
	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 práctico 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 gráfica.	
		4. Deberá generar un informe de la práctica 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.

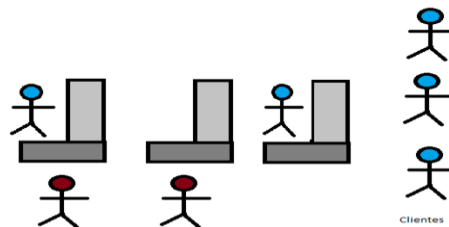
Se parte de una cuenta con 100 euros y se pueden tener procesos que ingresen 100 euros, 50 o 20. También se pueden tener procesos que retiran 100, 50 o 20 euros. Se desean tener los siguientes procesos:

- 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%

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

- Hilos 30%
- Sincronización 10%
- 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

ASIGNATURA: Programación Aplicada

NRO. PRÁCTICA:

2

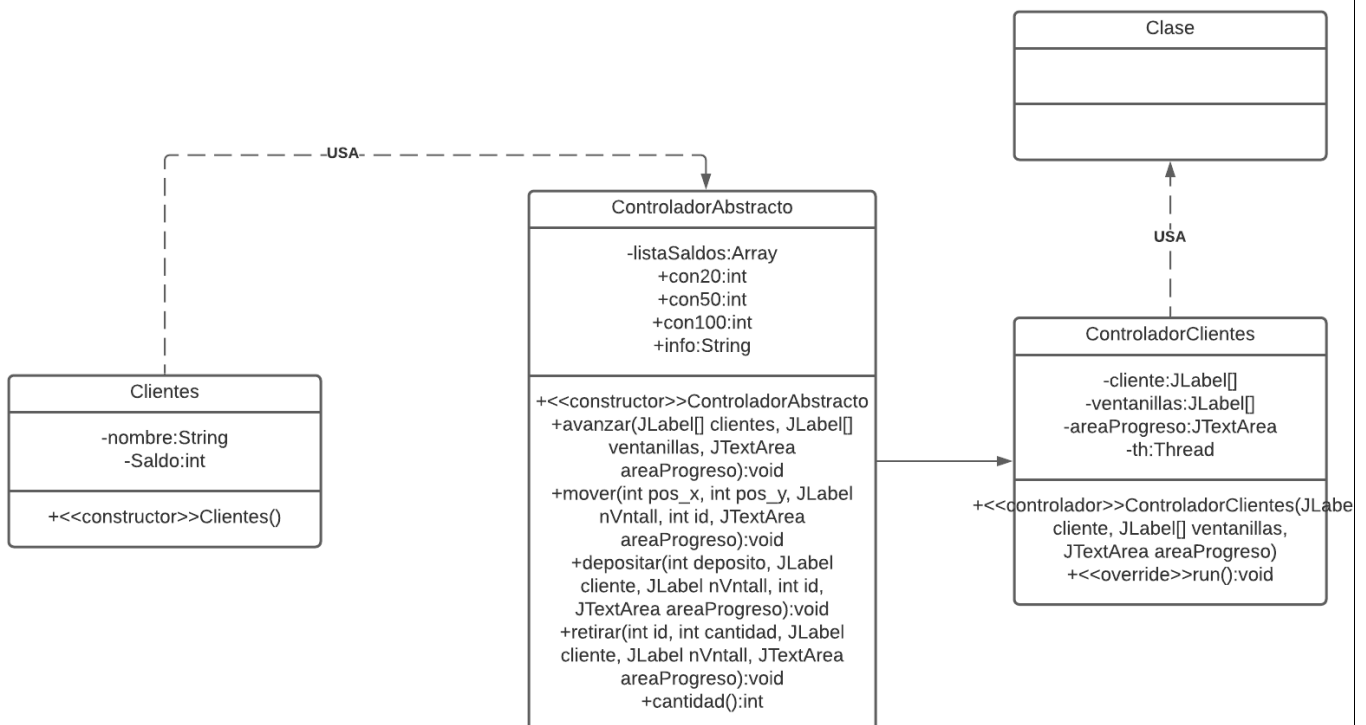
TÍTULO PRÁCTICA: Prueba practica Hilos

OBJETIVO ALCANZADO:

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

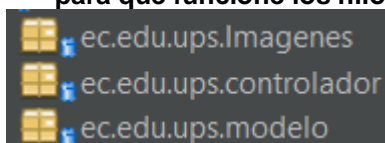
ACTIVIDADES DESARROLLADAS

1. Antes de comenzar a crear el proyecto creamos un diagrama UML para saber que atributos vamos a usar, que métodos creamos y las clases que vamos a usar.



