Universidad Politécnica de Tlaxcala Región Poniente

Ingeniería en Sistemas Computacionales

Materia: Programación Orientada a Objetos

Docente: Ing. Vanesa Tenopala Zavala

Alumnos:

22SIC008 Isaac Brandon Martínez Ramírez

22SIC016 Gabriel Pérez Tzompa

Tema: Reportes Ejemplos Hilos Parte 2

Ciclo escolar: mayo-agosto 2024

Fecha: 19 de julio de 2024

Reporte del Programa ComidaFilosofos

Descripción del Programa:

El programa "ComidaFilosofos" simula el clásico problema de los filósofos comensales, donde varios filósofos alternan entre comer y pensar, compartiendo recursos limitados (tenedores) para comer. Utiliza hilos y bloqueos para gestionar la concurrencia y evitar condiciones de carrera y bloqueos.

Objetivos:

- Capturar datos de entrada: Gestionar la alternancia de los filósofos entre comer y pensar.
- Controlar la concurrencia: Utilizar bloqueos (Locks) para asegurar que los filósofos no intenten usar los mismos tenedores simultáneamente.
- Mostrar resultados: Imprimir en consola el estado de los filósofos mientras piensan y comen.

Componentes del Programa:

Clases:

1. ComidaFilosofos: Clase principal que inicializa los tenedores y crea hilos de filósofos.
2. Filosofo: Clase que representa a un filósofo, alternando entre los estados de pensar y comer, y manejando los tenedores.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se inicializan los bloqueos (tenedores) y se crean los hilos correspondientes a cada filósofo.
- **Estado de Pensar y Comer:** Cada filósofo alterna entre pensar y comer, intentando tomar los tenedores necesarios antes de comer y liberándolos después.
- **Control de Concurrencia:** Los tenedores se gestionan mediante bloqueos, asegurando que cada filósofo adquiera ambos tenedores necesarios antes de comer.

- **Resultados en Consola:** El estado de cada filósofo se imprime en consola, mostrando cuándo están pensando y cuándo están comiendo.

Resultado:

El programa muestra en la consola los estados de los filósofos, asegurando que no haya condiciones de carrera ni bloqueos mediante el uso adecuado de bloqueos.

Conclusión:

El programa "ComidaFilosofos" proporciona una solución eficiente para el problema de los filósofos comensales mediante el uso de bloqueos para controlar la concurrencia. La simulación concurrente permite manejar de manera realista la alternancia de estados de los filósofos, asegurando que los recursos compartidos se gestionen de manera adecuada.

Reporte del Programa FilosofosCenando

Descripción del Programa:

El programa “FilosofosCenando” es otra implementación del problema de los filósofos comensales, utilizando bloqueos para gestionar el acceso a los tenedores compartidos y evitar condiciones de carrera.

Objetivos:

- Capturar datos de entrada: Gestionar la alternancia de los filósofos entre pensar y comer.
- Controlar la concurrencia: Utilizar bloqueos para asegurar que los filósofos no intenten usar los mismos tenedores simultáneamente.
- Mostrar resultados: Imprimir en consola el estado de los filósofos mientras piensan y comen.

Componentes del Programa:

Clases:

1. FilosofosCenando: Clase principal que inicializa los tenedores y crea hilos de filósofos.
2. Filosofo: Clase que representa a un filósofo, alternando entre los estados de pensar y comer, y manejando los tenedores.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se inicializan los bloqueos (tenedores) y se crean los hilos correspondientes a cada filósofo.
- **Estado de Pensar y Comer:** Cada filósofo alterna entre pensar y comer, intentando tomar los tenedores necesarios antes de comer y liberándolos después.
- **Control de Concurrencia:** Los tenedores se gestionan mediante bloqueos, asegurando que cada filósofo adquiera ambos tenedores necesarios antes de comer.
- **Resultados en Consola:** El estado de cada filósofo se imprime en consola, mostrando cuándo están pensando y cuándo están comiendo.

Resultado:

El programa muestra en la consola los estados de los filósofos, asegurando que no haya condiciones de carrera ni bloqueos mediante el uso adecuado de bloqueos.

