GENERALITAT VALENCIANA
Conselleria d'Educació, Investigació, Cultura i Esport
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Unió Europea
Fons Social Europeu
L'FSE inverteix en el teu futur
IES Benigasló
Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Entornos de Desarrollo

Code refactoring

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Contenidos:

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Introducción.-

«*Code refactoring is the process of restructuring existing computer code without changing its external behavior*» (https://en.wikipedia.org/wiki/Code_refactoring).

Como podéis observar en la definición, refactorizar es una técnica de ingeniería del software que se basa en modififcar el código, de forma que sea más comprensible y legible, siempre sin alterar su correcto funcionamiento.

La motivación de la refacción sería la evidencia de lo que se llama «code smell» (olores del código), que tiene su nacimiento en métodos muy largos, con posible código duplicado, nombres de variables y/o métodos poco intuitivos, etc. No siempre será fácil detectar malos olores, iremos mejorando fruto de nuestra experiencia en programación.

Tipos de refacción.-

Composición de los métodos.

I. Extraer un método (Extract Method).

Se basa en un patrón para extraer un conjunto de secuencias a un método que las sustituya. Este método debería recibir un nombre significativo de la acción que realiza. Es aconsejable crear métodos que realicen una única acción (métodos atómicos).

Imaginemos un método que debe imprimir varias líneas de código de una factura, las cuales, excepto por el contexto, no sabemos de qué parte de la factura son:

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Factura_NOK.java

```java
public class Factura_NOK {
    public void printFactura(){
        System.out.println("header-nombre");
        System.out.println("header-apellidos");
        System.out.println("header-DNI");

        System.out.println("body-linea1");
        System.out.println("body-linea2");
        System.out.println("body-linea3");
        System.out.println("body-linean");

        System.out.println("footer-subtotal");
        System.out.println("footer-total");
        System.out.println("footer-iva");
    }
}
```

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

La idea sería agrupar en métodos las líneas comunes de la cabecera, el cuerpo y el pie de la factura, tal como se muestra a continuación:

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Factura_OK.java

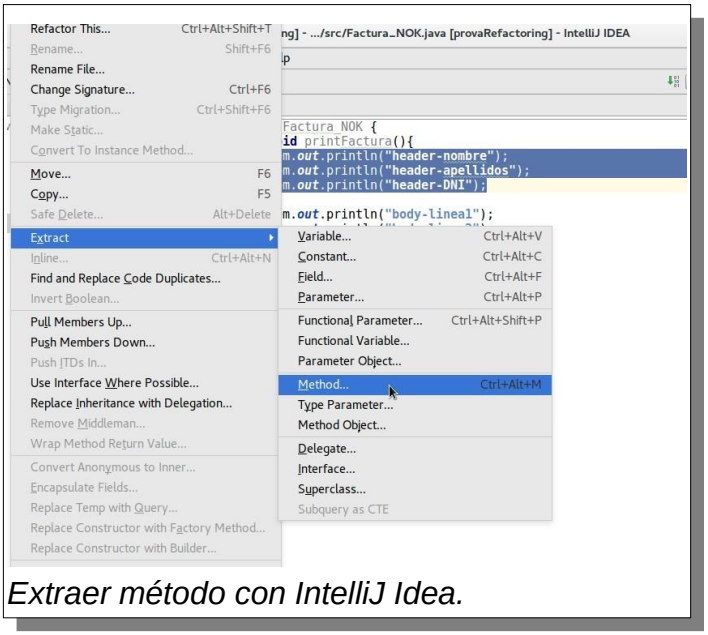
```java
public class Factura_OK {

    public void printFactura(){
        printHeader();
        printBody();
        printFooter();
    }

    private void printFooter() {
        System.out.println("footer-subtotal");
        System.out.println("footer-total");
        System.out.println("footer-iva");
    }

    private void printBody() {
        System.out.println("body-linea1");
        System.out.println("body-linea2");
        System.out.println("body-linea3");
        System.out.println("body-linean");
    }

    private void printHeader() {
        System.out.println("header-nombre");
        System.out.println("header-apellidos");
        System.out.println("header-DNI");
    }
}
```

Podemos ver que el método "printFactura" queda definido claramente por las acciones que realiza y tenemos la posibilidad de reutilizar los métodos de imprimir cabecera, cuerpo y pie.

