Proyecto Final Entornos del Desarrollo

En este proyecto vamos a realizar una refactorización de varios códigos los cuales iré desarrollando paso a paso para que pueda entenderse correctamente, para ello voy a comenzar dando una explicación acerca de lo que sucede con el primero de los códigos el cual nos plantea el proyecto.

- En primer lugar haremos funcionar los test que se nos proponen, por lo que el principal objetivo es comprobar si existe algún tipo de fallo en los mismos, una vez comprobado este aparece que ha sido pasado satisfactoriamente.

- El código que se nos propone es un poco tedioso a la hora de realizar la refactorización, por lo que nos han permitido en el proyecto utilizar nuestro propio código y realizar las técnicas de refactorización en el código, por lo que iré mostrando todo lo que sucede en el código mediante capturas de pantalla.

1. Para comenzar vamos a realizar un refactor de la técnica **Extract Method**

```
public class CalcularNotas {
    public String calcularNotas(int nota) {
        String respuesta = " ";
        if (nota<5) {
            suspenso();
        } else if (nota==5) {
            suficiente();
        } else if (nota>5 && nota<7) {
            bien();
        } else if (nota>=7 && nota<9) {
            notable();
        } else if (nota==10) {
            sobresaliente();
        }
        return respuesta;
    }
```

Este sería el código a refactorizar, primero he creado el código el cual se encarga de realizar la comprobación de las notas en un intervalo, dependiendo del número introducido lo organizará en una sección de notas. Una vez comprobado que el código generado anteriormente funciona exitosamente por lo que procedemos a usar la técnica de refactorización.

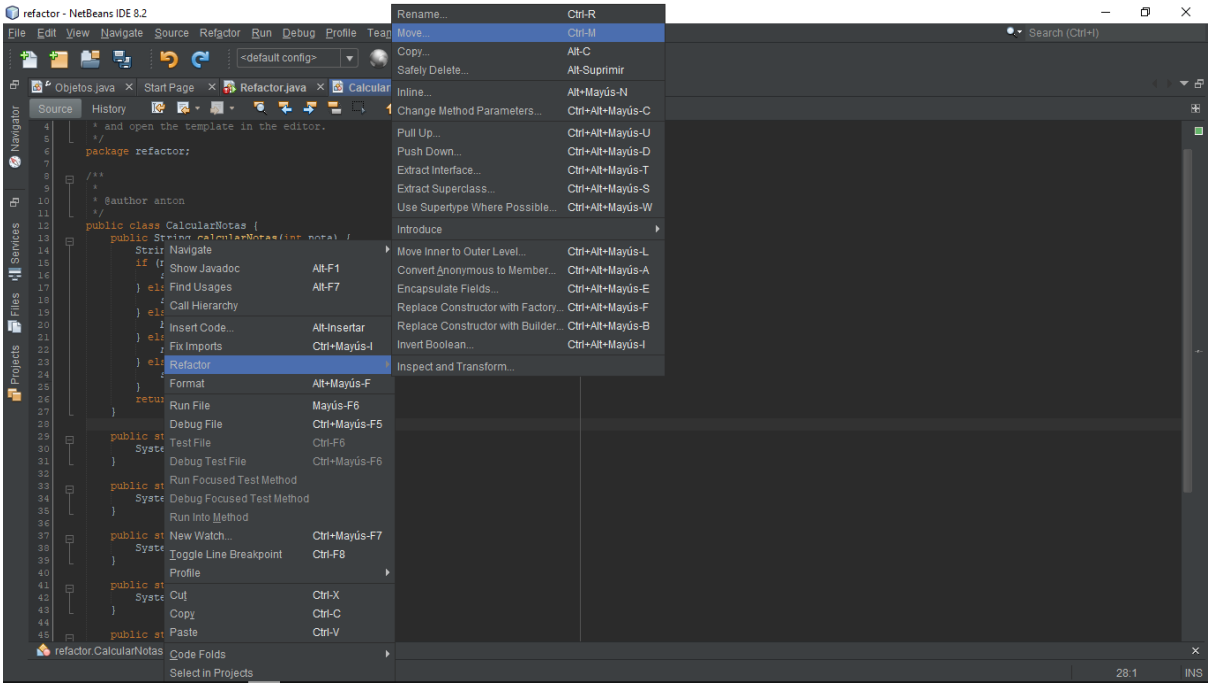
```
    public static void suspenso() {
        System.out.println("Suspenso");
    }

    public static void suficiente() {
        System.out.println("Suficiente");
    }

    public static void bien() {
        System.out.println("Bien");
    }

    public static void notable() {
        System.out.println("Notable");
    }

    public static void sobresaliente() {
        System.out.println("Sobresaliente");
    }
}
```

← El código generado quedaría como el que se aprecia en la imagen, por lo que ya habríamos completado el primero de los 6 tipos de refactorización existentes, por lo que una vez finalizado el refactor procederemos a continuar con los distintos tipos, que en este caso sería el move.

