
	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Septiembre 2020 – Febrero 2021

		FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES	
CARRERA: COMPUTACIÓN/INGENIERÍA DE SISTEMAS		ASIGNATURA: PROGRAMACIÓN APLICADA	
NRO. PROYECTO:	1.1	TÍTULO PROYECTO: Prueba Practica 2 Desarrollo e implementación de un sistema de simulación de acceso y atención bancaria	
OBJETIVO: Reforzar los conocimientos adquiridos en clase sobre la programación en Hilos en un contexto real.			
INSTRUCCIONES:		1. Revisar el contenido teórico y practico del tema	
		2. Profundizar los conocimientos revisando los libros guías, los enlaces contenidos en los objetos de aprendizaje Java y la documentación disponible en fuentes académicas en línea.	
		3. Deberá desarrollar un sistema informático para la simulación y una interfaz grafica.	
		4. Deberá generar un informe de la practica en formato PDF y en conjunto con el código se debe subir al GitHub personal y AVAC.	
		5. Fecha de entrega: El sistema debe ser subido al git hasta 17 de enero del 2021 – 23:55.	
ACTIVIDADES POR DESARROLLAR			

1. Enunciado:

Realizar un sistema de simulación de acceso y atención a través de colas de un banco.

Problema: Un banco necesita controlar el acceso a cuentas bancarias y para ello desea hacer un programa de prueba en Java que permita lanzar procesos que ingresen y retiren dinero a la vez y comprobar así si el resultado final es el esperado.

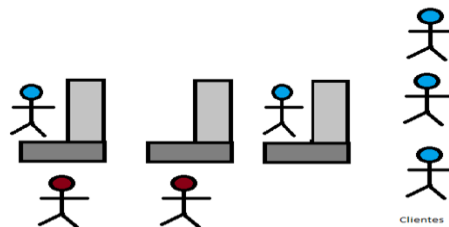
Se parte de una cuenta con 100 euros y se pueden tener procesos que ingresen 100 euros, 50 o 20. También se pueden tener procesos que retiran 100, 50 o 20 euros. Se desean tener los siguientes procesos:

- 40 procesos que ingresan 100
- 20 procesos que ingresan 50
- 60 que ingresen 20.

De la misma manera se desean lo siguientes procesos que retiran cantidades.

- 40 procesos que retiran 100
- 20 procesos que retiran 50
- 60 que retiran 20.


Ademas en el banco, existen 3 cajeros que pueden atender y hay un cola inicial de 10 clientes para ser atendidos, el proceso de atención es de 20 – 15 segundos y los clientes llegan constantemente cada 30 - 50 segundos. Ningún cajero puede atender simultáneamente, adicionalmente el tiempo de moverme de la cola al estante del cajero es de 2 - 5 segundos, esto deberán ser generados aleatoriamente entre los 100 clientes que disponen una cuenta, estos pueden volver a ingresar el numero de veces que sea necesario.



Se desea comprobar que tras la ejecución la cuenta tiene exactamente 100 euros, que era la cantidad de la que se disponía al principio. Realizar el programa Java que demuestra dicho hecho.

Calificación:

- Diagrama de Clase 10%
- MVC: 10%
- Técnicas de Programación aplicadas (Java 8, Reflexión y Programación Genérica): 10%
- Hilos 30%
- Sincronización 10%

	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Septiembre 2020 – Febrero 2021

- Interfaz Gráfica de simulación 20%
- Informe: 10%

2. Informe de Actividades:

- Planteamiento y descripción del problema.
- Diagramas de Clases.
- Patrón de diseño aplicado
- Descripción de la solución y pasos seguidos.
 - Comprobación de las cuentas bancarias e interfaz gráfica.
- Conclusiones y recomendaciones.
- Resultados.

RESULTADO(S) OBTENIDO(S):

- Interpreta de forma correcta los algoritmos de programación y su aplicabilidad.
- Identifica correctamente qué herramientas de programación se pueden aplicar.

CONCLUSIONES:

- Los estudiantes identifican las principales estructuras para la creación de sistemas informáticos.
- Los estudiantes implementan soluciones gráficas en sistemas.
- Los estudiantes están en la capacidad de implementar hilos.

RECOMENDACIONES:

- Revisar la información proporcionada por el docente previo a la práctica.
- Haber asistido a las sesiones de clase.
- **Consultar con el docente las dudas que puedan surgir al momento de realizar la prueba.**

BIBLIOGRAFIA:

[1]: <https://www.ups.edu.ec/evento?calendarBookingId=98892>

Docente / Técnico Docente: Ing. Diego Quisi Peralta Msc.

Firma: _____

CARRERA: COMPUTACIÓN/INGENIERÍA DE
SISTEMAS

ASIGNATURA: PROGRAMACIÓN APLICADA

NRO. PROYECTO:

1.1

TÍTULO PROYECTO: Prueba Practica 2

Desarrollo e implementación de un sistema de simulación de acceso y atención bancaria

OBJETIVO:

Reforzar los conocimientos adquiridos en clase sobre la programación en Hilos en un contexto real.

ACTIVIDADES DESARROLLADAS

1. Planteamiento y descripción del problema.

Enunciado:

Realizar un sistema de simulación de acceso y atención a través de colas de un banco.

Problema: Un banco necesita controlar el acceso a cuentas bancarias y para ello desea hacer un programa de prueba en Java que permita lanzar procesos que ingresen y retiren dinero a la vez y comprobar así si el resultado final es el esperado.

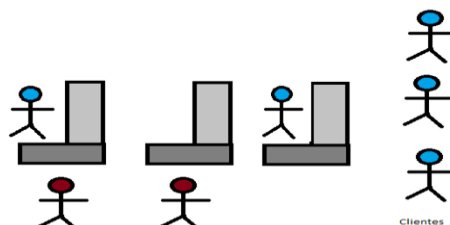
Se parte de una cuenta con 100 euros y se pueden tener procesos que ingresen 100 euros, 50 o 20. También se pueden tener procesos que retiran 100, 50 o 20 euros. Se desean tener los siguientes procesos:

- 40 procesos que ingresan 100
- 20 procesos que ingresan 50
- 60 que ingresen 20.

De la misma manera se desean lo siguientes procesos que retiran cantidades.