Conclusión:

El programa “FilosofosCenando” proporciona una solución eficiente para el problema de los filósofos comensales mediante el uso de bloqueos para controlar la concurrencia. La simulación concurrente permite manejar de manera realista la alternancia de estados de los filósofos, asegurando que los recursos compartidos se gestionen de manera adecuada.

Reporte del Programa FilosofosComensales

Descripción del Programa:

El programa "FilosofosComensales" implementa el problema de los filósofos comensales utilizando un monitor para gestionar la concurrencia y sincronización de los filósofos que comparten recursos (tenedores).

Objetivos:

- Capturar datos de entrada: Gestionar la alternancia de los filósofos entre pensar y comer.
- Controlar la concurrencia: Utilizar un monitor para asegurar que los filósofos no intenten usar los mismos tenedores simultáneamente.
- Mostrar resultados: Imprimir en consola el estado de los filósofos mientras piensan y comen.

Componentes del Programa:

Clases:

1. MesaCircular: Clase que representa la mesa de los filósofos y gestiona la disponibilidad de los tenedores.
2. FilosofoPensante: Clase que representa a un filósofo, alternando entre los estados de pensar y comer.

3. FilosofosComensales: Clase principal que inicializa la mesa y crea hilos de filósofos.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se inicializa la mesa con la cantidad de filósofos y se crean los hilos correspondientes a cada filósofo.
- **Estado de Pensar y Comer:** Cada filósofo alterna entre pensar y comer, intentando tomar los tenedores necesarios antes de comer y liberándolos después.
- **Control de Concurrencia:** La mesa se gestiona mediante un monitor, asegurando que cada filósofo adquiera ambos tenedores necesarios antes de comer.
- **Resultados en Consola:** El estado de cada filósofo se imprime en consola, mostrando cuándo están pensando y cuándo están comiendo.

Resultado:

El programa muestra en la consola los estados de los filósofos, asegurando que no haya condiciones de carrera ni bloqueos mediante el uso adecuado del monitor.

Conclusión:

El programa "FilosofosComensales" proporciona una solución eficiente para el problema de los filósofos comensales mediante el uso de un monitor para controlar la concurrencia. La simulación concurrente permite manejar de manera realista la alternancia de estados de los filósofos, asegurando que los recursos compartidos se gestionen de manera adecuada.

Reporte del Programa join

Descripción del Programa:

El programa "join" demuestra el uso del método join en Java para asegurar que un hilo espere a que otro termine antes de continuar.

Objetivos:

- Capturar datos de entrada: Crear varios hilos que impriman una secuencia de números.
- Controlar la concurrencia: Utilizar el método join para asegurar la secuencialidad en la ejecución de los hilos.
- Mostrar resultados: Imprimir en consola los números generados por cada hilo.

Componentes del Programa:

Clases:

1. join: Clase que extiende Thread y representa un hilo que imprime una secuencia de números.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de la clase join y se inician los hilos.
- **Control de Ejecución:** Se utiliza el método join para asegurar que un hilo espere a que otro termine antes de continuar.
- **Resultados en Consola:** Cada hilo imprime su secuencia de números en consola.

Resultado:

El programa muestra en la consola la secuencia de números generada por cada hilo, asegurando la secuencialidad mediante el uso de join.

Conclusión:

El programa "join" demuestra de manera efectiva el uso del método join para controlar la secuencialidad en la ejecución de hilos. Esta técnica es útil para sincronizar la finalización de tareas concurrentes en Java.

Reporte del Programa Main

Descripción del Programa:

El programa "Main" muestra el uso de una lista sincronizada en Java para asegurar que los accesos concurrentes a la lista se gestionen de manera segura.

Objetivos:

- Capturar datos de entrada: Gestionar una lista de cadenas accesible por múltiples hilos.
- Controlar la concurrencia: Utilizar una lista sincronizada para asegurar el acceso seguro a la lista.
- Mostrar resultados: Imprimir en consola los elementos de la lista.

