EjerciciosRefactoring

Ejercicios

Refactoring

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,

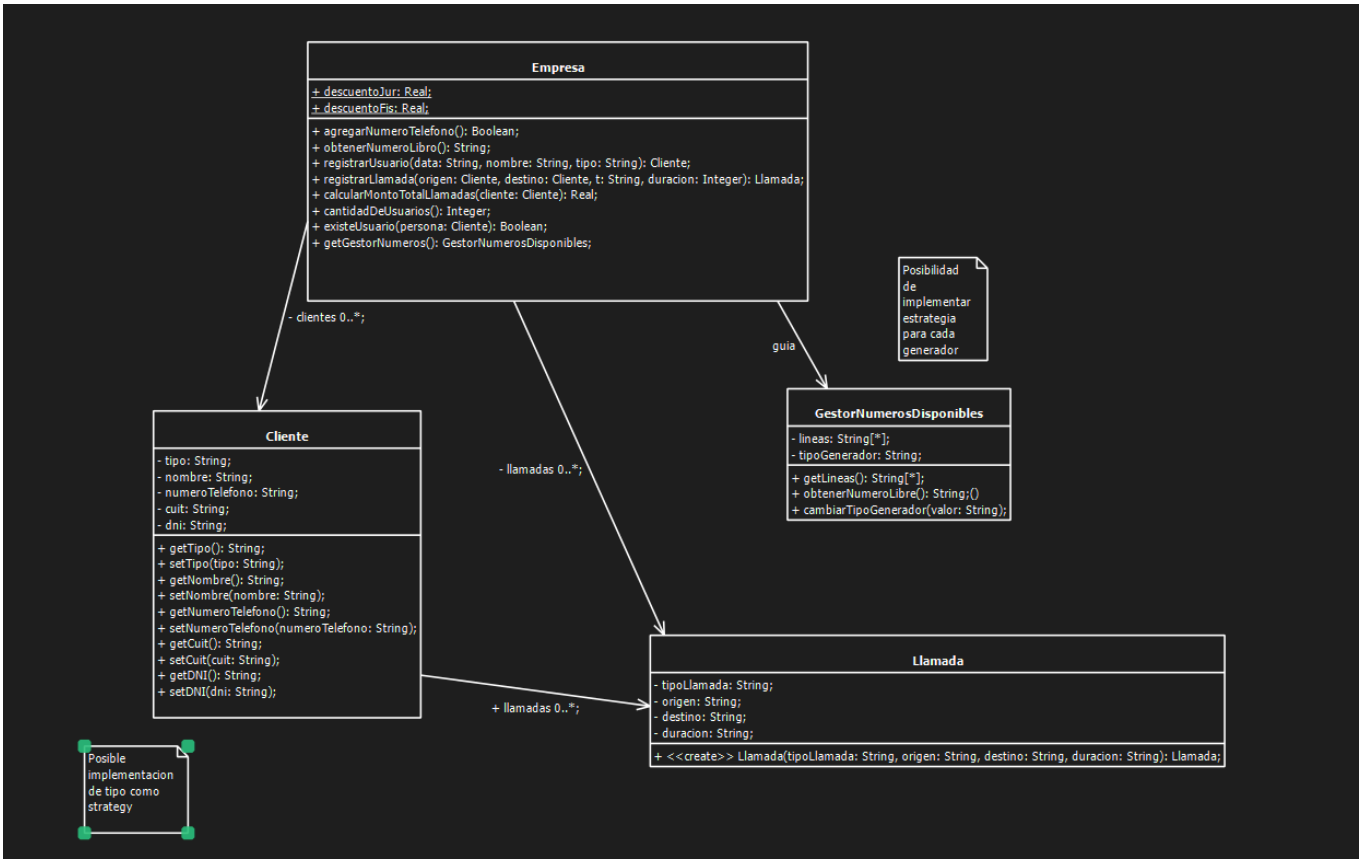
(iii) aplique el refactoring (modifique el código).

(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.



Clase GestorNumerosDisponibles:

Método obtenerNúmeroLibre:

Malos olores:

- Identifiqué un **bad smell del tipo “método largo”** en `obtenerNumeroLibre()`, ya que contenía varias ramas de lógica condicional con acceso repetido a los datos internos del objeto (`lineas`). Para simplificar su responsabilidad y mejorar la legibilidad y mantenibilidad del código, apliqué el refactoring **“Move Method”**, extrayendo esa lógica a una clase separada llamada `Generador`.

```
public class GestorNumerosDisponibles {
    private SortedSet<String> lineas = new TreeSet<String>();
    private Generador tipoGenerador = new Generador();

    public SortedSet<String> getLineas() {
       return lineas;
    }

    public String obtenerNumeroLibre() {
      return this.tipoGenerador.obtenerLibre(this);
    }
```

```
    public void cambiarTipoGenerador(String valor) {
       this.tipoGenerador.setValor(valor);
    }


}
public class Generador {
private String tipoGenerador = "ultimo";
public String obtenerLibre(GestorNumerosDisponibles gestor){
     String linea;
     SortedSet<String> lineas = gestor.getLineas();
     switch (tipoGenerador) {
          case "ultimo":
             linea = lineas.last();
             lineas.remove(linea);
             return linea;
          case "primero":
             linea = lineas.first();
             lineas.remove(linea);
             return linea;
          case "random":
             linea = new ArrayList<String>(lineas)
                   .get(new Random().nextInt(lineas.size()));
             lineas.remove(linea);
             return linea;
       }
       return null;
}
}
```

- Identifique el bad smell Switch statements y codigo duplicado en cada una de los tipo de generador y lo soluciono aplicando “Replace Conditional with Polymorphism" y "Move method" convirtiendo el generador en una interfaz, y 3 clases que sean por cada metodo, y cambiaria el metodo cambiarTipo, y que reciba uno de la interfaz Generador para setearlo y dejando la logica del remove en el Gestor

```
public class GestorNumerosDisponibles {
    private SortedSet<String> lineas = new TreeSet<String>();
    private Generador tipoGenerador = new UltimoGenerador();

    public SortedSet<String> getLineas() {
       return lineas;
    }
```

```
    public String obtenerNumeroLibre() {
      String linea = tipoGenerador.obtenerLibre(this.li);
      lineas.remove(linea);
      return linea;
    }

    public void cambiarTipoGenerador(Generador valor) {
       this.tipoGenerador = valor;
    }
}
public interface Generador {
    public String obtenerLibre(GestorNumerosDisponibles gestor);
}
public UltimoGenerador implements Generador {
    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return gestor.getLineas().last();
    }
}
public randomGenerador implements Generador {

    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return gestor.getLineas().last();
    }
public PrimeroGenerador implements Generador {

    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return new ArrayList<String>(gestor.getLineas())
                        .get(new Random().nextInt(gestor.getLineas().size()));
    }
}
```