- 40 procesos que retiran 100
- 20 procesos que retiran 50
- 60 que retiran 20.

Ademas en el banco, existen 3 cajeros que pueden atender y hay un cola inicial de 10 clientes para ser atendidos, el proceso de atención es de 20 – 15 segundos y los clientes llegan constantemente cada 30 - 50 segundos. Ningún cajero puede atender simultáneamente, adicionalmente el tiempo de moverme de la cola al estante del cajero es de 2 - 5 segundos, esto deberán ser generados aleatoriamente entre los 100 clientes que disponen una cuenta, estos pueden volver a ingresar el numero de veces que sea necesario.



Se desea comprobar que tras la ejecución la cuenta tiene exactamente 100 euros, que era la cantidad de la que se disponía al principio. Realizar el programa Java que demuestra dicho hecho.

Calificación:

- Diagrama de Clase 10%
- MVC: 10%
- Técnicas de Programación aplicadas (Java 8, Reflexión y Programación Genérica): 10%
- Hilos 30%
- Sincronización 10%
- Interfaz Gráfica de simulación 20%
- Informe: 10%

2. Diagramas de Clases.

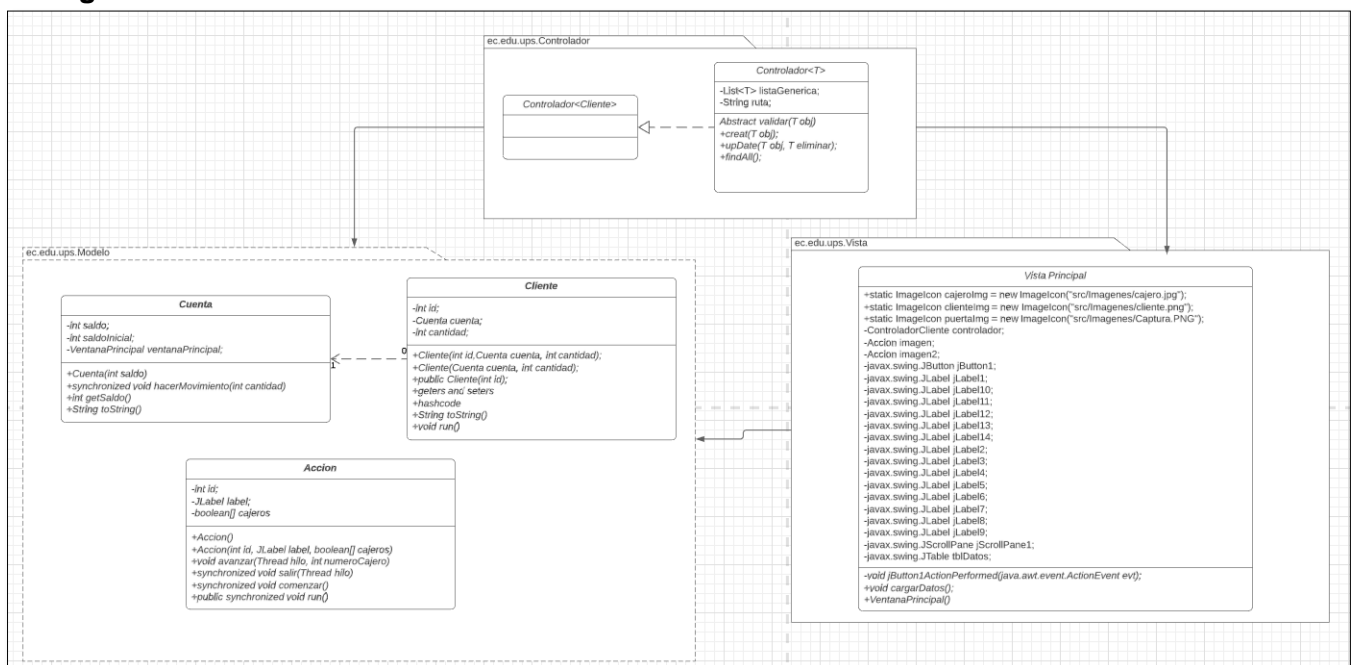
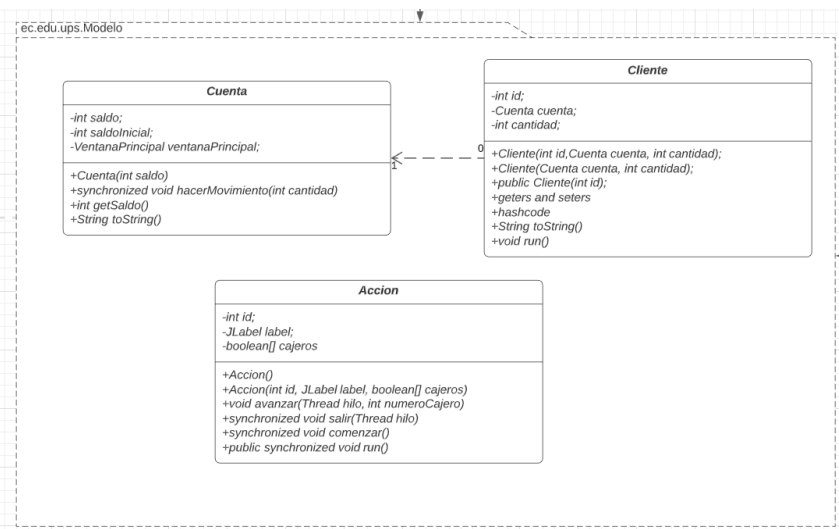
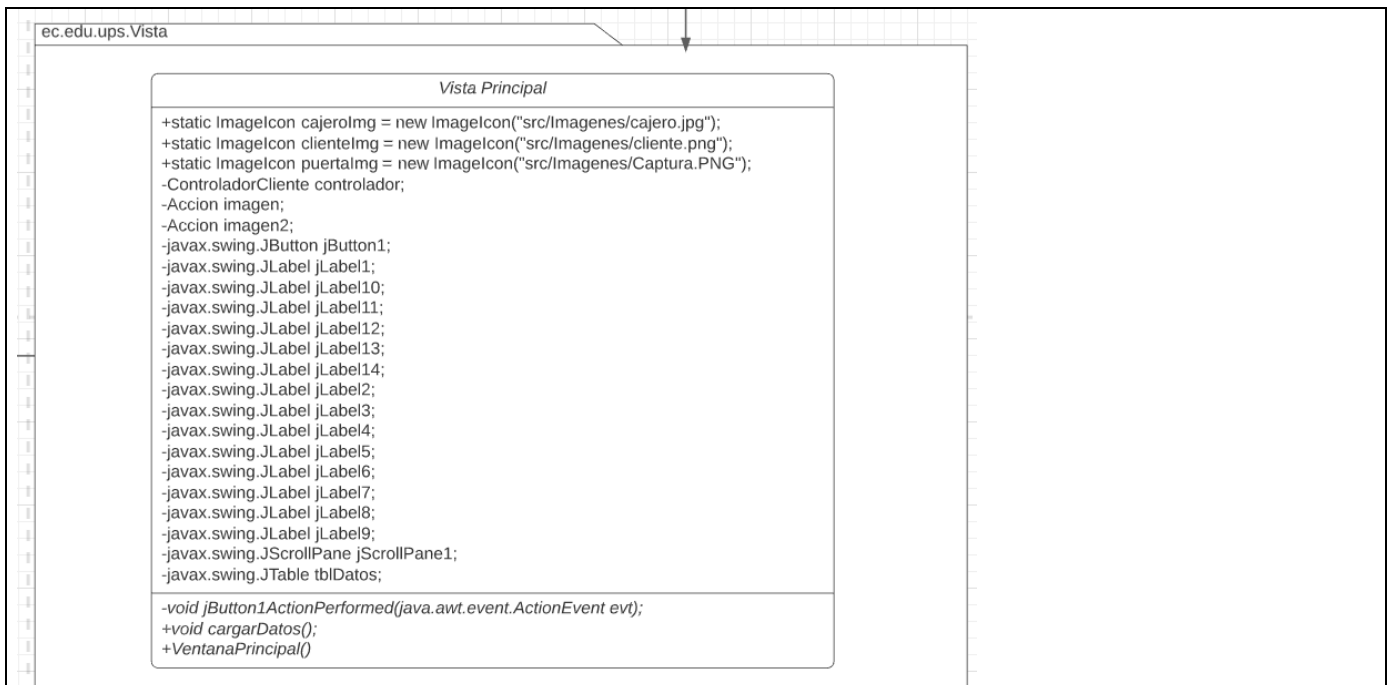


