ALEJANDRO SALGADO CERDEIRA

ENTORNOS DE DESARROLLO 2021

UD.4 – TAREA REFACTORIZACIÓN

INTRODUCCIÓN Y PLANTEAMIENTO:

Lo primero que hacemos es, siguiendo las instrucciones del pdf adjuntado por la profesora, crear las clases Zumo y Main con todos los parámetros requeridos. Adjunto algunas capturas significativas pero los archivos .java también serán entregados junto con este documento.

CLASE ZUMO:

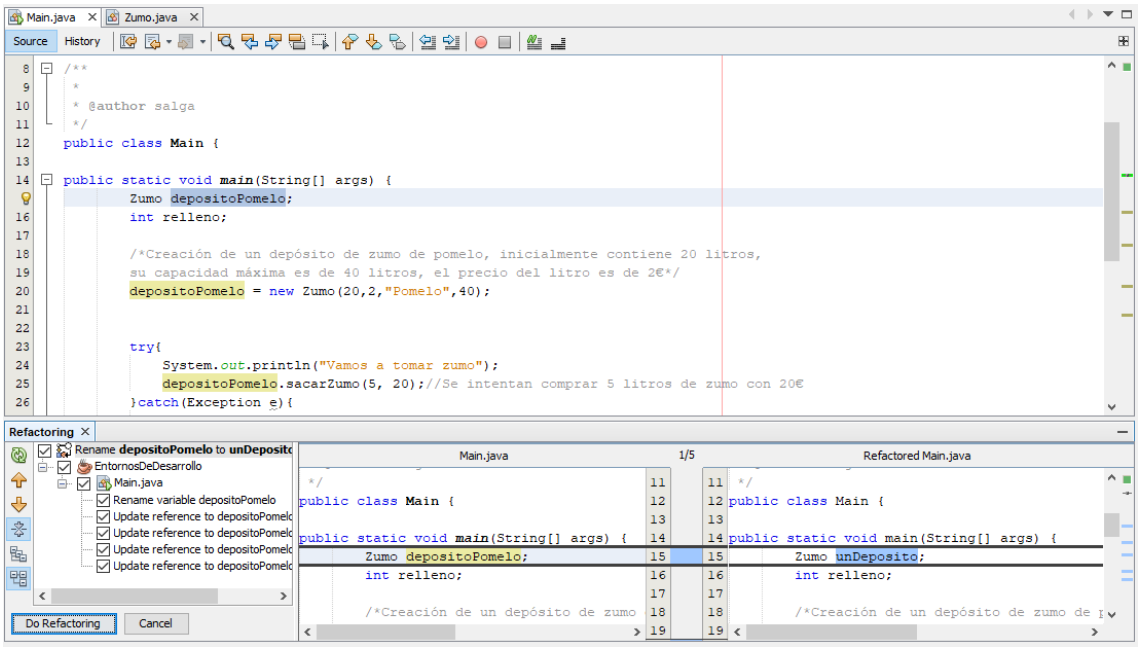
```
12  public class Zumo {
13      private int litros;//contenido actual del deposito de zumo.
14      private int precio_litro;
15      private String sabor;
16      private int litros_max;//capacidad máxima del deposito.
17
18      /*Constructor por defecto*/
19      public Zumo(){
20
21      }
22
23      /*Constructor con todos los atributos*/
24      public Zumo(int litros, int precio_litro, String sabor, int litros_max) {
25          this.litros = litros;
26          this.precio_litro = precio_litro;
27          this.sabor = sabor;
28          this.litros_max = litros_max;
29      }
34      public void rellenar(int litros) throws Exception{
35
36          if((this.litros + litros )> this.litros_max){
37              throw new Exception("No se puede sobrepasar la capacidad del depósito");
38          }
39
40          this.litros=this.litros+litros;
41          }
42
43      /*Método que permite sacar litros del depósito de zumo siempre y cuando
44      el deósito contenga los litros necesarios, y el dinero introducido sea suficiente.*/
45      public void sacarZumo(int litros, int dinero)throws Exception{
46          if(this.litros<litros){
47              throw new Exception("No se puede sacar tanta cantidad del depósito");
48          }
49          if((litros*this.precio_litro)>dinero){
50              throw new Exception("El dinero es insuficiente");
51          }
52          this.litros=this.litros-litros;
53      }
54
55      /*Método que devuelve la cantidad actual de litros del depçosito de zumo.*/
56      public int obtenerLitros(){
57          return this.litros;
58      }
```

