



GRADO EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

PECL1 – CORREOS SL

CURSO 2022/2023 – CONVOCATORIA ORDINARIA

RUBEN ADARVE

09124276S

ISMAEL RUIZ RANZ

09106137T



Índice

Análisis de alto nivel	2
Objetivo.....	2
Relación cliente – buzón	2
Relación buzón – empleado.....	3
Relación cliente – buzón – empleado	3
Programación distribuida.....	4
Diseño general del sistema y herramientas de sincronización	4
CorreosSL	5
CorreosSLCliente	5
Buzón	5
Empleado	5
InterfaceDistribuida	5
Persona	5
Pausar.....	5
Objeto remoto	6
LoggerUtils	6
Furgoneta	6
Clases principales que intervienen.....	7
Clase cliente	7
Clase Buzón	8
Clase Empleado.....	8
Clase Interfaz Distribuida	9
Clase Objeto Remoto	9
Clase Pausar	10
Clase CorreosSL.....	10
Clase CorreosSLCliente.....	11
Clase furgoneta	11
Diagrama de clases	12
Anexo.....	13

Análisis de alto nivel

Objetivo

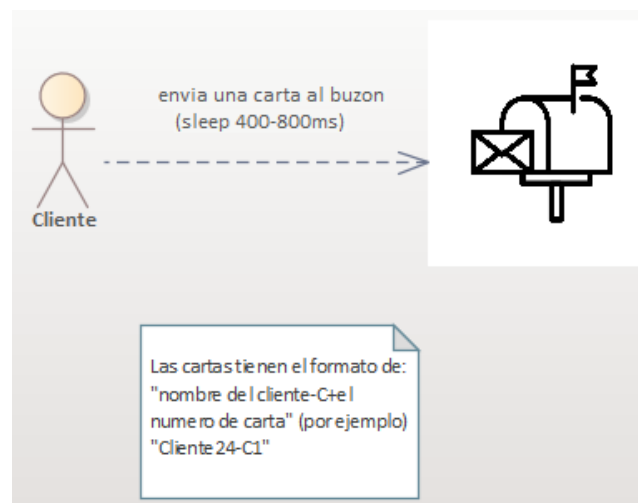
Se desea modelar el comportamiento de un grupo de 400 personas (que serán modeladas como hilos) que van dejando, cada una de ellas, dos cartas en un buzón. Cada persona tardará entre 400 y 800 ms entre dejar una carta y otra. La capacidad máxima del buzón es de 30 cartas. Por otro lado, hay 5 empleados (que también deberán ser modelados como hilos) de la empresa de cartas que se encargan de coger cada una de las cartas del buzón y depositarlas en dos furgonetas de reparto. Debido a las características de cada furgoneta, sólo un empleado puede entrar a depositar una carta a la vez en una determinada furgoneta. La primera carta de cada persona va a la primera furgoneta, y la segunda carta de cada persona va a la segunda furgoneta.

Las personas sólo podrán dejar las cartas en el buzón mientras haya hueco disponible, y si no deberán esperar a que lo haya. Por otro lado, los empleados deberán esperar a que haya cartas disponibles en el buzón para poderlas llevar a las distintas furgonetas. Los empleados tardarán entre 400 y 700 ms desde que cogen una carta hasta que la llevan a una furgoneta, y el mismo tiempo en volver al buzón. **Se busca que haya una relación entre cliente-buzón-repartidor.** Para ello tendremos que subdividir estas relaciones haciendo **cliente-buzón** y **buzón-repartidor**.

Relación cliente – buzón

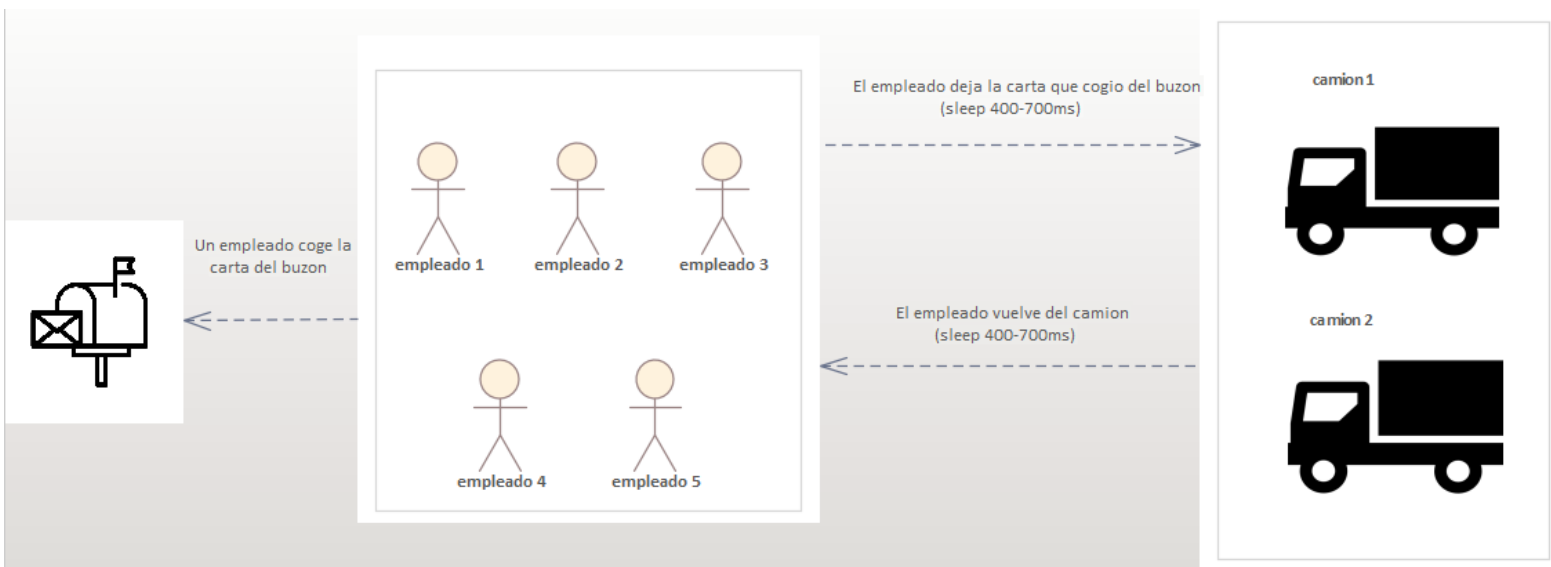
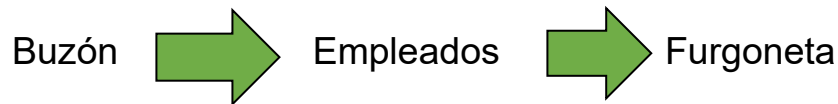
En esta relación trata de que el cliente deja cierto número de cartas en el buzón en nuestro caso como dice el enunciado deja 2 cartas. El formato de cada carta es el siguiente: “IDdeSuPersona+Cx”. (**ejemplo: Cliente24-C1**). El buzón va a hacer de contenedor para que las personas puedan dejar sus cartas y los empleados puedan coger dichas cartas y dejarlas en su furgoneta correspondiente. **El tamaño del buzón es de 30 cartas controlado con semáforos.** El cliente tarda en dejar la carta y volver a enviar la segunda entre 400-800ms.

