FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA

Orientación a Objetos 2
Cuadernillo Semestral de Actividades
- Refactoring -

Actualizado: 12 de mayo 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.

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 li´mite de cre´dito 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) {...
```

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:

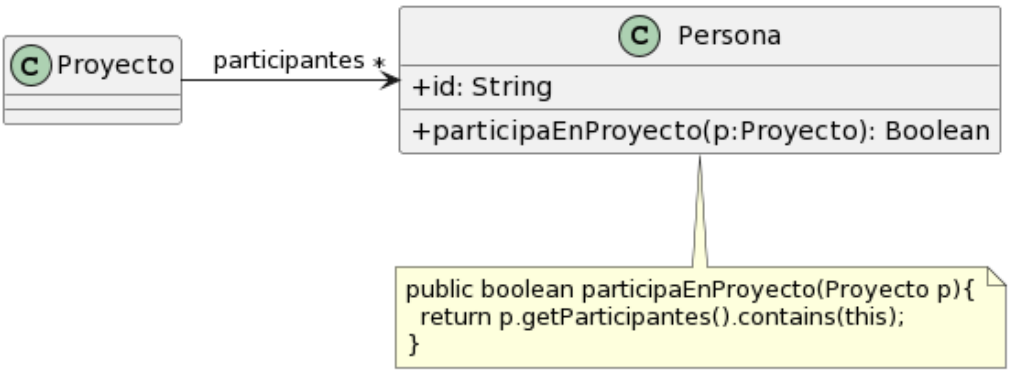


Figura 1: Diagrama de clases del diseño inicial.

Diseño revisado:

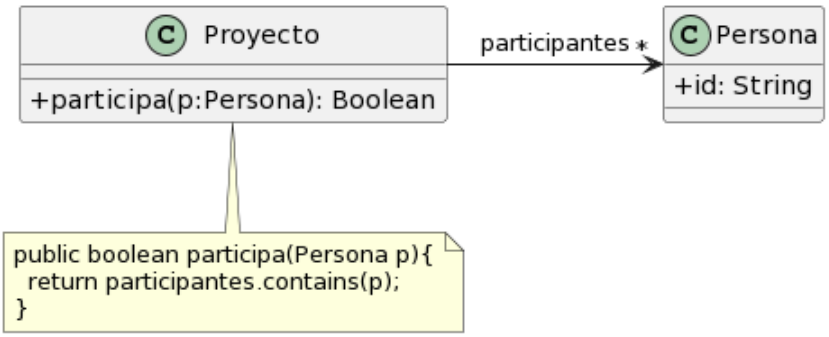


Figura 2: Diagrama de clases modificado.

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);
}
```

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);
    }
}
```

```java
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);
    }
}
```

2.2 Juego

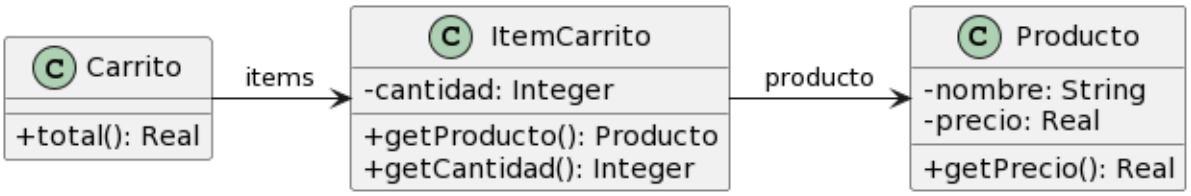
```java
public class Juego {
    // ......
    public void incrementar(Jugador j) {
        j.puntuacion = j.puntuacion + 100;
    }
    public void decrementar(Jugador j) {
        j.puntuacion = j.puntuacion - 50;
    }
}
```

```java
public class Jugador {
    public String nombre;
    public String apellido;
    public int puntuacion = 0;
}
```

2.3 Publicaciones

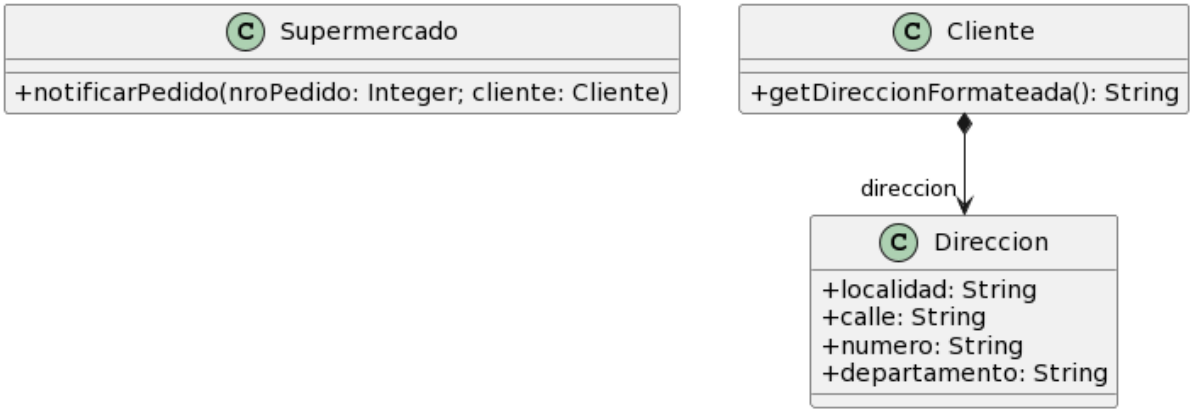

