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:**

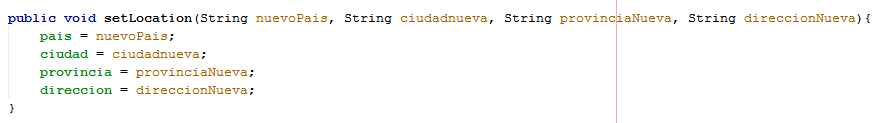
* Identificar y escribir en este archivo, los malos olores del código de cada clase del paquete ***conmalolor***.
* Aplicar el método de refactorización deseado utilizando la herramienta de refactor que está integrada en el IDE. Sugerencia: Cree copias de las clases en paquetes por separado para cada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

**Malos Olores**

**Clase Cliente**

**Mal Olor: Long Parameter List**

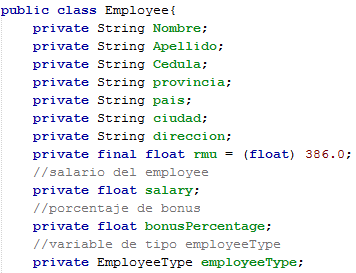
Este mal olor está presente debido a que el método recibe 4 parámetros, lo cual no se considera recomendable de hacer, este mal olor se refactoriza aplicando la **técnica Introduce Parameter Object,** es decir pasar como parámetro un objeto de tipo **Location.**



**Clase Employee**

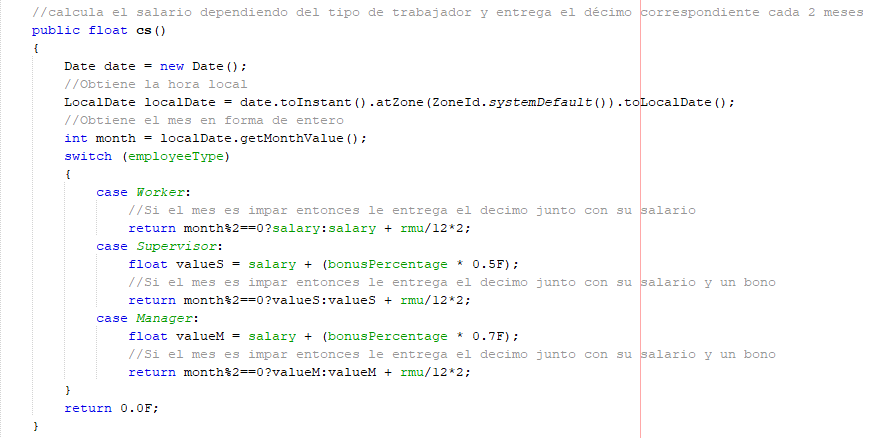
**Mal Olor: Primitive obsession**

En el caso de las variables *provincia*, *pais*, *ciudad* y *direccion,* el tipo de dato que poseen es String lo cual se considera como primitive obsession al solamente usar variables primitivas en lugar de crear un objeto, por ejemplo, una nueva clase **Location**, que reemplace a los mismos*.* Se refactoriza aplicando la **técnica Replace Type Code with Class.**



**Mal Olor: Comments**

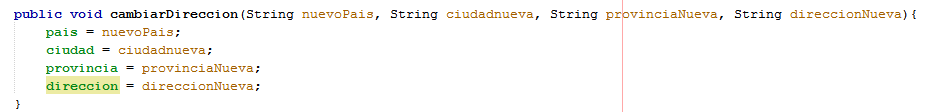
En el método cs(), para calcular el salario se observa que existen comentarios para cada uno de los casos del switch, ya que no es un código comprensible a simple vista, esto se refactoriza aplicando la **técnica Extract Variable** para poder entender que se está retornando en cierta línea de código, adicionalmente dado el nombre del método que no es muy explícito y requiere de un comentario para saber que realiza el mismo es necesario refactorizar aplicando la **técnica Rename Method**.



Adicionalmente, es importante considerar que este método se puede refactorizar aplicando la técnica de reemplazar switch con polimorfismo.

**Mal Olor: Duplicate Code**

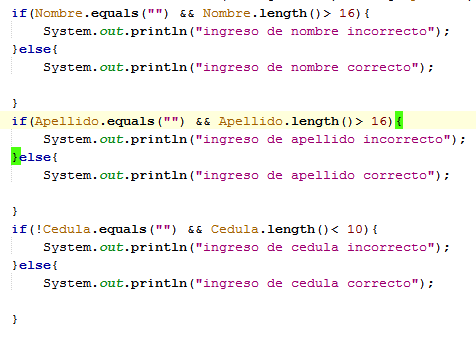
Puesto que el código del método cambiarDireccion es idéntico al de la clase **Cliente** y su método setLocation, aquí se refactoriza aplicando la **Extract SuperClass**.



**Clase Empresa**

**Mal Olor: Duplicate Code**

Tanto los métodos validarInformacion, como guardarCliente tienen en común algunos puntos ya que dada una condición en las sentencias condicionales (la misma que se repite en ambos métodos) se imprime si un ingreso es correcto o no. El método validarInformacion se podría llamar dentro de la función para guardar el cliente. Se refactoriza aplicando la **técnica Extract Method,** también se puede aplicar **Extract Variable** para hacer más entendible el código.



En este caso, se observa que el método *MostrarInformacion* se repite y realiza lo mismo, se refactoriza **parametrizando el método,** haciendo quereciba como parámetro un objeto de tipo **Persona** que se obtuvo alaplicar la **técnica Extract SuperClass** entre cliente y employee.