Clientes  Buzón



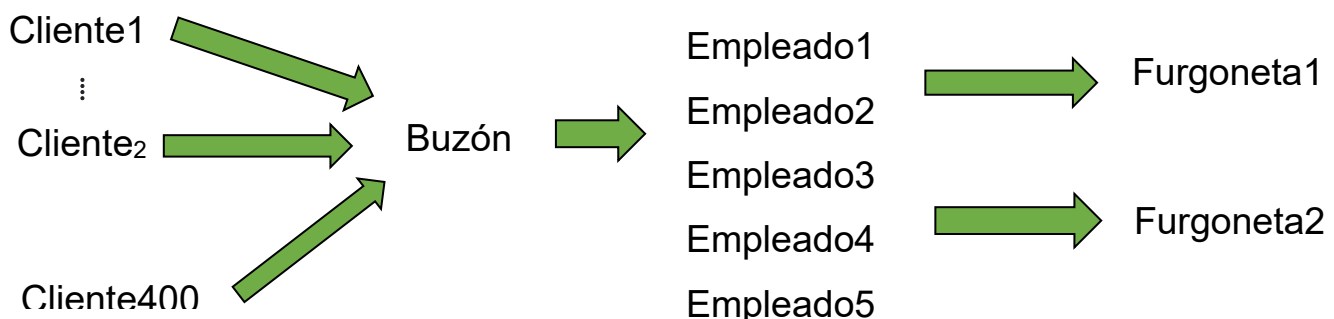
Relación buzón – empleado

Aquí podemos apreciar como si hay cartas en el buzón los repartidores procederán a dejar las cartas en la furgoneta correspondiente según el número que ponga al final de la carta. Solo puede haber un repartidor llevando las cartas al camión, este tardará entre 400-700ms en dejar la carta en la furgoneta y luego el mismo tiempo para volver al buzón. Hay 5 empleados que pueden ser parados en la interfaz en cualquier momento.



Relación cliente – buzón – empleado

Después de ver las relaciones divididas ahora veremos como quedaría todo el esquema para tener un mejor entendimiento de problema ya que anteriormente los resumí al máximo para que se vea mejor el funcionamiento del programa.



Se puede observar cómo debería funcionar la forma concurrente que sigue un procedimiento lineal, el cual es: el cliente deja una carta en el buzón, el repartidor mira el buzón y si puede coger cartas coje y la almacena en la furgoneta correspondiente y así todo el rato hasta que las personas no tengas más cartas que dejar y el buzón este vacío.

Cada elemento del esquema presentado son objetos de las clases, estas tienen ciertas relaciones que podremos ver en el diagrama de clases más adelante. Todo esto tendrán interacciones entre las clases correspondientes, pero habrá otras clases que las llamaremos clases auxiliares ya que no son esenciales para la funcionalidad del enunciado, pero sirven para cumplir otra función como por ejemplo la clase Pausar que sirve para parar los hilos empleado con cerrojos, también la clase que implementa cliente-servidor (RMI) el cual declara el objeto remoto en el **main** y la clase del cliente llama al buzón y las 2 furgonetas para ver su resultado por pantalla.

Programación distribuida

Hemos implementado el servidor en el **main de CorreosSL**, con su objeto remoto y su interfaz correspondiente para que el cliente pueda ver en tiempo real los resultados del buzón y las dos furgonetas.

CorreosSL (servidor)  Interfaz  **CorreosSLCliente** (Cliente)

La clase que se encarga de guardar los pasos que van ocurriendo en la simulación y sacar el txt, “**evolucionCartas.txt**”, es LoggerUtils. Esta nos sirve a nosotros para poder tener un seguimiento en cada momento de lo que está ocurriendo en la aplicación como copia de seguridad.

Diseño general del sistema y herramientas de sincronización

Este proyecto consta de diferentes clases que vamos a comentar a continuación para comprender el funcionamiento de la simulación por completo:

 CorreosSL	 Persona
 CorreosSLCliente	 Pausar
 Buzón	 Objeto Remoto
 Empleado	 LoggerUtils
 InterfaceDistribuida	 Furgoneta

CorreosSL

El servidor que inicializa la interfaz, se encarga de registrar el objeto remoto para que el cliente pueda recibir los mismos datos que estaremos mostrando en el buzón, furgoneta1 y furgoneta2. En esta clase se inicializarán las 400 personas, 5 empleados y la interfaz que mostrará el servidor.

CorreosSLCliente

Al inicializar este main primero tenemos que tener el servidor encendido sino no mostrará nada. Después de encender el servidor arrancaremos el cliente, este tendrá un método llamado mostrar que **estará mostrando en sus respectivos JTextPane** los resultados del buzón, furgoneta1, furgoneta 2 y el tamaño del buzón en tiempo real.

