendo actualmente ["+ this.libre +"] ==");
        this.notify();
    }
}
```


package examen;

public class Tablon {
private String nombre;
public int libre = 0;
private static int limite_escritura=2;
public boolean escribiendo = false;

public Tablon(String nombre) {
this.nombre = nombre;
}

public synchronized boolean escribirTablero(int id_socio) throws InterruptedException{
if(libre==limite_escritura)
return false;

System.out.println("== El socio "+id_socio+" va a escribir en el tablon ==");
libre +=1;
System.out.println("== Socios escribiendo actualmente ["+this.libre+"] ==");

return true;
}

public synchronized void liberarTablero(int id_socio) throws InterruptedException{
libre -=1;
System.out.println("==== El socio "+ id_socio +" libera el tablon ====");
System.out.println("== Socios escribiendo actualmente ["+ this.libre +"] ==");
this.notify();
}
}

Clase Socio:

Código de examen: 162955

```java
package examen;

import java.util.Random;

public class Socio extends Thread {
    private Random random = new Random();
    private int id;
    private Tablon tablon;

    private boolean escribirTablon() {
        try {
            if(tablon.escribirTablero(id)) {
                sleep(random.nextInt(1000) + 100);
                return true;
            }
            return false;
        }
        catch(InterruptedException ex) {
            return false;
        }
    }

    public Socio(int id, Tablon tablon) {
        this.id = id;
        this.tablon = tablon;
    }

    public void run() {
        while(true) {
            try {
                while(!escribirTablon()) {
                    if(!tablon.escribiendo) {
                        System.out.println("El socio "+this.id+" lee el tablero");
                        Socio.sleep(random.nextInt(1000)+1000);
                    }
                }

                System.out.println("El socio "+this.id+" esta escribiendo");
                tablon.escribiendo = true;
                try {
                    sleep(random.nextInt(1000)+700);
                }
                catch(InterruptedException ex){
                    System.out.println("Error." + ex.toString());
                }

                tablon.escribiendo = false;
                tablon.liberarTablero(id);

                try {
                    sleep(random.nextInt(1000)+1200);
                }
                catch(InterruptedException ex) {
                    System.out.println("Error."+ ex.toString());
                }

            }
            catch(InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

package examen;

import java.util.Random;

public class Socio extends Thread {
private Random random = new Random();
private int id;

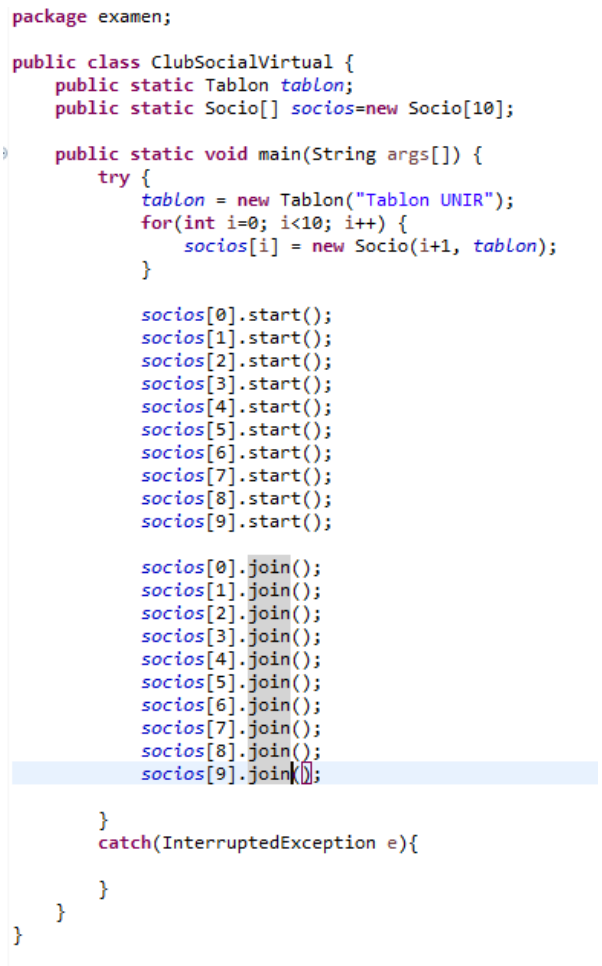
Código de examen: 162955

```
private Tablon tablon;

private boolean escribirTablon() {
try {
if(tablon.escribirTablero(id)) {
sleep(random.nextInt(1000) + 100);
return true;
}
return false;
}
catch(InterruptedException ex) {
return false;
}
}

public Socio(int id, Tablon tablon) {
this.id = id;
this.tablon = tablon;
}

public void run() {
while(true) {
try {
while(!escribirTablon()) {
if(!tablon.escribiendo) {
System.out.println("El socio "+this.id+" lee el tablero");
Socio.sleep(random.nextInt(1000)+1000);
}
}

System.out.println("El socio "+this.id+" esta escribiendo");
tablon.escribiendo = true;
try {
sleep(random.nextInt(1000)+700);
}
catch(InterruptedException ex){
System.out.println("Error." + ex.toString());
}

tablon.escribiendo = false;
tablon.liberarTablero(id);

try {
sleep(random.nextInt(1000)+1200);
}
catch(InterruptedException ex) {
System.out.println("Error."+ ex.toString());
}

}
catch(InterruptedException ex) {
ex.printStackTrace();
}
}
}
```

