Orientación a Objetos 2

Cuadernillo Semestral de Actividades

* Refactoring -

**Actualizado: 6 de marzo de 2025**

El presente cuadernillo estará en elaboración durante el semestre y tendrá un compilado con todos los ejercicios que se usarán durante la asignatura respecto al tema Refactoring.

**Recomendación importante:**

Los contenidos de la materia se incorporan y fijan mejor cuando uno intenta aplicarlos - no alcanza con ver un ejercicio resuelto por alguien más. Para sacar el máximo provecho de los ejercicios, es importante asistir a las consultas de práctica habiendo intentado resolverlos (tanto como sea posible). De esa manera las consultas estarán más enfocadas y el docente podrá dar un mejor feedback.

{

Refactoring:

En forma iterativa se realizan los siguientes pasos:

* Identificar Bad Smell.
* Identificar el refactoring que lo resuelve.
* Aplicar refactoring.
* Correr los tests.

}

# Ejercicio 1: Algo huele mal

Indique qué malos olores se presentan en los siguientes ejemplos.

## 1.1 Protocolo de Cliente

La clase Cliente tiene el siguiente protocolo. ¿Cómo puede mejorarlo?

/\*\*

\* Retorna el límite de crédito del cliente

\*/

**public** **double** **lmtCrdt**() {...

/\*\*

\* Retorna el monto facturado al cliente desde la fecha f1 a la fecha f2

\*/

**protected** **double** **mtFcE**(LocalDate f1, LocalDate f2) {...

/\*\*

\* Retorna el monto cobrado al cliente desde la fecha f1 a la fecha f2

\*/

**private double** **mtCbE**(LocalDate f1, LocalDate f2) {...

RTA {

Nombres de métodos poco descriptivos, aplicar:

* Rename method.
* Modificar cada llamado a las funciones a las cuales fue aplicado Rename Mehod.

}

## 1.2 Participación en proyectos

Al revisar el siguiente diseño inicial (Figura 1), se decidió realizar un cambio para evitar lo que se consideraba un mal olor. El diseño modificado se muestra en la Figura 2. Indique qué tipo de cambio se realizó y si lo considera apropiado. Justifique su respuesta.

**Diseño inicial:**

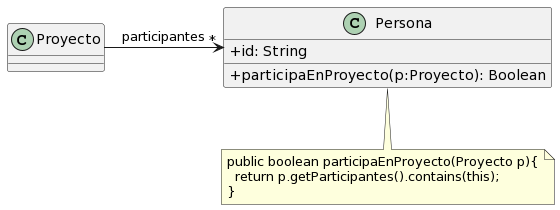


Figura 1: Diagrama de clases del diseño inicial.

**Diseño revisado:**

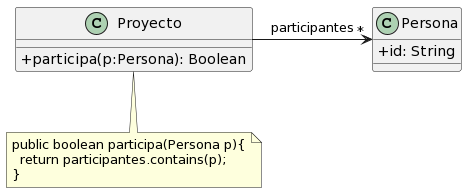


Figura 2: Diagrama de clases modificado.

## 

## RTA {

Envidia de atributos, aplicar:

Mecánica:

1. Revisar las variables de instancia usadas por el método a mover. Tiene sentido moverlas también?

2. Revisar super y subclases por otras declaraciones del método. Si hay otras tal vez no se pueda mover.

3. Crear un nuevo método en la clase target cuyo nombre explique su propósito

4. Copiar el código a mover al nuevo método. Ajustar lo que haga falta

5. Compilar la clase target

6. Determinar como referenciar al target desde el source

7. Reemplazar el método original por llamada a método en target

8. Compilar y testear

9. Decidir si remover el método original o mantenerlo como delegación

}

## 1.3 Cálculos

Analice el código que se muestra a continuación. Indique qué *code smells* encuentra y cómo pueden corregirse.