2. Creamos el proyecto con el nombre de Prueba Practica Hilos

- a. En este proyecto creamos 3 paquetes, en el primer paquete el modelo se encontrara la clase clientes que tendrá los datos del saldo en la cuenta de banco, en el paquete controlador estarán dos clases, la una clase que es la clase abstracta tendrá los métodos para que funcione los hilos y en el otro paquete estará la interfaz.



- b. En la clase de Clientes creamos: atributos nombre y saldo, constructor, getters y setters. Esta clase tendrá los datos de cada cliente en el banco.

[https://github.com/Bryambepz/Prueba Practica Hilos/blob/master/src/ec/edu/ec/modelo/Clientes.java](https://github.com/Bryambepz/Prueba_Practica_Hilos/blob/master/src/ec/edu/ec/modelo/Clientes.java)

```

public class Clientes {
    private String nombre;

```

```
private int saldo;

public Clientes(int saldo, String nombre) {
    this.nombre = nombre;
    this.saldo = saldo;
}

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

public Clientes() {
}

public int getSaldo() {
    return saldo;
}

public void setSaldo(int saldo) {
    this.saldo = saldo;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

@Override
public String toString() {
    return "nombres="+nombre+" saldo=" + saldo + ' ';
}

}
```

- c. En el siguiente paquete de los controladores estará la clase **ControladorAbstracto**, instanciaremos algunos atributos y creamos un método **synchronized** para el funcionamiento del hilo, existirán métodos para la movilidad de los labels, depositar o registrar dinero del cliente, esta clase será una clase abstracta y la clase **ControladorClientes** será la clase que herede de ella.

```
public abstract class ControladorAbstracto {

    private Clientes[] listaSaldos;
    int con20;
    int con50;
    int con100;
    String info;

    public ControladorAbstracto() {
        this.listaSaldos = new Clientes[10];
    }

    public synchronized void avanzar(JLabel[] clientes, JLabel[] ventanillas, JTextArea areaProgreso) {
        int pos_x = 0;
        int pos_y = 0;
        int contador = 0;

        for (int i = 0; i < 10; i++) {
            Clientes clientes1 = new Clientes(clientes[i].getName());
        }
    }
}
```

```

        listaSaldos[i] = clientes1;
    }
    for (int i = 0; i < clientes.length; i++) {
        int t = (int) (Math.random() * 2);
        System.out.println((i+1) + " -- " + t);
        try {
            JLabel cliente = clientes[i];
            pos_x = cliente.getX();
            pos_y = cliente.getY();
            JLabel nVntall = ventanillas[contador];
            if (i == 9) {
                Thread.sleep(500);
                mover(pos_x, pos_y, cliente, nVntall);
                if (t == 0) {
                    System.out.println("deposito");
                    if (listaSaldos[i].getSaldo() >= 100) {
                        depositar(cantidad(), cliente, nVntall, i,
//
areaProgreso);

                            retirar(i, cantidad(), cliente, nVntall, areaPro-
greso);

                            info = "ha llegado al limite en su cuenta\n";
                            areaProgreso.append(info);
                        } else {
                            depositar(cantidad(), cliente, nVntall, i, areaPro-
greso);

                        }
//
                        System.out.println(cliente.getName() + " ha lle-
gado al limite en su cuenta");
                    } else if (t == 1) {
                        System.out.println("retiro");
                        retirar(i, cantidad(), cliente, nVntall, areaProgreso);
                    }
                    Thread.sleep(500);
                    i = -1;
                    for (JLabel clientess : clientes) {
                        clientess.setVisible(true);
                    }
                    cliente.setLocation(759, 190 + (i * 84));
                } else {
                    mover(pos_x, pos_y, cliente, nVntall);
                    if (t == 0) {
                        System.out.println("deposito");
                        if (listaSaldos[i].getSaldo() >= 100) {
                            depositar(cantidad(), cliente, nVntall, i, areaPro-
greso);

                                retirar(i, cantidad(), cliente, nVntall, areaProgreso);
                                info = "ha llegado al limite en su cuenta\n";
                                areaProgreso.append(info);
                            } else {
                                depositar(cantidad(), cliente, nVntall, i, areaPro-
greso);

                            }
                        } else if (t == 1) {
                            System.out.println("retiro");
                            retirar(i, cantidad(), cliente, nVntall, areaProgreso);
                        }
                    }
                    Thread.sleep(500);
                    cliente.setVisible(false);
                    cliente.setLocation(759, 190 + (i * 84));
                }
            }
        }
    }
}

