

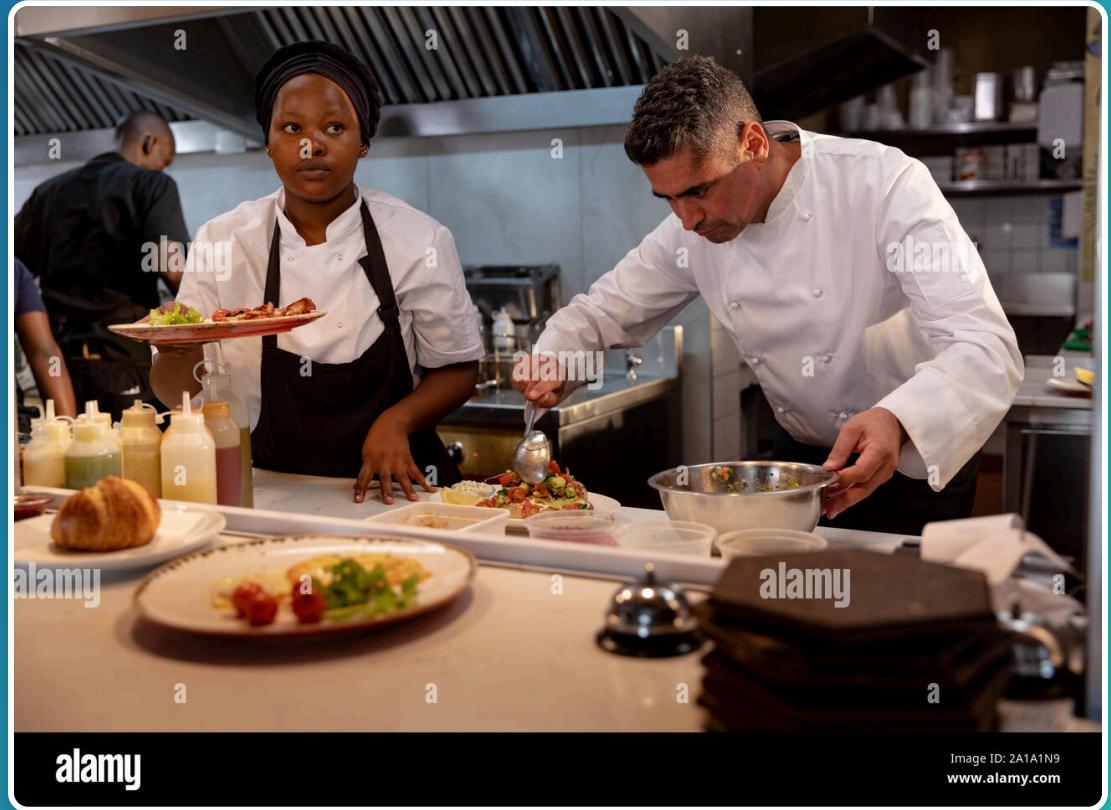
Concurrencia Moderna en Java

Haciendo Varias Cosas a la Vez

🍴 Imagina una cocina muy concurrida. Hay varios chefs trabajando en diferentes platos al mismo tiempo.

Esto es similar a la **concurrencia** en programación: tu programa puede estar haciendo varias tareas simultáneamente.

⚠ **Problema:** Gestionar muchos chefs (tareas) puede ser complicado. ¿Cómo se aseguran de no tropezar, de compartir los recursos y de que todo plato se termine?



Hilos Tradicionales

Los Chefs Dedicados

En Java, tradicionalmente usamos **hilos (threads)** para la concurrencia.

Piensa en cada hilo como un **chef dedicado**. Cada chef:

- Necesita su propio espacio de trabajo (memoria)
- Requiere recursos del sistema operativo
- Solo puede hacer una tarea a la vez

```
Thread chefThread = new Thread(() -> {  
    System.out.println("Cocinando un plato...");  
});  
chefThread.start();
```

⚠ Problema: Crear demasiados chefs (hilos) consume muchos recursos. Una cocina llena de gente que apenas se mueve puede ralentizar o incluso bloquear tu programa.



Chef 1

Memoria

Recursos



Chef 2

Memoria

Recursos



Chef 3

Memoria

Recursos

¡Llegan los Hilos Virtuales!

Los Ayudantes Eficientes (Virtual Threads)

💡 Los **hilos virtuales** son una nueva forma de manejar la concurrencia en Java, introducida con **Project Loom**.

🍴 **Analogía culinaria:** Imagina **muchos ayudantes de cocina** que pueden ayudar a los pocos chefs dedicados.

📞 **Analogía call center:**

- Muchos **clientes esperando** (tareas)
- Muchos **representantes virtuales** (hilos virtuales)
- Pocos **agentes reales** (hilos de plataforma)

★ **Ventaja clave:** Los representantes virtuales pueden atender a muchos clientes, pero solo un pequeño número de agentes reales los atienden físicamente en un momento dado.



Menos Recursos, Más Concurrencia

Ligereza y Escala

⚡ Los hilos virtuales son mucho más **ligeros** que los hilos tradicionales:

- No necesitan su propio espacio de trabajo grande todo el tiempo
- Comparten eficientemente los recursos del sistema
- Utilizan un modelo de memoria más eficiente
- Su creación es mucho menos costosa

↳ Esto significa que puedes tener **muchos más hilos virtuales** ejecutándose simultáneamente sin agotar los recursos del sistema.

“Virtual threads can number in the **tens of millions** by featuring small often managed stacks”

¿Por qué importa?

Una aplicación típica que utiliza hilos de plataforma podría estar limitada a **miles de hilos**, mientras que con hilos virtuales puede escalar a **millones** - crucial para microservicios, aplicaciones web y sistemas con alto nivel de concurrencia.



Características	Hilos de Plataforma	Hilos Virtuales
Memoria por hilo	~1-2 MB	~2 KB
Capacidad típica	Miles	Millones
Tiempo de creación	Alto	Muy bajo
Bloqueo de I/O	Bloquea el hilo OS	No bloquea el hilo OS



Plataforma
~Miles

↔ Diferencia de escala



Virtuales
~Millones

¿Cómo Funcionan?