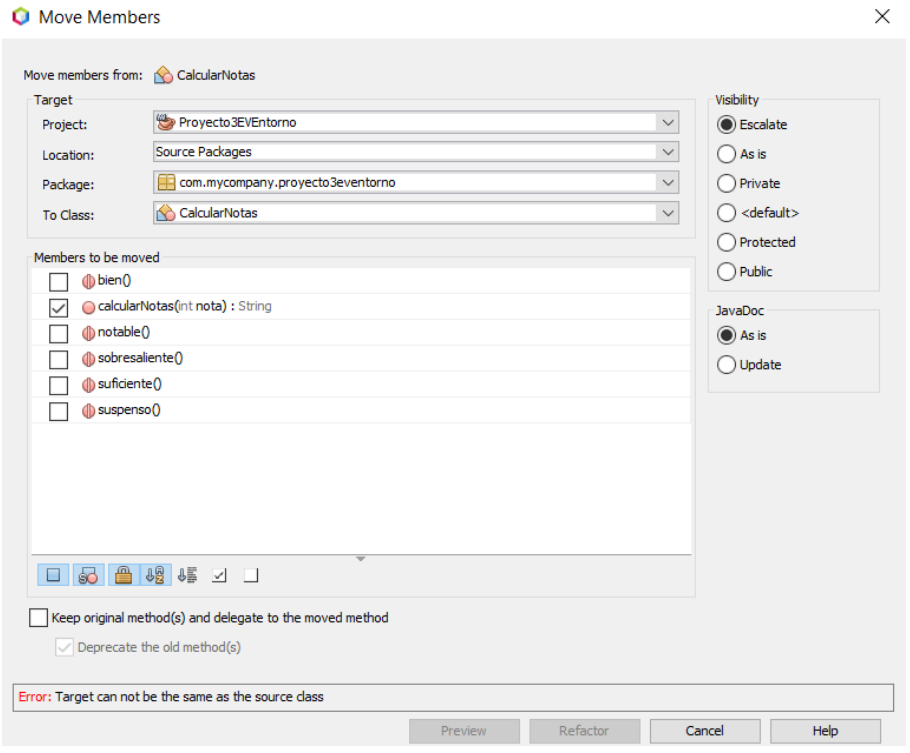
2. Move Method ()

La utilidad de dicho método consiste en la posibilidad de mover ficheros tipo java de un proyecto a otro cambiando las referencias para que no se produzcan problemas entre sus variables.

- En este caso tomaremos el código anteriormente mostrado y realizaremos click derecho sobre la interfaz de netbean y nos aparecerá lo siguiente:

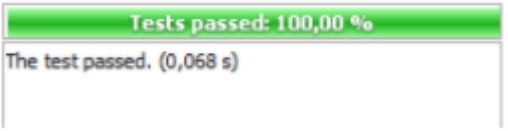


- De este modo comenzaremos con la refactorización de tipo move en nuestro código,por lo que seleccionamos move y se abrirá una pequeña interfaz la cual es de la siguiente forma



- Como se aprecia en la imagen nos permite mover los elementos de una zona a otra, por lo que lo ejecutaremos para comprobar si nos lanza algún tipo de error, hemos de recordar que dicho método cambiaba las referencias dependiendo del proyecto al cual lo queramos trasladar, por lo que hemos de revisar las variables que queramos cambiar antes de realizar el cambio.

- Una vez que comprobamos si todo está correcto, procedemos a realizar el testing del nuevo código generado para comprobar si se ejecuta correctamente, por lo que vamos a proceder a ello.



← Como apreciamos en la imagen el código pasa satisfactoriamente el testing por lo que ya habríamos completado la segunda de las técnicas existentes de la refactorización de código.

3. Extracting Frequent Renter Points Method ()

Este método se encarga de extraer como bien indica su nombre, en el caso que voy a mostrar a continuación realiza una separación de los métodos, es decir a la hora en mi código de realizar las cuentas, por una parte generará un método con los números incluidos por una parte y el valor de dichas operaciones se generará en otro método independiente del citado anteriormente en otro totalmente distinto, realizándose de este modo una extracción.