```

```
        System.out.println(con20 + " / " + con50 + " / " + con100);
        if (contador < 2) {
            contador++;
        } else {
            contador = 0;
        }
    } catch (InterruptedException ex) {
        Logger.getLogger(ContraladorAbstracto.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void mover(int pos_x, int pos_y, JLabel cliente, JLabel nVntall) {
    for (int j = pos_x; j > nVntall.getX() + 37; j--) {
        try {
            cliente.setLocation(j, pos_y);
            Thread.sleep(4);
            if (j == nVntall.getX() + 38) {
                for (int k = pos_y; k > 191; k--) {
                    cliente.setLocation(j, k);
                    Thread.sleep(4);
                }
                System.out.println(" nC " + cliente.getName());
            }
        } catch (InterruptedException ex) {
            Logger.getLogger(ContraladorAbstracto.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

public void depositar(int deposito, JLabel cliente, JLabel nVntall, int id,
JTextArea areaProgreso) {
    try {
        Thread.sleep(1000);
        var saldo = listaSaldos[id].getSaldo() + deposito;
        listaSaldos[id].setSaldo(saldo);
        info = "El " + cliente.getName() + " deposito " + deposito + "\nSaldo
actual = " + saldo + "\n";
        areaProgreso.append(info);
    } catch (InterruptedException ex) {
        Logger.getLogger(ContraladorAbstracto.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

public void retirar(int id, int cantidad, JLabel cliente, JLabel nVntall, JTex-
tArea areaProgreso) {
    try {
        Thread.sleep(1000);
        int saldoR = 0;
        if (listaSaldos[id].getSaldo() >= cantidad) {
            saldoR = listaSaldos[id].getSaldo() - cantidad;
            listaSaldos[id].setSaldo(saldoR);
            info = cliente.getName() + " retiro " + cantidad + "\nEl saldo ac-
tual es de " + saldoR + "\n";
            areaProgreso.append(info);
        } else {
            depositar(cantidad, cliente, cliente, id, areaProgreso);
            info = cliente.getName() + " no pudo retirar " + cantidad

```

```

        + "\nSaldo insuficiente en cuenta \n" + "Saldo actual = " +
listaSaldos[id].getSaldo() + "\n";
        areaProgreso.append(info);
    }
} catch (InterruptedException ex) {
    Logger.getLogger(ControladorAbstracto.class.getName()).log(Level.SEVERE,
null, ex);
}
}

public int cantidad() {

    int valor = (int) (Math.random() * 3);
    switch (valor) {
        case 0:
            while (con20 <= 60) {
                con20++;
                return 20;
            }
        case 1:
            while (con50 <= 20) {
                con50++;
                return 50;
            }
        case 2:
            while (con100 <= 40) {
                con100++;
                return 100;
            }
        default:
            break;
    }
    return 0;
}
}

```

- d. En la clase constructorClientes que será la que herede los métodos su clase padre también tendrá instanciados unos atributos que servirán como parámetros de algunos métodos que se llamaron de la clase padre y se encontrara el método run() para dar inicio al hijo.

```

public class ControladorClientes extends ControladorAbstracto implements Runnable {

    private JLabel[] cliente;
    private JLabel[] ventanillas;
    private JTextArea areaProgreso;
    private Thread th;
    // private static ControladorClientes ctrlCliente;

    public ControladorClientes() {
        super();
    }

    // public ControladorClientes getCtrlMonitor(){
    //     if (ctrlCliente == null) {
    //         ctrlCliente = new ControladorClientes();
    //     }
    //     return ctrlCliente;
    // }

    public ControladorClientes(JLabel[] cliente, JLabel[] ventanillas, JTextArea
areaProgreso) {
        super();
        this.cliente = cliente;
    }
}

