FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA

Orientación a Objetos 2

Práctica 5 - Parte I

Fecha de última edición: 26 abril 2022

Ejercicio 1: SpOOtify

SpOOtify es, obviamente, la versión orientada a objetos de Spotify. En Spotify cada usuario puede buscar obras musicales de diferentes maneras: por autor, por género, por album. Los usuarios pueden agregar las canciones a una lista privada conocida como My Music. Obviamente también puede remover temas de My Music. De cada autor se conocen sus álbumes y de cada álbum sus temas. SpOOtify conoce a los autores y a partir de allí, a todos los temas. SpOOtify conoce a los usuarios. Se pide implementar los mensajes necesarios para poder ofrecer la siguiente funcionalidad:

- **buscar por título**: dado un String retorna una lista de temas cuyo título contiene o es igual al String pasado como parámetro. La búsqueda es *case insensitive*.
- **buscar por autor:** dado un String retorna una lista con los temas cuyo nombre de autor contiene o es igual al String recibido como parámetro. La búsqueda es *case insensitive*.
- **buscar por álbum**: dado un String retorna una lista con los temas cuyo título de álbum contiene o es igual al String recibido como parámetro. La búsqueda es *case insensitive*.
- Además, el usuario debe ser capaz de agregar y quitar temas de su lista privada de música favorita.

Tareas:

1. Realice el diseño usando un diagrama de clases UML. Verifique el diseño con un ayudante.
2. Planifique qué tipo de pruebas desea hacer para los mensajes propuestos en el enunciado.
3. Implemente los casos de prueba en JUnit.
4. Implemente en Java el modelo de SpOOtify.
5. Ejecute los test cases.

Ejercicio 2: Algo huele mal

Indique qué malos olores se presentan en los siguientes ejemplos.

2.1 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) {...
```

2.2 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:

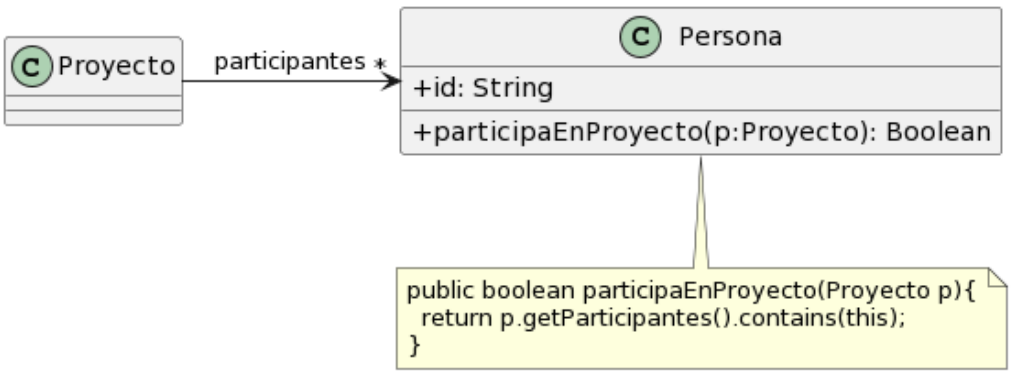


Figura 1: Diagrama de clases del diseño inicial.

Diseño revisado:

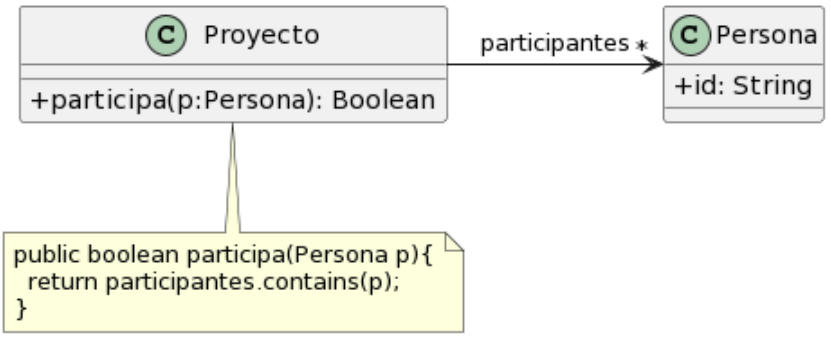


Figura 2: Diagrama de clases modificado.

2.3 Cálculos

Analice el código que se muestra a continuación. Indique qué defectos 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 3

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).

3.1 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);
```

```
    }
}
```