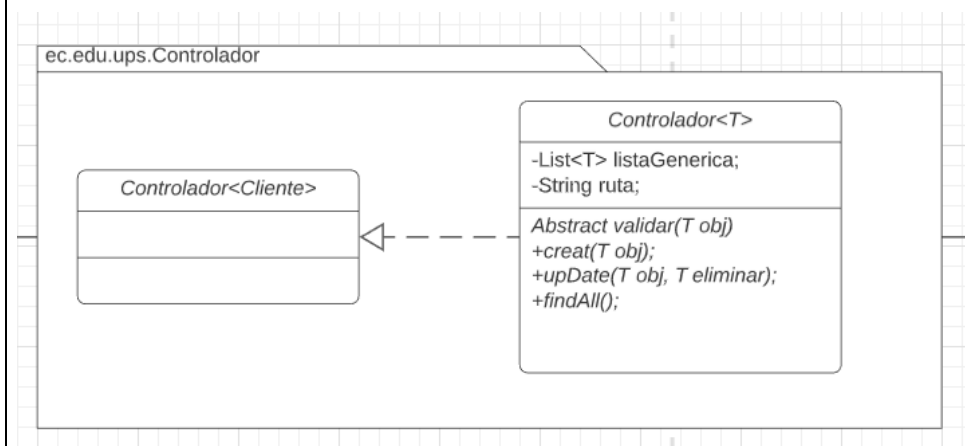
Diagrama de clase completo



Paquete modelo del diagrama de clase



Paquete vista del diagrama de clase

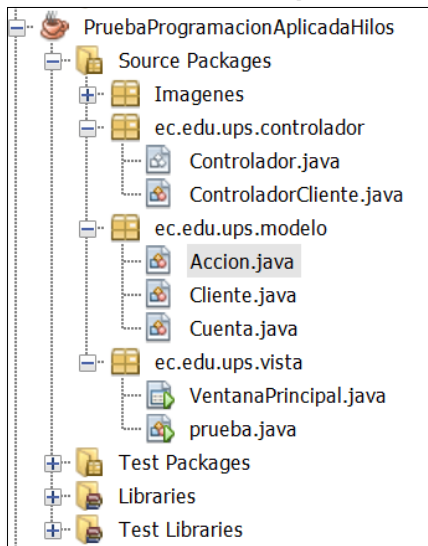


Paquete controlador del diagrama de clase

Link para ver el diagrama de una mejor manera.

https://lucid.app/lucidchart/dc31c9eb-6d73-4ac1-995c-7c9db3ea06dd/view?page=0_0#

3. Patrón de diseño aplicado



Se utiliza el patrón de diseño MVC para realizar el programa lo podemos verificar en la imagen donde se encuentra el nombre de los paquetes correspondientes al patrón de diseño

4. Descripción de la solución y pasos seguidos.

Técnicas de programación Aplicada

Programación genérica

- **Paquete Controlador**
- **Clase Controlador**

```
package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Cliente;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author NANCY
 */
public abstract class Controlador <T>{
    private List<T> listaGenerica;

    public Controlador() {
        listaGenerica = new ArrayList<>();
    }
}
```

```

public void create(T objeto) {
    listaGenerica.add(objeto);
}
public List<T> getListaGenerica() {
    return listaGenerica;
}
public void update(T objetoActualizado) {
    for (T objeto : listaGenerica) {
        if (objeto.equals(objetoActualizado)) {
            listaGenerica.set(listaGenerica.indexOf(objeto), objetoActualizado);
        }
    }
}
}

```

Código de la clase Controlador en donde podemos observar que es una clase abstracta para el uso de una programación genérica en donde verificamos el uso de esta programación con la sentencia al inicio de la clase <T> para que así pueda recibir cualquier objeto y accedan a los métodos de esta clase

