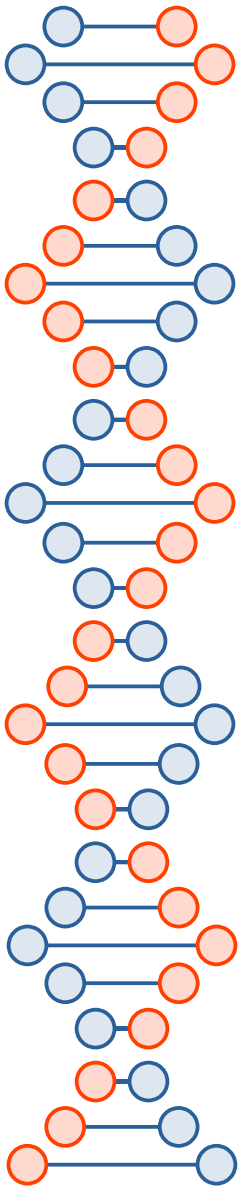
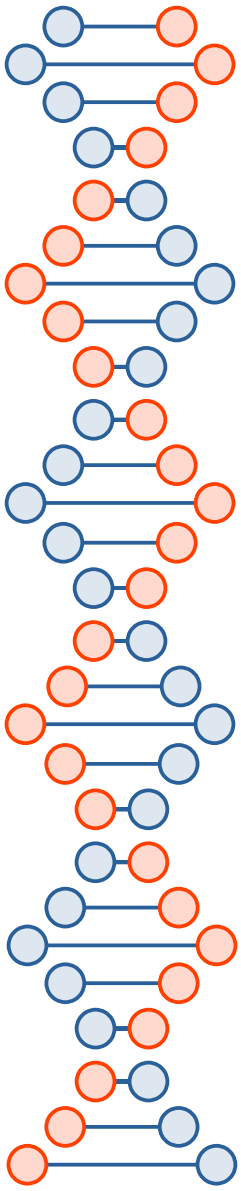


Refactorización



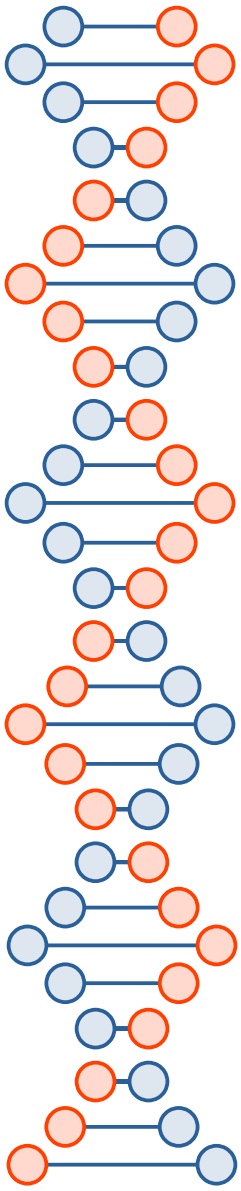
Refactorización

Definición de refactorización: “La refactorización consiste en realizar una transformación al software preservando su comportamiento, modificando su estructura interna para mejorarlo” (Opdyke, 1992).

La refactorización (informalmente llamada limpieza de código) ayuda a tener un código que es más sencillo de comprender, más compacto, más limpio y, por supuesto, más fácil de modificar.

Cambios realizados en el software para hacerlo más entendible y modificable.

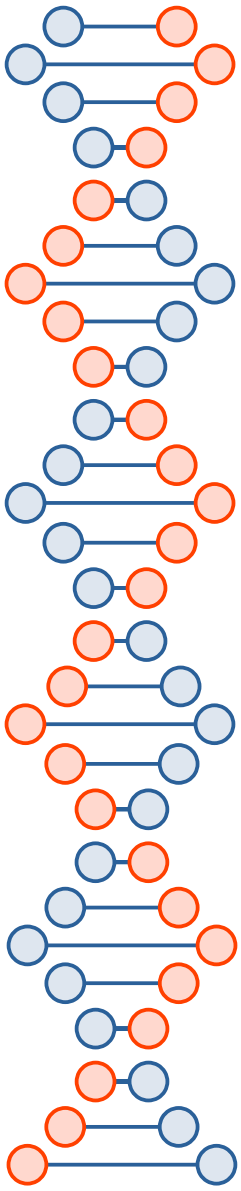
“No cambia la funcionalidad del código ni el comportamiento del programa, el programa deberá comportarse de la misma forma antes y después de efectuar las refactorizaciones”.



Refactorización

Al refactorizar no se está introduciendo nada nuevo, solo está extrayendo operaciones en métodos, moviendo código de un sitio a otro, dejando lo métodos más pequeños y con una nomenclatura más comprensible;

Una vez termina, el código funciona exactamente igual que antes, pero está hecho de otra manera que le facilita el trabajo o que evita caer en malas prácticas de código, como la redundancia y la duplicación de código.



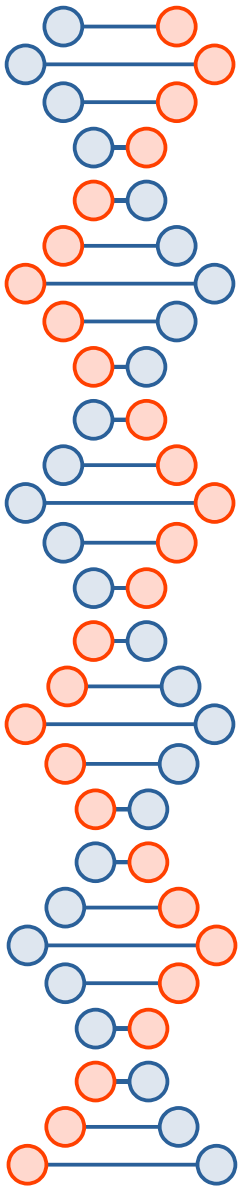
Refactorización

- ¿Cuando refactorizar?

  La refactorización se debe ir haciendo mientras se va realizando el desarrollo de la aplicación.

  También forma parte de la fase de mantenimiento.

- Analizamos los síntomas que indican la necesidad de refactorizar, los llamados bad smells (malos olores)



Refactorización

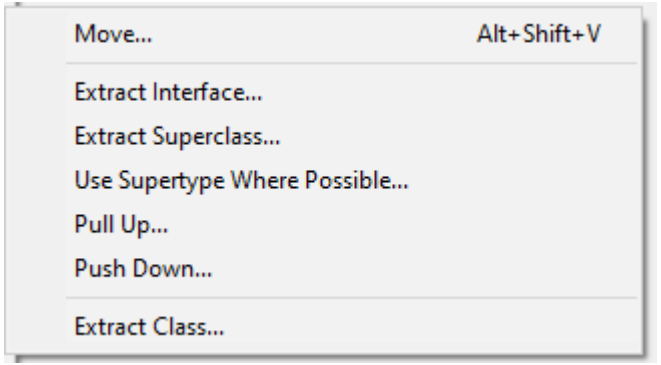
Estos síntomas son los siguientes:

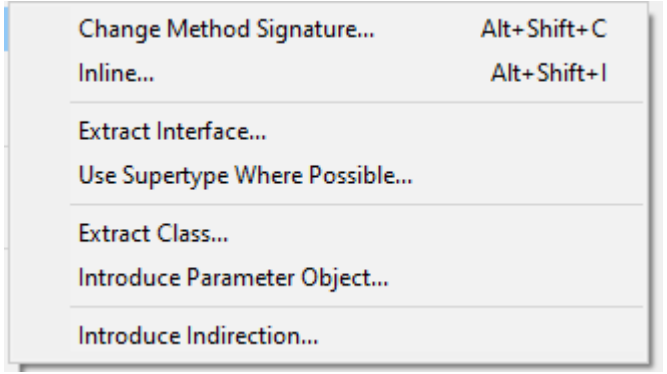
- **Código duplicado.**
- **Métodos muy largos**
- **Clases muy grandes**
- **Listas muy grandes de parámetros**
- **Cirugía a tiro pistola**(Shotgun surgery): este síntoma se presenta cuando después de un cambio en una determinada clase se deben realizar varias modificaciones adicionales en diversos lugares para compatibilizar dicho cambio.
- **Clase de solo datos**: clase que solo tienen atributos y métodos de acceso a ellos get y set , este tipo de clases debería cuestionarse dado que no suelen tener comportamiento alguno.

Refactorización en Eclipse

- Menús contextuales con diferentes opciones de refactorización

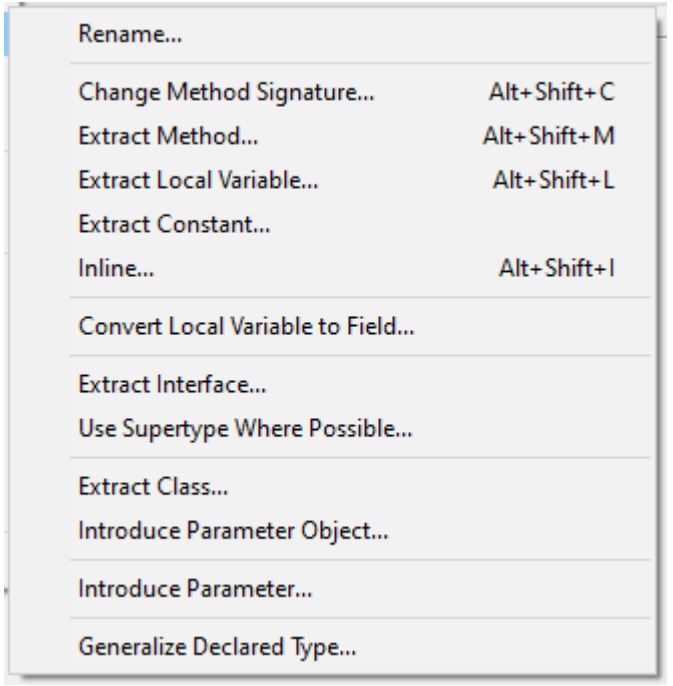
Clase

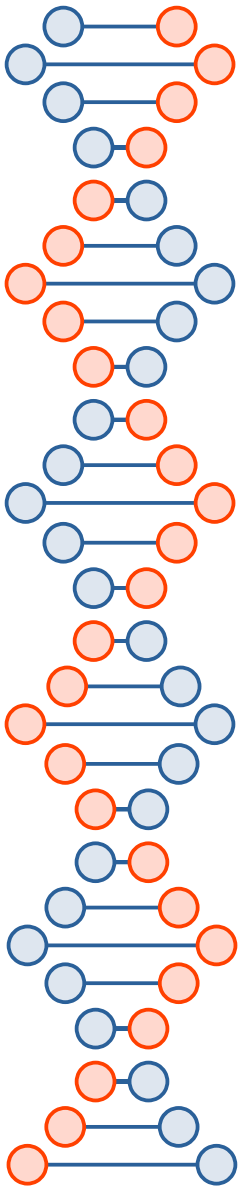




Método

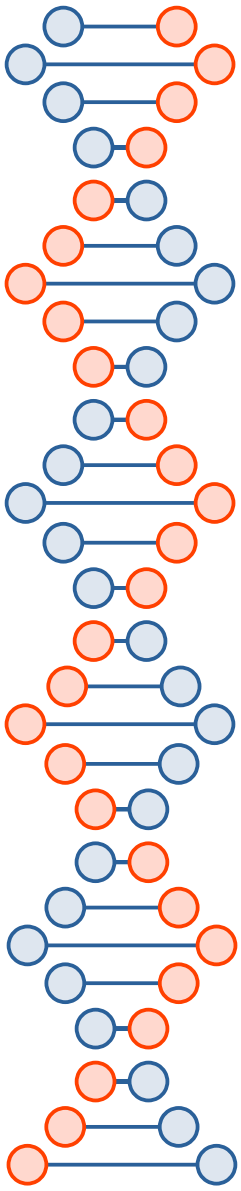
Variable





- TABULACIÓN

  La tabulación puede que no sea una técnica o una práctica propia de la refactorización en sí misma, ya que no se modifica código. No obstante, con la tabulación, el código queda más claro, con lo que también resulta más fácil de ver y entender, que es parte del objetivo de la refactorización.



PATRONES DE REFACTORIZACIÓN MÁS USUALES

RENAME:

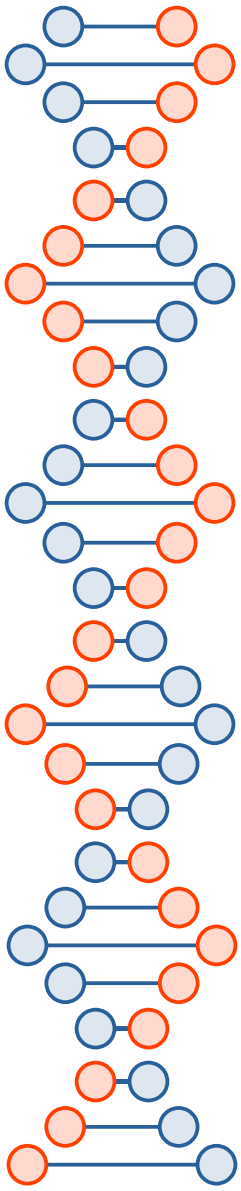
Es una de las opciones más utilizadas cambia el nombre de variables clases métodos paquetes tras la refactorización se modifican las referencias a ese identificador.

MOVE:

Mueve una clase de un paquete a otro, se mueve el archivo .Java a la carpeta, y se cambien todas las referencias. También se puede arrastrar y soltar una clase a un nuevo paquete, se realiza una refactorización. automática.

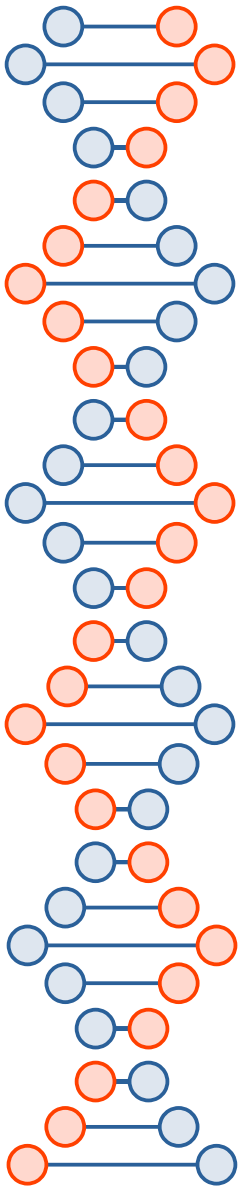
Extract constant

Convierte un número o cadena literal en una constante, al hacer la refactorización se mostrará donde se van a producir los cambios y se puede visualizar el estado antes y después de la refactorización.



PATRONES DE REFACTORIZACIÓN MÁS USUALES

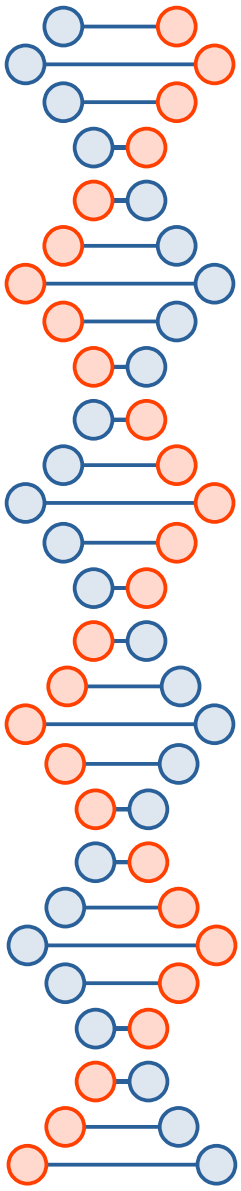
- Extract Local Variable asigna una expresión a una variable local. Tras la refactorización cualquier referencia a la expresión en el ámbito local se sustituye por la variable la misma expresión. En otro método no se modifica.
- Convert Local Variable to Field. Convierte una variable local en un atributo privado de la clase. Tras la refactorización todos los usos de la variable local se sustituyen por ese atributo.



PATRONES DE REFACTORIZACIÓN MÁS USUALES

- Extract Method:

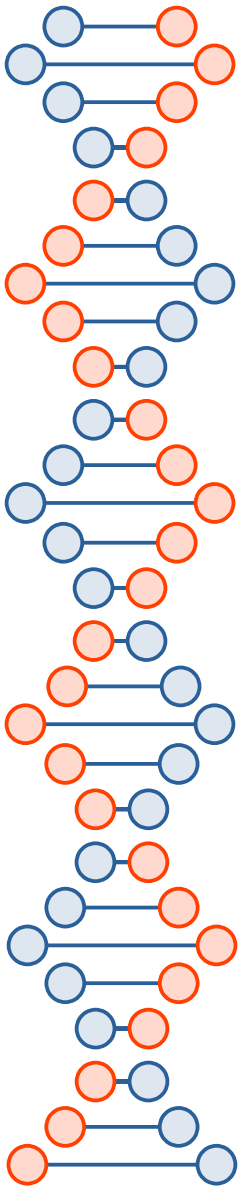
  Nos permite seleccionar un bloque de código y convertirlo en un método. El bloque de código no debe dejar llaves abiertas. Es muy útil extraer un método cuando se tiene un grupo de instrucciones que se repiten varias veces. Al extraer el método hay que indicar el modificador de acceso: público protegido privado...



- Change Method signature Este patrón permite cambiar la firma de un método. Es decir el nombre del método, los parámetros que tiene y lo que devuelve.

  Nota:Si al refactorizar cambiamos el tipo de dato de retorno del método aparecerán errores de compilación por lo que debemos modificarlo manualmente

- Inline nos permite ajustar una referencia a una variable o método con la línea en la que se utiliza y conseguir así una única línea de código.



PATRONES DE REFACTORIZACIÓN MÁS USUALES

- Extract Superclass Este patrón permite extraer una superclase. Si la clase ya utilizaba una superclase, la recién creada pasará a ser súperclase. Se pueden seleccionar los métodos y atributos que formarán parte de la superclase.
- Convert anonymous class to nested Este patrón de refactorización permite convertir una clase anónima a una clase anidada de la clase que la contiene. *Una clase anónima es una clase sin nombre de la que sólo se crea un único objeto*, de esta clase no se pueden definir constructores. Se utilizan con frecuencia cuando se crean ventanas para gestionar los eventos de los distintos componentes de la interfaz gráfica.

Extraer método . Ejemplo

- EXTRAER MÉTODO

```
void imprimirTodo() {
imprimirBanner();
//detalles de impresión
Console.WriteLine (“nombre: “ + _nombre);
Console.WriteLine(“cantidad “ + getCargoPendiente());
}
```

Refactorizamos

```
void ImprimirTodo() {
imprimirBanner();
imprimirDetalles(getCargoPendiente());
}
void imprimirDetalles(double cargoPendiente) {
Console.WriteLine (“nombre: “ + _nombre);
Console.WriteLine(“cantidad “ + cargoPendiente);
}
```