
	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Universidad Politécnica Salesiana

Vicerrectorado Docente

Código del Formato:	GUIA-PRL-001
Versión:	VF1.0
Elaborado por:	Directores de Área del Conocimiento Integrantes Consejo Académico
Fecha de elaboración:	2016/04/01
Revisado por:	Consejo Académico
Fecha de revisión:	2016/04/06
Aprobado por:	Lauro Fernando Pesántez Avilés Vicerrector Docente
Fecha de aprobación:	2016/14/06
Nivel de confidencialidad:	Interno

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Descripción General

Propósito


El propósito del presente documento es definir un estándar para elaborar documentación de guías de práctica de laboratorio, talleres o centros de simulación de las Carreras de la Universidad Politécnica Salesiana, con la finalidad de lograr una homogenización en la presentación de la información por parte del personal académico y técnico docente.


Alcance

El presente estándar será aplicado a toda la documentación referente a informes de prácticas de laboratorio, talleres o centros de simulación de las Carreras de la Universidad Politécnica Salesiana.

Formatos


- Formato de Guía de Práctica de Laboratorio / Talleres / Centros de Simulación – para Docentes
- Formato de Informe de Práctica de Laboratorio / Talleres / Centros de Simulación – para Estudiantes


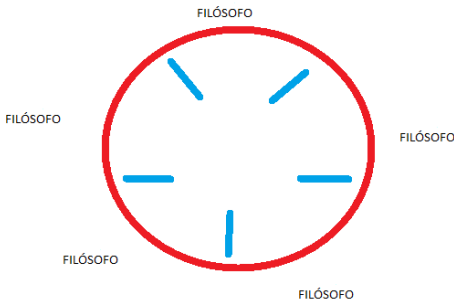
	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		


		FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES	
CARRERA: COMPUTACIÓN		ASIGNATURA: Programación Aplicada	
NRO. PRÁCTICA:	1	TÍTULO PRÁCTICA: Hilos en Java	
OBJETIVO: Identificar los cambios importantes de Java Diseñar e Implementar las nuevas técnicas de programación concurrente Entender cada una de las características de Thread en Java.			
INSTRUCCIONES (Detallar las instrucciones que se dará al estudiante):		1. Revisar los conceptos fundamentales de Thread en Java	
		2. Establecer como implementar Thread en Java	
		3. Implementar y diseñar los nuevos componentes de concurrencia	
		4. Realizar el informe respectivo según los datos solicitados.	
ACTIVIDADES POR DESARROLLAR (Anotar las actividades que deberá seguir el estudiante para el cumplimiento de la práctica)			
1. Revisar la teoría y conceptos de Thread en Java			
2. Diseñar e implementar las características de Java para generar una simulación 2D del siguiente enunciado:			
3. Probar y modificar el método para que nos permita cambiar el número de filósofos.			
4. Realizar práctica codificando con las nuevas características de Java, patrones de diseño, Thread, etc.			
5. Fecha de Entrega: 11 enero del 2021 23:55			
RESULTADO(S) OBTENIDO(S): Realizar procesos de Hilos en Java. Entender las aplicaciones de codificación de las nuevas características de concurrencia. Entender las funcionalidades de sincronización y manejo de grupo de Thread dentro de Java.			
CONCLUSIONES: Aprenden a trabajar en grupo dentro de plazos de tiempo establecidos, manejando el lenguaje de programación de Java.			
RECOMENDACIONES: Realizar el trabajo dentro del tiempo establecido.			

Docente / Técnico Docente: _____

Firma: _____

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES	
CARRERA: Computacion		ASIGNATURA: Programacion Aplicada	
NRO. PRÁCTICA:	04	TÍTULO PRÁCTICA: Hilos en Java	
OBJETIVO ALCANZADO: <ul style="list-style-type: none"> • Identificar los cambios importantes de Java • Diseñar e Implementar las nuevas técnicas de programación concurrente • Entender cada una de las características de Thread en Java. 			
ACTIVIDADES DESARROLLADAS			
<p>1. Revisar la teoria y conceptos de Thread en Java.</p> <p>2. Diseñar e implementar las características de Java para generar una simulación 2D del siguiente enunciado:</p>			
<p>Problema del Filósofo:</p> <p>En una mesa hay procesos que simulan el comportamiento de unos filósofos que intentan comer de un plato. Cada filósofo tiene un cubierto a su izquierda y uno a su derecha y para poder comer tiene que conseguir los dos. Si lo consigue, mostrará un mensaje en pantalla que indique «Filosofo 2 (numero) comiendo».</p> <p>Después de comer, soltará los cubiertos y esperará al azar un tiempo entre 1000 y 5000 milisegundos, indicando por pantalla «El filósofo 2 está pensando».</p> <p>En general todos los objetos de la clase Filósofo están en un bucle infinito dedicándose a comer y a pensar.</p> <p>Simular este problema en un programa Java que muestre el progreso de todos sin caer en problemas de sincronización a través de un método gráfico.</p> <div style="text-align: center; margin-top: 20px;">  </div>			

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

SOLUCION:

- Creación de paquetes con sus respectivas clases, métodos y vistas.