II. Extraer un método en diferentes entornos de desarrollo.

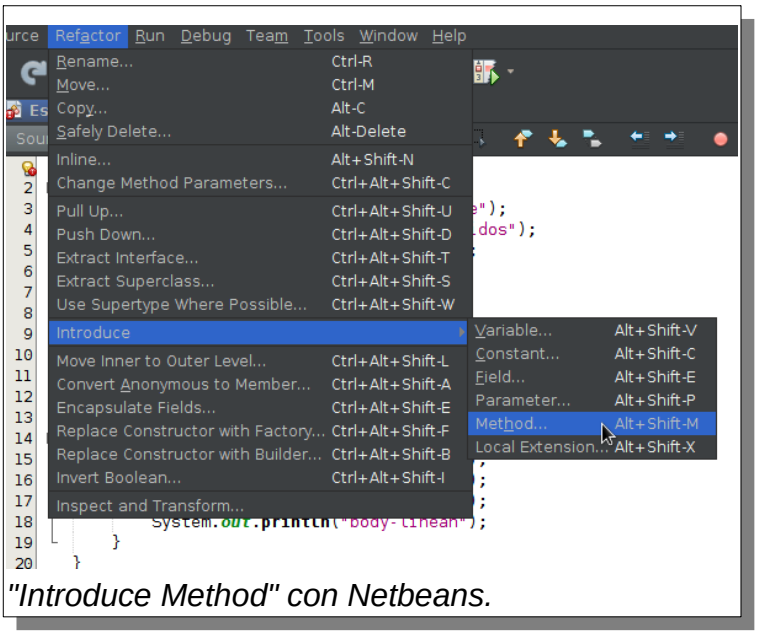
Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Normalmente, seleccionaremos el código al cual queremos aplicar la refacción, iremos al menú, haremos clic en "Refactor", seguido de "Extract" y después

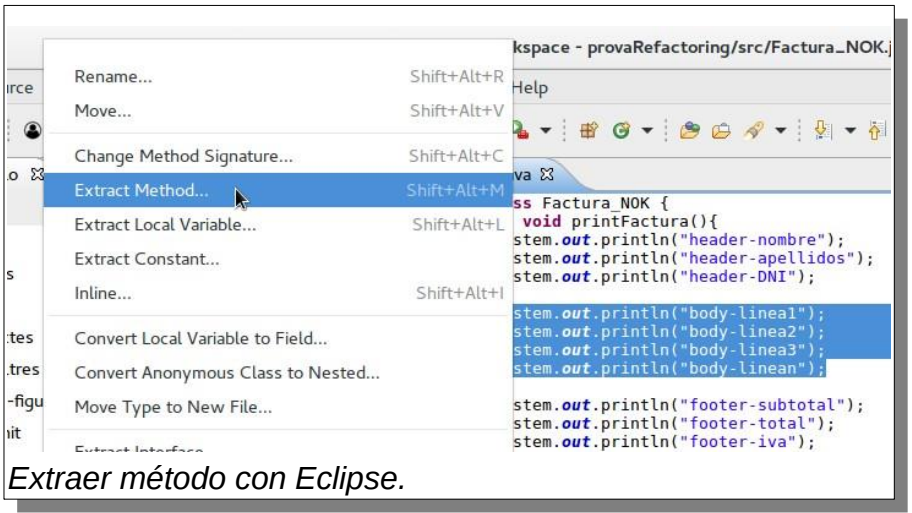


Extraer método con IntelliJ Idea.

"Method":



"Introduce Method" con Netbeans.



Extraer método con Eclipse.

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

III. Método en linea (Inline Method).

En ocasiones, una división excesiva del código puede llegar a complicar la comprensión del mismo y por lo tanto puede ser factible agrupar en un único método, por ejemplo:

Cercle_NOK.java

```java
public class Cercle_NOK {
    public double calculateAreaOfCircle (double radius) {
        double area = getValueOfPI() * Math.pow(radius, 2);
        return area;
    }

    private double getValueOfPI(){
        return Math.PI;
    }
}
```

Como se puede observar, la separación de la sentencia “Math.PI” en otro método, no aporta mayor beneficio y en este caso es mejor dejarlo dentro del método original «calculateAreaOfCircle»:

Cercle_OK.java

```java
public class Cercle_OK {

    public double calculateAreaOfCircle (double radius) {
        double area = Math.PI * Math.pow(radius, 2);
        return area;
    }
}
```

Aun así, todavía se puede aplicar una refacción más (**Inline Temp**) que veremos más adelante, ya que tenemos una variable temporal (area) que solo sirve para calcular un valor intermedio, quedaría más correcto de la siguiente forma:

Cercle_OK.java

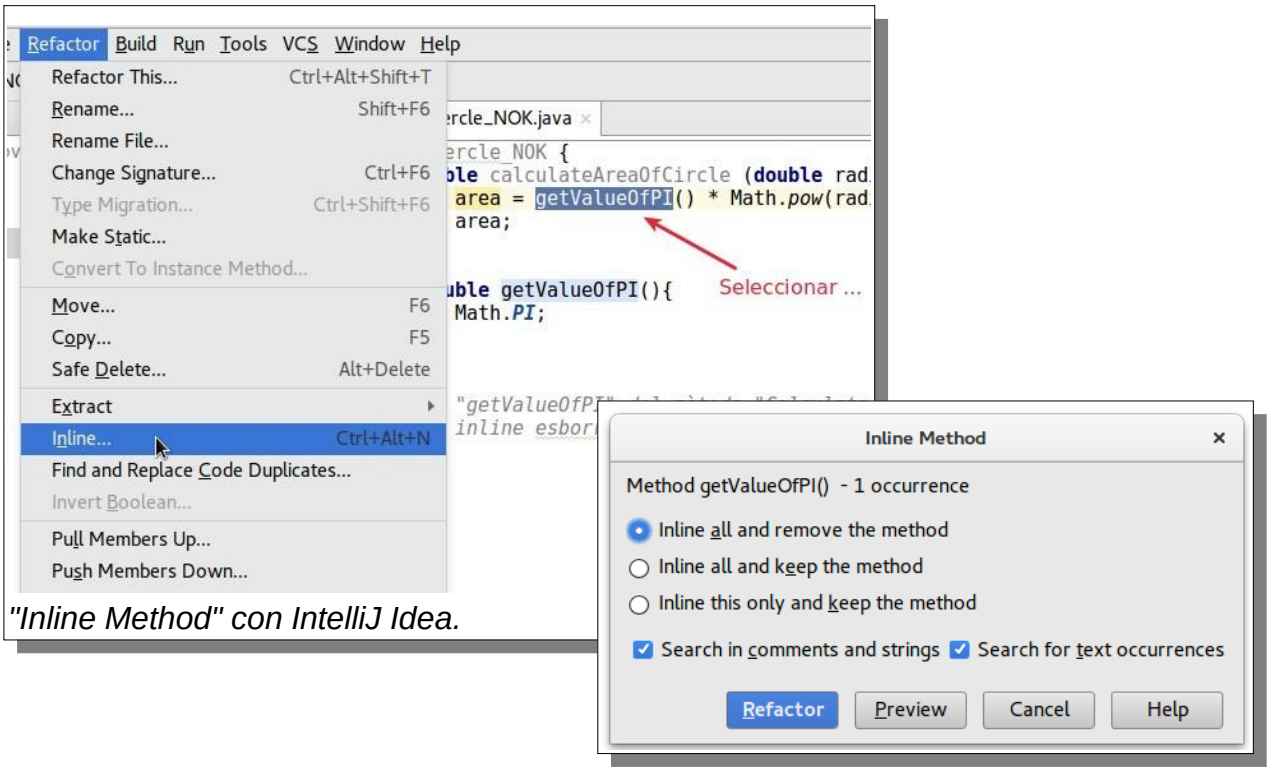
```java
public class Cercle_OK {

    public double calculateAreaOfCircle (double radius) {
        return Math.PI * Math.pow(radius, 2);
    }
}
```

Ya que no se pierde información y sigue quedando clara la función del método.

