Universidad de Alcalá

Cookie Factory

Curso 2024/2025 - Convocatoria Ordinaria
Grado en Ingeniería en Sistemas de Información (GISI)
Asignatura: Paradigmas de programación
Práctica Final Laboratorio
Tutores: **Eva García López**
David de Fitero Domínguez
Antonio García Cabot
Sergio Caro Álvaro

David Sánchez Sánchez
02751903E
Universidad de Alcalá
Madrid, Spain
d.sanchezsanchez@edu.uah.es

Índice

1.Análisis aplicación

En el siguiente apartado se tratará de manera general los puntos más claves de la aplicación presentada a rasgos generales, sobre todo empleado a la finalidad de la aplicación y al desarrollo de las características principales de la interfaz

1.1 Concepto de la aplicación

La aplicación trata de una fábrica de galletas completamente funcional, en donde contaremos con **5 reposteros**, que irán metiendo de manera ordenada las galletas que generen en **3 hornos** para que posteriormente, **3 empaquetadores** las muevan al almacén donde podrán ser consumidas por el cliente. Toda la aplicación ha sido diseñada de la manera **más eficaz y eficiente** posible, encapsulando todo el código posible en métodos para su reutilización, y hecho de forma que la **escalabilidad** del código no suponga una tarea imposible. Contamos con una aplicación **completamente distribuida** y con una **programación concurrente simulada** en un mismo ordenador, pero que, para la escalabilidad a varios ordenadores generando una concurrencia real, seria completamente viable. Finalmente, para el funcionamiento de la aplicación y su interacción con el usuario, contamos con **2 interfaces** y un sistema de **logs** que nos proporcionara en la terminal de ejecución lo que va ocurriendo en el programa y de manera paralela lo ira escribiendo en un fichero ***.txt situado en la raíz del proyecto** para su posterior lectura y análisis.

He de destacar que todo el código ha sido elaborado para que todos los errores sean gestionados y la interrupción de este nunca sea cancelada o interrumpida por un error.

Hay que mencionar también que **todo el código** está **debidamente documentado** para que se pueda comprender independientemente de que no se haya programado, los **comentarios** son **simples** y permiten seguir la ejecución del código de una manera ordenada y simple.

En cuanto a puntos que destacar, principalmente nos hemos centrado en detallar y **embellecer las interfaces** de manera que la aplicación se vuelva más atractiva para el usuario, ya que, en sí, la funcionalidad, es la que es, aunque como ya se ha mencionado antes, la mayoría de **código esta encapsulado para dar pie a una posible escalabilidad del código.**

Tenemos un **logotipo** que nos permite diferenciar cual es la aplicación del **servidor →** 

Y otro **logotipo** que nos permite diferenciar con un pequeño detalle de un C mayúscula cual es el logotipo del **cliente →** 

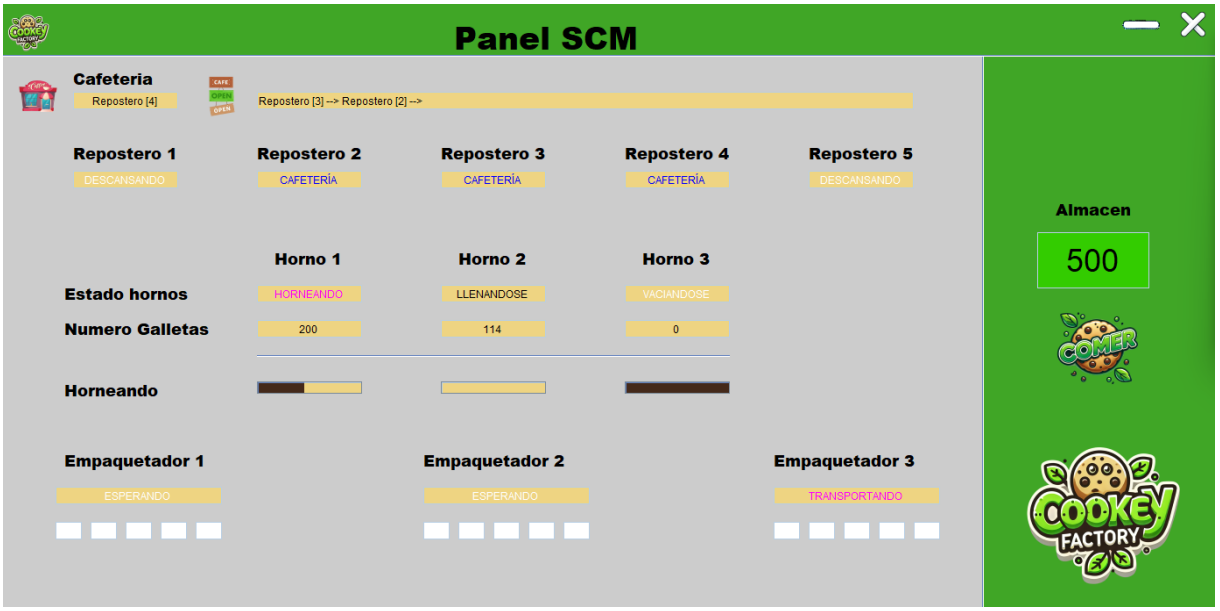
1.1.1 Interfaz Servidor (Panel SCM)

La **interfaz del servidor** ha sido pensada como un panel de para la **visualización de la cadena de suministro.**

Podemos apreciar detalles en ambas interfaces como son un **“header” personalizado** con un **botón de cerrado de la aplicación** y otro de **minimizado** con una **animación personalizada**. La ventana está programada de tal manera que **nos permita moverla por nuestra pantalla para alojarla donde mejor nos convenga.**

Podemos **visualizar constantemente todos los estados de todos los elementos de la fábrica**, y incluso el **procesado del horno** con **una barra de progreso**, o el **estado de empaquetado** con las **cajas** que va produciendo el empaquetador, siendo las rellenadas de color marrón aquellas que han sido completadas con las galletas indicadas.

Si nos fijamos en el panel de la derecha podemos ver una casilla en done vemos las galletas contenidas actualmente en el almacén, si **pulsamos** en el **logotipo personalizado de comer**, las **galletas se consumirán**, nunca bajando de 0.



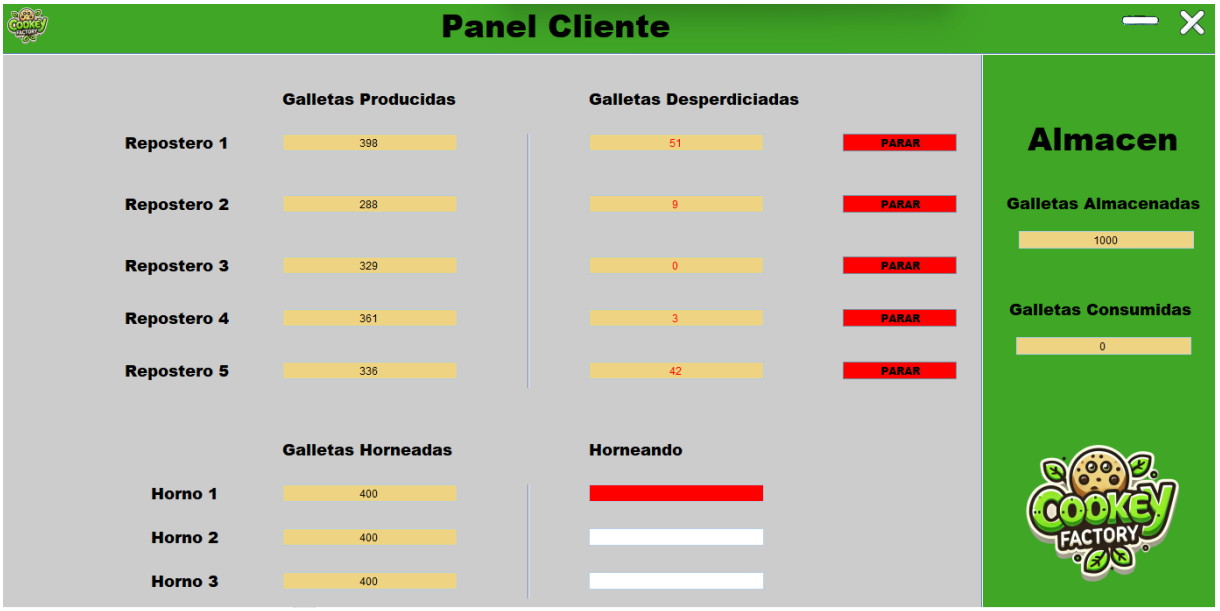
1.1.2 Interfaz Cliente

La **interfaz del cliente** ha sido pensada como panel de supervisión sobre los eventos que pasan en la fábrica, además cuenta con un **módulo de procesado de los reposteros**, que nos **permite bloquearlos remotamente**.

Podemos apreciar detalles en ambas interfaces como son un **"header" personalizado** con un **botón de cerrado de la aplicación** y otro de **minimizado** con una **animación personalizada**. La ventana está programada de tal manera que **nos permita moverla por nuestra pantalla para alojarla donde mejor nos convenga.**

Podemos apreciar que principalmente vemos datos específicos de cada sector, cabe destacar que desde el panel del cliente podemos ver si un **horno está horneando**, fijándonos **si la barra de uso está en color rojo.**

La interfaz y en sí, la parte del cliente ha sido programada y pensada de tal manera, que sí, **no está el servidor levantado**, **no se abra la interfaz**, y que **si estando el servidor levantado, se cae**, **la interfaz no se cierre**, sino que se **mantenga para poder guardar los últimos datos registrados**. Aunque pulsemos los botones de parar los reposteros, la interfaz se **mantendrá estable**, aunque los botones lógicamente no funcionen.



2.Diseño general del sistema y sincronización

En este apartado hablaremos de diseño de la aplicación, es decir de que forman se interconectan las distintas clases, y de qué manera se ha conseguido generar la conexión cliente-servidor.

2.1 Cadena de producción

En función de lo solicitado para el funcionamiento de la aplicación las clases han sido interconectadas de la siguiente manera. Los **reposteros empiezan la producción**, y **producen las galletas**, en el momento de depositado de las galletas, los hornos todavía no entran en juego, será **el propio repostero quien decida con los criterios seleccionados en que horno debe de introducir las galletas**.