Código de examen: 162955

}

Clase ClubSocialVirtual:



```
package examen;

public class ClubSocialVirtual {
    public static Tablon tablon;
    public static Socio[] socios=new Socio[10];

    public static void main(String args[]) {
        try {
            tablon = new Tablon("Tablon UNIR");
            for(int i=0; i<10; i++) {
                socios[i] = new Socio(i+1, tablon);
            }

            socios[0].start();
            socios[1].start();
            socios[2].start();
            socios[3].start();
            socios[4].start();
            socios[5].start();
            socios[6].start();
            socios[7].start();
            socios[8].start();
            socios[9].start();

            socios[0].join();
            socios[1].join();
            socios[2].join();
            socios[3].join();
            socios[4].join();
            socios[5].join();
            socios[6].join();
            socios[7].join();
            socios[8].join();
            socios[9].join();

        }
        catch(InterruptedException e){

        }
    }
}
```

package examen;

public class ClubSocialVirtual {
public static Tablon tablon;
public static Socio[] socios=new Socio[10];

public static void main(String args[]) {
try {
tablon = new Tablon("Tablon UNIR");
for(int i=0; i<10; i++) {
socios[i] = new Socio(i+1, tablon);
}

socios[0].start();
socios[1].start();
socios[2].start();
socios[3].start();
socios[4].start();
socios[5].start();
socios[6].start();
socios[7].start();
socios[8].start();

Código de examen: 162955

```
socios[9].start();

socios[0].join();
socios[1].join();
socios[2].join();
socios[3].join();
socios[4].join();
socios[5].join();
socios[6].join();
socios[7].join();
socios[8].join();
socios[9].join();

}
catch(InterruptedException e){

}
}
}
```

Logs:

En los logs se puede observar que solo 2 personas pueden escribir al mismo tiempo, mientras que los demas leen hasta que esta libre la cola. El programa se ha intentado desarrollar con las bases vistas en clase.

```
== El socio 1 va a escribir en el tablon ==
== Socios escribiendo actualmente [1] ==
== El socio 3 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 2 lee el tablero
El socio 4 lee el tablero
El socio 5 lee el tablero
El socio 6 lee el tablero
El socio 7 lee el tablero
El socio 8 lee el tablero
El socio 9 lee el tablero
El socio 10 lee el tablero
El socio 3 esta escribiendo
El socio 1 esta escribiendo
==== El socio 1 libera el tablon ====
El socio 6 lee el tablero
== Socios escribiendo actualmente [1] ==
== El socio 5 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 7 lee el tablero
El socio 4 lee el tablero
El socio 9 lee el tablero
El socio 2 lee el tablero
El socio 8 lee el tablero
El socio 10 lee el tablero
El socio 5 esta escribiendo
==== El socio 3 libera el tablon ====
== Socios escribiendo actualmente [1] ==
```

== El socio 2 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 9 lee el tablero
El socio 10 lee el tablero
El socio 7 lee el tablero
El socio 8 lee el tablero
El socio 2 esta escribiendo
==== El socio 5 libera el tablon ====
== Socios escribiendo actualmente [1] ==
El socio 4 lee el tablero
== El socio 6 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 1 lee el tablero
El socio 3 lee el tablero
El socio 9 lee el tablero
El socio 6 esta escribiendo
El socio 7 lee el tablero
==== El socio 2 libera el tablon ====
== Socios escribiendo actualmente [1] ==
== El socio 8 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 10 lee el tablero
El socio 8 esta escribiendo
==== El socio 6 libera el tablon ====
== Socios escribiendo actualmente [1] ==
== El socio 3 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 5 lee el tablero
El socio 4 lee el tablero
El socio 9 lee el tablero
El socio 1 lee el tablero
El socio 7 lee el tablero
==== El socio 8 libera el tablon ====
== Socios escribiendo actualmente [1] ==
El socio 3 esta escribiendo
== El socio 2 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 2 esta escribiendo
El socio 4 lee el tablero
==== El socio 3 libera el tablon ====
El socio 10 lee el tablero
== Socios escribiendo actualmente [1] ==
== El socio 1 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 5 lee el tablero
El socio 9 lee el tablero
El socio 6 lee el tablero
El socio 7 lee el tablero
El socio 1 esta escribiendo
==== El socio 2 libera el tablon ====
== Socios escribiendo actualmente [1] ==
== El socio 8 va a escribir en el tablon ==
== Socios escribiendo actualmente [2] ==
El socio 8 esta escribiendo