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

IV. Método en línea para diferentes entornos de desarrollo.

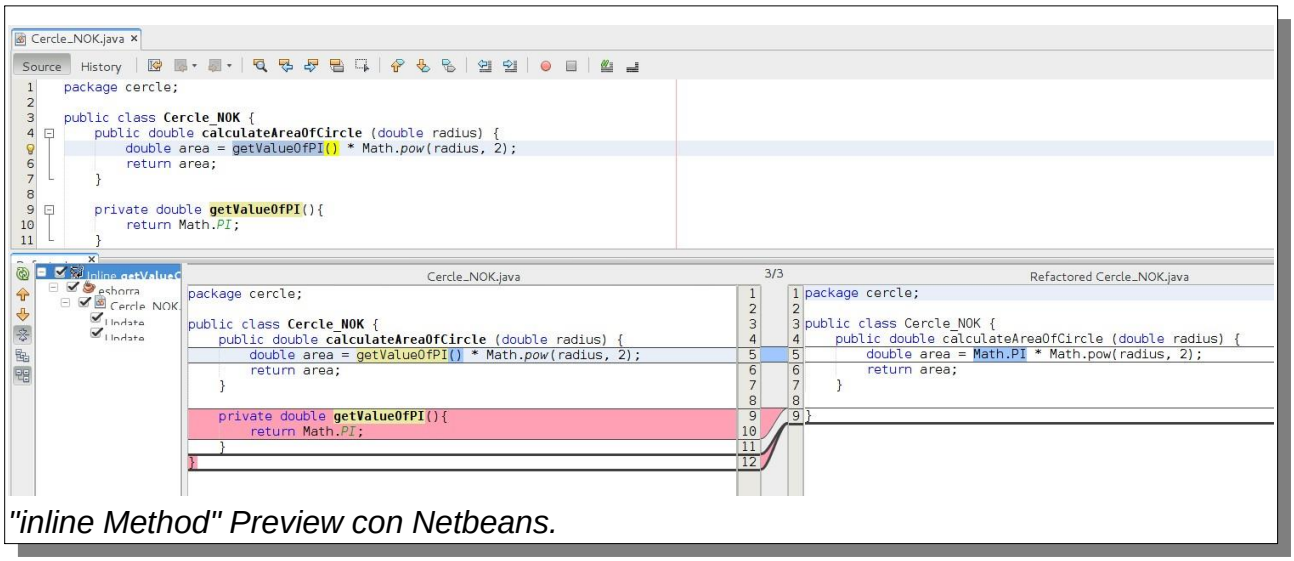
Generalmente, debemos seleccionar la "llamada" al método al cual queremos aplicar la refacción, hacer clic en "Refactor", seleccionar "Inline" y después eliminar el método que sobra. Se puede ver un ejemplo con *IntelliJ Idea*:



"Inline Method" con IntelliJ Idea.

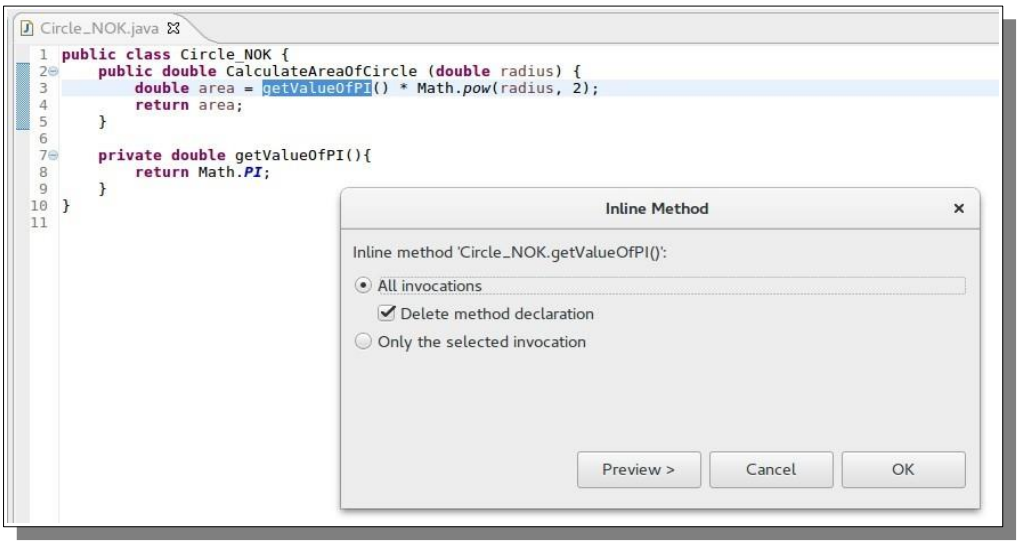
Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

