Simulación del Funcionamiento de un Sistema de Cajeros

Universidad de Alcalá
Grado en Ingeniería en Sistemas de Información (GISI)
Paradigmas de Programación

Curso 2023/2024 – Convocatoria Ordinaria

Héctor Benavente García
DNI: 09105469E
Universidad de Alcalá
Madrid, España
hector.benavente@edu.uah.es

José Sánchez Nicolás
DNI: 09121320A
Universidad de Alcalá
Madrid, España
jose.sanchezn@edu.uah.es

ÍNDICE

ANÁLISIS DE ALTO NIVEL

El objetivo de la presente práctica es simular el funcionamiento de un sistema de cajeros de forma eficiente y realista mediante el uso de hilos y técnicas de programación concurrente y distribuida.

El sistema cuenta con 200 clientes que realizan operaciones (ingresar o extraer dinero) en 4 cajeros. Además, se incorporan 2 operarios responsables de mantener el correcto funcionamiento de los cajeros a través de la retirada o depósito de dinero en los mismos cuando les sea requerido, gestionando las transacciones con el banco central. Las operaciones de ingreso y extracción son determinadas por los clientes de manera aleatoria. Dichos clientes deben esperar en una cola para acceder a un cajero libre y poder realizar la operación.

RELACIÓN PERSONA-CAJERO

Se crean 200 clientes (*Persona*) que deciden aleatoriamente qué operación van a realizar (insertar o extraer) y de cuánto dinero será dicha operación (5.000€ a 10.000€). Una vez se determina el tipo y cuantía de la operación, se introducen de forma escalonada en una *Cola* común para acceder a los cajeros.

Tras colocarse en la *Cola* común, cada *Cajero* sacará de la *Cola* a una *Persona*, la cual realizará la operación previamente seleccionada. Estas operaciones serán mostradas en tiempo real en la interfaz gráfica (*ModuloVisualizacion*) y registradas en un fichero de log (*evolucionCajeros.txt*) que podrá ser visualizado mediante la pulsación del botón "*Abrir fichero Log*".

La operación de insertar dinero tardará un tiempo aleatorio entre 2 y 4 segundos, mientras que la operación de extraer dinero tardará un tiempo aleatorio de entre 2,5 y 4,5 segundos.

RELACIÓN CAJERO-OPERARIO

Cuando algún *Cajero* no pueda ingresar más dinero debido a que sobrepasaría su capacidad (100.000€) o no tenga dinero suficiente para realizar una extracción, se introducirá en una cola de *Solicitudes*.

Si hay *Solicitudes* en cola el *Operario* sacará de la cola una solicitud, comprobará si la solicitud es de retirada o depósito de dinero y realizará dicha operación siempre y cuando sea posible, ingresando o extrayendo dinero del *BancoCentral* para cumplir con las necesidades del *Cajero*.

Las operaciones realizadas serán mostradas en tiempo real en la interfaz gráfica (*ModuloVisualizacion*) y registradas en un fichero de log (*evolucionCajeros.txt*).

La operación de retirar dinero del cajero tardará un tiempo 2 segundos, mientras que la operación de depósito de dinero en el cajero tardará un tiempo de 3 segundos.

RELACIÓN OPERARIO-BANCO CENTRAL

Cuando el *Operario* recibe una solicitud, comprueba si esta es de retirada o depósito de dinero.

En el primer caso el *Operario* extrae el dinero del *Cajero* y lo deposita en el *BancoCentral,* pausando la ejecución del *Cajero* durante el tiempo que tarda en extraer el dinero de este.

En el segundo caso el *Operario* comprueba si hay dinero suficiente en el *BancoCentral* para atender la solicitud. Si hay el suficiente dinero lo extrae y lo deposita en el *Cajero* que lo solicitó. En caso de no haber suficiente dinero no retirará el dinero del BancoCentral y el *Cajero* quedará detenido.

PROGRAMACIÓN DISTRIBUIDA

En la clase principal (*ModuloVisualizacion*) se encuentra el código del servidor RMI. También se crea una interfaz remota (*InterfaceDistribuida*) y una clase que implementa la interfaz remota (*InterfaceServidor*). Por último, se crea una interfaz gráfica para el cliente (*ModuloVisualizacionCli*), a través de la cual se permitirá al cliente interactuar con el servidor.