Destacamos que cuando se introducen las galletas, el hilo de la cadena de producción lo toma el horno, y el **repostero sigue con su hilo de producción normal** en donde **acudirá a la cafetería**, una vez entra **en la cafetería**, será **esta la encargada de gestionar los reposteros** que tiene dentro.

Una vez introducidas las galletas, **llamaremos al horno elegido**, donde le **pasaremos el repostero que le esta introduciendo las galletas** para que use ciertos parámetros. Antes de este proceso de llamado, **el horno estará en una constante espera activa a que algún repostero consiga llenarlo entero**, es decir, que contenga la capacidad máxima de galletas.

En el momento en que el **horno** detecta que esta **lleno comienza el proceso de horneado**, en el que **solo interviene él.**

Posteriormente cuando el **horno termina el horneado**, el mismo **será el encargado de avisar a su empaquetado**r asignado para que empiece a vaciarlo.

Una vez el **empaquetador es llamado**, comienza a **vaciar el horno**, produciendo los paquetes, **llegado a los 5 paquetes**, procede a **transportarlos al almacén** y **asi de nuevo** una última vez para terminar de vaciar el horno. Quien tiene **el control del hilo durante el vaciado del horno es el empaquetador**, pero en el momento en el que el empaquetador **transporta las cajas al almacén y las depositas, es el mismo almacén quien gestiona el hilo**, ya que, si no hay espacio en el almacén, **lo bloqueara con una espera activa destinada a ver cuándo el propio usuario decide consumir las galletas que hay en el almacén**.

2.2 Conexión clases con interfaz

Las interfaces han sido conectadas con las clases de tal manera que cada un breve lapso, el propio hilo de cada una de las interfaces es el encargado de consultar todos los datos necesarios para poder mostrarlos por pantalla al usuario. De esta manera se consigue un **encapsulamiento de las interfaces** completamente **independiente** de la **ejecución del código**, **permitiendo** asi la **escalabilidad** de este y la **implementación** de **RMI** ya que las interfaces no son “serializable” en java,

2.3 Conexión cliente-servidor (RMI)

La conexión de cliente servidor se ha implementado a través del **protocolo RMI** en java, en el main del servidor, es decir, en la inicialización de este, primero **se inicializan las clases**, con **sus respectivos hilos**, y estos **son almacenados en listas**.

Posteriormente se **inicializa el servidor**, el **primer paso** para esto es **crear un objeto que contenga todos los hilos necesarios para transportar la información al cliente**, y además que este, **contenga la llamada a todos los métodos necesarios que invocaremos desde el cliente**, de esta manera, **en el servidor tendremos la implementación** de dichos métodos que **se ejecutaran remotamente** desde el cliente.

Una vez creado el objeto **crearemos un registro** en un **puerto** que será donde **escucharemos** las **peticiones del cliente**

Finalmente **subiremos** al **puerto** el **objeto** que queremos traspasar al cliente.

En la parte del cliente, a la hora de **conectarse al servidor**, simplemente le **marcaremos la dirección en donde tiene que buscar y el objeto que desea obtener**.

Con esto habremos conseguido la conexión entre servidor y cliente.

3.Clases principales

En este punto se describirán las clases principales del proyecto, enfocándonos en sus **atributos** y sus **métodos principales**.

No se han considerado elemento esencial de descripción incluir todos aquellos momentos en los que se escriben datos del log.

3.1 InicializacionServer.java

PRIMER ARCHIVO EN EJECUTARSE

Esta clase es la **main class** de nuestro servidor, será la encargada **de inicializar los hilos de las clases** de Cafetería, Almacén, Reposteros, Hornos y Empaquetadores. Posteriormente **inicializara nuestra interfaz** del servidor asi como su respectivo hilo y **completara formando la conexión RMI por parte del servidor**.

3.2 InicializacionCliente.java

SEGUNDO ARCHIVO EN EJECUTARSE (Y ULTIMO)

Esta clase es la encargada de **recoger primeramente el objeto que se ha traspasado por el método de RMI** y posteriormente **inicializar la interfaz del cliente.**

3.3 InterfazImplementacionRMI.java

Esta clase se **comparte tanto en servidor como en cliente**, es decir, deberá de estar en **ambos dispositivos para que la aplicación funcione con normalidad.**

Sera la **encarga de contener las llamadas a los métodos**, pero **NO la implementación** de estos (la implementación está contenida en otra clase del servidor)

3.4 Repostero

3.4.1 Atributos

En el atributo **listaHornos** como su propio nombre indica guardaremos la lista de los hornos en donde el repostero puede depositar su fabricación de galletas.

Los atributos **totalGalletas** y **totalGalletasDesperdiciadas** se usan para mostrar en la interfaz del cliente el rendimiento global de cada repostero

Los atributos **tandaGalletas** y **tandaGalletasDesperdiciadas se** utilizan para contabilizar las galletas producidas y desperdiciadas en cada deposito al horno.

El atributo booleano **pausaCafe** indica si el repostero está en la cafetería o en su puesto de trabajo (aunque este descansando).

El atributo **cafetería** se usa para asociar al repostero a una cafetería.

El atributo booleano **paradaManual** es usado para marcar si el repostero debe de parar de trabajar por un bloqueo externo procedente del cliente.

El atributo candando **lockHorno** es usado para generar una zona de sección critica en donde se realiza el deposito de galletas a un horno, de esta manera generamos que nunca haya dos reposteros metiendo galletas a la vez en un horno, lo que generaría una perdida invisible para el cliente de galletas horneadas.

El atributo **acción** que utilizaremos para ir marcando en la interfaz que acciones se están realizando.

El atributo **log** que usamos para generar los logs del sistema

3.4.2 Métodos principales

Con el método **run** de la clase, empezaremos generando el numero de tandas de producción que realizara el repostero, después, como se mencionara varias veces durante la ejecución del método se hará una comprobación de si se ha pedido que el repostero se bloquee por parte del cliente seguidamente se producirá una tanda de galletas y asi durante el numero de tandas que se haya establecido en esa iteración del método.

Una vez producidas todas las tandas volvemos a comprobar si hace falta hacer el bloqueo del repostero.

Luego realizamos la acción de depósito de las galletas en el horno que llamara al método depositar galletas que posterior mente llamara al método añadir galletas de la clase horno.

Una vez depositadas las galletas se comprueba si se ha desperdiciado alguna para añadirla al log.

Volvemos a comprobar si hay que bloquear el repostero, de no ser asi, entramos en la cafetería.

Ya una vez hemos salido de la cafetería, comprobamos de nuevo si hace falta bloquear el repostero y seguidamente realizamos el descanso de este.

En el método **depositarGalletas** primeramente generaremos el bloqueo del candado lock, luego entraremos en un bucle que no acabare hasta que el deposito de las galletas no haya terminado.

Seguidamente iremos iterando sobre los hornos hasta que encontremos uno en el que podamos depositar las galletas, encontrado el horno, añadiremos las galletas, de ese proceso se encargara el horno.

Una vez este el deposito terminado saldremos del método.

Cabe destacar que, si no encontramos un horno en el que depositar las galletas, entraremos en una espera activa hasta encontrar un horno disponible en el que depositar las galletas.

En método **comprobarParadaManual** es usado para verificar si es necesario bloquear el repostero, en caso de que no, simplemente volverá a salir del método, pero en caso de que si, indicara en el atributo acción que está bloqueado, añadirá al log cuando ha sido bloqueado y entrara en una espera activa hasta que desde el cliente se le indique que ya no está bloqueado.

Cabe destacar que solo cambiara la acción del repostero una vez y de la misma manera solo redactara una vez en el log que esta bloqueado.

3.5 Cafetería

3.5.1 Atributos

El atributo **colaCafeteria** lo usaremos para ir metiendo hay los reposteros que vayan llegando en orden y poder llevar un control de la cafetería y su cafetera.

Con el atributo candado **lockColaCafeteria** impediremos que la cola de la cafetería sea escrita de manera incorrecta, es decir, que siempre se mantenga el orden y de que la cola no sea sobre escrita por dos reposteros a la vez quedándose uno fuera.

Con el atributo semáforo, **semaforoClienteAtendido,** nos aseguramos siempre de que, entre solo una persona a usar la cafetera, no de que entren justo dos a la vez, por ejemplo. Este tiene marcado el “fair” a true, para que los clientes sean atendidos en orden de llegada.

Con el atributo **reposteroAtendido** marcamos en la interfaz el repostero que esta siendo atendido en el instante en el que se consulte el atributo. De la misma con el atributo **colaReposterosCafeteria** marcaremos en la interfaz la cola actual que hay para usar la cafetera.

3.5.2 Métodos principales

En el método **tomarCafe** primeramente usaremos un lock, para entrar en la cola de la cafetería, seguidamente, solicitaremos al semáforo un “ticket” para usar la cafetera, esperaremos hasta que se nos conceda el turno.

Una vez concedido el turno, nos prepararemos nuestro café, pero no si antes borrarnos de la cola de la cafetería. Una vez nos hemos preparado el café, nos desharemos de nuestro “ticket” es decir, haremos un reléase del semáforo para que el siguiente repostero pueda ser atendido.

3.6 Horno

3.6.1 Atributos

Tenemos el atributo de **capacidadMaxima** que usaremos para saber cuantas galletas caben en el horno, luego tendremos otro atributo nombrado **empaquetador,** que no termina de ser el empaquetador asociado al horno para la recogida de galletas.

Para contabilizar las galletas que hay en el horno usaremos el atributo **cantidadGalletas.**

Para llevar un histórico de las galletas horneadas en cada instante y poder actualizar los progres bar usaremos el atributo **totalGalletasHorneadas.**

Con el booleano **estaHorneando** podremos bloquear el horno para que los reposteros no introduzcan galletas, además, de la misma manera usaremos el booleano **estaEmpaquetando** para seguir bloqueando a los reposteros e indicar a los empaquetadores que pueden empezar su proceso de recogida de galletas.

El atributo **acción** que utilizaremos para ir marcando en la interfaz que acciones se están realizando.

