# Ejercicio 3 Refactoring

1. Mal olor: **Long class y feature envy** en la clase **Empresa**. Vemos una clase que intenta acaparar todas las funciones posibles, dejandola como una clase gigante y con problemas al delegar el trabajo.

   Extracto de codigo con feature envy:

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

Refactoring: Move method. El metodo agregarNumeroTelefono se translada a la clase GestorNumerosDisponibles, la cual funciona como guia telefonica.

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

2. Mal olor: Long class y feature envy en la clase **Empresa**. Otro indicador de que Empresa acaparó todos los metodos es la clase **Cliente**, la cual funciona unicamente como data class.

   Extracto de codigo:

```
public class Empresa{
//..
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
                auxc += l.getDuracion() * 3 + (l.getDuraci
            } else if (l.getTipoDeLlamada() == "internacio
                // el precio es de 150 pesos por segundo m
                auxc += l.getDuracion() * 150 + (l.getDura
            }

            if (cliente.getTipo() == "fisica") {
```

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

Refactoring: Move method. Se translada el metodo calcularMontoTotalLlamadas desde Empresa a Cliente.

```
public class cliente{
//..
public double calcularMontoTotalLlamadas(Empresa empresa)
        double c = 0;
        for (Llamada l : llamadas) {
            double auxc = 0;
            if (l.getTipoDeLlamada() == "nacional") {
                // el precio es de 3 pesos por segundo más
                auxc += l.getDuracion() * 3 + (l.getDuraci
            } else if (l.getTipoDeLlamada() == "internacio
                // el precio es de 150 pesos por segundo m
                auxc += l.getDuracion() * 150 + (l.getDura
            }
            if (getTipo() == "fisica") {
                auxc -= auxc*Empresa.descuentoFis;
            } else if(getTipo() == "juridica") {
                auxc -= auxc*Empresa.descuentoJur;
            }
            c += auxc;
        }
        return c;
    }
```

3. Mal olor: Long method

```
public class cliente{
//..
public double calcularMontoTotalLlamadas(Empresa empres
        double c = 0;
        for (Llamada l : llamadas) {
            double auxc = 0;
            if (l.getTipoDeLlamada() == "nacional") {
                // el precio es de 3 pesos por segundo
                auxc += l.getDuracion() * 3 + (l.getDur
            } else if (l.getTipoDeLlamada() == "interna
                // el precio es de 150 pesos por segund
                auxc += l.getDuracion() * 150 + (l.getD
            }
            if (getTipo() == "fisica") {
                auxc -= auxc*Empresa.descuentoFis;
            } else if(getTipo() == "juridica") {
                auxc -= auxc*Empresa.descuentoJur;
            }
            c += auxc;
        }
        return c;
    }
```

Refactoring: **Extract method**

```
 public double calcularMontoTotalLlamadas(Empresa empre
     double c = 0;
     for (Llamada l : llamadas) {
         double auxc = costoPorTipo(l);
         if (this.getTipo() == "fisica") {
             auxc -= auxc*Empresa.descuentoFis;
         } else if(this.getTipo() == "juridica") {
             auxc -= auxc*Empresa.descuentoJur;
         }
         c += auxc;
     }
     return c;
 }
```

```
    private double costoPorTipo(Llamada l) {
        double auxc = 0;
        if (l.getTipoDeLlamada() == "nacional") {
            // el precio es de 3 pesos por segundo más IVA
            auxc += l.getDuracion() * 3 + (l.getDuracion()
        } else if (l.getTipoDeLlamada() == "internacional"
            // el precio es de 150 pesos por segundo más I
            auxc += l.getDuracion() * 150 + (l.getDuracion
        }
        return auxc;
    }
```

4. Mal olor: No se especifica el alcance de las variables descuentoJur y descuentoFis, por lo tanto se asigna por defecto el alcance package private.

```
public class Empresa{
//..
    static double descuentoJur = 0.15;
    static double descuentoFis = 0;
```

Refactoring: Encapsulate field

```
public class Empresa{
//..
    private static double descuentoJur = 0.15;
    private static double descuentoFis = 0;

    public static double getDescuentoJur() {
        return descuentoJur;
    }
    public static void setDescuentoJur(double descuentoJur
        Empresa.descuentoJur = descuentoJur;
    }
    public static double getDescuentoFis() {
        return descuentoFis;
    }
    public static void setDescuentoFis(double descuentoFis
```

```
        Empresa.descuentoFis = descuentoFis;
    }
```

