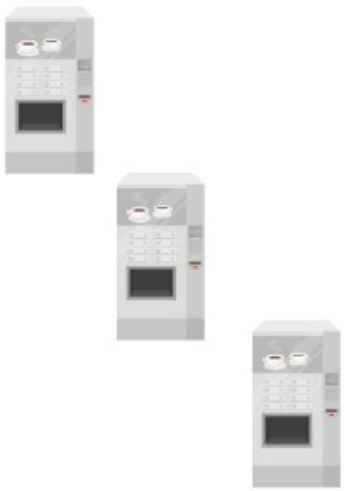
PROGRAMACIÓN DE SERVICIOS Y PROCESOS – TAREA 2

Ejercicio 01:

☕ Tarea: Desarrollar un programa en Java para gestionar una cafetería automatizada

📖 Descripción del Problema:

- El sistema debe contar con **3 máquinas de preparación de bebidas** para atender a los clientes.
- Cada cliente puede realizar un pedido de uno de los siguientes tipos:
    - CAFÉ
    - TÉ
    - CHOCOLATE CALIENTE
- Los clientes deben esperar en una **cola** hasta que una máquina esté disponible.
- Cuando una máquina esté libre, el cliente que está en la cabeza de la cola es asignado a esa máquina.
- La asignación de clientes a las máquinas debe garantizar que no haya más de un cliente utilizando la misma máquina al mismo tiempo.
- El tiempo de preparación de cada bebida debe variar dependiendo del tipo de bebida solicitado:
    - **CAFÉ:** 2000 ms
    - **TÉ:** 1500 ms
    - CHOCOLATE CALIENTE: 2500 ms
- Cada cliente debe indicar por pantalla:
    - En qué máquina está siendo atendido.
    - Qué bebida se le está preparando.
- Al finalizar, el programa debe mostrar cuántos pedidos de cada tipo (**CAFÉ**, **TÉ**, **CHOCOLATE CALIENTE**) ha preparado cada máquina.
- Se debe incluir un **registro de logs detallados** para facilitar la depuración y el seguimiento del estado del sistema. Los logs deben incluir:
    - Asignación de clientes a las máquinas.
    - Inicio y fin de preparación de cada bebida.
    - Cambios en la cola de clientes.



📊 Criterios de Calificación: Total 10 puntos

- **(0,5 puntos):** Implementación de la interfaz Runnable para simular las máquinas de preparación.
- **(1 punto):** Correcto uso del semáforo para controlar el acceso a las máquinas de preparación.
- **(1 punto):** Correcta asignación de clientes a las máquinas disponibles.
- **(0,5 puntos):** Simulación del tiempo de preparación de cada bebida, variando según el tipo de bebida solicitado.
- **(0,5 puntos):** Impresión del mensaje indicando qué máquina está atendiendo al cliente y qué bebida se está preparando.
- **(0,5 puntos):** Actualización correcta del conteo de pedidos por máquina y tipo de bebida.
- **(0,5 puntos):** Inicialización correcta de las máquinas disponibles y el semáforo.
- **(0,5 puntos):** Creación y lanzamiento de hilos para cada cliente.
- **(0,5 puntos):** Asignación aleatoria del tipo de bebida a cada cliente.
- **(0,5 puntos):** Correcta sincronización de acceso al mapa de conteo de pedidos por máquina.
- **(1 punto):** Impresión correcta y coherente del resultado final de los pedidos realizados por máquina y tipo de bebida.
- **(2 puntos):** Comprobación de que la concurrencia está manejada correctamente, sin condiciones de carrera ni bloqueos.
- **(0,5 puntos):** Código bien estructurado y organizado, uso adecuado de nombres de variables y métodos, con comentarios explicativos donde sea necesario.
- **(0,5 puntos):** Inclusión de un sistema de logs detallados que refleje el estado del sistema y facilite la depuración.

🛠️ Recursos necesarios para realizar la Tarea

- Lenguaje de programación: Java.
- **Entorno de desarrollo:** Eclipse, IntelliJ IDEA o Visual Studio Code.
- Conocimientos necesarios:
    - Manejo de hilos (Thread) e interfaz Runnable.
    - Uso de semáforos para sincronización.
    - Gestión de estructuras de datos concurrentes.
    - Buenas prácticas de programación concurrente.

