Orientación a Objetos 2
Cuadernillo Semestral de Actividades
- Refactoring -

Actualizado: 10 de mayo de 2023

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
*/
protected 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
*/
protected 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:

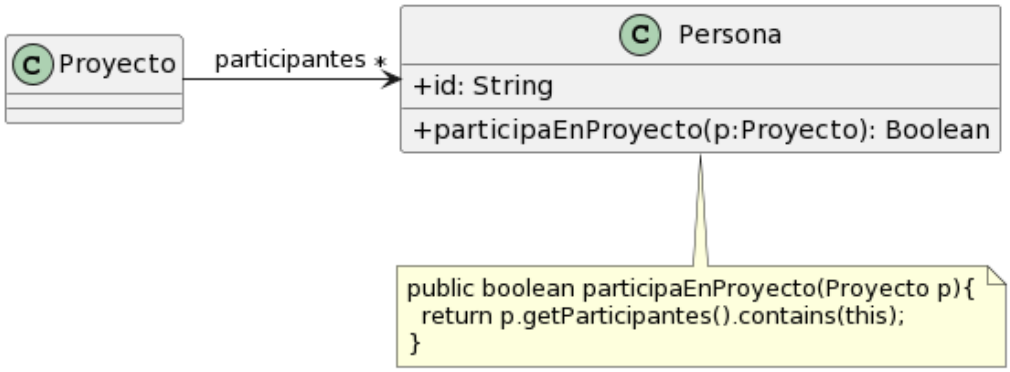


Figura 1: Diagrama de clases del diseño inicial.

Diseño revisado:

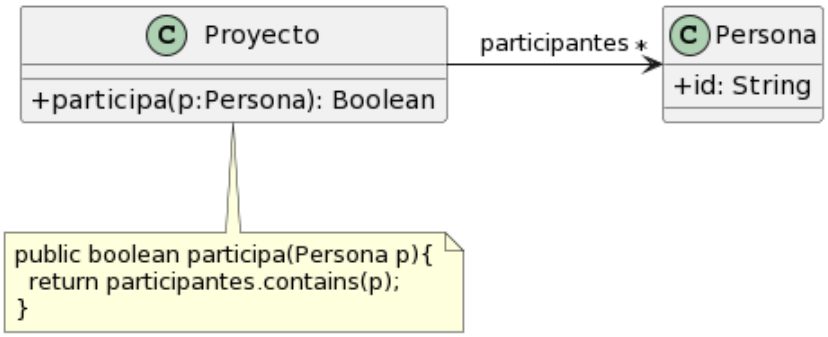


Figura 2: Diagrama de clases modificado.

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;
    // ......
```

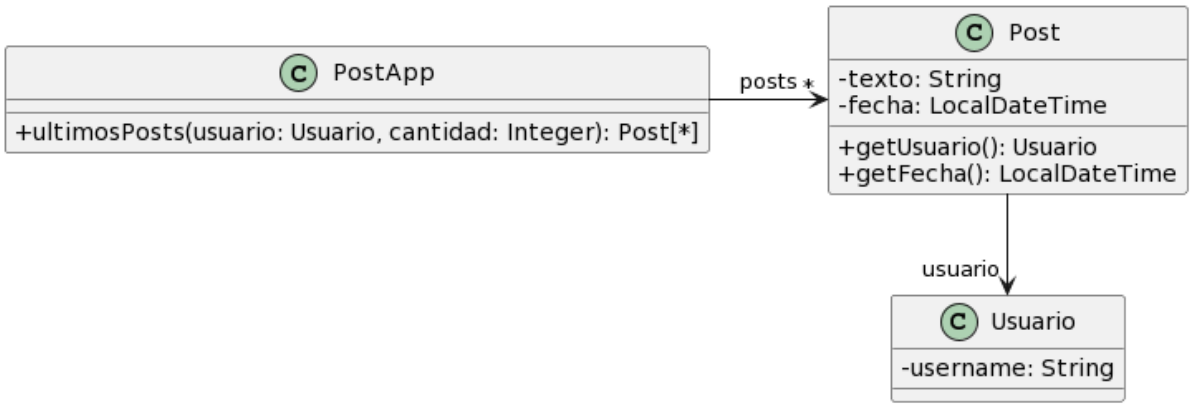
```
    public double sueldo() {
          return this.sueldoBasico
                  +  (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000)
                  -  (this.sueldoBasico * 0.13);
              }
          }

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

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

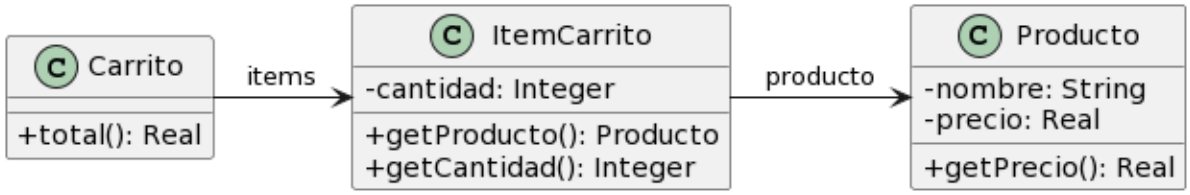
```
public class Jugador {
    public String nombre;
    public String apellido;
    public int puntuacion = 0;
}
```

2.3 Publicaciones



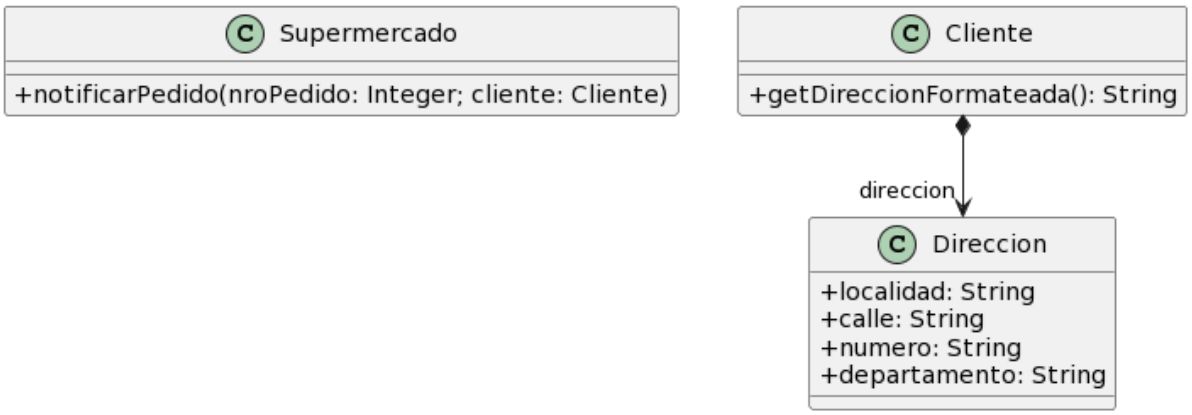

```java
/**
* Retorna los últimos N posts que no pertenecen al usuario user
*/
public List<Post> ultimosPosts(Usuario user, int cantidad) {

    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



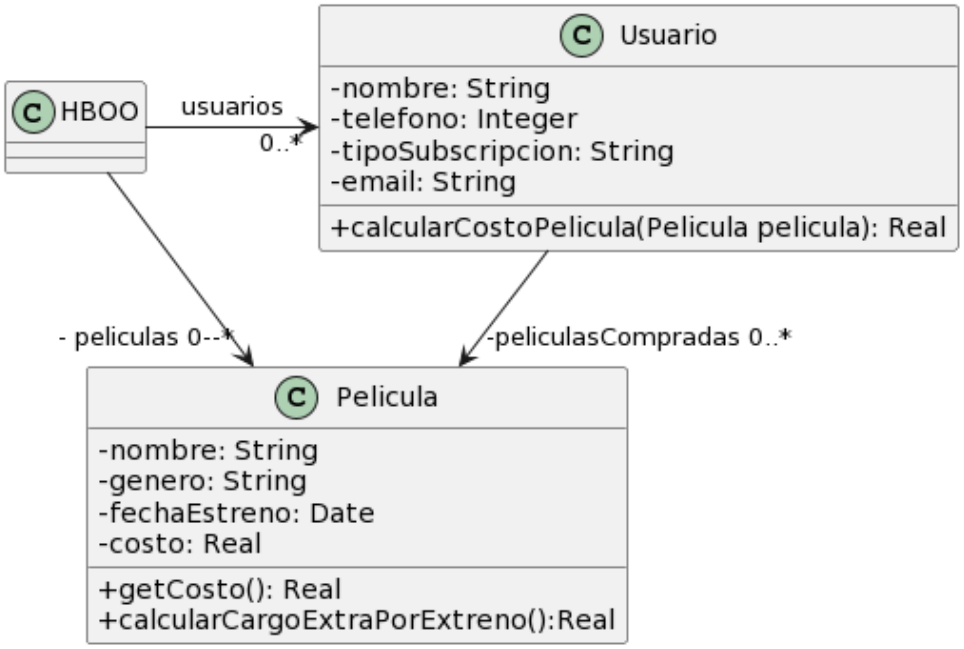

```
public class Producto {
    private String nombre;
    private double precio;

    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 - Facturación de llamadas

Importante: *Aprobando este ejercicio, no será necesario rendir el tema Refactoring en el parcial.*
Fecha límite de entrega: 24/05/2023 23:59 hs (máximo 2 integrantes por grupo).

El material adicional se corresponde con una aplicación de registro y facturación de 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).

Ud debe entregar:

- Un diagrama de clases UML con el diseño inicial de la solución provista
- La secuencia de refactorings aplicados, documentados cada uno de la siguiente manera:
    - Mal olor detectado en el código
    - Extracto del código que presenta el mal olor
    - Refactoring a aplicar que resuelve el mal olor
    - Código con el refactoring aplicado
- Un diagrama de clases UML con el diseño final
- El código java refactorizado

Importante: asegúrese de que no queden malos olores por identificar y refactorizar.