Buzón

Esta clase va a ser un objeto compartido tanto por las personas como los empleados, encargado de recibir las cartas de las personas e ir almacenándolas hasta llegar a su límite que en este caso es 30 cartas. Las personas estarán introduciendo cartas mientras haya sitio en el buzón sino estas esperarán. Los repartidores serán los encargados de recoger las cartas del buzón y mientras haya cartas podrán coger cartas sino esperarán a que introduzcan una.

Empleado

Esta clase va a ser un hilo y por lo tanto van a poder trabajar en paralelo a otros empleados. En el main se declaran 5 empleados que serán los encargados de coger las cartas del buzón y dejarlas en las furgonetas en función del identificador de la carta.

InterfaceDistribuida

Encargada de exportar los métodos que se van a usar en la programación distribuida o cliente – servidor.

Persona

Esta clase va a ser un hilo y por lo tanto va a poder trabajar en paralelo a otras personas. En el main se declaran 400 personas con un **bucle for que va desde 1 hasta 401**. Cada persona dejará una carta y esperará 400 – 800 ms hasta dejar la segunda carta.

Pausar

Clase encargada de pausar los hilos empleados o el programa entero gracias a los locks y los conditions.

Objeto remoto

La clase compartida para el funcionamiento del cliente – servidor, se encuentran los métodos que aparecen en **InterfaceDistribuida**


LoggerUtils

La clase encargada de guardar los pasos que van ocurriendo en la simulación y sacar el txt con todos los pasos que van ocurriendo en la simulación.

Furgoneta

Clase que hace de contenedor de cartas mediante un arrayList, en cuanto el repartidor termina su tiempo de coger la carta, introduce la carta correspondiente en la furgoneta que le toque. La furgoneta que le toque se puede ver gracias al método auxiliar que hemos realizado en el buzón llamado: **public int getFurgo (string carta)**, que se encarga de devolver el identificador de la carta para saber a qué camión tiene que ir la carta.

Programación concurrente CorreosSL (servidor)

 — □ ×

Numero de cartas en el Buzón

Contenido del Buzón

Contenido del buzón

[Persona222-C1, Persona350-C1, Persona297-C1, Persona247-C1, Persona281-C1, Persona149-C1, Persona29-C1, Persona129-C1, Persona43-C1, Persona371-C1, Persona60-C1, Persona400-C1, Persona344-C1, Persona78-C1, Persona110-C1, Persona257-C1, Persona97-C1, Persona151-C1, Persona91-C1, Persona192-C1, Persona342-C1, Persona18-C1, Persona346-C1, Persona317-C1, Persona122-C1, Persona331-C1, Persona161-C1, Persona13-C1, Persona63-C1, Persona244-C1]

Empleado 1

Empleado1 depositando Persona269-C1 en la furgoneta 1

Pausar Empleado 1

Empleado 2

Empleado2 volviendo de la furgoneta 1

Pausar Empleado 2

Empleado 3

Empleado3 coge la carta Persona38-C1 y va a depositarla en la furgoneta 1

Pausar Empleado 3

Empleado 4

Empleado4 coge la carta Persona52-C1 y va a depositarla en la furgoneta 1

Pausar Empleado 4

Empleado 5

Empleado5 coge la carta Persona117-C1 y va a depositarla en la furgoneta 1

Pausar Empleado 5


Contenido de la Furgoneta 1 :

[Persona74-C1, Persona295-C1, Persona382-C1, Persona229-C1, Persona170-C1, Persona325-C1, Persona34-C1, Persona234-C1, Persona69-C1, Persona5-C1, Persona72-C1, Persona269-C1]

Contenido de la Furgoneta 2

Pausar todo

Programación distribuida CorreosSLCliente (cliente)


— □ ×

Contenido del Buzón
Numero de cartas en el Buzón

[Persona161-C1, Persona13-C1, Persona63-C1, Persona244-C1, Persona96-C1, Persona188-C1, Persona155-C1, Persona23-C1, Persona221-C2, Persona227-C1, Persona281-C2, Persona193-C1, Persona351-C1, Persona175-C1, Persona214-C1, Persona345-C1, Persona279-C1, Persona51-C1, Persona84-C1, Persona164-C1, Persona150-C1, Persona147-C1, Persona189-C1, Persona162-C1, Persona283-C1, Persona85-C1, Persona319-C1, Persona352-C1, Persona34-C2, Persona49-C1]

Furgoneta 1 :

[Persona74-C1, Persona295-C1, Persona382-C1, Persona229-C1, Persona170-C1, Persona325-C1, Persona34-C1, Persona234-C1, Persona69-C1, Persona5-C1, Persona72-C1, Persona269-C1, Persona38-C1, Persona117-C1, Persona52-C1, Persona221-C1, Persona222-C1, Persona350-C1, Persona297-C1, Persona247-C1, Persona281-C1, Persona149-C1, Persona29-C1, Persona129-C1, Persona43-C1, Persona371-C1, Persona60-C1, Persona400-C1, Persona344-C1, Persona78-C1, Persona110-C1, Persona257-C1, Persona97-C1, Persona151-C1, Persona91-C1, Persona192-C1, Persona242-C1, Persona18-C1, Persona348-C1, Persona317-C1, Persona123-C1, Persona234-C1]

Furgoneta 2

Se puede ver en las imágenes que los datos son distintos no porque esté mal, sino porque el servidor se ejecutó para hacer la captura y la del cliente se hizo con otra ejecución del servidor no fueron en la misma sesión.

Clases principales que intervienen

Clase cliente

La función de dicha clase ya la comenté anteriormente ahora voy a comentar el funcionamiento de dicha clase con sus atributos y luego sus métodos.

Atributos

private final String nombre: nombre de dicha persona.

private final Buzon buzón: El objeto compartido para dejar las cartas

private final LoggerUtils logger: encargado de introducir datos en el txt.

Métodos

public Persona (String nombre, Buzon buzón, LoggerUtils logger): constructor de la clase
public void run (): método importantísimo para un hilo ya que sin este no podrá hacer la función que va a desempeñar. Primero se declara una carta de tipo string con su nombre y un identificador, después se introduce en el buzón y este hace un sleep de 400-800ms, después hace el mismo procedimiento y termina su ejecución.

