Habits

John Doe

March 22, 2005

Script: 40-Minute Interactive Presentation

Structured Concurrency + Scoped Values in Java 25

Total Duration: 35-40 minutes (core content) + 10-20 minutes Q&A **Format**: Code-driven, interactive demos with minimal slides **Key Equipment**: IDE + Terminal + Web UI + PDF slides

---

Pre-Presentation Setup Checklist

Terminal Setup

- **Terminal 1**: `cd demo-structured-concurrency` (for Gradle commands)
- **Terminal 2**: `./gradlew quarkusDev` (start 10 min before talk)

Browser Setup

- **Tab 1**: http://localhost:8080 (Web UI)
- Verify cards are loaded
- Create test cards:
  - Valid card: Any with balance > 0, expiration "2512"
  - **Expired card**: BIN 1111, expiration "1220" (critical for fail-fast demo)
  - Low balance card: Amount > 1000

IDE Setup (Files open in tabs, in order)

1. `ReactivePaymentProcessor.java`
2. `StructuredPaymentProcessor.java`
3. `FailFastStructuredPaymentProcessor.java`
4. `ReactivePaymentProcessorFailFast.java` (for comparison)
5. `ScopedPaymentProcessor.java`
6. `ScopedBalanceService.java`

PDF Presentation

- Open in full-screen mode
- Know slide transitions to demos

Screen Layout Practice

- **Primary display**: PDF slides
- **Secondary/switch to**: IDE + Browser + Terminal
- Practice smooth switching

Timing Overview (35-Minute Core + Optional)

Time	Section	Type	Slides
0:00-1:00	About Me	Slide	1
1:00-5:00	Introduction	Slides	4-5
5:00-6:00	Reactive Problem	Slide	1
6:00-11:00	Reactive Demo	Live Code	0
11:00-12:00	Structured Solution	Slide	1
12:00-17:00	Structured Demo	Live Code	0
17:00-18:00	Fail-Fast Concept	Slide	1
18:00-24:00	**FAIL-FAST DEMO**	**Live Code**	**0**
24:00-25:00	Performance Chart	Slide	1
25:00-30:00	**Scoped Values**	**Live Code**	**1**
30:00-33:00	Conclusions	Slides	2
33:00-38:00	**Advanced Features**	**Slides (Optional)**	**5**
38:00-50:00+	Q&A	Interactive	0

[0:00-1:00] Section 1: About Me (1 slide)

Slide: About Me

Script: > "Hola, soy [nombre]. Hoy vamos a hablar de dos funcionalidades clave en Java 25: **Structured Concurrency** y **Scoped Values**. Pero no vamos a ver muchos slides con código… vamos a ver el código funcionando en vivo."

Transition: → Next slide

[1:00-5:00] Section 2: Introduction (4-5 slides)

Slides: - Evolution of Concurrency diagram (TikZ) - Payment Transaction Flow - Problem statement

Script: > "Primero, un poco de contexto. La concurrencia en Java ha evolucionado desde threads básicos, pasando por ExecutorService, Streams paralelos, CompletableFuture… y ahora llegamos a **Structured Concurrency**." > > *[Show evolution diagram]* > > "Para demostrar estas funcionalidades, vamos a usar un caso de uso realista: **procesamiento de transacciones de pago**." > > *[Show payment flow diagram]* > > "Tenemos validaciones paralelas: merchant, card, balance, PIN, expiration. Algunas son independientes, otras dependen de resultados previos. Esto es perfecto para mostrar la diferencia entre el approach reactivo tradicional y la concurrencia estructurada."

Key Points: - Mention **both features** in title: Structured Concurrency **AND** Scoped Values - Set expectations: "Vamos a ver mucho código en vivo"

Transition: → "Empecemos con el approach tradicional… **reactive programming**"

[5:00-6:00] Section 3: Reactive Problem (1 slide)

Slide: Problemas del Enfoque Reactivo

Script: > "El enfoque reactivo con CompletableFuture funciona… pero tiene problemas:" > > *[Read bullet points]*: > - Callback hell - lógica anidada difícil de seguir > - Manejo de errores complejo con CompletionException > - Stack traces confusos > - Alta carga cognitiva > > "Veamos esto en código real…"

