devonfw

Coding Dojo – Sesion 1

Septiembre 2017

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

Introducción

Bienvenidos al Katayuno

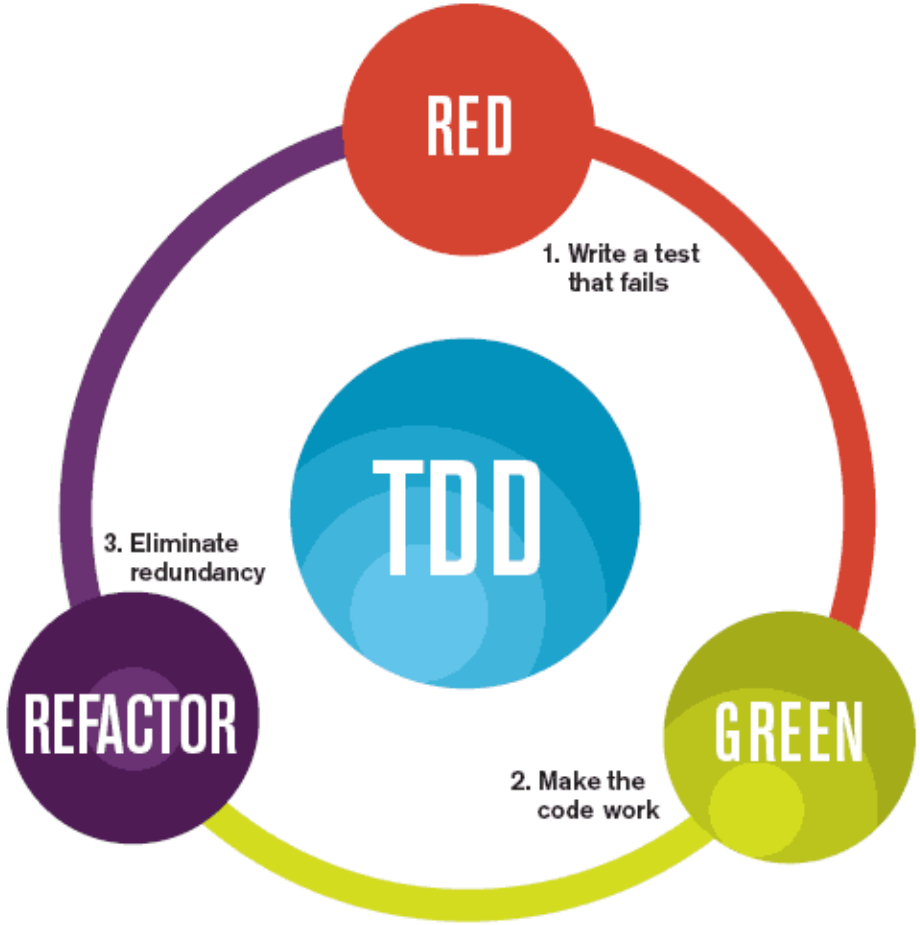
Agenda

15 min: Introducción ‘Coding Dojo’ y explicación de la Kata que vamos a hacer

60 min: Desarrollo de la Kata

15 min: Resumen general, sugerencias y preguntas

CODING
DOJO


RED
1. Write a test that fails
TDD
GREEN
2. Make the code work
REFACTOR
3. Eliminate redundancy

“Cambios en el código para hacerlo más fácil de entender y más barato de modificar, sin alterar su comportamiento observable”