```java
/**
* Retorna los últimos N posts que no pertenecen al usuario user
*/
public List<Post> ultimosPosts(Usuario user, int cantidad) {

    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;
}
```

2.4 Carrito de compras



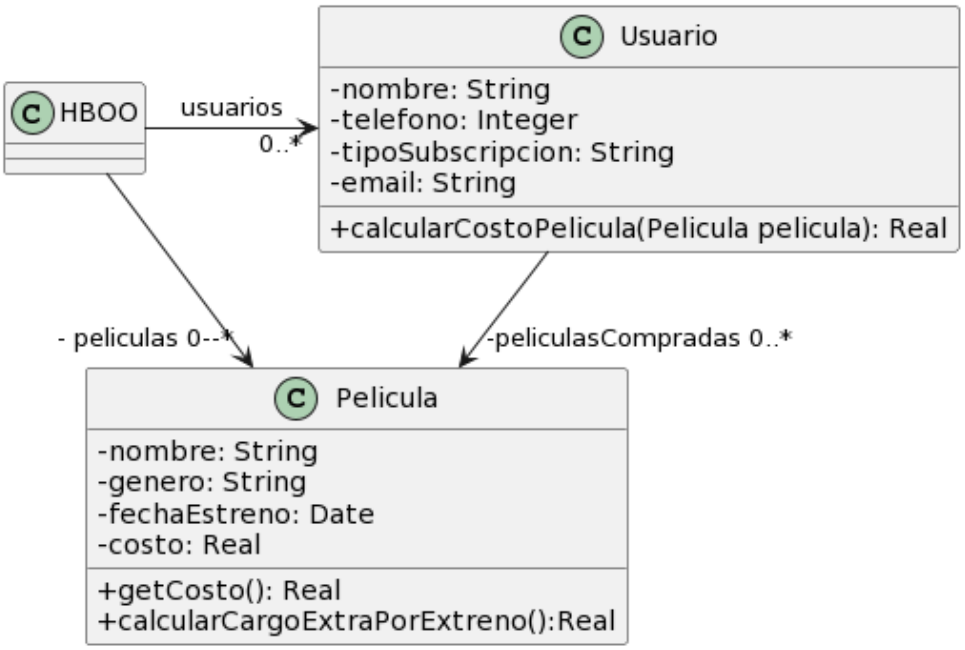

```
public class Producto {
    private String nombre;
    private double precio;

    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();
    }
}
```

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()
     ;
}
```

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;
     }
}
```

Ejercicio 3 - Documentos y estadísticas

Dado el siguiente código implementado en la clase Document y que calcula algunas estadísticas del mismo:

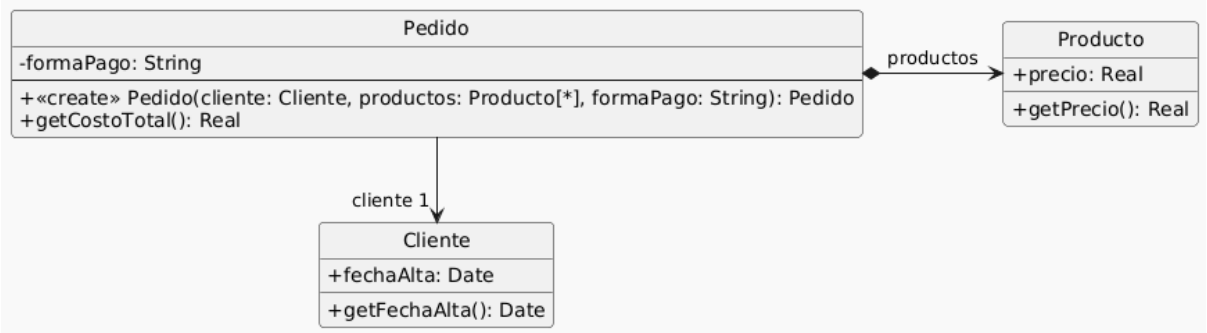
```
public class Document {
   List<String> words;