El atributo **log** que usamos para generar los logs del sistema

3.6.2 Métodos principales

El método **añadirGalletas** lo usaremos para a parte de añadir galletas, en el caso de que se intenten meter más galletas de las que caben, devolver el numero de galletas que se van a desperdiciar.

Con el método **run,** método principal de la clase y el hilo en sí, generamos al ejecución constante del horno, que estará **constantemente revisando si es necesario que empiece a hornear galletas**, una vez tenga que empezar a hornear galletas, hará el log, y **hará su horneado** y es justo después cuando **generara una llamada a su empaquetador** asociado para que este empiece a empaquetar las galletas, **cuando este finalice**, indicara que el **horno puede volver a llenarse** y se pondrá a **comprobar de nuevo** si esta lleno para **comenzar el horneado**.

3.7 Empaquetador

3.7.1 Atributos

Contamos con la constante **cantidadRecogidaGalletas** que usamos para marcar la cantidad de galletas que se recogen cada vez que accedemos al horno y la constante **cantidadEmpaquetadoGalletas** que utilizamos para marcar en función de cuantas galletas recogidas deberíamos de ir al almacén.

En cuanto a atributos tenemos el número de **galletasRecogidas** junto con el número de **tandasGalletasRecogidas** que utilizaremos para poder contabilizar el proceso de vaciado del horno

El atributo **horno** ya que empaquetadores tienen un horno asociado a él, y el atributo **almacén** que al igual que el horno este asociado al empaquetador.

El atributo **acción** que utilizaremos para ir marcando en la interfaz que acciones se están realizando.

El atributo **log** que usamos para generar los logs del sistema

3.7.2 Métodos principales

Tenemos principalmente el método **vaciarHorno** el cual es el encargado de **bloquear el horno para que no se le añadan más galletas** hasta que no se termine de vaciar. Posteriormente al bloqueo **comienza el vaciado, luego el empaquetado y finalmente el transporte al almacén**, en donde hace la llamada al almacén para que se añadan las galletas.

3.8 Almacén

3.8.1 Atributos

En cuanto a contantes, tenemos la **capacidad máxima del almacén**, que se utilizara para comprobar que nunca se exceda dicha cantidad y la **cantidad de galletas que se consumen por cada vez que el cliente pulsa el botón de comer**

Luego tenemos un atributo para **contabilizar el número de galletas totales** que han entrado en el almacén, otro para **contabilizar el número de galletas consumidas** por el usuario y otro para **contabilizar el número de galletas contenidas en el momento** por el almacén, que ira variando en función se introduzcan o consuman galletas.

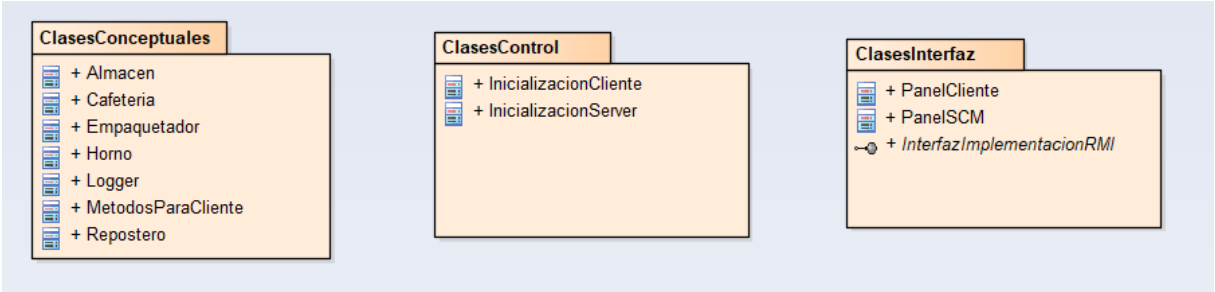
3.8.2 Métodos principales

Tenemos el método de **añadir galletas** que es el encargad de revisar si el almacén está lleno (de ser asi empezara una espera activa que bloquea el hilo principal de la cadena de producción) y mete las galletas al almacén.

Finalmente tenemos el método de **consumir galletas** que es el encargado de comprobar si cuando consumamos galletas el número final será menor de cero, para evitar un balance negativo en el almacén y después de no ser el caso, consumirá las galletas.

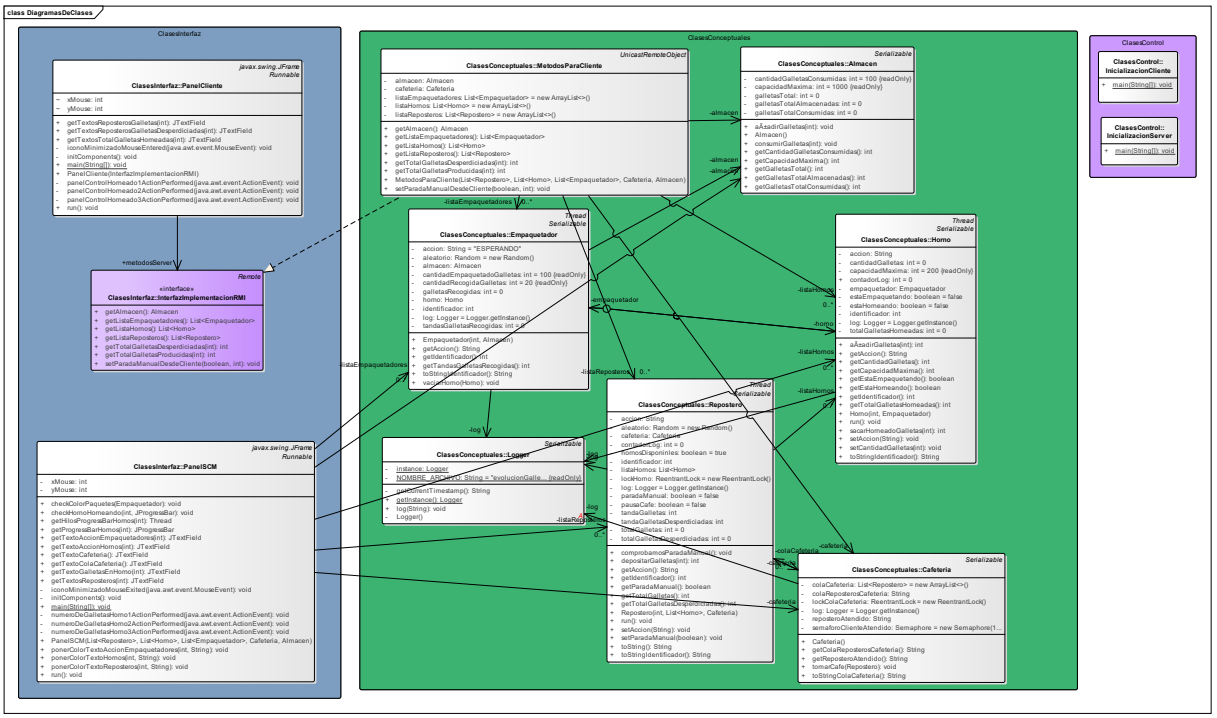
4.Diagrama de clases

A continuación, se muestra una lista con un separado en función del tipo de clases que contiene la aplicación (Conceptuales, Control, Interfaz).



A continuación, se muestra un diagrama de clases de la aplicación desarrollado en **Enterprise Architect** con sus respectivas **relaciones y cardinalidades**.

Para las interfaces se han **eliminado métodos y atributos NO esenciales** para el **entendimiento el diagrama**



(Haciendo zoom la imagen se puede ver con más calidad)

5.Anexos (Código fuente del programa)

Se mostrará de manera anexada (Los paquetes y dentro sus archivos correspondientes) cada uno de los códigos fuentes del programa en función de cada archivo *.java para que se pueda seguir de manera adecuada.

5.1 Server

5.1.1 InicializacionServer.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */


//Paquete al que pertenece la clase
package Server;

import ServerInterface.PanelSCM;
import java.util.List;
import java.util.ArrayList;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
//Importes
public class InicializacionServer
{
    public static void main(String[] args)
    {
        //Siempre agruparemos en listas los reposteros,horno y empaquetadores.
        List<Repostero> listaReposteros = new ArrayList<>();
        List<Horno> listaHornos = new ArrayList<>();
        List<Empaquetador> listaEmpaquetadores = new ArrayList<>();
        /*********************************
         *    INICIALIZACIÓN DE CLASES    *
         *********************************/
        //Creamos la cafeteria
        Cafeteria cafeteria = new Cafeteria();

        //Creamos el almacen
        Almacen almacen = new Almacen();

        //Creamos los empaquetadores
        for (int i = 1; i <= 3; i++){
```

```
            //Creamos el empaquetador
            Empaquetador empaquetador = new Empaquetador(i,almacen);
            //Lo añadimos a la lista para posteriormente referenciarlo en los hornos y al RMI
            listaEmpaquetadores.add(empaquetador);
        }
        //Creamos los hornos
        for (int i = 1; i <= 3; i++)
        {
            //Creamo el horno
            Horno horno = new Horno(i,listaEmpaquetadores.get(i-1));
            //Inicializamos el hilo
            horno.start();
            //Lo añadimos a la lista para posteriormente referenciarlo en los reposteros y al RMI
            listaHornos.add(horno);
        }
        //Creamos los reposteros
        for (int i = 1; i <= 5; i++)
        {
            //Creamos al repostero
            Repostero repostero = new Repostero(i, listaHornos,cafeteria);
            //Inicializamos el hilo
            repostero.start();
            //Lo añadimos a la lista para referenciarlo al RMI
            listaReposteros.add(repostero);
        }

         /**********************************
          *   INICIALIZACIÓN DE INTERFAZ   *
          **********************************/
        //Inicializamos la interfaz del servidor
        PanelSCM interfazServer = new PanelSCM(listaReposteros, listaHornos, listaEmpaquetadores,
cafeteria,almacen);
        interfazServer.setVisible(true);
        Thread hiloInterfazServidor = new Thread(interfazServer);
        hiloInterfazServidor.start();
        /**********************************
         *   INICIALIZACIÓN DE SERVIDOR   *
         **********************************/
        try
        {
            MetodosParaCliente objetoTraspasable = new MetodosParaCliente(listaReposteros, listaHornos,
listaEmpaquetadores, cafeteria,almacen);
            LocateRegistry.createRegistry(1099);
            //Le pasamos los objetos al "Server"
            Naming.rebind("//localhost/objetoTraspasable", objetoTraspasable);
        }
        catch (RemoteException error)
```

