

**Tema**

HILOS CON JAVA

SINCRONIZACÓN

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

27/11/2022

HILOS CON JAVA: SINCRONIZACIÓN

[1. INTRODUCCIÓN 4](#_Toc120697183)

[2. OBJETIVOS 4](#_Toc120697184)

[2.1 OBJETIVO GENERAL 4](#_Toc120697185)

[2.2 OBJETIVOS ESPECÍFICOS 4](#_Toc120697186)

[3. MARCO TEÓRICO 4](#_Toc120697187)

[3.1 HILOS 4](#_Toc120697188)

[3.1.1 HILOS CON JAVA 5](#_Toc120697189)

[3.1.2 SINCRONIZACIÓN 5](#_Toc120697190)

[3.2 PATRÓN MODELO VISTA CONTROLADOR 6](#_Toc120697191)

[3.2.1 MODELO 6](#_Toc120697192)

[3.2.2 VISTA 6](#_Toc120697193)

[3.2.3 CONTROLADOR 7](#_Toc120697194)

[4. PARTE PRÁCTICA 7](#_Toc120697195)

[4.1 CODIFICACIÓN VIDEO 173 7](#_Toc120697196)

[4.2 CODIFICACIÓN VIDEO 174 9](#_Toc120697197)

[4.3 CODIFICACIÓN VIDEO 175 11](#_Toc120697198)

[4.4 CODIFICACIÓN VIDEO 176 11](#_Toc120697199)

[4.5 CODIFICACIÓN VIDEO 177 13](#_Toc120697200)

[4.6 CODIFICACIÓN VIDEO 178 14](#_Toc120697201)

[4.7 CODIFICACIÓN FINAL DE LA PRACTICA 15](#_Toc120697202)

[4.8 FUNCIONALIDAD 21](#_Toc120697203)

[4.8.1 ESTRUCTURA DE LA APLICACIÓN 21](#_Toc120697204)

[4.8.2 EJECUCIÓN DE LA APLICACIÓN 21](#_Toc120697205)

[5. CONCLUSIONES 24](#_Toc120697206)

[6. RECOMENDACIONES 25](#_Toc120697207)

[7. REFERENCIAS 25](#_Toc120697208)

**INDICE DE IMÁGENES**

[Figura 1. Hilos y Procesos 4](#_Toc120670421)

[Figura 2. Logo identificativo de Java 5](#_Toc120670422)

[Figura 3. Sincronización de hilos 5](#_Toc120670423)

[Figura 4. Modelo Vista Controlador 7](#_Toc120670424)

[Figura 5. Codificación clase BancoModelo 8](#_Toc120670425)

[Figura 6. Codificación método acreditarSaldoCuentas 8](#_Toc120670426)

[Figura 7. Codificación método transferirEntreCuentas 9](#_Toc120670427)

[Figura 8. Codificación método ObtenerSaldoTotal 9](#_Toc120670428)

[Figura 9. Codificación clase TransferenciasControlador 9](#_Toc120670429)

[Figura 10. Codificación método run para el proceso de transferencia 10](#_Toc120670430)

[Figura 11. Codificación clase main con el método principal 10](#_Toc120670431)

[Figura 12. Codificación clase BancoControlador 11](#_Toc120670432)

[Figura 13. Codificación clase BancoVista 12](#_Toc120670433)

[Figura 14. Codificación método imprimirErrorTransferencia 12](#_Toc120670434)

[Figura 15. Codificación método imprimirTransferenciaExito 13](#_Toc120670435)

[Figura 16. Codificación método transferirEntreCuentas 13](#_Toc120670436)

[Figura 17. Codificación clase BancoControlador 14](#_Toc120670437)

[Figura 18. Codificación método transferirEntreCuentas 14](#_Toc120670438)

[Figura 19. Codificación método transferirEntreCuentas 15](#_Toc120670439)

[Figura 20. Estructura de la aplicación 21](#_Toc120670440)

[Figura 21. Ejecución video 173 21](#_Toc120670441)

[Figura 22. Ejecución video 174 22](#_Toc120670442)

[Figura 23. Ejecución video 175 22](#_Toc120670443)

[Figura 24. Ejecución video 176 23](#_Toc120670444)

[Figura 25. Ejecución video 177 23](#_Toc120670445)

[Figura 26. Ejecución video 178 24](#_Toc120670446)