Componentes del Programa:

Clases:

1. Main: Clase principal que crea y maneja una lista sincronizada.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crea una lista sincronizada y se añaden elementos a la lista.
- **Acceso Seguro:** La lista se accede de manera sincronizada para asegurar que los accesos concurrentes sean seguros.
- **Resultados en Consola:** Los elementos de la lista se imprimen en consola.

Resultado:

El programa muestra en la consola los elementos de la lista, demostrando el uso seguro de una lista sincronizada.

Conclusión:

El programa “Main” proporciona una demostración clara del uso de listas sincronizadas en Java para gestionar accesos concurrentes de manera segura, asegurando la integridad de los datos.

Reporte del Programa Main2

Descripción del Programa:

El programa "Main2" muestra la creación y manejo de una lista de cadenas en Java, imprimiendo sus elementos en consola.

Objetivos:

- Capturar datos de entrada: Gestionar una lista de cadenas.
- Mostrar resultados: Imprimir en consola los elementos de la lista.

Componentes del Programa:

Clases:

1. Main2: Clase principal que crea y maneja una lista de cadenas.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crea una lista de cadenas y se añaden elementos a la lista.
- **Resultados en Consola:** Los elementos de la lista se imprimen en consola.

Resultado:

El programa muestra en la consola los elementos de la lista.

Conclusión:

El programa "Main2" proporciona una demostración simple y efectiva de la creación y manejo de listas en Java.

Reporte del Programa Main3

Descripción del Programa:

El programa "Main3" demuestra el uso del ExecutorService en Java para gestionar la ejecución de tareas concurrentes en un pool de hilos.

Objetivos:

- Capturar datos de entrada: Crear tareas concurrentes que impriman una secuencia de números.
- Controlar la concurrencia: Utilizar ExecutorService para gestionar la ejecución de tareas en un pool de hilos.
- Mostrar resultados: Imprimir en consola los números generados por cada tarea.

Componentes del Programa:

Clases:

1. Main3: Clase principal que gestiona la ejecución de tareas concurrentes.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crea un ExecutorService con un pool de dos hilos y se envían tareas para su ejecución.
- **Ejecución de Tareas:** Las tareas se ejecutan concurrentemente en los hilos del pool.
- **Resultados en Consola:** Las tareas imprimen una secuencia de números en consola.

Resultado:

El programa muestra en la consola la secuencia de números generada por cada tarea, demostrando el uso de ExecutorService para la gestión de concurrencia.

Conclusión:

El programa "Main3" proporciona una demostración clara y efectiva del uso de ExecutorService para gestionar la ejecución concurrente de tareas en Java, mejorando la eficiencia y control de los recursos.

Reporte del Programa Monitor

Descripción del Programa:

El programa "Monitor" demuestra el uso de métodos sincronizados en Java para gestionar el acceso concurrente a un contador compartido.

Objetivos:

- Capturar datos de entrada: Gestionar el incremento y decremento de un contador.
- Controlar la concurrencia: Utilizar métodos sincronizados para asegurar el acceso seguro al contador.
- Mostrar resultados: Imprimir en consola el valor del contador.

Componentes del Programa:

Clases:

1. Monitor: Clase que gestiona el contador con métodos sincronizados.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de la clase Monitor que incrementan y decrementan el contador.
- **Acceso Seguro:** Los métodos increment, decrement, y getCounter están sincronizados para asegurar el acceso seguro al contador.
- **Resultados en Consola:** El valor del contador se imprime en consola.

Resultado:

El programa muestra en la consola el valor del contador, demostrando el uso seguro de métodos sincronizados.

Conclusión:

El programa "Monitor" proporciona una demostración efectiva del uso de sincronización en métodos para gestionar el acceso concurrente a recursos compartidos, asegurando la integridad de los datos.

Reporte del Programa Monitor1

Descripción del Programa:

El programa "Monitor1" es una variación del programa "Monitor", mostrando el uso de métodos sincronizados en Java para gestionar un contador.