Transition: **Switch to IDE**

[6:00-11:00] Section 4: Reactive Demo (LIVE CODE - 5 min)

Part 1: Code Walkthrough in IDE (2 min)

File: `ReactivePaymentProcessor.java`

Script: > "Aquí tenemos el procesador reactivo. Veamos la complejidad:" > > *[Navigate through code, point out]*: > - "PATH A y PATH B ejecutan en paralelo" (line ~8-12) > - "Nested parallel validations con `.stream().map(CompletableFuture::supplyAsync)`" (line ~15-17) > - "allOf() para esperar todas" (line ~21) > - "Manual error handling con `.thenCompose()`, `.map()`, `.findFirst()`" (line ~22-28) > - "Release manual del balance lock" (line ~24) > > "Son ~30 líneas aquí, pero si agregamos fail-fast manual… **80 líneas**."

Part 2: Terminal Demo (1 min)

Terminal: `./gradlew demoReactive`

Script: > "Corramos esto:" > > *[Run command, wait for output]* > > "Funciona. ~520ms. Pero miren el código que tuvimos que escribir…"

Part 3: Web UI Demo - Valid Transaction (2 min)

Browser: http://localhost:8080

Script: > "Ahora veámoslo en la interfaz web. Aquí tengo tarjetas de prueba con balances." > > *[Select valid card, click "Usar"]* > > "Selecciono una tarjeta válida, elijo 'Reactive: Basic', y proceso…" > > *[Click "Procesar", wait for result]* > > "Red box: Success, ~520ms. Funciona… pero ya vimos la complejidad del código."

Transition: → "¿Hay una mejor forma? Sí: **Structured Concurrency**" → **Back to slides**

[11:00-12:00] Section 5: Structured Solution (1 slide)

Slide: Structured Concurrency = El Código Se Lee Como Se Ejecuta

Script: > "Structured Concurrency cambia todo:" > > *[Read key principles]*: > - El código se lee como se ejecuta (linear) > - Excepciones tradicionales (no más CompletionException wrapping) > - Try-with-resources automático > - Fail-fast **por defecto** > > "Y lo más importante: **mismo performance**, la mitad del código."

Transition: **Switch to IDE**

[12:00-17:00] Section 6: Structured Demo (LIVE CODE - 5 min)

Part 1: Code Walkthrough in IDE (2 min)

File: `StructuredPaymentProcessor.java`

Script: > "Aquí está la versión con Structured Concurrency. Comparen:" > > *[Navigate through code, point out]*: > - "try-with-resources con `StructuredTaskScope.open()`" (line ~6) > - "`scope.fork()` para PATH A y PATH B" (line ~8, ~10) > - "Nested scope para validaciones paralelas" (line ~14-18) > - "`scope.join()` espera automáticamente" (line ~22) > - "Lógica de negocio clara, sin callbacks" (line ~22-26) > > "~**40 líneas total**. La mitad que reactive. Y es mucho más fácil de leer."

Part 2: Terminal Demo (1 min)

Terminal: `./gradlew demoStructured`

Script: > "Ejecutemos:" > > *[Run command, wait for output]* > > "~520ms. **Mismo performance que reactive**, pero con código mucho más simple."

Part 3: Web UI Demo - Valid Transaction (2 min)

Browser: http://localhost:8080

Script: > "En la web UI, selecciono 'Structured: Normal' y proceso la misma tarjeta…" > > *[Click "Procesar", wait for result]* > > "Green box: Success, ~520ms. Paridad de performance. Pero miren qué código más limpio."

Transition: → "Pero ahora viene **la funcionalidad estrella**: Fail-Fast" → **Back to slides**

[17:00-18:00] Section 7: Fail-Fast Concept (1 slide)

Slide: La Funcionalidad Estrella: Fail-Fast

Script: > "Aquí es donde Structured Concurrency realmente brilla." > > *[Read comparison]*: > - **Reactive**: `allOf()` espera TODAS las tareas, incluso si una falla (~570ms) > - **Reactive Fixed**: Se puede hacer fail-fast… pero requiere ~80 líneas y coordinación manual > - **Structured**: `open()` sin joiner = **fail-fast automático**. ~40 líneas. Cancelación en cascada. > > "Esta es **la demo más importante** de la presentación. Vamos a ver la diferencia dramática…"

Transition: **Switch to Web UI** (already open)

[18:00-24:00] Section 8: FAIL-FAST DEMO (LIVE CODE - 6 min)

Part 1: Setup the Scenario (1 min)

Browser: http://localhost:8080

Script: > "Voy a crear una tarjeta **expirada**. Expiration: 1220 (diciembre 2020)." > > *[Create/select card with expiration "1220"]* > > "La validación de expiration falla en ~200ms. Pero hay otras validaciones que toman 400-500ms." > > "**Pregunta**: ¿Debemos esperar esas validaciones si ya sabemos que la transacción falló?"

Part 2: Reactive Demo - Waits for All (2 min)

Browser: Select "Reactive: With Exceptions"

Script: > "Primero, veamos el approach reactivo con exceptions:" > > *[Click "Procesar", wait for result]* > > "Red box: Failed, ~**570ms**." > > "¿Por qué 570ms si la expiration falló en 200ms? Porque `allOf()` **espera que todas las tareas terminen**, incluso después del primer fallo."

Part 3: Structured Demo - Automatic Cancellation (2 min)

Browser: Select "Structured: Fail-Fast Automatic"

Script: > "Ahora, con Structured Concurrency:" > > *[Click "Procesar", wait for result]* > > "Green box: Failed, ~**230ms**." > > *[Point to improvement banner]*: > > "**60% más rápido!** ¿Por qué? Porque cuando la expiration falla, `StructuredTaskScope` **cancela automáticamente las demás tareas**. No esperamos trabajo innecesario."

Part 4: Terminal Confirmation (1 min)

Terminal: `./gradlew demoCompareFailure`

Script: > "Confirmemos esto en terminal:" > > *[Run command, show output]* > > "Reactive: ~570ms. Structured: ~230ms. **60% improvement on failure**."

Part 5: Code Explanation (Optional, if time) (1 min)

IDE: `FailFastStructuredPaymentProcessor.java` vs `ReactivePaymentProcessorFailFast.java`

Script: > "Y miren el código:" > > *[Show Structured version]*: > - "Structured: Just `StructuredTaskScope.open()` - sin joiner = fail-fast automático. ~40 líneas." > > *[Show Reactive version - if time]*: > - "Reactive: AtomicBoolean,

manual coordination, completeExceptionally()… ~80 líneas.” > > “**Cero líneas extra para fail-fast** vs 80 líneas de coordinación manual.”

Key Message: “60% más rápido en errores, sin código extra. Esto es el poder de Structured Concurrency.”

Transition: → “Veamos el resumen de performance…” → **Back to slides**

[24:00-25:00] Section 9: Performance Chart (1 slide)

Slide: Resumen de Performance: Éxito vs Fallo

Script: > “Resumen de performance:” > > *[Point to left column]*: > - “Caso exitoso: Reactive ~520ms, Structured ~520ms. **Paridad**.” > > *[Point to right column + chart]*: > - “Caso fallido: Reactive ~570ms, Structured ~230ms. **60% más rápido**.” > > *[Point to TikZ chart]*: > > “Aquí está la diferencia visual. Structured es dramáticamente más rápido cuando hay errores.” > > “Y esto importa: mejor UX, feedback inmediato, menor uso de recursos. **Esto es arquitectural**, no solo performance.”

Transition: → “Ahora hablemos del segundo feature del título: **Scoped Values**” → Next slide

---

[25:00-30:00] Section 10: Scoped Values (LIVE CODE - 5 min)

Part 1: The Problem (From Slide) (1 min)

Slide: Scoped Values = Contexto Sin Parameter Drilling

Script: > “Tenemos validaciones paralelas. ¿Cómo pasamos **contexto** (audit info, request ID, user context) a través de todas ellas?” > > *[Read problems]*: > - “**Parameter drilling**: Pasar en cada firma de método (verbose, invasivo)” > - “**ThreadLocal**: Problemático con hilos virtuales masivos” > > “La solución: **Scoped Values**.”

Transition: **Switch to IDE**

Part 2: Code Walkthrough - ScopedValue Binding (2 min)

File: `ScopedPaymentProcessor.java`

Script: > “Aquí definimos un `ScopedValue<TransactionRequest>`:” > > *[Point to line ~6]*: > - “`public static final ScopedValue<TransactionRequest> TRANSACTION_REQUEST = ScopedValue.newInstance();`” > > *[Point to binding, line ~8]*: > - “`ScopedValue.where(TRANSACTION_REQUEST, request).call(() -> { ... })`” > - “Esto **propaga automáticamente** el request a todas las tareas forked.” > > *[Point to service calls, line ~15]*: > - “`balanceService::validate` - **sin parámetros!**” > > “Ningún método necesita recibir el `request` como parámetro.”

File: `ScopedBalanceService.java`

Script: > “Ahora veamos el servicio de balance:” > > *[Point to line ~32]*: > - “`TransactionRequest request = ScopedPaymentProcessor.TRANSACTION_REQUEST.get();`” > - “**Acceso directo** al contexto, sin pasar parámetros.” > > *[Point to audit logs, lines ~34, ~37, ~41]*: > - “Todos los audit logs usan el mismo contexto propagado automáticamente.”

Part 3: Terminal Demo (1 min)

Terminal: `./gradlew demoScopedValues`

Script: > “Ejecutemos esto y veamos los audit logs:” > > *[Run command, show output with audit logs]* > > “Cada validación tiene acceso al contexto sin que lo pasemos explícitamente. **Propagación automática**.”

Part 4: Emphasize Stability (1 min)

Slide: (Already showing - emphasize the alert block)

Script: > *[Point to alertblock on slide]*: > > "Y esto es **crítico**: A diferencia de Structured Concurrency (que está en 5th preview), **Scoped Values son ESTABLES en Java 25**." > > "**JEP 506** está finalizado. Pueden usar esta funcionalidad **en producción hoy**." > > "Casos de uso: audit trails, request IDs, user context, distributed tracing… todo sin parameter drilling."

Key Message: "Contexto resuelto. Production-ready en Java 25."

Transition: → "Veamos las conclusiones…" → Next slide

[30:00-33:00] Section 11: Conclusions (2 slides)

Slides: Conclusions + Roadmap

Script: > "Recapitulemos:" > > *[Slide 1: Key messages]*: > - "**Reactive**: Complejo, espera todas las validaciones, 80 líneas para fail-fast" > - "**Structured**: Simple, fail-fast automático, 40 líneas, **60% más rápido en errores**" > - "**Scoped Values**: Contexto sin parameter drilling, **STABLE hoy**" > > *[Slide 2: Roadmap]*: > - "**Structured Concurrency**: JEP 484, 5th preview. Production-ready muy pronto." > - "**Scoped Values**: JEP 506, **STABLE**. Usen esto hoy con `--enable-preview`." > > "La combinación de estas dos funcionalidades: **código paralelo limpio con contexto**."

Decision Point: Check time - **If 33-35 min**: Skip Advanced Features → Go to Q&A - **If 30-33 min**: Include Advanced Features (5 min) - **If <30 min**: Include full Advanced Features

Transition: - **If including Advanced Features**: → "Para los técnicos, veamos algunas features avanzadas…" → Next section - **If skipping**: → "¿Preguntas?" → Q&A

[33:00-38:00] Section 12: Advanced Features (OPTIONAL - 5 min)

Slides: Section 07 - Advanced Features (5 slides)

Script: > "Si les interesa la profundidad técnica, aquí hay algunas features avanzadas:"

Slide 1: Custom Joiners

> "Pueden crear **custom joiners** para lógica de agregación específica. Por ejemplo, recopilar solo los primeros N resultados exitosos."

Slide 2: Nested Scopes

> "**Scopes anidados** permiten estructurar lógica compleja. Ya vimos esto: PATH A y PATH B en paralelo, luego validaciones nested dentro de PATH B."

Slide 3: Timeouts

> "Pueden agregar **timeouts** con `.joinUntil(deadline)`. Si alguna tarea tarda más del deadline, se cancelan todas."

Slide 4: Anti-patterns

> "Y eviten estos anti-patterns: no cierren scopes manualmente (use try-with-resources), no guarden referencias a Subtasks fuera del scope…"

Script: (Brief walkthrough, no live demos)

Transition: → "Eso es todo. ¿Preguntas?" → Q&A

[38:00-50:00+] Q&A (Interactive)

Standard Q&A Format

Script: > "¿Preguntas?"

Potential Questions & Responses:

1. **"¿Cuándo puedo usar esto en producción?"** > "**Scoped Values: hoy**. Son estables en Java 25. Para Structured Concurrency, está en 5th preview. Dado el progreso (5 previews), probablemente finalice en Java 26-27."
2. **"¿Funciona con Kotlin/Scala/Groovy?"** > "Scoped Values sí, son una feature del JDK. Structured Concurrency también, aunque esos lenguajes tienen sus propios mecanismos (coroutines en Kotlin, por ejemplo)."
3. **"¿Hay overhead de performance?"** > "Virtual threads tienen overhead mínimo (~1KB vs ~2MB). Structured Concurrency tiene overhead similar a ExecutorService. En nuestras demos, vimos paridad de performance en casos exitosos."
4. **"¿Cómo manejo errores de múltiples tareas?"** > "Con `ShutdownOnFailure`, obtienes la primera exception. Con custom joiners o `awaitAll()`, puedes recopilar todas las exceptions y procesarlas."

Bonus: Web UI Deep Dive (If Audience Engaged)

Browser: http://localhost:8080

Script: > "¿Quieren ver algo más interactivo? Puedo crear tarjetas de prueba en vivo y probar diferentes escenarios."

Actions: - Create new card with specific balance - Test invalid PIN scenario - Test insufficient balance scenario - Show how each demonstrates fail-fast

Audience-Driven: > "¿Qué escenario quieren probar?" > > *[Create card based on suggestion, run comparison]*

Decision Points During Presentation

At 24 minutes (after Fail-Fast demo):

- **On schedule**: Proceed with Performance Chart → Scoped Values → Conclusions
- **Behind 2-3 min**: Skip Performance Chart slide, proceed directly to Scoped Values
- **Behind 5+ min**: Shorten Scoped Values demo (show code in IDE only, skip terminal demo)

At 30 minutes (after Scoped Values):

- **On schedule**: Proceed to Conclusions, then Advanced Features optional module
- **Behind**: Go straight to Conclusions, skip Advanced Features
- **Ahead**: Full Conclusions + Advanced Features

At 33 minutes (after Conclusions):

- **Significantly ahead**: Include Advanced Features module (5 min)
- **Slightly ahead**: Brief Advanced Features highlights (2-3 min)
- **On track or behind**: Skip to Q&A

During Q&A:

- **If audience engaged**: Web UI deep dive, live scenario testing
- **If technical questions**: Show relevant code sections in IDE
- **If time constrained**: Standard Q&A format

Key Messages to Emphasize Throughout

1. **Simplicity**: "Half the code, same performance"
2. **Fail-Fast**: "60% faster on errors, automatically"
3. **Readability**: "Code reads as it executes"
4. **Scoped Values - Production Ready**: "STABLE in Java 25, use them in production TODAY"
5. **Structured Concurrency - Coming Soon**: "5th preview, production-ready very soon"
6. **Perfect Together**: "Scoped Values + Structured Concurrency = Clean parallel code with context"

Emergency Backup Plans

Technical Risks:

- **Quarkus not starting**: Have terminal demos ready, use screenshots of Web UI
- **Web UI slow/errors**: Fall back to terminal demos exclusively
- **Code navigation slow**: Use IDE bookmarks, practice transitions
- **Demo failure**: Have screenshots of successful runs as backup slides

Timing Risks:

- **Running over at 25 min**: Skip Performance Chart slide, shorten Scoped Values to 3 min (code only, no terminal)
- **Running over at 30 min**: Skip Advanced Features entirely
- **Audience questions mid-talk**: Budget 2-3 min for questions, compress remaining sections

---

Success Metrics Checklist

- Audience sees 60% fail-fast improvement in Web UI (KEY MOMENT)
- Code comparison is clear (80 lines vs 40 lines visible in IDE)
- Interactive element engages audience (Web UI cards, live demos)
- **Scoped Values demo clearly shows context propagation** (REQUIRED)
- Audience understands Scoped Values are STABLE (production-ready)
- Timing stays within 35 minutes for core content
- Q&A has sufficient time (10+ minutes)

Post-Presentation Notes Section

[Use this space during practice runs to note:] - Timing adjustments needed - Difficult transitions - Questions that came up - Demos that need more/less time - Slides that need tweaking

---

END OF SCRIPT