   public long characterCount() {
         long count = this.words
                  .stream()
                  .mapToLong(w -> w.length())
                  .sum();
         return count;
         }
   public long calculateAvg() {
         long avgLength = this.words
                  .stream()
                  .mapToLong(w -> w.length())
                  .sum() / this.words.size();
         return avgLength;
         }
// Resto del código que no importa
}
```

Tareas:

1. Enumere los code smell y que refactorings utilizará para solucionarlos.
2. Aplique los refactorings encontrados, mostrando el código refactorizado luego de aplicar cada uno.
3. Analice el código original y detecte si existe un problema al calcular las estadísticas. Explique cuál es el error y en qué casos se da ¿El error identificado sigue presente luego de realizar los refactorings? En caso de que no esté presente, ¿en qué momento se resolvió? De acuerdo a lo visto en la teoría, ¿podemos considerar esto un refactoring?

Ejercicio 4 - Pedidos

Se tiene el siguiente modelo de un sistema de pedidos y la correspondiente implementación.




```
01: public class Pedido {
02:  private Cliente cliente;
03:  private List<Producto> productos;
04:  private String formaPago;
05:  public Pedido(Cliente cliente, List<Producto> productos, String formaPago)
{
06:     if (!"efectivo".equals(formaPago)
07:        && !"6 cuotas".equals(formaPago)
08:        && !"12 cuotas".equals(formaPago)) {
09:          throw new Error("Forma de pago incorrecta");
10:    }
11:    this.cliente = cliente;
12:    this.productos = productos;
13:    this.formaPago = formaPago;
14:   }
15:   public double getCostoTotal() {
16:     double costoProductos = 0;
17:     for (Producto producto : this.productos) {
18:       costoProductos += producto.getPrecio();
19:     }
20:     double extraFormaPago = 0;
21:     if ("efectivo".equals(this.formaPago)) {
22:       extraFormaPago = 0;
23:     } else if ("6 cuotas".equals(this.formaPago)) {
24:       extraFormaPago = costoProductos * 0.2;
25:     } else if ("12 cuotas".equals(this.formaPago)) {
26:       extraFormaPago = costoProductos * 0.5;
27:     }
28:     int añosDesdeFechaAlta = Period.between(this.cliente.getFechaAlta(),
LocalDate.now()).getYears();
29:     // Aplicar descuento del 10% si el cliente tiene más de 5 años de
antiguedad
30:     if (añosDesdeFechaAlta > 5) {
31:       return (costoProductos + extraFormaPago) * 0.9;
32:     }
33:     return costoProductos + extraFormaPago;
34:   }
```

```
35: }
36: public class Cliente {
37:   private LocalDate fechaAlta;
38:   public LocalDate getFechaAlta() {
39:     return this.fechaAlta;
40:   }
41: }
42: public class Producto {
43:   private double precio;
44:   public double getPrecio() {
45:     return this.precio;
46:   }
47: }
```

Tareas:

1. Dado el código anterior, aplique **únicamente** los siguientes refactoring:
    - Replace Loop with Pipeline (líneas 16 a 19)
    - Replace Conditional with Polymorphism (líneas 21 a 27)
    - Extract method y move method (línea 28)
    - Extract method y replace temp with query (líneas 28 a 33)
2. Realice el diagrama de clases del código refactorizado.

Ejercicio 5 - Facturación de llamadas

En el material adicional encontrará una aplicación que registra y factura llamadas telefónicas. Para lograr tal objetivo, la aplicación permite administrar números telefónicos, como así también clientes asociados a un número. Los clientes pueden ser personas físicas o jurídicas. Además, el sistema permite registrar las llamadas realizadas, las cuales pueden ser nacionales o internacionales. Luego, a partir de las llamadas, la aplicación realiza la facturación, es decir, calcula el monto que debe abonar cada cliente.

Importe el material adicional provisto por la cátedra y analícelo para identificar y corregir los malos olores que presenta. En forma iterativa, realice los siguientes pasos:

(i) indique el mal olor,
(ii) indique el refactoring que lo corrige,
(iii) aplique el refactoring (modifique el código).
(iv) asegúrese de que los tests provistos corran exitosamente.

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

Tareas:

- Describa la solución inicial con un diagrama de clases UML.
- Documente la secuencia de refactorings aplicados, como se indica previamente.
- Describa la solución final con un diagrama de clases UML.

Ejercicio 6 - Árboles binarios

Un **árbol binario** es una estructura de datos en la que cada nodo puede tener como máximo dos hijos: uno izquierdo y uno derecho. Es común utilizar esta estructura para representar jerarquías o realizar operaciones de búsqueda, recorrido y ordenamiento.

El código correspondiente a la implementación de un árbol binario se encuentra en el material adicional. En varias partes del código se realizan verificaciones repetitivas para comprobar si el árbol tiene hijo izquierdo o derecho antes de realizar los recorridos.

Tareas:

1. Describa la solución inicial con un diagrama de clases UML.
2. Refactorice la implementación para evitar estas verificaciones repetidas, explicando los pasos realizados.
3. Describa la solución final con un diagrama de clases UML.