# Orientación a Objetos 2 – Práctica 2

Rev: 2020 Diego Torres
Rev: 2019 Sergio Firmenich, Leandro Antonelli
Rev: 2018 Sergio Firmenich, Leandro Antonelli
Rev: 2017 Diego Torres, Vanesa Mola, Juan Ignacio Vidal

## Ejercicio 1: SpOOtify

SpOOtify es, obviamente, la versión orientada a objetos de Spotify. En Spotify cada usuario puede buscar obras musicales de diferentes maneras: por autor, por género, por album. Los usuarios pueden agregar las canciones a My Music. Obviamente también puede remover temas de My Music. Cada autor conoce sus álbumes, cada álbum conoce sus temas. SpOOtify conoce a los autores y a partir de allí, a todos los temas. SpOOtify conoce a los usuarios.
Los mensajes que los usuarios pueden enviar a SpOOtify son los siguientes:

```
SpOOtify>>buscarPorTitulo: aString
“Retorna una colección de temas cuyo titulo contiene o es igual a aString. La búsqueda es case insensitive"

SpOOtify>>buscarPorAutor: aString
“Retorna una colección de temas cuyo nombre de autor contiene o es igual a aString. La búsqueda es case insensitive"

SpOOtify>>buscarPorAlbum: aString
“Retorna una colección de temas cuyo titulo de álbum contiene o es igual a aString. La búsqueda es case insensitive"

User>>agregar:aSong
"El usuario agrega un tema a su colección de música favorita"

User>>remover:aSong
"El usuario elimina un tema de su colección de música favorita"
```

### Tareas:

1. Realice el diseño usando un diagrama de clases UML. Verifique el diseño con un ayudante.
2. Planifique qué tipo de pruebas desea hacer para los mensajes propuestos en el enunciado.
3. Implemente los casos de prueba.

4. Implemente en Pharo el modelo de SpOOtify.
5. Ejecute los test cases..
6. Considerando las formas de verificar la funcionalidad que conoce: Test Cases y workspaces, ¿qué diferencias puede observar entre éstas?

# Ejercicio 2: EvernOOte

EvernOOte es un organizador de notas. El usuario tiene notebooks, cada notebook contiene muchas notas. Cada nota tiene un titulo, una fecha de creación y tags.

El título es un string, los tags son una colección de strings, el contenido es un string (potencialmente largo, pero no importa).

Ud. debe modelar la aplicación EvernOOte e implementarla.

Ud. es en el encargado de definir el protocolo de los objetos. Es importante que EvernOOte provea la siguiente funcionalidad:

1. Crear y borrar notas.
2. Crear y borrar notebooks.
3. Buscar notas por título, fecha o notebook.
4. Conocer el tamaño de un notebook, que es igual a la sumatoria de los tamaños de las notas. El tamaño de una nota es la suma de texto, los tags, y el título de la misma, más el tamaño de la fecha que siempre es 10 (bytes).

## Tareas:

1. Realice el diseño usando un diagrama de clases UML. Verifique el diseño con un ayudante.
2. Planifique qué tipo de pruebas desea hacer para el protocolo propuesto por Ud.
3. Implemente en Pharo los casos de prueba y el modelo de EvernOOte.
4. Ejecute los Test Cases.

# Ejercicio 3: 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 solo paso en el que se unen todos los cambios.

# Ejercicio 4: 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 (implementacion de trabajos prácticos).
3. Basados en la bibliografía indicada en las clases teóricas de la materia, indique tres razones por las cuales es importante hacer refactoring.

# Ejercicio 5: Algo huele mal

Indique en los siguientes ejemplos la presencia de malos olores.

1. La clase `Cliente` tiene el siguiente protocolo público. ¿Cómo puede mejorarlo?

- **`#lmtCrdt`** devuelve el límite de crédito del cliente.
- **`#mtFcE: unaFecha y: otraFecha`** devuelve el monto facturado al cliente desde `unaFecha` hasta `otraFecha`
- **`#mtCbE: unaFecha y: aux`** devuelve el monto cobrado al cliente desde `unaFecha` hasta `otraFecha`

2. Al revisar el siguiente diseño inicial (Figura 2), se decidió realizar un cambio para evitar lo que se consideraba un mal olor. El diseño modificado se muestra en la Figura 3. Indique qué tipo de cambio se realizó y si lo considera apropiado. Justifique su respuesta.