Clase Buzón

Encargado de hacer de contenedor en este productor – consumidor de personas y repartidores. Si se llena el buzón las personas tendrán que esperar a dejar cartas y si no hay cartas para recoger los repartidores tendrán que esperar.

Atributos

private final ArrayList<String> buzón: contenedor de las cartas
private final Semaphore ExclusionMutua = new Semaphore(1): control de Exclusión mutua
private final Semaphore vacio = new Semaphore(0): número de cartas
private final Semaphore lleno: huecos que quedan libres
private final JTextPane salidaBuzon: salida del buzón en la interfaz
private final JTextPane TamanoBuzon: salida del tamaño del buzón en la interfaz

Métodos

public Buzon(int maximo, JTextPane panel, JTextPane tamanoBuzon): constructor
public String cogerCarta(): coger carta y devolver la carta que coge del array
public void dejarCarta(String carta): Introduce la carta de la persona en el array
public int getFurgo(String carta): Obtiene el identificador de la carta para ir a la furgoneta
public int getCartasBuzon(): control para el empleado en su TextPane

Clase Empleado

Se encarga de coger las cartas del buzón y llevarlas a su furgoneta correspondiente según el identificador de dichas cartas. Solo cogerá cartas cuando haya cartas disponibles en el buzón sino este esperará a que las personas envíen sus cartas.

Atributos

private final String nombre: nombre del repartidor
private final Buzon buzón: objeto compartido donde coje las cartas
private final Furgoneta furgoneta1: donde van las cartas con el ID 1
private final Furgoneta furgoneta2: donde van las cartas con el ID 2
private final LoggerUtils logger: sirve para almacenar información en el txt
private final JTextPane salidaEmpleado: jtextPane de las acciones del empleado
private final Pausar pausa: encargado de hacer parar el hilo

Métodos

public Empleado(String nombre, Buzon buzón, Furgoneta furgo1, Furgoneta furgo2, LoggerUtils logger, JTextPane panel, Pausar pausaEspecifica): constructor

public void run(): método importantísimo para un hilo ya que sin este no podrá hacer la función que va a desempeñar. ejecución que va a seguir el empleado todo el rato hasta dejar el buzón vacío con todas las cartas entregadas. El funcionamiento de este es el siguiente: mirar si alguien ha dado al botón de pausar su funcionamiento, después obtiene la carta que sea y según el identificador de la carta lo envía a la furgoneta 1 o la 2.

Clase Interfaz Distribuida

Clase que contiene la interfaz remota, es decir, los métodos que serán implementados por el servidor. Encargada de exportar los métodos que se van a usar en la programación distribuida o cliente – servidor.

Métodos

String getBuzon() throws RemoteException: devuelve el contenido del buzón

String getfurgo1() throws RemoteException devuelve el contenido de la furgo 1

String getfurgo2() throws RemoteException: devuelve el contenido de la furgo 2

String getCartasBuzon() throws RemoteException: devuelve el tamaño del buzón

Clase Objeto Remoto

Esta clase la implementa el servidor registrando el objeto remoto en cuestión para que pueda obtener los siguientes métodos el cliente.

Atributos

private final CorreosSL interfaces: se declara cuando se instancia el constructor para acceder a sus métodos.

Métodos

public ObjetoRemoto(CorreosSL inter) throws RemoteException: constructor de clase

public String getBuzon(): obtiene el contenido del buzón

public String getfurgo1(): obtiene el contenido de la furgo 1

public String getfurgo2(): obtiene el contenido de la furgo 1

public String getCartasBuzon(): obtiene el tamaño del buzón

Clase Pausar

La clase de implementar la funcionalidad de los botones de los empleados para que estos se queden parados cuando sean pulsados.

Atributos

private boolean cerrado = false: control para saber si esta activada la pausa

private final Lock cerrojo = new ReentrantLock(): generar una espera activa

private final Condition parar = cerrojo.newCondition(): sirve para implementar espera activa en un método en específico

Métodos

public void mirar(): Si esta pausado hace una espera activa hasta que se vuelva a pulsar

public void abrir(): sirve para terminar la espera activa del hilo

public void cerrar(): sirve para generar la espera activa.

public boolean estaPausado(): devuelve si está abierto o no mediante un boolean

Clase CorreosSL

La clase encargada de comenzar la simulación se instancian los hilos personas, empleados, objetos remotos, pausas y objetos compartidos.

Atributos

private boolean Pausa = true: sirve para la interfaz sepa si está parada o no

private final Pausar TOTALPAUSE = new Pausar(): funcionalidad para parar todo

private final Pausar pause1 = new Pausar(): funcionalidad para parar un empleado, hay pause hasta 5 pero no he puesto todos por no repetirlo en la memoria.

Métodos

public CorreosSL(): encargado de inicializar los hilos y pasarles los objetos correspondientes

public String getBuzon(): getter que devuelve el texto del buzón

public String getfurgo1(): getter que devuelve el texto de la furgoneta 1

public String getfurgo2(): getter que devuelve el texto de la furgoneta 2

public String getCartasBuzon(): getter que devuelve el texto del tamaño del buzón

private void jButton1ActionPerformed: acción para darle funcionalidad al botón parar todo

private void jToggleButton1ActionPerformed: acción para darle funcionalidad al botón de para empleado, hay 5 métodos de estos pero no he puesto todos para no repetirlo en la memoria

public static void main(String args[]): crea la interfaz y la muestra todo el rato, también instanciamos el objeto remoto para que actúe de servidor entre el cliente.

Clase CorreosSLCliente

Clase que de forma remota es capaz de visualizar el contenido del buzón, furgoneta 1, furgoneta 2 y el tamaño del buzón. Para eso tiene que estar funcionando primero la **clase CorreosSL**.

