Orientación a Objetos 2 – Práctica 6

Ejercicio 1: Sensores de temperatura

Un automóvil tiene un dispositivo de sensado que mide la temperatura en varios lugares del motor. El sistema de software que controla al motor implementa el registro de esas medidas como una instancia de la clase `RegistroDeTemperaturas`, que entiende los siguientes mensajes:

```
#temperaturaDelCarter
#temperaturaDelCombustible
#temperaturaDeEscape
#reset
```

Los valores de temperatura retornados por los primeros tres mensajes están expresados en *centésimas de grados Fahrenheit* (es decir, un valor de retorno de 3200 indica una temperatura de 32,0F).

El mensaje `#reset` *re-inicializa el registro* y se usa en casos en los que se desconfía del funcionamiento correcto del registro (algo así como un *reboot*).

Se desea implementar una serie de extensiones al registro con los siguientes **requerimientos**:

- No se puede modificar la implementación de la clase `RegistroDeTemperatura`
- Cada una de las extensiones es independiente de las demás y debe ser posible activarla o desactivarla en tiempo de ejecución.
- Un objeto que interactuaba con un `RegistroDeTemperatura`, debe poder interactuar sin necesidad de cambios en su implementación con un registro al que se han aplicado las extensiones.

A continuación se detallan las extensiones que se desean implementar:

1. Traducir los valores retornados por el registro: se desea retornar los valores en grados Celsius (La conversion es `C=(F-32)/1.8`)
2. Corregir el rango: se desea que antes de retornar el valor sensado, el sistema lo multiplique por un valor provisto en el momento de la configuración. Por ejemplo, multiplicar por 0,01 para permitir la obtención de valores en grados Farenheit en lugar de centésimas.
3. Agregar mensajes que permitan obtener el valore mínimo y el máximo para cada temperatura, computando desde el último `#reset`.

Tarea:

1. Realice el diseño de la aplicación con las extensiones solicitadas y represéntelo con un diagrama UML de clases.
2. Realice los diagramas de secuencia UML que muestren la colaboración del `RegistroDeTemperatura` con las extensiones para el siguiente caso: se desea que los valores de retorno se expresen grados Celsius y con un rango corregido usando centésimas.
3. Implemente en forma completa en Smalltalk.
4. Implemente y ejecute test cases para probar cada extensión en forma aislada.

Ejercicio 2: Refactoring

Indique si cada una de las siguientes aseveraciones es verdadera o falsa. Explique.

1. Cuando un código es refactorizado cambia su comportamiento agregando más funcionalidad.
2. Si el código está bien refactorizado no es necesario testearlo.
3. Después de ser refactorizado, la estructura interna del código permanece igual que antes.
4. La refactorización del código se hace en un sólo paso en el que se unen todos los cambios

Ejercicio 3: Software con olores

Responda:

1. Qué significa la expresión “código con mal olor” según lo visto en la teoría?.
2. Encuentre tres casos de los “malos olores” explicados en la teoría en su propio código de los prácticos anteriores.
3. Indique tres razones por las cuales es importante hacer refactoring

Ejercicio 4: TFD

Tome como base el esquema de la Figura 1 e implemente un proceso TFD para modificar el modelo del Contact Manager que elaboró en la práctica 3.

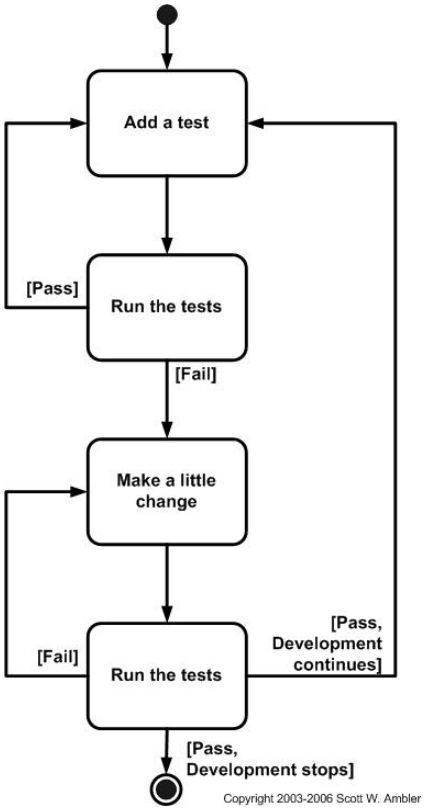

Figura 1: Pasos en un proceso TDD

Se desea que el nuevo modelo (además de los requerimientos originales) sea capaz de almacenar para cada contacto su fecha de nacimiento y el nombre de un archivo con su foto.

Para cada paso del diagrama indique las tareas que tiene que realizar y las clases y/o métodos involucrados.

Implemente todo el proceso.

Ejercicio 5: Testing y refactoring

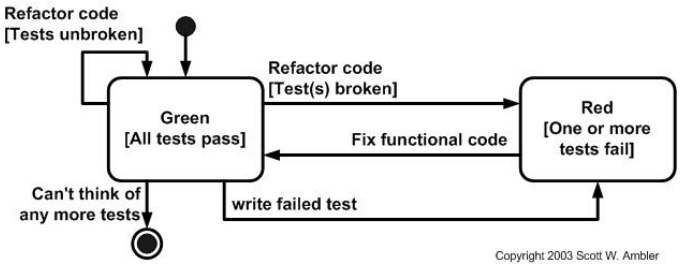

Figura 2: Estados en TTD y Refactoring

Tomando como base el diagrama de Ambler de la Figura 2 sobre el proceso de testing y refactorización y el ejercicio del Contact Manager que realizó en prácticas anteriores indique:

1. clases afectadas en la transición *Write failed test*
2. clases afectadas en la transición *Fix functional code*