Redictado Orientación a Objetos 2
Refactoring

Fecha de última edición: 26 abril 2022

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:

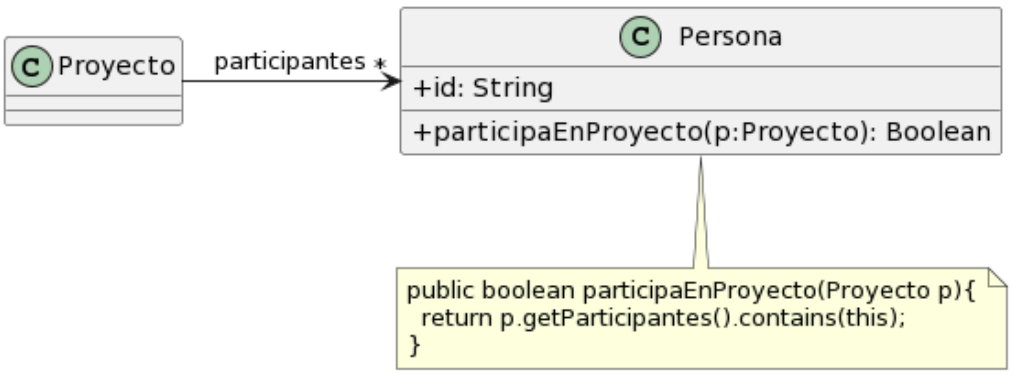


Figura 1: Diagrama de clases del diseño inicial.

Diseño revisado:

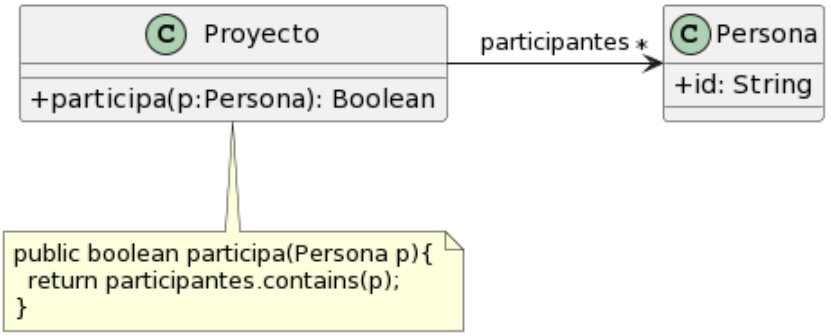


Figura 2: Diagrama de clases modificado.

1.3 Cálculos

Analice el código que se muestra a continuación. Indique qué 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 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

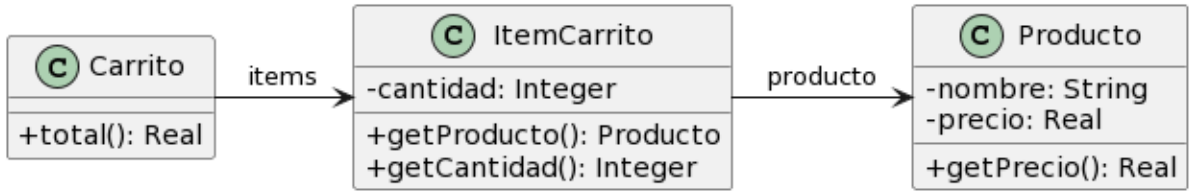
```
public class Juego {
    // ......
    public void incrementar(Jugador j) {
        j.puntuacion = j.puntuacion + 100;
    }
    public void decrementar(Jugador j) {
        j.puntuacion = j.puntuacion - 50;
    }
}

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

2.3 Publicaciones


```
/**
* 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;
}
```

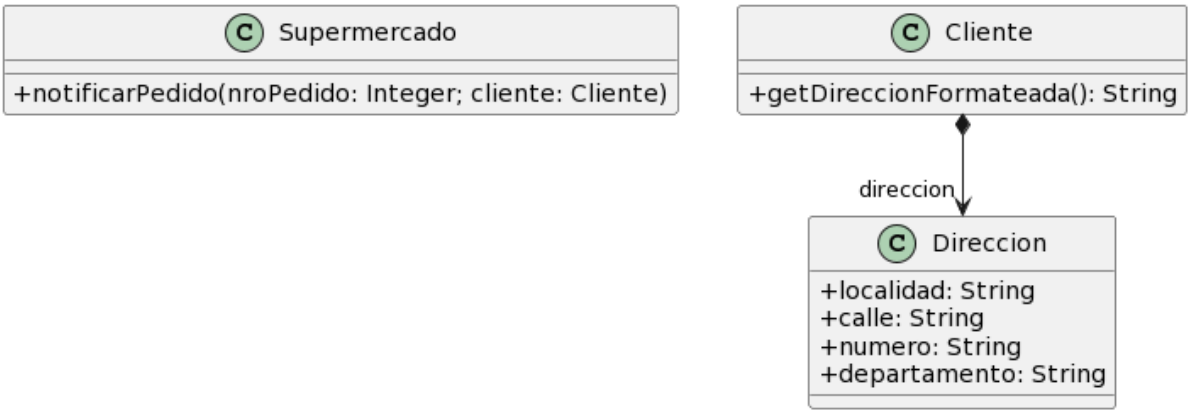
2.4 Carrito de compras




```
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();
    }
}
```

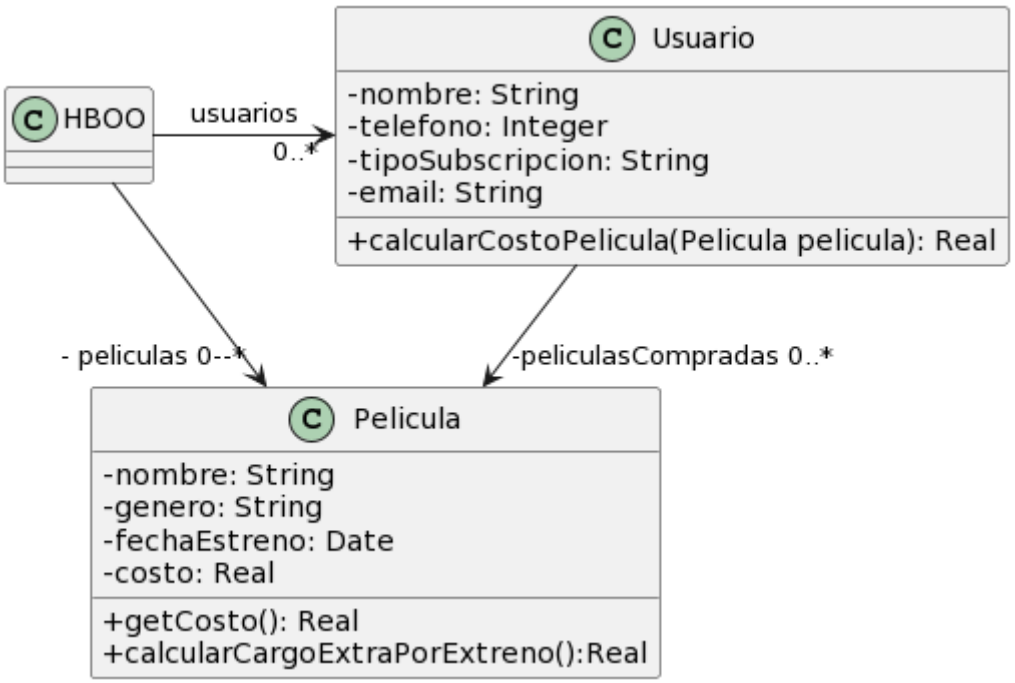
2.5 Envío de pedidos




```
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;
    }
```

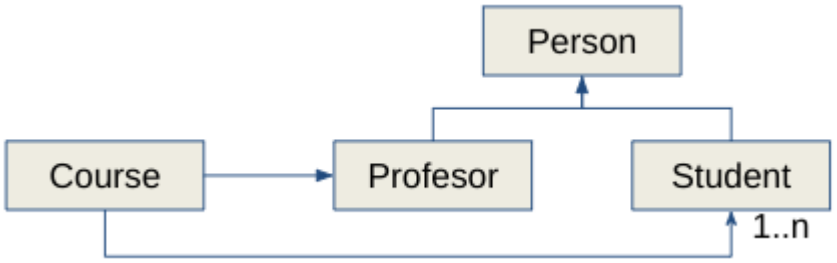
```
    public double calcularCostoPelicula(Pelicula pelicula) {
       double costo = 0;
       if (tipoSubscripcion=="Basico") {
             costo = pelicula.getCosto() +
pelicula.calcularCargoExtraPorEstreno();
       }
       else if (tipoSubscripcion== "Familia") {
             costo = (pelicula.getCosto() +
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

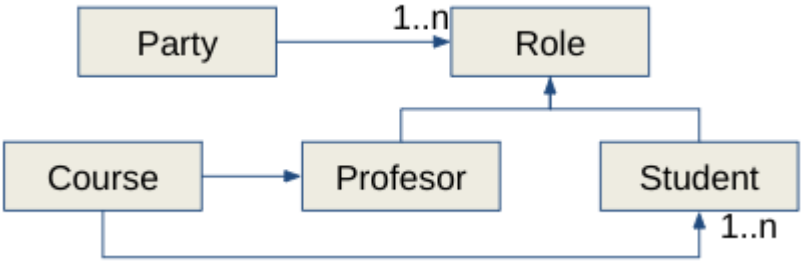
Una plataforma de cursos on-line esta preparando su sistema para hacer una actualización en sus sistema de administración de cursos, alumnos y docentes. Los requerimientos iniciales del sistema fueron:

- El sistema permite registrar los cursos, los docentes y a los alumnos.
- Un Curso tiene un docente
- Un Curso debe tener al menos un alumno

La plataforma busca poder implementar otro tipo de cursos como talleres en donde los alumnos forman grupos de trabajo y se agrega la figura de mentor. Al mismo tiempo se requiere que una persona pueda ser profesor y mentor. El diagrama de clases simplificado del diseño original se presenta a continuacion.