- **Clase Controlador Cliente**

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Cliente;

/**
 *
 * @author NANCY
 */
public class ControladorCliente extends Controlador<Cliente>{
}

```

Código de la clase controlador cliente en donde podemos verificar el uso de la programación genérica debido a que esta clase hereda del controlador abstracto ya antes mencionado pasando como objeto un cliente que accede a los métodos del controlador

Con estas dos clases del paquete controlador vemos el uso de una programación genérica correctamente utilizada

Java 8

```

controlador.getListaGenerica().stream().map(c -> {
    if(c.getId()<40){

```



```

    Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 100);
    Thread hilosIngresan100= new Thread(ingresa);
    hilosIngresan100.start();
    controlador.update(ingresa);
    //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
    cargarDatos();
}
return c;
}).map(c -> {
    if(c.getId()>39 && c.getId()<60){
        Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 50);
        Thread hilosIngresan50= new Thread(ingresa);
        hilosIngresan50.start();
        controlador.update(ingresa);
        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
        cargarDatos();
    }
    return c;
}).map(c -> {
    if(c.getId()>59){
        Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 20);
        Thread hilosIngresan20= new Thread(ingresa);
        hilosIngresan20.start();
        controlador.update(ingresa);
        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
        cargarDatos();
    }
    return c;
}).filter(c -> (c.getId()>39 && c.getId()<60)).map(c -> new Cliente(c.getId(),c.getCuenta(), 20)).map(ingresa ->
new Thread(ingresa)).map(hilosIngresan20 -> {
    hilosIngresan20.start();
    return hilosIngresan20;
}).forEachOrdered(_item -> {
    //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
    cargarDatos();
});

```

Como podemos ver el código que es una parte del método del botón en la ventana principal se utiliza un stream map para recorrer una lista en este caso la lista del cliente y así ir haciendo las condiciones respectivas en el método filter.

Solución del Problema

Paquete Modelo

- **Clase Cuenta**

```
public class Cuenta {  
    private int saldo;  
    private int saldoInicial;  
    private VentanaPrincipal ventanaPrincipal;  
    public Cuenta(int saldo){  
        this.saldoInicial=saldo;  
        this.saldo=saldo;  
    }  
    public synchronized void hacerMovimiento(int cantidad){  
        this.saldo = this.saldo+cantidad;  
        //ventanaPrincipal.cargarDatos();  
    }  
    public boolean esSimulacionCorrecta(){  
        if (this.saldo==this.saldoInicial) return true;  
        return false;  
    }  
    public int getSaldo(){  
        return this.saldo;  
    }  
    @Override  
    public String toString() {  
        return "Cuenta{" + "saldo=" + saldo + ", saldoInicial=" + saldoInicial + '}';  
    }  
}
```

Código de la clase cuenta que nos permitirá sincronizar la acción del hilo haciendo que los clientes puedan hacer un solo procesos a la vez para que puedan retirar o depositar su dinero hasta que vuelvan a tener su saldo inicial de 100 dólar

- **Clase Cliente.**

```
package ec.edu.ups.modelo;

import ec.edu.ups.vista.VentanaPrincipal;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author NANCY
 */
public class Cliente implements Runnable{

    private int id;
    private Cuenta cuenta;
    int cantidad;

    public Cliente(int id,Cuenta cuenta, int cantidad) {
        this.id=id;
        this.cuenta = cuenta;
        this.cantidad = cantidad;
    }

    public Cliente(Cuenta cuenta, int cantidad) {
        this.cuenta = cuenta;
        this.cantidad = cantidad;
    }

    public Cliente(int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```
public Cuenta getCuenta() {
    return cuenta;
}
public void setCuenta(Cuenta cuenta) {
    this.cuenta = cuenta;
}
public int getCantidad() {
    return cantidad;
}
public void setCantidad(int cantidad) {
    this.cantidad = cantidad;
}
@Override
public int hashCode() {
    int hash = 7;
    hash = 13 * hash + this.id;
    return hash;
}
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Cliente other = (Cliente) obj;
    if (this.id != other.id) {
        return false;
    }
}
```

```

return true;
}
@Override
public String toString() {
    return "Cliente{" + "id=" + id + ", cuenta=" + cuenta + ", cantidad=" + cantidad + '}';
}
@Override
public void run() {
    cuenta.hacerMovimiento(cantidad);
}
}

```

Código de la clase cliente esta clase implementa la interface Runnable para después poder ser un hilo en esta clase damos al cliente sus respectivos atributos y a su vez en el método run llamamos al método de la cuenta que nos ayuda a realizar los procesos correspondientes para que la cuenta del cliente tenga el valor deciado en este caso el valor inicial que se le da al crear el cliente que es de 100 dólares

- **Clase Acción**

```

package ec.edu.ups.modelo;

import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JLabel;

/**
 *
 * @author NANCY
 */
public class Accion implements Runnable{
    int id;
    JLabel label;
    boolean[] cajeros = new boolean[3];

    public Accion() {
    }
}

```

```

public Accion(int id, JLabel label, boolean[] cajeros) {
    this.id = id;
    this.label = label;
    this.cajeros = cajeros;
}

public void avanzar(Thread hilo, int numeroCajero) {

    int tiempoCajero = (int) (Math.random() * (20 - 15 + 1) + 15);
    try {

        Thread.sleep(tiempoCajero * 100);
        cajeros[numeroCajero] = false;
    } catch (InterruptedException ex) {
        Logger.getLogger(Accion.class.getName()).log(Level.SEVERE, null, ex);
    }

    System.out.println((int) (Math.random() * (20 - 15 + 1) + 15));

}

public synchronized void salir(Thread hilo) {
    for (int i = 0; i < 13; i++) {
        notifyAll();
        label.setLocation(label.getLocation().x, label.getLocation().y + 30);
        try {
            Thread.sleep(100);
        } catch (InterruptedException ex) {
            Logger.getLogger(Accion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

public synchronized void comenzar() {
    int numeroCajero = 0;
    boolean ocupados = false;

```

```
int ubicacionX = label.getLocation().x;
int ubicacionY = label.getLocation().y;
try {
    for (int i = 0; i < 3; i++) {
        if (!cajeros[i]) {
            numeroCajero = i;
            ocupados = false;
            cajeros[i] = true;
            break;
        } else {
            ocupados = true;
        }
    }
    if (ocupados) {
        this.wait();
    }
    int tiempoLlegarCajero = (int) (Math.random() * (9 - 7 + 1) + 7);

    switch (this.id) {
        case 14 -> {
            switch (numeroCajero) {
                case 0 -> {
                    int movimiento = 190 / tiempoLlegarCajero;
                    for (int i = 0; i < tiempoLlegarCajero; i++) {
                        label.setLocation(label.getLocation().x + movimiento, label.getLocation().y);
                        Thread.sleep(308);
                    }
                }
            }
        }
        case 1 -> {
            int movimiento2 = 320 / tiempoLlegarCajero;
            for (int i = 0; i < tiempoLlegarCajero; i++) {
                label.setLocation(label.getLocation().x + movimiento2, label.getLocation().y);
                Thread.sleep(300);
            }
        }
    }
}
```

```

    }
}
case 2 -> {
    int movimiento3 = 450 / tiempoLlegarCajero;
    for (int i = 0; i < tiempoLlegarCajero; i++) {
        label.setLocation(label.getLocation().x + movimiento3, label.getLocation().y);
        Thread.sleep(290);
    }
}
}
}
case 5 -> {
    switch (numeroCajero) {
        case 0 -> {
            int movimiento = 240 / tiempoLlegarCajero;
            for (int i = 0; i < tiempoLlegarCajero; i++) {
                label.setLocation(label.getLocation().x + movimiento, label.getLocation().y);
                Thread.sleep(310);
            }
        }
        case 1 -> {
            int movimiento2 = 370 / tiempoLlegarCajero;
            for (int i = 0; i < tiempoLlegarCajero; i++) {
                label.setLocation(label.getLocation().x + movimiento2, label.getLocation().y);
                Thread.sleep(300);
            }
        }
        case 2 -> {
            int movimiento3 = 500 / tiempoLlegarCajero;
            for (int i = 0; i < tiempoLlegarCajero; i++) {
                label.setLocation(label.getLocation().x + movimiento3, label.getLocation().y);
                Thread.sleep(290);
            }
        }
    }
}
}

```



```
    }  
    }  
}  
default -> {  
    switch (numeroCajero) {  
        case 0 -> {  
            int movimiento = 290 / tiempoLlegarCajero;  
            for (int i = 0; i < tiempoLlegarCajero; i++) {  
                label.setLocation(label.getLocation().x + movimiento, label.getLocation().y);  
                Thread.sleep(310);  
            }  
        }  
        case 1 -> {  
            int movimiento2 = 420 / tiempoLlegarCajero;  
            for (int i = 0; i < tiempoLlegarCajero; i++) {  
                label.setLocation(label.getLocation().x + movimiento2, label.getLocation().y);  
                Thread.sleep(300);  
            }  
        }  
        case 2 -> {  
            int movimiento3 = 550 / tiempoLlegarCajero;  
            for (int i = 0; i < tiempoLlegarCajero; i++) {  
                label.setLocation(label.getLocation().x + movimiento3, label.getLocation().y);  
                Thread.sleep(290);  
            }  
        }  
    }  
}  
this.avanzar(new Thread(this), numeroCajero);  
this.salir(new Thread(this));  
notifyAll();  
label.setLocation(ubicacionX, ubicacionY);
```

```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

@Override
public synchronized void run() {
    boolean b = false;
    while (b == false) {
        comenzar();
    }

}

}
}

```

Código de la clase Acción que nos permite mover los labels en la parte grafica del programa es por ello que esta clase también implementa la interface runnable para poder llamar un hilo y sincronizar así los procesos de los mismos así es como controlamos cuando tiempo se demora en llegar a que ventanilla o con que cajero va a ir eso de ahí controlamos con el switch case que indica que ventanilla esta libre y otro switch case que indica que label hay que mover en esa dirección esto es apoyado de los métodos de salir que ayuda a salir al cliente de la ventana para después ocupar nuevamente el puesto y así realizar la simulación controlando los tiempos solicitados de espera tanto de salida como de espera en ventanilla.

```

package ec.edu.ups.vista;

import ec.edu.ups.controlador.Controlador;
import ec.edu.ups.controlador.ControladorCliente;
import ec.edu.ups.modelo.Cliente;
import ec.edu.ups.modelo.Cuenta;
import ec.edu.ups.modelo.Accion;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author NANCY
 */

```

```
public class VentanaPrincipal extends javax.swing.JFrame {  
    static public ImageIcon cajeroImg = new ImageIcon("src/Imagenes/cajero.jpg");  
    static public ImageIcon clienteImg = new ImageIcon("src/Imagenes/cliente.png");  
    static public ImageIcon puertaImg = new ImageIcon("src/Imagenes/Captura.PNG");  
    private ControladorCliente controlador;  
    private Accion imagen;  
    private Accion imagen2;  
    int ubicacionx;  
    int ubicaciony;  
    /**  
     * Creates new form VentanaPrincipal  
     */  
    public VentanaPrincipal() {  
        initComponents();  
        this.controlador= new ControladorCliente();  
        this.imagen= new Accion();  
        this.imagen2= new Accion();  
        jLabel1.setIcon(cajeroImg);  
        jLabel2.setIcon(cajeroImg);  
        jLabel3.setIcon(cajeroImg);  
        jLabel4.setIcon(puertaImg);  
        jLabel14.setIcon(clienteImg);  
        jLabel5.setIcon(clienteImg);  
        jLabel6.setIcon(clienteImg);  
        jLabel7.setIcon(clienteImg);  
        jLabel8.setIcon(clienteImg);  
        jLabel9.setIcon(clienteImg);  
        jLabel10.setIcon(clienteImg);  
        jLabel11.setIcon(clienteImg);  
        jLabel12.setIcon(clienteImg);  
        jLabel13.setIcon(clienteImg);  
        ubicacionx= jLabel14.getX();  
        ubicaciony= jLabel14.getY();  
    }  
}
```