**ÍNDICE DE TABLAS**

[Tabla 1. Codificación del archivo Main.java. 17](file:///C:\Users\santi\Downloads\Informes\videosbanco.docx#_Toc120670447)

[Tabla 2. Codificación del archivo BancoModelo.java 18](file:///C:\Users\santi\Downloads\Informes\videosbanco.docx#_Toc120670448)

[Tabla 3. Codificación del archivo BancoControlador.java 19](file:///C:\Users\santi\Downloads\Informes\videosbanco.docx#_Toc120670449)

[Tabla 4. Codificación del archivo TransferenciasControlador.java 20](file:///C:\Users\santi\Downloads\Informes\videosbanco.docx#_Toc120670450)

[Tabla 5. Codificación del archivo BancoVista.java 21](file:///C:\Users\santi\Downloads\Informes\videosbanco.docx#_Toc120670451)

# INTRODUCCIÓN

Las aplicaciones de software siempre necesitan realizar varias tareas a la vez y no una después de que finalice la otra, esto puede provocar que el sistema y el usuario no sean eficientes al interactuar el uno con el otro. En un flujo de tareas normal una se ejecutará después de la otra, pero lo ideal es mantener varios flujos de ejecución para poder realizar una tarea sin necesidad de esperar a las otras (Nakayama, 2009).

Una aplicación comúnmente debe realizar varias operaciones a la vez como descargar información o contenido, actualizar información del catálogo de productos, sincronizar información con otros usuarios, etc. Los hilos permiten ejecutar varios procesos de forma paralela, tomando en cuenta el acceso a los recursos para que no se provoque una pérdida de información.

# OBJETIVOS

### **OBJETIVO GENERAL**

Analizar y comprender la sincronización de hilos, para que se usa y como implementarla con el lenguaje de programación JAVA.

### **OBJETIVOS ESPECÍFICOS**

* Realizar un estudio del estado del arte sobre la sincronización de hilos en JAVA.
* Analizar la importancia del uso de la sincronización de hilos.
* Desarrollar un ejemplo práctico para comprender el comportamiento de los hilos al sincronizarlos.

# MARCO TEÓRICO

## HILOS

Un hilo es un solo flujo de ejecución de instrucciones dentro de un proceso, debido a que no pueden ejecutarse por sí solos, requieren la supervisión de un proceso (Nakayama, 2009). Existen varios lenguajes de programación que permiten crear y gestionar hilos, entre los más conocidos están JAVA y C++.

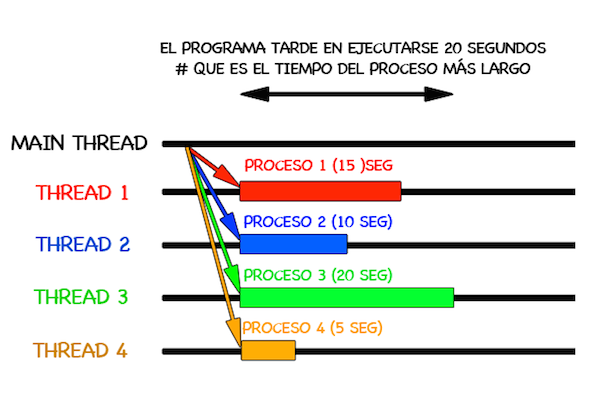


Figura 1. Hilos y Procesos

### **HILOS CON JAVA**

JAVA es un lenguaje de programación de altor nivel con tipado estático de datos, está en casi todos los sistemas y dispositivos informáticos convirtiéndolo en uno de los lenguajes más usados y demandados de la actualidad. Soporta múltiples paradigmas de programación y ha sido la base para crear lenguajes de programación modernos como SCALA o Kotlin.

El lenguaje de programación JAVA permite crear hilos y administrarlos gracias a su JVM (Java Virtual Machine). Java es capaz de manejar múltiples hilos con varios flujos de instrucciones y un flujo de datos de forma simultánea (Díaz, 2022).



Figura 2. Logo identificativo de Java

### **SINCRONIZACIÓN**

Los hilos no siempre trabajan de forma independiente, en algunos casos deben trabajar de forma coordinada para resolver algún problema o necesidad del usuario. Cuando los hilos trabajan con el mismo flujo o fuente de información se puede provocar inconsistencias, errores de cálculos y en el peor de los casos el colapso del sistema.

JAVA dispone de un mecanismo de sincronización de hilos, los métodos declarados con synchronized utilizan el cerrojo de la clase Object para evitar que más de un hilo entre en ellos al mismo tiempo (Universidad de Alicante, 2012).