Trabajo en Equipo Eficiente

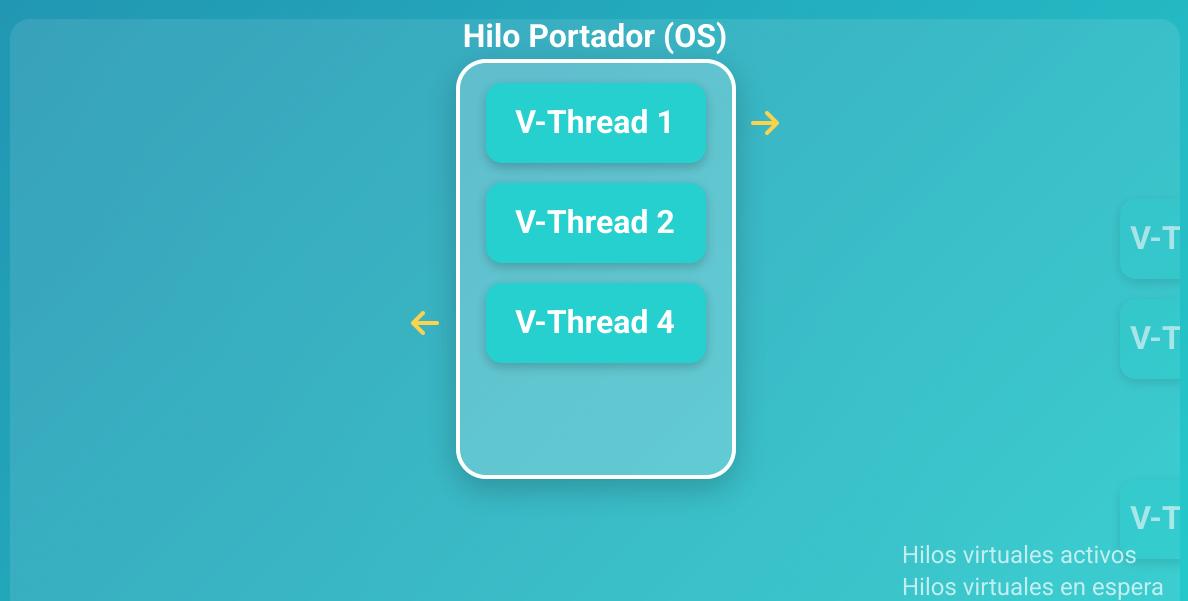
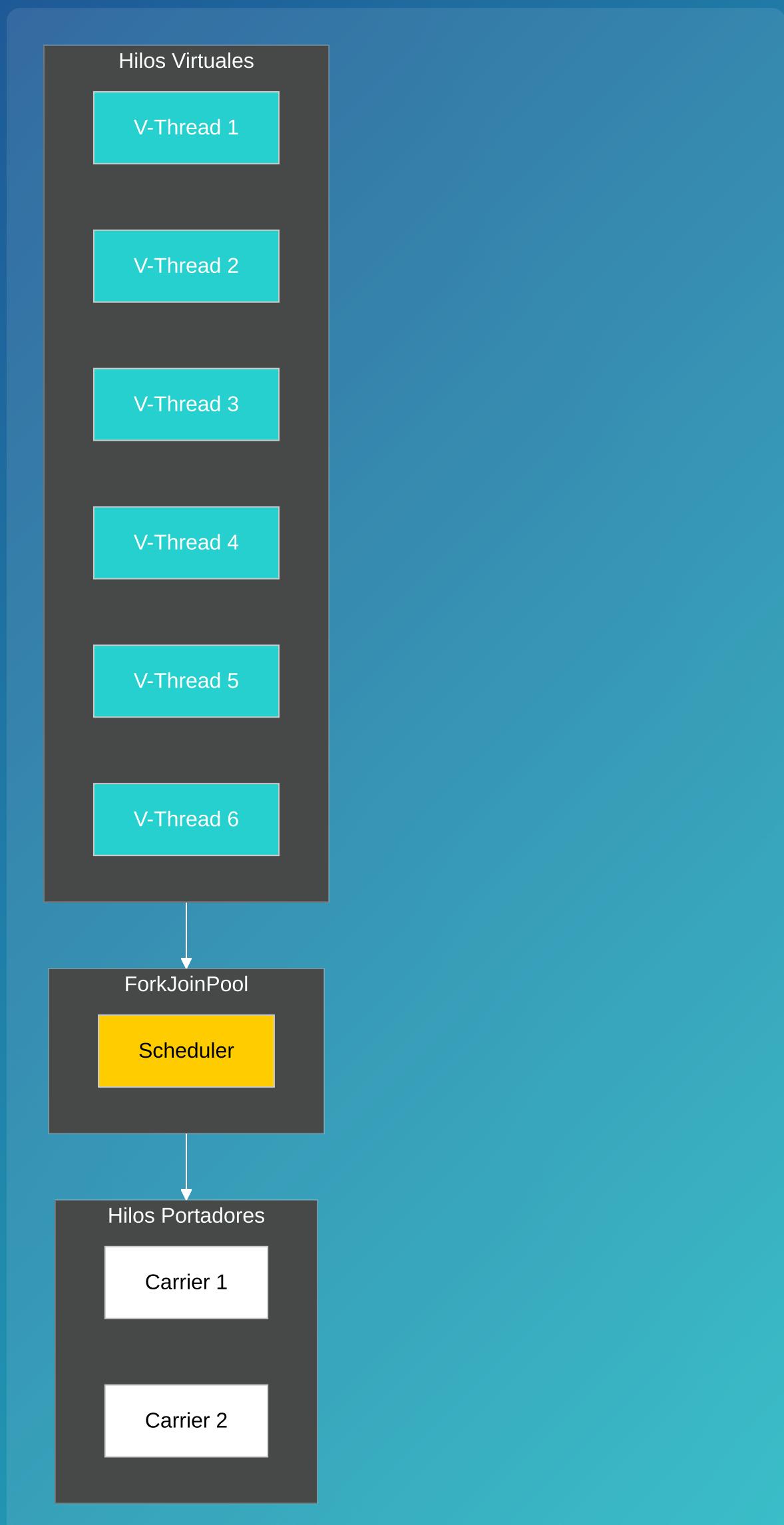
💡 Los hilos virtuales se ejecutan sobre un pequeño grupo de hilos tradicionales llamados **hilos de portador** (carrier threads).

1. Montaje
El hilo virtual se monta en un hilo portador disponible
2. Espera
Si necesita esperar (I/O, red), se desmonta liberando el portador
3. Reanudación
Cuando los datos están listos, se vuelve a montar en un portador

Analogía del call center:

- Cuando un representante virtual (hilo virtual) necesita esperar una respuesta, **deja libre la línea telefónica** (hilo portador)
- Otro representante puede usar esa línea mientras tanto
- Cuando llega la respuesta, el representante original **retoma su tarea** en la primera línea disponible

“ “The virtual thread is des-scheduled from the OS thread which is called the carrier thread and the carrier thread now is free to pick up some other virtual thread” ”



Creando Hilos Virtuales

Fácil de Usar

La forma de crear y usar hilos virtuales es muy similar a la de los hilos tradicionales en Java.

Hay múltiples formas de crear hilos virtuales, todas ellas con una API intuitiva y fácil de usar:

- Usando **Thread.ofVirtual()** directamente
- Utilizando **Executors** para crear pools de hilos virtuales
- Con **StructuredTaskScope** para concurrencia estructurada

Hilo Tradicional
`new Thread(() -> {
 System.out.println("Tarea");
}).start();`

Hilo Virtual
`Thread.ofVirtual().start(() -> {
 System.out.println("Tarea");
});`

“The thread API is intact as well as executor services”

💡 Beneficio clave: El código existente puede actualizarse fácilmente para utilizar hilos virtuales con cambios mínimos.

Creando un hilo virtual simple

```
// Creando y ejecutando un hilo virtual  
Thread.ofVirtual().start(() -> {  
    System.out.println("¡Hola desde un hilo virtual!");  
});
```

Ejecutor de hilos virtuales

```
// Creando un ExecutorService con hilos virtuales  
try (ExecutorService executor = Executors.newVirtualThreadPerTaskExecutor())  
    for (int i = 0; i < 10_000; i++) {  
        final int id = i;  
        executor.submit(() -> {  
            System.out.println("Tarea " + id + " en hilo virtual");  
            return id;  
        });
```

Ejecutor Tradicional

```
ExecutorService exec = Executors  
    .newFixedThreadPool(100);
```

⚠ Limitado a 100 hilos concurrentes

Ejecutor Virtual

```
ExecutorService exec = Executors  
    .newVirtualThreadPerTaskExecutor();
```

✓ Un hilo virtual por tarea, sin límite práctico

💡 **Millones** de hilos virtuales pueden ejecutarse con la misma sintaxis familiar de Java

Beneficios: Escalabilidad

Tu Programa Crece sin Problemas

Gracias a su ligereza, tu aplicación puede manejar **muchas más peticiones o tareas concurrentes** sin problemas de rendimiento.



Aplicaciones Web

Maneja miles de conexiones HTTP simultáneas con facilidad



Microservicios

Gestiona más solicitudes por instancia, reduciendo la necesidad de escalar horizontalmente

