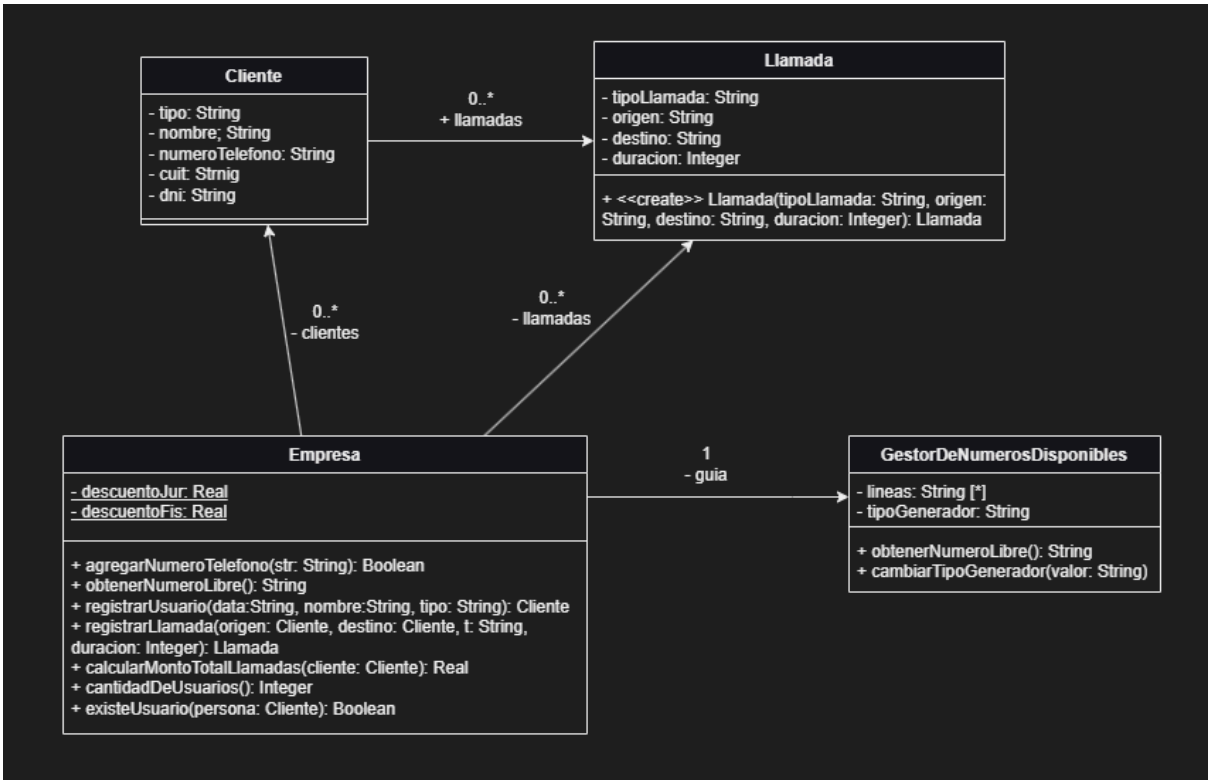
Ejercicio 3 - Facturación de llamadas:

Hecho por: Lautaro Perone 18107/5

- **Diagrama de clases UML con el diseño inicial de la solución provista:**



- **Secuencia de refactoring aplicados:**

En Clase *Empresa*:

Mal olor: ***Clase Grande***

1) Mal olor detectado: ***Envidia de Atributo.***

```
public boolean agregarNumeroTelefono(String str) {
    boolean encontre = guia.getLineas().contains(str);
    if (!encontre) {
        guia.getLineas().add(str);
        encontre= true;
        return encontre;
    }
    else {
        encontre= false;
        return encontre;
    }
}
```

- Refactoring a aplicar:

Refactoring **"Move Method"** a clase *GestorNumerosDisponibles*, indicando que *GestorNumerosDisponibles* sea el responsable de agregar un número de teléfono.

2) Mal olor detectado: ***Código Duplicado, Método Largo.***

```
public Cliente registrarUsuario(String data, String nombre, String tipo) {
    Cliente var = new Cliente();
    if (tipo.equals("fisica")) {
        var.setNombre(nombre);
        String tel = this.obtenerNumeroLibre();
        var.setTipo(tipo);
        var.setNumeroTelefono(tel);
        var.setDNI(data);
    }
    else if (tipo.equals("juridica")) {
        String tel = this.obtenerNumeroLibre();
        var.setNombre(nombre);
        var.setTipo(tipo);
        var.setNumeroTelefono(tel);
        var.setCuit(data);
    }
    clientes.add(var);
    return var;
}
```

- Refactoring a aplicar:

En este método aplico un **"Extract Method"** y además como se pregunta por tipos, aplico **"Replace Conditional with Polymorphism"** creando las clases *PersonaJuridica* y *PersonaFisica.*

- Código corregido:

```
public Cliente registrarClienteFisico(String nombre,String dni) {
    String numeroTelefono = guia.obtenerNumeroLibre();
    Cliente c = new PersonaFisica(nombre,numeroTelefono,dni);
    this.clientes.add(c);
    return c;
}

public Cliente registrarClienteJuridico(String nombre,String cuit) {
    String numeroTelefono = guia.obtenerNumeroLibre();
    Cliente c = new PersonaJuridica(nombre,numeroTelefono,cuit);
    this.clientes.add(c);
    return c;
}
```

3) Mal olor detectado: ***Código Duplicado, Método Largo.***

```
public double calcularMontoTotalLlamadas(Cliente cliente) {
    double c = 0;
    for (Llamada l : cliente.llamadas) {
        double auxc = 0;
        if (l.getTipoDeLlamada() == "nacional") {
            // el precio es de 3 pesos por segundo más IVA sin adicional por establecer la llamada
            auxc += l.getDuracion() * 3 + (l.getDuracion() * 3 * 0.21);
        } else if (l.getTipoDeLlamada() == "internacional") {
            // el precio es de 150 pesos por segundo más IVA más 50 pesos por establecer la llamada
            auxc += l.getDuracion() * 150 + (l.getDuracion() * 150 * 0.21) + 50;
        }

        if (cliente.getTipo() == "fisica") {
            auxc -= auxc*descuentoFis;
        } else if(cliente.getTipo() == "juridica") {
            auxc -= auxc*descuentoJur;
        }
        c += auxc;
    }
    return c;
}
```

- Refactoring a aplicar:

Primero se puede observar que se preguntan por tipos , por ende es conveniente aplicar **“Replace Conditional with Polymorphism”** creando las clases LlamadaNacional y LlamadaInternacional.

En Clase *GestorNumerosDisponibles*:

1) El mal olor encontrado es el sentencias Switch, Método Largo.

Aplique un Strategy utilizando **Replace Conditional Logig with Strategy** para los diferentes tipos de generadores.

```
public String obtenerNumeroLibre() {
    String linea;
    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;
}
```

Y código corregido:

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

```
public interface Generador {

    public String obtenerNumero(SortedSet<String> lineas);
}
```

```
public class GeneradorPrimero implements Generador {

    @Override
    public String obtenerNumero(SortedSet<String> lineas) {
        return lineas.first();
    }
```

```java
public class GeneradorUltimo implements Generador{

    @Override
    public String obtenerNumero(SortedSet<String> lineas) {
        return lineas.last();
    }
```

```java
public class GeneradorRandom implements Generador {

    @Override
    public String obtenerNumero(SortedSet<String> lineas) {
        return new ArrayList<String>(lineas)
                .get(new Random().nextInt(lineas.size()));
    }
```

En Clase *Cliente*:

1) El mal olor que encontre es que es una Clase de Datos por lo que aplique **Move Method** del registro de llamadas y el cálculo de monto total.

```java
public Llamada registrarLlamadaNacional(Cliente destino, int duracion) {
    Llamada l = new LlamadaNacional(this.getNumeroTelefono(), destino.getNumeroTelefono(),duracion);
    this.llamadas.add(l);
    return l;
}

public Llamada registrarLlamadaInternacional(Cliente destino, int duracion) {
    Llamada l = new LlamadaInternacional(this.getNumeroTelefono(), destino.getNumeroTelefono(),duracion);
    this.llamadas.add(l);
    return l;
}
```

2) Aca aplique el uso de streams.

```java
public double calcularMontoTotalLlamadas() {
    return this.llamadas.stream()
            .mapToDouble(llamada -> {
                double auxc = llamada.calcularMonto();
                return auxc -= this.aplicarDescuento(auxc);
            })
            .sum();
}

public abstract double aplicarDescuento(double monto);
```

Y en las subclases de Cliente:

PersonaFisica:

```
@Override
public double aplicarDescuento(double monto) {
    return monto * this.getDescuentoFis();
}
```

PersonaJuridica:

```
@Override
public double aplicarDescuento(double monto) {
    return monto * this.getDescuentoJur();
}
```

Donde si se quiere cambiar el descuento en un futuro se puede hacer.

En clase Llamada:
Que es otra clase de datos le delegue el método de calcular su monto:

```
public abstract double calcularMonto();
```

Para que sus subclases puedan calcularlo:
En clase LlamadaNacional:

```
@Override
public double calcularMonto() {
    // el precio es de 3 pesos por segundo más IVA sin adicional por establecer la llamada
    return this.getDuracion() * this.getPrecioPorSegundo()
            + this.getDuracion() * this.getPrecioPorSegundo() * this.getIva();
}
```

En clase LlamadaInternacional:

```
@Override
public double calcularMonto() {
    // el precio es de 150 pesos por segundo más IVA más 50 pesos por establecer la llamada
    return (this.getDuracion() * this.getPrecioPorSegundo())
            + (this.getDuracion() * this.getPrecioPorSegundo()
                    * this.getIva()) + this.precioLlamada;
}
```

Los valores de precioPorSegundo , iva , precioLlamada se pueden modificar en un futuro ya que antes eran valores explicitos en el código.

Conclusiones:

Me sucede que se como solucionar un problema(un code smells) pero a veces no sé cual es el nombre del refactoring que use.

Espero que puedan corregirme , no tuve mucho tiempo para hacer el trabajo, todas las correcciones las tomaré en cuenta. Muchas gracias!