- A continuación voy a mostrar una captura de cómo quedaría el método una vez realizada la extracción explicada anteriormente.

```
    public int calcular(int total,int descuento,int pago){
        total =250;
        descuento =30;
        pago = total - descuento;
        return pago;

    }
    public String precio(int precio,int pago){
     pago = 200;
     precio = pago;
     return "El valor total es: " + precio;

}
```

4. **Refactor Nº 4**

Esta técnica de refactorización se utiliza para añadir comentarios en el código, para que este sea más intuitivo para la persona la cual vaya a usar dicho código, el objetivo principal de esta técnica se basa principalmente en el diseño.

- A continuación voy a mostrar un ejemplo de cómo sería el uso de dicha técnica de refactorización, la cual quedaría del siguiente modo:

```
// Dicho método nos permitirá calcular el precio a pagar
// Contamos con tres variables
public int calcular(int total,int descuento,int pago){
    //1. Nos indica el precio total a pagar
     total =250;
     //2. Nos indica el dinero el cual se descontará
     // del total
     descuento =30;
     //3. La variable pago aplicará el descuento al total
     pago = total - descuento;
     //4. Devolveremos el total a pagar con desccuento incluido
     return pago;

}
// Dicho método nos devuelve el precio total a pagar
public String precio(int precio,int pago){
 pago = 220;
 precio = pago;
 return "El valor total es: " + precio;
```

5. **Refactor Nº5**

Dicha técnica de refactorización tiene como objetivo también el diseño y que la interfaz sea mucho más organizada e intuitiva, esta se encarga crear clases de manera más eficiente, es decir, que si queremos realizar un una clase principal podemos crear subclases las cuales extienden de la clase principal.

- Una vez realizada una breve explicación sobre la funcionalidad de la quinta técnica de refactorización voy a mostraros un ejemplo de la creación de subclases.

```
public class PokemonFuego extends Pokemon{

    public PokemonFuego() {
        super();
    }

    public PokemonFuego(String apodo, int nivel) {
        super(apodo, nivel);
    }

    public PokemonFuego(int salud, int ataque, int defensa, int veloci
        super(salud, ataque, defensa, velocidad);
    }

    public PokemonFuego(String apodo, int nivel,
            int salud, int ataque, int defensa, int velocidad) {
        super(apodo, nivel, salud, ataque, defensa, velocidad);

    }
}
```

← Como se aprecia en la imagen dicha clase se extiende de una clase la cual es la principal y contiene todos los atributos, en este caso lo que realizamos es extender como he comentado anteriormente de la clase principal, por lo que esta clase proviene de una gran clase y como esta podemos declarar más subclases.

```java
public class Pokemon {
    protected static int nextId = 0;
    private static float[] daños = {0.5f, 1.0f, 2.0f};
    private static String[] mensajesDaños = {"No es muy eficaz...", "Ataque normal...", "Es superefectivo!"};
    protected String apodo;
    protected int salud;
    protected int ataque;
    protected int defensa;
    protected int velocidad;
    protected int nivel;
    protected int id;

    protected static int getNextAvailableId() {
        nextId++;
        return nextId;
    }

    public Pokemon() {
        apodo = "default name";
        id = getNextAvailableId();
    }

    public Pokemon(String apodo, int nivel) {
        this(); // Esto ya llama a getNextAvailableId
```

- Como he citado anteriormente esta sería la clase principal de la que extenderán el resto de subclases.

6. Refactorización Nº6

La última de las refactorizaciones consiste en una clase la cual heredará todo lo que tiene una instancia, de esta forma podremos aplicar los métodos que tiene la instancia y aplicarlos en nuestra clase, hacemos que una clase herede una instancia poniendo “implements” después del nombre de la clase.

```java
package proyecto3ev;

import java.util.concurrent.ThreadLocalRandom;

/**
 *
 * @author antonionav
 */
public class Personaje implements Luchador {
    //Atributos
    protected String apodo;
    protected int salud;
    protected int ataque;
    protected int defensa;
    protected int velocidad;

    //Constructores

    public Personaje(String apodo, int salud, int ataque, int defensa) {
        this.apodo = apodo;
        this.salud = salud;
        this.ataque = ataque;
        this.defensa = defensa;
    }

    public Personaje(String apodo, int salud, int ataque, int defensa, int velocidad) {
        this.apodo = apodo;
```