codely

Finder kata en Scala - Dejando los bucles atrás ƛ

19 DE ABRIL DE 2017 | SCREENCASTS

Ya vimos cómo enfocar esta misma kata de refactorización en PHP. Básicamente a través de pequeños procesos de refactorización, fuimos puliendo el código para aportarle semántica del dominio y mayor legibilidad. Hoy os traemos otro enfoque de solución partiendo del repositorio con el código inicial que preparamos para el dojo de la Software Craftsmanship Barcelona. **Os animamos a proponer vuestra solución vía Pull Request**, ¡Seguro que salen alternativas interesantes y aprendemos más!

Estado inicial

En este enfoque veremos cómo partimos del estado inicial del código (coged aire antes de seguir con el scroll 😅): https://gist.github.com/JavierCane/b1933d88c98aa8e4ee70ebfaecf3b2c0

Refactor semántico

Primeramente pasamos a una Iteración de Chapa y Pintura™. **Nos centramos en los nombres de clases y variables, y mover algunos métodos** para fomentar la cohesión, y quitar responsabilidades al método principal. **Introduciremos así conceptos de dominio** que hasta ahora no se veían reflejados en el código. De esta forma conseguimos sentirnos más cómodos a la hora de trabajar con el código ya que vamos conociendo lo que hace. **Esto es algo perfectamente extrapolable a nuestro día a día**: Primero un poco de **semántica para tantear el terreno e ir cogiendo contexto**, y luego añadir la funcionalidad/bugfix que nos ha motivado a abrir esa clase 👌 (refactoring oportunista, o regla del Boy Scout). Aquí tenéis la rama de esta primera iteración por si queréis hacer una PR con vuestra solución partiendo de ella 🙂 https://gist.github.com/JavierCane/224879683189641b31a6f08b6c6fc749

Scala idiomático

Finalmente lo que hemos hecho ha sido aplicar elementos más idiomáticos de Scala:

- **Reemplazamos las colecciones de Java por las de Scala** (¡Ya no hay ni un import! :P). Con esto ganamos métodos que nos vendrán de lujo, como por ejemplo SeqLike#combinations.
- **Type Class Ordering**: Haciendo uso de ella nos podemos llevar la gestión de criterios para definir la mejor pareja de personas a objetos aislados y promover el Open/Closed Principle de SOLID.
- **Option[PeoplePair] como tipo de retorno**: De esta forma hacemos explícita la posibilidad de no encontrar una pareja de personas bajo el criterio de búsqueda recibido (None). Esto es una de las principales ganancias con respecto la iteración anterior. Las alternativas a esto serían: devolver null, lanzar excepción, o como se venía haciendo, devolver una instancia de PeoplePair inconsistente (con los atributos de clase sin inicializar).
- **Eliminamos la mutabilidad** gracias a las colecciones de Scala (inmutables por defecto), y los métodos Iterator#map y GenTraversableOnce#reduce.

https://gist.github.com/JavierCane/1b70f6572277e5795fe0badee8759d64 Si te ha gustado esta kata y ves margen de mejora, agradeceríamos infinito que lo compartieras a través de un comentario en el vídeo, Twitter, o incluso si te animas, una PR al repo

🚀🦄 Próximamente seguiremos metiéndole caña al tema de la programación funcional. En breves publicaremos la entrevista a Juan Manuel Serrano donde hablamos de esto, higher kinded types, y mucho más 😬.

TAGS

Clean Code DDD Nivel Medio

Refactoring Scala

Software Craftsmanship SOLID

ANTERIOR
Entrevista Juan Manuel Serrano 👨‍🏫 - Universidad, Programación funcional, y Type Classes

SIGUIENTE
Entrevista a Ricard Clau: Director técnico HolaLuz 💡