💡 Consejos y recomendaciones

Se pueden usar estructuras de datos como Queue para gestionar la cola de clientes.

Asegúrate de manejar correctamente la concurrencia con semáforos o bloques sincronizados para evitar condiciones de carrera.

Implementa un sistema de logs que incluya eventos clave del programa para facilitar la depuración.

- Resultados obtenidos.

Nomenclatura del archivo comprimido: PSPXX_TareaZZ_apellido1_apellido2_nombre, donde XX corresponde a la unidad y ZZ al número de la tarea.

Por ejemplo, si la unidad es la 02 y la tarea es la 01, un estudiante llamado Juan Pérez López deberá nombrar el archivo así:

PSP02_Tarea01_Perez_Lopez_Juan.zip

Sube el archivo comprimido a la plataforma antes de la fecha de entrega.

🔧 Resultado de aprendizaje y criterios de evaluación

📌Desarrolla aplicaciones compuestas por varios hilos de ejecución analizando y aplicando librerías específicas del lenguaje de programación:

- **a)** Se han identificado situaciones en las que resulte útil la utilización de varios hilos en un programa.
- **b)** Se han reconocido los mecanismos para crear, iniciar y finalizar hilos.
- **c)** Se han programado aplicaciones que implementen varios hilos.
- **d)** Se han identificado los posibles estados de ejecución de un hilo y programado aplicaciones que los gestionen.
- **e)** Se han utilizado mecanismos para compartir información entre varios hilos de un mismo proceso.
- **f)** Se han desarrollado programas formados por varios hilos sincronizados mediante técnicas específicas.
- **h)** Se han depurado y documentado los programas desarrollados.

Resolución de la tarea:

Se ha creado una aplicación que simulará una cafetería en la que entran clientes y realizan un pedido entre tres tipos diferentes de Bebida (Café, Té y Chocolate caliente).

Para la realización del código que forma la aplicación se han creado las diferentes clases:

TipoBebida:

En esta clase se ha sustituido la palabra reservada "class" que declara la clase por un "enum" para convertir la clase en una enumeración y así poder definir un conjunto que será constante y está relacionado entre sí.

Ahora como se ha cambia el tipo de clase, ya podemos definir los tres tipos de bebidas.

- CAFE(2000): se le asigna como parámetro el valor 2000, que será el tiempo de preparación de la bebida.
- TE(1500) : se le asigna como parámetro el valor 1500, que será el tiempo de preparación de la bebida.
- CHOCOLATE_CALIENTE(2500): se le asigna como parámetro el valor 2500, que será el tiempo de preparación de la bebida.

Esta clase contiene un variable que almacenará el tiempo de la preparación de la bebida. Además, encontramos un constructor para construir el objeto que recibe como parámetro el tiempo que contiene el tipo de bebida que hemos declarado. También hay un getTiempoPreparación(), que devuelve el valor del tiempo de preparación de la bebida.

Calse Registroslogs:

Esta clase va a gestionar los registros que generen las máquinas mientras realizan atienden a los clientes.

Atributos principales de esta clase.

- Logger: para capturar y manejar los registros.
- Lista: para almacenar los registros.
- DateTimeFormatter: para dar formato a la fecha, el formato que se va a utilizar es:
    - [viernes, 27 de diciembre de 2024, 06:02:58 p. m]

Funciones que realiza la clase:

- **Registrologs():** constructor de la clase.

Se inicializa el Array para almacenar los registros, y se declara el logger para escribir los mensajes de registro que se generen.

Se configura un FileHandler que nos permitirá escribir los registros en un archivo de tipo txt, que nos permitirá consultar los registros una vez finalizado el turno. Se ha configurado para que no se sobrescriban los datos y se puedan mantener todos los registros independientemente de cuando se ha ejecutado la aplicación.

- **Clase interna CustomFormatter():**

Con esta clase interna se pretende modificar el formato con el que se registrarán los mensajes, evitando que se agregue una línea con la fecha, clase y el método que se usan para obtener la información:

- **fechaActual():**

Función para obtener cada vez que se llame la hora actual.