- **Ec.edu.ups.controlador**

- ControladorExcepciones

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

```
import utilidades.Log;
import java.util.logging.Level;
import java.util.logging.Logger;
import ec.edu.ups.vista.ControladoresGenerales;

/**
 * Esta clase maneja las excepciones que puedan ser provocadas por la ejecución de los hilos
 *
 * @author paul_
 */
//manejador de excepciones para toda la aplicación
public class ControladorExcepciones implements Thread.UncaughtExceptionHandler {
    //implementa el método uncaughtException()
    @Override
    public void uncaughtException(Thread t, Throwable e){
        System.out.printf("Thread que lanzó la excepción: %s \n", t.getName());
        //muestra en consola el hilo que produce la excepción
        e.printStackTrace();
        //muestra en consola la pila de llamadas

        // Añado este código para que los saque en el Log de la interface gráfica
        if (ControladoresGenerales.getjTextArea_Log()!=null) try {
            log.escribirLog("\n Thread que lanzó la excepción: " + t.getName() + "\n");
            log.escribirLog(e.toString() + "\n\n");
        } catch (InterruptedException ex) {
            Logger.getLogger(ControladorExcepciones.class.getName()).log(Level.SEVERE, null,
ex);
        }
        // Fin Añado este código
        // Fin Añado este código
    }
    // Añado este código: Un Constructor que recibe la clase Log que es la que escribe en la
interface gráfica
    /**
     *
     * @param log Clase Log que escribe en la Interface Gráfica
     */
    public ControladorExcepciones(Log log) {
        this.log = log;
    }
    // Variable que recibe la clase Log
    Log log;
    // Fin Añado este código
}
```

- ControladorPrincipal

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

```
import utilidades.Log;
import ec.edu.ups.modelo.Filosofos;
import ec.edu.ups.modelo.Tenedor;
import ec.edu.ups.modelo.Portero_del_Comedor;
import javax.swing.JLabel;
import javax.swing.JTextField;
import ec.edu.ups.vista.ControladoresGenerales;
import ec.edu.ups.vista.VentanaPrincipal;

/**
 * Esta clase genera las instancias de los 5 hilos Filósofos {@link Filósofos} ,
 * Estos 5 hilos se ejecutan de manera concurrente gestionados por monitores ,
 * La finalidad del programa es crear un algoritmo que permita que los filósofos coman y
 * piensen,
 * para ello, el problema se resuelve que siempre haya un filósofo como mínimo comiendo,
 * lo ideal que haya dos.
 * El algoritmo tiene un recurso compartido {@link Portero_del_Comedor} el cual deja pasar a
 * n-1 filósofos y después otro recurso {@link Tenedor} que es necesario tener por duplicado
 * para
 * que cada filósofo pueda comer. Además el algoritmo se completa creando una regla en la que
 * si un
 * filósofo no consigue el segundo tenedor en x tiempo aleatorio, éste debe abandonar su
 * turno de comida,
 * salir y ponerse a la cola.
 * Además si se selecciona crear un Log en el área de texto de la interface gráfica {@link
 * VentanaPrincipal}
 * todos los hilos de manera concurrente tendran que ir escribiendo en el log sus acciones,
 * con lo cual
 * puede relentizar un poco la ejecución del programa.
 *
 * @author paul_
 */
public class ControladorPrincipal {

    private JLabel[] jLabel_F = new JLabel[5];
    private JLabel[] jLabel_T = new JLabel[5];
    private JTextField[] jTextField_C = new JTextField[5];

    /**
     * Se generan las instancias de 5 de {@link Filósofos}, 5 {@link Tenedor},
     * una de {@link Portero_del_Comedor}, una de {@link Log} y una de {@link
     * ControladorExcepciones},
     * Y se ponen en funcionamiento los hilos filósofos.
     *
     * @param clase10Control Contiene todos los elemnetos de la interface gráfica
     */
    public ControladorPrincipal(ControladoresGenerales clase10Control) {
        this.jLabel_F = clase10Control.getjLabel_F();
        this.jLabel_T = clase10Control.getjLabel_T();
        this.jTextField_C = clase10Control.getjTextField_C();
    }
}
```

```
// Se crea el Array para contener las 5 instancias de Tenedores:
Tenedor[] tenedor = new Tenedor[5];
// Se crea el Array para contener las 5 instancias de Filósofos:
Filosofos[] filosofo = new Filsofos[5];
// Se crea una sola instancia de Portero_del_Comedor:
Portero_del_Comedor comensal = new Portero_del_Comedor();
// Se crea una sola instancia de Log:
Log log = new Log();
// Se crea la instancia del manejador de excepciones para los Thread:
ControladorExcepciones manejador=new ControladorExcepciones(log);

// Se crean las 5 instancias de Tenedores:
for(int i=0; i<tenedor.length; i++){
    tenedor[i] = new Tenedor(i);
}

// Se crean las 5 instancias de Filósofos:
for(int i=0; i<filosofo.length; i++){
    /* El filósofo coge el tenedor de la izquierda
    * y el de la derecha se contabiliza con el módulo(%)
    * porque cuando llega a cuatro el siguiente es cero
    */
    // Ahora al filósofo se le pasa: un ID, un tenedor Dercho, un tenedor Izdo, el
    comensal, los componentes gráficos correspondientes y un log
    filosofo[i] = new Filsofos(i, tenedor[i], tenedor[(i+1)%5], comensal,
    jLabel_F[i], jLabel_T[i], jLabel_T[(i+1)%5], log, jTextField_C[i]);
}

// Se echa a andar todos los Filósofos:
for(int i=0; i<filosofo.length; i++){
    filosofo[i].setUncaughtExceptionHandler(manejador);
    filosofo[i].start();
}
}
```