- Martin Fowler -

COMMENTS

MAGIC NUMBER

LONG METHOD

DUPLICATE METHOD

LARGE CLASS

LONG PARAMETER LIST

https://refactoring.guru/refactoring/smells

EXTRACT CLASS

INTRODUCE EXPLAINING VARIABLE

EXTRACT METHOD

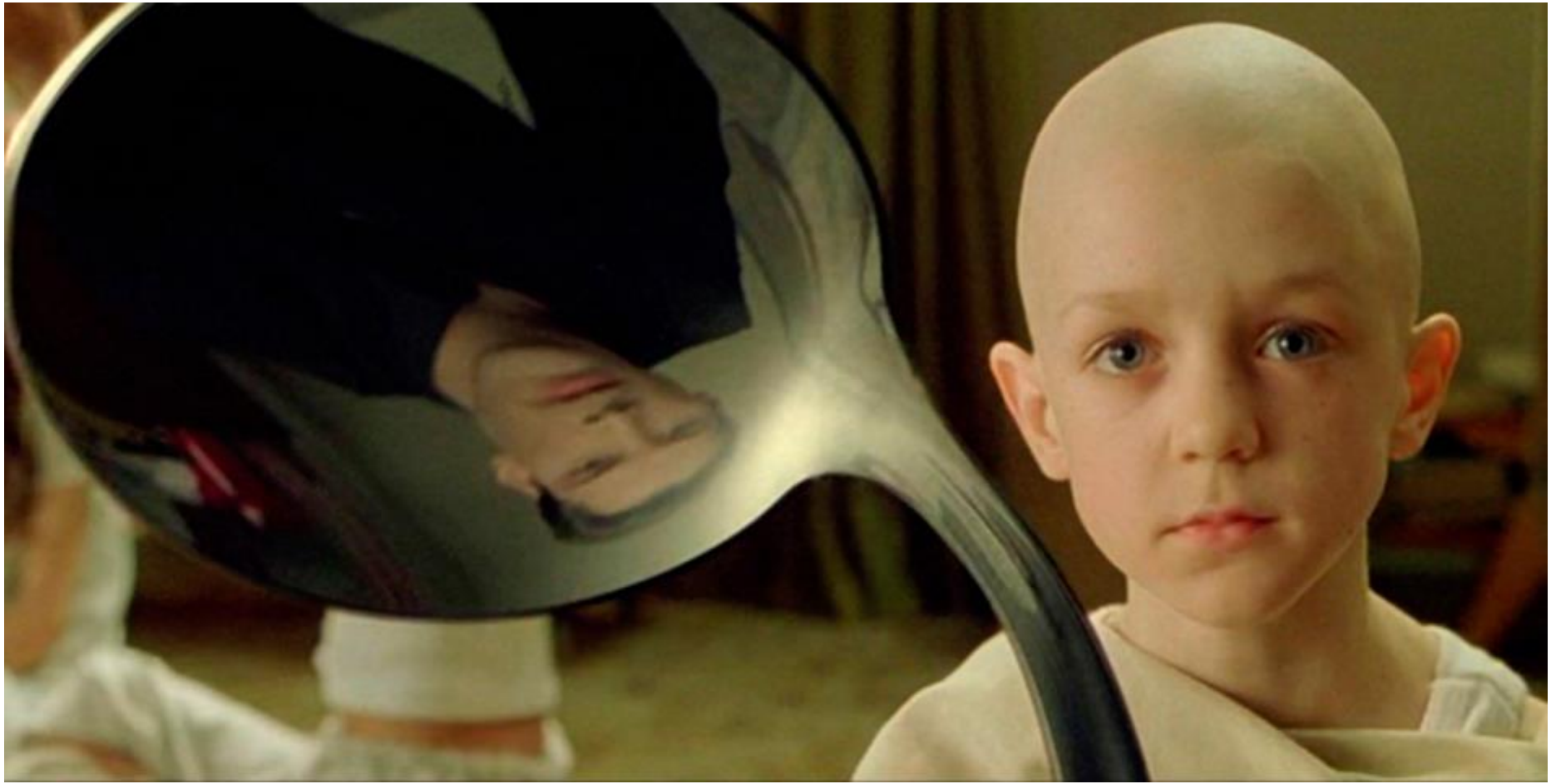
REMOVE ASSIGMENTS TO PARAMETER

SELF ENCAPSULATE FIELDS

INLINE METHODS

ENCAPSULATE COLLECTIONS

https://refactoring.guru/refactoring/smells



SWITCHS

PRIMITIVE OBSESSION

MESSAGE CHAINS

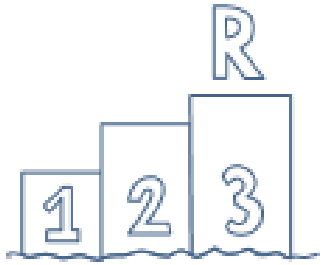
SPECULATIVE GENERALIZATION

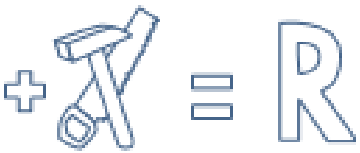
DATA CLUMPS

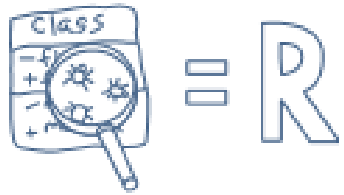
FEATURE ENVY

https://refactoring.guru/refactoring/smells

¿Cuando refactorizamos?