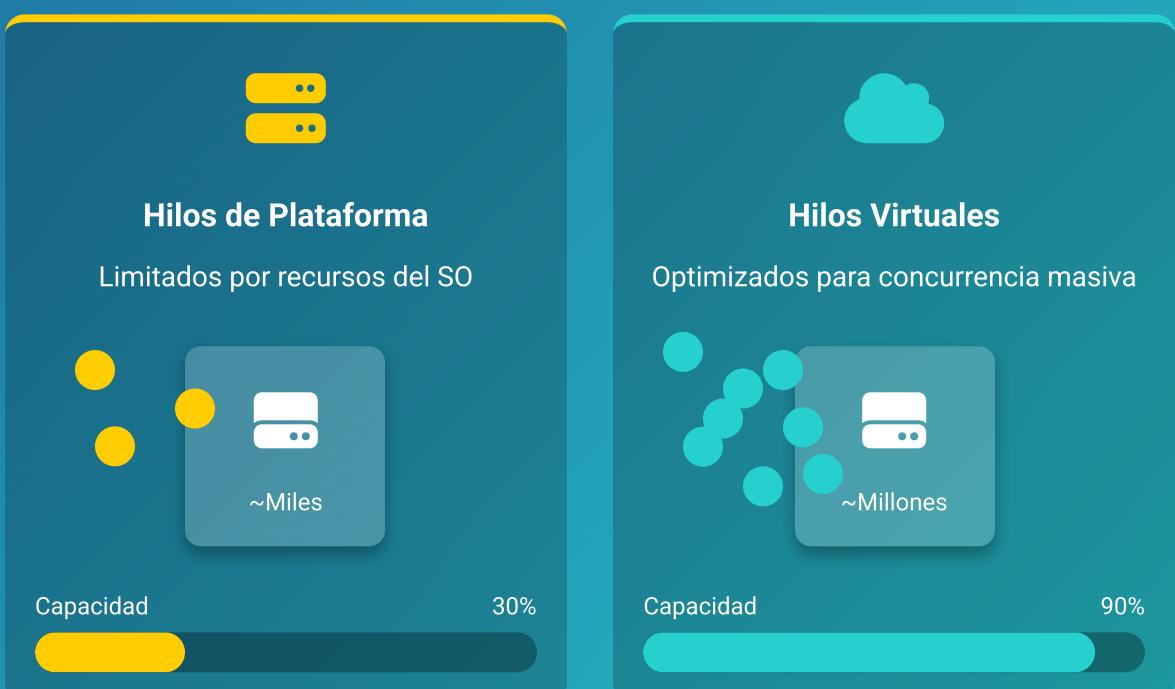
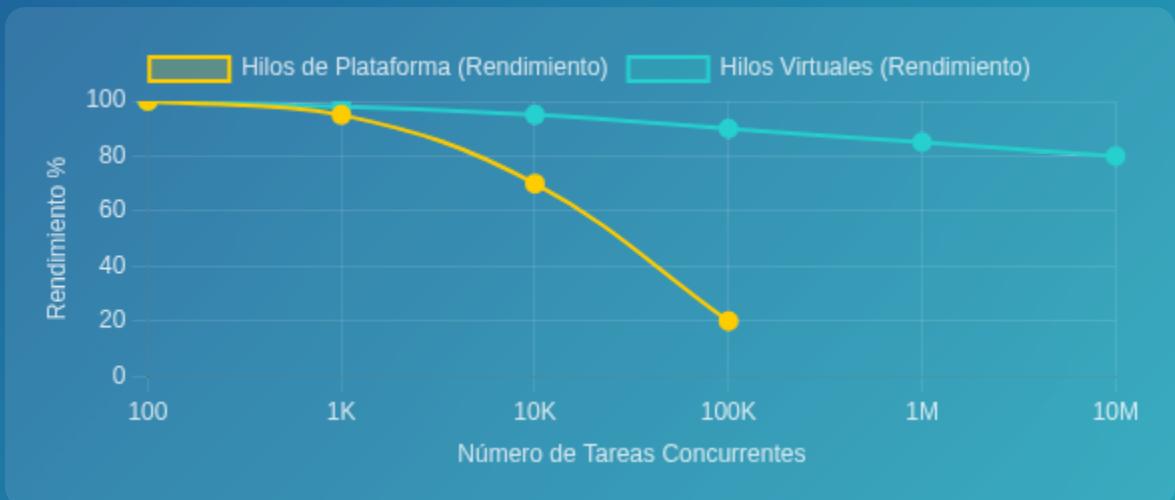


Procesamiento de Datos

Realiza operaciones de E/S y consultas a bases de datos de forma más eficiente

“Un call center con muchos representantes virtuales puede atender a muchos más clientes simultáneamente que uno con solo unos pocos agentes reales.”

Impacto: Los sistemas pueden **escalar verticalmente** (aprovechando mejor cada máquina) antes de necesitar **escalar horizontalmente** (agregar más máquinas).



Beneficios: Eficiencia de Recursos

Menos Gasto, Más Potencia

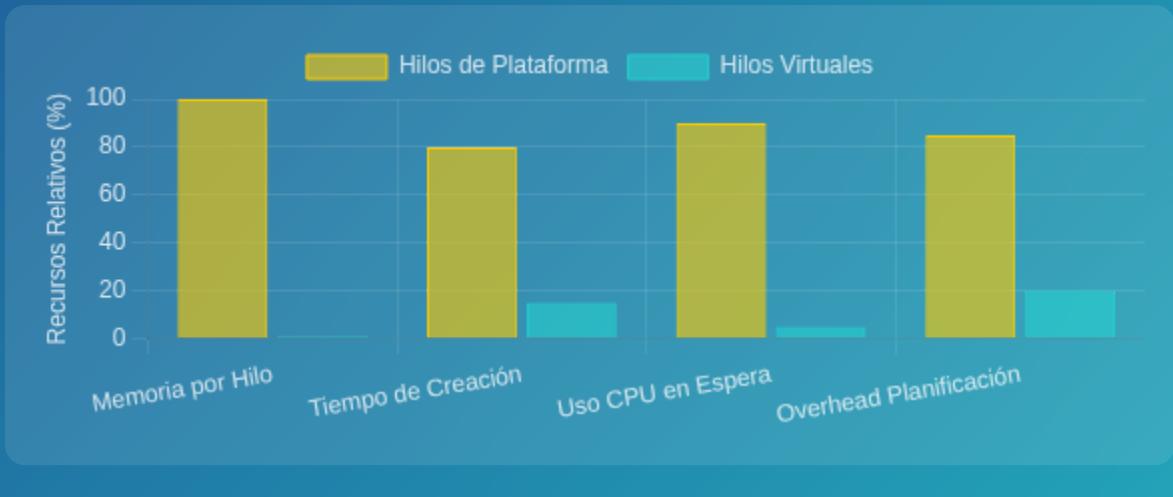
- 💡 Al usar menos recursos por hilo, tu aplicación es más **eficiente** en el uso de la memoria y la CPU.

Hilo de Plataforma
~1-2 MB

por hilo (memoria)

Hilo Virtual
~2 KB

por hilo (memoria)



Esta eficiencia se traduce directamente en:

- **Menores costos** de infraestructura (servidores, contenedores, nubes)
- **Mejor rendimiento** general del sistema
- **Menor consumo** de energía (más sostenible)
- **Mejor uso** de la capacidad existente

“A virtual thread that is not currently running requires a lot less resources than a platform thread”

- 💡 **Resultado práctico:** Puedes ejecutar más de **1 millón** de hilos virtuales en una máquina que apenas podría manejar unos **miles** de hilos de plataforma.

Beneficios en Recursos

Menor Uso de Memoria

Hasta un 99% de reducción en el consumo de memoria por hilo, permitiendo ejecutar más tareas concurrentes.



Uso de CPU Optimizado

Mejor aprovechamiento de los núcleos de CPU al reducir bloqueos y optimizar cambios de contexto.



Ahorro en Infraestructura

Menos servidores necesarios para manejar la misma carga de trabajo, lo que reduce los costos de infraestructura.



Casos de Uso de Alta Eficiencia



Aplicaciones Cloud



Backends Móviles



Gateway APIs



Operaciones I/O

Beneficios: Bloqueo sin Costo

Esperar ya no Bloquea Todo

☒ En hilos tradicionales, cuando un hilo se bloquea esperando algo (E/S, una respuesta), el hilo del sistema operativo subyacente también se bloquea.

💡 Con hilos virtuales, la espera es **semántica**:

- El hilo virtual se **desmonta** del hilo portador
- El hilo portador queda **libre** para ejecutar otros hilos virtuales
- Cuando la operación de E/S completa, el hilo virtual **vuelve a montarse** en un hilo portador disponible

“ “The virtual thread is des-scheduled from the OS thread which is called the carrier thread and the carrier thread now is free to pick up some other virtual thread”



➡ Comparativa: Bloqueo en Operaciones de E/S

Beneficios del Bloqueo sin Costo:

- ⌚ Mayor throughput de aplicaciones intensivas en E/S
- 💾 Mejor aprovechamiento de los recursos del sistema
- ⚡ Código secuencial simple en lugar de callbacks complejos
- ⚡ Menos hilos de OS necesarios para manejar muchas tareas concurrentes

Escenarios de Espera Comunes



Concurrencia Estructurada: El Caos

Hilos Desorganizados

⚠ Sin una buena estructura, la gestión de múltiples hilos puede volverse **caótica**.

Imagina una obra de construcción con muchos trabajadores haciendo cosas sin un plan claro:



Finalización Incierta

Es difícil saber cuándo termina todo el trabajo



Gestión de Errores

Si un hilo falla, los otros siguen ejecutándose sin darse cuenta



Sincronización Manual

Complejos mecanismos de coordinación con locks, semáforos, etc.

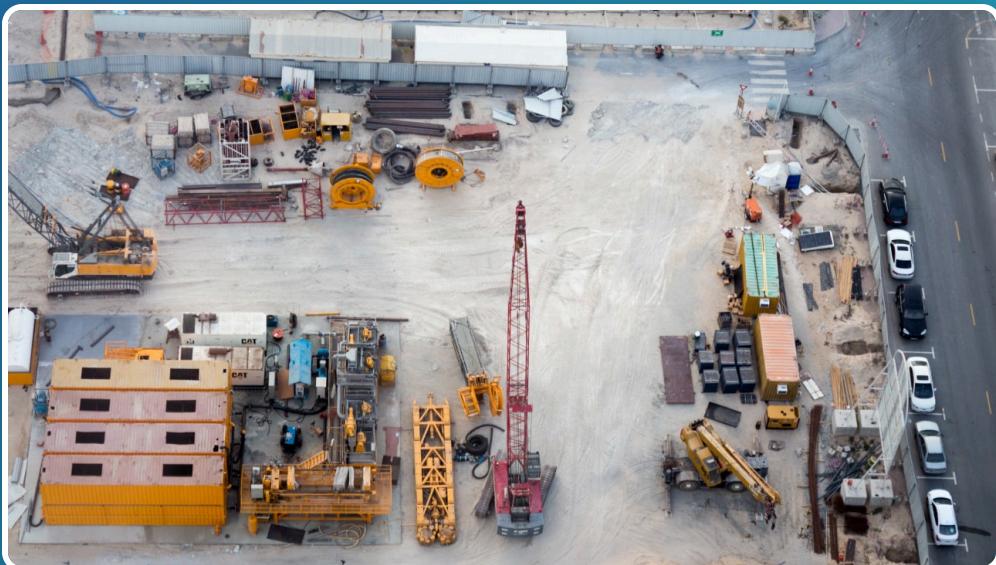


Limpieza de Recursos

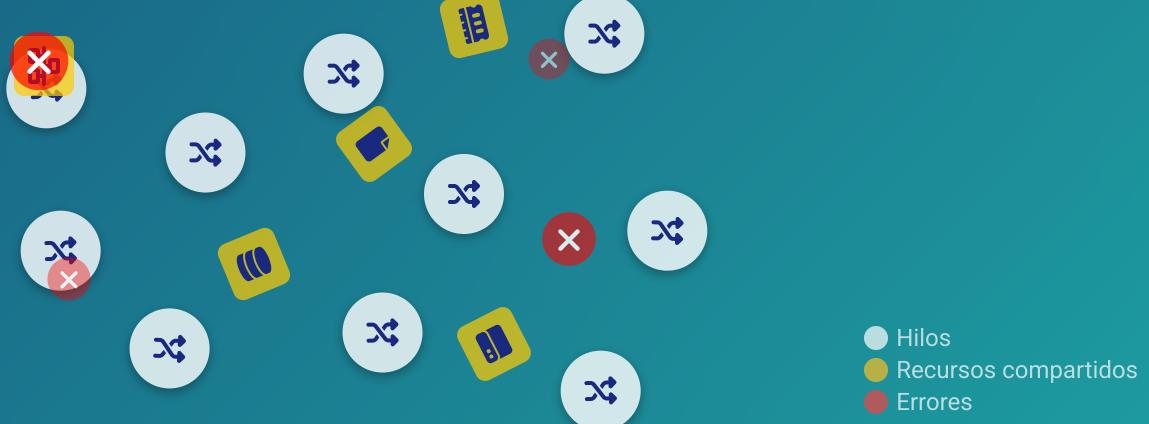
Difícil garantizar que todos los recursos se liberan correctamente



❗ **Problema:** Esto se llama **conurrencia no estructurada**. El código se vuelve complejo, difícil de razonar y propenso a errores sutiles.



Visualización de Concurrencia No Estructurada



→ Necesitamos una mejor forma de organizar el trabajo concurrente

Concurrencia Estructurada: El Orden

Organizando el Trabajo Concurrente

💡 La **concurrencia estructurada** es un paradigma que organiza los hilos concurrentes en **estructuras de control claras**, con puntos de entrada y salida definidos.

Piensa en **grupos de tareas relacionadas** que se tratan como una **unidad de trabajo**:



Estructura Jerárquica

Las subtareas tienen una relación padre-hijo con su tarea principal



Finalización Explícita

Se sabe exactamente cuándo terminan todas las tareas relacionadas



Gestión de Errores Unificada

Los errores se propagan a la tarea padre y se pueden manejar centralmente

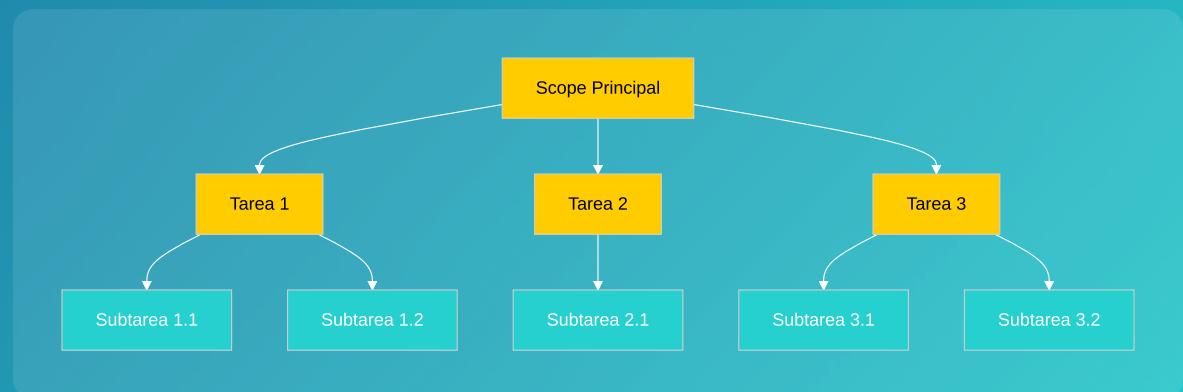
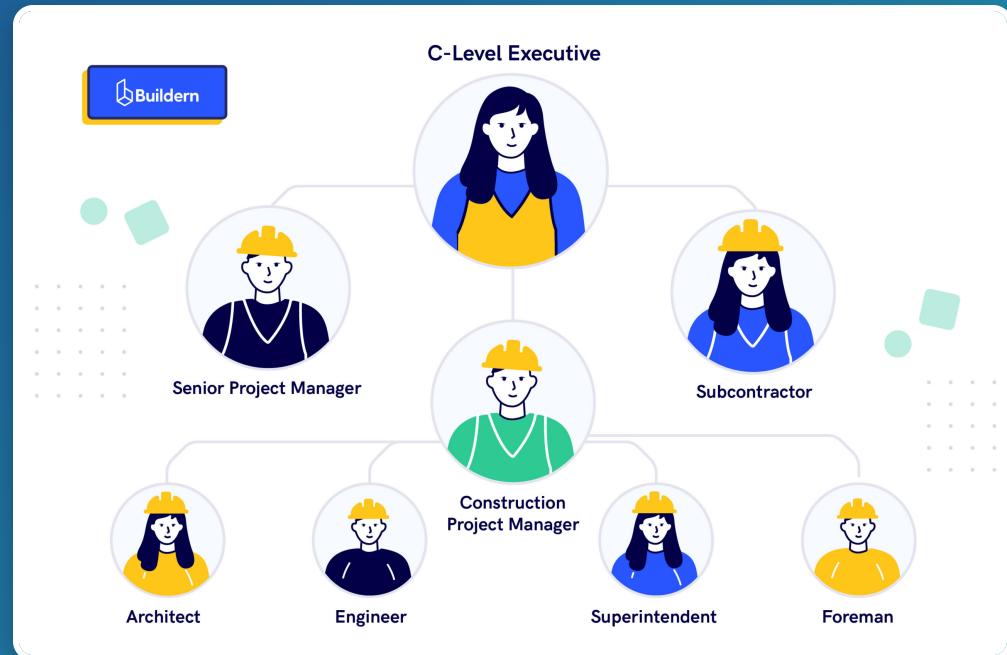


Limpieza Automática

Los recursos se liberan de forma organizada cuando termina el ámbito



👷 Analogía: En la obra, ahora hay **equipos con un líder** para cada tarea específica (excavación, cimentación). Sabemos cuándo empieza y termina el trabajo de cada equipo.



StructuredTaskScope

El Director de Orquesta

Java introduce **StructuredTaskScope** para implementar la concurrencia estructurada.

Permite **crear un ámbito** donde se lanzan tareas concurrentes (hilos virtuales) y se gestionan como un grupo.

Ámbito Definido

Se cierra solo cuando todas las tareas dentro han terminado

Manejo de Errores

Propagación de excepciones a la tarea padre

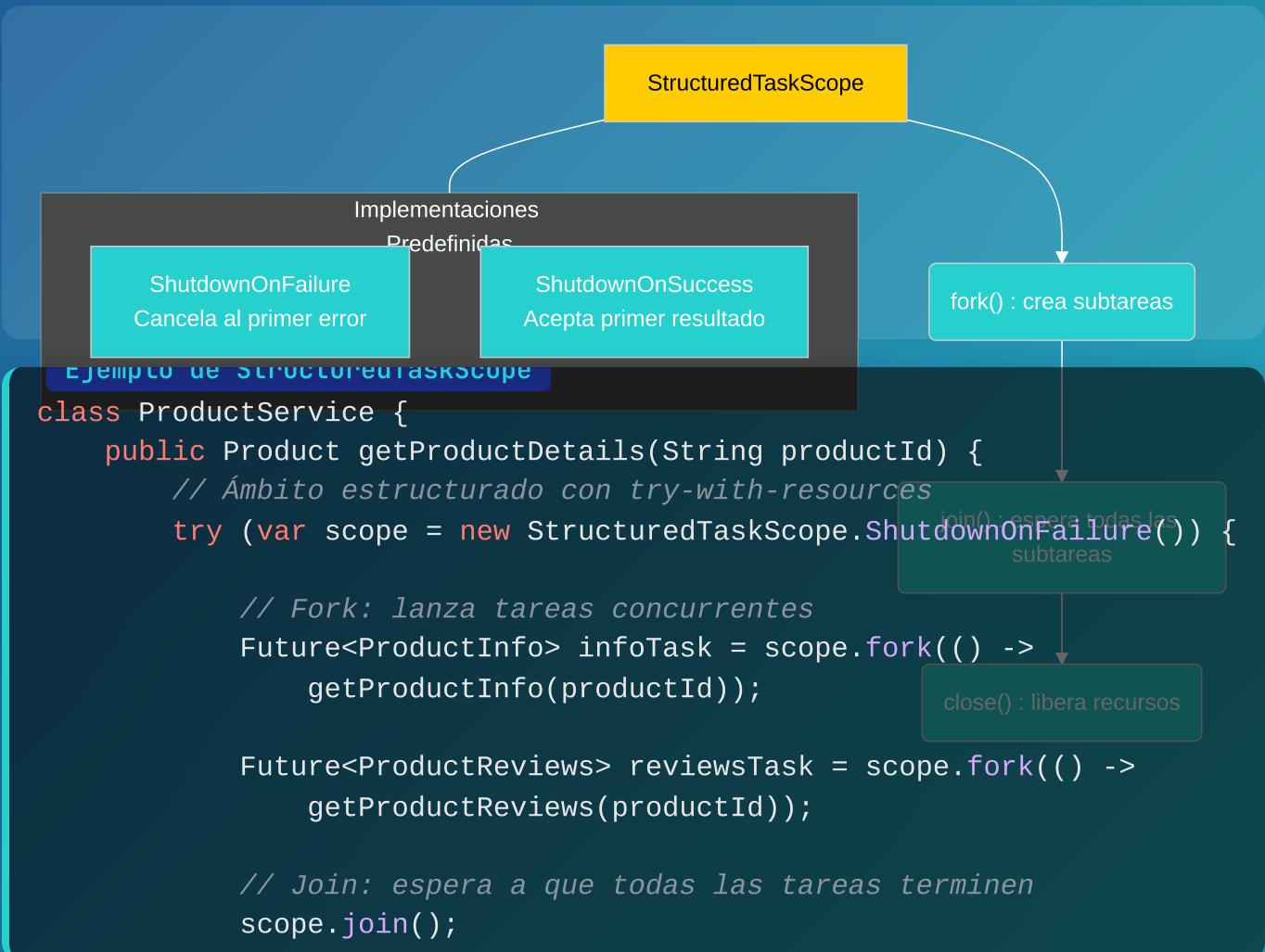
Cancelación

Cancela automáticamente tareas secundarias si es necesario

Limpieza de Recursos

Los recursos se liberan al finalizar el ámbito

"The try-with-resources block delimits the scope of the structured concurrency area. We'll see in a moment that the computation will leave the try block only when all the subtasks created inside it are completed"



Variantes de StructuredTaskScope

ShutdownOnFailure

Cancela todas las tareas pendientes si alguna falla

```
new  
StructuredTaskScope.ShutdownOnFailure()
```

ShutdownOnSuccess

Cancela todas las tareas cuando una termina con éxito

```
new  
StructuredTaskScope.ShutdownOnSuccess<T>()  
()
```

Compatible con hilos virtuales para máxima eficiencia

Manejo de Errores y Cancelación

Controlando los Imprevistos

- 💡 La concurrencia estructurada facilita el **manejo de errores**.

Si una tarea dentro de un ámbito falla, se puede:

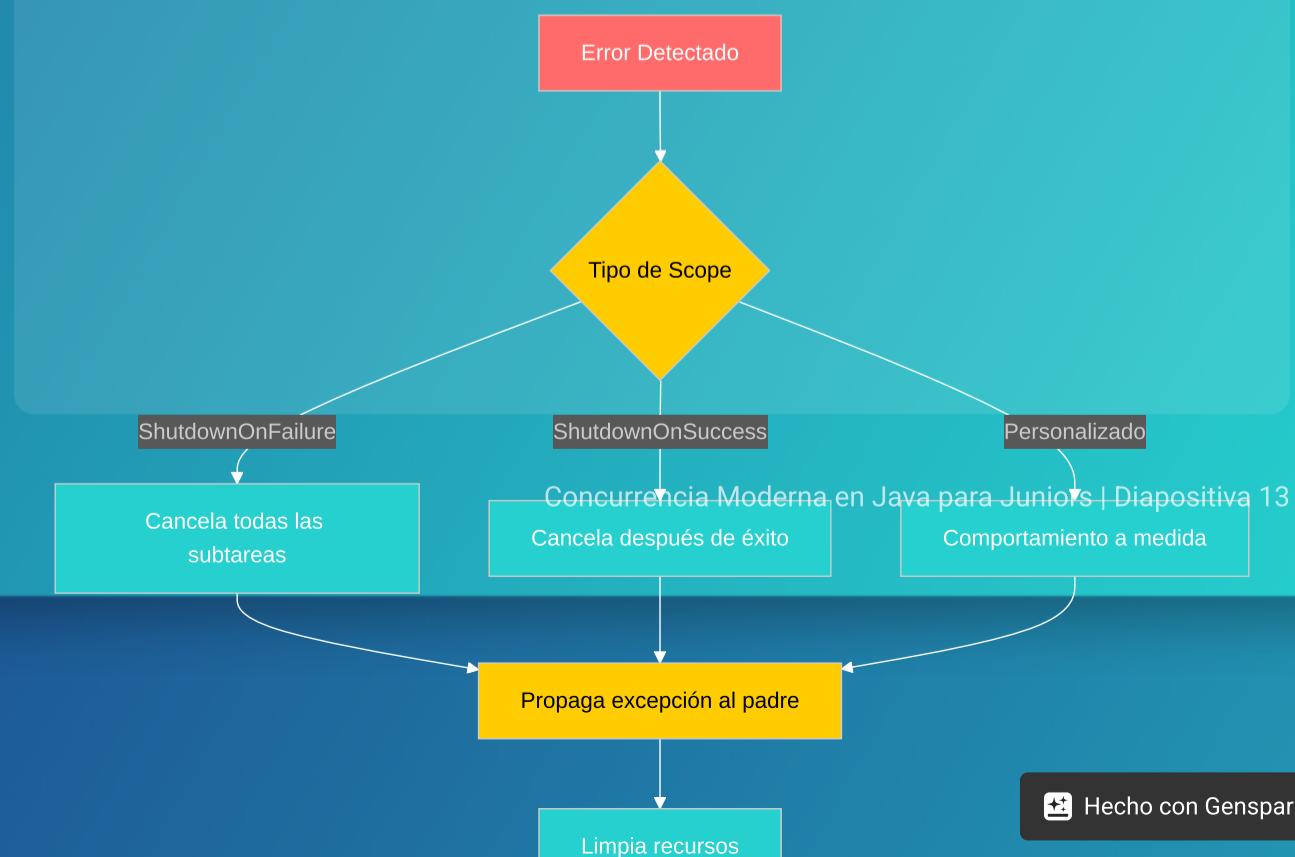
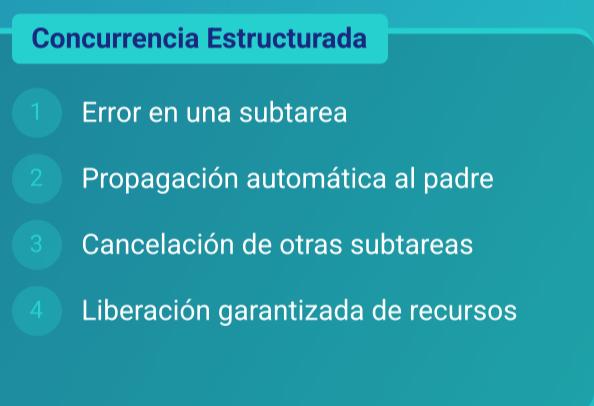
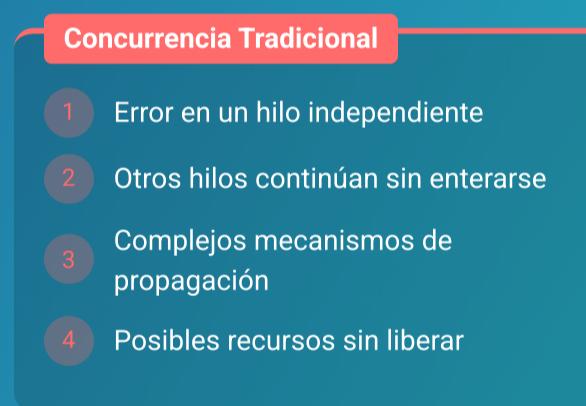
- Propagar el Error**
La excepción se transmite automáticamente al padre
- Cancelar Otras Tareas**
Opcionalmente, cancelar las demás tareas en el mismo ámbito
- Manejo Centralizado**
Tratar todas las excepciones de forma unificada en un solo lugar



“It gives us a new concept and API to create straightforward and easier-to-maintain concurrent code by treating tasks and their related sub-tasks as a single unit of work, giving us back control over error handling and cancellation”

⚙️ Implementaciones útiles:

- `ShutdownOnFailure` : cancela todas las tareas si alguna falla
- `ShutdownOnSuccess` : cancela tareas pendientes cuando una tiene éxito
- Personalizada: extiende `StructuredTaskScope` para comportamiento a medida



Scoped Values

Notas Temporales y Seguras

💡 Los **Scoped Values** permiten compartir datos **inmutables** de forma segura y eficiente:

Entre Métodos

Sin necesidad de pasarlo como parámetros

A Través de Hilos

Con herencia automática para hilos virtuales creados en el mismo ámbito

De Forma Segura

Los valores son inmutables, evitando efectos secundarios no deseados

💡 **Analogía:** Imagina una **pizarra compartida** en una reunión. Alguien escribe una **nota temporal** que es relevante para la discusión actual:

- Cada participante puede **leerla**
- Nadie puede **borrarla ni modificarla**
- Una vez que la discusión cambia, la nota **ya no es relevante**

“Therefore, Java 20 introduces the scoped values API as a solution to maintain immutable and inheritable per-thread data built to support millions of virtual threads”



ThreadLocal	
Mutabilidad	Mutable
Rendimiento	Más pesado
Herencia	Costosa
Memory Leaks	Posibles

ScopedValue	
Mutabilidad	Inmutable
Rendimiento	Optimizado
Herencia	Eficiente
Memory Leaks	Evitados

```
usando ScopedValues
// Definición del ScopedValue
static final ScopedValue<String> USERNAME = ScopedValue.newInstance();

// Estableciendo un valor dentro de un ámbito
ScopedValue.runWhere(USERNAME, "usuario123", () -> {

    // Accediendo al valor desde cualquier método en este ámbito
    System.out.println("Usuario: " + USERNAME.get());

    // También disponible en hilos virtuales lanzados dentro de este ámbito
    Thread.ofVirtual().start(() -> {
        System.out.println("En hilo virtual: " + USERNAME.get());
    });
});
```



Syntax error in text
mermaid version 11.6.0

Scoped Values vs. ThreadLocal

Mejor que el ThreadLocal Tradicional

⚠️ ThreadLocal

también permite compartir datos por hilo, pero:

ThreadLocal permite **datos mutables**, lo que aumenta el riesgo de efectos secundarios inesperados

La **herencia** entre hilos hijos es costosa y puede ser problemática

Fácil crear **fugas de memoria** si no se eliminan adecuadamente

✓ Scoped Values

mejora estos aspectos:

Datos **inmutables** que previenen efectos secundarios

Herencia eficiente con StructuredTaskScope, optimizada para hilos virtuales

Limpieza automática cuando termina el ámbito, evitando fugas de memoria

“Therefore, Java 20 introduces the scoped values API as a solution to maintain immutable and inheritable per-thread data built to support millions of virtual threads”

Característica	ThreadLocal	ScopedValue
Mutabilidad	Mutable ✅	Inmutable ✅
Herencia entre hilos	Costosa con InheritableThreadLocal	Eficiente, automática con StructuredTaskScope
Alcance (Scope)	Tiempo de vida del hilo	Delimitado explícitamente
Riesgo de fugas	Alto, requiere limpieza manual	Bajo, limpieza automática
Rendimiento	Degradoación con muchos hilos	Optimizado para millones de hilos virtuales

⌚ ThreadLocal

```
// Definición
private static final ThreadLocal<User> userContext =
    new ThreadLocal<>();

// Establecer valor (mutable)
userContext.set(new User("user123"));

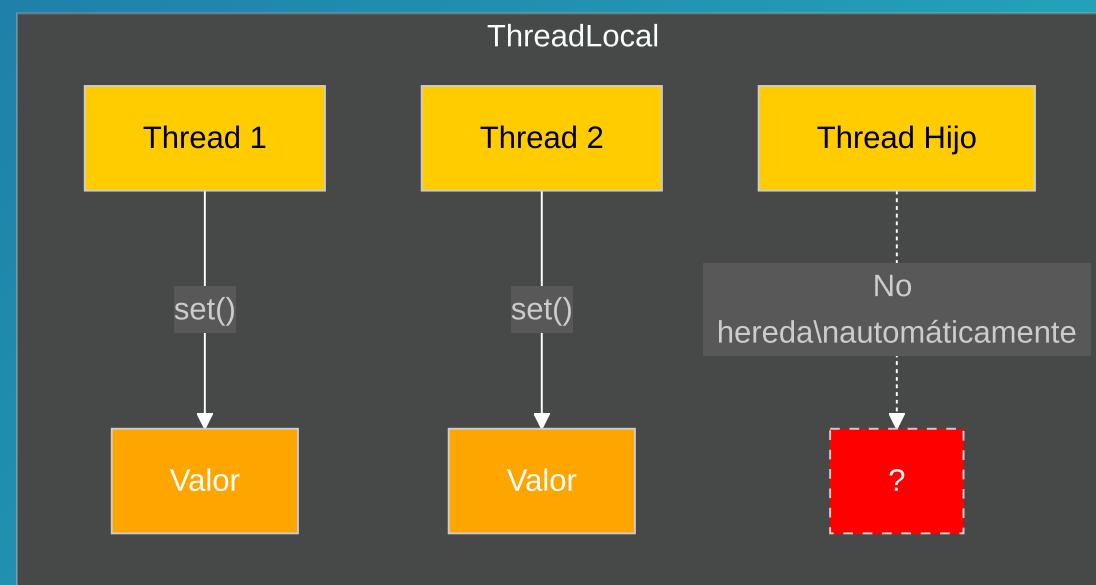
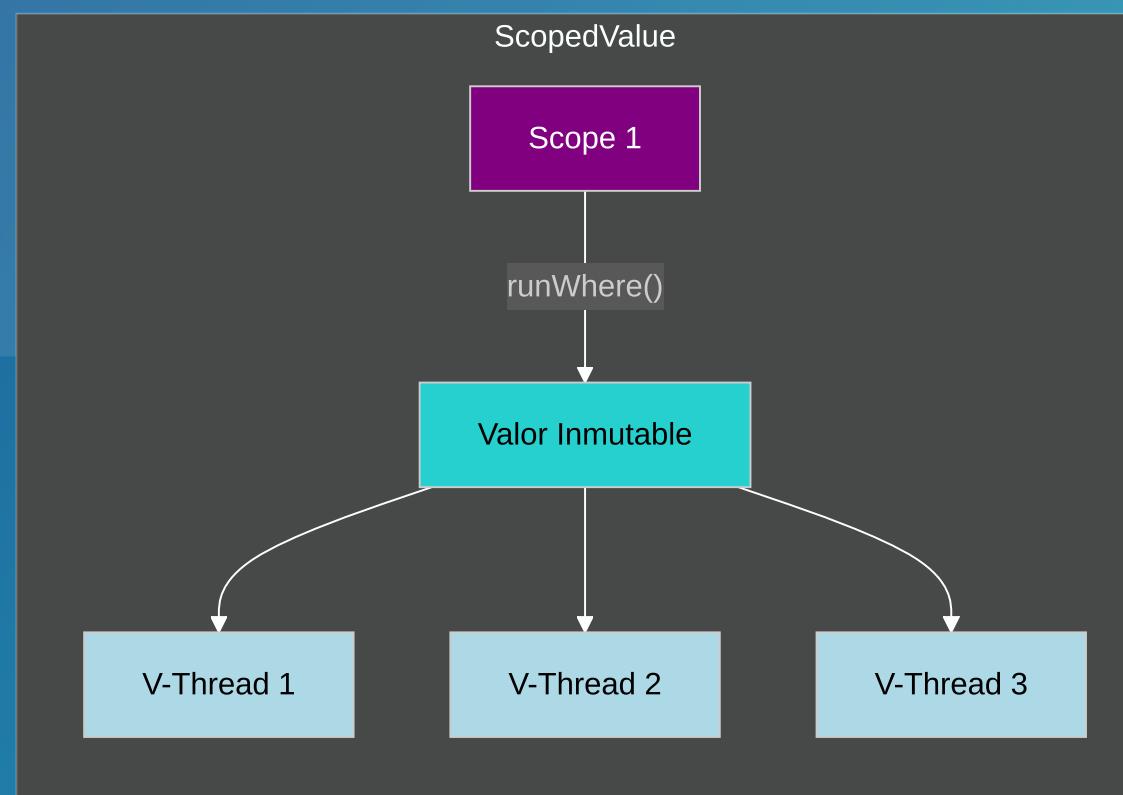
// La modificación es posible
```

☰ ScopedValue

```
// Definición
private static final ScopedValue<User> USER =
    ScopedValue.newInstance();

// Ámbito explícito e inmutable
ScopedValue.runWhere(USER, new User("user123"), () ->

// Solo lectura, sin modificación
```



Usando Scoped Values

Definiendo y Accediendo a Scoped Values

Usar **Scoped Values** requiere tres pasos fundamentales:

1 Definir el ScopedValue

Crea una instancia usando `ScopedValue.newInstance()`

2 Establecer valor dentro de un ámbito

Utiliza `ScopedValue.runWhere()` para definir un ámbito con un valor específico

3 Acceder al valor

Dentro del ámbito definido, usa `get()` para recuperar el valor

💡 Los hilos virtuales creados dentro del ámbito **heredan automáticamente** los Scoped Values

Casos de uso comunes:



Contexto de Usuario
ID de usuario, roles, preferencias



Seguridad
Tokens, permisos, contexto de autenticación



Transacciones
IDs de transacción, conexiones a BD



Localización
Idioma, preferencias regionales

Ejemplo completo de ScopedValue

```
// 1. Definir el ScopedValue
public static final ScopedValue<String> USERNAME = ScopedValue.newInstance();
public static final ScopedValue<String> REQUEST_ID = ScopedValue.newInstance();

// 2. Establecer valores dentro de un ámbito
public void processRequest(String username, String requestId) {
    ScopedValue.where(USERNAME, username)
        .where(REQUEST_ID, requestId)
        .run(() -> {
            // 3. Acceder a los valores dentro del ámbito
            System.out.println("Procesando solicitud para: " + USERNAME.get());
        });
}
```



Definición

Definir el contenedor inmutable para compartir datos



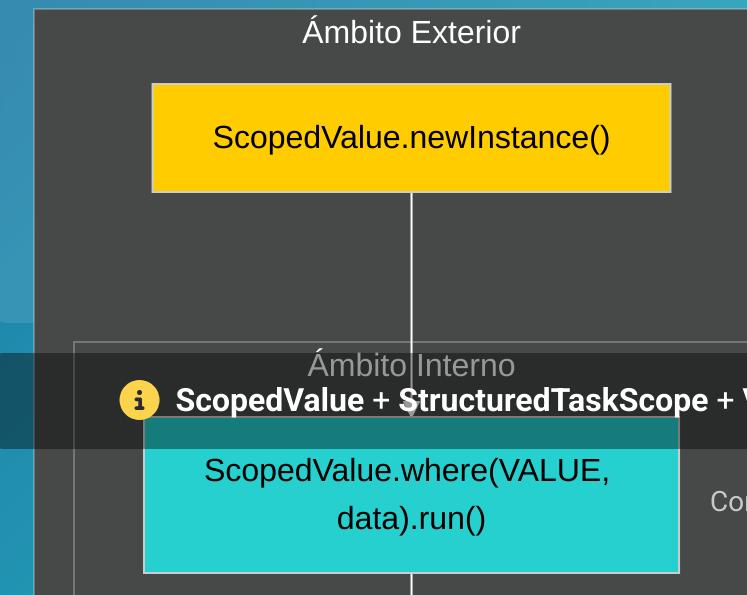
Ámbito

Definir un valor concreto durante la ejecución de un bloque



Acceso

Leer el valor desde cualquier punto del ámbito



Concurrencia Moderna en Java para Juniors | Diapositiva 16

El Poder de la Combinación (Project Loom)

Trabajando Juntos para un Mejor Rendimiento

- 💡 Los hilos virtuales, la concurrencia estructurada y los scoped values son las piezas clave de **Project Loom**.

Estas tecnologías trabajan juntas para hacer que la programación concurrente en Java sea más:



Más Fácil

Código más legible y mantenible, con modelos de programación intuitivos



Más Segura

Mejor gestión de errores, datos inmutables y controles estructurados claros



Más Eficiente

Mayor concurrencia, mejor uso de recursos y bloqueo sin costo

Analogía

En la cocina:

- Los **ayudantes eficientes** (hilos virtuales) trabajando bajo
- La dirección de **equipos organizados** (conurrencia estructurada) y
- Compartiendo **información relevante** de forma segura (scoped values)
- Hacen que la cocina (tu programa) funcione mucho mejor

Hilos Virtuales
Concurrencia ligera y eficiente

- Millones de hilos concurrentes
- Uso mínimo de recursos
- Bloqueo sin costos

Concurrencia Estructurada
Organización y control

- Manejo de errores unificado
- Cancelación controlada
- Limpieza automática

Scoped Values
Compartir datos con seguridad

- Valores inmutables
- Herencia automática
- Sin fugas de memoria



Syntax error in text
mermaid version 11.6.0

Java 21 (y posteriores)

El Presente y el Futuro de la Conurrencia en Java

⌚ Los hilos virtuales y la concurrencia estructurada fueron introducidos como características de **previsualización** en versiones anteriores de Java y se han ido refinando.

➡ Estado actual:

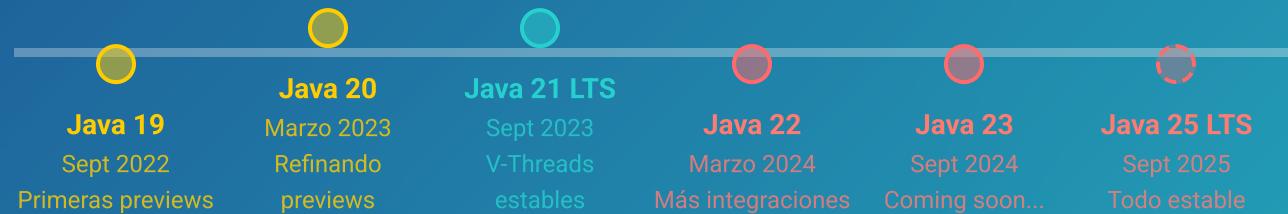
- **Hilos Virtuales**: característica estable en Java 21 (LTS)
- **Concurrencia Estructurada**: característica de previsualización en Java 21
- **Scoped Values**: característica de previsualización en Java 21

“*This is a preview feature. A preview feature is a feature whose design, specification, and implementation are complete, but is not permanent*”

💡 Usa hoy, aprovecha mañana:

Para usar características de previsualización, necesitas habilitar las opciones correspondientes al compilar y ejecutar.

```
// Compilación con una característica de previsualización  
javac --enable-preview --release 21 MiApp.java  
  
// Ejecución con una característica de previsualización  
java --enable-preview MiApp
```



Característica	Java 19	Java 20	Java 21 (LTS)	Java 22+
Hilos Virtuales	Preview 1	Preview 2	Estable	Estable
Concurrencia Estructurada	Preview 1	Preview 2	Preview 3	Estable*
Scoped Values	Incubator	Preview 1	Preview 2	Estable*

* Previsión estimada

💡 JEPs Relevantes (Java Enhancement Proposals)

JEP 444

Virtual Threads

JEP 453

Structured Concurrency (Preview)

JEP 446

Scoped Values (Preview)

JEP 425/437

Virtual Threads (Incubator/Preview)

Aprovecha el futuro hoy



Java 21 LTS te permite usar hilos virtuales en producción y experimentar con las demás características de Project Loom en entornos de desarrollo.

Puntos Clave

Recapitulando



Hilos Virtuales

Hilos **ligeros** que permiten una **alta concurrencia** con menos recursos:

- Millones de hilos concurrentes posibles
- Menos overhead de memoria (KB vs MB)
- Bloqueo sin costo en operaciones I/O



Concurrencia Estructurada

Organiza las tareas concurrentes para un **mejor control** y manejo de errores:

- Propagación de errores consistente
- Cancelación coordinada de tareas
- Limpieza automática de recursos



Scoped Values

Permiten compartir datos **inmutables** de forma **segura** entre hilos:

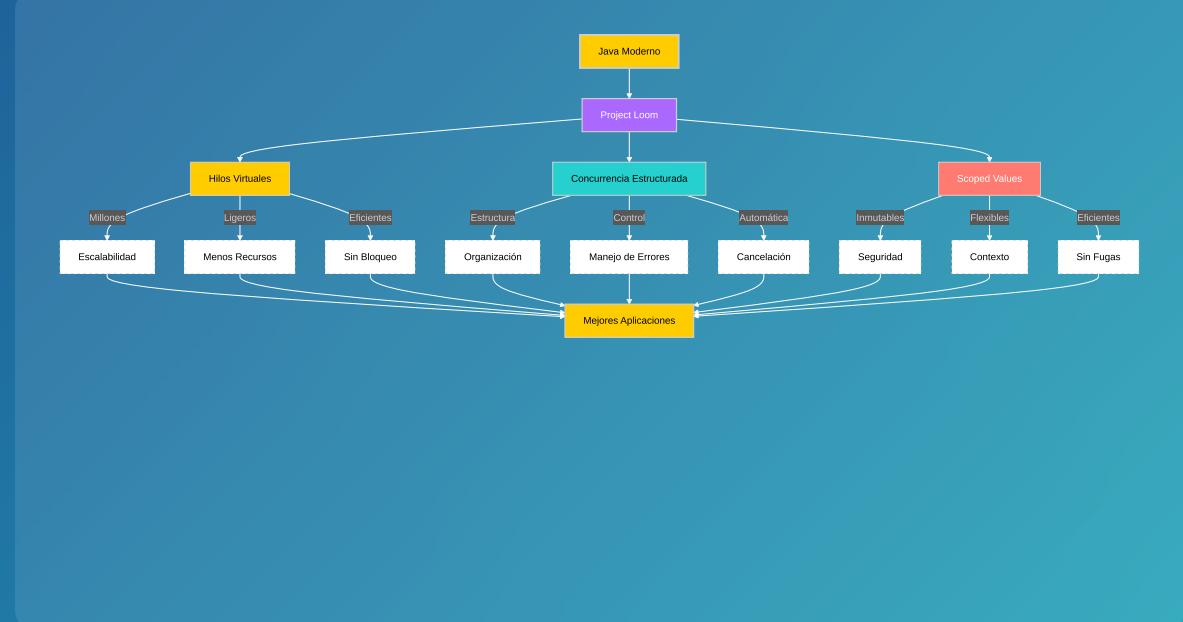
- Evita modificaciones no deseadas
- Herencia automática entre hilos
- Sin fugas de memoria



Project Loom

Introduce estas características para **simplificar** y **mejorar** la programación concurrente:

- Disponible en Java 21 LTS (Virtual Threads)
- Concurrencia fácil, segura y eficiente
- Funciona con el modelo de programación familiar



★ Concurrencia Moderna en Java

El modelo de **concurrencia ligera** con **Project Loom** permite crear aplicaciones altamente escalables, mantenibles y eficientes sin sacrificar la simplicidad del código.



Hilos Concurrentes



Código Sencillo



Alto Rendimiento



Mayor Seguridad

¿Qué Sigue?

A Seguir Aprendiendo

1

Explorar las APIs

Familiarízate con las nuevas APIs de concurrencia:

- `Thread.ofVirtual()` - Para crear hilos virtuales
- `ExecutorService.newVirtualThreadPerTaskExecutor()` - Para ejecutores
- `StructuredTaskScope` - Para concurrencia estructurada

2

Experimentar con Ejemplos

Practica con ejemplos concretos para entender mejor los conceptos:

- Convertir aplicaciones existentes de hilos tradicionales a virtuales
- Implementar patrones de concurrencia estructurada
- Usar Scoped Values para compartir contextos entre hilos

3

Profundizar en la Documentación

Consulta los JEPs (Java Enhancement Proposals) y la documentación oficial:

- JEP 444: Virtual Threads
- JEP 453: Structured Concurrency
- JEP 446: Scoped Values
- Javadocs de las nuevas APIs en Java 21+

4

Aplicar en Proyectos Reales

Implementa lo aprendido en tus propios proyectos:

- Mejora el rendimiento de aplicaciones con E/S intensiva
- Simplifica tu código concurrente con patrones estructurados
- Compara el rendimiento antes y después de la migración

API y Repositorios



API y Repositorios

Explore ejemplos de código en repositorios oficiales de Oracle y la comunidad. La API de Java 21 incluye documentación detallada sobre hilos virtuales, StructuredTaskScope y ScopedValue.

Documentación Oficial



Documentación Oficial

La documentación oficial de Java 21 incluye guías completas, tutoriales y ejemplos prácticos sobre las nuevas características de concurrencia. Visita docs.oracle.com para más información.

Libros y Tutoriales



Libros y Tutoriales

Autores como Heinz Kabutz, Brian Goetz, y Venkat Subramaniam han publicado excelentes recursos sobre concurrencia moderna en Java. Busca también cursos en plataformas de aprendizaje en línea.

Comunidad y Conferencias



Comunidad y Conferencias

Participa en foros como Stack Overflow, Reddit r/java, y conferencias como Devoxx, JavaOne, y JCrete para aprender de expertos en concurrencia de Java.



Enlaces Rápidos



Inside Java Newscast (YouTube)



[GitHub: java/loom-playground](https://github.com/java/loom-playground)



[Virtual Threads Deep Dive \(Oracle\)](#)



[Java Concurrency in Practice 2nd Ed.](#)