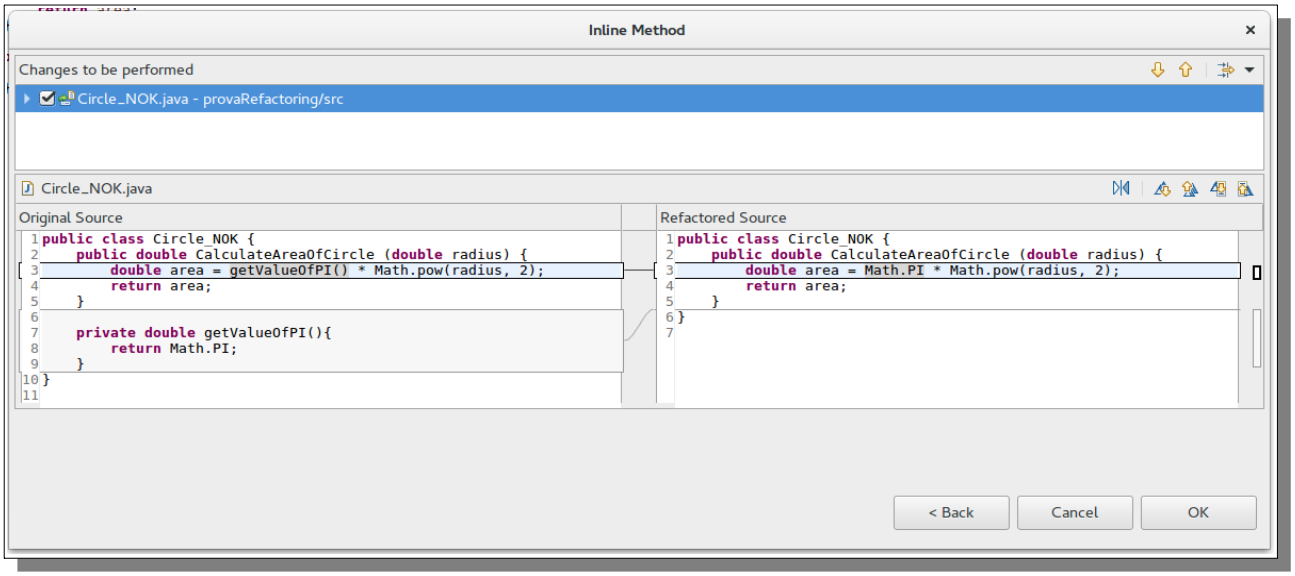
El mismo proceso, realizado con Netbeans:



"inline Method" Preview con Netbeans.

Inline Method con Eclipse:





Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

V. Variables temporales (Inline Temp).

A menudo, se utilizan variables temporales para realizar cálculos intermedios, aunque en ocasiones, pueden ser prescindibles siempre que no perjudiquen la legibilidad:

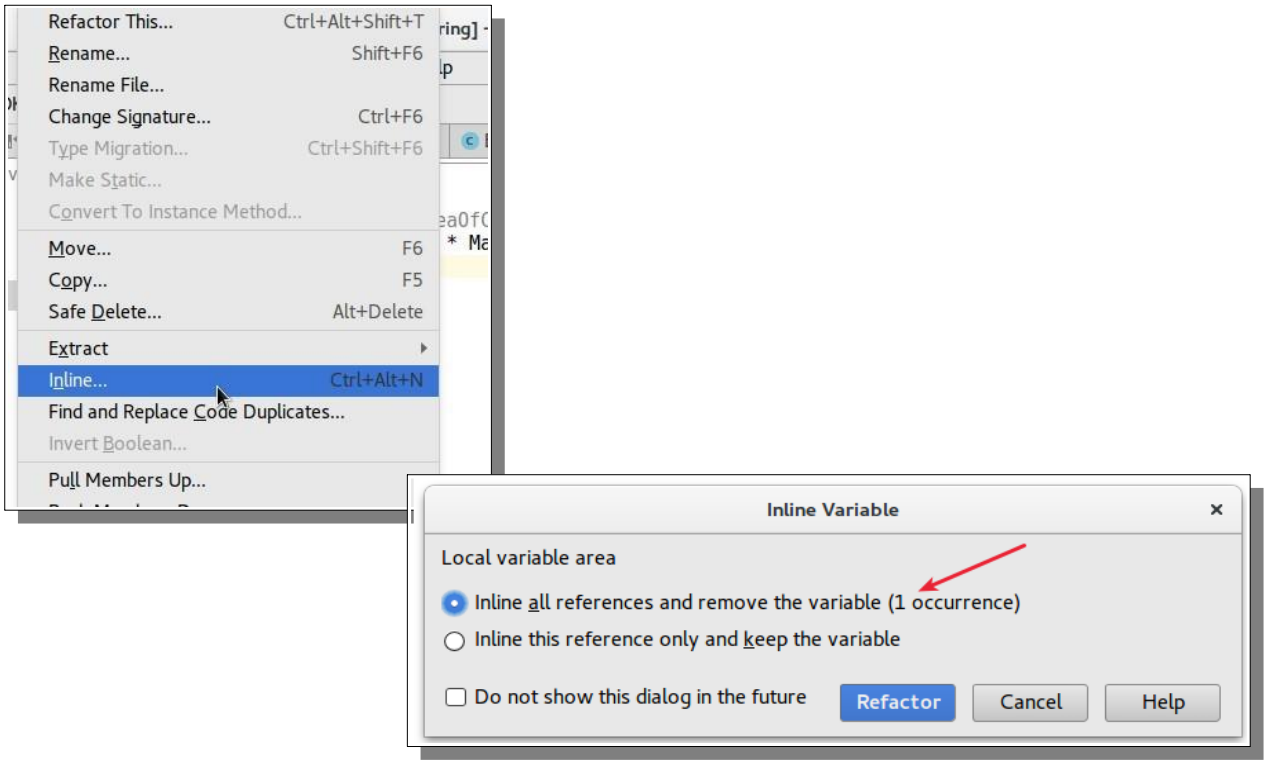
```
bool hasDiscount(Order order) {
  double basePrice = order.basePrice();
  return basePrice > 1000;
}
```

```
bool hasDiscount(Order order) {
  return order.basePrice() > 1000;
}
```

