

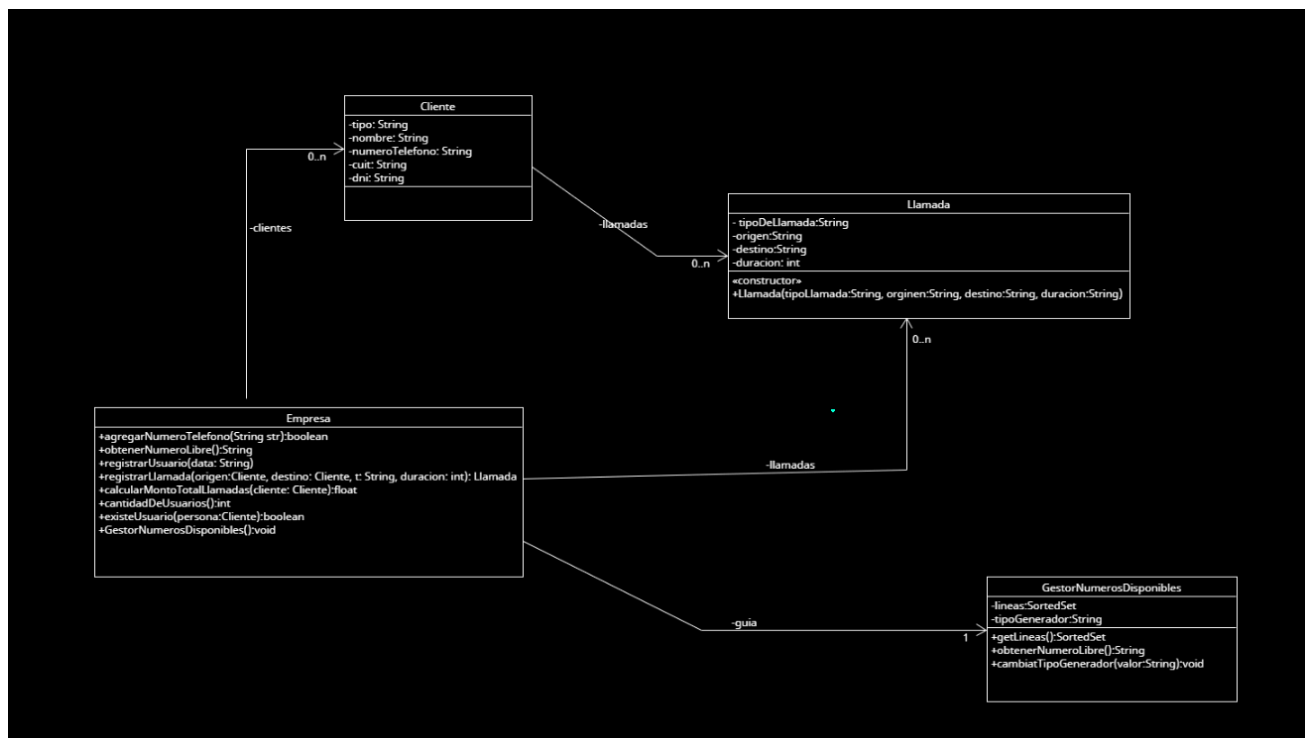
# Ejercicio 3 - Facturación de Llamadas

Integrantes del grupo:

-Francisco Manuel Jorge

-Lautaro Gutierrez

-Diagrama UML con la solución provista



## 1- Mal olor detectado: "Switch statements"

Al tener ifs dentro del método para instanciar objetos según el tipo en registrar usuario, podemos aplicar polimorfismo y distinguir entre clientes físicos y jurídicos.

```
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: Replace Conditional with Polymorphism./  
push down method./ push down fields

```
public Cliente registrarClienteJuridico(String cuit, String nombre, String tipo){
    Juridico cliente = new Juridico(tipo,nombre,this.obtenerNumeroLibre(),cuit);
    this.clientes.add(cliente);
    return cliente;
}

tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public Cliente registrarClienteFisico(String dni, String nombre, String tipo){
    Fisico cliente = new Fisico(tipo,nombre,this.obtenerNumeroLibre(),dni);
    this.clientes.add(cliente);
    return cliente;
}
```

Clase Cliente:

```
1  public abstract class Cliente {
2      private List<Llamada> llamadas = new ArrayList<Llamada>();
3      private String nombre;
4      private String numeroTelefono;
5      protected double descuento;
6
7
8
9      public Cliente (String nombre, String numeroTelefono){
10         this.nombre = nombre;
11         this.numeroTelefono = numeroTelefono;
12     }
```

Clase Juridico:

```
1  public class Juridico extends Cliente {
2
3      private String cuit;
4
5      public Juridico (String nombre, String numeroTelefono, String cuit){
6          super(nombre,numeroTelefono);
7          this.cuit= cuit;
8          this.descuento = 0.15;
9      }
10
11
12
13 }
```

Clase Fisico:

```

1  public class Fisico extends Cliente {
2
3  private String dni;
4
5      public Fisico (String nombre, String numeroTelefono, String dni){
6          super(nombre,numeroTelefono);
7          this.dni= dni;
8          this.descuento = 0;
9      }
10
11
12 }

```

Esto genera un cambio en el test.

Antes:

```

@Test
void testcalcularMontoTotalLlamadas() {
    Cliente emisorPersonaFisca = sistema.registrarUsuario("11555666", "Brendan Eich", "fisica");
    Cliente remitentePersonaFisca = sistema.registrarUsuario("00000001", "Doug Lea", "fisica");
    Cliente emisorPersonaJuridica = sistema.registrarUsuario("17555222", "Nvidia Corp", "juridica");
    Cliente remitentePersonaJuridica = sistema.registrarUsuario("25765432", "Sun Microsystems", "juridica");
}

```

Despues:

```

@Test
void testcalcularMontoTotalLlamadas() {
    Cliente emisorPersonaFisca = sistema.registrarClienteFisico(dni:"11555666", nombre:"Brendan Eich");
    Cliente remitentePersonaFisca = sistema.registrarClienteFisico(dni:"00000001", nombre:"Doug Lea");
    Cliente emisorPersonaJuridica = sistema.registrarClienteJuridico(cuit:"17555222", nombre:"Nvidia Corp");
    Cliente remitentePersonaJuridica = sistema.registrarClienteJuridico(cuit:"25765432", nombre:"Sun Microsystems");
}

```

## 2- Mal olor detectado: “dead code”

Podemos observar que el parámetro “tipo” ya no es necesario al haber realizado la jerarquía anteriormente.

```

public Cliente registrarClienteJuridico(String cuit, String nombre, String tipo){
    Juridico cliente = new Juridico(tipo,nombre,this.obtenerNumeroLibre(),cuit);
    this.clientes.add(cliente);
    return cliente;
}

tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public Cliente registrarClienteFisico(String dni, String nombre, String tipo){
    Fisico cliente = new Fisico(tipo,nombre,this.obtenerNumeroLibre(),dni);
    this.clientes.add(cliente);
    return cliente;
}

```

Refactoring aplicado: “remove parameter”

```

1
2 public Cliente registrarClienteJuridico(String cuit, String nombre){
3     Juridico cliente = new Juridico(nombre,this.obtenerNumeroLibre(),cuit);
4     this.clientes.add(cliente);
5     return cliente;
6 }
7
8 public Cliente registrarClienteFisico(String dni, String nombre){
9     Fisico cliente = new Fisico(nombre,this.obtenerNumeroLibre(),dni);
10    this.clientes.add(cliente);
11    return cliente;
12 }

```

### 3- Mal olor detectado: “Feature envy”

El método registrarLlamada toma un atributo del cliente origen y trabaja sobre él, por lo que se considera envidia de atributos y necesitamos crear el método a la clase cliente para que agregue una llamada.

```

public Llamada registrarLlamada(Cliente origen, Cliente destino, String t, int duracion) {
    Llamada llamada = new Llamada(t, origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);
    llamadas.add(llamada);
    origen.llamadas.add(llamada);
    return llamada;
}

```

## Refactoring aplicado: “Extract method”

```
public Llamada registrarLlamada(Cliente origen, Cliente destino, String t, int duracion) {  
    Llamada llamada = new Llamada(t, origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);  
    llamadas.add(llamada);  
    origen.agregarLlamada(llamada);  
    return llamada;  
}
```

## 4- Mal olor detectado: “Switch statements”

Al tener ifs dentro del método “calcularmontototalLlamadas”, podemos aplicar polimorfismo y distinguir entre llamadas nacionales e internacionales, y entre clientes jurídicos y físicos. De esta forma también se elimina la variable tipodeLlamada.

```

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 aplicado: Replace Conditional with Polymorphism/  
 “remove dead variable”

Clase Llamada:

```
1  public abstract class Llamada {
2      private String origen;
3      private String destino;
4      private int duracion;
5
6      public Llamada(String origen, String destino, int duracion) {
7          this.origen = origen;
8          this.destino = destino;
9          this.duracion = duracion;
10     }
11
12     public abstract double calcularMontoTotaldeLlamada();
13
14     public String getRemitente() {
15         return destino;
16     }
17
18     public int getDuracion() {
19         return this.duracion;
20     }
21
22     public String getOrigen() {
23         return origen;
24     }
25 }
26
```

Clase Internacional:

```
1  public class Internacional extends Llamada{
2
3
4      public Internacional (String origen, String destino, int duracion){
5          super(origen,destino,duracion);
6      }
7      @Override
8      public double calcularMontoTotaldeLlamada(){
9          return this.getDuracion() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
10     }
11
12 }
```



Clase Nacional:

```
1 public class Nacional extends Llamada{
2
3     public Nacional (String origen, String destino, int duracion){
4         super(origen,destino,duracion);
5     }
6     @Override
7     public double calcularMontoTotaldeLlamada(){
8         return this.getDuracion() * 3 + (this.getDuracion() * 3 * 0.21);
9     }
}
```

## 5- Mal olor detectado: “Feature envy”

En el método calcularMontoTotalLlamadas podemos observar que se utiliza el atributo de cliente “llamadas” por lo que se considera envidia de atributos y debe moverse el método hacia la clase del propio cliente.

```
public double calcularMontoTotalLlamadas(Cliente cliente) {
    double c = 0;
    for (Llamada l : cliente.llamadas) {
        double auxc = 0;
        auxc = l.calcularMontoTotaldeLlamada();
        auxc -= cliente.getDescuento();
        c+= auxc;
    }
    return c;
}
```

Refactoring aplicado: “Move method”/ “Push down methods”

Clase Empresa:

```
public double calcularMontoTotalLlamadas(Cliente cliente) {  
    return cliente.calcularMontoTotalLlamadas();  
}
```

Clase Cliente:

```
1 public double calcularMontoTotalLlamadas(){  
2     double total = llamadas.stream().mapToDouble(llamada -> llamada.calcularMontoTotaldeLlamada()).sum();  
3     total -= this.getDescuento() * total;  
4     return total;  
5 }
```

Clase Llamada:

```
public abstract double calcularMontoTotaldeLlamada();
```

Subclase de llamada, Nacional:

```
@Override  
public double calcularMontoTotaldeLlamada(){  
    return super.getDuracion() * 3 + super.getDuracion() * 3 * 0.21;  
}
```

Subclase de llamada, Internacional:

```
@Override  
public double calcularMontoTotaldeLlamada(){  
    return this.getDuracion() * 150 + (this.getDuracion() * 150 * 0.21) + 50;  
}
```

## 6- Mal olor detectado: “Feature envy”

En el método agregarNumeroTelefono podemos observar que se utiliza el atributo de la guia “líneas” por lo que se considera envidia de atributos y debe moverse el método hacia la clase GestorNumerosDisponibles.

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

Refactoring aplicado: “Move method”/ “extract method”

Clase Empresa:

```
1 public boolean agregarNumeroTelefono(String numero) {  
2     if (! this.guia.Verificar(numero)){  
3         guia.agregarNumero(numero);  
4         return true;  
5     }  
6     return false;  
7 }
```

Clase GestorNumerosDisponibles:

```
1 public boolean Verificar(String numero){  
2     return this.gestor.Verificar(numero);  
3 }
```

```
public void agregarNumero(String numero){  
    this.lineas.add(numero);  
}
```

## 7- Mal olor detectado: “Dead code”

En estas líneas se establecen dos variables estáticas para los distintos descuentos, al utilizar el polimorfismo estos datos son atributos del cliente jurídico y físico respectivamente.

```
static double descuentoJur = 0.15;  
static double descuentoFis = 0;
```

Refactoring aplicado: “Remove dead code”.

## 8- Mal olor detectado: “Exposed Field”

Este método rompe completamente el encapsulamiento, por lo que se lo encapsula y busca otra manera de acceder a la guía para utilizar los métodos de esta misma.

```
public GestorNumerosDisponibles getGestorNumeros() {  
    return this.guia;  
}
```

Refactoring aplicado: “rename method”/“add parameter”

```
1 public void cambiarTipoGenerador(GestorNumeros gest){  
2     this.guia.cambiarTipoGenerador(gest);  
3 }  
4
```

## 9- Mal olor detectado: “Exposed Field”

Al tener un get y set para cada atributo de la clase se rompe el encapsulamiento, por lo que se eliminan los getters y setters que no son necesarios.

Clase Persona:

```
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public String getTipo() {
    return tipo;
}
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public void setTipo(String tipo) {
    this.tipo = tipo;
}
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public String getNombre() {
    return nombre;
}
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public void setNombre(String nombre) {
    this.nombre = nombre;
}
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public String getNumeroTelefono() {
    return numeroTelefono;
}
tabnine: test | explain | document | ask | tabnine: test | explain | document | ask
public void setNumeroTelefono(String numeroTelefono) {
    this.numeroTelefono = numeroTelefono;
}
```

Refactoring aplicado: “Remove setting method”.

```
1 public String getNumeroTelefono(){
2     return this.numeroTelefono;
3 }
```

## 10- Mal olor detectado: “Exposed Field”

Tener un atributo público rompe el encapsulamiento, por lo que se debe cambiar el alcance.

```
public List<Llamada> llamadas = new ArrayList<Llamada>();
```

Refactoring aplicado: “Encapsulate field”

```
private List<Llamada> llamadas = new ArrayList<Llamada>();
```

## 11- Mal olor detectado: “Switch statements”

La sentencia case dentro del método “obtenerNumeroLibre” se puede cambiar por polimorfismo distinguiendo entre “PrimerNumero”, “UltimoNumero” y “NumeroRandom”. Esto genera un push down de los métodos a las clases concretas y la creación de metodos get y set para los cambios de variable de instancia. (ej: queremos cambiar el tipo de generador a primer numero de las lineas, entonces pasamos una instancia de “PrimerNumero” y antes de asignarla tenemos que settear a esta instancia con la coleccion de numeros disponibles)  
Esto genera cambios en el test.

```

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;
}

```

Refactoring aplicado: “Replace Conditional with Polymorphism”/  
 “add variable”/ “push down methods”/ “rename method”/ “move  
 field”



```


1  public class GestorNumerosDisponibles {
2      private GestorNumeros gestor = new UltimoNumero();
3
4
5      public void agregarNumero(String numero){
6          this.gestor.agregarNumero(numero);
7      }
8
9      public boolean Verificar(String numero){
10         return this.gestor.Verificar(numero);
11     }
12     public String obtenerNumeroDisponible(){
13         return gestor.obtenerNumeroDisponible();
14     }
15
16     public void cambiarTipoGenerador(GestorNumeros gest){
17         gest.setLineas(this.gestor.getLineas());
18         this.gestor = gest;
19     }
20 }

```


```

1  public abstract class GestorNumeros {
2
3      protected SortedSet<String> lineas = new TreeSet<String>();
4
5
6      public SortedSet<String> getLineas(){
7          return this.lineas;
8      }
9      public void setLineas(SortedSet<String> lineas){
10         this.lineas = lineas;
11     }
12
13
14
15     public void agregarNumero(String numero){
16         this.lineas.add(numero);
17     }
18     public boolean Verificar(String numero){
19         return this.lineas.contains(numero);
20     }
21     public abstract String obtenerNumeroDisponible();
22
23
24 }


```



```
1 public class UltimoNumero extends GestorNumeros{
2
3     @Override
4     public String obtenerNumeroDisponible(){
5         String num = this.lineas.last();
6         lineas.remove(num);
7         return num;
8     }
9
10 }
```



```
1 public class PrimerNumero extends GestorNumeros{
2
3     @Override
4     public String obtenerNumeroDisponible(){
5         String num = this.lineas.first();
6         lineas.remove(num);
7         return num;
8     }
9
10 }
11
```



```
1 public class NumeroRandom extends GestorNumeros{
2
3     @Override
4     public String obtenerNumeroDisponible(){
5         String num = new ArrayList<String>(this.lineas).get(new Random().nextInt(this.lineas.size()));
6         this.lineas.remove(num);
7         return num;
8     }
9
10
11 }
```

## Cambios en el test:

Antes:

```
1  @Test
2  void obtenerNumeroLibre() {
3      // por defecto es el ultimo
4      assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
5
6      this.sistema.getGestorNumeros().cambiarTipoGenerador("primero");
7      assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
8
9      this.sistema.getGestorNumeros().cambiarTipoGenerador("random");
10     assertNotNull(this.sistema.obtenerNumeroLibre());
11 }
```

Despues:

```
1  @Test
2  void obtenerNumeroLibre() {
3      // por defecto es el ultimo
4      assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
5
6      this.sistema.cambiarTipoGenerador(new PrimerNumero());
7      assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
8
9      this.sistema.cambiarTipoGenerador(new NumeroRandom());
10     assertNotNull(this.sistema.obtenerNumeroLibre());
11 }
```

## 12- Mal olor detectado: “dead code”

Al realizar la jerarquía anterior, el atributo tipoGenerador ya no es útil, por lo que se elimina.

```
private String tipoGenerador = "ultimo";
```

Refactoring utilizado: “Remove variable”

## 13- Mal olor detectado: “Expose field”

Al igual que en la clase empresa, el `getLineas` rompe el encapsulamiento. Por lo que agregamos un parámetro para poder acceder y utilizar el metodo `cambiarTipoGenerador`.

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

Refactoring aplicado: “Add Parameter”/ “rename method”

```
1 public void cambiarTipoGenerador(GestorNumeros gest){  
2     this.guia.cambiarTipoGenerador(gest);  
3 }
```

## 14- Mal olor detectado: “Dead code”

En este caso tenemos 4 parámetros de los cuales el “t” que referencia al tipo de llamada ya no nos es útil por la jerarquía antes aplicada.

```
1 public Llamada registrarLlamada(Cliente origen, Cliente destino, String t, int duracion) {
2     Llamada llamada = new Llamada(t, origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);
3     llamadas.add(llamada);
4     origen.llamadas.add(llamada);
5     return llamada;
6 }
```

## Refactoring aplicado: “Remove parameter”

```
1 public Llamada registrarLlamadaInternacional(Cliente origen, Cliente destino, int duracion){
2     Internacional llamada = new Internacional(origen.getNumeroTelefono(),destino.getNumeroTelefono(),duracion);
3     this.llamadas.add(llamada);
4     origen.agregarLlamada(llamada);
5     return llamada;
6 }
7 public Llamada registrarLlamadaNacional(Cliente origen, Cliente destino, int duracion){
8     Nacional llamada = new Nacional(origen.getNumeroTelefono(),destino.getNumeroTelefono(),duracion);
9     this.llamadas.add(llamada);
10    origen.agregarLlamada(llamada);
11    return llamada;
12 }
13
14
```

## 15- Mal olor detectado: “Exposed fields”

Los getters como “getRemitente” o “get Origen” no son utilizados, por lo que se los eliminan para no romper el encapsulamiento.

```
1  public abstract class Llamada {
2      private String origen;
3      private String destino;
4      private int duracion;
5
6      public Llamada(String origen, String destino, int duracion) {
7          this.origen = origen;
8          this.destino = destino;
9          this.duracion = duracion;
10     }
11
12     public abstract double calcularMontoTotaldeLlamada();
13
14     public String getRemitente() {
15         return destino;
16     }
17
18     public int getDuracion() {
19         return this.duracion;
20     }
21
22     public String getOrigen() {
23         return origen;
24     }
25 }
```

## Refactoring aplicado: "Remove setting method"

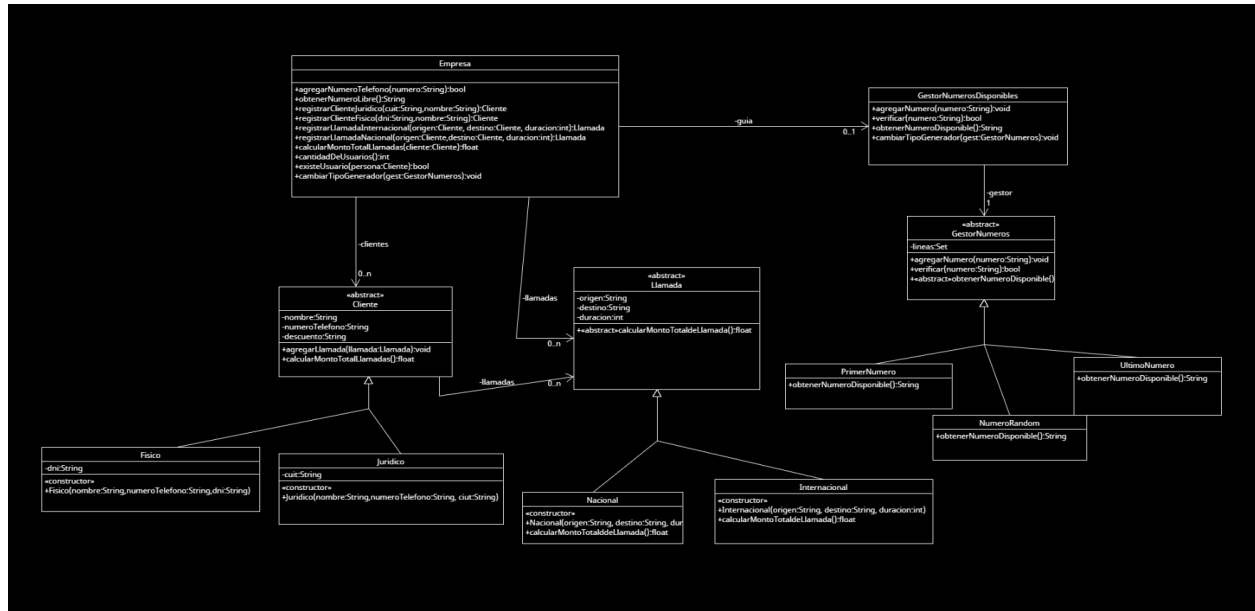
```
1  public abstract class Llamada {
2      private String origen;
3      private String destino;
4      private int duracion;
5
6      public Llamada(String origen, String destino, int duracion) {
7          this.origen = origen;
8          this.destino = destino;
9          this.duracion = duracion;
10     }
11
12     public abstract double calcularMontoTotaldeLlamada();
13
14
15     public int getDuracion() {
16         return this.duracion;
17     }
18
19 }
```

Aclaración: En cuanto al alto acoplamiento que existe entre la clase empresa y Llamada, a nuestra forma de verlo esto puede ser una especificación del dominio del problema y no lo vemos como un mal olor.

## -UML luego de refactorizar el código:

En caso que no se pueda ver bien en la foto, dejo el link con el archivo del UML.

[https://drive.google.com/file/d/1wT6\\_sg99CxO9qMKDoUnUdxhqdQojB3rY/view?usp=sharing](https://drive.google.com/file/d/1wT6_sg99CxO9qMKDoUnUdxhqdQojB3rY/view?usp=sharing)



## Código completo refactorizado:

[https://github.com/frnJJ/002/tree/main/ej3refact/src/main/java/ar/edu/unlp/info/oo2/facturacion\\_llamadas](https://github.com/frnJJ/002/tree/main/ej3refact/src/main/java/ar/edu/unlp/info/oo2/facturacion_llamadas)