Taller Refactoring

# Objetivos:

* Buscar malos olores en el código.
* Aplicar técnicas de refactorización dado un código con malos olores.

En el código adjunto se encuentran varias clases en el paquete conmalolor, en donde debe identificar cuál de esas clases necesita aplicar una de las siguientes técnicas de refactorización:

* Extract Class:
  1. Crear nueva clase que tendrá la información relevante.
  2. Crear relación entre la vieja clase y la nueva. Por ejemplo, agregar atributo de nueva clase en la vieja clase.
  3. Aplicar refactorización ***Move***: **Clic derecho 🡪 Refactor 🡪 Move** o **Ctrl + M** para cada campo y método deseado.
  4. Renombrar campos y métodos para que sean más naturales (generales).
  5. Decidir cuál será la visibilidad del nuevo atributo en la clase vieja. Getter, setters, public, private, etc.
* Introduce Local Extension:
  1. Seleccionar con el cursor el tipo de dato que desea extender.
  2. Aplicar refactorización ***Local Extension***: **Clic derecho 🡪 Refactor 🡪 Introduce 🡪 Local Extension …** o **Ctrl + Shift + X** y seleccionar **SubType** e ingresar el nombre de la nueva clase que desea crear.
  3. Mover el o los métodos que deben pertenecer a esa nueva clase. También se puede aplicar refactorización ***Move***.
  4. Remplazar la vieja clase con la nueva en la clase cliente.
* Extract Superclass:
  1. Seleccionar un campo o método de las clases que desea generalizar (crear la super clase).
  2. Aplicar refactorización ***Extact Superclass***: **Clic derecho 🡪 Refactor 🡪 Extract Superclass …** o **Ctrl + Alt + Shift + S** .
  3. Seleccionar los atributos y métodos que se desea enviar a la clase padre, indicar cuales son abstractos e indicar el nombre de la clase padre.
  4. Extender la clase padre desde las otras clases.
  5. Eliminar los atributos y métodos que ya están en la clase padre.
  6. Renombrar los métodos que corresponden a los de la clase padre pero tienen distinto nombre. También se puede aplicar refactorización ***Rename*** de los métodos para asegurarnos que se cambie la llamada en todos los lugares donde está siendo utilizada.
  7. Verificar si hay algún atributo o método que necesite ser enviado a la clase padre. En caso de ser necesario, aplicar refactorización ***Pull Up***, indicando si es abstracto o no.
  8. Verificar si hay algún método que deba cambiar el algoritmo de implementación, especialmente los que ahora están declarados en la clase padre. Se puede aplicar la técnica de ***Substitute Algorithm*** (manualmente).
  9. Finalmente, recuerde que puede utilizar patrones de diseño como Composite.

Ref:

<http://www.pellissier.co.za/hermien/?p=794>

<http://www.codejava.net/ides/netbeans/10-netbeans-shortcut-keys-for-code-refactoring>

<http://www.informit.com/articles/article.aspx?p=519945&seqNum=27>

**A usted se le ha pedido lo siguiente:**

* Identificar y escribir en este archivo, los malos olores del código de cada clase del paquete ***conmalolor***.
* Aplicar el método de refactorización deseado utilizando la herramienta de refactor que está integrada en el IDE. Sugerencia: Cree copias de las clases en paquetes por separado para cada refactorización y luego siga los pasos indicados para cada tipo.

**Extract Method**

En la clase **Empresa** existen problemas de duplicación de código, ya que hay muchos métodos parecidos; lo cual puede ser solucionado usando el método de Refactoring Extract Method. En esta misma clase se puede separar el método que nos permite validar la información para que pueda ser llamado por otros métodos. Uno de los beneficios de este método es que ya evitamos la duplicación del código y en este caso el nuevo método puede ser llamado en varias partes del código.

**Duplicated Code:** La clase empleado tiene atributos idénticos a los de la clase cliente, se podría usar herencia para evitar la repetición del código.

**Primitive Obsession:** Entre algunos de los atributos de las clases empleado y cliente existen datos que se podrían separar en una sola clase donde se podrían manejar de manera más eficiente.

**Switch Statements**: Dentro de la clase empleado se encuentran métodos que usan sentencias switch si el sistema se extendiera generaría la necesidad de modificar la clase al aumentar un nuevo case.

**Data Class:** Presente en la clase EmployeeType debido a que es una clase que solo maneja datos más no realiza operaciones con ellos.

**Shotgun Surgery:** Está presente en la clase Employee ya que si por ejemplo se ingresa un rol de trabajo, se deberia cambiar los métodos en los que estos se realizan como cs().