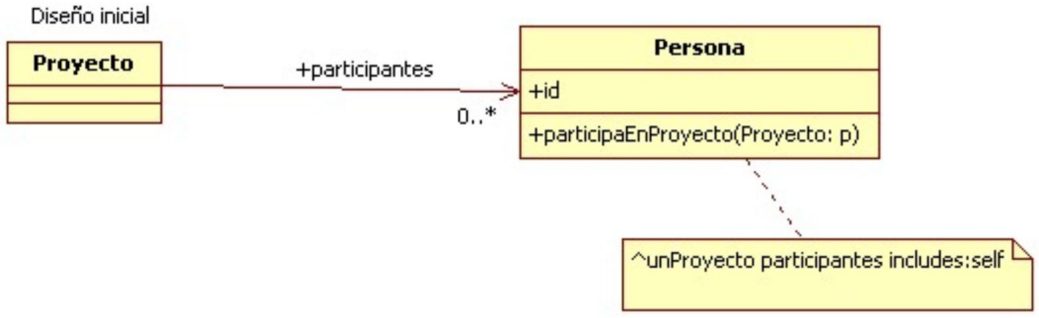


Figura 2: DIagrama de clases del diseño inicial.

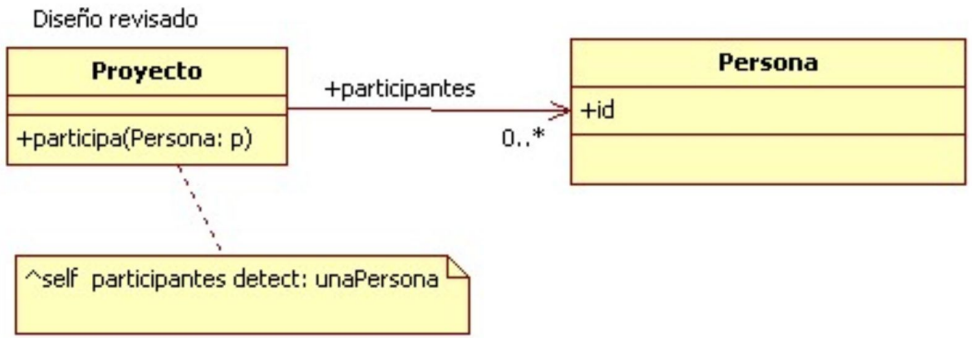


Figura 3: Diagrama de clases modificado.

3. Analice el código que se muestra a continuación. Indique qué defectos encuentra y cómo pueden corregirse.

```
imprimirValores
  |totalEdades promedioEdades totalSalarios|
  totalEdades := 0.
  totalSalarios := 0.
  personal do: [ :empleado |
                 totalEdades := totalEdades + empleado edad.
                 totalSalarios := totalSalarios + empleado salario
               ].
  promedioEdades := totalEdades  / personal size.
  Transcript show: 'Edad promedio: ' ,
                   promedioEdades printString ,
                   ' - Total de salarios: ' ,
                   totalSalarios printString.
```

# Ejercicio 6: TweetBase

Considere la clase TweetBase que contiene una lista de tweets, y los siguientes métodos:

```
TweetBase>>tweetCountByUser
      "returns a dictionary with the number of tweets per user in this tweet base"
      | result |
      result := Dictionary new.
      tweets do: [ :tweet |
                  | user|
                   user:= tweet user.
                   result at: user ifAbsentPut: 0.
                   result at: user put: (result at: user) + 1 ].
      ^ result
```

**TweetBase>>tweetCountByUserGender**

```
"Count the occurrences of the gender of the users"
| result |
result := Dictionary new.
tweets do: [ :tweet |
            | g |
            g := tweet user gender.
            result at: g ifAbsentPut: 0.
            result at: g put: (result at: g) + 1 ].
^ result
```

## Tareas:

1. Indique qué problemas (malos olores) encuentra (puede usar sus propias palabras).
2. Escriba un Test Case antes de refactorizar para #tweetCountByUser
3. Refactorice ambos métodos.