
	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.

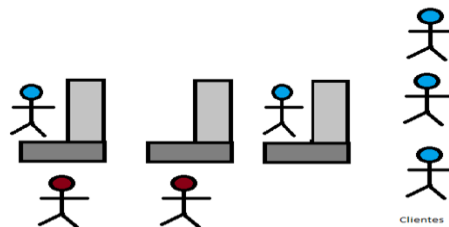
Se parte de una cuenta con 100 euros y se pueden tener procesos que ingresen 100 euros, 50 o 20. También se pueden tener procesos que retiran 100, 50 o 20 euros. Se desean tener los siguientes procesos:

- 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 una 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 número 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:

ASIGNATURA:

NRO. PRÁCTICA:

TÍTULO PRÁCTICA:

OBJETIVO ALCANZADO:

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

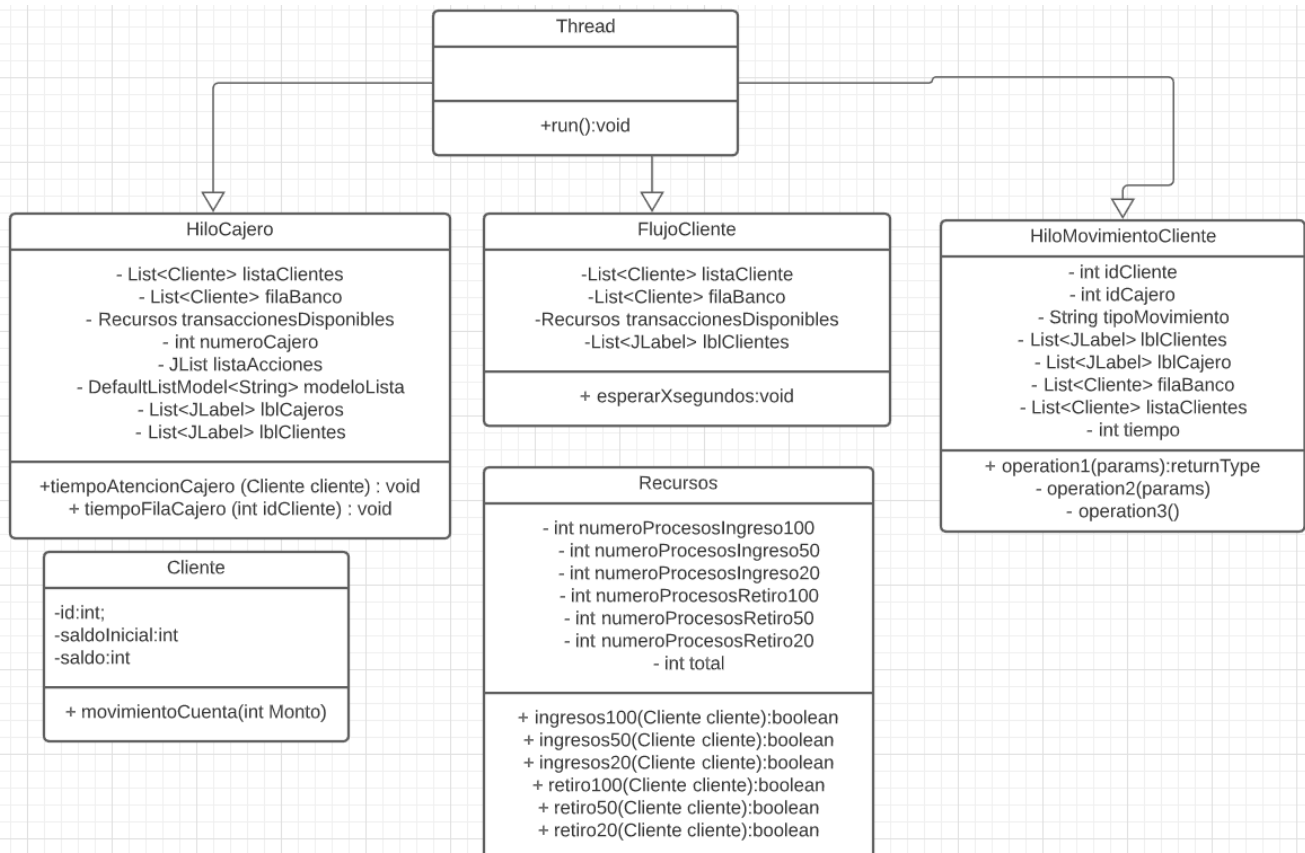
ACTIVIDADES DESARROLLADAS

1. Diseñar un programa que cumpla con los requerimientos del problema.

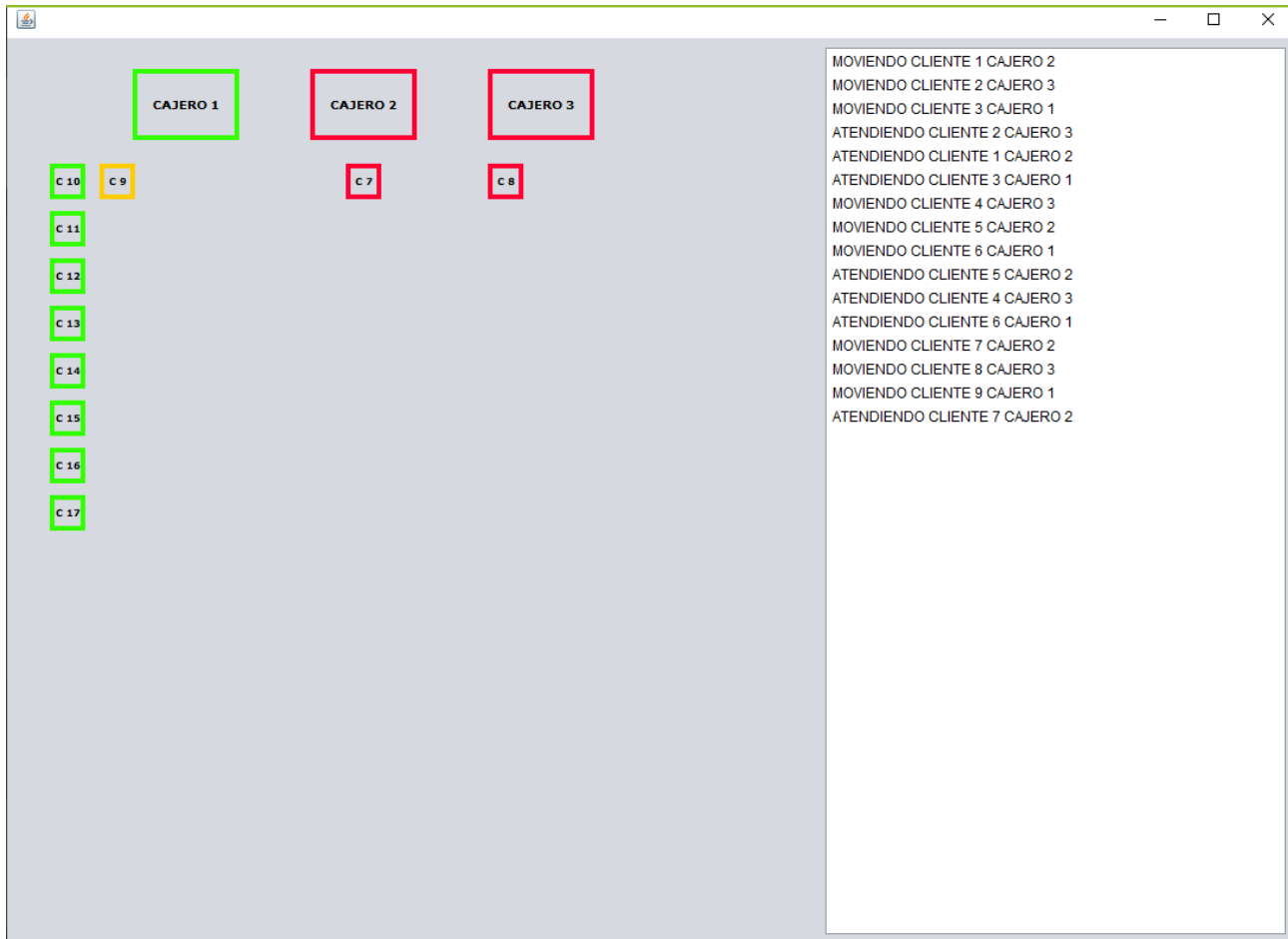
2. Descripción de la solución de problema

RESULTADO(S) OBTENIDO(S):

Diseño del UML:



Para esta práctica se hizo el uso de Hilos y sus sincronizaciones para evitar el problema de cruce de datos.



Se implementó una interfaz gráfica donde podemos observar como los clientes y los cajeros se ocupan con cada tarea. Para esto se usa una lista donde contiene todos los labels creados para que los hilos puedan dar uso de los mismos.