3.2 Juego

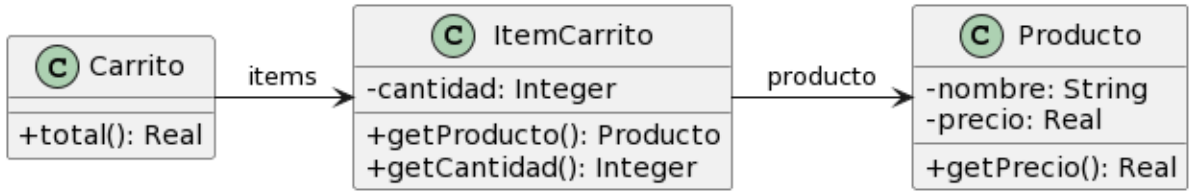
```
public class Juego {
    // ......
    public void incrementar(Jugador j) {
        j.puntuacion = j.puntuacion + 100;
    }
    public void decrementar(Jugador j) {
        j.puntuacion = j.puntuacion - 50;
    }
}

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

3.3 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;
}
```

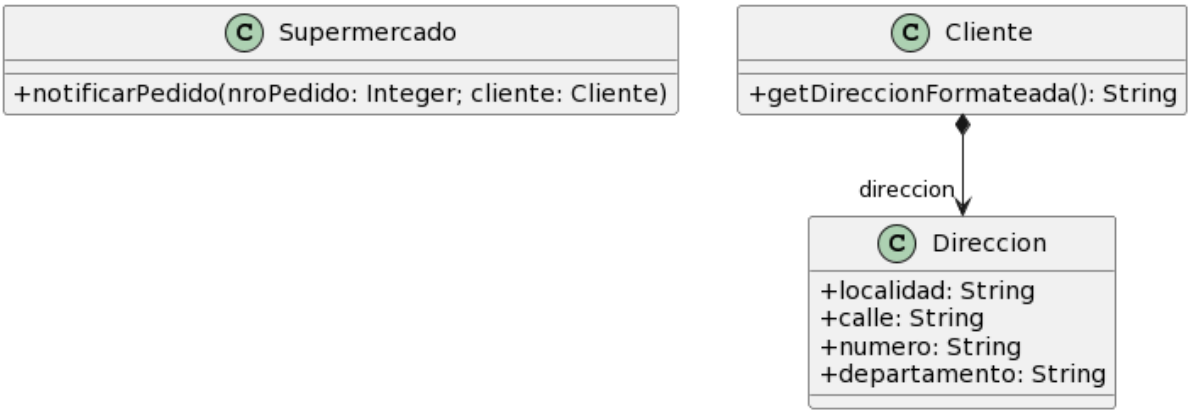
3.4 Carrito de compras




```
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();
    }
}
```

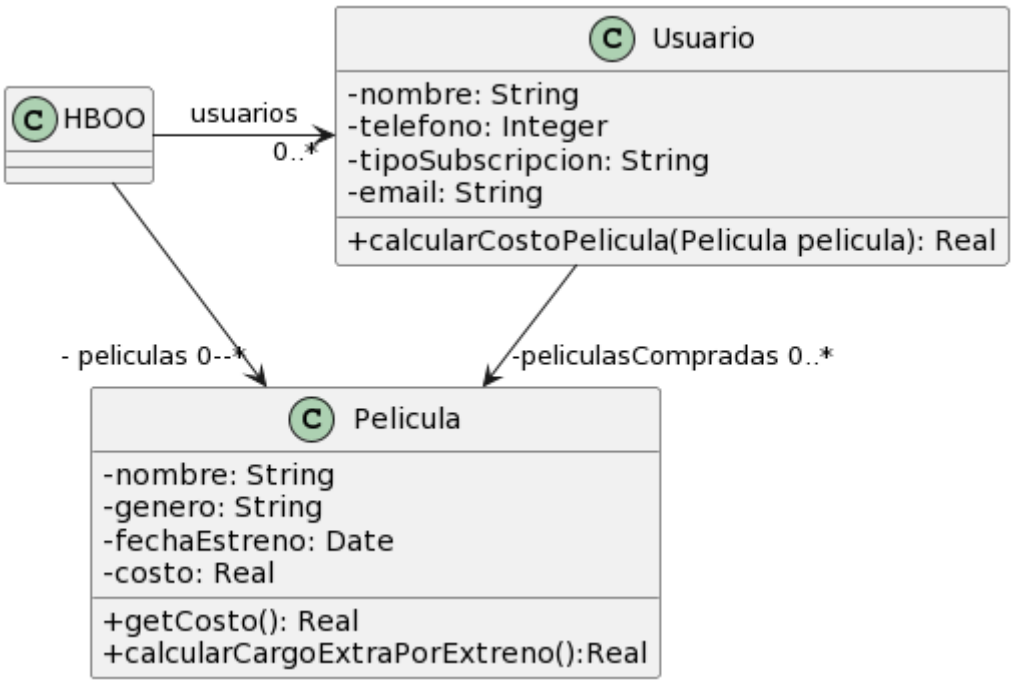
3.5 Envío de pedidos




```
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()
      ;
}
```

3.6 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;
    }
}
```