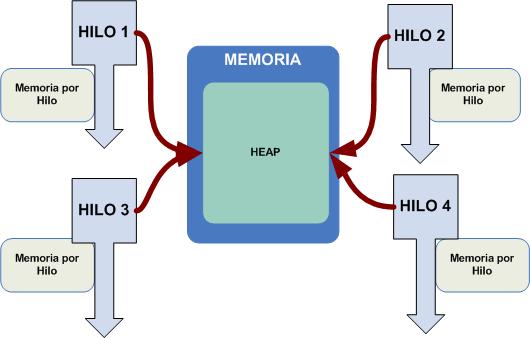


Figura 3. Sincronización de hilos

## PATRÓN MODELO VISTA CONTROLADOR

Es un patrón de arquitectura para el desarrollo de aplicaciones software donde su objetivo es el separar la lógica del negocio de la interfaz de usuario lo cual lo hace escalable facilitando la evolución por separado de sus partes, incrementa la reutilización y flexibilidad.

Tuvo su origen en 1979 y es también conocido como patrón MVC, divide una aplicación software en tres grandes partes bien diferenciadas las cuales son Modelo, Vista y Controlador.

MODELO: Es una capa donde se localiza la funcionalidad central y los datos, se comunica con el controlador y la base de datos.

VISTA: Es la capa donde se muestra la información al usuario donde es posible definir una o más vista de usuario en un software, se comunica con el controlador y el usuario.

CONTROLADOR: Es la capa donde se manejan las entradas del usuario se separa la representación interna de la información y la forma en la que se le muestra al usuario, se comunica con el modelo.

### **MODELO**

Es una capa donde se localiza la funcionalidad central y los datos, se comunica con el controlador y la base de datos.

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

### **VISTA**

Es la capa donde se muestra la información al usuario donde es posible definir una o más vista de usuario en un software, se comunica con el controlador y el usuario.

Las vistas, como su nombre nos hacen entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generarán la salida, tal como nuestra aplicación requiera.

### **CONTROLADOR**

Es la capa donde se manejan las entradas del usuario se separa la representación interna de la información y la forma en la que se le muestra al usuario, se comunica con el modelo.

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

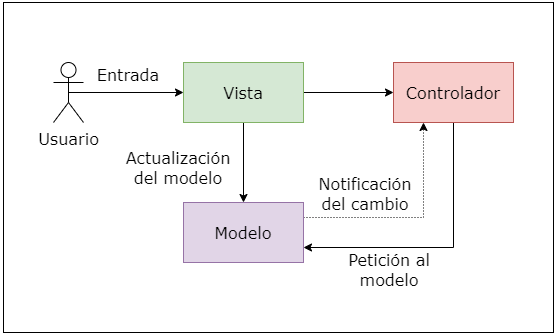


Figura 4. Modelo Vista Controlador

# PARTE PRÁCTICA

## CODIFICACIÓN VIDEO 173

Para esta práctica se nos propone un caso de estudio el cual nos menciona el siguiente enunciado.

El siguiente aplicativo permitirá transferencias bancarias mediante hilos, para lo cual deberá cumplir con las siguientes especificaciones:

* En el banco existirá 100 cuentas bancarias las mismas que contarán con un saldo inicial de 2000 $
* Ya que existen 100 cuentas con 2000 $, el saldo total de todas las cuentas o generalmente la cuenta total del banco deberá ser de 200 000 $
* Al momento de iniciar la transferencia cada cuenta deberá realizar una transacción hacia otra cuenta, tomar en cuenta que la cantidad a transferir debe ser mayor a saldo actual de la cuenta, además el saldo total del banco siempre deberá ser de 200 000 $
* Para realizar cada transferencia se deberá usar hilos

**Paquete modelo**

Especificado lo anterior, para la realización de esta práctica se utilizará el patrón MVC, para lo cual se creará en primer lugar el modelo del banco.

Dentro de la clase BancoModelo, se declara un arreglo de tipo double para definir las cuentas del banco y su respectivo constructor, mencionando que se declara 100 espacios por el hecho de que el requerimiento nos menciona que se tienen 100 cuentas.

Text

Description automatically generated with medium confidence

Figura . Codificación clase BancoModelo

**Paquete Controlador**

Luego se procede a crear el controlador denominado BancoControlador y del cual obtendremos 3 métodos que servirán como base para la realización del proyecto.