```
        {
            System.out.println("Se ha producido un error mientras se levantaba el servidor en el Server
--> " + error);
        }
        catch (MalformedURLException error)
        {
            System.out.println("Se ha producido un error mientras se subia delsde el Servidor el Objeto
Traspasable --> " + error);
        }
    }
}
```

5.1.2 Repostero.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
//Importes
import Logger.Logger;
import java.io.Serializable;
import java.util.List;
import java.util.Random;
import java.util.concurrent.locks.ReentrantLock;
import Logger.Logger;
//CLASE
public class Repostero extends Thread implements Serializable
{
    //Atributos de la clase
    private int identificador;
    //En esta ArrayList almacenaremos los horno de manera ordenada
    private List<Horno> listaHornos;
    //Este stributo se utilizara para mandar la accion que esta realizando el repostero a la interfaz a
la interfaz
    private String accion;
        //Atributos de producción
    private int totalGalletas = 0;                  //Cuando se crea el repostero aun no ha creado
ninguna galleta
    private int totalGalletasDesperdiciadas = 0;    //Cuando se crea el repostero aun no ha
desperdiciado ninguna galleta
    private int tandaGalletas;
    private int tandaGalletasDesperdiciadas;
    private boolean pausaCafe = false;              //Por defecto al crearse el repostero primero
debera de hacer una tanda de galletas antes de descansar
        //Para la cafeteria
    private Cafeteria cafeteria;
        //Usamos estos atributos para generar la pausa del hilo
    private boolean paradaManual = false;       //Por defecto la parada manual se pondra a true despues
de crear el hilo
    private ReentrantLock lockHorno = new ReentrantLock();

        //Usaremos este objeto para la aletoridad
    private Random aleatorio = new Random();
        //Usaremos este objeto para los logs del sistema
    private Logger log = Logger.getInstance();
    private int contadorLog = 0;
    //Contructor
```

```java
public Repostero(int _identificador, List<Horno> _listaHornos,Cafeteria _cafeteria)
{
    this.identificador = _identificador;
    this.listaHornos = _listaHornos;
    this.cafeteria = _cafeteria;
}
//Contructor vacio [NO HAREMOS USO DE ESTE]
//Contructores alternativos o sobrecargados [NO HAREMOS USO DE ESTOS}
//Getters
public int getTotalGalletas(){return totalGalletas;}
public int getTotalGalletasDesperdiciadas(){return totalGalletasDesperdiciadas;}
public int getIdentificador(){return identificador;}
public boolean getParadaManual(){return paradaManual;}
public String getAccion(){return accion;}
//Setters
public void setParadaManual(boolean _paradaManual){this.paradaManual = _paradaManual;}
public void setAccion(String _accion){this.accion = _accion;}
//To String (Solo si es necesario)
public String toStringIdentificador()
{
    return String.valueOf(identificador);
}
@Override
public String toString()
{
    return "Repostero ["+identificador+"]";
}
//Metodos de la clase
@Override
public void run()
{
    //Los reposteros nunca dejaran de trabajar
    while (true)
    {
        try{
            //Realizaran esta secuencia entre 3 y 5 veces
            for (int i=0; i <= (3 + aleatorio.nextInt(3)); i++)
            {
                //Comprobamos si hay paradaManual
                comprobamosParadaManual();

                //Producción de la tanda de galletas
                accion = "PRODUCIENDO";
                log.log("Repostero["+identificador+"] -->"+accion);
                Thread.sleep(2000 + aleatorio.nextInt(2001));
                tandaGalletas = 37 + aleatorio.nextInt(9);
                totalGalletas += tandaGalletas;
```

```
                    log.log("Repostero["+identificador+"] =========> Ha producido "+tandaGalletas+"
galletas");

                    //Comprobamos si hay parada manual
                    comprobamosParadaManual();

                    //Depositamos las galletas en cualquier horno disponible
                    accion = "DEPOSITANDO";
                    log.log("Repostero["+identificador+"] -->"+accion);
                    tandaGalletasDesperdiciadas = depositarGalletas(tandaGalletas);

                    //Si se han desperdiciado galletas se registrara en el log
                    if(tandaGalletasDesperdiciadas > 0)
                    {
                        log.log("Repostero["+identificador+"] =========> Ha desperdiciado
"+tandaGalletasDesperdiciadas+" galletas");
                    }
                    totalGalletasDesperdiciadas += tandaGalletasDesperdiciadas;
                }

                //Comprobamos si hay parada manual
                comprobamosParadaManual();

                //Parada para el cafe
                cafeteria.tomarCafe(this);              //Le pasamos el propio hilo a la cafeteria
                //Comprobamos si hay parada manual
                comprobamosParadaManual();

                //Descanso hasta que vuelvan a generar más tandas de galletas
                accion = "DESCANSANDO";
                log.log("Repostero["+identificador+"] -->"+accion);
                Thread.sleep(3000 + aleatorio.nextInt(3001));
            }
            catch (InterruptedException error)
            {
                System.out.println("Durante la ejecucion del Repostero["+identificador+"] --> " +
error);
            }
        }
    }


    //El metodo implica synchronized para evitar que dos reposteros metan a la vez galletas en el mismo
horno
    public int depositarGalletas(int _tandaGalletas)
    {
        //Aqui depositaremos las galletas que se desperdicien
```

```
        int _tandaGalletasDesperdiciadas = 0;   //Por defecto ninguna

        //Para indicar que ya hemos depositado las galletas en el horno
        boolean depositoTerminado = false;

        try
        {
            lockHorno.lock();
            //Antes de elegir el horno en al que vamos a depositar las galletas
            while (!depositoTerminado)
            {
                for(int indexHornos = 0; indexHornos < listaHornos.size(); indexHornos++)
                {
                    if(listaHornos.get(indexHornos).getCantidadGalletas() <
listaHornos.get(indexHornos).getCapacidadMaxima() && !listaHornos.get(indexHornos).getEstaHorneando()
&& !listaHornos.get(indexHornos).getEstaEmpaquetando())
                    {
                        _tandaGalletasDesperdiciadas =
listaHornos.get(indexHornos).añadirGalletas(_tandaGalletas);
                        //Registramos cuantas galletas se han depositado en el horno
                        int galletasDepositadasHorno = _tandaGalletas - _tandaGalletasDesperdiciadas;
                        log.log("Repostero["+identificador+"] =========> Ha depositado
"+galletasDepositadasHorno+" galletas en el
Horno["+listaHornos.get(indexHornos).getIdentificador()+"]");
                        depositoTerminado = true;
                        break;  //Break del bucle for
                    }
                    else
                    {
                        try
                        {
                            Thread.sleep(100);
                        }
                        catch(InterruptedException error)
                        {
                            System.out.println("Durante una espera de seguridad para liberar memoria a
la hora de encontrar un horno que no este lleno se a interrumpido la ejecucion de codigo generando el
siguiente error --> " + error);
                        }
                    }
                }
            }
        }
        catch (Error error)
        {
            System.out.println("Se ha producido un error en el repostero ["+identificador+"] mientras e
le asignaba un horno al que transportar las galletas producidas --> ) + error");
```

```
        }
        finally
        {
            //Siempre desbloqueamos el lock de los hornos
            lockHorno.unlock();
        }

        return _tandaGalletasDesperdiciadas;
    }
        public void comprobamosParadaManual()
    {
        try
        {
            while(paradaManual)
            {
                //Espera activa


                //Solo mostramos una vez en el log que esta bloqueado
                if(contadorLog == 0)
                {
                     accion = "BLOQUEADO";
                    log.log("Repostero["+identificador+"] -->"+accion);
                    contadorLog++;
                }
                Thread.sleep(1000);
            }
            //Si pasa por aqui significa que puede escribir el log de que esta bloqueado, entonces
            contadorLog = 0;
        }
        catch(InterruptedException error)
        {
            System.out.println("Se ha producido un error mientras se hacia la espera activa para
reanudar el repostero["+identificador+"] --> " + error);
        }
    }

}
```

5.1.3 Horno.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
```

```java
//Importes
import Logger.Logger;
import java.io.Serializable;
//CLASE
public class Horno extends Thread implements Serializable
{
    //Contantes de la clase (Bloqueamos la variable con la sentencia final)
    private final int capacidadMaxima = 200;
    //Atributos de la clase
    private Empaquetador empaquetador;          //Su empaquetador asociado
    private int identificador;
    private int cantidadGalletas = 0;           //Por defecto cuando se cree el horno siempre estara
vacio
    private int totalGalletasHorneadas = 0;     //Por defecto cuando se cree el horno no llevara
ninguna galleta horneada
    private boolean estaHorneando = false;      //Por defecto cuando se cree el horno no estara en
proceso de horneado hasta que no se llene
    private boolean estaEmpaquetando = false;
    private String accion;

    //Para los logs
    private Logger log = Logger.getInstance();
    //Contructor
    public Horno(int _identificador, Empaquetador _empaquetador)
    {
        this.identificador = _identificador;
        this.empaquetador = _empaquetador;
    }
    //Contructor vacio
    //Contructores alternativos o sobrecargados
    //Getters
    public int getIdentificador(){return identificador;}
    public int getCantidadGalletas(){return cantidadGalletas;}
    public int getCapacidadMaxima(){return capacidadMaxima;}
    public int getTotalGalletasHorneadas(){return totalGalletasHorneadas;}
    public boolean getEstaHorneando(){return estaHorneando;}
    public boolean getEstaEmpaquetando(){return estaEmpaquetando;}
    public String getAccion(){return accion;}
    public int contadorLog = 0;


    //Setters
    public void setCantidadGalletas(int _cantidadGalletas){this.cantidadGalletas = _cantidadGalletas;}
    public void setAccion(String _accion){this.accion = _accion;}


    //To String (Solo si es necesario)
```

```
public String toStringIdentificador()
{
    return "Horno ["+identificador+"]";
}