```



```

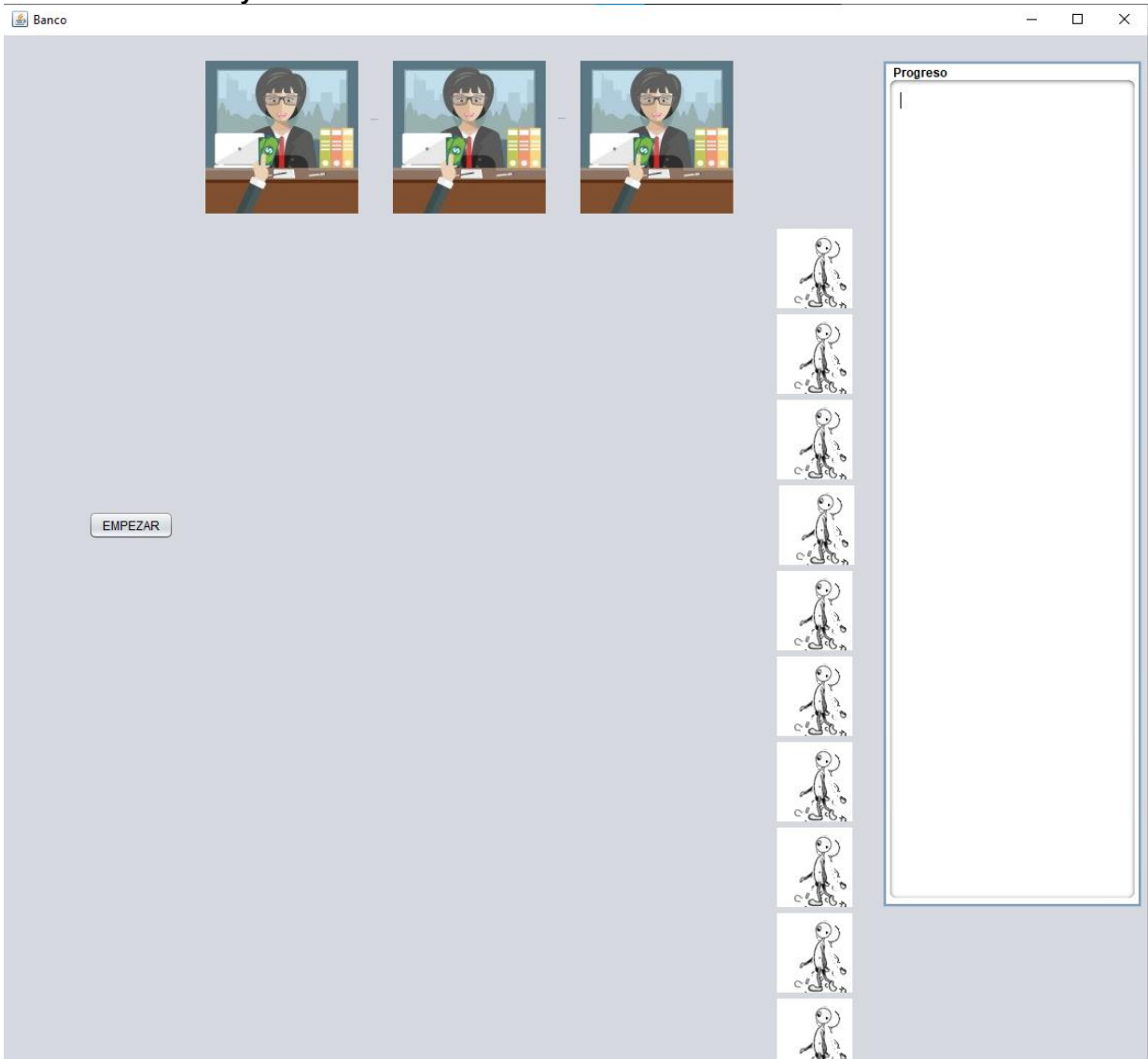
        this.ventanillas = ventanillas;
        this.areaProgreso = areaProgreso;
        th = new Thread(this);
        th.start();
    }

    @Override
    public void run() {
        this.avanzar(cliente, ventanillas, areaProgreso);
    }
}
3. }

```

4. FUNCIONAMIENTO

Al iniciar la aplicación se visualizará la interfaz gráfica, al presionar en “EMPEZAR” el programa iniciará a dar funcionamiento a los hilos, comenzará a moverse cada label a una ventanilla y el cliente elegirá si deposita o retira una cantidad de dinero, ya sea 20, 50 o 100. A un lado se encontrará un JTextArea en el que estará los datos del progreso, según la acción que realice cada cliente y un estado del avance de la cuenta del cliente.