Como primer método encontramos el de acreditarSaldoCuentas que toma 2 parametros para poder funcionar, los cuales son un modelo de banco y un saldo como valor double, este a su vez declara un arreglo de 100 espacios para las cuentas acreditadas y mediante un ciclo for asigna a cada cuenta el valor especificado en el parámetro pasado.

Text

Description automatically generated

Figura . Codificación método acreditarSaldoCuentas

Como segundo método tenemos el de transferirCuentas que solicita 4 parámetros, los cuales son un modelo de banco, un valore entero para la cuenta de origen, un valor entero para la cuenta de destino y una cantidad de tipo double para el valor de transacción.

Realiza una comparación para verificar si la cuenta de origen posee más dinero que la cantidad especificada e imprime el hilo actual de ejecución, después se resta la cantidad especificada a la cuenta de origen y se le asigna a la cuenta de destino para finalmente obtener el saldo total de las cuentas.

Text

Description automatically generated

Figura . Codificación método transferirEntreCuentas

Como último método se encuentra el de obtenerSaldoTotal() que se utiliza en el método antes descrito, este método sirve para obtener el saldo total que posee una cuenta y recibe como parámetro a un modelo de banco.

Text

Description automatically generated

Figura . Codificación método ObtenerSaldoTotal

**Main**

Finalmente, en la función principal no se realiza nada por el hecho de que en esta parte de la practica aún no se la desarrolla.

## CODIFICACIÓN VIDEO 174

Para la realización de esta práctica se seguirá trabajando sobre la base ya realizada en la práctica anterior con algunas modificaciones en la parte del controlador y la función principal.

**Paquete Controlador**

Se creará un nuevo controlador denominado TransferenciasControlador que además implementa el método Runnable, para ello se declaran variables de tipo modelo banco, controlador banco, un int con el número de cuenta de origen y un double con el número de valor máximo a transferir.

Text

Description automatically generated

Figura . Codificación clase TransferenciasControlador

Además, en el método run se debe escribir las condiciones para realizar las transferencias, asignando para efectos de la practica un valor al azar multiplicado por 100 para elegir la cuenta de destino y una cantidad a transferir que de igual manera es un al azar multiplicado por el valor máximo permitido.

Texto

Descripción generada automáticamente

Figura . Codificación método run para el proceso de transferencia

**Main**

Dentro la función principal se declara un modelo de banco, un controlador de banco al cual se le envía la función acreditarSaldoCuentas y se le pasa el parámetro del banco con el que se va a trabajar y el saldo que va a tener las cuentas, luego se le pasa un ciclo for declarando un controlador de transferencia con el modelo de banco, controlador de banco, cuenta de origen y un valor máximo.

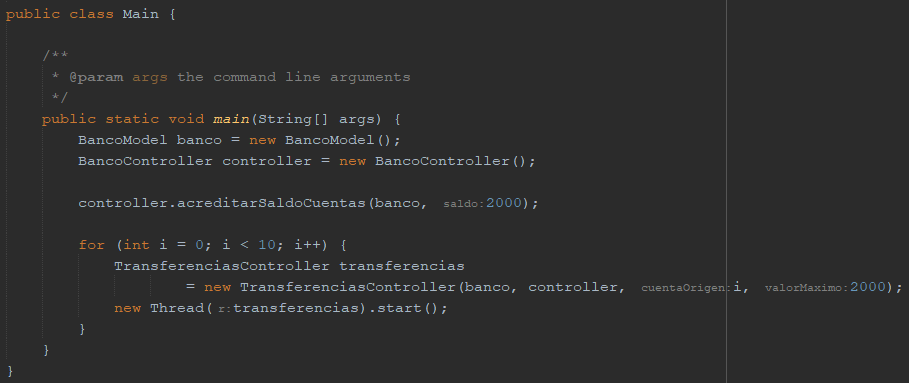


Figura . Codificación clase main con el método principal

Al momento de ejecutar este proyecto se puede observar que las transacciones fueron realizadas correctamente, sin embargo, existe un problema ya que los hilos no se encuentran sincronizados por lo que en el saldo total existe un desfase ya que como ingresan varios hilos a la vez, dos o más hilos pueden estar realizando una transferencia a la misma cuenta y por lo tanto descontar un valor que no toma en cuenta la función para obtener valores, lo cual se resuelve en la siguiente práctica.