Objetivos:

- Capturar datos de entrada: Gestionar el incremento y decremento de un contador.
- Controlar la concurrencia: Utilizar métodos sincronizados para asegurar el acceso seguro al contador.
- Mostrar resultados: Imprimir en consola el valor del contador.

Componentes del Programa:

Clases:

1. Monitor1: Clase que gestiona el contador con métodos sincronizados.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de la clase Monitor1 que incrementan y decrementan el contador.
- **Acceso Seguro:** Los métodos increment, decrement, y getCount están sincronizados para asegurar el acceso seguro al contador.
- **Resultados en Consola:** El valor del contador se imprime en consola.

Resultado:

El programa muestra en la consola el valor del contador, demostrando el uso seguro de métodos sincronizados.

Conclusión:

El programa "Monitor1" proporciona una demostración efectiva del uso de sincronización en métodos para gestionar el acceso concurrente a recursos compartidos, asegurando la integridad de los datos.

Reporte del Programa PrInterruptedException

Descripción del Programa:

El programa "PrInterruptedException" muestra el manejo de interrupciones en hilos en Java, demostrando cómo interrumpir un hilo y gestionar la excepción InterruptedException.

Objetivos:

- Capturar datos de entrada: Crear y gestionar hilos que se pueden interrumpir.
- Controlar la concurrencia: Utilizar el método interrupt para interrumpir hilos y gestionar la excepción resultante.
- Mostrar resultados: Imprimir en consola el estado de los hilos y las interrupciones.

Componentes del Programa:

Clases:

1. PrInterruptedException: Clase principal que gestiona la creación y interrupción de hilos.
2. PrThread: Clase que extiende Thread y representa un hilo que se puede interrumpir.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de PrThread y se inician los hilos.
- **Manejo de Interrupciones:** Los hilos se interrumpen utilizando el método interrupt, y la excepción InterruptedException se maneja adecuadamente.
- **Resultados en Consola:** El estado de los hilos y las interrupciones se imprimen en consola.

Resultado:

El programa muestra en la consola el estado de los hilos y las interrupciones, demostrando el manejo adecuado de InterruptedException.

Conclusión:

El programa "PrInterruptedException" proporciona una demostración clara y efectiva del manejo de interrupciones en hilos en Java, asegurando que los hilos se gestionen de manera segura y eficiente.

Reporte del Programa PrRunnable

Descripción del Programa:

El programa "PrRunnable" implementa la interfaz Runnable en Java para crear y ejecutar tareas concurrentes.

Objetivos:

- Capturar datos de entrada: Crear tareas concurrentes que impriman una secuencia de números.
- Controlar la concurrencia: Utilizar la interfaz Runnable para ejecutar tareas en hilos.
- Mostrar resultados: Imprimir en consola los números generados por cada tarea.

Componentes del Programa:

Clases:

1. PrRunnable: Clase que implementa Runnable y representa una tarea que imprime una secuencia de números.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de PrRunnable y se ejecutan en hilos.
- **Ejecución de Tareas:** Las tareas se ejecutan concurrentemente en los hilos.
- **Resultados en Consola:** Las tareas imprimen una secuencia de números en consola.

Resultado:

El programa muestra en la consola la secuencia de números generada por cada tarea, demostrando el uso de Runnable para la gestión de concurrencia.

Conclusión:

El programa "PrRunnable" proporciona una demostración clara y efectiva del uso de la interfaz Runnable para gestionar la ejecución concurrente de tareas en Java, mejorando la flexibilidad y control de los recursos.

Reporte del Programa PrThread

Descripción del Programa:

El programa "PrThread" extiende la clase Thread en Java para crear y ejecutar hilos que imprimen una secuencia de números.

Objetivos:

- Capturar datos de entrada: Crear hilos que impriman una secuencia de números.
- Controlar la concurrencia: Utilizar la clase Thread para gestionar la ejecución de hilos.
- Mostrar resultados: Imprimir en consola los números generados por cada hilo.

Componentes del Programa:

Clases:

1. PrThread: Clase que extiende Thread y representa un hilo que imprime una secuencia de números.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de PrThread y se inician los hilos.
- **Ejecución de Hilos:** Los hilos se ejecutan concurrentemente.
- **Resultados en Consola:** Los hilos imprimen una secuencia de números en consola.