CLASE MAIN:

```
10      * @author salga
11      */
12     public class Main {
13         
14     public static void main(String[] args) {
15             Zumo depositoPomelo;
16             int relleno;
17     
18             /*Creación de un depósito de zumo de pomelo, inicialmente contiene 20 litros,
19             su capacidad máxima es de 40 litros, el precio del litro es de 2€*/
20             depositoPomelo = new Zumo(20,2,"Pomelo",40);
21     
22     
23             try{
24                 System.out.println("Vamos a tomar zumo");
25                 depositoPomelo.sacarZumo(5, 20);//Se intentan comprar 5 litros de zumo con 20€
26             }catch(Exception e){
27                 System.out.println("Error al sacar zumo");
28             }
29             try{
30                 System.out.println("Rellenando depósito.");
31                 depositoPomelo.rellenar(30);//Se intenta rellenar el depósito añadiendo 30 litros
32             }catch(Exception e){
33                 System.out.println("Fallo al rellenar el depósito");
34             }
35             relleno=depositoPomelo.obtenerLitros();
36             System.out.println("El depósito contiene "+relleno+" litros.");
37             }
38     }
```

1.- Renombrar la variable depositoPomelo por unDeposito. Realizar una vista previa para comprobar los cambios que se van a efectuar:

Hacemos la preview de los cambios desde Refactor>Rename:

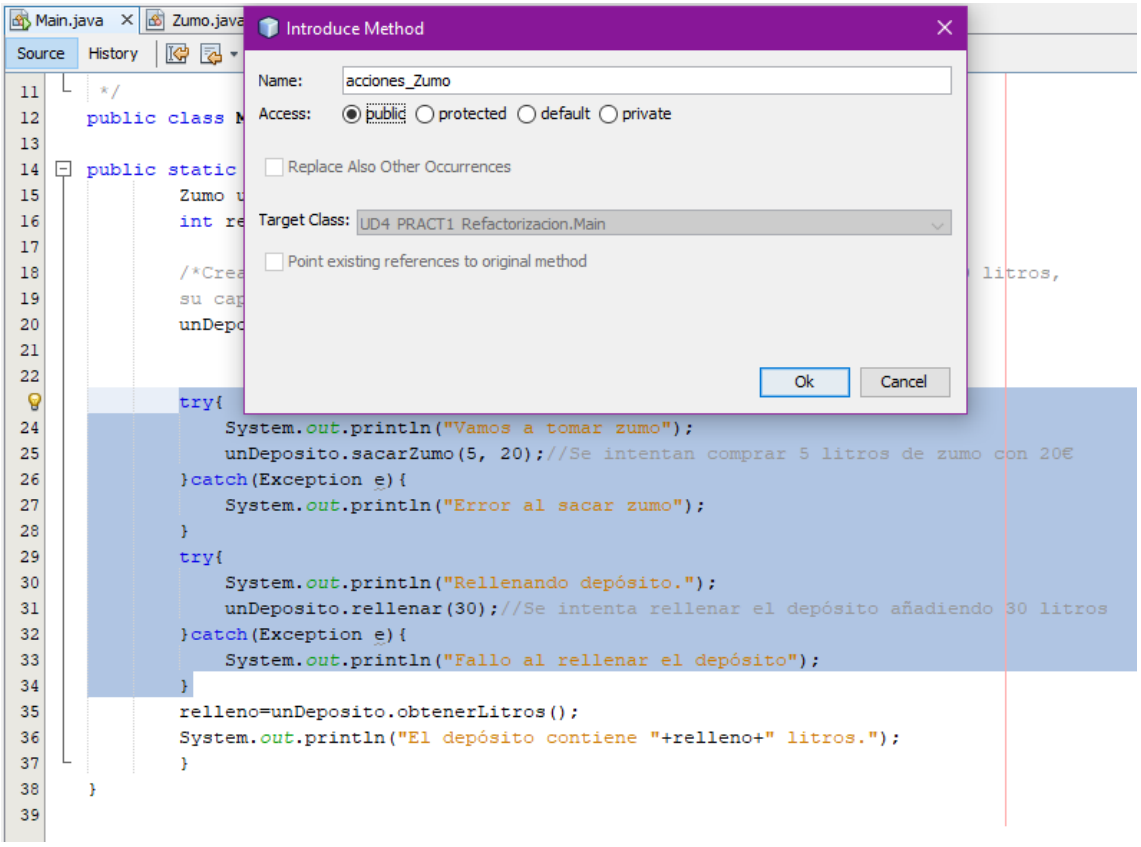


Y aplicamos, siendo este el resultado:

```
14  public static void main(String[] args) {
15          Zumo unDeposito;
16          int relleno;
17
18          /*Creación de un depósito de zumo de pomelo, inicialmente contiene 20 litros,
19          su capacidad máxima es de 40 litros, el precio del litro es de 2€*/
20          unDeposito = new Zumo(20,2,"Pomelo",40);
21
22
23          try{
24              System.out.println("Vamos a tomar zumo");
25              unDeposito.sacarZumo(5, 20);//Se intentan comprar 5 litros de zumo con 20€
26          }catch(Exception e){
27              System.out.println("Error al sacar zumo");
28          }
29          try{
30              System.out.println("Rellenando depósito.");
31              unDeposito.rellenar(30);//Se intenta rellenar el depósito añadiendo 30 litros
32          }catch(Exception e){
33              System.out.println("Fallo al rellenar el depósito");
34          }
35          relleno=unDeposito.obtenerLitros();
36          System.out.println("El depósito contiene "+relleno+" litros.");
37          }
38  }
```

2. - Introducir un método que recoja las acciones sobre el zumo (sacarZumo y rellenar). Llámalo acciones_Zumo. Realizar una vista previa para ver los cambios que se van a efectuar.

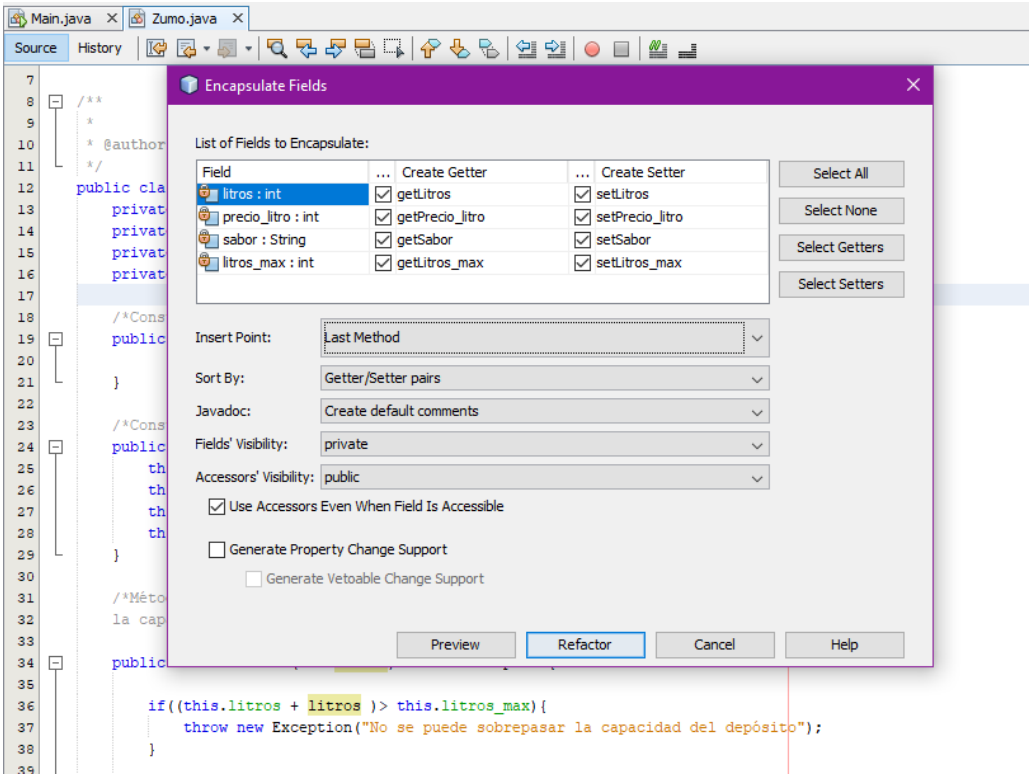
Sobre la clase Main navegamos a Refactor > Introduce > Method



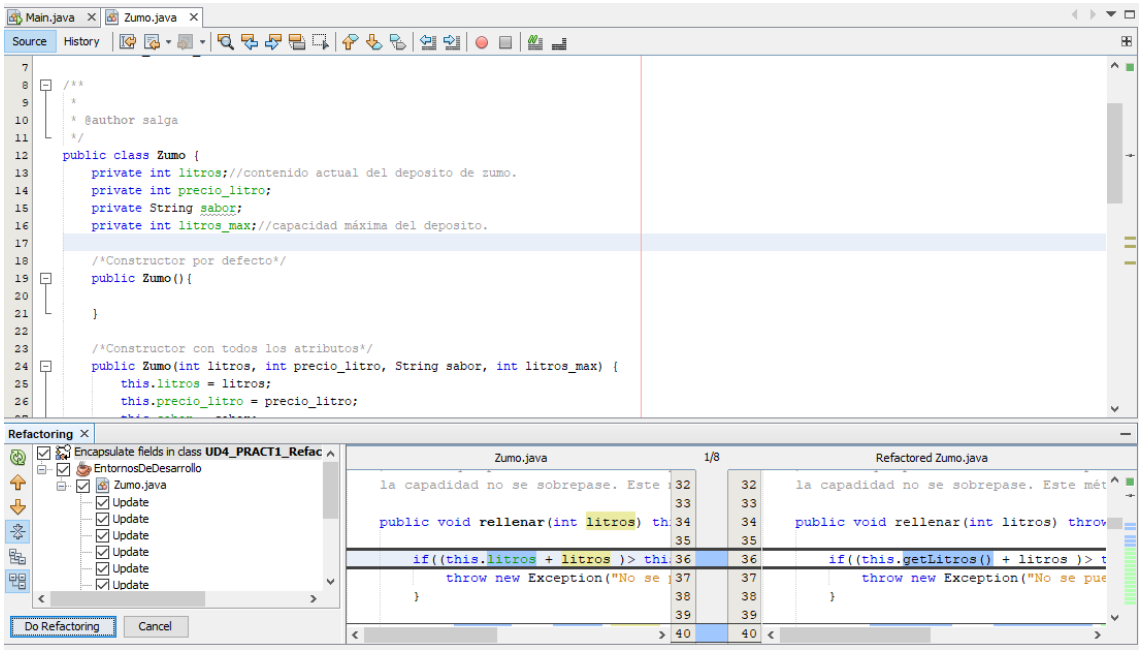
```java
public class Main {

public static void main(String[] args) {
        Zumo unDeposito;
        int relleno;

        /*Creación de un depósito de zumo de pomelo, inicialmente contiene 20 litros,
        su capacidad máxima es de 40 litros, el precio del litro es de 2€*/
        unDeposito = new Zumo(20,2,"Pomelo",40);


        acciones_Zumo(unDeposito);
        relleno=unDeposito.obtenerLitros();
        System.out.println("El depósito contiene "+relleno+" litros.");
        }

public static void acciones_Zumo(Zumo unDeposito) {
        try{
            System.out.println("Vamos a tomar zumo");
            unDeposito.sacarZumo(5, 20);//Se intentan comprar 5 litros de zumo con 20€
        }catch(Exception e){
            System.out.println("Error al sacar zumo");
        }
        try{
            System.out.println("Rellenando depósito.");
            unDeposito.rellenar(30);//Se intenta rellenar el depósito añadiendo 30 litros
        }catch(Exception e){
            System.out.println("Fallo al rellenar el depósito");
        }
}
}
```

3.- Encapsular todos los atributos de la clase Zumo.Realizar una vista previa para ver los cambios que se van a efectuar.

En la clase Zumo: Refactor > Encapsulate Field y seleccionamos las siguientes opciones:



Previsualizamos los cambios y los aceptamos, obteniendo el siguiente resultado.