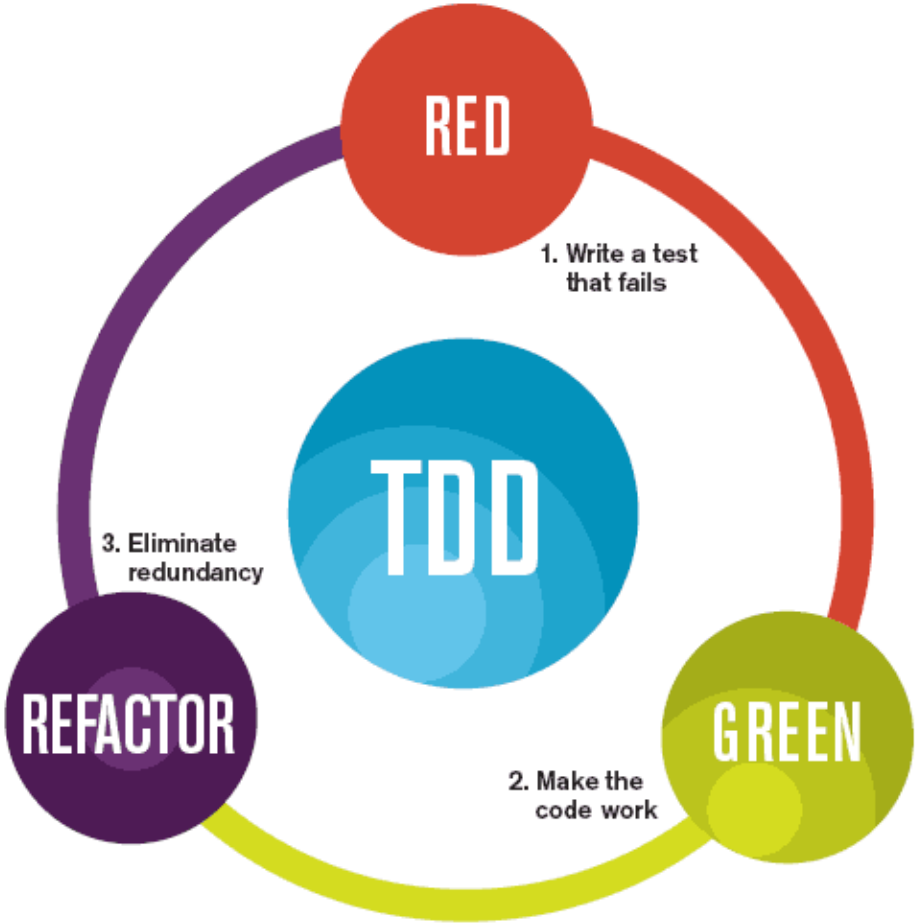






https://refactoring.guru/refactoring/smells

CODING DOJO


RED
1. Write a test that fails
TDD
GREEN
2. Make the code work
REFACTOR
3. Eliminate redundancy

Práctica

Kata String Calculator

String Calculator (Consejos)

Antes de comenzar

Intentaremos no leer el enunciado completo.

Iremos leyendo poco a poco mientras lo resolvemos.

Haremos sólo una tarea a la vez.

El truco está en aprender a trabajar de forma incremental.

Nos aseguraremos de testear únicamente las entradas correctas.

No es necesario testear las entradas incorrectas para esta kata.

Llegaremos hasta donde nos de tiempo.

No se trata de terminar la Kata entera sino en aprender durante el proceso.

Aprendizajes que intentaremos conseguir con esta Kata:

Adaptabilidad ante los cambios.
Uso correcto del TDD.
Aprender a hacer refactoring.
Aprender a tener un código menos sucio.

devonfw

String Calculator (Enunciado)

1. Crea una String Calculator con el método: int add(String input)
   - El parámetro del método puede contener 0, 1 o 2 números y devolverá su suma (para un string vacío devolverá 0). Por ejemplo: "" o "1" o "1,2”
   - Compieza por un test simple para un string vacío y luego para 1 y 2 números.
   - Recuerda resolver el problema de la manera más simple posible para que te fuerce a escribir las pruebas que aún no se te habían ocurrido.
   - Recuerda refactorizar después de conseguir pasar cada test.
2. Permite al metodo "Add" manejar cualquier cantidad de números.
3. Permite al método "add" manejar saltos de línea entre números en lugar de usar comas.
   - La siguiente entrada es correcta: "1\n2,3" (el resultado será 6)
   - La siguiente entrada NO es correcta: "1,\n" (no hace falta que la pruebes, es simplemente para clarificar)
4. Soporta diferentes delimitadores
   - Para cambiar un delimitador, el comienzo del string debe contener una línea separada que sea como esta: "//[delimitador]\n[números...]". Por ejemplo: "//;\n1;2" debe dar como resultado 3 donde el delimitador por defecto es ";".
   - La primera línea es opcional. Todos los escenarios existentes hasta ahora, deben estar soportados.

String Calculator (Enunciado)

5. Llamar al método "Add" con números negativos deberá lanzar una excepción con el texto "negativos no soportados" y el número negativo que ha sido pasado. Si hay múltiples números negativos, muestra todos ellos en el mensaje de la excepción.
6. Los numeros mayores de 1000 deben ser ignorados.
   - Por ejemplo "2,1001" dará como resultado 2.
7. Los delimitadores pueden ser de cualquier longitud con el siguiente formato: "//[delimiter]\n".
   - Por ejemplo: "//[;;;]\n1;;;2;;;3" debe dar como resultado 6.
8. Permite múltiples delimitadores de la siguiente manera: "//[delim1][delim2]\n".
   - Por ejemplo: "//[#][%]\n1#2%3" debe dar como resultado 6.
9. Asegúrate de que puedes manejar delimitadores de cualquier longitud mayor de un caracter.