- **regInicioTurno(int idMaquina):**

Registra el inicio del turno. Recibe como parámetro el id de la Máquina. Datos que se registran son la fecha actual con el formato especificado y el id de máquina.

- **regInicio(int id, Cliente Cliente):**

Registra la asignación del cliente. Recibe como parámetros el id de la Máquina, y el objeto Cliente. Se registrará la fecha de cuando a sido el cliente asignado, el id del cliente y el id de la máquina.

- **regInicioPedido(Cliente cliente):**

Registra el comienzo de la preparación de la bebida. Recibe como parámetro el objeto cliente. Se registra la fecha del inicio de la preparación del pedido mostrando el tipo de bebida.

Se ha agregado un variable de tipo String, que contendrá el tipo de bebida en caracteres para que se muestre por pantalla sin faltas de ortografía el tipo de bebida que ha seleccionado el cliente.

Para asignar a esa variable el valor correcto se ha utilizado una estructura de tipo switch que dará el valor que corresponde entre los tres tipos de bebida predefinidos.

- **regFinPedido(Cliente cliente):**

Registra la finalización del pedido. Recibe como parámetro el objeto cliente. Se registra la fecha con la finalización del pedido y el pedido que se ha preparado.

Se utiliza el mismo método con una variable un switch para que se muestre el pedido con su ortografía correcta.

- **regFinTurno(String cafes, String tes, String chocos):**

Registra el fin del turno de trabajo. Recibe como parámetros el mensaje indicando cuantos cafés, tés y chocolates calientes que se han preparado por ese turno.

- **regError(int id, String men):**

Registra un posible error que surja. Recibe como parámetros el id de la máquina y el mensaje de error.

- **mostrarRegistro():**

Se recorre la lista para mostrar por pantalla el registro completo de eventos. Para realizar esto se ha utilizado un bucle for.

- **guardarRegArchivo():**

Función para escribir en el archivo logs_cafeteria.txt todos los registros que se producido durante el turno completo de trabajo. Se escribirá al final de fichero, permitiendo mantener los registros de los turnos anteriores.

Clase Cliente:

En esta clase representa al cliente. Si fuera una cafetería real el cliente seleccionaría su café directamente en la máquina, pero en este caso, se ha de generar aleatoriamente los tipos de bebida que desea cada cliente.

Atributos principales:

- **contadorId:** variable static de tipo int para que se aumente durante la creación de todos los clientes.
- **Id:** variable de tipo entero que almacenará el id generado automático. Se le asigna final, ya que no cambiará durante todo el proceso.
- **Pedido:** variable TipoBebida, para almacenar los atributos de la bebida que se genere de forma automática.

Funciones principales:

- **Constructor:**

No se recibe ningún parámetro, ya que se generan los valores automáticamente. Para generar el id del cliente se aumenta el contadorId, en uno. Para generar el pedido, se utiliza la clase Random, para que se seleccione de forma aleatoria entre los tres tipos de bebida.

- **Getters:**

Funciones para devolver el id del cliente y el tipo de bebida que va a pedir el cliente.

- **Mensaje(int idMaquina, TipoBebida bebida):**

Función para devolver el pedido y la máquina que esta atendiendo al cliente. Recibe como parámetros el id de la máquina y el tipo de la bebida que va a pedir.

Se a utilizado una estructura switch para mostrar un mensaje u otro en función del tipo de bebida que vaya a pedir el cliente.

Si ha introducido un condicional para mostrar el mensaje siempre y cuando se contenga un mensaje.

Clase Maquina:

En esta clase se implementa la interfaz Runnable, la que permite definir la lo lógica que será ejecutada como un hilo independientemente y que sea pasada como argumento a un objeto de tipo Thread y se facilite su ejecución.