**public** **void** **imprimirValores**() {

**int** totalEdades = **0**;

**double** promedioEdades = **0**;

**double** totalSalarios = **0**;

**for** (Empleado empleado : personal) {

totalEdades = totalEdades + empleado.getEdad();

totalSalarios = totalSalarios + empleado.getSalario();

}

promedioEdades = totalEdades / personal.size();

String message = String.format("El promedio de las edades es %s y el total de salarios es %s", promedioEdades, totalSalarios);

System.out.println(message);

}

RTA {

Variable temporal, metodo largo y reinventar la rueda, aplicar Replace temp with query, Extract Method y Replace loop with pipeline:

**Replace temp with query:**

1. Extraer la expresión en un método

2. Remplazar TODAS las referencias a la var. temporal por la expresión

3. El nuevo método luego puede ser usado en otros métodos

**Extract Method:**

1. Crear un nuevo método cuyo nombre explique su propósito

2. Copiar el código a extraer al nuevo método

3. Revisar las variables locales del original

4. Si alguna se usa sólo en el código extraído, mover su

declaración

5. Revisar si alguna variable local es modificada por el código

extraido. Si es solo una, tratar como query y asignar. Si hay

más de una no se puede extraer.

6. Pasar como parámetro las variables que el método nuevo lee.

7. Compilar

8. Reemplazar código en método original por llamada

9. Compilar

}

# Ejercicio 2

Para cada una de las siguientes situaciones, realice en forma iterativa los siguientes pasos:

(i) indique el mal olor,

(ii) indique el refactoring que lo corrige,

(iii) aplique el refactoring, mostrando el resultado final (código y/o diseño según corresponda).

Si vuelve a encontrar un mal olor, retorne al paso (i).

## 2.1 Empleados

**public** **class** **EmpleadoTemporario** {

**public** String nombre;

**public** String apellido;

**public** **double** sueldoBasico = **0**;

**public** **double** horasTrabajadas = **0**;

**public** **int** cantidadHijos = **0**;

// ......

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico

* (**this**.horasTrabajadas \* **500**)
* (**this**.cantidadHijos \* **1000**)
* (**this**.sueldoBasico \* **0.13**);

}

}

**public** **class** **EmpleadoPlanta** {

**public** String nombre;

**public** String apellido;

**public** **double** sueldoBasico = **0**;

**public** **int** cantidadHijos = **0**;

// ......

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico

+ (**this**.cantidadHijos \* **2000**)

- (**this**.sueldoBasico \* **0.13**);

}

}

**public** **class** **EmpleadoPasante** {

**public** String nombre;

**public** String apellido;

**public** **double** sueldoBasico = **0**;

// ......

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico - (**this**.sueldoBasico \* **0.13**);

}

RTA {  
Código duplicado, aplicar Pull Up Field:

**Pull Up Field:**

1. Asegúrese de que los campos se utilicen para las mismas necesidades en las subclases.
2. Si los campos tienen nombres diferentes, asígneles el mismo nombre y reemplace todas las referencias a los campos en el código existente.
3. Cree un campo con el mismo nombre en la superclase. Tenga en cuenta que si los campos fueran privados, el campo de la superclase debería estar protegido.
4. Eliminar los campos de las subclases.
5. Es posible que desee considerar el uso de [Self](https://refactoring.guru/es/self-encapsulate-field) encapsulated field para el nuevo campo, a fin de ocultarlo detrás de los métodos de acceso.

**public** **class** **Empleado** {

**public** String nombre;

**public** String apellido;

**public** **double** sueldoBasico = **0**;

**public** **double** horasTrabajadas = **0**;

**public** **int** cantidadHijos = **0**;

**public** abstract **double** **sueldo**();

}

**public** **class** **EmpleadoTemporario** extends **Empleado** {

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico

* (**this**.horasTrabajadas \* **500**)
* (**this**.cantidadHijos \* **1000**)
* (**this**.sueldoBasico \* **0.13**);

}

}

**public** **class** **EmpleadoPlanta** extends **Empleado** {

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico

+ (**this**.cantidadHijos \* **2000**)

- (**this**.sueldoBasico \* **0.13**);

}

}

**public** **class** **EmpleadoPasante** extends **Empleado** {

**public** **double** **sueldo**() {

**return** **this**.sueldoBasico - (**this**.sueldoBasico \* **0.13**);

}

}

## 2.2 Juego

**public class Juego** {

// ......