Esta interfaz gráfica del cliente permite visualizar los datos del sistema bancario en tiempo real además de pausar y reanudar tanto los operarios como el programa.

DISEÑO GENERAL DEL SISTEMA Y HERRAMIENTAS DE SINCRONIZACIÓN

El diseño del sistema se basa en la programación concurrente, utilizando hilos para representar clientes, cajeros y operarios. Se emplean mecanismos de sincronización, como synchronized, locks y conditions, para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos, como la cola de clientes, los cajeros, la cola de solicitudes y el banco central.

CLASES PRINCIPALES Y DESCRIPCIÓN

INTERFACES GRÁFICAS

ModuloVisualizacion

Esta clase es la que se encarga de inicializar el programa junto con la interfaz gráfica del servidor RMI, registrando el objeto remoto que utilizará el cliente para visualizar los datos.

En la inicialización de dicha interfaz se crean e inicializan 200 clientes, la cola común de los clientes, 4 cajeros, la cola de las solicitudes de los cajeros a los operarios, 2 operarios y el banco central. Además, mostrará el estado de la cola de personas, las transacciones de los cajeros y operarios, el dinero actual de los cajeros y del banco central y los logs del sistema. *(Anexo VII)*

ModuloVisualizacionCli

Esta clase es la interfaz gráfica del cliente RMI y para poder inicializarla primero tenemos que inicializar el servidor (*ModuloVisualizacion*), de lo contrario no mostrará ninguna información.

Una vez inicializado el servidor, esta interfaz mostrará todos los datos que se visualizan en la interfaz del servidor a excepción de los logs. Además, permitirá pausar y reanudar la ejecución de cada uno de los operarios y la del programa. *(Anexo X)*

CLASES DEL PROGRAMA PRINCIPAL

Persona (hilo)

Esta clase modela el comportamiento de los clientes. Dichos clientes decidirán de forma aleatoria el tipo (insertar o extraer) y cuantía (5.000€ – 10.000€) de la operación. Después se colocan en la *Cola* a la espera de que haya un cajero libre.

Implementa la interfaz “Runnable” que permite ejecutar una tarea en paralelo usando la clase Thread. *(Anexo I)*

Cola

Esta clase estática modela el comportamiento de la cola de clientes. Esta cola es un array en el que se introducen personas gracias a su método “meter” y se extraen personas gracias a su método “sacar”.

Esta clase conecta la lógica de los clientes (*Persona*) con la de los cajeros (*Cajero*).

Utiliza locks y conditions para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos. *(Anexo II)*

Cajero (hilo)

Esta clase modela el comportamiento de los cajeros. Dichos cajeros extraen personas de la cola para que realicen la operación deseada. Para ello el cajero comprobará qué operación desea realizar la Persona y utilizará sus métodos privados “insertar” y “extraer” para realizar dicha operación.

La capacidad máxima de los cajeros es de 100.000€. Si no dispone de la capacidad suficiente para recibir un nuevo ingreso o no tiene de efectivo suficiente para efectuar una nueva retirada se meterá en la cola de solicitudes para que un operario atienda su petición.

Implementa la interfaz “Runnable” que permite ejecutar una tarea en paralelo usando la clase Thread.

Utiliza la palabra reservada synchronized para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos. *(Anexo III)*

Solicitudes

Esta clase estática modela el comportamiento de la cola de solicitudes de los cajeros. Como se ha comentado antes, los cajeros se introducen en dicha cola cuando necesitan retirar parte de su dinero o recibir dinero del banco central. Para ello dispone de un array en el que se introducirán los cajeros haciendo uso de su método “meter”. Posteriormente los operarios utilizarán su método “sacar” para coger los cajeros de la cola de solicitudes y atender a sus peticiones.

Esta clase conecta la lógica de los cajeros (*Cajero*) con la de los operarios (*Operario*).

Utiliza locks y conditions para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos. *(Anexo IV)*

Operario (hilo)

Esta clase modela el comportamiento de los operarios. Dichos operarios esperarán pasivamente a que un cajero se introduzca en la cola de solicitudes. Una vez el cajero ha realizado la solicitud (introduciéndose en la cola), el operario lo saca de la cola y comprueba si está solicitando una retirada o un depósito de dinero. Para realizar la operación que se le solicita utiliza sus métodos privados “retirar” y “depositar”.