//Metodos de la clase
public int añadirGalletas(int galletas)
{
    //Una vez que entran aqui significa que hay huecos disponibles

    int galletasDesperdiciadas = 0;

    if((cantidadGalletas+galletas) <= capacidadMaxima)
    {
        cantidadGalletas += galletas;
    }
    else
    {
        galletasDesperdiciadas = (cantidadGalletas+galletas)-capacidadMaxima;
        cantidadGalletas = capacidadMaxima;
    }
    return galletasDesperdiciadas;
}

@Override
public void run()
{
    try
    {
        //Un horno nunca va a parar de trabajar
        while(true)
        {
            //Solo lo escribimos una vez en el log cada vez que se empieza a llenar
            if(contadorLog == 0)
            {
                //Si esta aqui significa que esta en estado de llenado
                accion = "LLENANDOSE";
                log.log("Horno["+identificador+"] -->"+accion);
                contadorLog++;
            }

            if(cantidadGalletas == capacidadMaxima && !estaHorneando)
            {
                //Comienza el horneado
                accion = "HORNEANDO";
                log.log("Horno["+identificador+"] -->"+accion);
                estaHorneando = true;
```

```
                    Thread.sleep(8000);             //Tardan en hornear la tanda 8000 milisegundos (8
segundos)
                    //Anotamos que las galletas ya estan horneadas
                    totalGalletasHorneadas += capacidadMaxima;
                    //Termina el horneado
                    estaHorneando = false;
                    //Empieza empaquetado, es decir, se empieza a vaciar
                    accion = "VACIANDOSE";
                    log.log("Horno["+identificador+"] -->"+accion);
                    estaEmpaquetando = true;
                    //Avisamos al empaquetador para que vacie el horno
                    empaquetador.vaciarHorno(this);
                    //Ahora esperamos a que se vacie el horno
                    //Termina empaquetado
                    estaEmpaquetando = false;
                    //Cuando termina todo el proceso significa que ya esta vacio por lo que cambiamos
el contador del log
                    contadorLog = 0;
                }
                else
                {
                    //Pequeña espera hasta que vuelva a comprobar si ya esta lleno
                    Thread.sleep(500);
                }
            }
        }
        catch(InterruptedException error)
        {
            System.out.println("Se ha producido un error mientras se ejecutaba el
Horno["+identificador+"] --> " + error);
        }
    }
    public int sacarHorneadoGalletas(int numeroGalletasSacar)
    {
        cantidadGalletas -= numeroGalletasSacar;
        return numeroGalletasSacar;
    }

}
```

5.1.4 Empaquetador.java

```java
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
//Importes
import java.io.Serializable;
import java.time.Duration;
import java.util.Random;
import Logger.Logger;
//CLASE
public class Empaquetador extends Thread implements Serializable
{
    //Constantes de la clase
    private final int cantidadRecogidaGalletas = 20;
    private final int cantidadEmpaquetadoGalletas = 100;
    //Atributos de la clase
    private int identificador;
    private int galletasRecogidas = 0;
    private int tandasGalletasRecogidas = 0;
    private Horno horno;                        //Los empaquetadores tienen un horno asociado a el
    private Almacen almacen;
    private String accion = "ESPERANDO";        //Por defecto cuando se cree el objeto estara esperando
        //Con esta semilla generaremos la aleatoriedad
    private Random aleatorio = new Random();
        //Para los logs
    private Logger log = Logger.getInstance();
    //Contructor
    public Empaquetador(int _identificador, Almacen _almacen)
    {
        this.identificador = _identificador;
        this.almacen = _almacen;
    }
    //Getters
    public int getIdentificador(){return identificador;}
    public String getAccion(){return accion;}
    public int getTandasGalletasRecogidas(){return tandasGalletasRecogidas;}
    //To String (Solo si es necesario)
    public String toStringIdentificador()
    {
        return "Empaquetador ["+identificador+"]";
    }
    //Metodos de la clase
    public void vaciarHorno(Horno horno)
```

```
    {
        boolean hornoVaciado = false;
        while(!hornoVaciado)
        {
            try
            {
                //Antes de empezar a empaquetar no habra recogido ninguna tanda aun
                tandasGalletasRecogidas = 0;
                accion = "EMPAQUETANDO";
                log.log("Empaquetador["+identificador+"] -->"+accion);
                //Realizamos las tandas de 5 en 5 para que las empaquetemos siempre de 100 en 100
                for(tandasGalletasRecogidas = 1; tandasGalletasRecogidas < 6;
tandasGalletasRecogidas++)
                {
                    //Tanda de recogida galletas
                    galletasRecogidas += horno.sacarHorneadoGalletas(cantidadRecogidaGalletas);
                    log.log("Empaquetador["+identificador+"] =========> Ha sacado
"+cantidadRecogidaGalletas+" galletas del Horno["+horno.getIdentificador()+"]");
                    Thread.sleep(500 + aleatorio.nextInt(1000));
                }
                //Empaquetamos las galletas para llevarlas al almacen
                accion = "TRANSPORTANDO";
                log.log("Empaquetador["+identificador+"] -->"+accion);
                //Cuando termina de transportar, las tandas recogidas vuelve a ser 0
                tandasGalletasRecogidas = 0;
                Thread.sleep(2000 + aleatorio.nextInt(4000));
                almacen.añadirGalletas(cantidadEmpaquetadoGalletas);
                log.log("Empaquetador["+identificador+"] =========> Ha transportado
"+cantidadEmpaquetadoGalletas+" galletas  al almacen");
                //Comprobamos si ha terminado de vaciar el horno
                if(galletasRecogidas == horno.getCapacidadMaxima()){ = true;}
            }
            catch(InterruptedException error)
            {
                System.out.println("Se ha generado un error mientras el empaquetador["+identificador+"]
estaba vaciando el Horno["+horno.getIdentificador()+"] --> " + error);
            }
        }
        //Como ya ha terminado su vaciado de horno, restauramos las galletas recogidas a 0
        galletasRecogidas = 0;
        accion = "ESPERANDO";
        log.log("Empaquetador["+identificador+"] -->"+accion);
    }

}
```

5.1.5 Almacen.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
//Importes
import java.io.Serializable;
import java.util.concurrent.locks.ReentrantLock;
public class Almacen implements Serializable
{
    //Constantes de la clase
    private final int capacidadMaxima = 1000;
    private final int cantidadGalletasConsumidas = 100;
    //Atributos de la clase
    private int galletasTotal = 0;
    private int galletasTotalConsumidas = 0;
    private int galletasTotalAlmacenadas = 0;
    //Contructor vacio
    public Almacen(){};
    //Contructores alternativos o sobrecargados
    //Getters
    public int getCapacidadMaxima(){return capacidadMaxima;}
    public int getGalletasTotal(){return galletasTotal;}
    public int getCantidadGalletasConsumidas() {return cantidadGalletasConsumidas;}
    public int getGalletasTotalConsumidas(){return galletasTotalConsumidas;}
    public int getGalletasTotalAlmacenadas(){return galletasTotalAlmacenadas;}
    //Metodos de la clase
    public synchronized void añadirGalletas(int galletas)
    {
        //Revisamos si el almacen esta lleno
        while(galletasTotal == capacidadMaxima)
        {
            //Espera activa
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException error)
            {
                System.out.println("Se ha producido un error mientras se realizaba una espera activa
para poder añadir galletas al almacen --> " + error);
            }
        }
```

```
        galletasTotal += galletas;
        //Log de almacenamiento total
        galletasTotalAlmacenadas += galletas;
    }

    public void consumirGalletas(int cantidadGalletasConsumir)
    {
        int galletasRestantes = galletasTotal - cantidadGalletasConsumir;
        //Comprobamos que si es inferior a 0, se establezca en 0
        if (galletasRestantes < 0)
        {
            galletasTotal = 0;
        }
        else
        {
            galletasTotal = galletasRestantes;
            galletasTotalConsumidas += cantidadGalletasConsumir;
        }
    }

}
```

5.1.6 Cafeteria.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
//Importes
import java.util.List;
import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.Semaphore;
import java.io.Serializable;
import Logger.Logger;
//CLASE
public class Cafeteria implements Serializable
{
    //Atributos
    private List<Repostero> colaCafeteria = new ArrayList<>();      //La cola en la creacion del objeto estara vacia por defecto
    private ReentrantLock lockColaCafeteria = new ReentrantLock();
    private Semaphore semaforoClienteAtendido = new Semaphore(1, true);     //Generamos que se atienda solo y solo a un Repostero y que se les atienda en funcion del orden de llegada
    private String reposteroAtendido;
    private String colaReposterosCafeteria;
        //Para lo logs
    private Logger log = Logger.getInstance();
    //Contructor vacio
    public Cafeteria(){}
    //Contructores alternativos o sobrecargados
    //Getters
    public String getReposteroAtendido(){return reposteroAtendido;}
    public String getColaReposterosCafeteria(){return colaReposterosCafeteria;}
    //ToString
    public String toStringColaCafeteria()
    {
        String cadena = "";
        for (int indice = 0; indice < colaCafeteria.size(); indice++)
        {
            cadena += colaCafeteria.get(indice).toString() + " --> ";
        }

        return cadena;

    }
```

```
    //Metodos de la clase
    public void tomarCafe(Repostero repostero)
    {
        //Sacamos el identificador del repostero para usarlo a la hora de cambiar el texto en la
interfaz
        int identificadorRepostero = repostero.getIdentificador();
        //Marcamos como que el repostero acaba de entrar en la cafeteria
        repostero.setAccion("CAFETERÍA");
        log.log("Repostero["+repostero.getIdentificador()+"] -->"+repostero.getAccion());
        //Primero nos aseguramos de que la cola sea escrita correctamente
        try
        {
            lockColaCafeteria.lock();
            colaCafeteria.add(repostero);
            colaReposterosCafeteria = toStringColaCafeteria();
        }
        catch(Error error)
        {
            System.out.println("Se ha producido un error mientras el
repostero["+repostero.toStringIdentificador()+"] estaba añadiendose a la cola de la cafeteria --> " +
error);
        }
        finally
        {
            //Nos aseguramos de liberar el cerrojo siempre
            lockColaCafeteria.unlock();
        }
        //Una vez nos hemos añadido a la cola de la cafeteria adquirimos nuestro Ticket
        //Pidiendo aqui el Ticket nos aseguramos de que no se cuele nadie y el orden se mantenga segun
lo imprimimos en el JTextField de la cola de la cafeteria
        try
        {
            semaforoClienteAtendido.acquire();
            //Cuando entro aqui significa que estoy siendo atendido, entonces me quito de la cola y
actualizo el texto
            try
            {
                lockColaCafeteria.lock();
                colaCafeteria.remove(repostero);
                colaReposterosCafeteria = toStringColaCafeteria();
            }
            catch(Error error)
            {
                System.out.println("Se ha producido un error mientras el
repostero["+repostero.toStringIdentificador()+"] estaba quitandose de la cola de la cafeteria --> " +
error);
```