5. Mal olor: Long method. El metodo calcularMontoTotalLlamadas aun es capaz de seguirse descomponiendo y modularizando.

```
    public class Cliente{
    //..
    public double calcularMontoTotalLlamadas(Empresa empre
        double c = 0;
        for (Llamada l : llamadas) {
            double auxc = costoPorTipo(l);
            if (this.getTipo() == "fisica") {
                auxc -= auxc*Empresa.getDescuentoFis();
            } else if(this.getTipo() == "juridica") {
                auxc -= auxc*Empresa.getDescuentoJur();
            }
            c += auxc;
        }
        return c;
```

Refactoring: Extract method

```
    public class Cliente{
    //..
public double calcularMontoTotalLlamadas(Empresa empresa)
        double c = 0;
        for (Llamada l : llamadas) {
            double auxc = costoPorTipoDeLlamada(l);
            auxc = multiplicadorCostoPorTipoDeCliente(auxc
            c += auxc;
        }
        return c;
    }

        private double multiplicadorCostoPorTipoDeCliente(
        if (this.getTipo() == "fisica") {
            auxc -= auxc*Empresa.getDescuentoFis();
```

```
        } else if(this.getTipo() == "juridica") {
            auxc -= auxc*Empresa.getDescuentoJur();
        }
        return auxc;
    }
```

6. Mal olor: Reinventar la rueda con el recorrido del vector en vez de utilizar las herramientsa que nos prevee el lenguaje, el recorrido del vector ya esta implementado con streams.

```
public class Cliente{
//..
public double calcularMontoTotalLlamadas(Empresa empresa)
        double c = 0;
        for (Llamada l : llamadas) {
            double auxc = costoPorTipoDeLlamada(l);
            auxc = multiplicadorCostoPorTipoDeCliente(auxc
            c += auxc;
        }
        return c;
    }
```

Refactoring: Replace loop with pipeline

```
    public class Cliente{
    //..
    public double calcularMontoTotalLlamadas(Empresa empre
        return this.llamadas.stream().mapToDouble(llamada-
```

7. Mal olor: Logica de condicionales evitables

```
public class Cliente{
private double costoPorTipoDeLlamada(Llamada l) {
        double auxc = 0;
        if (l.getTipoDeLlamada() == "nacional") {
            // el precio es de 3 pesos por segundo más IVA
            auxc += l.getDuracion() * 3 + (l.getDuracion()
```

```
        } else if (l.getTipoDeLlamada() == "internacional"
            // el precio es de 150 pesos por segundo más I
            auxc += l.getDuracion() * 150 + (l.getDuracion
        }
        return auxc;
    }
```

Refactoring: Replace conditional logic with polymorphism.

Se modifico:

- La llamada al metodo costoPorTipoDeLlamada().
- Se eliminó la variable de clase para el tipo de llamada y se reemplazo por dos subclases de llamada.
- Fue necesario adaptar los test a los cambios.

```
public class Cliente{
//..
public double calcularMontoTotalLlamadas(Empresa empresa)
        return this.llamadas.stream().mapToDouble(llamada-
    }
public class Empresa(){
    public Llamada registrarLlamadaInternacional(Cliente o
        Llamada llamada = new LlamadaInternacional(origen.
        llamadas.add(llamada);
        origen.llamadas.add(llamada);
        return llamada;
    }
    public Llamada registrarLlamadaNacional(Cliente origen
        Llamada llamada = new LlamadaNacional(origen.getNu
        llamadas.add(llamada);
        origen.llamadas.add(llamada);
        return llamada;
    }
public class EmpresaTest(){
//..
        this.sistema.registrarLlamadaNacional(emisorPerson
        this.sistema.registrarLlamadaInternacional(emisorP
        this.sistema.registrarLlamadaNacional(emisorPerson
```

```
        this.sistema.registrarLlamadaInternacional(emisorP
        this.sistema.registrarLlamadaNacional(emisorPerson
        this.sistema.registrarLlamadaInternacional(emisorP
        this.sistema.registrarLlamadaNacional(emisorPerson
        this.sistema.registrarLlamadaInternacional(emisorP
```

8. Mal olor: Nuevamente logica condicional evitable.

```
public class Cliente{
//..
    private double multiplicadorCostoPorTipoDeCliente(doub
        if (this.getTipo() == "fisica") {
            auxc -= auxc*Empresa.getDescuentoFis();
        } else if(this.getTipo() == "juridica") {
            auxc -= auxc*Empresa.getDescuentoJur();
        }
        return auxc;
```