Para aplicar la refacción, por ejemplo con IntelliJ Idea, debemos hacer:

Seleccionar la variable en la linea del «return» y aplicar «Refactor» → «Inline...».

```
public class Cercle_OK {

    public double calculateAreaOfCircle (double radius) {
        double area = Math.PI * Math.pow(radius, 2);
        return area;
    }
}
```



Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Que quedará:

Cercle_OK.java

```java
public class Cercle_OK {

    public double calculateAreaOfCircle (double radius) {
        return Math.PI * Math.pow(radius, 2);
    }
}
```

VI. Extraer una variable (Extract Variable).

Suele ser aplicado cuando tenemos una expresión difícil de evaluar, intentando que el código quede más claro. Si esta expresión aparece en más lugares, posiblemente deberíamos aplicar extraer un método. El proceso inverso de extraer una variable es el de «Inline Temp».

```java
public void makeEnroll(){
    if (age >= 15 && hasFathers && previousQualification >= 5){
        /*
        Do Something ...
         */
    }
}
```

Antes de extraer la variable

```java
public void makeEnroll(){
    final boolean ageOK = age >= 15;
    final boolean prevQisOK = previousQualification >= 5;

    if (ageOK && hasFathers && prevQisOK){
        /*
        Do Something ...
         */
    }
}
```

Después de extraer la variable

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

VII. Renombrar un método, variable, etc.

Una técnica sencilla que nos permite ahorrar comentarios en el código y que al mismo tiempo ayuda a entenderlo mejor, es la de aplicar nombres autoexplicativos a las constantes, variables, métodos, etc. Estos nombres deben permitir entender el tipo de datos que contiene la constante o variable o bien, la acción que realiza el método o función.

Los IDE suelen tener la opción de «**Refactor**» → «**Rename**».
Algunos ejemplos de nombres que se han cambiado para dar más sentido a la acción que realiza un método o función, o el contenido de una constante o variable:

```
public void ip(Factura fact){
    //fes alguna cosa
}

public void imprimirFactura(Factura fact){
    //fes alguna cosa
}
```

La segunda opción nos da más información que la primera.

Un ejemplo de una constante, juzgad vosotros mismos cual aclara más el contenido del fichero "users.txt":

```
private static final String M = "users.txt";

private static final String FILE = "users.txt";

private static final String USERS_FILE = "users.txt";
```

Mover características entre objetos.

Esta técnica o conjunto de técnicas, consiste en mover una más funcionalidades a otras clases, o incluso crear nuevas clases.

I. Mover un método (Move Method).

Podemos aplicar esta técnica cuando un método de una clase se utiliza más en otra clase, o tiene más sentido por los datos que trata (o sea, que sería más

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

coherente en otra clase). Un ejemplo un poco "forzado" sería el siguiente:
Una clase círculo (Cercle) que dispone de los métodos para calcular el área, el perímetro y el volumen. El cálculo del volumen debería pertenecer a una clase específica que tenemos llamada «Esfera» y que solo dispone de un método para calcular el área.

Cercle.java

```
1  public class Cercle {
2      private double radi;
3
4      public Cercle (double unRadi){
5          radi = unRadi;
6      }
7      public double area(){
8          return Math.PI * Math.pow(radi, 2);
9      }
10
11     public double perimetre(){
12         return 2 * Math.PI * radi;
13     }
14
15     public double volum(){
16             return (4 * Math.PI * Math.pow(radi, 3)) / 3;
17     }
18 }
```

Esfera.java

```
1  public class Esfera {
2      private double radi;
3
4      public Esfera(double unRadi) {
5          radi = unRadi;
6      }
7
8      public double area(){
9          return 4 * Math.PI * Math.pow(radi, 2);
10     }
11 }
```

La siguiente imagen muestra el método "volum" dentro de la clase "Esfera". Esta acción se ha realizado de forma **manual**.