Atributos principales:

- **ContadorId:** variable estática que permite asignar un id nuevo incrementado a cada máquina.
- **Variables para contabilizar las bebidas que sean preparado:** serán de tipo entero, y se irán incrementando según se vayan preparando las bebidas.
- **Id:** variable de tipo entero que almacenará el id que se le asigne a la máquina.
- **Semáforo:** se instancia un objeto de tipo Semaphore para controlar los procesos.
- **Registros:** se instancia un objeto de la clase Registroslogs, para manejar los registros de eventos que se produzcan durante la preparación de los pedidos.

Funciones:

- **Constructor de máquina(Semaphore s):**

Método para construir el objeto de la clase máquina, se recibe como parámetro un objeto Semaphore. Para sincronizar los procesos.

Se generará un id automáticamente, se instancia el semáforo y se instancia el registro, para manejar y registrar los eventos de la máquina.

- **PreparaciónPedido(Cliente cliente):**

Método que procesará los pedidos que hagan los clientes. Se recibe como parámetro el objeto cliente, que contiene el tipo de bebida que ha de preparar la máquina.

Se almacena el objeto TipoBebida para posteriormente poder usarlo en un switch.

Se llama al método mensaje del objeto cliente que mostrará por pantalla la máquina que le esta atendiendo y el pedido que le va a preparar.

Por último, encontramos una estructura switch que procesará el pedido según el tipo de bebida que haya pedido el cliente.

  - Se registrará el inicio del pedido
  - Se congelará el proceso para simular la preparación del pedido,
  - se incrementará en uno el tipo de bebida para contabilizar las bebidas que se hacen de cada uno.
  - SE muestra un mensaje por pantalla indicando que se ha preparado la bebida
  - Y por último se registra la finalización del proceso.

- **finTurno():**

Función para manejar y registrar el fin del turno de trabajo.

Se instanciarán tres mensajes uno por cada una de la bebidas para registrarlo, se llama al método regFinTurno() para que los registre y se escriban en el fichero del registro.

Por último, se muestra el registro completo.

- **run():**

Función que la lógica que se ejecutará.

Lo primero que se hace es registrar el inicio del turno y se imprime un mensaje por pantalla indicando que ha comenzado el proceso de trabajo.

Mediante un bucle while que se ejecutará mientras la cola de cliente no este vacía.

  - Se adquiere un semáforo para empezar el trabajo.
  - Se obtiene el cliente que este en la cabeza de la cola de clientes.
  - Si el cliente no es null, se registra el inicio del pedido, y se llama a la función preparacionPedido que procesará el pedido del cliente.
  - Cuando haya procesado el pedido, se libera el semáforo.

Una vez se hayan procesado todos los pedidos se ejecuta la función finTurno().

Con esto se finalizará la ejecución del hilo máquina.

Clase Principal Cafetería:

Es la clase que simula el recinto de la cafetería, si principal función es coordinar los clientes con las máquinas.

Como no se cuenta con una máquina intermedia donde el cliente realice su pedido, ni existen clientes reales, se va a proceder a crear un grupo aleatorio de clientes, de entre 10 y 40 clientes. De este modo, cada vez que se ejecute la aplicación se simulara un escenario diferente y así ver el comportamiento de la aplicación.

También se ha tenido en cuenta una simulación de que al ser una cafetería se puede dar el caso que accedan más clientes durante el proceso de preparación de los clientes iniciales. Para conseguir esto se ha declarado una función que es llamada por un hilo independiente para que este cada segundo supervisando la cantidad de pedidos que se han realizado, y cuando se hayan realizado más del 55%, se crearán otros 10 o 40 clientes adicionales.

Para poder gestionar la cola de los clientes se ha agregado un semáforo de este modo se coordinarán los clientes con las máquinas evitando que varios clientes puedan acceder a una máquina y surjan errores.

En esta clase se van a definir las tres máquinas, que formarán los hilos que ejecutarán el proceso de la preparación de los pedidos de los clientes.