```
            }
            finally
            {
                lockColaCafeteria.unlock();
            }
            //Estando aqui ya se que puedo marcarme como que estoy siendo atendido
            reposteroAtendido = repostero.toString();
            //Me preparo el cafe
            Thread.sleep(2000);
        }
        catch(InterruptedException error)
        {
            System.out.println("Se ha producido un error mientras el
repostero["+repostero.toStringIdentificador()+"] estaba cogiendo un ticket para la cafeteria y
preparandose el cafe --> " + error);
        }
        finally
        {
            //Si ocurriese un error muy inesperado nos aseguramos de unlockear el candado
            if(lockColaCafeteria.isLocked()){lockColaCafeteria.unlock();}

            //Una vez tengo mi cafe preparado salgo de la cafeteria
            reposteroAtendido = "";
            //Nos aseguramos de confirmar que hemos sido atentidos siempre
            semaforoClienteAtendido.release();
        }
    }
}
```

5.1.7 MetodosParaCliente.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Server;
//Importes
import SharedZone.InterfazImplementacionRMI;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;
//CLASE
public class MetodosParaCliente extends UnicastRemoteObject implements InterfazImplementacionRMI
{
    private List<Repostero> listaReposteros = new ArrayList<>();
    private List<Horno> listaHornos = new ArrayList<>();
    private List<Empaquetador> listaEmpaquetadores = new ArrayList<>();
    private Cafeteria cafeteria;
    private Almacen almacen;
    //Contructor
    public MetodosParaCliente (List<Repostero> _listaReposteros, List<Horno> _listaHornos,
List<Empaquetador> _listaEmpaquetadores, Cafeteria _cafeteria, Almacen _almacen) throws RemoteException
    {
        this.listaReposteros = _listaReposteros;
        this.listaHornos = _listaHornos;
        this.listaEmpaquetadores = _listaEmpaquetadores;
        this.cafeteria = _cafeteria;
        this.almacen = _almacen;
    }
    //GETTERS (ESENCIALES)
    public int getTotalGalletasProducidas(int indiceRepostero){return
listaReposteros.get(indiceRepostero).getTotalGalletas();}
    public int getTotalGalletasDesperdiciadas(int indiceRepostero){return
listaReposteros.get(indiceRepostero).getTotalGalletasDesperdiciadas();}
    public List<Repostero> getListaReposteros(){return listaReposteros;}
    public List<Horno> getListaHornos(){return listaHornos;}
    public List<Empaquetador> getListaEmpaquetadores(){return listaEmpaquetadores;}
    public Almacen getAlmacen(){return almacen;}
    public void setParadaManualDesdeCliente(boolean _paradaManual, int indexRepostero)
    {
        getListaReposteros().get(indexRepostero).setParadaManual(_paradaManual);
    }
}
```

5.2 ServerInterface

5.2.1 PanelSCM.java (Se ha eliminado el código generado automáticamente por NetBeans)

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */

//Paquete al que pertenece la clase
package ServerInterface;

//Importes
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import Server.Repostero;
import Server.Empaquetador;
import Server.Cafeteria;
import Server.Horno;
import Server.Almacen;

//CLASE
public class PanelSCM extends javax.swing.JFrame implements Runnable{

    //Atributos

    //Guardamos las posiciones del mouse en la pantalla
    private int xMouse;
    private int yMouse;
    private List<Repostero> listaReposteros;
    private List<Horno> listaHornos;
    private List<Empaquetador> listaEmpaquetadores;
    private Cafeteria cafeteria;
    private Almacen almacen;

    //Hilos para las barras de progres
    Thread hiloBarraProgresoHorno1 = null;
    Thread hiloBarraProgresoHorno2 = null;
    Thread hiloBarraProgresoHorno3 = null;


    public PanelSCM(List<Repostero> _listaReposteros, List<Horno> _listaHorno, List<Empaquetador> _listaEmpaquetadores,
Cafeteria _cafeteria, Almacen _almacen) {
        initComponents();
        Image icono = Toolkit.getDefaultToolkit().getImage(PanelSCM.class.getResource("/Fototeca/iconoAplicacion.png"));
        setIconImage(icono);

        //Centramos la ventana en la pantalla del usuario que ejecute la aplicacion
        setLocationRelativeTo(null);


        //Asignamos los atributos
        this.listaReposteros = _listaReposteros;
        this.listaHornos = _listaHorno;
        this.listaEmpaquetadores = _listaEmpaquetadores;
        this.cafeteria = _cafeteria;
        this.almacen = _almacen;


        //Para que cuando vuelva de minimizarse se coloque bien
        this.addWindowStateListener(e -> {
            if (e.getNewState() == JFrame.NORMAL) {
                setLocationRelativeTo(null); // Centra la ventana en la pantalla
            }
        });
```

```
    }

    public JTextField getTextoGalletasEnHorno(int identificadorTexto)
    {
          switch (identificadorTexto)
          {
             case 0:
                return numeroDeGalletasHorno1;
             case 1:
                return numeroDeGalletasHorno2;
             case 2:
                return numeroDeGalletasHorno3;
          }
         //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
          return null;
    }


    public JTextField getTextoAccionEmpaquetadores(int identificadorTexto)
    {
          switch (identificadorTexto)
          {
             case 0:
                return accionEmpaquetador1;
             case 1:
                return accionEmpaquetador2;
             case 2:
                return accionEmpaquetador3;
          }
         //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
          return null;
    }

    public void ponerColorTextoAccionEmpaquetadores(int identificador, String accion)
    {
       if (accion.equals("EMPAQUETANDO")) {getTextoAccionEmpaquetadores(identificador).setForeground(Color.black);}
       else if (accion.equals("TRANSPORTANDO"))
{getTextoAccionEmpaquetadores(identificador).setForeground(Color.magenta);}
       else if (accion.equals("ESPERANDO")) {getTextoAccionEmpaquetadores(identificador).setForeground(Color.white);}
    }


    public void checkColorPaquetes(Empaquetador empaquetador)
    {
       if(empaquetador.getIdentificador() == 1)
       {
          //Almacenamos en una variable temporal las tandas que lleva
          int tandasRecogidas = empaquetador.getTandasGalletasRecogidas();
          if(tandasRecogidas == 0)
          {
             paquete1_1.setBackground(Color.white);
             paquete1_2.setBackground(Color.white);
             paquete1_3.setBackground(Color.white);
             paquete1_4.setBackground(Color.white);
             paquete1_5.setBackground(Color.white);
          }
          else if( tandasRecogidas >= 1)
          {
             paquete1_1.setBackground(Color.black);
             if(tandasRecogidas >= 2)
             {
                paquete1_2.setBackground(Color.black);
                if(tandasRecogidas >=3)
                {
                   paquete1_3.setBackground(Color.black);
                   if(tandasRecogidas >=4)
                   {
                      paquete1_4.setBackground(Color.black);
                      if(tandasRecogidas >=5)
                      {
                         paquete1_5.setBackground(Color.black);
                      }
                   }
```

```
            }
        }
    }
}
else if(empaquetador.getIdentificador() == 2)
{
    //Almacenamos en una variable temporal las tandas que lleva
    int tandasRecogidas = empaquetador.getTandasGalletasRecogidas();
    if(tandasRecogidas == 0)
    {
        paquete2_1.setBackground(Color.white);
        paquete2_2.setBackground(Color.white);
        paquete2_3.setBackground(Color.white);
        paquete2_4.setBackground(Color.white);
        paquete2_5.setBackground(Color.white);
    }
    else if( tandasRecogidas >= 1)
    {
        paquete2_1.setBackground(Color.black);
        if(tandasRecogidas >= 2)
        {
            paquete2_2.setBackground(Color.black);
            if(tandasRecogidas >=3)
            {
                paquete2_3.setBackground(Color.black);
                if(tandasRecogidas >=4)
                {
                    paquete2_4.setBackground(Color.black);
                    if(tandasRecogidas >=5)
                    {
                        paquete2_5.setBackground(Color.black);
                    }
                }
            }
        }
    }
}
else if(empaquetador.getIdentificador() == 3)
{
    //Almacenamos en una variable temporal las tandas que lleva
    int tandasRecogidas = empaquetador.getTandasGalletasRecogidas();
    if(tandasRecogidas == 0)
    {
        paquete3_1.setBackground(Color.white);
        paquete3_2.setBackground(Color.white);
        paquete3_3.setBackground(Color.white);
        paquete3_4.setBackground(Color.white);
        paquete3_5.setBackground(Color.white);
    }
    else if( tandasRecogidas >= 1)
    {
        paquete3_1.setBackground(Color.black);
        if(tandasRecogidas >= 2)
        {
            paquete3_2.setBackground(Color.black);
            if(tandasRecogidas >=3)
            {
                paquete3_3.setBackground(Color.black);
                if(tandasRecogidas >=4)
                {
                    paquete3_4.setBackground(Color.black);
                    if(tandasRecogidas >=5)
                    {
                        paquete3_5.setBackground(Color.black);
                    }
                }
            }
        }
    }
}
}
public void ponerColorTextoHornos(int identificador, String accion)
{
```