**public** void incrementar(Jugador j) {

j.puntuacion = j.puntuacion + 100;

}

**public** void decrementar(Jugador j) {

j.puntuacion = j.puntuacion - 50;

}

**public class Jugador** {

**public** String nombre;

**public** String apellido;

**public int** puntuacion = 0;

}

}

RTA {

Envidia de atributos, aplicar Move Method:

**Move Method:**

1. Revisar las v.i. usadas por el método a mover. Tiene sentido moverlas también?

2. Revisar super y subclases por otras declaraciones del método. Si hay otras tal vez no se pueda mover.

3. Crear un nuevo método en la clase target cuyo nombre explique su propósito

4. Copiar el código a mover al nuevo método. Ajustar lo que haga falta

5. Compilar la clase target

6. Determinar como referenciar al target desde el source

7. Reemplazar el método original por llamada a método en target

8. Compilar y testear

9. Decidir si remover el método original o mantenerlo como delegación

**public class Juego** {

// ......

**public** void incrementar(Jugador j) {

j.incrementar();

}

**public** void decrementar(Jugador j) {

j.decrementar();

}

**public class Jugador** {

**public** String nombre;

**public** String apellido;

**public int** puntuacion = 0;

**public** void incrementar() {

this.puntuacion = this.puntuacion + 100;

}

**public** void decrementar(Jugador j) {

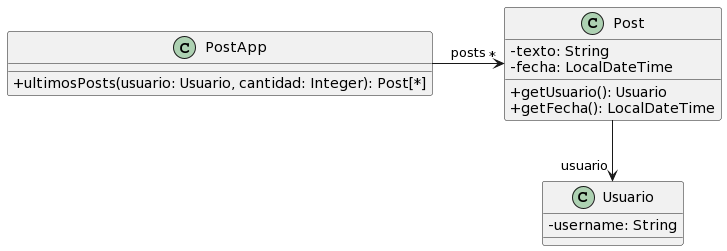
this.puntuacion = this.puntuacion - 50;

}

}

}

## 2.3 Publicaciones



/\*\*

\* Retorna los últimos N posts que no pertenecen al usuario user

\*/

**public** List<Post> **ultimosPosts**(Usuario user, **int** cantidad) {

List<Post> postsOtrosUsuarios = **new** ArrayList<Post>();

**for** (Post post : **this**.posts) {

**if** (!post.getUsuario().equals(user)) {

postsOtrosUsuarios.add(post);

}

}

// ordena los posts por fecha

**for** (**int** i = **0**; i < postsOtrosUsuarios.size(); i++) {

**int** masNuevo = i;

**for**(**int** j= i +**1**; j < postsOtrosUsuarios.size(); j++) {

**if** (postsOtrosUsuarios.get(j).getFecha().isAfter(

postsOtrosUsuarios.get(masNuevo).getFecha())) {

masNuevo = j;

}

}

Post unPost = postsOtrosUsuarios.set(i,postsOtrosUsuarios.get(masNuevo));

postsOtrosUsuarios.set(masNuevo, unPost);

}

List<Post> ultimosPosts = **new** ArrayList<Post>();

**int** index = **0**;

Iterator<Post> postIterator = postsOtrosUsuarios.iterator();

**while** (postIterator.hasNext() && index < cantidad) {

ultimosPosts.add(postIterator.next());

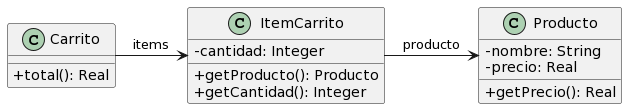
}

**return** ultimosPosts;

}

RTA {  
Reinventa la rueda, método largo, aplicar Replace Loop with pipeline x3 y Extract Method x3. También las variables deberían ser privadas.  
}

## 2.4 Carrito de compras



**public** **class** **Producto** {

**private** String nombre;

**private** **double** precio;

**public** **double** **getPrecio**() {

**return** **this**.precio;

}

}

**public** **class** **ItemCarrito** {

**private** Producto producto;

**private** **int** cantidad;

**public** Producto **getProducto**() {

**return** **this**.producto;

}

**public** **int** **getCantidad**() {

**return** **this**.cantidad;

}

}

**public** **class** **Carrito** {

**private** List<ItemCarrito> items;

**public** **double** **total**() {

**return** **this**.items.stream()

.mapToDouble(item ->

item.getProducto().getPrecio() \* item.getCantidad())