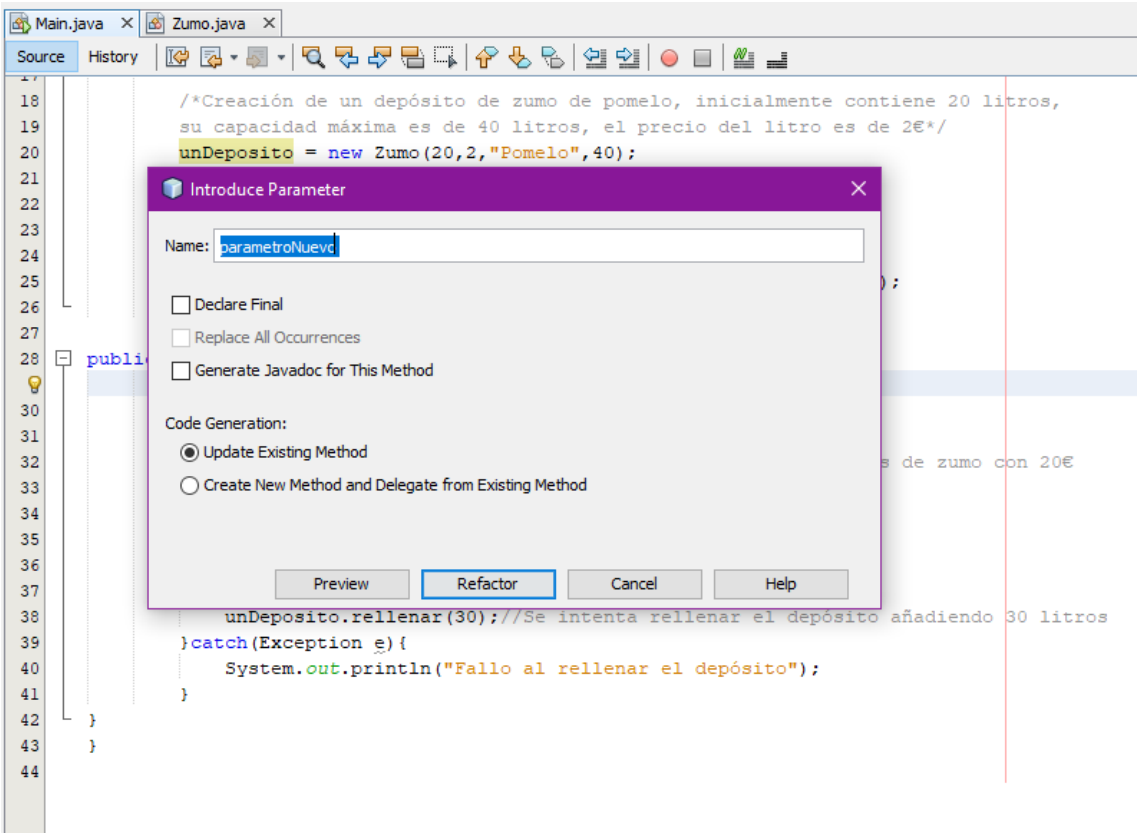

```java
/*Método que permite rellenar el deposito de zumo siempre y cuando
la capadidad no se sobrepase. Este método sera probado con JUnit.*/

public void rellenar(int litros) throws Exception{

    if((this.getLitros() + litros )> this.getLitros_max()){
        throw new Exception("No se puede sobrepasar la capacidad del depósito");
    }

    this.setLitros(this.getLitros() + litros);
    }

/*Método que permite sacar litros del depósito de zumo siempre y cuando
el deósito contenga los litros necesarios, y el dinero introducido sea suficiente.*/
public void sacarZumo(int litros, int dinero)throws Exception{
    if(this.getLitros()<litros){
        throw new Exception("No se puede sacar tanta cantidad del depósito");
    }
    if((litros*this.getPrecio_litro())>dinero){
        throw new Exception("El dinero es insuficiente");
    }
    this.setLitros(this.getLitros() - litros);
}

/*Método que devuelve la cantidad actual de litros del depçosito de zumo.*/
public int obtenerLitros(){
    return this.getLitros();
}
```

4. Añadir un parámetro al método acciones_Zumo. Este parámetro es del tipo doble con valor predeterminado 1.0. Realizar una vista previa para ver los cambios que se van a efectuar.