Implementa la interfaz “Runnable” que permite ejecutar una tarea en paralelo usando la clase Thread.

Utiliza la palabra reservada synchronized para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos. *(Anexo V)*

BancoCentral

Esta clase estática modela el comportamiento del banco central. Incorpora los métodos “insertar” y “extraer” que utilizan los operarios para insertar dinero en el banco central cuando un cajero ha solicitado una retirada de dinero o extraer dinero del banco central cuando un cajero ha solicitado un deposito de dinero. El método extraer comprueba que haya dinero suficiente en el banco central para cumplimentar la solicitud de los cajeros.

Utiliza la palabra reservada synchronized para garantizar la exclusión mutua y coordinar el acceso a los recursos compartidos. *(Anexo VI)*

CLASES DEL PROGRAMA DISTRIBUIDO

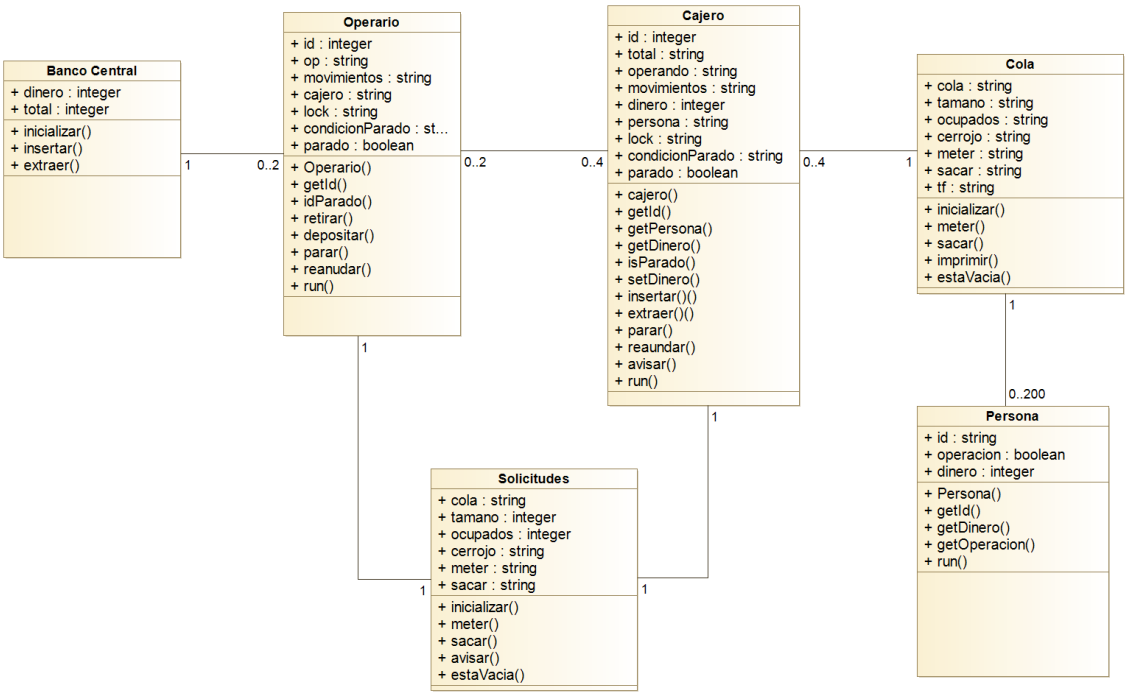
InterfaceDistribuida

Esta interfaz distribuida permite al programa cliente (ModuloVisualizacionCli) invocar métodos en el objeto remoto. *(Anexo IX)*

InterfaceServidor

Esta clase implementa la interfaz distribuida y proporciona la implementación concreta de los métodos definidos en dicha interfaz. *(Anexo VIII)*

DIAGRAMA DE CLASES



ANEXO: CÓDIGO FUENTE

El código fuente de los siguientes anexos incluye únicamente la parte relevante del código y prescinde de elementos como la importación de paquetes y librerías, la creación e inicialización de los componentes gráficos o los comentarios del código.

El código fuente original está incluido en el proyecto NetBeans que se adjunta con la entrega.