```

public void cargarDatos() {
    DefaultTableModel modeloTabla = (DefaultTableModel) tblDatos.getModel();

    modeloTabla.setRowCount(0);
    for (Cliente vehiculo : controlador.getListaGenerica()) {
        Object[] rowData = {vehiculo.getId(),vehiculo.getCuenta().getSaldo()};
        modeloTabla.addRow(rowData);
    }
    tblDatos.setModel(modeloTabla);
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    jButton1.setEnabled(false);
    boolean[] cajeros = new boolean[3];
    imagen= new Accion(14, jLabel14, cajeros);
    imagen2= new Accion(5, jLabel5, cajeros);
    Accion imagen3 = new Accion(6, jLabel6, cajeros);
    Thread hilo2= new Thread(imagen);
    Thread hilo3= new Thread(imagen2);
    Thread hilo4= new Thread(imagen3);
    //hilo1.start();
    hilo2.start();
    hilo3.start();
    hilo4.start();

    for (int i = 0; i < 100; i++) {
        Cuenta cuenta = new Cuenta (100);
        Cliente cliente = new Cliente(i, cuenta,0);
        controlador.create(cliente);
        //ventanaPrincipal.modelo.setValueAt(cliente, i, cliente.getCuenta().getSaldo());
        //System.out.println(cliente);
        cargarDatos();
    }

    controlador.getListaGenerica().stream().map(c -> {
        if(c.getId())<40){
            Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 100);

```

```
Thread hilosIngresan100= new Thread(ingresa);
hilosIngresan100.start();
controlador.update(ingresa);
//ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
cargarDatos();

}

return c;
}).map(c -> {
    if(c.getId()>39 && c.getId()<60){
        Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 50);
        Thread hilosIngresan50= new Thread(ingresa);
        hilosIngresan50.start();
        controlador.update(ingresa);
        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
        cargarDatos();

    }

    return c;
}).map(c -> {
    if(c.getId()>59){
        Cliente ingresa = new Cliente(c.getId(),c.getCuenta(), 20);
        Thread hilosIngresan20= new Thread(ingresa);
        hilosIngresan20.start();
        controlador.update(ingresa);
        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
        cargarDatos();

    }

    return c;
}).filter(c -> (c.getId()>39 && c.getId()<60)).map(c -> new Cliente(c.getId(),c.getCuenta(), 20)).map(ingresa ->
new Thread(ingresa)).map(hilosIngresan20 -> {
    hilosIngresan20.start();
    return hilosIngresan20;
}).forEachOrdered(_item -> {
    //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());
```

```
cargarDatos();

});

controlador.getListaGenerica().forEach(cliente -> {

    System.out.println(cliente);

});

controlador.getListaGenerica().stream().map(c -> {

    if(c.getId()<40){

        Cliente retirar = new Cliente(c.getId(),c.getCuenta(), -100);

        Thread hilosIngresan100= new Thread(retirar);

        hilosIngresan100.start();

        controlador.update(retirar);

        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());

    }

    return c;

}).map(c -> {

    if(c.getId()>39 && c.getId()<60){

        Cliente retirar = new Cliente(c.getId(),c.getCuenta(), -50);

        Thread hilosIngresan50= new Thread(retirar);

        hilosIngresan50.start();

        controlador.update(retirar);

        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());

    }

    return c;

}).map(c -> {

    if(c.getId()>59){

        Cliente retirar = new Cliente(c.getId(),c.getCuenta(), -20);

        Thread hilosIngresan20= new Thread(retirar);

        hilosIngresan20.start();

        controlador.update(retirar);

        //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());

    }

    return c;

});
```

```

    }).filter(c -> (c.getId()>39 && c.getId()<60)).map(c -> new Cliente(c.getId(),c.getCuenta(), -
20)).forEachOrdered(retirar -> {

    Thread hilosIngresan20= new Thread(retirar);

    hilosIngresan20.start();

    controlador.update(retirar);

    //ventanaPrincipal.modelo.setValueAt(c, c.getId(), c.getCuenta().getSaldo());

});

controlador.getListaGenerica().forEach(cliente -> {

    System.out.println(cliente);

});

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    /* Set the Nimbus look and feel */

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
    * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
    */

    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 ex) {
        java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    }
}

```

```

    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(VentanaPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
    }
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new VentanaPrincipal().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 jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
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.JTable tblDatos;