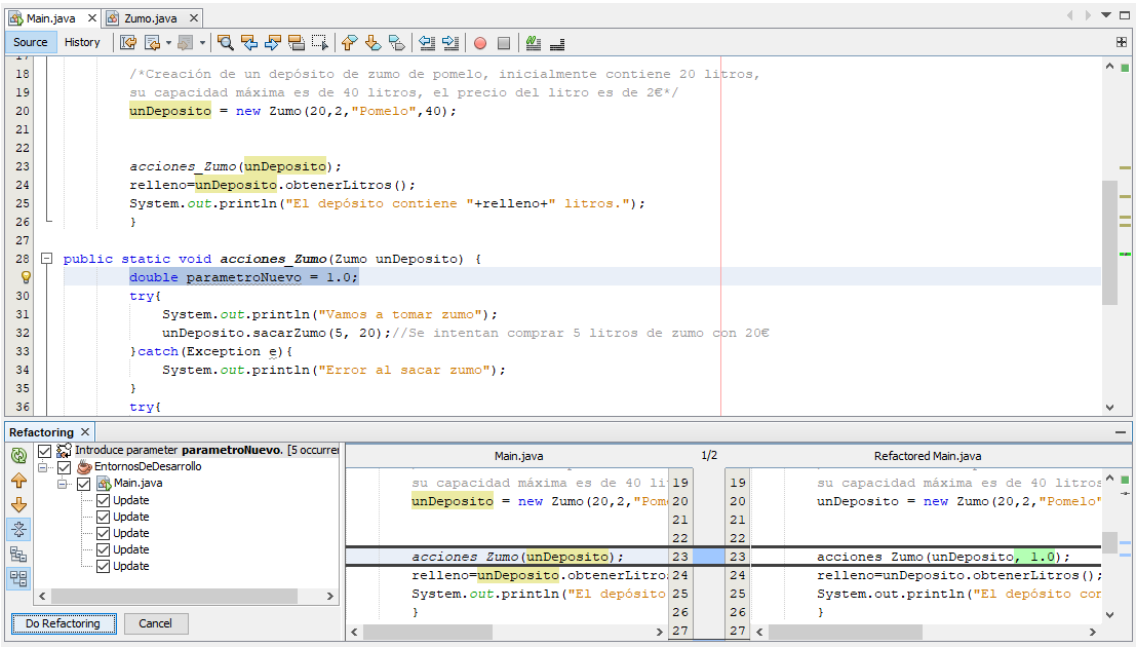
Lo primero es crear la variable en el método:

```
28  public static void acciones_Zumo(Zumo unDeposito) {
29          double parametroNuevo = 1.0;
30          try{
31              System.out.println("Vamos a tomar zumo");
32              unDeposito.sacarZumo(5, 20);//Se intentan comprar 5 litros de zumo con 20€
33          }catch(Exception e){
34              System.out.println("Error al sacar zumo");
35          }
36          try{
37              System.out.println("Rellenando depósito.");
38              unDeposito.rellenar(30);//Se intenta rellenar el depósito añadiendo 30 litros
39          }catch(Exception e){
40              System.out.println("Fallo al rellenar el depósito");
41          }
42      }
43      }
44
```

Después la seleccionamos, y navegamos Refactor > Introduce > Parmeter



Vemos la vista previa y aceptamos:




```
14  public static void main(String[] args) {
15          Zumo unDeposito;
16          int relleno;
17
18          /*Creación de un depósito de zumo de pomelo, inicialmente contiene 20 litros,
19          su capacidad máxima es de 40 litros, el precio del litro es de 2€*/
20          unDeposito = new Zumo(20,2,"Pomelo",40);
21
22
23          acciones_Zumo(unDeposito, 1.0);
24          relleno=unDeposito.obtenerLitros();
25          System.out.println("El depósito contiene "+relleno+" litros.");
26          }
27
28  public static void acciones_Zumo(Zumo unDeposito, double parametroNuevo) {
29          try{
30              System.out.println("Vamos a tomar zumo");
31              unDeposito.sacarZumo(5, 20);//Se intentan comprar 5 litros de zumo con 20€
32          }catch(Exception e){
33              System.out.println("Error al sacar zumo");
34          }
35          try{
36              System.out.println("Rellenando depósito.");
37              unDeposito.rellenar(30);//Se intenta rellenar el depósito añadiendo 30 litros
38          }catch(Exception e){
39              System.out.println("Fallo al rellenar el depósito");
40          }
41  }
42  }
```

Terminando así el ejercicio.