Antes de avanzar con la actualizacion la empresa que desarrolló la plataforma propone hacer un refactoring que prepara el diseño para ser extendido en un siguiente paso mientras que implementa los requerimientos originales. El diseño resulta de aplicar el patrón Roles en donde una persona (Party) puede tener diferentes roles dentro de un modelo.
El diagrama de clases simplificado del diseño resultante del refactoring se presenta a continuacion.



La plataforma le solicita que audite el diseño propuesto demostrando si la transformación propuesta es o no un refactoring.

Ejercicio 4

Considere el código fuente de la clase Student que se presenta a continuación.

```
import java.util.Vector;
import java.util.Iterator;

class Student{
  class ClassTaken{
     String name;
     Boolean done;
     int grade;
     public ClassTaken(String aName){
        name = aName;
        done = false;
        grade =0;
     }
  }

  String name;
  private Vector<ClassTaken> classes; //plan de estudio
  String id;

  public Student(String newName, String ident){
     name = newName;
     id = ident;
     classes = new Vector<ClassTaken>();
  }
  void addClass(String name){
     classes.add(new ClassTaken(name));
  }
  void addDoneClass(String name){
     ClassTaken aClass =new ClassTaken(name);
     aClass.done = true;
     classes.add(aClass);
  }
```

```
  void addGradedClass(String name, int grade){
     ClassTaken aClass =new ClassTaken(name);
     aClass.done = true;
     aClass.grade = grade;
     classes.add(aClass);
  }
  float finalsOverTaken(){
     int finals =0;
     int done=0;
     ClassTaken taken;
     Iterator it = classes.iterator();
     while(it.hasNext()){
        taken=(ClassTaken)it.next();
        if(taken.done)
           done++;
        if(taken.grade>=6)
           finals++;
     }
     return (float)finals/done ;
  }

  boolean fitForScolarship(){
     int tally=0;
     Iterator it = classes.iterator();
     ClassTaken taken;
     while(it.hasNext()){
        taken=(ClassTaken)it.next();
        if (taken.grade >= 4)
           tally++;
     }
     return ((float) tally/classes.size() > 0.5) &&
(this.finalsOverTaken()>0.66);

  }
}
```

Responda las siguientes consignas

1. Documente el código con un diagrama de clases UML
2. Escriba las reglas que hacen que un alumno este en condiciones de recibir una beca que esta implementado por el método fitForScholarship().
3. Desde el código original se implementarons algunas transformaciones

Cambio de nombre	Se renombro el método `finalsOverTaken()` a `goodGradesOverTaken()`

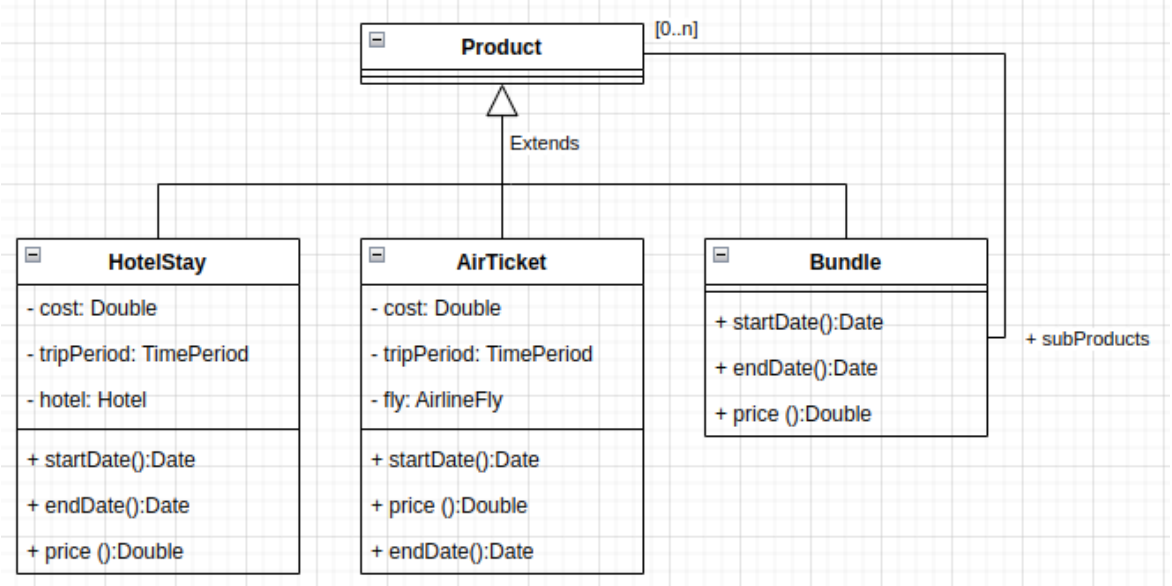
Se elimino constante (hardcoded)	`this.finalsOverTaken()>0.66)` se cambio a
	`this.finalsOverTaken()>fitRatio()`
	y se implemento el metodo
	