```
        if (accion.equals("LLENANDOSE")) {getTextoAccionHornos(identificador).setForeground(Color.black);}
        else if (accion.equals("HORNEANDO")) {getTextoAccionHornos(identificador).setForeground(Color.magenta);}
        else if (accion.equals("VACIANDOSE")) {getTextoAccionHornos(identificador).setForeground(Color.white);}
    }
    public JTextField getTextoAccionHornos(int identificadorTexto)
    {
            switch (identificadorTexto)
            {
                case 0:
                    return estadoHorno1;
                case 1:
                    return estadoHorno2;
                case 2:
                    return estadoHorno3;
            }
           //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
            return null;
    }

    public void checkHornoHorneando(int indexHorno, JProgressBar progressBar)
    {
        //Comprobamos si el horno esta horneando
        if(listaHornos.get(indexHorno).getEstaHorneando() && getHilosProgressBarHornos(indexHorno) == null)
        {
            //Hilo base para las progressBar
            //Crear el hilo solo si no está en ejecución
            Thread animacionThread = new Thread(new Runnable()
            {
                @Override
                public void run()
                {
                    try
                    {
                        //Tiempo total de la animación (8 segundos)
                        int tiempoTotal = 8000;
                        int unidadesPorAvance = 1;                          //Aumentar la barra en 1 unidad por vez (a mas unidades menos
fluido)
                        int totalAvances = 100 / unidadesPorAvance;         //Número de avances (100 avances)

                        //Configurar la barra para que empiece en 0
                        progressBar.setValue(0);

                        //Ciclo para actualizar la barra de progreso
                        for (int i = 0; i <= 100; i += unidadesPorAvance) {
                            progressBar.setValue(i);                        //Actualiza el valor de la barra
                            Thread.sleep(tiempoTotal / totalAvances);       //Espera para cada avance (400 ms) (en este caso)
                        }
                    }
                    catch (InterruptedException error) {
                        System.out.println("Se ha producido un error mientras se elabora el hilo para el llenado de la progressBar --> " +
error);
                    }
                    finally
                    {
                        // Al terminar, liberamos el hilo para que se pueda crear uno nuevo la próxima vez
                        if (indexHorno == 0)
                        {
                            hiloBarraProgresoHorno1 = null;
                        }
                        if (indexHorno == 1)
                        {
                            hiloBarraProgresoHorno2 = null;
                        }
                        if (indexHorno == 2)
                        {
                            hiloBarraProgresoHorno3 = null;
                        }
                    }
                }
            });

            //Iniciar el hilo de animación
            if (indexHorno == 0) {
```

```
            hiloBarraProgresoHorno1 = animacionThread;
        }
        else if (indexHorno == 1) {
            hiloBarraProgresoHorno2 = animacionThread;
        }
        else if (indexHorno == 2) {
            hiloBarraProgresoHorno3 = animacionThread;
        }

        //Iniciamos el hilo
        animacionThread.start();
    }
    else if (!listaHornos.get(indexHorno).getEstaHorneando() && !listaHornos.get(indexHorno).getEstaEmpaquetando()&&
listaHornos.get(indexHorno).getAccion().equals("LLENANDOSE"))
    {
        //Se vacia si esta en proceso de llenado
        progressBar.setValue(0);
    }
}



public JProgressBar getProgressBarHornos(int identificadorHorno)
{
        switch (identificadorHorno)
        {
            case 0:
                return barraProgresoHorno1;
            case 1:
                return barraProgresoHorno2;
            case 2:
                return barraProgresoHorno3;
        }
       //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
        return null;
}

public Thread getHilosProgressBarHornos(int identificadorHilo)
{
    switch (identificadorHilo)
        {
            case 0:
                return hiloBarraProgresoHorno1;
            case 1:
                return hiloBarraProgresoHorno2;
            case 2:
                return hiloBarraProgresoHorno3;
        }
       //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
        return null;
}

@Override
public void run()
{
    //No queremos que el hilo pare nunca de ejecutarse
    while(true)
    {
        try
        {
            //Checkeo de estado para los reposteros
            for (int identificador = 0; identificador < listaReposteros.size(); identificador++)
            {
                String accion = listaReposteros.get(identificador).getAccion();

                //Primero cambio el color del texto
                ponerColorTextoReposteros(identificador, accion);
                //Luego cambio el texto
                getTextosReposteros(identificador).setText(accion);
            }

            //Checkeo la cafeteria
            textoCafeteria.setText(cafeteria.getReposteroAtendido());
```

```
            textoColaCafeteria.setText(cafeteria.getColaReposterosCafeteria());



            //Checkeo hornos
            for (int identificador = 0; identificador < listaHornos.size(); identificador++)
            {
               //Actualizamos las galletas que hay en cada horno
               int numeroGalletas = listaHornos.get(identificador).getCantidadGalletas();
               getTextoGalletasEnHorno(identificador).setText(String.valueOf(numeroGalletas));

               //Actualizamos el estado de los hornos
               String accion = listaHornos.get(identificador).getAccion();
               //Primero cambio el color del texto
               ponerColorTextoHornos(identificador, accion);
               //Luego cambio el texto
               getTextoAccionHornos(identificador).setText(accion);

               //Actualizamos los progressBar
               checkHornoHorneando(identificador, getProgressBarHornos(identificador));
            }


            //Checkeo empaquetadores
            for (int identificador = 0; identificador < listaEmpaquetadores.size(); identificador++)
            {
               //Recogemos la accion que estan realizando los empaquetadores
               String accion = listaEmpaquetadores.get(identificador).getAccion();

               //Escribimos la accion que esten realizando
               getTextoAccionEmpaquetadores(identificador).setText(accion);

               //Cambiamos el color del texto
               ponerColorTextoAccionEmpaquetadores(identificador, accion);


               //Checkeo el color de los paquete
               checkColorPaquetes(listaEmpaquetadores.get(identificador));
            }


            //Checkeo almacen
            textoAlmacen.setText(String.valueOf(almacen.getGalletasTotal()));



            //Generamos un breve retardo para no colapsar memoria con comprobaciones
            Thread.sleep(100);
         }
         catch (InterruptedException error)
         {
            System.out.println("Se ha producido un error mientras se ejecutaba el hilo de la interfaz de PanelSCM --> " + error);
         }
      }
   }
}
```

5.3 SharedZone

5.3.1 InterfazImplementacionRMI.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package SharedZone;
//Importes
import Server.Almacen;
import java.rmi.Remote;
import java.rmi.RemoteException;
import Server.Repostero;
import Server.Horno;
import Server.Empaquetador;
import java.util.List;
//CLASE
public interface InterfazImplementacionRMI extends Remote
{
    //TODOS LOS METODOS AÑADIDOS TENDRAN UN --> throws RemoteExecption
    public int getTotalGalletasProducidas(int indiceRepostero) throws RemoteException;
    public int getTotalGalletasDesperdiciadas(int indiceRepostero) throws RemoteException;
    public List<Repostero> getListaReposteros() throws RemoteException;
    public List<Horno> getListaHornos() throws RemoteException;
    public List<Empaquetador> getListaEmpaquetadores() throws RemoteException;
    public Almacen getAlmacen()throws RemoteException;
    public void setParadaManualDesdeCliente(boolean _paradaManual, int indexRepostero) throws
RemoteException;
}
```

5.4 Client

5.4.1 InicializacionCliente.java

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Client;
//Importes
import ClientInterface.PanelCliente;
import SharedZone.InterfazImplementacionRMI;
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
//CLASE
public class InicializacionCliente
{
    public static void main(String[] args)
    {
        /**********************************
         *   INICIALIZACIÓN DE CLIENTE    *
         **********************************/
        try
        {
            InterfazImplementacionRMI metodosServer = (InterfazImplementacionRMI)
Naming.lookup("//localhost/objetoTraspasable");

            /**********************************
            *   INICIALIZACIÓN DE INTERFAZ   *
            **********************************/
            //Inicializamos la interfaz del servidor
            PanelCliente interfazCliente = new PanelCliente(metodosServer);
            interfazCliente.setVisible(true);
            Thread hiloInterfazCliente = new Thread(interfazCliente);
            hiloInterfazCliente.start();

        }
        catch (MalformedURLException error)
        {
            System.out.println("Se ha producido un error de MalformedURLException cuando se importaba
un objeto del Servidor en el cliente --> " + error );
        }
        catch (NotBoundException error)
```

```
        {
            System.out.println("Se ha producido un error de NotBoundException cuando se importaba un
objeto del Servidor en el cliente --> " + error );
        }
        catch (RemoteException error)
        {
            System.out.println("Se ha producido un error de RemoteException cuando se importaba un
objeto del Servidor en el cliente --> " + error );
        }
    }
}
```

5.5 ClientInterface

5.5.1 PanelCliente.java (Se ha eliminado el código generado automáticamente por NetBeans)

```
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package ClientInterface;
//Importes
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import SharedZone.InterfazImplementacionRMI;
import java.rmi.RemoteException;
import Server.Repostero;
import Server.Horno;
import Server.Empaquetador;
import Server.Almacen;
//CLASE

public class PanelCliente extends javax.swing.JFrame implements Runnable{

    //Atributos
    //Aqui guardaremos la info de todos los reposteros, hornos y empaquetadores
    InterfazImplementacionRMI metodosServer;
    //Guardamos las posiciones del mouse en la pantalla
    int xMouse;
    int yMouse;
    public PanelCliente(InterfazImplementacionRMI _metodosServer) {
        //Gestiono el argumento
        this.metodosServer = _metodosServer;
        initComponents();
        Image icono =
Toolkit.getDefaultToolkit().getImage(PanelCliente.class.getResource("/Fototeca/iconoAplicacionCliente.p
ng"));
        setIconImage(icono);
        //Centramos la ventana en la pantalla del usuario que ejecute la aplicacion
        setLocationRelativeTo(null);
        //Para que cuando vuelva de minimizarse se coloque bien
        this.addWindowStateListener(e -> {
            if (e.getNewState() == JFrame.NORMAL) {
```