CLASE HILO CAJERO

Esta clase se encarga de gestionar a cada uno de los clientes que llegan a ventanilla, para eso se sincroniza las listas para evitar que dos hilos modifiquen los recursos al mismo tiempo.

```
public class HiloCajero extends Thread {

    private List<Cliente> listaClientes;
    private List<Cliente> filaBanco;
    private Recursos transaccionesDisponibles;
    private int numeroCajero;
    private JList listaAcciones;
    private DefaultListModel<String> modeloLista;
    private List<JLabel> lblCajeros;
    private List<JLabel> lblClientes;

    public HiloCajero(List<Cliente> listaClientes, List<Cliente> filaBanco, Recursos transac-
cionesDisponibles, int numeroCajero, JList listaAcciones, DefaultListModel<String> modelo-
Lista, List<JLabel> lblCajeros, List<JLabel> lblClientes) {
        this.listaClientes = listaClientes;
        this.filaBanco = filaBanco;
        this.transaccionesDisponibles = transaccionesDisponibles;
        this.numeroCajero = numeroCajero;
        this.listaAcciones = listaAcciones;
    }
}
```

```

        this.modeloLista = modeloLista;
        this.lblCajeros = lblCajeros;
        this.lblClientes = lblClientes;
    }

    @Override
    public void run() {
        while (transaccionesDisponibles.getNumeroProcesosIngreso100() != 0 || transaccionesDisponibles.getNumeroProcesosIngreso20() != 0 || transaccionesDisponibles.getNumeroProcesosIngreso50() != 0
            || transaccionesDisponibles.getNumeroProcesosRetiro100() != 0 || transaccionesDisponibles.getNumeroProcesosRetiro20() != 0 || transaccionesDisponibles.getNumeroProcesosRetiro50() != 0) {
            Cliente clienteAtencion = null;
            synchronized (filaBanco) {
                if (!filaBanco.isEmpty()) {
                    clienteAtencion = filaBanco.get(0);
                    filaBanco.remove(0);
                }
            }
            if (clienteAtencion != null) {
                //COLOR AMARILLO MOVIMIENTO
                lblClientes.get(clienteAtencion.getId()).setBorder(javax.swing.BorderFactory.createMatteBorder(4, 4, 4, 4, new Color(255, 204, 0)));
                tiempoFilaCajero(clienteAtencion.getId());
                //CAJERO ROJO OCUPADO
                lblClientes.get(clienteAtencion.getId()).setBorder(javax.swing.BorderFactory.createMatteBorder(4, 4, 4, 4, new Color(255, 0, 51)));
                lblCajeros.get(numeroCajero).setBorder(javax.swing.BorderFactory.createMatteBorder(4, 4, 4, 4, new Color(255, 0, 51)));
                boolean aleatorio = true;
                while (aleatorio) {
                    int tipoTransaccion = (int) (Math.random() * 10);
                    if (tipoTransaccion > 5) {
                        tipoTransaccion = 1;
                    } else {
                        tipoTransaccion = 0;
                    }
                    int montoTransaccion = (int) (Math.random() * 10);
                    if (montoTransaccion < 3) {
                        montoTransaccion = 0;
                    } else if (montoTransaccion > 3 && montoTransaccion < 7) {
                        montoTransaccion = 1;
                    } else {
                        montoTransaccion = 2;
                    }
                    if (tipoTransaccion == 0) {
                        //DEPOSITO
                        switch (montoTransaccion) {
                            case 0:
                                aleatorio = transaccionesDisponibles.ingresos100(clienteAtencion);
                                break;
                            case 1:
                                aleatorio = transaccionesDisponibles.ingresos50(clienteAtencion);
                                break;
                            case 2:
                                aleatorio = transaccionesDisponibles.ingresos20(clienteAtencion);
                                break;
                        }
                    } else {
                        //Retiro
                        switch (montoTransaccion) {
                            case 0:
                                aleatorio = transaccionesDisponibles.retiro100(clienteAtencion);