	`float fitRatio(){`
	`  return 2/3;`
	`  }`

Se elimino constante (hardcoded)	`(float)tally/classes.size() >= 0.5` se cambio por
	`(float)tally/classes.size() >= finalsRatio()`
	y se implemento el método
	
	`float finalsRatio(){`
	`  return 1/2;`
	`  }`

a. Evalúe (analizando el código) si estos cambios pueden ser refactorings o no?
b. Compile el código del proyecto original y del refactorizado que se encuentra en el moodle de la materia y compare los resultados de correr el programa StudentCmd (original y refactorizado).

Ejercicio 5

Estudie el siguiente diagrama de clases



HotelStay	**startDate()**{ return tripPeriod.start} | **endDate()**{ return tripPeriod.end} **price()**{ return tripPeriod.duration()* hotel.nightPrice()* hotel.discountRate()}
AirTicket	**startDate()**{ return tripPeriod.start} | **endDate()**{ return tripPeriod.end} **price()**{ return fly.price()* fly.promotionRate() }
Bundle	**startDate()**{ return min(subProducts.startDate())} | **endDate()**{ return max(subProducts.startDate())} **price()**{ return sum(subProducts.price())}

Refactorizar de manera que cumpla con las siguientes condiciones

A. Se requiere eliminar la definicion de variables duplicadas.
B. Se requiere eliminar código repetido
C. Se requiere minimizar la cantidad de definiciónes del método **price()**

Ejercicio 6: Antlr lab

Con cada uno de los siguientes ejemplos,

1. Considere que en el lenguaje dado por la cátedra el llamado a una función o método retorna el valor generado por la evaluacion de la última sentencia (***stat***, en la gramática)
2. Determine si hay un code smell y cual es.
3. Evalue el código en antlr lab usando las especificaciones del lexer y el parser
4. Estudie el árbol generado por antlr lab
5. Escriba un pseudocódigo que permita detectar el code smell en el árbol

6.1

```
f(x,y) {
        a = 4
        a = 3 + y;
        x * x + y * x;
}
```

6.2

```
f(x, y, z) {
        a = 3 + y;
        x * x + y * x;
}
```

6.3

```
f(x,y) {
        a = 4;
        x + a;
}
```

6.4

```
f(x) {
        a = 4;
        a = 5;
}
```

6.5

```
f(x) {
        a = x ? 3 : 3;
}
```

6.6

```
f(y) {
        a = g.x() + g.x() + g.x();
}
```

6.7

```
f(y) {
        a = not not y;
}
```

6.8

```
f(y, z) {
        z + 12;
}
```

6.9

```
elegirSueldo(empleado) {
    clase = empleado.class;
    clase.equals(pasante) ? empleado.setSueldo(20000);
    clase.equals(planta) ? empleado.setSueldo(50000);
}
```

6.10

```
agregarOnceNumeros(lista) {
    lista.agregar(1);
    lista.agregar(2);
    lista.agregar(3);
    lista.agregar(4);
    lista.agregar(5);
    lista.agregar(6);
    lista.agregar(7);
    lista.agregar(8);
    lista.agregar(9);
    lista.agregar(10);
    lista.agregar(11);
}
```

6.11

```
numeroTelefonoCompleto(telefono, numero) {
    numero = telefono.codigoArea + telefono.prefijo + telefono.numero;
}
```

6.11

```
f(x,y) {
    x or not x ? y + 1;
    x and x ? y - 1;
    x ? x ? y - 1;
}
```

6.12

```
f(x) {
    x = x;
}
```

6.13

```
f(x,y) {
    x ? y - 1;
    not x ? y - 2;
}
```

6.14

```
f(a,b,c,d,e,f,g,h,i,j,k) {
    a+b+c+d+e+f+g+h+i+j+k;
}
```

6.15

```
someOperation(x,y,z) {
    other.someOperation(x,y,z);
}
```