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Esfera.java

```java
public class Esfera {
    private double radi;

    public Esfera(double unRadi) {
        radi = unRadi;
    }

    public double area(){
        return 4 * Math.PI * Math.pow(radi, 2);
    }

    public double volum(){
        return (4 * Math.PI * Math.pow(radi, 3)) / 3;
    }
}
```

Debemos tener en cuenta si este método se utilizaba en otras clases, ya que deberá ser redirigido a la clase correcta.

II. Extraer una clase (Extract class).

Debemos aplicar esta técnica cuando una clase realice más tareas (tenga más responsabilidades) de las que debería hacer. Observando la siguiente clase:

Forma.java

```java
public class Forma {

    public void dibuixar(){
        try {
            // Intenta dibuixar
        }
        catch (Exception e){
            manegaErrors(e);
        }
    }

    public void manegaErrors(Exception e){
        // Tractament dels diferents errors
    }
}
```

Esta clase, que representa una "Forma" cualquiera, además del método paradibujar dicha forma, dispone de un método para el tratamiento de errores. El problema en este caso radica en que el tratamiento de errores debería ser realizado por una clase especializada, y quitar esta responsabilidad a la clase "forma".

Si aplicamos la refacción, obtenemos:

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Forma.java

```java
public class Forma {

    public void dibuixar(){
        try {
            // Intenta dibuixar
        }
        catch (Exception e){
            Manegador.manegaError(e);
        }
    }
}
```

Manegador.java

```java
public class Manegador {
    public static void manegaError(Exception e){
        // Tractament dels diferents errors
    }
}
```

Se ha descargado la responsabilidad de los errorres sobre una clase especializada que maneja los errores llamada “Manegador” (manejador).

Otras consideraciones a tener en cuenta en la refacción.

I. Principio Tell-Don’t-Ask (evitar getters y setters).

Este principio intenta aplicar de forma más eficiente la característica de la programación orientada a objetos. Trata de evitar que se pidan datos del objeto para hacer cálculos con ellos en otro objeto distinto, más bien lo que pretendemos es indicar/pedir al objeto qué queremos que haga. Se persigue evitar errores en el tratamiento de datos, además de no dar detalles de la implementación.
Para entender mejor lo que se quiere, veremos un ejemplo:

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Cliente.java

```java
public class Cliente {
    private int edad;

    public Cliente(int unaEdad){
        edad = unaEdad;
    }

    public int obtenerEdad(){
        return edad;
    }

    public boolean esAdulto(){
        return (edad >= 18)? true : false;
    }
}
```

Pendiente de refacción

Operador Ternario.

Se puede observar que la clase "Cliente" tiene dos métodos, "obtenerEdad", que simplemente retorna la edad del cliente, y "esAdulto", que realiza una operación para determinar si el cliente es mayor de edad o no.

Ahora observamos la aplicación de esta clase y sus métodos mediante la clase "Taberna":

Taberna.java

```java
public class Taberna {

    public static void main(String[] args){
        Cliente unCliente = new Cliente( unaEdad: 12);

        if (unCliente.obtenerEdad() >= 18){
            System.out.println("Servir cubata");
        }

        if (unCliente.esAdulto()){
            System.out.println("Servir cubata");
        }

    }
}
```

Observando el código, ¿qué condición «if» queda més clara? Evidentemente, la segunda. Se puede observar que prácticamente se puede leer como una frase normal (en inglés) «**Si un cliente es adulto ...**».

Todavía podríamos realizar otra refacción en la clase «Cliente», se puede extraer

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

una constante:

Cliente.java

```java
public class Cliente {
    private static final int MAYORIA_EDAD = 18;
    private int edad;

    public Cliente(int unaEdad){
        edad = unaEdad;
    }

    public int obtenerEdad(){
        return edad;
    }

    public boolean esAdulto(){
        return (edad >= MAYORIA_EDAD)? true : false;
    }
}
```

Podemos también simplificar la expresión en el método «esAdulto»:

Cliente.java

```java
public class Cliente {
    private static final int MAYORIA_EDAD = 18;
    private int edad;

    public Cliente(int unaEdad){
        edad = unaEdad;
    }

    public int obtenerEdad(){
        return edad;
    }

    public boolean esAdulto(){
        return edad >= MAYORIA_EDAD;
    }
}
```

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

II. Evitar las cláusulas «ELSE».

Cuando el nivel de identación de un código comienza a ser excesivo, se corre el riesgo de perder legibilidad. Esto puede pasar cuando se acumulan varios IF anidados.