Atributos principales:

- **numMaquinas:** variable de tipo entero que final ya que no cambiará durante todo el proceso. que almacenará la cantidad de máquinas que hay dentro de la cafetería, de este modo si en un futuro se agregasen más máquinas no tendrías que cambiar el código, y simplemente cambiando este valor, ya se crearían más máquinas.
- **BlockingQueue<Cliente>:** se instancia un blockingQueue con el objeto cliente para crear la cola de los clientes, con este objeto nos evitamos tener que manejar manualmente la sincronización en la cola de clientes, ya que BlockingQueue lo hace internamente y más eficiente. Se le asigna un máximo de 90 espacios a la cola. Se le asignan 10 espacios adicionales por si acaso, pero el máximo de clientes que se pueden generar en el simulador es como mucho 65 clientes a la vez.
- **Semaphore:** Se creará un semáforo con el número de máquinas declaradas, esto permitirá limitar el

número de hilos que pueden acceder a la cola de clientes simultáneamente.
- **Random:** Se utilizará esta variable del objeto random para crear el número de clientes aleatoriamente.
- **generarClientes:** es una variable auxiliar de tipo booleano, que nos permite controlar que solo se agregen clientes una vez más durante la ejecución de la aplicación.
- **cliTotales:** variable de tipo entero que almacenará el total de clientes cuando se generen de forma aleatoria.

Funciones:

- **generarClientes(int cliTotales):**

Esta función genera los primeros clientes. Para realización de esto se utiliza un bucle for que se recorre tantas veces como el valor del parámetro que se recibe. Una vez se creen los clientes se irán agregando a la cola de clientes. Este código contiene un capturador de errores para controlar las excepciones que puedan surgir. En caso que la cola este llena el hilo se bloqueará hasta que haya espacio.

- **masClientes(int cliTotales):**

Con esta función se pretende controlar la cantidad de pedidos que se van realizando, cuando queden por realizarse el 45% de los pedidos, se llamará a la función generar clientes a la que se le pasará un valor generado aleatoriamente entre 10 y 40 clientes. Cuando se hayan generado los clientes se cambia el valor de la variable genearClientes por true. Se sincroniza el acceso a esta función, evitando que múltiples hilos intenten modificar la variable auxiliar generarClientes al mismo tiempo.

- **crearMonitorClientes(int cliTotales):**

Se crea el hilo que permitirá monitorizar a los clientes, y que cuando se haya realizado el 55% de los pedidos se generen más clientes. Este hilo se ejecutará cada segundo, para conseguir esto se congela el hilo durante 1.000 milisegundos. Una vez se hayan generado los nuevos clientes este hilo se finalizará.

- **siguienteCliente():**

Con esta función devolvemos el siguiente cliente de la cola utilizando el método take(), que es bloqueante. Esto método controla de forma automática los bloqueos, esto quiere decir que bloqueará la cola si está vacía o haya más clientes.

- **colaVacia():**

con esta función se pretende devolver un valor booleano para saber si la cola está vacía o no.

- **Main():**

Se crea un pool de hilos con el total de las máquinas que se hayan indicado, esto indica que solo pueden trabajar las máquinas designadas al mismo tiempo procesando los pedidos.

A continuación, se utiliza un bucle for que se recorrerá tantas veces como máquinas hayamos indicado, para crear las máquinas que procesarán los pedidos. Según se van creadon las máquinas los agregamos a ejecutador de servicios.

También se generará un número aleatorio con la variable random entre 10 y 40 para indicar cuantos clientes se van a atender en el primer turno. Una vez tengamos el número se llama a la función que crear a los clientes.

Creamos un hilo para monitorizar los clientes y saber cuantos pedidos se han realizado y ejecutar la función para generar más clientes, y lo ejecutamos para que comience a trabajar.

Ahora se ejecuta el ejecutador de procesos utilizando shutdown(), con esto estamos indicando que comience el proceso y que no se acepten más tareas.

Por último, se finaliza el monitoreo de clientes mediante el método join() que asegura que no se finaliza el proceso hasta que no han sido todos los clientes procesados.

Conclusión:

La función principal que se indicaba se ha conseguido y además, se ha tenido en cuenta la posibilidad de añadir más máquinas, y mas tipos de bebidas. Por otro lado, se ha agregado un fichero que almacenará los registros de todos los turnos, permitiendo ver al administrador de la cafetería si sean procesamos bien los pedidos y cual es el tipo de bebida que más se pide en la cafetería.