➤ **Ec.edu.ups.modelo**


▪ **Filosofos**

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

```
import java.awt.Color;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JLabel;
import javax.swing.JTextField;
import ec.edu.ups.controlador.ControladorPrincipal;
import utilidades.Log;
import ec.edu.ups.vista.ControladoresGenerales;
```

```
/**
```

```
 * Hilo Filósofo: se ejecuta de manera concurrente gestionados por monitores ,
```


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

* La finalidad del programa es crear un algoritmo que permita que los filósofos coman y
piensen,
* para ello, el problema se resuelve que siempre haya un filósofo como mínimo comiendo,
* lo ideal que haya dos.
* El algoritmo tiene un recurso compartido {@link Portero_del_Comedor} el cual deja pasar a
* n-1 filósofos y después otro recurso {@link Tenedor} que es necesario tener por duplicado
para
* que cada filósofo pueda comer. Además el algoritmo se completa creando una regla en la que
si un
* filósofo no consigue el segundo tenedor en x tiempo aleatorio, éste debe abandonar su
turno de comida,
* salir y ponerse a la cola.
*
* @author paul_
*/
public class Filósofos extends Thread {
    // Variable para generar números aleatorios:
    private Random random = new Random();
    // Variable para la ID del Filósofo:
    private int id;
    // Variables para los tenedores:
    private Tenedor izqda, dcha;
    // Variable para el comensal:
    private Portero_del_Comedor comensal;
    // Variables para los elementos gráficos:
    private JLabel jLabel_F, jLabel_T_dcha, jLabel_T_izqda;
    private Log log; // Para escribir en el Log
    private JTextField jTextField_C; // Contador de comidas
    // Variable pública y estática para que se pueda detener el método run() de esta clase:
    public static boolean finalizado = false;

    /**
     * Esta clase pone en marcha los hilos Filósofos
     * Método run() del Thread
     *
     * @param id ID del Filósofo
     * @param dcha Tenedor
     * @param izqda Tenedor
     * @param comensal Turno para comer
     * @param jLabel_F Elementos gráficos
     * @param jLabel_T_dcha Elementos gráficos
     * @param jLabel_T_izqda Elementos gráficos
     * @param log Para escribir en el Log
     * @param jTextField_C Contador de comidas
     */
    public Filósofos(int id, Tenedor dcha, Tenedor izqda, Portero_del_Comedor comensal,
        JLabel jLabel_F,
        JLabel jLabel_T_dcha, JLabel jLabel_T_izqda,
        Log log,
        JTextField jTextField_C){
        // Se asignan los valores recibidos a las variables
        this.id = id;
        this.dcha = dcha;
        this.izqda = izqda;

```


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```


this.comensal = comensal;
this.jLabel_F = jLabel_F;
this.jLabel_T_dcha = jLabel_T_dcha;
this.jLabel_T_izqda = jLabel_T_izqda;
this.log = log; // Puede ser null y por consiguiente no escribir el log
this.jTextField_C = jTextField_C;
}

/**
 * Método que se ejecuta indefinidamente, una por cada hilo creado en {@link
ControladorPrincipal}
 */
@Override
public void run(){
    while(true){ // Se repite infinitamente While

        try { // try / catch
            // Obtener el comensal para poder comer:
            comensal.cogerComensal(id, log);
            this.jLabel_F.setBackground(Color.PINK); // Componente gráfico
            // Obtener el Tenedor Derecho:
            dcha.cogerTenedor(id, log);
            this.jLabel_F.setBackground(Color.CYAN); // Componente gráfico
            this.jLabel_T_dcha.setBackground(Color.BLUE); // Componente gráfico
            // Obtener el Tenedor Izquierdo:
            if (!izqda.cogerTenedorIzqdo(id, log)){
                // -----
                // Si no se consigue el izquierdo: el filósofo tendra que volver a
casilla de salida y volver a obtener el comensal:
                System.out.println("El Filósofo " + (id+1) + " tendrá que soltar el
tenedor " + (id+1) + " por exceso de tiempo y salir a pensar.");
                // Siempre se valora si el log es distinto a null, si lo es se escribe en
la interface gráfica:
                if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El
Filósofo " + (id+1) + " tendrá que soltar el tenedor " + (id+1) + " por exceso de tiempo y
salir a pensar.");
                // Como no ha conseguido el Tenedor izquierdo suelta el derecho
dcha.soltarTenedor(id, log);
                this.jLabel_T_dcha.setBackground(Color.LIGHT_GRAY); // Componente
gráfico

                // Como no ha conseguido el Tenedor izquierdo suelta el comensal
comensal.soltarComensal(id, log);
                // Y ahora el Filósofo piensa *****
                System.out.println("El Filósofo " + (id+1) + " está pensando.");
                if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El
Filósofo " + (id+1) + " está pensando.");
                try {
                    // El tiempo que tarda el filósofo en pensar, entre 100 y 1000
milisegundos:
                    Filósofos.sleep(random.nextInt(1000) + 100);
                } catch (InterruptedException ex) {
                    System.out.println("Error. Descripción: " + ex.toString());


```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

        if (ControladoresGenerales.getjTextArea_Log()!=null)
log.escribirLog("\n\n Error. Descripción: " + ex.toString() + "\n\n");
    }
    // Fin de Ahora el Filósofo piensa *****
    continue; // Se vuelve a poner en la casilla de salida y volver a obtener
el comensal.
    // -----
}
// Si ha conseguido el Tenedor Izquierdo. El filósofo sigue adelante:
this.jLabel_T_izqda.setBackground(Color.BLUE); // Componente gráfico
// Y ahora el Filósofo come
=====
        this.jLabel_F.setBackground(Color.ORANGE); // Componente gráfico
        this.jLabel_F.setForeground(Color.BLUE); // Componente gráfico
        jTextField_C.setText(" " + (++ControladoresGenerales.filoCount[id])); //
Su contador de comidas incrementa una unidad.
        System.out.println("El Filósofo " + (id+1) + " está comiendo.");
        if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El
Filósofo " + (id+1) + " está comiendo.");
        // Simular el tiempo que tarda el filósofo en comer, entre 0.5 y 1
segundos:
        try {
            sleep(random.nextInt(1000) + 500);
        } catch (InterruptedException ex) {
            System.out.println("Error. Descripción: " + ex.toString());
            if (ControladoresGenerales.getjTextArea_Log()!=null)
log.escribirLog("\n\n Error. Descripción: " + ex.toString() + "\n\n");
        } // Fin de Simular el tiempo que tarda el filósofo en comer, entre 0.5 y
1 segundos // Fin de Simular el tiempo que tarda el filósofo en comer, entre 0.5 y 1 segundos
        this.jLabel_F.setBackground(Color.WHITE); // Componente gráfico
        this.jLabel_F.setForeground(Color.BLACK); // Componente gráfico
        // Fin de Ahora el Filósofo come
=====
        // Suelta el Tenedor izquierdo:
        izqda.soltarTenedor(id, log);
        this.jLabel_T_izqda.setBackground(Color.LIGHT_GRAY); // Componente
gráfico
        // Suelta el Tenedor derecho:
        dcha.soltarTenedor(id, log);
        this.jLabel_T_dcha.setBackground(Color.LIGHT_GRAY); // Componente gráfico
        // Suelta el comensal:
        comensal.soltarComensal(id, log);
        // Ahora el Filósofo piensa
*****
        System.out.println("El Filósofo " + (id+1) + " está pensando.");
        if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El
Filósofo " + (id+1) + " está pensando.");
        // El tiempo que tarda el filósofo en pensar, entre 100 y 1000 milisegundos:
        try {
            Filósofos.sleep(random.nextInt(1000) + 100);
        } catch (InterruptedException ex) {
            System.out.println("Error. Descripción: " + ex.toString());

```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

        if (ControladoresGenerales.getjTextArea_Log()!=null)
log.escribirLog("\n\n Error. Descripción: " + ex.toString() + "\n\n");
    }
    // Fin de Ahora el Filósofo piensa
*****
    // Fin de Ahora el Filósofo piensa
*****
    } catch (InterruptedException ex) {
        ex.printStackTrace();
        System.err.println("Se ha producido un error. Descripción: " +
ex.toString());
        try {
            if (ControladoresGenerales.getjTextArea_Log()!=null)
log.escribirLog("\n\n Se ha producido un error. Descripción: " + ex.toString() + "\n\n");
        } catch (InterruptedException ex1) {
            Logger.getLogger(Filosofos.class.getName()).log(Level.SEVERE, null, ex1);
        }
    } // Fin del try / catch // Fin del try / catch // Fin del try / catch // Fin del
try / catch

    if(finalizado){ // Si se ha pulsado el botón en la interface de 'Pausar' (public
static boolean finalizado = true):
        break; // Se sale
    }

    } // Fin de Se repite infinitamente While

    // Se ha pulsado el botón de la interface 'Pausar' (public static boolean finalizado
= true):
    System.out.println("La cena ha terminado para todos: El Filósofo " + (id+1) +" se ha
puesto a pensar.\n\nPulsar Iniciar para continuar.\n\n");
    try {
        if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" La cena ha
terminado para todos: El Filósofo " + (id+1) +" se ha puesto a pensar.\n\n Pulsar Iniciar
para continuar.\n\n");
    } catch (InterruptedException ex) {
        Logger.getLogger(Filosofos.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}

    ■ PorteroDelComedor
package ec.edu.ups.modelo;

import utilidades.Log;
import ec.edu.ups.vista.ControladoresGenerales;

/**
 * Recurso compartido por los hilos Filósofos,
 * Se crea una única instancia con n-1 comensales donde n es el número de filósofos
 *
 * @author paul_
 */
public class Portero_del Comedor {

```

```
private int comensal = 4; // Es el número de comensales total de filósofos menos 1


/**
 * Monitor para coger un comensal de los 4 y poder seguir el proceso de ejecución de los
 * filósofos.
 *
 * @param id_f ID del filósofo
 * @param log Clase Log para escribir el log en la interface gráfica
 * @throws InterruptedException Posibles errores
 */
public synchronized void cogerComensal(int id_f, Log log) throws InterruptedException{
    while(comensal==0){ // Si no hay comensales libres toca esperar
        this.wait();
    }
    System.out.println("El Filósofo " + (id_f+1) + " es el comensal " + comensal);
    // Siempre se valora si el log es distinto a null, si lo es se escribe en la interface
    gráfica:
    if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El Filósofo "
+ (id_f+1) + " es el comensal " + comensal);
    comensal--; // Conteo de comensales
}

/**
 * Monitor para soltar un comensal de los 4 y poder seguir el proceso de ejecución de los
 * filósofos.
 *
 * @param id_f ID del filósofo
 * @param log Clase Log para escribir el log en la interface gráfica
 * @throws InterruptedException Posibles errores
 */
public synchronized void soltarComensal(int id_f, Log log) throws InterruptedException{
    comensal++; // Conteo de comensales
    System.out.println("El Filósofo " + (id_f+1) + " ya NO es el comensal " + comensal);
    // Siempre se valora si el log es distinto a null, si lo es se escribe en la interface
    gráfica:
    if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El Filósofo "
+ (id_f+1) + " ya NO es el comensal " + comensal);
    this.notify(); // Notificación al siguiente de que hay comensal disponible.
}
}

    ■ Tenedor
package ec.edu.ups.modelo;

import java.util.Random;
import utilidades.Log;
import ec.edu.ups.vista.ControladoresGenerales;

/**
 * Recurso compartido por los hilos Filósofos,
 * Se crean 5 instancias con n-1 comensales donde n es el número de filósofos con el total de
 * los tenedores,
 * Cada tenedor tiene su ID y a cada filósofo le corresponde 2 tenedores concretos.
 */
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		


```

* @author paul_
*/
public class Tenedor {
    // Variable para generar números aleatorios:
    private Random random = new Random();
    // ID del Tenedor
    private int id;
    // Está ocupado el tenedor o no?:
    private boolean libre = true;

    /**
     * Constructo de la clase Tenedor
     *
     * @param id ID del Tenedor
     */
    public Tenedor(int id){
        this.id = id;
    }

    // Crear métodos synchronized => Monitores
    // Solo puede acceder un Thread a la vez.
    /**
     * Monitor para coger el tenedor derecho y poder seguir el proceso de ejecución de los
     filósofos.
     *
     * @param id_f ID del Filósofo
     * @param log Clase Log para escribir el log en la interface gráfica
     * @throws InterruptedException Posibles errores
     */
    public synchronized void cogerTenedor(int id_f, Log log) throws InterruptedException{
        while(!libre)
            this.wait();
        System.out.println("El Filósofo " + (id_f+1) + " coge el tenedor " + (id+1));
        // Siempre se valora si el log es distinto a null, si lo es se escribe en la interface
gráfica:
        if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El Filósofo "
+ (id_f+1) + " coge el tenedor " + (id+1));
        libre = false;
    }
    /**
     * Monitor para coger el tenedor izquierdo y poder seguir el proceso de ejecución de los
     filósofos,
     * Pero si no consigue cogerlo en un tiempo x retornará false y tendrá que salir a pensar
     y no podrá comer,
     * Tendrá que volver a empezar el proceso de comer.
     *
     * @param id_f ID del Filósofo
     * @param log Clase Log para escribir el log en la interface gráfica
     * @return Boolean True: puede continuar tiene los dos tenedores Y False: tiene que
     volver a empezar el proceso
     * @throws InterruptedException Posibles errores
     */
    public synchronized boolean cogerTenedorIzqdo(int id_f, Log log) throws
InterruptedException{

```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```


while(!libre){
    this.wait(random.nextInt(1000) + 500); // Sólo espera aleatoriamente entre 0.5 y
1 seg y si no, retorna false
    return false;
}
System.out.println("El Filósofo " + (id_f+1) + " coge el tenedor " + (id+1));
// Siempre se valora si el log es distinto a null, si lo es se escribe en la interface
gráfica:
    if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El Filósofo "
+ (id_f+1) + " coge el tenedor " + (id+1));
    libre = false;
    return true;
}
/**
 * Monitor para soltar un tenedor izquierdo o derecho y salir a pensar.
 *
 * @param id_f ID del Filósofo
 * @param log Clase Log para escribir el log en la interface gráfica
 * @throws InterruptedException Posibles errores
 */
public synchronized void soltarTenedor(int id_f, Log log) throws InterruptedException {
    libre = true;
    System.out.println("El Filósofo " + (id_f+1) + " suelta el tenedor " + (id+1));
    // Siempre se valora si el log es distinto a null, si lo es se escribe en la interface
gráfica:
    if (ControladoresGenerales.getjTextArea_Log()!=null) log.escribirLog(" El Filósofo "
+ (id_f+1) + " suelta el tenedor " + (id+1));
    this.notify();
}
}

➤ Ec.edu.ups.test
    ▪ ClaseTest
package ec.edu.ups.test;

import ec.edu.ups.modelo.Filosofos;
import ec.edu.ups.modelo.Portero_del_Comedor;
import ec.edu.ups.modelo.Tenedor;
import javax.swing.JFrame;
import ec.edu.ups.vista.VentanaPrincipal;

/**
 * Programa 2 de la tarea 02 de PSP de DAM, Problema de la cena de los Filósofos,
 * Se resuelve utilizando un Portero del Comedor con n-1 plazas o turnos en relación al
número
 * de Filósofos y si un filósofo tarda en conseguir el segundo tenedor pierde el turno.
 *
 * El programa ejecuta los hilos de forma concurrente mediante monitores.
 *
 * Paquete: {@link vistas} contiene todas las interfaces gráficas y sus clases relacionadas
con la interface.
 * Paquete: {@link utilidades} contiene todas las clases que aportan utilidad a la ejecución
del programa.

```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

* Paquete: {@link logica} contiene todas las clases que tiene la estructura principal del
programa.
* Paquete: {@link filosofos} contiene todas las clases relacionadas con los hilos {@link
Filosofos}.
* y con los recursos {@link Portero_del_Comedor} y con los {@link Tenedor}.
* Paquete: images contiene todas las imágenes del programa.
*
*
* @author paul_
*/
public class ClaseTest {

    /**
     * @param args No contempla argumentos
     */
    public static void main(String[] args) {
        /**
         * Clase que crea la interface gráfica del programa y el arranque del mismo:
         */
        JFrame vl0control=new VentanaPrincipal();

    }

}

➤ Ec.edu.ups.vista
    ▪ VistaGeneral
package ec.edu.ups.vista;

import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 * Esta clase contiene todos los elementos de la Interface Gráfica para que
 * puedan ser manejados desde las otras clases del programa.
 *
 * @author paul_
 */
public class ControladoresGenerales {
    // Se crean los arrays de elementos y los elementos:
    private JLabel[] jLabel_F = new JLabel[5];
    private JLabel[] jLabel_T = new JLabel[5];
    private static JTextArea jTextArea_Log;
    private JTextField[] jTextField_C = new JTextField[5];
    private static int[] filoCount = new int[5];
    public ControladoresGenerales(JLabel jLabel_Filo01, JLabel jLabel_Filo02, JLabel
jLabel_Filo03, JLabel jLabel_Filo04, JLabel jLabel_Filo05,
        JLabel jLabel_Ten01, JLabel jLabel_Ten02, JLabel jLabel_Ten03, JLabel
jLabel_Ten04, JLabel jLabel_Ten05,
        JTextArea jTextArea_Log,

```

```
JTextField jTextField_Cont01, JTextField jTextField_Cont02, JTextField  
jTextField_Cont03, JTextField jTextField_Cont04, JTextField jTextField_Cont05) {
```

```
// Se asignan los valores a las variables:
```

```
this.jLabel_F[0] = jLabel_Filo01;  
this.jLabel_F[1] = jLabel_Filo02;  
this.jLabel_F[2] = jLabel_Filo03;  
this.jLabel_F[3] = jLabel_Filo04;  
this.jLabel_F[4] = jLabel_Filo05;
```

```
this.jLabel_T[0] = jLabel_Ten01;  
this.jLabel_T[1] = jLabel_Ten02;  
this.jLabel_T[2] = jLabel_Ten03;  
this.jLabel_T[3] = jLabel_Ten04;  
this.jLabel_T[4] = jLabel_Ten05;
```

```
this.jTextArea_Log = jTextArea_Log;
```

```
this.jTextField_C[0] = jTextField_Cont01;  
this.jTextField_C[1] = jTextField_Cont02;  
this.jTextField_C[2] = jTextField_Cont03;  
this.jTextField_C[3] = jTextField_Cont04;  
this.jTextField_C[4] = jTextField_Cont05;
```

```
ControladoresGenerales.filoCount[0] = 0;  
ControladoresGenerales.filoCount[1] = 0;  
ControladoresGenerales.filoCount[2] = 0;  
ControladoresGenerales.filoCount[3] = 0;  
ControladoresGenerales.filoCount[4] = 0;
```

▪ ModeladoDeLaVentana

```
package ec.edu.ups.vista;
```

```
import java.awt.Image;  
import java.awt.Toolkit;  
import javax.swing.JFrame;
```

```
public ModeladoDeLaVentana(String title, String icon){  
    this.setTitle(title);  
    this.setSize(700, 700);  
    this.setLocationRelativeTo(null); // para centrar la pantalla en la ventana  
    this.setResizable(false); // permite no maximizar la pantalla  
    //Image icono = Toolkit.getDefaultToolkit().getImage("src/images/" + icon); // para  
NetBeans  
    Image icono = Toolkit.getDefaultToolkit().getImage("images/" + icon); // para  
Ejecución  
    this.setIconImage(icono);  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.setVisible(true);  
}
```


▪ VentanaPrincipal

```
package ec.edu.ups.vista;

import ec.edu.ups.modelo.Filosofos;
import java.awt.Color;
import static java.lang.Thread.sleep;
import javax.swing.JFrame;
import ec.edu.ups.controlador.ControladorPrincipal;

/**
 * Esta clase contiene la interface gráfica de la aplicación
 * Hereda de {@link ModeladoDeLaVentana}
 * Contiene todos los elementos gráficos del programa y los botones
 * que ejecutan el programa.
 *
 * @author paul_
 */
public class VentanaPrincipal extends ModeladoDeLaVentana {

    /**
     * Creates new form Vista10Control
     */
    public VentanaPrincipal() {
        super("La Cena de los Filósofos", "icono.png");
        initComponents();

        //Botón Finalizar:
        this.jButton_Finalizar.setEnabled(false);

        // Contadores de Comida:
        this.jTextField_Cont01.setText(" 0");
        this.jTextField_Cont02.setText(" 0");
        this.jTextField_Cont03.setText(" 0");
        this.jTextField_Cont04.setText(" 0");
        this.jTextField_Cont05.setText(" 0");

        // Filósofos Configuración inicial:
        this.jLabel_Filo01.setOpaque(true);
        this.jLabel_Filo01.setBackground(Color.WHITE);
        this.jLabel_Filo02.setOpaque(true);
        this.jLabel_Filo02.setBackground(Color.WHITE);
        this.jLabel_Filo03.setOpaque(true);
        this.jLabel_Filo03.setBackground(Color.WHITE);
        this.jLabel_Filo04.setOpaque(true);
        this.jLabel_Filo04.setBackground(Color.WHITE);
        this.jLabel_Filo05.setOpaque(true);
        this.jLabel_Filo05.setBackground(Color.WHITE);

        // Tenedores Configuración inicial:
        this.jLabel_Ten01.setText(" 1 ");
        this.jLabel_Ten01.setOpaque(true);
        this.jLabel_Ten01.setBackground(Color.LIGHT_GRAY);
        this.jLabel_Ten01.setForeground(Color.WHITE);
    }
}
```

```
this.jLabel_Ten02.setText(" 2 ");
this.jLabel_Ten02.setOpaque(true);
this.jLabel_Ten02.setBackground(Color.LIGHT_GRAY);
this.jLabel_Ten02.setForeground(Color.WHITE);

this.jLabel_Ten03.setText(" 3 ");
this.jLabel_Ten03.setOpaque(true);
this.jLabel_Ten03.setBackground(Color.LIGHT_GRAY);
this.jLabel_Ten03.setForeground(Color.WHITE);

this.jLabel_Ten04.setText(" 4 ");
this.jLabel_Ten04.setOpaque(true);
this.jLabel_Ten04.setBackground(Color.LIGHT_GRAY);
this.jLabel_Ten04.setForeground(Color.WHITE);

this.jLabel_Ten05.setText(" 5 ");
this.jLabel_Ten05.setOpaque(true);
this.jLabel_Ten05.setBackground(Color.LIGHT_GRAY);
this.jLabel_Ten05.setForeground(Color.WHITE);

// Leyenda de colores:
this.jLabel_est01.setText(" ");
this.jLabel_est01.setOpaque(true);
this.jLabel_est01.setBackground(Color.PINK);

this.jLabel_est02.setText(" ");
this.jLabel_est02.setOpaque(true);
this.jLabel_est02.setBackground(Color.CYAN);

this.jLabel_est03.setText(" ");
this.jLabel_est03.setOpaque(true);
this.jLabel_est03.setBackground(Color.ORANGE);


this.jLabel_est04.setText(" ");
this.jLabel_est04.setOpaque(true);
this.jLabel_est04.setBackground(Color.WHITE);

this.jLabel_est05.setText(" ");
this.jLabel_est05.setOpaque(true);
this.jLabel_est05.setBackground(Color.BLUE);

this.jLabel_est06.setText(" ");
this.jLabel_est06.setOpaque(true);
this.jLabel_est06.setBackground(Color.LIGHT_GRAY);
}

private void jButton_IniciarActionPerformed(java.awt.event.ActionEvent evt) {
    Filósofos.finalizado = false; // Para poder pausar o reiniciar.