ANEXO I – Clase Persona

```java
public class Persona implements Runnable {

    private String id;
    private boolean operacion;
    private int dinero;

    public Persona(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public int getDinero() {
        return dinero;
    }

    public boolean getOperacion() {
        return operacion;
    }

    @Override
    public void run() {
        try {
            Random rand = new Random();
            Thread.sleep(rand.nextInt(58000) + 2000);
            operacion = rand.nextBoolean();
            dinero = (rand.nextInt(6) + 5) * 1000;

            Cola.meter(this);

        } catch (InterruptedException ex) {}
    }
}
```

ANEXO II – Clase Cola

```
public class Cola {

    private static Persona[] cola;
    private static int tamano;
    private static int ocupados;
    private static Lock cerrojo = new ReentrantLock();
    private static Condition meter;
    private static Condition sacar;
    private static JTextField tf;

    public static void inicializar(int size, JTextField textField) {
        tamano = size;
        cola = new Persona[tamano];
        ocupados = 0;
        tf = textField;
        meter = cerrojo.newCondition();
        sacar = cerrojo.newCondition();
    }

    public static void meter(Persona p) {
        try {
            cerrojo.lock();
            while (ocupados == tamano) {
                try {
                    meter.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
            cola[ocupados] = p;
            ocupados++;
            sacar.signal();
            imprimir();
        } finally {
            cerrojo.unlock();
        }
    }

    public static Persona sacar() {
        try {
            cerrojo.lock();
            while (ocupados == 0) {
                try {
                    sacar.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
```

```
            }
            Persona datoRecibido = cola[0];
            System.arraycopy(cola, 1, cola, 0, ocupados - 1);
            ocupados--;
            meter.signal();
            imprimir();
            return datoRecibido;
        } finally {
            cerrojo.unlock();
        }
    }

    private static void imprimir() {
        StringBuilder contenido = new StringBuilder();

        for (int i = 0; i < ocupados; i++) {
            contenido.append("  [ ").append(cola[i].getId()).append(" ]     ");
        }

        tf.setText(contenido.toString());
    }

    public static boolean estaVacia() {
        return ocupados == 0;
    }
}
```

ANEXO III – Clase Cajero

```
public class Cajero implements Runnable {

    private int id;
    private JTextField total;
    private JTextField operando;
    private JTextArea movimientos;
    private int dinero;
    private Persona persona;
    private final Lock lock = new ReentrantLock();
    private final Condition condicionParado = lock.newCondition();
    private boolean parado = false;

    public Cajero(int id, JTextField total, JTextField operando, JTextArea movimientos) {
        this.id = id;
        this.total = total;
        this.operando = operando;
        this.movimientos = movimientos;
        dinero = 50000;
    }

    public int getId() {
        return id;
    }

    public Persona getPersona() {
        return persona;
    }

    public int getDinero() {
        return dinero;
    }

    public boolean isParado() {
        return parado;
    }

    public void setDinero(int dinero) {
        this.dinero = dinero;
    }

    public synchronized void insertar(int cantidad) {
        try {
            operando.setText(persona.getId() + "-I+" + persona.getDinero());
            movimientos.append(persona.getId() + "-I+" + persona.getDinero() + "\n");
            FileWriter fileWriter = new FileWriter("evolucionCajeros.txt", true);

            Date fechaActual = new Date();
```

```
            SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            String fecha = formatoFecha.format(fechaActual);

            String mensajeLog = "[" + fecha + "] - " + persona.getId() + "-I+" +
persona.getDinero() + " en el Cajero " + id + "\n";

            try (BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {
                bufferedWriter.write(mensajeLog);
            }

            Random rand = new Random();
            Thread.sleep(rand.nextInt(2000) + 2000);
            dinero += cantidad;
            if (dinero >= 100000) {
                Solicitudes.meter(this);
                while (dinero >= 100000) {
                    this.wait();
                }
            }
            total.setText(String.valueOf(dinero));
        } catch (IOException | InterruptedException ex) {
            Logger.getLogger(Cajero.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public synchronized void extraer(int cantidad) {
        try {
            operando.setText(persona.getId() + "-E-" + persona.getDinero());
            movimientos.append(persona.getId() + "-E-" + persona.getDinero() + "\n");
            FileWriter fileWriter = new FileWriter("evolucionCajeros.txt", true);

            Date fechaActual = new Date();
            SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            String fecha = formatoFecha.format(fechaActual);

            String mensajeLog = "[" + fecha + "] - " + persona.getId() + "-E-" +
persona.getDinero() + " en el Cajero " + id + "\n";

            try (BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {
                bufferedWriter.write(mensajeLog);
            }

            Random rand = new Random();
            Thread.sleep(rand.nextInt(2000) + 2500);
            dinero -= cantidad;
            if (dinero <= 0) {
                Solicitudes.meter(this);
                while (dinero <= 0) {
                    this.wait();
```

```
                }
            }
            total.setText(String.valueOf(dinero));
        } catch (IOException | InterruptedException ex) {
            Logger.getLogger(Cajero.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void parar() throws InterruptedException {
        lock.lock();
        try {
            parado = true;
        } finally {
            lock.unlock();
        }
    }

    public void reanudar() {
        lock.lock();
        try {
            parado = false;
            condicionParado.signal();
        } finally {
            lock.unlock();
        }
    }

    public synchronized void avisar() {
        this.notify();
    }

    @Override
    public void run() {
        try {
            total.setText(String.valueOf(dinero));
            Thread.sleep(5000);
            while (!Cola.estaVacia()) {
                lock.lock();
                try {
                    if (parado) {
                        condicionParado.await();
                    }
                } finally {
                    lock.unlock();
                }

                persona = Cola.sacar();
                if (persona != null) {
                    if (persona.getOperacion()) {
```

```
                        insertar(persona.getDinero());
                    } else {
                        extraer(persona.getDinero());
                    }
                }
            }
        } catch (InterruptedException e) {}
    }
}
```