```

```

        break;
        case 1:
            aleatorio = transaccionesDisponibles.retiro50(clienteAten-
cion);
            break;
        case 2:
            aleatorio = transaccionesDisponibles.retiro20(clienteAten-
cion);
            break;
    }
}
synchronized (modeloLista) {
    modeloLista.addElement("ATENDIENDO CLIENTE " + (clienteAtencion.getId() +
1) + " CAJERO " + (numeroCajero + 1));
    listaAcciones.setModel(modeloLista);
}
System.out.println("ATENDIENDO CLIENTE " + (clienteAtencion.getId() + 1) + "
CAJERO " + (numeroCajero + 1));
tiempoAtencionCajero(clienteAtencion);
listaClientes.add(clienteAtencion);
}
}
System.out.println(transaccionesDisponibles.getNumeroProcesosIngreso100() + " - " +
transaccionesDisponibles.getNumeroProcesosIngreso20() + " - " + transaccionesDisponibles.get-
NumeroProcesosIngreso50() + " - "
+ transaccionesDisponibles.getNumeroProcesosRetiro100() + " - " + transaccio-
nesDisponibles.getNumeroProcesosRetiro20() + " - " + transaccionesDisponibles.getNumeroProce-
sosRetiro50());
}

private void tiempoAtencionCajero(Cliente clienteAtencion) {
    try {
        //CADA 15 - 20 segundos se realizara 1.5 - 2 segundos
        int tiempo = (int) (Math.random() * 500) + 2500;
        Thread.sleep(tiempo);
        new Thread(new HiloMovimientoClientes(clienteAtencion.getId(), numeroCajero, "sa-
lirCaja", lblClientes, lblCajeros, tiempo, filaBanco)).start();
    } catch (InterruptedException ex) {
        Thread.currentThread().interrupt();
    }
}

private void tiempoFilaCajero(int idCliente) {
    try {
        //CADA 2 - 5 segundos se realizara 0.2 - 0.5 segundos
        synchronized (modeloLista) {
            modeloLista.addElement("MOVIENDO CLIENTE " + (idCliente + 1) + " CAJERO " +
(numeroCajero + 1));
            listaAcciones.setModel(modeloLista);
        }
        System.out.println("MOVIENDO CLIENTE " + (idCliente + 1) + " CAJERO " + (numero-
Cajero + 1));
        int tiempo = (int) (Math.random() * 300) + 1000;
        new Thread(new HiloMovimientoClientes(idCliente, numeroCajero, "irCaja", lbl-
Clientes, lblCajeros, tiempo, filaBanco)).start();
        Thread.sleep(tiempo);
    } catch (InterruptedException ex) {
        Thread.currentThread().interrupt();
    }
}
}
}

```

CLASE FLUJO CLIENTES

Esta clase se encarga de gestionar los componentes en la parte gráfica y además la lista de los clientes en el banco.

```
public class FlujoCliente extends Thread {

    private List<Cliente> listaClientes;
    private List<Cliente> filaBanco;
    private Recursos transaccionesDisponibles;
    private List<JLabel> lblClientes;

    public FlujoCliente(List<Cliente> listaClientes, List<Cliente> filaBanco, Recursos
transaccionesDisponibles, List<JLabel> lblClientes) {
        this.listaClientes = listaClientes;
        this.filaBanco = filaBanco;
        this.transaccionesDisponibles = transaccionesDisponibles;
        this.lblClientes = lblClientes;
    }

    @Override
    public void run() {
        while (transaccionesDisponibles.getTotal() > filaBanco.size()) {
            esperarXsegundos();
            filaBanco.add(listaClientes.get(0));
            int posicionY = 0;
            if (listaClientes.get(0).getId() == 0) {
                posicionY = lblClientes.get(99).getY();
            } else {
                posicionY = lblClientes.get(listaClientes.get(0).getId() - 1).getY();
            }

            posicionY += 40;
            lblClientes.get(listaClientes.get(0).getId()).setBounds(30, posicionY, 30, 30);
            lblClientes.get(listaClientes.get(0).getId()).setVisible(true);
            listaClientes.remove(0);
            System.out.println("AGREGANDO NUEVO CLIENTE A LA FILA TOTAL: " + filaBanco.size()
+ " OTRA LISTA: " + listaClientes.size());
        }


        private void esperarXsegundos() {
            try {
                //CADA 30 - 50 segundos se realizara 0.5 - 1.5segundos
                int tiempo = (int) (Math.random() * 500) + 1000;
                Thread.sleep(tiempo);
            } catch (InterruptedException ex) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Repositorio github:

<https://github.com/psidrovo/PruebaPractica2.git>

CONCLUSIONES:

El tema de los hilos es uno que nos ayuda a gestionar procesos en multitarea, en ciertas situaciones tenemos que analizar bien la finalidad de nuestro proyecto, además de que podemos tener problemas al momento de sincronizar, porque se puede necesitar un recurso por varios hilos.

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

RECOMENDACIONES:

Para fomentar el trabajo en equipo con los futuros colegas, sería ideal que se realice prácticas en grupos.

Nombre de estudiante: Paul Sebastian Idrovo Berrezueta

Firma de estudiante:

