Unidad 6: Refactorización

Módulo: EDE

Profesora: Estefania Monerri

¿Qué es la refactorización?

Técnica disciplinada para efectuar cambios en la estructura interna de un código sin cambiar su comportamiento externo.

¿Por qué refactorizar?

- Para mejorar su diseño

  a. Conforme se modifica, el software cambia su estructura.

  b. Eliminar código duplicado simplifica su mantenimiento.

- Para hacerlo más entendible

  La legibilidad del código facilita su mantenimiento.

- Para encontrar errores

  Al reorganizar un programa, se pueden apreciar con mayor facilidad las suposiciones que hayamos podido hacer.

- Para programar más rápido

  Al mejorar el diseño del código, mejorar su legibilidad y reducir los errores que se cometen al programar, se mejora la productividad de los programadores.

¿Cuándo se debe refactorizar?

- Cuando se está escribiendo nuevo código

Al añadir nueva funcionalidad a un programa (o modificar su funcionalidad existente), puede resultar conveniente refactorizar:

a. para que éste resulte más fácil de entender, o

b. para simplificar la implementación de las nuevas funcionalidades.

- Cuando se corrige un error

La mayor dificultad de la depuración de programas radica en que hemos de entender exactamente cómo funciona el programa para encontrar el error. Cualquier refactorización que mejore la calidad del código tendrá efectos positivos en la búsqueda del error.

- Cuando se revisa el código

Una de las actividades más productivas desde el punto de vista de la calidad del software es la realización de revisiones del código (recorridos e inspecciones).

¿Por qué es importante la refactorización?

Cuando se corrige un error o se añade una nueva función, el valor actual de un programa aumenta. Sin embargo, para que un programa siga teniendo valor, debe ajustarse a nuevas necesidades (mantenerse), que puede que no sepamos prever con antelación. La refactorización, precisamente, facilita la adaptación del código a nuevas necesidades.

¿Qué síntomas indican que se debería refactorizar?

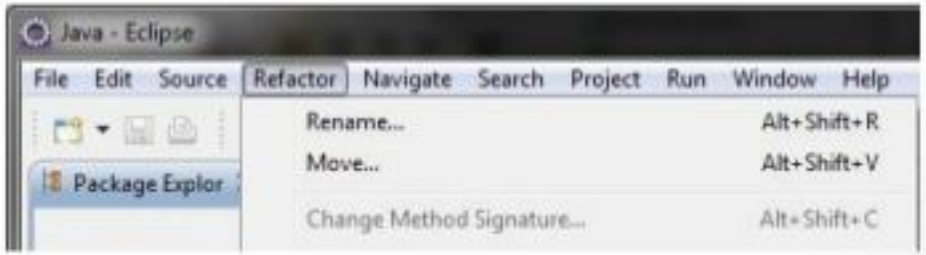
El código es difícil de entender cuando:

1. Usa identificadores mal escogidos
2. Incluye fragmentos de códigos duplicados
3. Incluye lógica condicional compleja
4. Los métodos usan un número elevado de parámetros
5. Está dividido en módulos enormes
6. Un método accede continuamente a los datos de un objeto de una clase diferente a la clase en la que está definida (posiblemente, el método debería pertenecer a la otra clase).
7. Etc.

Patrones de refactorización más comunes:

1. *Rename*

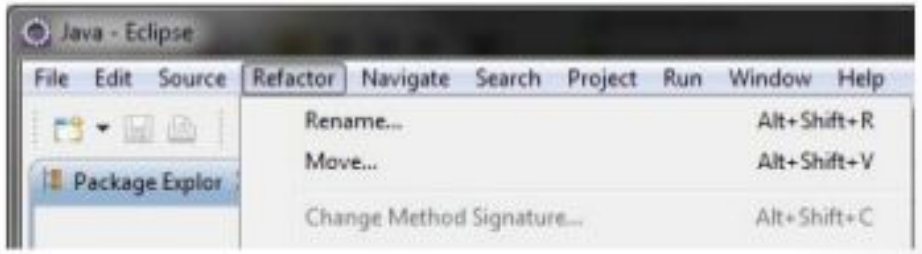
- Cambia el nombre de variables, clases, métodos, paquetes, directorios y casi cualquier identificador Java.
- Tras la refactorización, se modifican las referencias a ese identificador.
- Refactor > Rename...
- Alt + Shift + R



Patrones de refactorización más comunes:

2. *Move*

- Mover una clase de un paquete a otro.
  - Se mueve el archivo *.java* a la carpeta.
  - Se cambian todas las referencias.
- Arrastrar y soltar una clase a un nuevo paquete.
  - Refactorización automática.
- Refactor > Move...
- Alt + Shift + V



Patrones de refactorización más comunes:

3. *Extract Local Variable*

- Asignar expresión a 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.

```
public class ExtractLocalVariable {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

```
public class ExtractLocalVariable {

    public static void main(String[] args) {
        String string = "Hello World!";
        System.out.println(string);
    }
}
```

Patrones de refactorización más comunes:

4. *Extract Constant*

- Convierte un número o cadena literal en una constante.
- Tras la refactorización, todos los usos del literal se sustituyen por esa constante.
- Objetivo: Modificar el valor del literal en un único lugar.

```java
public class ExtractConstant {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

```java
public class ExtractConstant {

    private static final String HELLO_WORLD = "Hello World!";

    public static void main(String[] args) {
        System.out.println(HELLO_WORLD);
    }
}
```

Patrones de refactorización más comunes:

5. *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.

```
public class ConvertLocalVariable {

    public static void main(String[] args) {
        String msg = "Hello World!";
        System.out.println(msg);
    }
}
```

```
public class ConvertLocalVariable {

    private static String msg;

    public static void main(String[] args) {
        msg = "Hello World!";
        System.out.println(msg);
    }
}
```