ANEXIO IV – Clase Solicitudes

```
public class Solicitudes {

    private static Cajero[] cola;
    private static int tamano;
    private static int ocupados;
    private static Lock cerrojo = new ReentrantLock();
    private static Condition meter;
    private static Condition sacar;

    public static void inicializar(int size) {
        tamano = size;
        cola = new Cajero[tamano];
        ocupados = 0;
        meter = cerrojo.newCondition();
        sacar = cerrojo.newCondition();
    }

    public static void meter(Cajero p) {
        try {
            cerrojo.lock();
            while (ocupados == tamano) {
                try {
                    meter.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
            cola[ocupados] = p;
            ocupados++;
            sacar.signal();
        } finally {
            cerrojo.unlock();
        }
    }
```

```
    public static Cajero sacar(Operario operario) {
        try {
            cerrojo.lock();
            while (ocupados == 0 || operario.isParado()) {
                try {
                    sacar.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
            Cajero datoRecibido = cola[0];
            System.arraycopy(cola, 1, cola, 0, ocupados - 1);
            ocupados--;
            meter.signal();
            return datoRecibido;
        } finally {
            cerrojo.unlock();
        }
    }

    public static void avisar() {
        try {
            cerrojo.lock();
            sacar.signal();
        } finally {
            cerrojo.unlock();
        }
    }

    public static boolean estaVacia() {
        return ocupados == 0;
    }
}
```

ANEXO V – Clase Operario

```
public class Operario implements Runnable {

    private int id;
    private JTextField op;
    private JTextArea movimientos;
    private Cajero cajero;
    private final Lock lock = new ReentrantLock();
    private final Condition condicionParado = lock.newCondition();
    private boolean parado = false;

    public Operario(int id, JTextField op, JTextArea movimientos) {
        this.id = id;
        this.op = op;
        this.movimientos = movimientos;
        parado = false;
    }

    public int getId() {
        return id;
    }

    public boolean isParado() {
        return parado;
    }

    public synchronized void retirar(int cantidad) throws InterruptedException, IOException {
        FileWriter fileWriter = new FileWriter("evolucionCajeros.txt", true);

        Date fechaActual = new Date();

        SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        String fecha = formatoFecha.format(fechaActual);

        String mensajeLog = "[" + fecha + "] - " + "Operario" + id + "-C" + cajero.getId() + "+" +
cantidad + "\n";

        try (BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {
            bufferedWriter.write(mensajeLog);
        }

        Thread.sleep(2000);
        int dinero = cajero.getDinero();
        cajero.setDinero(dinero - cantidad);
        cajero.avisar();
        BancoCentral.insertar(cantidad);
        movimientos.append("Operario" + id + "-C" + cajero.getId() + "+50.000" + "\n");
    }
```

```
    public synchronized void depositar(int cantidad) throws InterruptedException, IOException {
        FileWriter fileWriter = new FileWriter("evolucionCajeros.txt", true);

        Date fechaActual = new Date();

        SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        String fecha = formatoFecha.format(fechaActual);

        String mensajeLog = "[" + fecha + "] - " + "Operario" + id + "-C" + cajero.getId() + "-" +
cantidad + "\n";

        try (BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)) {
            bufferedWriter.write(mensajeLog);
        }

        Thread.sleep(3000);
        int cant = BancoCentral.extraer(cantidad);
        int dinero = cajero.getDinero();
        cajero.setDinero(dinero + cant);
        movimientos.append("Operario" + id + "-C" + cajero.getId() + "-50.000" + "\n");
        cajero.avisar();
    }

    public void parar() throws InterruptedException {
        lock.lock();
        try {
            parado = true;
        } finally {
            lock.unlock();
        }
    }

    public void reanudar() {
        lock.lock();
        try {
            parado = false;
            condicionParado.signal();
            Solicitudes.avisar();
        } finally {
            lock.unlock();
        }
    }

    @Override
    public void run() {
        while (true) {
            try {
                lock.lock();
```

```
                while (parado) {
                    condicionParado.await();
                }
            } catch (InterruptedException ex) {
                Logger.getLogger(Operario.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                lock.unlock();
            }
                cajero = Solicitudes.sacar(this);
                if (cajero.getPersona().getOperacion()) {
                try {
                    op.setText("Llevando 50.000 del Cajero" + cajero.getId() + " al Banco
Central");
                    retirar(50000);
                    op.setText("");
                } catch (InterruptedException | IOException ex) {
                    Logger.getLogger(Operario.class.getName()).log(Level.SEVERE, null, ex);
                }
                } else {
                try {
                    op.setText("Trayendo 50.000 del Banco Central al Cajero" + cajero.getId());
                    depositar(50000);
                    op.setText("");
                } catch (InterruptedException | IOException ex) {
                    Logger.getLogger(Operario.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
```

ANEXO VI – Clase BancoCentral

```
public class BancoCentral {

    private static int dinero;
    private static JTextField total;

    public static void inicializar(JTextField total) {
        dinero = 250000;
        BancoCentral.total = total;
        total.setText(String.valueOf(dinero));
    }

    public static synchronized void insertar(int cantidad) {
        dinero += cantidad;
        total.setText(String.valueOf(dinero));
    }

    public static synchronized int extraer(int cantidad) {
        if (dinero < cantidad) {
            cantidad = 0;
        } else {
            dinero -= cantidad;
        }
        total.setText(String.valueOf(dinero));
        return cantidad;
    }
}
```

ANEXO VII – Clase ModuloVisualizacion

```
public class ModuloVisualizacion extends javax.swing.JFrame {

    private FileWriter fileWriter;
    private PrintWriter pw;

    private BancoCentral bancoCentral;
    private Cajero[] cajeros;
    private Operario[] operarios;

    public ModuloVisualizacion() {
        initComponents();
        setLocationRelativeTo(null);

        try {
            fileWriter = new FileWriter("evolucionCajeros.txt", true);
            pw = new PrintWriter(fileWriter);
        } catch (IOException e) {}

        Cola.inicializar(10, JTextCola);
        Solicitudes.inicializar(4);
        BancoCentral.inicializar(total5);

        inicializarPersonas();
        inicializarCajeros();
        inicializarOperarios();
    }

    private void inicializarPersonas() {
        for (int i = 1; i <= 200; i++) {
            Persona persona = new Persona("Persona" + i);
            Thread hiloPersona = new Thread(persona);
            hiloPersona.start();
        }
    }

    private void inicializarCajeros() {
        cajeros = new Cajero[4];

        cajeros[0] = new Cajero(1, total1, operando1, movimientos1);
        cajeros[1] = new Cajero(2, total2, operando2, movimientos2);
        cajeros[2] = new Cajero(3, total3, operando3, movimientos3);
        cajeros[3] = new Cajero(4, total4, operando4, movimientos4);

        for (Cajero cajero : cajeros) {
            Thread hiloCajero = new Thread(cajero);
            hiloCajero.start();
        }
```

```
    }

    private void inicializarOperarios() {
        operarios = new Operario[2];

        operarios[0] = new Operario(1, operario1, movimientos5);
        operarios[1] = new Operario(2, operario2, movimientos5);

        for (Operario operario : operarios) {
            Thread hiloOperario = new Thread(operario);
            hiloOperario.start();
        }
    }

    public String[] getTotales(){
        String[] totales = {total1.getText(), total2.getText(), total3.getText(),
total4.getText(), total5.getText()};
        return totales;
    }

    public String[] getOperandos(){
        String[] operandos = {operando1.getText(), operando2.getText(), operando3.getText(),
operando4.getText()};
        return operandos;
    }

    public String[] getOperarios(){
        String[] op = {operario1.getText(), operario2.getText()};
        return op;
    }

    public String[] getBotones(){
        String[] botones = {pausarOp1.getText(), pausarOp2.getText(), pausar.getText()};
        return botones;
    }

    public void pausarOp1(java.awt.event.ActionEvent evt){
        pausarOp1ActionPerformed(evt);
    }

    public void pausarOp2(java.awt.event.ActionEvent evt){
        pausarOp2ActionPerformed(evt);
    }

    public void pausar(java.awt.event.ActionEvent evt){
        pausarActionPerformed(evt);
    }

    private void pausarOp1ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
        if(operarios[0].isParado()){
            pausarOp1.setText("Pausar Operario");
            operarios[0].reanudar();
            Solicitudes.avisar();
        }else{
            try {
                pausarOp1.setText("Reanudar Operario");
                operarios[0].parar();
            } catch (InterruptedException ex) {
                Logger.getLogger(ModuloVisualizacion.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    private void pausarOp2ActionPerformed(java.awt.event.ActionEvent evt) {
        if(operarios[1].isParado()){
            pausarOp2.setText("Pausar Operario");
            operarios[1].reanudar();
            Solicitudes.avisar();
        }else{
            try {
                pausarOp2.setText("Reanudar Operario");
                operarios[1].parar();
            } catch (InterruptedException ex) {
                Logger.getLogger(ModuloVisualizacion.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    private void pausarActionPerformed(java.awt.event.ActionEvent evt) {
        String texto = null;

        if("Pausar".equals(pausar.getText())){
            pausar.setText("Reanudar");

            for(Operario operario : operarios){
                try {
                    texto = "Reanudar Operario";
                    operario.parar();
                } catch (InterruptedException ex) {
                    Logger.getLogger(ModuloVisualizacion.class.getName()).log(Level.SEVERE, null,
ex);
                }
            }

            for(Cajero cajero : cajeros){
                try {
                    cajeros[cajero.getId()-1].parar();
                } catch (InterruptedException ex) {
```

```
                    Logger.getLogger(ModuloVisualizacion.class.getName()).log(Level.SEVERE, null,
ex);
                }
            }
        }else{
            pausar.setText("Pausar");

            for(Operario operario : operarios){
                texto = "Pausar Operario";
                operarios[operario.getId()-1].reanudar();
                Solicitudes.avisar();
            }

            for(Cajero cajero : cajeros){
                cajeros[cajero.getId()-1].reanudar();
            }
        }

        pausarOp1.setText(texto);
        pausarOp2.setText(texto);
    }

    private void logActionPerformed(java.awt.event.ActionEvent evt) {
        String nombreArchivo = "evolucionCajeros.txt";

        try {
            File archivo = new File(nombreArchivo);

            if (archivo.exists()) {
                Desktop.getDesktop().open(archivo);
            } else {
                System.out.println("El archivo no existe.");
            }
        } catch (IOException ex) {
            System.out.println("Error al abrir el archivo: " + ex.getMessage());
        }
    }

    public static void main(String args[]) throws RemoteException, MalformedURLException {
        ModuloVisualizacion modulo = new ModuloVisualizacion();
        modulo.setVisible(true);
        modulo.setTitle("Servidor");

        InterfaceServidor interfaz = new InterfaceServidor(modulo);
        Registry registry = LocateRegistry.createRegistry(1099);
        Naming.rebind("//127.0.0.1/AccesoServidor", interfaz);
    }
}
```

ANEXO VIII – Clase InterfaceServidor

```
public class InterfaceServidor extends UnicastRemoteObject implements InterfaceDistribuida{
    private final ModuloVisualizacion interfaces;

    public InterfaceServidor(ModuloVisualizacion modulo) throws RemoteException{
        this.interfaces = modulo;
    }

    @Override
    public String[] getTotales() throws RemoteException{
        return interfaces.getTotales();
    }

    @Override
    public String[] getOperandos() throws RemoteException{
        return interfaces.getOperandos();
    }

    @Override
    public String[] getOperarios() throws RemoteException{
        return interfaces.getOperarios();
    }

    @Override
    public String[] getBotones() throws RemoteException{
        return interfaces.getBotones();
    }

    @Override
    public void pausarOp1(java.awt.event.ActionEvent evt) throws RemoteException{
        interfaces.pausarOp1(evt);
    }

    @Override
    public void pausarOp2(java.awt.event.ActionEvent evt) throws RemoteException{
        interfaces.pausarOp2(evt);
    }

    @Override
    public void pausar(java.awt.event.ActionEvent evt) throws RemoteException{
        interfaces.pausar(evt);
    }
}
```

ANEXO IX – Interfaz InterfaceDistribuida

```java
public interface InterfaceDistribuida extends Remote {

    public String[] getTotales() throws RemoteException;

    public String[] getOperandos() throws RemoteException;

    public String[] getOperarios() throws RemoteException;

    public String[] getBotones() throws RemoteException;

    public void pausarOp1(java.awt.event.ActionEvent evt) throws RemoteException;

    public void pausarOp2(java.awt.event.ActionEvent evt) throws RemoteException;

    public void pausar(java.awt.event.ActionEvent evt) throws RemoteException;

}
```

ANEXO X – Clase ModuloVisualizacionCli

```java
public class ModuloVisualizacionCli extends javax.swing.JFrame {

    private InterfaceDistribuida sistemaBancario;

    public ModuloVisualizacionCli() {
        initComponents();
        try {
            sistemaBancario = (InterfaceDistribuida) Naming.lookup("//127.0.0.1/AccesoServidor");
        } catch (MalformedURLException | NotBoundException | RemoteException e) {
            System.out.println("Excepción : " + e.getMessage());
        }
    }

    public void Mostrar() throws InterruptedException, RemoteException {
        total1.setText(sistemaBancario.getTotales()[0]);
        total2.setText(sistemaBancario.getTotales()[1]);
        total3.setText(sistemaBancario.getTotales()[2]);
        total4.setText(sistemaBancario.getTotales()[3]);
        total5.setText(sistemaBancario.getTotales()[4]);

        operando1.setText(sistemaBancario.getOperandos()[0]);
        operando2.setText(sistemaBancario.getOperandos()[1]);
        operando3.setText(sistemaBancario.getOperandos()[2]);
        operando4.setText(sistemaBancario.getOperandos()[3]);
```

```
        operario1.setText(sistemaBancario.getOperarios()[0]);
        operario2.setText(sistemaBancario.getOperarios()[1]);

        pausarOp1.setText(sistemaBancario.getBotones()[0]);
        pausarOp2.setText(sistemaBancario.getBotones()[1]);
        pausar.setText(sistemaBancario.getBotones()[2]);

        sleep(500);
    }

    private void pausarOp1ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            sistemaBancario.pausarOp1(evt);
        } catch (RemoteException ex) {
            Logger.getLogger(ModuloVisualizacionCli.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void pausarOp2ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            sistemaBancario.pausarOp2(evt);
        } catch (RemoteException ex) {
            Logger.getLogger(ModuloVisualizacionCli.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void pausarActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            sistemaBancario.pausar(evt);
        } catch (RemoteException ex) {
            Logger.getLogger(ModuloVisualizacionCli.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String args[]) throws InterruptedException, RemoteException {
        ModuloVisualizacionCli moduloCli = new ModuloVisualizacionCli();
        moduloCli.setVisible(true);
        moduloCli.setTitle("Cliente");

        while(true){
            moduloCli.Mostrar();
        }
    }
}
```