## CODIFICACIÓN VIDEO 175

Para la práctica, se trabajará sobre el proyecto ya realizado, a diferencia de que para corregir el error que se dio en la práctica anterior se hará uso de la función lock y unlock, pertenecientes a la clase ReentrantLock, que sirven para bloquear y desbloquear líneas de código con la finalidad de que sean accedidas solo por un hilo a la vez, entonces en nuestro proyecto tiene la función de evitar que varios hilos accedan al mismo tiempo y eliminar errores de dinero entre cuentas.

**Paquete Controlador**

El cambio más significativo que se realiza es hacer uso de las funciones antes mencionadas dentro de la función transferirEntreCuentas y crear un cierreBanco.lock() al inicio de la función y un cierreBanco.unlock() antes de finalizar la función.

Text

Description automatically generated

Figura . Codificación clase BancoControlador

## CODIFICACIÓN VIDEO 176

En esta práctica, se modificarán las carpetas para completar el modelo MVC, para tal efecto se tiene que crear un paquete Vista, del cual se instanciara una clase llamada BancoVista.

**Paquete Vista**

Dentro del paquete vista se tiene la clase BancoVista, la cual va a contener lo siguiente:

Text

Description automatically generated

Figura . Codificación clase BancoVista

Como se puede observar, se declaran datos de tipos string con los códigos de color que se van a utilizar, después se declara la función imprimirErrorTransferencia que recibe un modelo de banco, una cuenta de origen y una cantidad, finalmente se mostrara un mensaje de color rojo informando sobre fondos insuficientes e información de la transacción.

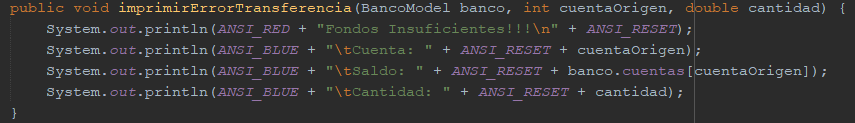


Figura . Codificación método imprimirErrorTransferencia

Luego se declara una función para imprimirErrorTransferenciaExito que nos muestra un mensaje en color verde de transferencia exitosa y la cuenta de origen, finalmente se crea una función imprimirDetalleTransferencia donde se muestran datos del hilo en ejecución y datos de la cuenta.

Texto

Descripción generada automáticamente

Figura . Codificación método imprimirTransferenciaExito

**Paquete Controlador**

Dentro de este paquete se realizará una modificación a la función de transferirEntreCuentas, haciendo uso de la función antes desarrollada denominada imprimirDetalleTransferencia.

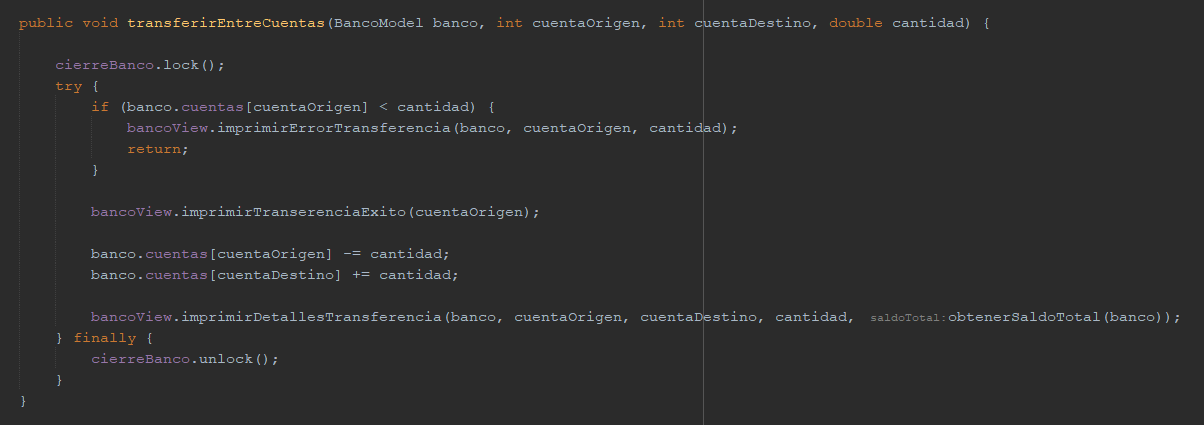


Figura 16. Codificación método transferirEntreCuentas

## CODIFICACIÓN VIDEO 177

En esta parte de la práctica, se procederá a solucionar un error que se daba en la anterior, el cual consiste en que no pudo realizar la transferencia y retorna a la vista, pero con ello habremos perdido un hilo lo que a lo largo significa tiempo de procesamiento, para solucionar este problema, hilos nos permite especificar condiciones en su proceso con el fin de que un hilo se quede bloqueado mientras se cumpla cierta condición.

**Paquete Controlador**

Se procederá a aumentar una condición dentro de la clase BancoControlador, para definir la condición que el hilo esperará.

Text

Description automatically generated

Figura . Codificación clase BancoControlador

Luego de realizada la declaración de la condición, se procede a aplicar método await que lo que realizara las siguientes funciones: si un hilo entra a realizar la transferencia, y cumple con la condición, lo que hará es quedar bloqueado y esperando a realizar un acción por lo que permitirá desbloquear el fragmento de código ya que lo tenía bloqueado para que pueda ingresar otro hilo, supongamos que el hilo que entró pudo haber realizado una transferencia al hilo anterior por lo que el saldo de su cuenta ya sería superior al cantidad máxima a transferir, entonces el hilo se activará nuevamente y realizaría el proceso hasta finalizarlo, antes de finalizar el proceso el hilo deberá informar a los demás hilos que se encuentran esperando, si ya no se cumple su condición para ello utilizamos el método signalAll.

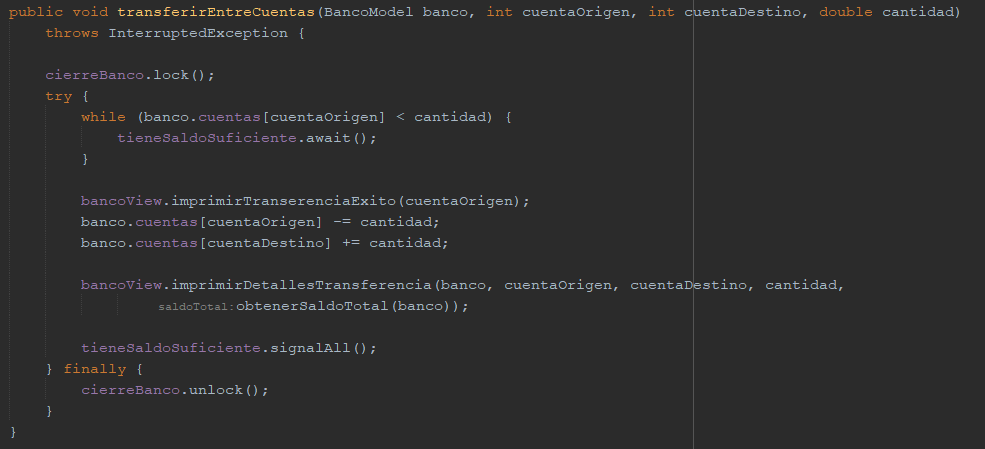


Figura . Codificación método transferirEntreCuentas

## CODIFICACIÓN VIDEO 178

Ya solucionado el problema que se venía acarreando desde practicas anteriores, se busca otra forma de solucionarlo, esto se logra de una forma más sencilla utilizando la palabra reservada *synchronized*. Esto lleva a reemplazar los métodos lock y unlock, además de eliminar la condición.

Por lo demás, se mantiene la misma estructura cambiando fragmentos mínimos de código y manteniendo los métodos wait y notifyAll para esperar y notificar a todos los hilos, por lo que funcionaran de la misma manera que las practicas anteriores.

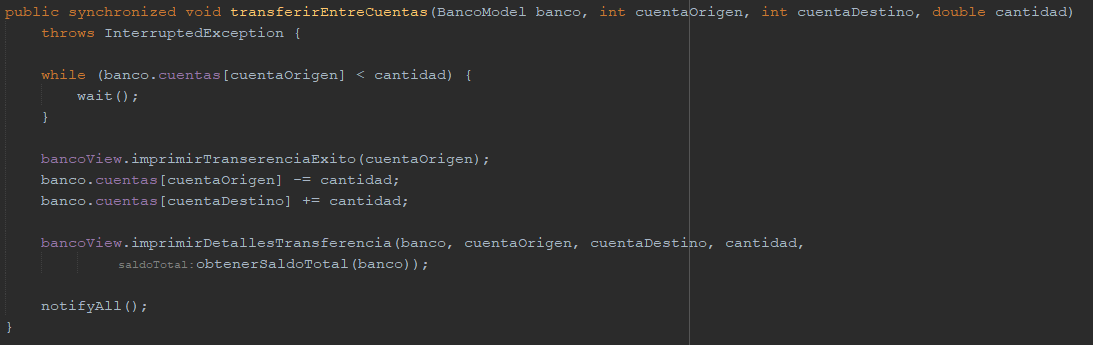


Figura . Codificación método transferirEntreCuentas

A pesar de que este método es más sencillo de utilizar, tiene una desventaja que requiere ser estudiada para ver si afecta de manera significativa al proyecto, esta desventaja trata de que este método no nos permite establecer más de una condición para que los hilos puedan esperar, es decir, se ejecuta una sola condición por fragmento.

## CODIFICACIÓN FINAL DE LA PRACTICA

package ec.edu.monster;

import ec.edu.monster.controlador.BancoControlador;

import ec.edu.monster.controlador.TransferenciasControlador;

import ec.edu.monster.modelo.BancoModelo;

/\*\*

\*

\* @author Grupo5

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

BancoModelo banco = new BancoModelo();

BancoControlador controlador = new BancoControlador();

controlador.acreditarSaldoCuentas(banco, 2000);

for (int i = 0; i < 100; i++) {

TransferenciasControlador transferencias

= new TransferenciasControlador(banco, controlador, i, 2000);

new Thread(transferencias).start();

}

}

}

Tabla . Codificación del archivo Main.java.

package ec.edu.monster.modelo;

public class BancoModelo {

public double[] cuentas;

public BancoModelo() {

this.cuentas = new double[100];

}

}

Tabla . Codificación del archivo BancoModelo.java

Tabla . Codificación del archivo BancoControlador.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.BancoModelo;

import ec.edu.monster.vista.BancoVista;

public class BancoControlador {

private BancoVista bancoVista = new BancoVista();

public void acreditarSaldoCuentas(BancoModelo banco, double saldo) {

double[] cuentasAcreditadas = new double[100];

for (int i = 0; i < cuentasAcreditadas.length; i++) {

cuentasAcreditadas[i] = saldo;

}

banco.cuentas = cuentasAcreditadas;

}

public synchronized void transferirEntreCuentas(BancoModelo banco, int cuentaOrigen, int cuentaDestino, double cantidad)

throws InterruptedException {

while (banco.cuentas[cuentaOrigen] < cantidad) {

wait();

}

bancoVista.imprimirTranserenciaExito(cuentaOrigen);

banco.cuentas[cuentaOrigen] -= cantidad;

banco.cuentas[cuentaDestino] += cantidad;

bancoVista.imprimirDetallesTransferencia(banco, cuentaOrigen, cuentaDestino, cantidad,

obtenerSaldoTotal(banco));

notifyAll();

}

public double obtenerSaldoTotal(BancoModelo banco) {

double saldoTotal = 0;

for (double saldo : banco.cuentas) {

saldoTotal += saldo;

}

return saldoTotal;

}

}

Tabla . Codificación del archivo TransferenciasControlador.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.BancoModelo;

public class TransferenciasControlador implements Runnable {

private BancoModelo \_banco;private BancoControlador \_controller;

private int \_cuentaOrigen; private double \_valorMaximo;

public TransferenciasControlador(BancoModelo banco, BancoControlador controller, int cuentaOrigen, double valorMaximo) {

this.\_banco = banco;

this.\_controller = controller;

this.\_cuentaOrigen = cuentaOrigen;

this.\_valorMaximo = valorMaximo;

}

@Override

public void run() {

while (true) {

int cuentaDestino = (int) (100 \* Math.random());

double cantidad = this.\_valorMaximo \* Math.random();

try {

this.\_controller.transferirEntreCuentas(

this.\_banco,

this.\_cuentaOrigen,

cuentaDestino,

cantidad

);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

try {

Thread.sleep((int) (Math.random() \* 10));

} catch (InterruptedException e) {

e.printStackTrace();

} } }

}

Tabla . Codificación del archivo BancoVista.java

package ec.edu.monster.vista;

import ec.edu.monster.modelo.BancoModelo;

public class BancoVista {

public static final String ANSI\_RESET = "\u001B[0m";

public static final String ANSI\_BLACK = "\u001B[30m";

public static final String ANSI\_RED = "\u001B[31m";

public static final String ANSI\_GREEN = "\u001B[32m";

public static final String ANSI\_YELLOW = "\u001B[33m";

public static final String ANSI\_BLUE = "\u001B[34m";

public static final String ANSI\_PURPLE = "\u001B[35m";

public static final String ANSI\_CYAN = "\u001B[36m";

public static final String ANSI\_WHITE = "\u001B[37m";

public void imprimirErrorTransferencia(BancoModelo banco, int cuentaOrigen, double cantidad) {

System.out.println(ANSI\_RED + "Fondos Insuficientes!!!\n" + ANSI\_RESET);

System.out.println(ANSI\_BLUE + "\tCuenta: " + ANSI\_RESET + cuentaOrigen);

System.out.println(ANSI\_BLUE + "\tSaldo: " + ANSI\_RESET + banco.cuentas[cuentaOrigen]);

System.out.println(ANSI\_BLUE + "\tCantidad: " + ANSI\_RESET + cantidad);

}

public void imprimirTranserenciaExito(int cuentaOrigen) {

System.out.println(ANSI\_GREEN + "Transferencia realizada\n" + ANSI\_RESET);

System.out.println(ANSI\_BLUE + "\tCuenta: " + ANSI\_RESET + cuentaOrigen);

}

public void imprimirDetallesTransferencia(

BancoModelo banco,

int cuentaOrigen,

int cuentaDestino,

double cantidad,

double saldoTotal

) {

System.out.println(Thread.currentThread() + "\n");

System.out.printf("%10.2f de %d para %d\n", cantidad, cuentaOrigen, cuentaDestino);

System.out.printf(ANSI\_BLUE + "Saldo Total:" + ANSI\_RESET + "%10.2f\n", saldoTotal);

}}

## FUNCIONALIDAD

### **ESTRUCTURA DE LA APLICACIÓN**

A screenshot of a computer

Description automatically generated with medium confidence

Figura 20. Estructura de la aplicación

### **EJECUCIÓN DE LA APLICACIÓN**

**Ejecución Video 173**

Graphical user interface, text

Description automatically generated

Figura 21. Ejecución video 173

**Ejecución Video 174**

Text

Description automatically generated

Figura 22. Ejecución video 174

**Ejecución Video 175**

Text

Description automatically generated

Figura . Ejecución video 175

**Ejecución Video 176**

Text

Description automatically generated

Figura . Ejecución video 176

**Ejecución Video 177**

Text

Description automatically generated

Figura . Ejecución video 177

**Ejecución Video 178**

Text

Description automatically generated

Figura . Ejecución video 178

# CONCLUSIONES

* La sincronización de hilos permite que varios hilos trabajen de forma simultánea sin problemas cuando operan o usan la misma fuente de datos, de esta forma se evitan errores de cálculos o en los procesos que se estén ejecutando.
* En java para lograr que varios hilos no usen o entren al mismo tiempo a un recurso se proporciona la palabra clave synchronized, de esta forma los hilos usaran la rutina o recurso de forma organizada, uno después del otro.
* El uso de los hilos permite resolver una o varias tareas de forma más rápida, pero para que el resultado sea el esperado y no se presenten inconvenientes se debe tomar en cuenta conceptos de sincronización e interrupción de hilos, de esta forma se puede controlar y establecer el orden en el que operan y evitar inconsistencias en el flujo de información y la salida de los datos o recursos esperados.

# RECOMENDACIONES

* El ejemplo propuesto funciona con un bucle infinito, por lo tanto, tomar precauciones en caso de que el computador presente problemas de rendimiento, para obtener los resultados esperados basta con un tiempo de ejecución de 15 segundos.
* La clase Thread incluye varios métodos deprecados, por lo tanto, se recomienda leer la documentación de Java para usar los métodos correctos a la hora de trabajar con hilos.

# REFERENCIAS

Blancarte, O. (29 de Marzo de 2017). *Concurrencia vs Paralelismo*. Obtenido de oscarblancarteblog: https://www.oscarblancarteblog.com/2017/03/29/concurrencia-vs-paralelismo/

Díaz, J. (2022). *Píldoras Informáticas*. Obtenido de YouTube: https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk

Nakayama, A. (2009). Guía práctica de estudio 12: Hilos. México.

Universidad de Alicante. (17 de Octubre de 2012). *Introdoucción al lenguaje Java y Eclipse*. Obtenido de JTECH: http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=Interrupci%C3%B3n%20de%20un%20hilo,-Los%20objetos%20de&text=interrupt()%20que%20permite%20al,para%20soportar%20su%20propia%20interrupci%C3%B3n.