Refactoringr: Replace conditional logic with polymorphism.

```
public class Cliente{
//..
    public abstract double multiplicadorCostoPorTipoDeClie
}
//Se añaden las subclases PersonaFisica y PersonaJuridica
public class PersonaFisica extends Cliente {
    private String DNI;

    public PersonaFisica() {
    }
    public PersonaFisica(List<Llamada> llamadas, String no
        super(llamadas, nombre, numeroTelefono);
        this.DNI = DNI;
    }
    public  double multiplicadorCostoPorTipoDeCliente(doub
        return auxc*Empresa.getDescuentoFis();
}
```

```
    public String getDNI() {
        return this.DNI;
    }
    public void setDNI(String dni) {
        this.DNI= dni;
    }

}
public class PersonaJuridica extends Cliente {
    private String cuit;

    public PersonaJuridica(List<Llamada> llamadas, String
        super(llamadas, nombre, numeroTelefono);
        this.cuit = cuit;
    }
    public PersonaJuridica() {
    }


    public  double multiplicadorCostoPorTipoDeCliente(doub
        return auxc*Empresa.getDescuentoJur();
}
    public String getCuit() {
        return cuit;
    }
    public void setCuit(String cuit) {
        this.cuit = cuit;
    }

}
//Se adapto la clase Empresa
public class Empresa{
//..
public Cliente registrarPersonaFisica(String data, String
        PersonaFisica var = new PersonaFisica();
            var.setNombre(nombre);
            String tel = this.obtenerNumeroLibre();
            var.setNumeroTelefono(tel);
```

```
            var.setDNI(data);
            clientes.add(var);
            return var;
            }

    public Cliente registrarPersonaJuridica(String data, S
        PersonaJuridica var = new PersonaJuridica();
        String tel = this.obtenerNumeroLibre();
        var.setNombre(nombre);
        var.setNumeroTelefono(tel);
        var.setCuit(data);
        clientes.add(var);
        return var;
    }
    }
    // Se adaptaron los test acordes a las nuevas clases
 class EmpresaTest{
        void testcalcularMontoTotalLlamadas() {
        Cliente emisorPersonaFisca = sistema.registrarPers
        Cliente remitentePersonaFisica = sistema.registrar
        Cliente emisorPersonaJuridica = sistema.registrarP
        Cliente remitentePersonaJuridica = sistema.registr
//..
}
```

9. Mal olor: Declaracion de atributo publico

```
public abstract class Cliente {
    public List<Llamada> llamadas = new ArrayList<Llamada>
    //..
```

Refactoring: Encapsulate field

```
public abstract class Cliente {
    private List<Llamada> llamadas = new ArrayList<Llamada

        public List<Llamada> getLlamadas() {
        return llamadas;
```

```
    }


    public void setLlamadas(List<Llamada> llamadas) {
        this.llamadas = llamadas;
    }
    //Se modifico tambien aquellos accesos a la variable l
```

10. Mal olor: No se utiliza el constructor, en cambio, inicializa una a una las variables del objeto

```
public class Empresa{
//..
    public Cliente registrarPersonaFisica(String data, Str
        PersonaFisica var = new PersonaFisica();
            var.setNombre(nombre);
            String tel = this.obtenerNumeroLibre();
            var.setNumeroTelefono(tel);
            var.setDNI(data);
            clientes.add(var);
            return var;
            }

    public Cliente registrarPersonaJuridica(String data, S
        PersonaJuridica var = new PersonaJuridica();
        String tel = this.obtenerNumeroLibre();
        var.setNombre(nombre);
        var.setNumeroTelefono(tel);
        var.setCuit(data);
        clientes.add(var);
        return var;
    }
```

Refactoring: Constructor initialization

```
    public class Empresa{
    //..
    public Cliente registrarPersonaFisica(String data, Str
```

```
            String tel = this.obtenerNumeroLibre();
            PersonaFisica var = new PersonaFisica(new Arra
            clientes.add(var);
            return var;
            }

    public Cliente registrarPersonaJuridica(String data, S
        String tel = this.obtenerNumeroLibre();
        PersonaJuridica var = new PersonaJuridica(new Arra
        clientes.add(var);
        return var;
    }
```

11. Mal olor: Variable temporal.

```
public class Empresa{
    //..
    public Cliente registrarPersonaFisica(String data, Str
            String tel = this.obtenerNumeroLibre();
            PersonaFisica var = new PersonaFisica(new Arra
            clientes.add(var);
            return var;
            }

    public Cliente registrarPersonaJuridica(String data, S
        String tel = this.obtenerNumeroLibre();
        PersonaJuridica var = new PersonaJuridica(new Arra
        clientes.add(var);
        return var;
    }
```

Refactoring: Replace temp with query

```
public class Empresa{
//..
public Cliente registrarPersonaFisica(String data, String
            PersonaFisica var = new PersonaFisica(new Arra
```

```
            clientes.add(var);
            return var;
            }

    public Cliente registrarPersonaJuridica(String data, S
        PersonaJuridica var = new PersonaJuridica(new Arra
        clientes.add(var);
        return var;
    }
```