.sum();

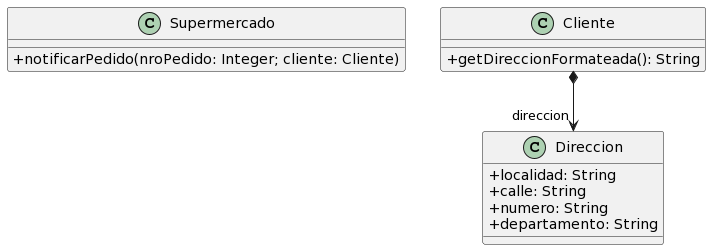
}

}

RTA {  
Envidia de atributo, aplicar Move Method (Consultar porque no se debe mover el método completo sino que parte de el).

Item Carrito debería tener un método getPrecio() donde calcule el su precio y desde Carrito iterar sumando el precio de cada ItemCarrito mandando el mensaje ItemCarrito.getPrecio().  
}

## 2.5 Envío de pedidos



**public** **class Supermercado** {

public **void notificarPedido**(**long** nroPedido, **Cliente** cliente) {

**String** notificacion = MessageFormat.format(“Estimado cliente, se le informa que hemos recibido su pedido con número {0}, el cual será enviado a la dirección {1}”, new **Object**[] { nroPedido, cliente.getDireccionFormateada() });

// lo imprimimos en pantalla, podría ser un mail, SMS, etc..

**System**.out.println(notificacion);

}

}

**public** **class Cliente** {

public **String getDireccionFormateada**() {

return

this.direccion.getLocalidad() + “, ” +

this.direccion.getCalle() + “, ” +

this.direccion.getNumero() + “, ” +

this.direccion.getDepartamento()

;

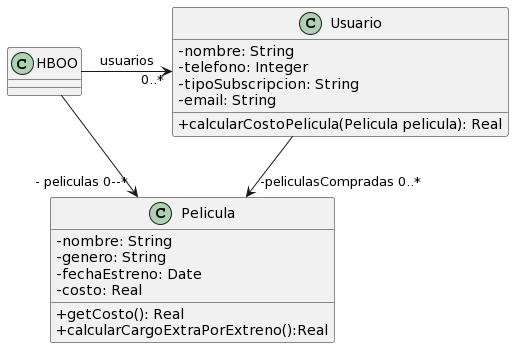
}

## RTA {

## Crear una clase Direccion, y componerla en Cliente. Usando Move Method.

## }

## 2.6 Películas



**public** **class Usuario** {

String tipoSubscripcion;

// ...

public **void** **setTipoSubscripcion**(**String** unTipo) {

this.tipoSubscripcion = unTipo;

}

public **double** **calcularCostoPelicula**(**Pelicula** pelicula){

double costo = 0;

if (tipoSubscripcion=="Basico") {

costo = pelicula.getCosto() + pelicula.calcularCargoExtraPorEstreno();

}

else if (tipoSubscripcion== "Familia") {

costo = (pelicula.getCosto() + pelicula.calcularCargoExtraPorEstreno()) \* 0.90;

}

else if (tipoSubscripcion=="Plus") {

costo = pelicula.getCosto();

}

else if (tipoSubscripcion=="Premium") {

costo = pelicula.getCosto() \* 0.75;

}

return costo;

}

}

**public class Pelicula** {

LocalDate fechaEstreno;

// ...

public **double** **getCosto**() {

return this.costo;

}

public **double** **calcularCargoExtraPorEstreno**(){

// Si la Película se estrenó 30 días antes de la fecha actual, retorna un cargo de 0$, caso contrario, retorna un cargo extra de 300$

return (ChronoUnit.DAYS.between(this.fechaEstreno, LocalDate.now()) ) > 30 ? 0 : 300;

}

}

RTA {

Eliminar condicional en Usuario por medio de polimorfismo.

}