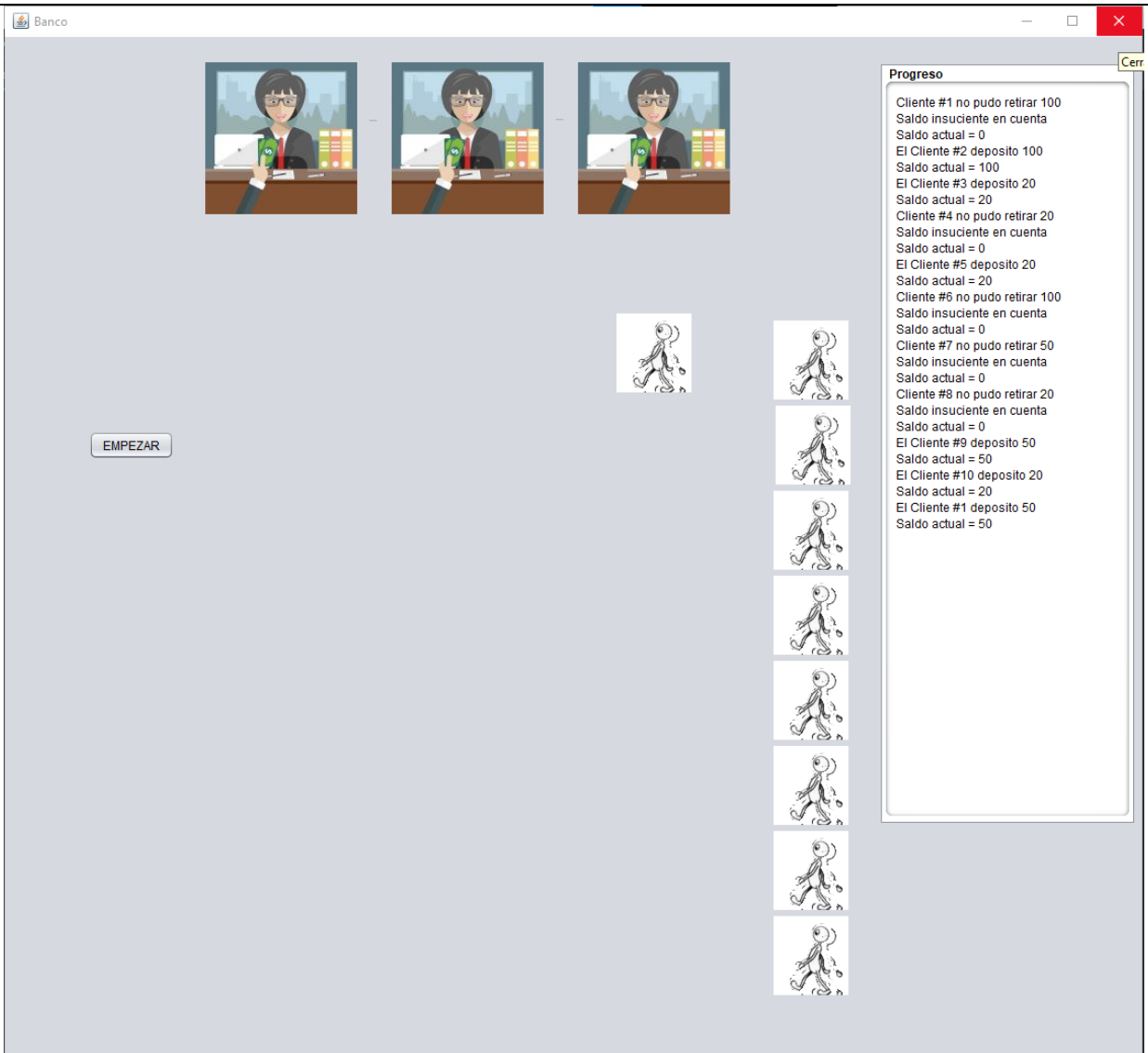


EMPEZAR



Progreso

Ciente #1 no pudo retirar 100
Saldo insuficiente en cuenta
Saldo actual = 0
El Ciente #2 deposito 100
Saldo actual = 100
El Ciente #3 deposito 20
Saldo actual = 20
Ciente #4 no pudo retirar 20
Saldo insuficiente en cuenta
Saldo actual = 0



N.

RESULTADO(S) OBTENIDO(S):

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

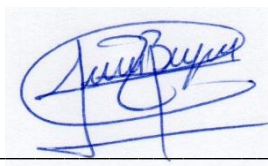
CONCLUSIONES:

- Los estudiantes identifican las principales estructuras para la creación de sistemas informáticos.
- Los estudiantes implementan soluciones graficas 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.

Nombre de estudiante: Bryam Parra



Firma de estudiante: _____