Para comprobar la diferencia, se muestra a continuación un código que calcula el entero mayor de entre tres que se pasan como argumentos. Podemos ver la forma original y la que tiene aplicada la refacción:

```java
public int obtener_mayor(){
    if (valor1 > valor2){
        if (valor1 > valor3){
            return valor1;
        }
        else {
            return valor3;
        }
    }
    else if (valor2 > valor3){
       return valor2;
    }
    else {
        return valor3;
    }
}
```

Original

```java
public int obtener_mayor(){
    // valor1 és el major
    if (valor1 >= valor2 && valor1 >= valor3) {
        return valor1;
    }
    // valor2 és el major
    if (valor2 >= valor3) {
        return valor2;
    }
    // valor3 és el major
    return valor3;
}
```

Con refacción

III. «Bad Smell».

Podemos encontrar muchos ejemplos de malos olores, por ejemplo:

```java
private boolean areSameImages(){
    if (selectedImages.get(0).equals(selectedImages.get(1))){
        return true;
    }
    else {
        return false;
    }
}
```

Bad Smell

```java
private boolean areSameImages(){

    return selectedImages.get(0).equals(selectedImages.get(1));
}
```

Forma más correcta

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Ejemplo de IF's anidados y valores enteros repartidos por el código:

```java
public class Banquete {

    public int obtenerPrecio(int personas){

        int total = 0;
        if(personas > 200){
            total = (personas * 50);
        }
        else if(personas > 100){
            total = (personas * 75);
        }
        else {
            total = (personas * 95);
        }
        return total;
    }
}
```

El mismo resultado sin cláusulas ELSE, utilizando constantes y renombrando variables:

```java
public class Banquete {

    private final int LIMITE_SUP = 200;
    private final int LIMITE_INF = 100;
    private final int PRECIO_MAYOR_L_SUP = 50;
    private final int PRECIO_MAYOR_L_INF = 75;
    private final int PRECIO_NORMAL = 95;

    public int obtenerPrecioBanquete(int numComensales) {

        if (numComensales > LIMITE_SUP) {
            return PRECIO_MAYOR_L_SUP * numComensales;
        }

        if (numComensales > LIMITE_INF) {
            return PRECIO_MAYOR_L_INF * numComensales;
        }

        return PRECIO_NORMAL * numComensales;
    }
}
```

Avda. Arcadi Garcia, 1
12.600 La Vall d'Uixó
Tel. 964 738955
Mail: 12005751@gva.es
Web: iesbenigaslo.es

Bibliografía.-

- Refactoring Guru <https://refactoring.guru/ >
- Catalog of Refactorings <https://refactoring.com/catalog/ >
- A qué huele tu código <https://www.slideshare.net/rubenbp/a-qu-huele-tu-cdigo-afinando-nuestro-olfato-7400175?ref=http://www.mikelnino.com/2014/04/a-que-huele-tu-codigo-presentacion-ruben-bernardez-buenas-practicas-programacion-refactorizacion-codigo.html >
- Refactorización en Eclipse <https://es.slideshare.net/srcid/eclipse-refactoring >
- Java Eclipse Tutorial – Part 6.2: Refactoring Code (Extract Methods, Rename Methods and Variables) <http://www.luv2code.com/2014/08/13/java-eclipse-tutorial-part-6- 2-refactoring-code-extract-methods-rename-methods-and-variables/ >
- Refactoring Techniques: Extract Method <https://zaengle.com/blog/refactoring-techniques-extract-method >
- La guía definitiva del código espagueti II: Que tus variables y funciones confundan al enemigo. Urlan Heat <https://urlanheat.com/blog/la-guia-definitiva-del-codigo-espagueti-ii-que-tus-variables-y-funciones-confundan-al-enemigo/ >
- La guía definitiva del código espagueti III: Comenta y miente. Urlan Heat <https://urlanheat.com/blog/la-guia-definitiva-del-codigo-espagueti-iii-comenta-y-miente/ >
- Extract Class refactoring

  <https://www.jetbrains.com/help/resharper/Refactorings_Extract_Class.html >
- TellDontAsk. <https://martinfowler.com/bliki/TellDontAsk.html >
- Por qué programar sin usar "else" – Cláusulas de guarda – #Refactoring #MoviolaCodelyTV <https://codely.tv/screencasts/clausulas-guarda-refactoring/ >