12. Mal olor: Sentencia switch

```
public class GestorNumerosDisponibles{
//..
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.s
                lineas.remove(linea);
                return linea;
        }
        return null;
    }
```

Refactoring: Replace conditional logic with strategy pattern

```
//Se añadieron las siguentes clases:
```

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

public class ObtenerNumeroLibreUltimo implements IObtenerN

    @Override
    public String obtenerNumero(SortedSet<String> lineas)
        String linea= lineas.last();
        lineas.remove(linea);
        return linea;
    }
}

public class ObtenerNumeroLibrePrimero implements IObtener
    @Override
    public String obtenerNumero(SortedSet<String> lineas)
        String linea = lineas.first();
        lineas.remove(linea);
        return linea;
    }
}

public class ObtenerNumeroLibreRandom implements IObtenerN
    @Override
    public String obtenerNumero(SortedSet<String> lineas)
        String linea = new ArrayList<String>(lineas)
                .get(new Random().nextInt(lineas.size()));
        lineas.remove(linea);
        return linea;
    }
}
//Se modifico la clase GestorNumerosDisponibles
public class GestorNumerosDisponibles {
    private SortedSet<String> lineas = new TreeSet<String>
    private IObtenerNumeroStrategy tipoGenerador = new Obt

    public String obtenerNumeroLibre() {
```

```
        return tipoGenerador.obtenerNumero(this.getLineas(
    }

    public void cambiarTipoGenerador(IObtenerNumeroStrateg
        this.tipoGenerador = tipoGenerador;

    //.. }
    //Se adapto el test para el nuevo tipo de cambio de ge
        @Test
    void obtenerNumeroLibre() {
        // Por defecto es el ultimo
        assertEquals("2214444559", this.sistema.obtenerNum

        this.sistema.getGestorNumeros().cambiarTipoGenerad
        assertEquals("2214444554", this.sistema.obtenerNum

        this.sistema.getGestorNumeros().cambiarTipoGenerad
        assertNotNull(this.sistema.obtenerNumeroLibre());
    }
```

13. Mal olor: Codigo repetido

```
    public class Empresa{
    //..
    public Llamada registrarLlamadaInternacional(Cliente o
        Llamada llamada = new LlamadaInternacional(origen.
        llamadas.add(llamada);
        origen.getLlamadas().add(llamada);
        return llamada;
    }
    public Llamada registrarLlamadaNacional(Cliente origen
        Llamada llamada = new LlamadaNacional(origen.getNu
        llamadas.add(llamada);
        origen.getLlamadas().add(llamada);
        return llamada;
    }
```

Refactoring: Extract method

```
public class Empresa{
//..
    public Llamada registrarLlamadaInternacional(Cliente o
        Llamada llamada = new LlamadaInternacional(origen.
        agregarLlamadaAListas(origen, llamada);
        return llamada;
    }
    public Llamada registrarLlamadaNacional(Cliente origen
        Llamada llamada = new LlamadaNacional(origen.getNu
        agregarLlamadaAListas(origen, llamada);
        return llamada;
    }

    private void agregarLlamadaAListas(Cliente origen, Lla
        llamadas.add(llamada);
        origen.getLlamadas().add(llamada);
    }
```

14. Mal olor: Long method, variable temporal

```
public class GestorNumerosDisponibles{
//..
public boolean agregarNumeroTelefono(String str) {
            boolean encontre = this.getLineas().contains(s
            if (!encontre) {
                this.getLineas().add(str);
                encontre= true;
                return encontre;
            }
            else {
                encontre= false;
                return encontre;
            }
        }
```

Refactoring: Substitute algorithm

```
    public class GestorNumerosDisponibles{
    //..
    public boolean agregarNumeroTelefono(String str) {
        if(!this.getLineas().contains(str)) {
            return this.getLineas().add(str);
        }
        else return false;
    }
```

15. Mal olor: Nombre de variables poco claros

```
public class GestorNumerosDisponibles(){
//..
    public boolean agregarNumeroTelefono(String str) {
        if(!this.getLineas().contains(str)) {
            return this.getLineas().add(str);
        }
        else return false;
    }

    public class Empresa {

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

Refactoring: Rename field

```
public class Empresa {

    private static double descuentoPersonaJuridica = 0.15;
    private static double descuentoPersonaFisica = 0;
    //..
    }
    public class GestorNumerosDisponibles(){
    //..
    public boolean agregarNumeroTelefono(String numero) {
        if(!this.getLineas().contains(numero)) {
            return this.getLineas().add(numero);
```

```
        }
        else return false;
    }
```