Resultado:

El programa muestra en la consola la secuencia de números generada por cada hilo, demostrando el uso de Thread para la gestión de concurrencia.

Conclusión:

El programa “PrThread” proporciona una demostración clara y efectiva del uso de la clase Thread para gestionar la ejecución concurrente de tareas en Java, mejorando la flexibilidad y control de los recursos.

Reporte del Programa ReaderWriterMonitor

Descripción del Programa:

El programa "ReaderWriterMonitor" implementa un monitor para gestionar el acceso concurrente de lectores y escritores a un recurso compartido en Java.

Objetivos:

- Capturar datos de entrada: Gestionar el acceso concurrente de lectores y escritores.
- Controlar la concurrencia: Utilizar un monitor para asegurar el acceso seguro al recurso compartido.
- Mostrar resultados: Imprimir en consola el estado de los lectores y escritores.

Componentes del Programa:

Clases:

1. ReaderWriterMonitor: Clase que gestiona el acceso concurrente de lectores y escritores mediante un monitor.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de tareas de lectura y escritura y se ejecutan en hilos.
- **Control de Acceso:** Los métodos lockRead y lockWrite gestionan el acceso seguro al recurso compartido.
- **Resultados en Consola:** El estado de los lectores y escritores se imprime en consola.

Resultado:

El programa muestra en la consola el estado de los lectores y escritores, demostrando el uso de un monitor para la gestión segura de concurrencia.

Conclusión:

El programa "ReaderWriterMonitor" proporciona una demostración efectiva del uso de un monitor para gestionar el acceso concurrente a recursos compartidos en Java, asegurando la integridad de los datos y mejorando la eficiencia del sistema.

Reporte del Programa run

Descripción del Programa:

El programa "run" extiende la clase Thread en Java para crear y ejecutar hilos que imprimen una secuencia de números.

Objetivos:

- Capturar datos de entrada: Crear hilos que impriman una secuencia de números.
- Controlar la concurrencia: Utilizar la clase Thread para gestionar la ejecución de hilos.
- Mostrar resultados: Imprimir en consola los números generados por cada hilo.

Componentes del Programa:

Clases:

1. run: Clase que extiende Thread y representa un hilo que imprime una secuencia de números.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean instancias de run y se inician los hilos.
- **Ejecución de Hilos:** Los hilos se ejecutan concurrentemente.
- **Resultados en Consola:** Los hilos imprimen una secuencia de números en consola.

Resultado:

El programa muestra en la consola la secuencia de números generada por cada hilo, demostrando el uso de Thread para la gestión de concurrencia.

Conclusión:

El programa “run” proporciona una demostración clara y efectiva del uso de la clase Thread para gestionar la ejecución concurrente de tareas en Java, mejorando la flexibilidad y control de los recursos.

Reporte del Programa SyncExample

Descripción del Programa:

El programa "SyncExample" demuestra el uso de métodos sincronizados en Java para gestionar el acceso concurrente a un contador compartido.

Objetivos:

- Capturar datos de entrada: Gestionar el incremento y decremento de un contador.
- Controlar la concurrencia: Utilizar métodos sincronizados para asegurar el acceso seguro al contador.
- Mostrar resultados: Imprimir en consola el valor del contador.

Componentes del Programa:

Clases:

1. SyncExample: Clase que gestiona el contador con métodos sincronizados.

Funcionamiento del Programa:

- **Inicialización:** Al ejecutar el programa, se crean hilos que incrementan el contador.
- **Acceso Seguro:** Los métodos increment y decrement están sincronizados para asegurar el acceso seguro al contador.
- **Resultados en Consola:** El valor del contador se imprime en consola.

Resultado:

El programa muestra en la consola el valor del contador, demostrando el uso seguro de métodos sincronizados.

Conclusión:

El programa "SyncExample" proporciona una demostración efectiva del uso de sincronización en métodos para gestionar el acceso concurrente a recursos compartidos, asegurando la integridad de los datos.