Atributos

private InterfaceDistribuida correos: interfaz remota que trae consigo sus métodos

Métodos

public CorreosSLCliente(): encargado de obtener el objeto remoto instanciado en el servidor

public void Mostrar(): usa los 4 métodos de la interfaz para mostrarlos al cliente los datos

public static void main(String args[]): solo crea la interfaz y hace que este visible

Clase furgoneta

Clase final encargada de obtener las cartas que el repartidor le introduce y mostrárselo tanto en la programación concurrente como en la programación distribuida.

Atributos

private final ArrayList<String> furgo: contenedor simulando la furgoneta

private final String nombreFurgo: nombre de la furgoneta

private final Semaphore ExclusionMutua = new Semaphore(1): control exclusión mutua

private final Semaphore lleno = new Semaphore(1): solo puede haber 1 repartidor

private final LoggerUtils logger: encargado de mostrar los datos en el txt

private final JTextPane salidaFurgo: mostrando el contenido del arrayList

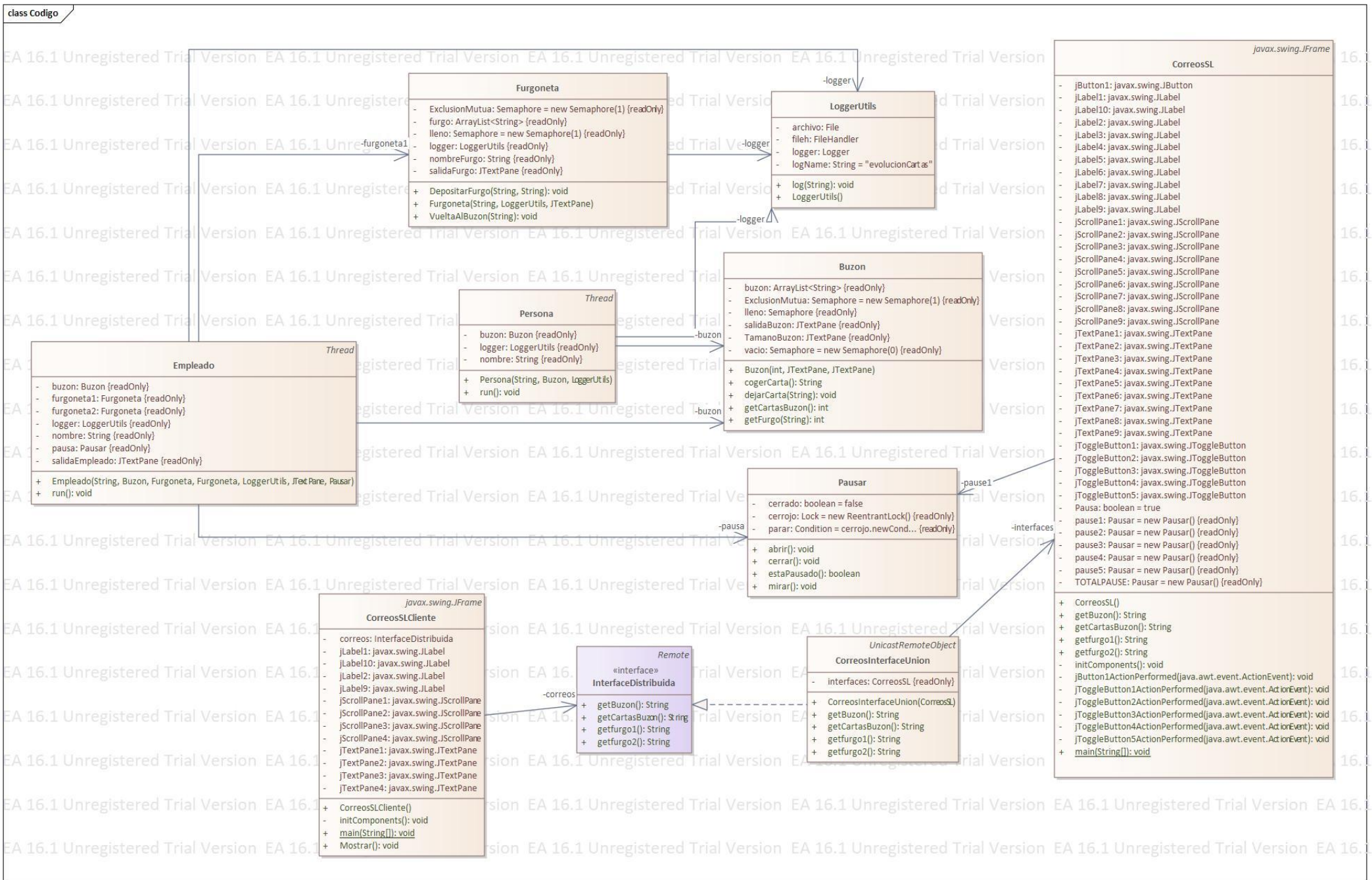
Métodos

public Furgoneta(String nombre, LoggerUtils logger, JTextPane panel): constructor de la clase

public void DepositarFurgo(String carta, String nombre): acción para insertar la carta

public void VueltaAlBuzon(String nombre): libera el semáforo para que dejen las cartas

Diagrama de clases



Anexo

Clase buzón

```
packageCodigo;
import java.util.ArrayList;
import java.util.concurrent.Semaphore;
import javax.swing.JOptionPane;
public class Buzon {
    private final ArrayList<String> buzon;
    private final Semaphore ExclusionMutua = new Semaphore(1);
    private final Semaphore vacio = new Semaphore(0); // numero de elementos
    private final Semaphore lleno; // numero de huecos
    private final JOptionPane salidaBuzon;
    private final JOptionPane TamanoBuzon;
    public Buzon(int maximo, JOptionPane panel, JOptionPane tamanoBuzon) {
        this.buzon = new ArrayList<>(maximo);
        this.lleno = new Semaphore(maximo);
        this.salidaBuzon = panel;
        this.TamanoBuzon = tamanoBuzon;
    }
    public String cogerCarta() throws InterruptedException {
        try {
            vacio.acquire();
            ExclusionMutua.acquire();
            String carta = buzon.get(0);
            buzon.remove(0);
            if (buzon.isEmpty()) {
                salidaBuzon.setText(""); // para cuando se inicializa no salga vacio
            } else {
                salidaBuzon.setText("" + buzon);
            }
            TamanoBuzon.setText(String.valueOf(buzon.size()));
            return carta;
        } finally {
            ExclusionMutua.release();
            lleno.release();
        }
    }
    public void dejarCarta(String carta) throws InterruptedException {
        try {
            lleno.acquire();
            ExclusionMutua.acquire();
```



```
buzon.add(carta);
if (buzon.isEmpty()) {
    salidaBuzon.setText(""); //para cuando se inicializa no salga vacio esto : []
} else {
    salidaBuzon.setText("" + buzon);
}
TamanoBuzon.setText(String.valueOf(buzon.size()));
} finally {
    ExclusionMutua.release();
    vacio.release();
}
}

public int getFurgo(String carta) {
    String Identificador;
    Identificador = carta;
    int id;
    id = Integer.parseInt(Identificador.substring(Identificador.length() - 1));
    return id;
}

public int getCartasBuzon() {
    return this.buzon.size();
}
}
```

Clase Empleado

```
packageCodigo;
import javax.swing.JOptionPane;
public class Empleado extends Thread {
    private final String nombre;
    private final Buzon buzon;
    private final Furgoneta furgoneta1;
    private final Furgoneta furgoneta2;
    private final LoggerUtils logger;
    private final JOptionPane salidaEmpleado;
    private final Pausar pausa;
    public Empleado(String nombre, Buzon buzon, Furgoneta furgo1, Furgoneta furgo2,
        LoggerUtils logger, JOptionPane panel, Pausar pausaEspecificas) {
        this.nombre = nombre;
        this.buzon = buzon;
        this.furgoneta1 = furgo1;
        this.furgoneta2 = furgo2;
        this.logger = logger;
    }
}
```

```
this.salidaEmpleado = panel;
this.pausa = pausaEspecificas;
}
public void run() {
    long TiempoDejarCarta = 0;
    String salida;
    String carta;
    while (true) {
        pausa.mirar();
        TiempoDejarCarta = (long) 400 + (int) ((700 - 400) * Math.random());
        try {
            carta = buzon.cogerCarta();
            sleep(TiempoDejarCarta);
            TiempoDejarCarta = (long) 400 + (int) ((700 - 400) * Math.random());
            if (buzon.getFurgo(carta) == 1) {
                salida = nombre + " coge la carta " + carta + " y va a depositarla en la
furgoneta 1";
                salidaEmpleado.setText(salida);
                logger.log(nombre + " coge la carta " + carta + " y va a depositarla en la
furgoneta 1");
                furgoneta1.DepositarFurgo(carta, nombre);
                salida = nombre + " depositando " + carta + " en la furgoneta 1";
                salidaEmpleado.setText(salida);
                sleep(TiempoDejarCarta);
                salida = nombre + " volviendo de la furgoneta 1";
                salidaEmpleado.setText(salida);
                furgoneta1.VueltaAlBuzon(nombre);
            } else {
                salida = nombre + " coge la carta " + carta + " y va a depositarla en la
furgoneta 2";
                salidaEmpleado.setText(salida);
                furgoneta2.DepositarFurgo(carta, nombre);
                salida = nombre + " depositando " + carta + " en la furgoneta 2";
                salidaEmpleado.setText(salida);
                logger.log(nombre + " coge la carta " + carta + " y va a depositarla en la
furgoneta 2");
                sleep(TiempoDejarCarta);
                salida = nombre + " volviendo de la furgoneta 2";
                salidaEmpleado.setText(salida);
                furgoneta2.VueltaAlBuzon(nombre);
            }
        }
        if (pausa.estaPausado()) {
            salidaEmpleado.setText(" Esta pausado");
        }
    }
}
```



```
    }  
    if (buzon.getCartasBuzon() == 0) {  
        salidaEmpleado.setText(nombre + " no tiene cartas que repartir");  
    }  
} catch (InterruptedException ex) {  
}  
}  
}  
}
```

Clase Furgoneta

```
packageCodigo;  
import java.util.ArrayList;  
import java.util.concurrent.Semaphore;  
import javax.swing.JOptionPane;  
public class Furgoneta {  
    private final ArrayList<String> furgo;  
    private final String nombreFurgo;  
    private final Semaphore ExclusionMutua = new Semaphore(1);  
    private final Semaphore lleno = new Semaphore(1);  
    private final LoggerUtils logger;  
    private final JOptionPane salidaFurgo;  
    public Furgoneta(String nombre, LoggerUtils logger, JOptionPane panel) {  
        this.nombreFurgo = nombre;  
        this.furgo = new ArrayList<>();  
        this.logger = logger;  
        this.salidaFurgo = panel;  
    }  
    public void DepositarFurgo(String carta, String nombre) throws InterruptedException  
    {  
        try {  
            lleno.acquire();  
            ExclusionMutua.acquire();  
            furgo.add(carta);  
            salidaFurgo.setText("" + furgo);  
            logger.log("La carta/" + carta + "/" ha sido depositada por /" + nombre + "/" en " +  
nombreFurgo);  
        } finally {  
            ExclusionMutua.release();  
        }  
    }  
    public void VueltaAlBuzon(String nombre) throws InterruptedException {
```

```
try {
    ExclusionMutua.acquire();
    logger.log(nombre + " ha vuelto de la - " + nombreFurgo);
} finally {
    ExclusionMutua.release();
    lleno.release();
}
}
```

Clase InterfaceDistribuida

```
packageCodigo;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface InterfaceDistribuida extends Remote {
    String getBuzon() throws RemoteException;
    String getfurgo1() throws RemoteException;
    String getfurgo2() throws RemoteException;
    String getCartasBuzon() throws RemoteException;
}
```

Clase LoggersUtils

```
packageCodigo;
import java.io.File;
import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
public class LoggerUtils {
    private Logger logger;
    private FileHandler fileh;
    private File archivo ;
    private String logName = "evolucionCartas";
    public LoggerUtils(){
        logger = Logger.getLogger(logName);
        archivo = new File("src\\main\\java\\logs\\"+logName+".txt");
        if(archivo.exists()){
            try {
                System.out.println("Entra");
                fileh = new FileHandler("src\\main\\java\\logs\\"+logName+".txt");
                logger.addHandler(fileh);
                SimpleFormatter formato = new SimpleFormatter();
            }
        }
    }
}
```

```
        fileh.setFormatter(formato);
    } catch (IOException | SecurityException ex) {
        System.out.println(ex);
    }
} else {
    try {
        boolean result = archivo.createNewFile();
        fileh = new FileHandler("src\\main\\java\\logs\\"+logName+".txt");
        logger.addHandler(fileh);
        SimpleFormatter formato = new SimpleFormatter();
        fileh.setFormatter(formato);
    } catch (IOException | SecurityException ex) {
        System.out.println(ex);
    }
}
}
public void log(String mensaje){
    logger.info(mensaje);
}
}
```

Clase Objeto Remoto

```
packageCodigo;
import InterfazGrafica.CorreosSL;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class ObjetoRemoto extends UnicastRemoteObject implements
InterfaceDistribuida{
    private final CorreosSL interfaces;
    public ObjetoRemoto(CorreosSL inter) throws RemoteException{
        this.interfaces = inter;
    }
    @Override
    public String getBuzon() throws RemoteException {
        return interfaces.getBuzon();
    }
    @Override
    public String getfurgo1() throws RemoteException {
        return interfaces.getfurgo1();
    }
    @Override
    public String getfurgo2() throws RemoteException {
```

```
        return interfaces.getfurgo2();
    }
    @Override
    public String getCartasBuzon() throws RemoteException {
        return interfaces.getCartasBuzon();
    }
}
```

Clase Pausar

```
packageCodigo;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class Pausar {
    private boolean cerrado = false;
    private final Lock cerrojo = new ReentrantLock();
    private final Condition parar = cerrojo.newCondition();
    public void mirar() {
        try {
            cerrojo.lock();
            while (cerrado) {
                try {
                    parar.await();
                } catch (InterruptedException ie) {
                }
            }
        } finally {
            cerrojo.unlock();
        }
    }
    public void abrir() {
        try {
            cerrojo.lock();
            cerrado = false;
            parar.signalAll();
        } finally {
            cerrojo.unlock();
        }
    }
    public void cerrar() {
        try {
            cerrojo.lock();
```

```
        cerrado = true;
    } finally {
        cerrojo.unlock();
    }
}
public boolean estaPausado() {
    return this.cerrado;
}
}
```

Clase Persona

```
packageCodigo;
public class Persona extends Thread {
    private final String nombre;
    private final Buzon buzon;
    private final LoggerUtils logger;
    public Persona(String nombre, Buzon buzon, LoggerUtils logger) {
        this.nombre = nombre;
        this.buzon = buzon;
        this.logger = logger;
    }
    public void run() {
        long TiempoDejarCarta = 0;
        int i = 3;
        for (int x = 1; x < i; x++) {
            try {
                sleep((long) 1000 + (int) ((25000 - 1000) * Math.random()));
                TiempoDejarCarta = (long) 400 + (int) ((800 - 400) * Math.random());
                String carta = nombre + "-C" + x;
                buzon.dejarCarta(carta);
                logger.log(nombre + " deja la carta " + carta + " en el buzon.");
                sleep(TiempoDejarCarta);
            } catch (InterruptedException ex) {
            }
        }
    }
}
```

Clase CorreosSL

```
package InterfazGrafica;
importCodigo.Buzon;
importCodigo.ObjetoRemoto;
```

```
import Codigo.Empleado;
import Codigo.Furgoneta;
import Codigo.LoggerUtils;
import Codigo.Pausar;
import Codigo.Persona;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CorreosSL extends javax.swing.JFrame {
    private boolean Pausa = true;
    private final Pausar TOTALPAUSE = new Pausar();
    private final Pausar pause1 = new Pausar();
    private final Pausar pause2 = new Pausar();
    private final Pausar pause3 = new Pausar();
    private final Pausar pause4 = new Pausar();
    private final Pausar pause5 = new Pausar();
    public CorreosSL() {
        initComponents();
        LoggerUtils logger = new LoggerUtils();
        int NunPersona = 401;
        Buzon buzon = new Buzon(30, jTextPane1, jTextPane2);
        Furgoneta furgoneta1 = new Furgoneta("furgoneta 1", logger, jTextPane3);
        Furgoneta furgoneta2 = new Furgoneta("furgoneta 2", logger, jTextPane4);
        Empleado E1 = new Empleado("Empleado1", buzon, furgoneta1, furgoneta2,
logger, jTextPane5, pause1);
        Empleado E2 = new Empleado("Empleado2", buzon, furgoneta1, furgoneta2,
logger, jTextPane6, pause2);
        Empleado E3 = new Empleado("Empleado3", buzon, furgoneta1, furgoneta2,
logger, jTextPane7, pause3);
        Empleado E4 = new Empleado("Empleado4", buzon, furgoneta1, furgoneta2,
logger, jTextPane8, pause4);
        Empleado E5 = new Empleado("Empleado5", buzon, furgoneta1, furgoneta2,
logger, jTextPane9, pause5);
        E1.start();
        E2.start();
        E3.start();
        E4.start();
        E5.start();
        for (int i = 1; i < NunPersona; i++) {
```

```
String nombre;
nombre = "Persona" + i;
Persona p = new Persona(nombre, buzón, logger);
p.start();
}
try {
    for (javax.swing.UIManager.LookAndFeelInfo info :
        javax.swing.UIManager.getInstalledLookAndFeels()) {
        if ("Nimbus".equals(info.getName())) {
            javax.swing.UIManager.setLookAndFeel(info.getClassName());
            break;
        }
    }
} catch (ClassNotFoundException | InstantiationException | IllegalAccessException
    | javax.swing.UnsupportedLookAndFeelException ex) {
    java.util.logging.Logger.getLogger(CorreosSL.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
}
}
public String getBuzon() {
    String str = jTextPane1.getText();
    return str;
}
public String getfurgo1() {
    String str = jTextPane3.getText();
    return str;
}
public String getfurgo2() {
    String str = jTextPane4.getText();
    return str;
}
public String getCartasBuzon() {
    String str = jTextPane2.getText();
    return str;
}
}
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {construcción del display gráfico }
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    if (Pausa == true) {
        TOTALPAUSE.cerrar();
        jButton1.setText("Reanudar TODO");
        Pausa = false;
    }
}
```



```
        if (!jToggleButton1.isSelected()) {
            jToggleButton1.doClick();
        }
        if (!jToggleButton2.isSelected()) {
            jToggleButton2.doClick();
        }
        if (!jToggleButton3.isSelected()) {
            jToggleButton3.doClick();
        }
        if (!jToggleButton4.isSelected()) {
            jToggleButton4.doClick();
        }
        if (!jToggleButton5.isSelected()) {
            jToggleButton5.doClick();
        }
    } else {
        TOTALPAUSE.abrir();
        jButton1.setText("Parar TODO");
        Pausa = true;
        if (jToggleButton1.isSelected()) {
            jToggleButton1.doClick();
        }
        if (jToggleButton2.isSelected()) {
            jToggleButton2.doClick();
        }
        if (jToggleButton3.isSelected()) {
            jToggleButton3.doClick();
        }
        if (jToggleButton4.isSelected()) {
            jToggleButton4.doClick();
        }
        if (jToggleButton5.isSelected()) {
            jToggleButton5.doClick();
        }
    }
}

private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jToggleButton1.isSelected()) {
        pause1.cerrar();
        jToggleButton1.setText("Reanudar empleado 1");
    } else {
        pause1.abrir();
    }
}
```



```
jToggleButton1.setText("Parar empleado 1");
}
}
private void jToggleButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jToggleButton4.isSelected()) {
        pause4.cerrar();
        jToggleButton4.setText("Reanudar empleado 4");
    } else {
        pause4.abrir();
        jToggleButton4.setText("Parar empleado 4");
    }
}
private void jToggleButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jToggleButton2.isSelected()) {
        pause2.cerrar();
        jToggleButton2.setText("Reanudar empleado 2");
    } else {
        pause2.abrir();
        jToggleButton2.setText("Parar empleado 2");
    }
}
private void jToggleButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jToggleButton3.isSelected()) {
        pause3.cerrar();
        jToggleButton3.setText("Reanudar empleado 3");
    } else {
        pause3.abrir();
        jToggleButton3.setText("Parar empleado 3");
    }
}
private void jToggleButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (jToggleButton5.isSelected()) {
        pause5.cerrar();
        jToggleButton5.setText("Reanudar empleado 5");
    } else {
        pause5.abrir();
        jToggleButton5.setText("Parar empleado 5");
    }
}
```

```
public static void main(String args[]) throws RemoteException,
MalformedURLException {
    CorreosSL interfaces = new CorreosSL();
    ObjetoRemoto correos = new ObjetoRemoto(interfaces);
    Registry registry = LocateRegistry.createRegistry(1099); //rmiregistry en puerto
1099
    Naming.rebind("//localhost/AccesoServidor", (Remote) correos); //funciona en
equipo local
    interfaces.setVisible(true);
}
// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JScrollPane jScrollPane6;
private javax.swing.JScrollPane jScrollPane7;
private javax.swing.JScrollPane jScrollPane8;
private javax.swing.JScrollPane jScrollPane9;
private javax.swing.JTextPane jTextPane1;
private javax.swing.JTextPane jTextPane2;
private javax.swing.JTextPane jTextPane3;
private javax.swing.JTextPane jTextPane4;
private javax.swing.JTextPane jTextPane5;
private javax.swing.JTextPane jTextPane6;
private javax.swing.JTextPane jTextPane7;
private javax.swing.JTextPane jTextPane8;
private javax.swing.JTextPane jTextPane9;
private javax.swing.JToggleButton jToggleButton1;
private javax.swing.JToggleButton jToggleButton2;
private javax.swing.JToggleButton jToggleButton3;
```

```
private javax.swing.JToggleButton jToggleButton4;  
private javax.swing.JToggleButton jToggleButton5;  
// End of variables declaration  
}
```

Clase CorreosSLCliente

```
package InterfazGrafica;  
import Codigo.InterfaceDistribuida;  
import static java.lang.Thread.sleep;  
import java.net.MalformedURLException;  
import java.rmi.Naming;  
import java.rmi.NotBoundException;  
import java.rmi.RemoteException;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
public class CorreosSLCliente extends javax.swing.JFrame {  
    private InterfaceDistribuida correos;  
    public CorreosSLCliente() {  
        initComponents();  
        try {  
            correos = (InterfaceDistribuida) Naming.lookup("//127.0.0.1/AccesoServidor");  
        } catch (MalformedURLException | NotBoundException | RemoteException e) {  
            System.out.println("Excepción : " + e.getMessage());  
        }  
    }  
    public void Mostrar() throws InterruptedException {  
        try {  
            JTextPane2.setText(correos.getCartasBuzon()); // seteamos las cartas que hay en  
            el buzón  
            JTextPane1.setText(correos.getBuzon()); // contenido del buzón  
            JTextPane3.setText(correos.getfurgo1()); // contenido de la furgoneta1  
            JTextPane4.setText(correos.getfurgo2()); // contenido de la furgoneta2  
            sleep(100);  
        } catch (RemoteException ex) {  
            Logger.getLogger(CorreosSLCliente.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
    @SuppressWarnings("unchecked")  
    // <editor-fold defaultstate="collapsed" desc="Generated Code">  
    private void initComponents() { construcción del display gráfico }
```

```
public static void main(String args[]) throws InterruptedException,
InterruptedException {
    CorreosSLCliente inter = new CorreosSLCliente();
    inter.setVisible(true);
    while(true){
        inter.Mostrar();
    }
}
// Variables declaration - do not modify
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JTextPane jTextPane1;
private javax.swing.JTextPane jTextPane2;
private javax.swing.JTextPane jTextPane3;
private javax.swing.JTextPane jTextPane4;
// End of variables declaration
}
```