```
                setLocationRelativeTo(null); // Centra la ventana en la pantalla
            }
        });
    }
    private void iconoMinimizadoMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_iconoMinimizadoMouseClicked
        Point startLocation = this.getLocation();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int taskbarHeight =
Toolkit.getDefaultToolkit().getScreenInsets(this.getGraphicsConfiguration()).bottom;
        Point endLocation = new Point(
            (int) screenSize.getWidth() / 2,
            (int) screenSize.getHeight() - taskbarHeight
        );
        Timer timer = new Timer(10, null);
        timer.addActionListener(new ActionListener() {
            double progress = 0;
            final double step = 0.05;
            @Override
            public void actionPerformed(ActionEvent evt) {
                progress += step;
                if (progress >= 1.0) {
                    timer.stop();
                    setExtendedState(JFrame.ICONIFIED);
                } else {
                    int x = (int) (startLocation.x + progress * (endLocation.x - startLocation.x));
                    int y = (int) (startLocation.y + progress * (endLocation.y - startLocation.y));
                    setLocation(x, y);
                }
            }
        });
        timer.start();
    }//GEN-LAST:event_iconoMinimizadoMouseClicked
    private void botonPararRepostero2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_botonPararRepostero2ActionPerformed
        //Hay que generar las dos casuisticas en funcion del texto que haya en el boton
        if (botonPararRepostero2.getText().equals("PARAR"))
        {
            //Debemos de para la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(true, 1);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 2");
```

```
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero2.setText("REANUDAR");
            botonPararRepostero2.setBackground(Color.green);
        }
        else
        {
            //Debemos de reanudar la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(false, 1);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 2");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero2.setText("PARAR");
            botonPararRepostero2.setBackground(Color.red);
        }
    }//GEN-LAST:event_botonPararRepostero2ActionPerformed

    private void botonPararRepostero3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_botonPararRepostero3ActionPerformed
        //Hay que generar las dos casuisticas en funcion del texto que haya en el boton
        if (botonPararRepostero3.getText().equals("PARAR"))
        {
            //Debemos de para la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(true, 2);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 3");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero3.setText("REANUDAR");
            botonPararRepostero3.setBackground(Color.green);
        }
        else
        {
```

```
            //Debemos de reanudar la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(false, 2);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 3");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero3.setText("PARAR");
            botonPararRepostero3.setBackground(Color.red);
        }
    }//GEN-LAST:event_botonPararRepostero3ActionPerformed

    private void botonPararRepostero4ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_botonPararRepostero4ActionPerformed
        //Hay que generar las dos casuisticas en funcion del texto que haya en el boton
        if (botonPararRepostero4.getText().equals("PARAR"))
        {
            //Debemos de para la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(true, 3);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 4");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero4.setText("REANUDAR");
            botonPararRepostero4.setBackground(Color.green);
        }
        else
        {
            //Debemos de reanudar la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(false, 3);
            }
            catch(RemoteException error)
            {
```

```
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el repostero 4");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero4.setText("PARAR");
            botonPararRepostero4.setBackground(Color.red);
        }
    }//GEN-LAST:event_botonPararRepostero4ActionPerformed

    private void botonPararRepostero5ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_botonPararRepostero5ActionPerformed
        //Hay que generar las dos casuisticas en funcion del texto que haya en el boton
        if (botonPararRepostero5.getText().equals("PARAR"))
        {
            //Debemos de para la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(true, 4);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el repostero 5");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero5.setText("REANUDAR");
            botonPararRepostero5.setBackground(Color.green);
        }
        else
        {
            //Debemos de reanudar la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(false, 4);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el repostero 5");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero5.setText("PARAR");
            botonPararRepostero5.setBackground(Color.red);
        }
```

```
    }//GEN-LAST:event_botonPararRepostero5ActionPerformed

    private void botonPararRepostero1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_botonPararRepostero1ActionPerformed
       //Hay que generar las dos casuisticas en funcion del texto que haya en el boton
        if (botonPararRepostero1.getText().equals("PARAR"))
        {
            //Debemos de para la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(true, 0);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 1");
            }

            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero1.setText("REANUDAR");
            botonPararRepostero1.setBackground(Color.green);
        }
        else
        {
            //Debemos de reanudar la ejecucion del repostero
            try
            {
                metodosServer.setParadaManualDesdeCliente(false, 0);
            }
            catch(RemoteException error)
            {
                System.out.println("Se ha producido el siguiente error mientras se intentaba parar el
repostero 1");
            }


            //Una vez cambiado el estado del repostero actualizamos la apariencia del boton
            botonPararRepostero1.setText("PARAR");
            botonPararRepostero1.setBackground(Color.red);
        }
    }//GEN-LAST:event_botonPararRepostero1ActionPerformed

    //METODOS CREADOS A MANO
    public JTextField getTextosReposterosGalletas(int identificadorTexto)
    {
            switch (identificadorTexto)
            {
```

```
            case 0:
                return textoRepostero1Galletas;
            case 1:
                return textoRepostero2Galletas;
            case 2:
                return textoRepostero3Galletas;
            case 3:
                return textoRepostero4Galletas;
            case 4:
                return textoRepostero5Galletas;
        }

        //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
        return null;
}

public JTextField getTextosReposterosGalletasDesperdiciadas(int identificadorTexto)
{
        switch (identificadorTexto)
        {
            case 0:
                return textoRepostero1GalletasDesperdiciadas;
            case 1:
                return textoRepostero2GalletasDesperdiciadas;
            case 2:
                return textoRepostero3GalletasDesperdiciadas;
            case 3:
                return textoRepostero4GalletasDesperdiciadas;
            case 4:
                return textoRepostero5GalletasDesperdiciadas;
        }
       //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
        return null;
}


public JTextField getTextosTotalGalletasHorneadas(int identificadorTexto)
{
        switch (identificadorTexto)
        {
            case 0:
                return textoHorno1GalletasHorneadas;
            case 1:
                return textoHorno2GalletasHorneadas;
            case 2:
                return textoHorno3GalletasHorneadas;
        }
```

```
            //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
            return null;
    }

    public JTextField getPanelControlHorneado(int identificadorTexto)
    {
            switch (identificadorTexto)
            {
                case 0:
                    return panelControlHorneado1;
                case 1:
                    return panelControlHorneado2;
                case 2:
                    return panelControlHorneado3;
            }

            //Si no entra en ningun case, no devolvemos anda porque no han seleccionado el indice bien
            return null;
    }

    @Override
    public void run() {
        //la interfaz del cliente estara constantemente refrescando la informacion
                while(true)
                {
                    try
                    {
                        //REPOSTEROS
                        //Checkeamos las galletas totales producidas por cada repostero y lo imprimimos
en el JTextField
                        //Checkeamos las galletas totales desperdiciadas por cada repostero y lo
imprimimos en el JTextField
                        List<Repostero> listaReposteros = metodosServer.getListaReposteros();
                        for (int indice = 0; indice < listaReposteros.size(); indice++ )
                        {
                            int galletasTotales = listaReposteros.get(indice).getTotalGalletas();
                            int galletasTotalesDesperdiciadas =
listaReposteros.get(indice).getTotalGalletasDesperdiciadas();
                            getTextosReposterosGalletas(indice).setText(String.valueOf(galletasTotales)
);
                            getTextosReposterosGalletasDesperdiciadas(indice).setText(String.valueOf(ga
lletasTotalesDesperdiciadas));
                        }

                        //HORNOS
                        List<Horno> listaHornos = metodosServer.getListaHornos();
```

```
                        for (int indice = 0; indice < listaHornos.size(); indice++)
                        {
                            //Checkeamos el total galletas horneadas
                            int totalGalletasHorneadas =
listaHornos.get(indice).getTotalGalletasHorneadas();
                            getTextosTotalGalletasHorneadas(indice).setText(String.valueOf(totalGalleta
sHorneadas));

                            //Check si esta horneando (SI o NO)
                            if(listaHornos.get(indice).getEstaHorneando())
                            {
                                getPanelControlHorneado(indice).setBackground(Color.red);
                            }
                            else
                            {
                                getPanelControlHorneado(indice).setBackground(Color.white);
                            }
                        }
                        //ALMACEN
                        textoAlmacenGalletasAlmacenadas.setText(String.valueOf(metodosServer.getAlmacen
().getGalletasTotalAlmacenadas()));
                        textoAlmacenGalletasConsumidas.setText(String.valueOf(metodosServer.getAlmacen(
).getGalletasTotalConsumidas()));

                        //Finalizamos el checkeo para el cliente y esperamos un segundo hasta el
siguiente checkeo
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException error)
                    {
                        System.out.println("Se ha producido un error mientras la interfaz del cliente
estaba actualizando los datos de la fabrica --> " + error);
                    }
                    catch (RemoteException error)
                    {
                        System.out.println("Se ha producido un error mientras se usaba algun hilo
(Reposteros,Hornos,Empaquetadores) del servidor --> " + error);
                    }
                }
            }
        }
```

5.6 Logger

5.6.1 Logger.java

```java
/**
 * version 1.0
 * contacto --> https://www.linkedin.com/in/david-sanchez-1366b1253/
 * @author David Sánchez
 */
//Paquete al que pertenece la clase
package Logger;
//Importes
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Serializable;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class Logger implements Serializable
{
    private static final String NOMBRE_ARCHIVO = "evolucionGalletas.txt";
    private static Logger instance;

    private Logger()
    {
        //Crear el archivo si no existe (opcional, para asegurar su existencia al inicio)
        try (BufferedWriter salida = new BufferedWriter(new FileWriter(NOMBRE_ARCHIVO, true)))
        {
            salida.write("=== Inicio de Logs: " + getCurrentTimestamp() + " ===\n");
        }
        catch (IOException error)
        {
            System.out.println("Error al inicializar el log --> " + error);
        }
    }
    //Método para obtener la instancia única
    public static synchronized Logger getInstance()
    {
        if (instance == null)
        {
            instance = new Logger();
        }
        return instance;
    }
    //Método para registrar un mensaje en el log
    public synchronized void log(String mensaje)
    {
        String mensajeLog = "[" + getCurrentTimestamp() + "] " + mensaje;
```

```
        //Mostrar el mensaje en consola
        System.out.println(mensajeLog);

        //Guardar el mensaje en el archivo
        try (BufferedWriter salida = new BufferedWriter(new FileWriter(NOMBRE_ARCHIVO, true)))
        {
            salida.write(mensajeLog + "\n");
        }
        catch (IOException error)
        {
            System.out.println("Error al escribir en el log -->" + error);
        }
    }
    //Método para obtener la marca de tiempo actual
    private String getCurrentTimestamp()
    {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        return LocalDateTime.now().format(formatter);
    }
}
```