Esta kata ofrece más práctica en el diseño de software característica por característica - una parte importante de los procesos de desarrollo Agile - utilizando TDD. Cada vez que añadas una nueva característica, tendrás que adaptar tu algoritmo mediante refactorización para que soporte el nuevo comportamiento.

Las reglas de esta kata se hacen cada vez más complejas. Es recomendable abordarla añadiendo una regla a la vez. Intenta no mirar hacia delante: imagina que las reglas hubieran llegado en sesiones informativas separadas repartidas a lo largo de un proyecto de seis meses.

Instrucciones

Paso 1: Calculadora sencilla

Crea una String calculator sencilla con un solo método:

```
class StringCalculator {
    int Add(string numbers);
}
```

El método puede tomar 1 o 2 números separados por comas, y devolverá su suma.

El método devuelve 0 cuando se le pasa la cadena vacía. Ejemplo:

```
Add("") // 0
Add("4") // 4
Add("1,2") // 3
```

Paso 2: Tamaño arbitrario de los números

Permite que el método Add maneje una cantidad desconocida de números.

Ejemplo:

```
Add("1,2,3,4,5,6,7,8,9") // 45
```

Paso 3: Separador de líneas nuevas

Permite que el método Add reconozca como separadores tanto las nuevas líneas como las comas. Los dos tipos de separadores pueden utilizarse indistintamente.

NB: Céntrate en el camino feliz - ya que esto no es código de producción, no pasa nada si el código se bloquea si se le da un input no válido (e.g. `"1,\n2"`).

Ejemplo:

```
Add("1\n2,3") // 6
```

Paso 4: Separadores personalizados

Admite de forma opcional separadores personalizados. Para cambiar el separador, el principio de la cadena contendrá una línea separada con el siguiente aspecto: “`//<separator>\n<numbers>`”

Ejemplo:

```
Add("//;\n1;2") // 3
```

Paso 5: Rechazar los negativos

Llamar a Add con un número negativo lanzará una excepción `negatives not allowed`, y el negativo que se ha pasado.

Si hay varios negativos, muéstralos todos en el mensaje de excepción.

Ejemplo:

```
Add("1,-2,-3") // error: negatives not allowed: -2 -3
```

Paso 6: Ignorar los números superiores a 1000

Los números superiores a 1000 deben ignorarse.

Ejemplo:

```
Add("1001, 2") // 2
```

Paso 7: Separadores de longitud arbitraria

Los separadores pueden tener cualquier longitud si van rodeados de corchetes.

Ejemplo:

```
Add("//[***]\n1***2***3") // 6
```

Paso 8: Separadores múltiples de un solo carácter

Permite múltiples separadores de un solo carácter como estos: “`//[delim1][delim2]\n`”

Paso 9: Separadores múltiples de mayor longitud

Gestiona separadores múltiples con cualquier longitud de carácter.

Ejemplo:

```
Add("//[foo][bar]\n1foo2bar3") // 6
```

Crédito: Roy Osherove

Consejos

Si añades un nuevo test y luego te das cuenta de que vas a necesitar un gran cambio en el código para que pase, puedes retirar el test (borrándolo o marcándolo como ignorado) y refactorizar el código hasta que esté listo para el nuevo test y la nueva funcionalidad.

De hecho, es una forma muy eficaz de trabajar: sólo hay que tener cuidado de no añadir nuevas funcionalidades involuntariamente mientras se refactoriza.

Enlaces útiles

Soluciones

- Soluciones parciales en muchos lenguajes funcionales (en inglés)