En un futuro, se podría mejorar el código añadiendo un tipo de selector de pedidos, que permitiera seleccionar a un cliente real el pedido.

Como se puede ver en la consola comienza a realizar el proceso de la preparación de los pedidos de los clientes.

```
run:
Se va a inciar el proceso de trabajo de la máquina n. 2
Se va a inciar el proceso de trabajo de la máquina n. 3
Se va a inciar el proceso de trabajo de la máquina n. 1
Cliente id: 1. Estoy siendo atendido en la máquina número 2 y me esta preparando un chocolate calietne
Cliente id: 2. Estoy siendo atendido en la máquina número 3 y me esta preparando un té
Cliente id: 3. Estoy siendo atendido en la máquina número 1 y me esta preparando un chocolate calietne
Té listo.
Cliente id: 4. Estoy siendo atendido en la máquina número 3 y me esta preparando un café
Chocolate calietne listo.
Chocolate calietne listo.
Cliente id: 5. Estoy siendo atendido en la máquina número 1 y me esta preparando un café
Cliente id: 6. Estoy siendo atendido en la máquina número 2 y me esta preparando un café
Café listo.
Cliente id: 7. Estoy siendo atendido en la máquina número 3 y me esta preparando un té
Café listo.
Café listo.
Cliente id: 8. Estoy siendo atendido en la máquina número 1 y me esta preparando un café
Cliente id: 9. Estoy siendo atendido en la máquina número 2 y me esta preparando un chocolate calietne
Té listo.
Cliente id: 10. Estoy siendo atendido en la máquina número 3 y me esta preparando un chocolate calietne
Café listo.
```

En la siguiente imagen se puede apreciar el registro completo que se hace tras la finalización del turno.

```
[domingo, 29 de diciembre de 2024, 01:21:24 p. m.] - Inicio del turno de máquina id# 3
-----------------------------------------------------------
[domingo, 29 de diciembre de 2024, 01:21:24 p. m.] - Cliente id# 2 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:24 p. m.] - Se ha iniciado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:26 p. m.] - Se ha finalizado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:26 p. m.] - Cliente id# 4 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:26 p. m.] - Se ha iniciado la preparación del pedido café.
[domingo, 29 de diciembre de 2024, 01:21:28 p. m.] - Se ha finalizado la preparación del pedido café.
[domingo, 29 de diciembre de 2024, 01:21:28 p. m.] - Cliente id# 7 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:28 p. m.] - Se ha iniciado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:29 p. m.] - Se ha finalizado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:29 p. m.] - Cliente id# 10 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:29 p. m.] - Se ha iniciado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:32 p. m.] - Se ha finalizado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:32 p. m.] - Cliente id# 13 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:32 p. m.] - Se ha iniciado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:34 p. m.] - Se ha finalizado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:34 p. m.] - Cliente id# 17 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:34 p. m.] - Se ha iniciado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:37 p. m.] - Se ha finalizado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:37 p. m.] - Cliente id# 20 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:37 p. m.] - Se ha iniciado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:38 p. m.] - Se ha finalizado la preparación del pedido té.
[domingo, 29 de diciembre de 2024, 01:21:38 p. m.] - Cliente id# 22 asignado a la máquina 3
[domingo, 29 de diciembre de 2024, 01:21:38 p. m.] - Se ha iniciado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:41 p. m.] - Se ha finalizado la preparación del pedido chocolate caliente.
[domingo, 29 de diciembre de 2024, 01:21:41 p. m.] - Cliente id# 26 asignado a la máquina 3
```

Aquí podemos ver como al final de cada turno, se muestra la cantidad de bebidas que se han preparados por los diferentes tipos.

```
-----------------------------------------------------------
[domingo, 29 de diciembre de 2024, 01:22:06 p. m.] - Se ha finalizado el turno.
Se han preparado 6 cafés.
Se han preparado 8 tés.
Se han preparado 7 chocolates calientes.
-----------------------------------------------------------
```

En la carpeta que se indica en la ruta donde se quiere escribir los registros se puede observar como se ha creado el fichero. Por configuración predeterminada del FileHandler, se crear backups de seguridad del registro.