    // Clase que contiene todos los objetos de la interface y que sera pasada por
    // parámetro a la clase principal
    ControladoresGenerales clase10Control = new ControladoresGenerales(jLabel_Filo01,
        jLabel_Filo02, jLabel_Filo03, jLabel_Filo04, jLabel_Filo05,
        jLabel_Ten01, jLabel_Ten02, jLabel_Ten03, jLabel_Ten04, jLabel_Ten05,
        jTextArea_Log,
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

jTextField_Cont01, jTextField_Cont02, jTextField_Cont03, jTextField_Cont04,
jTextField_Cont05);

// Clase ControladorPrincipal con la lógica del programa:
// Recibe por parámetro la clase ControladoresGenerales que contiene todos los
elemntos de la interface
ControladorPrincipal principal = new ControladorPrincipal(clase10Control);


this.jButton_Finalizar.setEnabled(true);
this.jButton_Iniciar.setEnabled(false);
this.jCheckBox_Log.setEnabled(false);
}
/**
 * Dispara la pausa de la ejecución
 * @param evt Recibe el evento de hacer click sobre el botón
 */
private void jButton_FinalizarActionPerformed(java.awt.event.ActionEvent evt) {
    Filósofos.finalizado = true;
    try {
        sleep(3000);
    } catch (InterruptedException ex) {
        System.out.println("Error. Descripción: " + ex.toString());
    }
    this.jButton_Iniciar.setEnabled(true);
    this.jButton_Finalizar.setEnabled(false);
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

private void jButton_ResetActionPerformed(java.awt.event.ActionEvent evt) {
    Filósofos.finalizado = true;
    try {
        sleep(3000);
    } catch (InterruptedException ex) {
        System.out.println("Error. Descripción: " + ex.toString());
    }
    this.dispose();
    JFrame v10control=new VentanaPrincipal();
}

private void jCheckBox_LogItemStateChanged(java.awt.event.ItemEvent evt) {
    if (jCheckBox_Log.isSelected()) {
        this.jTextArea_Log.append("\n\n Atención:"
            + "\n Acabas de seleccionar crear un log en esta pantalla. Tienes que tener
en cuenta"
            + "\n que los diferentes procesos tienen que escribir lo que hacen antes de
continuar"
            + "\n con lo cual puede relentizar la ejecución del programa, ya que todos
tiene que usar"
            + "\n este mismo recurso."
            + "\n Utiliza el Log sólo para fines de comprobación.\n\n"
            + "\n Pulsa en Iniciar\n\n\n");
    }
}

```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

this.jTextArea_Log.setCaretPosition(this.jTextArea_Log.getDocument().getLength());
}
if (!jCheckBox_Log.isSelected()) {
    this.jTextArea_Log.append("\n\n Atención:"
        + "\n Acabas de deseleccionar crear un log en esta pantalla. Esta es la
mejor"
        + "\n forma de ejecutar el programa. El algoritmo se desarrolla íntegro, sin
la escritura"
        + "\n de todos los procesos."
        + "\n Se debe usar el Log sólo para fines de comprobación.\n\n"
        + "\n Pulsa en Iniciar");
}

this.jTextArea_Log.setCaretPosition(this.jTextArea_Log.getDocument().getLength());
}

```

3. Probar y modificar el método para que nos permita cambiar el número de filósofos.
Ventana Principal de los Filósofos:



La Cena de los Filósofos

Filósofo 01, Filósofo 02, Filósofo 03, Filósofo 04, Filósofo 05

Log

Cuántas veces han comido:

Filósofo 1:

Filósofo 2:

Filósofo 3:

Filósofo 4:


Filósofo 5:

Controles

☐ Crear un log

Código de colores:

- Filósofo entra a comer
- Filósofo tiene un tenedor
- Filósofo está comiendo
- Filósofo está pensando
- Tenedor ocupado
- Tenedor libre

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

4. Realizar práctica codificando con las nuevas características de Java, patrones de diseño, Thread, etc.

5. **Fecha de Entrega:** 11 enero del 2021 23:55

RESULTADO(S) OBTENIDO(S):

- Realizar procesos de Hilos en Java.
- Entender las aplicaciones de codificación de las nuevas características de concurrencia.
- Entender las funcionalidades de sincronización y manejo de grupo de Thread dentro de Java.

CONCLUSIONES:

- Aprender a trabajar en grupo dentro de plazos de tiempo establecidos, manejando el lenguaje de programación de Java.

RECOMENDACIONES:

- Realizar el trabajo dentro del tiempo establecido.

Nombre de estudiante:

PAUL ALEXANDER GUAPUCAL CARDENAS

Firma de estudiante:

