Hola!

Gracias por acompañar a nuestra líder técnica Laura Camacho en su #S4NTalk "Programación funcional usando Java".

A continuación encuentras las respuestas a las preguntas que nos llegaron a lo largo de la charla.

¿Existe algún tipo de problema que no pueda resolver con programación funcional?

Los paradigmas OO y FP no son de propósito específico. Quizá los lenguajes que sigan o implementen uno y otro paradigma podrán ser de propósito específico. En este orden de ideas, y para responder concretamente la pregunta, no hay un problema en computación que pueda ser solucionado vía programación que esté fuera del alcance de FP, como tampoco lo está por fuera de OO y como tampoco lo está por fuera de la programación estructurada. Una respuesta formal y bien desarrollada a esta pregunta la puede encontrar en la tesis de Church-Turing.

¿En dónde deberían estar definidas las funciones para poder reutilizarlas en otros objetos?

En Java nada puede estar definido por fuera de una clase. La única forma en la que se puede usar bien datos o comportamientos es en clases (omitamos las enumeraciones de la discusión). Así las cosas, las funciones deben estar definidas dentro de clases que serán luego o instanciadas en un objeto para su uso o usadas directamente como métodos de clase si están declaradas como *static*.

Con respecto al ejemplo de la preparación del plato de comida. ¿Qué diferencia existiría entre el ejemplo y aplicar un patrón de Chain of responsability?

En OO el catálogo más conocido y estándar de facto para definición de patrones es el de GoF (ver Gamma et al). En FP se suele decir que cualquier patrón de GoF se reduce (y es superado por) en FP a composición de funciones. En el caso específico de Chain of Responsibility la diferencia se explica fácilmente al revisar la firma del método que deben implementar los *Handler* concretos. En la definición abstracta de *Handler* se está obligando a que todos los concretos obedezcan dicha firma, lo que significa que deben tener el mismo nombre, los mismos parámetros, el mismo tipo de retorno (si es que no es *void*) y lanzar las mismas excepciones. Cuando se componen funciones esta restricción no existe; la única restricción existente consiste en que el tipo al que evalúa *g(y)* corresponda con el tipo que espera *f(x)* en el parámetro *x*.

En la explicación de composición de funciones surgió esta pregunta: ¿y si cambiamos el orden no da igual? finalmente retorna un Int

La composición de funciones no es conmutativa, por lo tanto si se cambia el orden de la composición, cambiaría el orden en el que se evalúan la funciones. Puede ver la diferencia en este test.

(sic) La JVM es criticada por performance comparado con emergentes que apalancan desarrollo oriented to microservicios usando lambdas… ¿qué opinas de estas tech/lenguajes emergentes que ofrecen un mejor performance pero al parecer el foco no es Functional pero que evitan hacer cosas como heartbeats sobre mis componentes en la nube? (pongo un ejemplo golang)

La respuesta a la pregunta puede resultar en una larga conversación alrededor del proceso de análisis de atributos de calidad y toma de decisiones de arquitectura. En una reducción simplista de la pregunta y alejándome de responder desde la opinión podría decir lo siguiente:

- El lenguaje de programación es una decisión de diseño detallado que viene luego de decidir sobre los componentes que conformarán (al menos inicialmente) un producto o sistema de software. En conjunto a la decisión de los componentes se debe proceder a definir la integración entre dichos componentes balanceando las necesidades funcionales con las no funcionales y tomando las decisiones técnicas correspondientes (protocolos, sincronía/asincronía, estilos y patrones de integración, formato de los mensajes).
- Si este análisis concluye que se debe favorecer una arquitectura Serverless se debe considerar el tiempo de inicio del entorno de cómputo. En este aspecto, la JVM tiene un punto que no le favorece, pues como lo enuncia la pregunta, es lento para iniciar y *warm up*.
- Que el foco sea o no sea funcional en una arquitectura serverless parece intentar comparar peras con manzanas. Para ser sucinta en la respuesta: lo que *normalmente* se usa en esta arquitectura de referencia como lenguaje de programación es JS sobre V8 (NodeJS). Hacer FP en JS es totalmente posible, pues cuenta con HoF como las describí en la charla. Si sobre JS quiere usar FP con patrones funcionales puede usar librerías como RamdaJS o similares.
- En conclusión, una cosa es hablar de arquitectura y otra cosa es hablar de diseño detallado. Serverless nos lleva a una conversación de arquitectura. FP nos lleva a una conversación de diseño detallado. Ambas conversaciones se complementan y no enfocan o desenfocan.

- Como nota adicional, hace poco (Marzo 29 de 2020) se hizo relevante en HN esta lectura acerca de cold start y warm up de la JVM para arquitecturas Serverless.

Si se miran las últimas versiones de Java y su roadmap, básicamente, sólo agregan features que Scala tiene desde hace años. Entonces, si quiere ir por funcional en la JVM ¿por qué no pasar de una vez a Scala?

La adopción de un lenguaje de programación (cualquiera sea) va más allá de las capacidades que el lenguaje ofrezca. Esto puede sonar contra intuitivo pero si se consideran variables adicionales como curvas de aprendizaje, tamaño del equipo de ingeniería (desarrollo de software), soporte de un vendedor o fabricante y licenciamiento se puede comprender que la elección va más allá de las capacidades del lenguaje. Sobre la JVM existen aproximaciones 100% funcionales (no híbridas como Scala) como es el caso de Eta (https://eta-lang.org/) que podrían, de entrada, sonar como una alternativa más fuerte como respuesta a la pregunta. Así entonces, considerando como premisa que está trabajando en Java, considerar una aproximación a la programación funcional en ese lenguaje no resulta contra intuitivo y por el contrario se determina necesario. En los casos en los que las variables mencionadas (y algunas otras) lo permitan, se deberá hacer el análisis correspondiente para determinar la mejor herramienta (lenguaje de programación) que solucione un problema.

Sabemos que hay patrones de diseños para POO, ¿existen algunos para la funcional?

En OO, los llamados patrones son recetas prescriptivas que son sujetas de ser mezcladas y modificadas al antojo y necesidad del desarrollador. Se dice que deben ser descubiertos y posteriormente documentados para su posterior mezcla y redescubrimiento. Esto es cierto pero deja espacio para la exageración y la improvisación, por no decir la imprecisión y posibilidad de inyección de complejidades innecesarias. En el caso de FP los patrones sí que existen y con una diferencia sutil: deben tener reglas algebraicas que los definan a tal punto que si las reglas algebraicas no se cumplen, simplemente no se sigue el patrón. Ejemplos de estos patrones los encuentras en Functor, Semigroup, Monoid, Applicative, Monad, Adjunctions (para el caso de comportamientos) y Lenses (para el caso de datos). Estos patrones son ampliamente utilizados en lenguajes funcionales *puros*, mientras que en lenguajes como Java que es OO se pueden encontrar implementaciones de algunos de estos patrones (o instancias de ellos) en librerías como vavr (e.g. el tipo de dato Either<L,R> o Validation<E,T>).

¿Es posible tener el código fuente de los dos casos de estudio? Solo para fines académicos y didácticos para estudiar con más detalle lo que Laura nos está presentando

https://github.com/lcamach/programacion-funcional-usando-java

Finalmente, te invitamos a seguirnos en nuestras redes sociales:

Linkedin

Twitter

YouTube

Sitio web

Y si deseas ser parte de S4N, te invitamos a visitar nuestra página de carreras

En S4N te retas, te inspiras, te transformas ¡Evolucionas!

BOGOTÁ - COLOMBIA
Cra. 18 # 84 - 84 - Bldg. Calle 85
Floor 5 - Tel: (+571) 749 8821 - 703 5882
http://s4n.co

SEATTLE - USA
107 Spring St. - Seattle, WA 98104
Tel: (650) 695 - 1 298

MEDELLÍN - COLOMBIA
Cra. 29c # 10C - 125 - Select Business Tower
Office 802 - Tel: (+574) 3668113