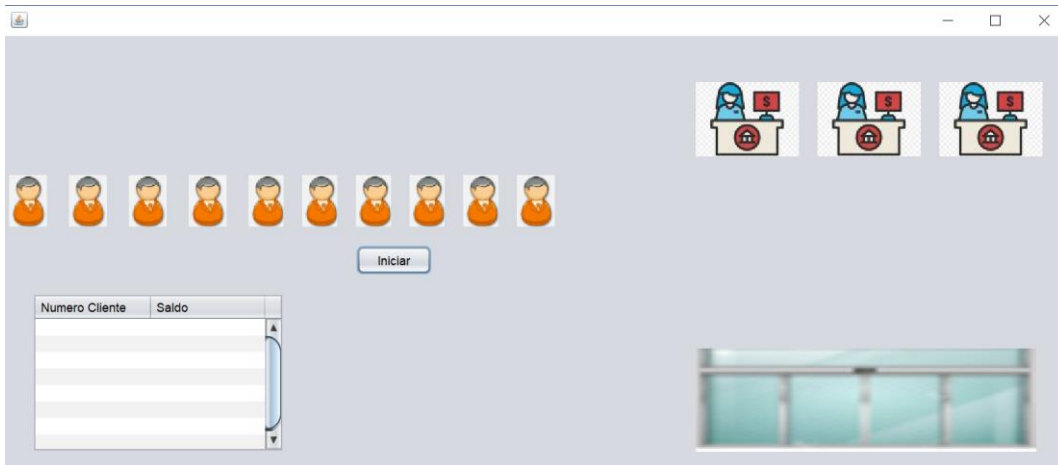
// End of variables declaration
}

```

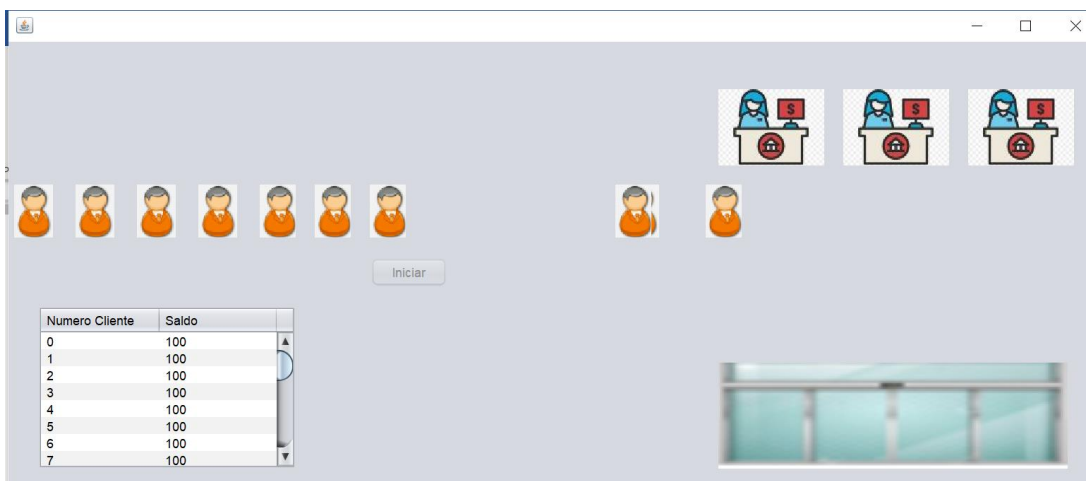
Código de la ventana principal que se encuentra en el paquete de la vista en esta ventana tenemos todos los lables para la vista del usuario y a su vez una tabla en donde se cargaran los datos respectivos al hacer las transacciones bancarias tenemos que al accionar el botón el método de accionbutton

ejecutara el programa creando los hilos de las clases con la Interface runnable para así poder ejecutar el programa de una manera correcta dentro de este método se encuentran el uso de los streams ya antes mencionados que nos sirve para recorrer la lista de los 100 clientes creados.

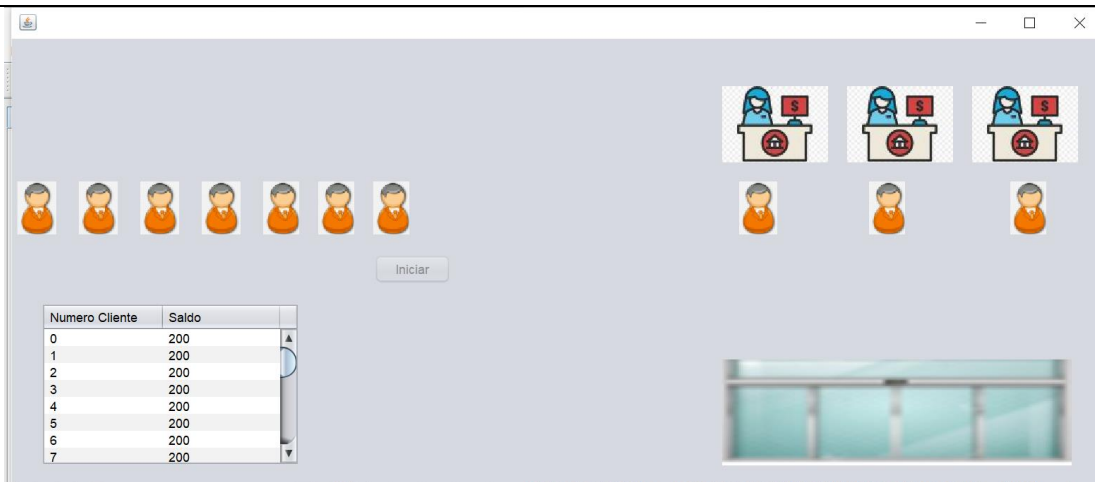
- **Comprobación de las cuentas bancarias e interfaz gráfica.**



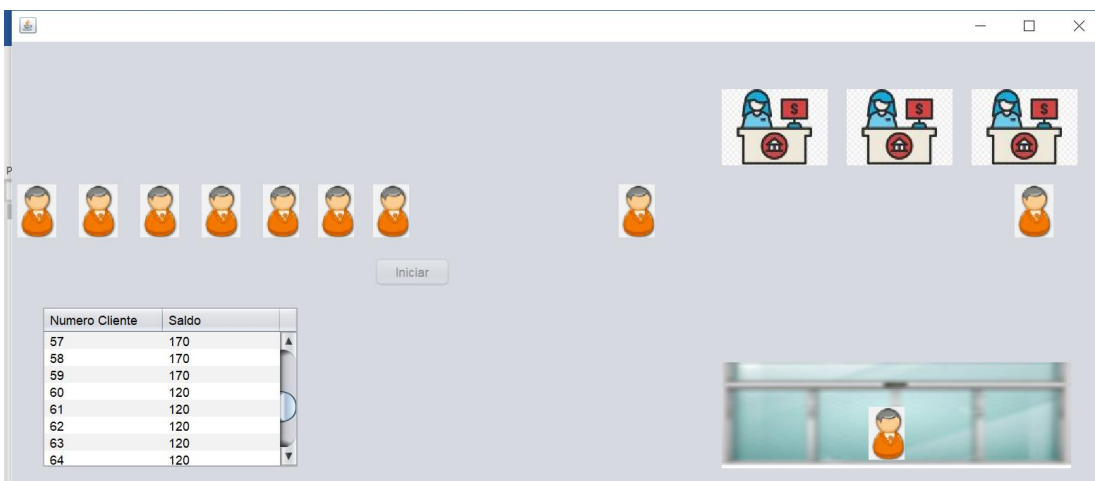
Al ejecutar el programa aparecerá la siguiente ventana en donde se observa la tabla sin crear nada los clientes en la fila los cajeros y la puerta de salida



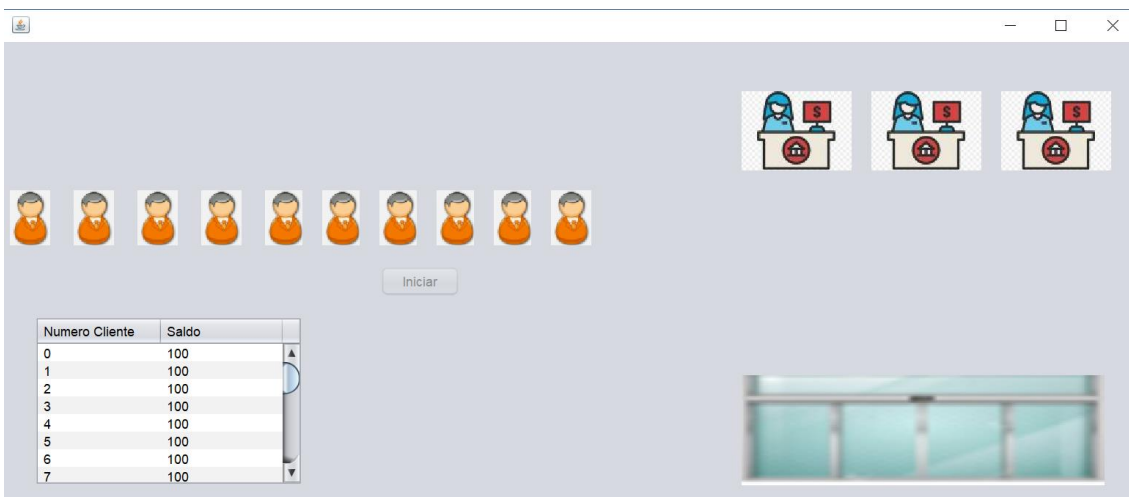
Al presionar el botón de iniciar podemos ver como las cuentas ya empiezan en 100 dólares y como los labels se muevan para llegar a ocupar un cajero

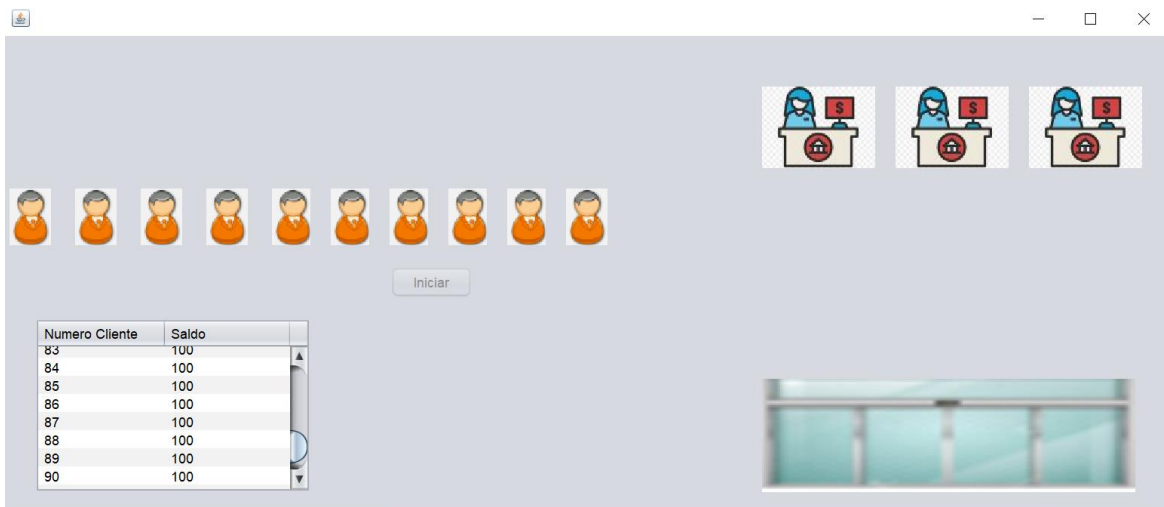
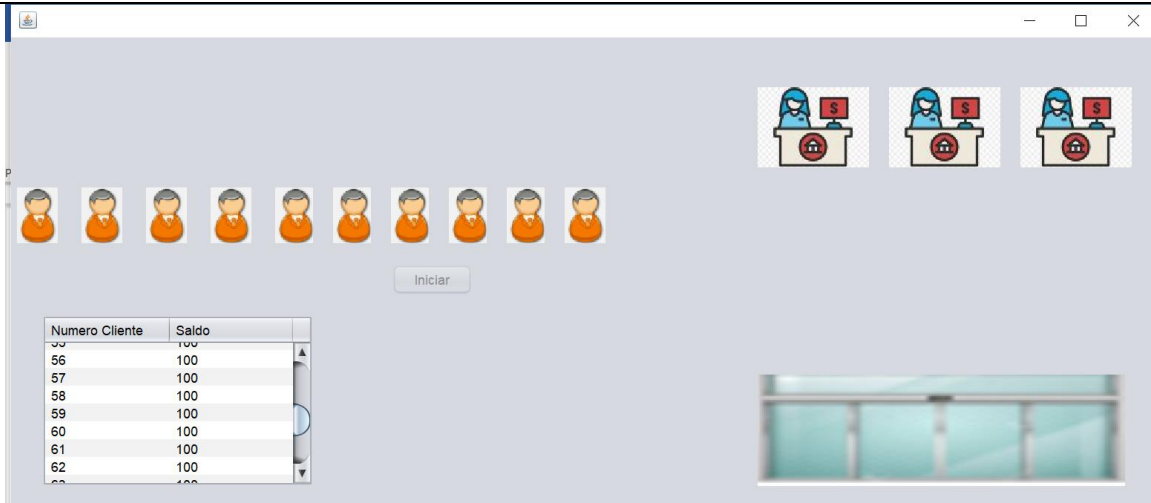


Observamos como se ubican en los cajeros y realizan sus respectivas transferencias las cuales podemos ver como cambian en la parte de la tabla



Podemos observar como los clientes salen del banco con dirección a la puerta





En las tres imágenes podemos ver como terminado el programa y las cuentas en 100 dólares los clientes ya no se mueven el botón sigue bloqueado y verificamos en la tabla que todas las cuentas de los clientes tienen un saldo de 100 dólares

RESULTADO(S) OBTENIDO(S):

Conocemos el manejo de hilos para agilizar los procesos a la hora de ejecutar el programa lo hagan de una manera más fácil y rápida.

CONCLUSIONES:

En Conclusión, esta prueba nos ayuda a entender de mejor manera los hilos y como pueden trabajar con varios procesos a la vez de una manera sincronizada y fácil

RECOMENDACIONES:

No hay recomendaciones

Estudiantes: Romel Ávila

Firma: