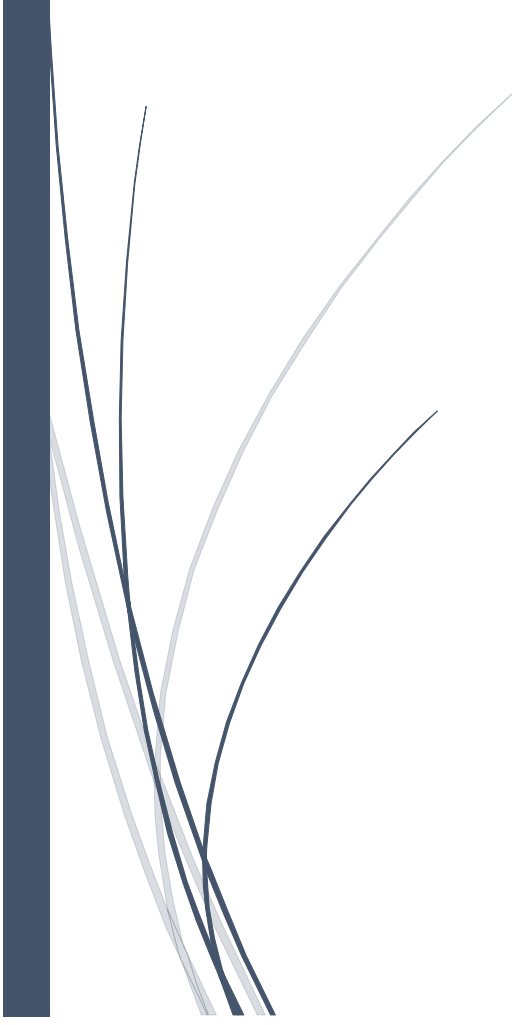
5/22/2023

Refactoring

Ejercicio 3 - OO2



Mal olor: **Nombres poco descriptivos**

Mal olor en Clase Persoona:

```
public class Persoona {
        public List<Llamada> lista1 = new ArrayList<Llamada>();
        public String t;
        public String nya;
        public String tel;
        public String cuit;
        public String doc;
        public Persoonal sis;

        public List<Llamada> getLista1() {
                return lista1;
        }
        public void setLista1(List<Llamada> lista1) {
                this.lista1 = lista1;
        }
        public String getT() {
                return t;
        }
        public void setT(String t) {
                this.t = t;
        }
        public String getNya() {
                return nya;
        }
        public void setNya(String nya) {
                this.nya = nya;
        }
        public String getTel() {
                return tel;
        }
        public void setTel(String tel) {
                this.tel = tel;
        }
        public String getCuit() {
                return cuit;
        }
        public void setCuit(String cuit) {
                this.cuit = cuit;
        }
        public String getDoc() {
                return doc;
        }
        public void setDoc(String doc) {
                this.doc = doc;
        }
}
```

Mal olor en Clase Llamada:

```
public int dur;
public String emisor;
public String remitente;


public Llamada(String s, String p, String p2, Persoona p3, int dur) {
        this.tipoDeLlamada = s;
        this.emisor= p;
        this.remitente= p2;
```

```
        this.dur = dur;
}

public void setEmisor(String emisor) {
        this.emisor = emisor;
}

public String getRemitente() {
        return remitente;
}

public void setRemitente(String remitente) {
        this.remitente = remitente;
}
```

Mal olor en Clase Persoonal:

```
public class Persoonal {
        List<Persoona> lista1 = new ArrayList<Persoona>();
        List<Llamada> lista2 = new ArrayList<Llamada>();
        GuiaTelefonica lista3 = new GuiaTelefonica();
        static double descuentoJur = 0.15;
        static double descuentoFis = 0;

        public boolean agregarTelefono(String str) {
                boolean encontre = lista3.guia.contains(str);
                if (!encontre) {
                        lista3.guia.add(str);
                        encontre= true;
                        return encontre;
                }
                else {
                        encontre= false;
                        return encontre;
                }
        }

        public Persoona registrarUsuario(String data, String nombre, String t) {
                Persoona var = new Persoona();
                if (t.equals("fisica")) {
                        var.setNombre(nombre);
                        String tel = lista3.guia.last();
                        lista3.guia.remove(tel);
                        var.setTipo(t);
                        var.setTelefono(tel);
                        var.setDocumento(data);
                }
                else if (t.equals("juridica")) {
                        String tel = lista3.guia.last();
                        lista3.guia.remove(tel);
                        var.nombre =nombre;
                        var.tipo =t;
                        var.telefono = tel;
                        var.cuit =data;
                }
                var.sistema = this;
                lista1.add(var);
                return var;

        }

        public boolean eliminarUsuario(Persoona p) {
```

```
                List<Persoona> l = p.sistema.lista1.stream().filter(persona -> persona
!= p).collect(Collectors.toList());
                boolean borre = false;
                if (l.size() < lista1.size()) {
                        this.lista1 = l;
                        this.lista3.guia.add(p.getTelefono());
                        borre = true;
                }
                return borre;

        }

        public Llamada registrarLlamada(Persoona q, Persoona q2, String t, int d) {
                Llamada x = new Llamada();
                x.tipoDeLlamada = t;
                x.setNumeroDeEmisor(q.telefono);
                x.setNumeroDeRemitente(q.getTelefono());
                x.duracion= d;
                lista2.add(x);
                q.llamadas.add(x);
                return x;

        }

        public double calcularMontoTotalLlamadas(Persoona p) {
                double c = 0;
                Persoona aux = null;
                for (Persoona pp : lista1) {
                        if (pp.telefono == p.getTelefono()) {
                                aux = pp;
                                break;
                        }
                } if (aux == null) return c;
                if (aux != null) {
                        for (Llamada l : aux.llamadas) {
                                double auxc = 0;
                                if (l.tipoDeLlamada == "nacional") {
                                        auxc += l.duracion *3 + (l.duracion*3*0.21);
                                } else if (l.tipoDeLlamada == "internacional") {
                                        auxc += l.duracion *200 + (l.duracion*200*0.21);
                                }

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

        public int cantidadDeUsuarios() {
                return lista1.size();
        }

        public boolean existeUsuario(Persoona persona) {
                return lista1.contains(persona);
        }

}
```

Refactoring: **Rename Field, Rename Method**

Refactoring en Clase Persoona:

```
public class Persoona {
	public List<Llamada> llamadas = new ArrayList<Llamada>();
	public String tipo;
	public String nombre;
	public String telefono;
	public String cuit;
	public String documento;
	public Persoonal sistema;

	public List<Llamada> getLlamadas() {
		return llamadas;
	}
	public void setLlamadas(List<Llamada> llamadas) {
		this.llamadas = llamadas;
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getTelefono() {
		return telefono;
	}
	public void setTelefono(String telefono) {
		this.telefono = telefono;
	}
	public String getCuit() {
		return cuit;
	}
	public void setCuit(String cuit) {
		this.cuit = cuit;
	}
	public String getDocumento() {
		return documento;
	}
	public void setDocumento(String documento) {
		this.documento = documento;
	}
}
```

Refactoring en Clase Llamada:

```
public int duracion;
public String numeroDeEmisor;
public String numeroDeRemitente;

public Llamada(String tipoDeLlamada, String numeroDeEmisor, String numeroDeRemitente,
Persoona persona, int duracion) {
	this.tipoDeLlamada = tipoDeLlamada;
	this.numeroDeEmisor= numeroDeEmisor;
```

```
        this.numeroDeRemitente = numeroDeRemitente;
        this.duracion = duracion;
}

public void setNumeroDeEmisor(String numeroDeEmisor) {
        this.numeroDeEmisor = numeroDeEmisor;
}

public String getNumeroDeRemitente() {
        return numeroDeRemitente;
}

public void setNumeroDeRemitente(String numeroDeRemitente) {
        this.numeroDeRemitente = numeroDeRemitente;
}
```

Refactoring: **Rename Field, Rename Variable y Rename Method**

Refactoring en Clase Persoonal:

```
public class Persoonal {
        List<Persoona> personas = new ArrayList<Persoona>();
        List<Llamada> llamadas = new ArrayList<Llamada>();
        GuiaTelefonica guiaTelefonica = new GuiaTelefonica();
        static double descuentoJuridica = 0.15;
        static double descuentoFisica = 0;

        public boolean agregarTelefono(String telefono) {
                boolean encontreTelefono = guiaTelefonica.guia.contains(telefono);
                if (!encontreTelefono) {
                        guiaTelefonica.guia.add(telefono);
                        encontreTelefono= true;
                        return encontreTelefono;
                }
                else {
                        encontreTelefono= false;
                        return encontreTelefono;
                }
        }

        public Persoona registrarUsuario(String identificador, String nombre, String
tipo) {
                Persoona persona = new Persoona();
                if (tipo.equals("fisica")) {
                        persona.setNombre(nombre);
                        String telefono = guiaTelefonica.guia.last();
                        guiaTelefonica.guia.remove(telefono);
                        persona.setTipo(tipo);
                        persona.setTelefono(telefono);
                        persona.setDocumento(identificador);
                }
                else if (tipo.equals("juridica")) {
                        String telefono = guiaTelefonica.guia.last();
                        guiaTelefonica.guia.remove(telefono);
                        persona.nombre =nombre;
                        persona.tipo =tipo;
                        persona.telefono = telefono;
                        persona.cuit =identificador;
                }
                persona.sistema = this;
                personas.add(persona);
                return persona;
```

```
	}

	public boolean eliminarUsuario(Persoona usuario) {
		List<Persoona> listaSinUsuario =
usuario.sistema.personas.stream().filter(persona -> persona !=
usuario).collect(Collectors.toList());
		boolean borreUsuario = false;
		if (listaSinUsuario.size() < personas.size()) {
			this.personas = listaSinUsuario;
			this.guiaTelefonica.guia.add(usuario.getTelefono());
			borreUsuario = true;
		}
		return borreUsuario;

	}

	public Llamada registrarLlamada(Persoona emisor, Persoona remitente, String
tipo, int duracion) {
		Llamada llamada = new Llamada();
		llamada.tipoDeLlamada = tipo;
		llamada.setNumeroDeEmisor(emisor.telefono);
		llamada.setNumeroDeRemitente(remitente.getTelefono());
		llamada.duracion= duracion;
		llamadas.add(llamada);
		emisor.llamadas.add(llamada);
		return llamada;

	}

	public double calcularMontoTotalLlamadas(Persoona persona) {
		double costo = 0;
		Persoona personaEnPersoonal = null;
		for (Persoona usuario : personas) {
			if (usuario.telefono == persona.getTelefono()) {
				personaEnPersoonal = usuario;
				break;
			}
		} if (personaEnPersoonal == null) return costo;
		if (personaEnPersoonal != null) {
			for (Llamada llamada : personaEnPersoonal.llamadas) {
				double costoAuxiliar = 0;
				if (llamada.tipoDeLlamada == "nacional") {
					costoAuxiliar += llamada.duracion *3 +
(llamada.duracion*3*0.21);
				} else if (llamada.tipoDeLlamada == "internacional") {
					costoAuxiliar += llamada.duracion *200 +
(llamada.duracion*200*0.21);
				}

				if (personaEnPersoonal.tipo == "fisica") {
					costoAuxiliar -= costoAuxiliar*descuentoFisica;
				} else if(personaEnPersoonal.tipo == "juridica") {
					costoAuxiliar -= costoAuxiliar*descuentoJuridica;
				}
				costo += costoAuxiliar;
			}
		}
		return costo;
	}

	public int cantidadDeUsuarios() {
		return personas.size();
```

```
        }

        public boolean existeUsuario(Persoona persona) {
                return personas.contains(persona);
        }

}
```

Mal olor: **Declaración de atributo publico**

Mal olor en Clase GuiaTelefonica:

```
public SortedSet<String> guia = new TreeSet<String>();
```

Mal olor en Clase Llamada:

```
protected String tipoDeLlamada;
public int duracion;
```

Mal olor en Clase Persoona:

```
public List<Llamada> llamadas = new ArrayList<Llamada>();
public String tipo;
public String nombre;
public String telefono;
public String cuit;
public String documento;
public Persoonal sistema;
```

Refactoring: **Encapsulate Field**

Refactoring en Clase GuiaTelefonica:

```
public class GuiaTelefonica {
        private SortedSet<String> guia = new TreeSet<String>();

        public SortedSet<String> getGuia() {
                return guia;
        }

        public void setGuia(SortedSet<String> guia) {
                this.guia = guia;
        }
}
```

Refactoring en Clase Llamada:

```
private String tipoDeLlamada;
private int duracion;
// ...
public int getDuracion() {
        return duracion;
}

public void setDuracion(int duracion) {
        this.duracion = duracion;
}
```

Refactoring en Clase Persoona:

```
private List<Llamada> llamadas = new ArrayList<Llamada>();
private String tipo;
private String nombre;
private String telefono;
private String cuit;
private String documento;
private Persoonal sistema;
// ...
public Persoonal getSistema() {
        return sistema;
}
public void setSistema(Persoonal sistema) {
        this.sistema = sistema;
}
```

Mal olor: **No hay declaración de visibilidad**

Al no declarar la visibilidad se coloca por defecto el alcance package private.

Mal olor en Clase Persoonal:

```
List<Persoona> personas = new ArrayList<Persoona>();
List<Llamada> llamadas = new ArrayList<Llamada>();
GuiaTelefonica guiaTelefonica = new GuiaTelefonica();
static double descuentoJuridica = 0.15;
static double descuentoFisica = 0;
```

Refactoring: **Encapsulate Field**

Refactoring en Clase Persoonal:

```
private List<Persoona> personas = new ArrayList<Persoona>();
private List<Llamada> llamadas = new ArrayList<Llamada>();
private GuiaTelefonica guiaTelefonica = new GuiaTelefonica();
private static double descuentoJuridica = 0.15;
private static double descuentoFisica = 0;

public List<Persoona> getPersonas() {
        return personas;
}

public void setPersonas(List<Persoona> personas) {
        this.personas = personas;
}

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

public void setLlamadas(List<Llamada> llamadas) {
        this.llamadas = llamadas;
}

public GuiaTelefonica getGuiaTelefonica() {
        return guiaTelefonica;
}

public void setGuiaTelefonica(GuiaTelefonica guiaTelefonica) {
        this.guiaTelefonica = guiaTelefonica;
}
```

Mal olor: **Generalidad especulativa / Data Class / Feature Envy**

Guía telefónica únicamente provee de una colección de teléfonos que esta siendo usada por personal, lo que la hace una clase perezosa y contenedora de datos. Dado esto la clase Persoonal tiene envidia de atributos de la clase guía telefónica.

Mal olor en Clase Guia Telefonica:

```
public class GuiaTelefonica {
        private SortedSet<String> guia = new TreeSet<String>();

        public SortedSet<String> getGuia() {
                return guia;
        }

        public void setGuia(SortedSet<String> guia) {
                this.guia = guia;
        }
}
```

Mal olor en Clase Persoonal:

```
private GuiaTelefonica guiaTelefonica = new GuiaTelefonica();
// ...
guiaTelefonica.getGuia().add(telefono);
```

Refactoring: **Inline Class (Move Method / Move Fields)**

Primero se mueve el campo de Guía telefónica (contiene la colección de teléfonos) y luego sus métodos subyacentes. Por últimos eliminamos la clase perezosa ya que no es necesaria.

Refactoring en Clase Persoonal:

```
private SortedSet<String> guiaTelefonica = new TreeSet<String>();
// ...
guiaTelefonica.add(telefono);
// ...
public SortedSet<String> getGuiaTelefonica() {
        return guiaTelefonica;
}

public void setGuiaTelefonica(SortedSet<String> guiaTelefonica) {
        this.guiaTelefonica = guiaTelefonica;
}
```

Mal olor: **Asociación bidireccional innecesaria**

La clase Persoona conoce al sistema (Persoonal) pero no hace uso de sus funciones y guardar este objeto no tiene sentido lógico en el problema.

Mal olor en Clase Persoona:

```
private List<Llamada> llamadas = new ArrayList<Llamada>();
private String tipo;
private String nombre;
private String telefono;
private String cuit;
private String documento;
private Persoonal sistema;
//...
public Persoonal getSistema() {
```

```
        return sistema;
}
public void setSistema(Persoonal sistema) {
        this.sistema = sistema;
}
```

Mal olor en Clase Persoonal:

```
public Persoona registrarUsuario(String identificador, String nombre, String tipo) {
        //...
        persona.setSistema(this);
        //...
}

public boolean eliminarUsuario(Persoona usuario) {
      List<Persoona> listaSinUsuario =
      usuario.getSistema().personas.stream().filter(persona -> persona !=
      usuario).collect(Collectors.toList());
        //...
}
```

Refactoring: **Change Bidirectional Association to Unidirectional**

Refactoring en Clase Persoona:

```
private List<Llamada> llamadas = new ArrayList<Llamada>();
private String tipo;
private String nombre;
private String telefono;
private String cuit;
private String documento;
```

Refactoring en Clase Persoonal:

```
public boolean eliminarUsuario(Persoona usuario) {
        List<Persoona> listaSinUsuario = personas.stream().filter(persona -> persona !=
        usuario).collect(Collectors.toList());
        //...
}
```

Mal olor: **No existe constructor para inicializar los objetos de la clase con valores**

En la clase Persoona no se ha implementado un constructor para que en el momento de instanciación sus variables internas tengan valores útiles, trabajo que hace luego la clase Persoonal a través de setters (le da mayor volumen al método).

Mal olor en Clase Persoonal:

```
public Persoona registrarUsuario(String identificador, String nombre, String tipo) {
        Persoona persona = new Persoona();
        if (tipo.equals("fisica")) {
                persona.setNombre(nombre);
                String telefono = guiaTelefonica.last();
                guiaTelefonica.remove(telefono);
                persona.setTipo(tipo);
                persona.setTelefono(telefono);
                persona.setDocumento(identificador);
        }
        else if (tipo.equals("juridica")) {
                String telefono = guiaTelefonica.last();
```

```
                guiaTelefonica.remove(telefono);
                persona.setNombre(nombre);
                persona.setTipo(tipo);
                persona.setTelefono(telefono);
                persona.setCuit(identificador);
        }
        persona.setSistema(this);
        personas.add(persona);
        return persona;
}
```

Refactoring: **Constructor Initialization**

Source: https://martinfowler.com/bliki/ConstructorInitialization.html

No existe este refactoring como tal, pero consideramos que tener un constructor ofrece ciertas ventajas como: la inicialización de objetos y establecer valores iniciales, mantener la encapsulación y garantizar la integridad de los datos, mejorar la claridad y legibilidad del código al especificar valores necesarios y permite tener un objeto en un estado razonablemente bien formado listo para usarse.

Refactoring en Clase Persoona:

```
public class Persoona {
        private List<Llamada> llamadas = new ArrayList<Llamada>();
        //...
        public Persoona(String tipo, String nombre, String telefono, String cuit,
      String documento) {
                this.tipo = tipo;
                this.nombre = nombre;
                this.telefono = telefono;
                this.cuit = cuit;
                this.documento = documento;
        }
        //...
}
```

Refactoring en Clase Persoonal:

```
public Persoona registrarUsuario(String identificador, String nombre, String tipo) {
        Persoona persona = new Persoona();
        if (tipo.equals("fisica")) {
                String telefono = guiaTelefonica.last();
                guiaTelefonica.remove(telefono);
                persona = new Persoona(tipo, nombre, telefono, "", identificador);
                personas.add(persona);
        }
        else if (tipo.equals("juridica")) {
                String telefono = guiaTelefonica.last();
                guiaTelefonica.remove(telefono);
                persona = new Persoona(tipo, nombre, telefono, identificador, "");
                personas.add(persona);
        }
        return persona;
}
```

Mal olor: **No se utiliza el parámetro**

En el constructor de la clase Llamada no se utiliza el parámetro persona.

Mal olor en Clase Llamada:

```
public Llamada(String tipoDeLlamada, String numeroDeEmisor, String numeroDeRemitente,
Persoona persona, int duracion) {
        this.tipoDeLlamada = tipoDeLlamada;
        this.numeroDeEmisor= numeroDeEmisor;
        this.numeroDeRemitente = numeroDeRemitente;
        this.duracion = duracion;
}
```

Refactoring: **Remove Parameter**

Refactoring en Clase Llamada

```
public Llamada(String tipoDeLlamada, String numeroDeEmisor, String numeroDeRemitente,
int duracion) {
        this.tipoDeLlamada = tipoDeLlamada;
        this.numeroDeEmisor = numeroDeEmisor;
        this.numeroDeRemitente = numeroDeRemitente;
        this.duracion = duracion;
}
```

Mal olor: **No se utiliza el constructor que existe para inicializar los objetos de la clase con valores**

En la clase Llamada si se ha implementado un constructor, pero no se está utilizando en la clase Persoonal sino que se inicializan los valores a través de setters (le da mayor volumen al método).

Mal olor en Clase Persoonal:

```
public Llamada registrarLlamada(Persoona numeroDeEmisor, Persoona remitente, String
tipo, int duracion) {
        Llamada llamada = new Llamada();
        llamada.setTipoDeLlamada(tipo);
        llamada.setNumeroDeEmisor(emisor.getTelefono());
        llamada.setNumeroDeRemitente(remitente.getTelefono());
        llamada.setDuracion(duracion);
        llamadas.add(llamada);
        emisor.getLlamadas().add(llamada);
        return llamada;
}
```

Refactoring: **Constructor Initialization**

Refactoring en Clase Persoonal:

```
public Llamada registrarLlamada(Persoona emisor, Persoona remitente, String tipo, int
duracion) {
        Llamada llamada = new Llamada(tipo, emisor.getTelefono(),
remitente.getTelefono(), duracion);
        llamadas.add(llamada);
        emisor.getLlamadas().add(llamada);
        return llamada;
}
```

Mal olor: **Long Method**

Mal olor en Clase Persoonal:

```
public double calcularMontoTotalLlamadas(Persoona persona) {
```

```
        double costo = 0;
        Persoona personaEnPersoonal = null;
        for (Persoona usuario : personas) {
                if (usuario.getTelefono() == persona.getTelefono()) {
                        personaEnPersoonal = usuario;
                        break;
                }
        } if (personaEnPersoonal == null) return costo;
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        double costoAuxiliar = 0;
                        if (llamada.getTipoDeLLamada() == "nacional") {
                                costoAuxiliar += llamada.getDuracion()*3 +
(llamada.getDuracion()*3*0.21);
                        } else if (llamada.getTipoDeLLamada() == "internacional") {
                        costoAuxiliar += llamada.getDuracion() *200 +
(llamada.getDuracion()*200*0.21);
                                }
                        if (personaEnPersoonal.getTipo() == "fisica") {
                                costoAuxiliar -= costoAuxiliar*descuentoFisica;
                        } else if(personaEnPersoonal.getTipo() == "juridica") {
                                costoAuxiliar -= costoAuxiliar*descuentoJuridica;
                        }
                        costo += costoAuxiliar;
                }
        }
        return costo;
}
```

Refactoring: **Extract Method**

Este método largo se puede dividir en otros tres más pequeños con funcionalidades atómicas.

Refactoring en Clase Persoonal:

```
private Persoona buscarPersoonaEnPersoonal(Persoona persona) {
        for (Persoona usuario : personas) {
                if (usuario.getTelefono() == persona.getTelefono()) {
                        return usuario;
                }
        }
        return null;
}

private double calcularCostoLlamada(Llamada llamada) {
        if (llamada.getTipoDeLLamada() == "nacional") {
                return llamada.getDuracion()*3 + (llamada.getDuracion()*3*0.21);
        } else if (llamada.getTipoDeLLamada() == "internacional") {
                return llamada.getDuracion() *200 + (llamada.getDuracion()*200*0.21);
        }
        return 0;
}

private double calcularDescuentoPersoona(Persoona personaEnPersoonal, double costo) {
        if (personaEnPersoonal.getTipo() == "fisica") {
                return costo*descuentoFisica;
        } else if(personaEnPersoonal.getTipo() == "juridica") {
                return costo*descuentoJuridica;
        }
        return 0;
}
```

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        double costo = 0;
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal == null) return costo;
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        costo += calcularCostoLlamada(llamada);
                        costo -= calcularDescuentoPersoona(persona, costo);
                }
        }
        return costo;
}
```

Mal olor: **Feature Envy y Data Class**

La clase Persoonal está accediendo y calculando con datos de otros objetos, conviertiendolos en clases de datos ya que no tienen otra funcionalidad (Llamada y Persoona).

Mal olor en Clase Persoonal:

```
private static double descuentoJuridica = 0.15;
private static double descuentoFisica = 0;
//...

private double calcularCostoLlamada(Llamada llamada) {
        if (llamada.getTipoDeLlamada() == "nacional") {
                return llamada.getDuracion()*3 + (llamada.getDuracion()*3*0.21);
        } else if (llamada.getTipoDeLlamada() == "internacional") {
                return llamada.getDuracion() *200 + (llamada.getDuracion()*200*0.21);
        }
        return 0;
}

private double calcularDescuentoPersoona(Persoona personaEnPersoonal, double costo) {
        if (personaEnPersoonal.getTipo() == "fisica") {
                return costo*descuentoFisica;
        } else if(personaEnPersoonal.getTipo() == "juridica") {
                return costo*descuentoJuridica;
        }
        return 0;
}
```

Refactoring: **Move Method y Move Field**

Refactoring Clase Persoona:

Movemos los campos de descuentos a la clase Persoona y el metodo calcularDescuentoPersoona() y lo cambiamos a público.

```
public class Persoona {
        //...
        private static double descuentoJuridica = 0.15;
        private static double descuentoFisica = 0;

        public double calcularDescuentoPersoona(double costo) {
                if (this.getTipo() == "fisica") {
                        return costo*descuentoFisica;
                } else if(this.getTipo() == "juridica") {
                        return costo*descuentoJuridica;
                }
```

```
                return 0;
        }
        //...
}
```

Refactoring Clase Llamada:

Movemos el metodo calcularCostoLlamada() y lo cambiamos a público.

```
public class Llamada {
        //...
        public double calcularCostoLlamada() {
                if (this.getTipoDeLlamada() == "nacional") {
                        return this.getDuracion()*3 + (this.getDuracion()*3*0.21);
                } else if (this.getTipoDeLlamada() == "internacional") {
                        return this.getDuracion() *200 + (this.getDuracion()*200*0.21);
                }
                return 0;
        }
        //...
}
```

Refactoring Clase Persoonal:

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        double costo = 0;
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal == null) return costo;
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        double costoAuxiliar= llamada.calcularCostoLlamada();
                        costo += costoAuxiliar;
                        costo -= persona.calcularDescuentoPersoona(costoAuxiliar);
                }
        }
        return costo;
}
```

Mal olor: **Switch Statements**

Utilizar una sola clase de persona para usuarios de dos tipos de diferentes que comparten parte de su estructura conlleva a que se usen variables innecesarias en tal clase, repetición de código y uso de if solucionable con herencia.

Mal olor en Clase Persoona:

```
private static double descuentoJuridica = 0.15;
private static double descuentoFisica = 0;

public double calcularDescuentoPersoona(double costo) {
        if (this.getTipo() == "fisica") {
                return costo*descuentoFisica;
        } else if(this.getTipo() == "juridica") {
                return costo*descuentoJuridica;
        }
        return 0;
}
```

Mal olor en Clase Persoonal:

```
public Persoona registrarUsuario(String identificador, String nombre, String tipo) {
```

```
        Persoona persona = new Persoona();
        if (tipo.equals("fisica")) {
                String telefono = guiaTelefonica.last();
                guiaTelefonica.remove(telefono);
                persona = new Persoona(tipo, nombre, telefono, "", identificador);
                personas.add(persona);
        }
        else if (tipo.equals("juridica")) {
                String telefono = guiaTelefonica.last();
                guiaTelefonica.remove(telefono);
                persona = new Persoona(tipo, nombre, telefono, identificador, "");
                personas.add(persona);
        }
        return persona;
}
```

Refactoring: **Replace Type code with Subclasses y Replace Conditional with Polimorfism**

Creamos subclases para cada tipo específico, heredando de la clase base la estructura en comun y agregando los atributos y comportamientos específicos de cada tipo. Para ello es necesario hacer un push down field de los descuentos a sus clases respectivas y un push down method del cálculo del descuento.

Refactoring en Clase Persoona:

```
public abstract class Persoona {
        private List<Llamada> llamadas = new ArrayList<Llamada>();
        private String nombre;
        private String telefono;

        public Persoona() {};

        public Persoona(String nombre, String telefono) {
                this.nombre = nombre;
                this.telefono = telefono;
        }

        public abstract double calcularDescuentoPersoona(double costo);

        //...
}
```

Refactoring en nueva Clase PersoonaJuridica:

```
public class PersoonaJuridica extends Persoona {
        private String cuit;
        private static double descuentoJuridica = 0.15;

        public PersoonaJuridica(String nombre, String telefono, String cuit) {
                super(nombre, telefono);
                this.cuit = cuit;
        }

        public String getCuit() {
                return cuit;
        }

        public void setCuit(String cuit) {
                this.cuit = cuit;
        }
        //...
```

```
        public static double getDescuentoJuridica() {
                return descuentoJuridica;
        }

        public static void setDescuentoJuridica(double descuentoJuridica) {
                PersoonaJuridica.descuentoJuridica = descuentoJuridica;
        }

        public double calcularDescuentoPersoona(double costo) {
                return costo*PersoonaJuridica.getDescuentoJuridica();
        }
}
```

Refactoring en nueva Clase PersoonaFisica:

```
public class PersoonaFisica extends Persoona {
        private String documento;
        private static double descuentoFisica = 0;

        public PersoonaFisica(String nombre, String telefono, String documento) {
                super(nombre, telefono);
                this.documento = documento;
        }

        public String getDocumento() {
                return documento;
        }

        public void setDocumento(String documento) {
                this.documento = documento;
        }

        //...

        public static double getDescuentoFisica() {
                return descuentoFisica;
        }

        public static void setDescuentoFisica(double descuentoFisica) {
                PersoonaFisica.descuentoFisica = descuentoFisica;
        }

        public double calcularDescuentoPersoona(double costo) {
                return costo*PersoonaFisica.getDescuentoFisica();
        }
}
```

Refactoring en Clase Persoonal:

Por los cambios anteriores la instanciación de las personas debe ser modificada en la clase Persoonal.

```
public Persoona registrarUsuarioFisica(String documento, String nombre) {
        String telefono = guiaTelefonica.last();
        guiaTelefonica.remove(telefono);
        PersoonaFisica persona = new PersoonaFisica(nombre, telefono, documento);
        personas.add(persona);
        return persona;
}

public Persoona registrarUsuarioJuridica(String cuit, String nombre) {
        String telefono = guiaTelefonica.last();
        guiaTelefonica.remove(telefono);
```

```
        PersoonaJuridica persona = new PersoonaJuridica(nombre, telefono, cuit);
        personas.add(persona);
        return persona;
}
```

Al aplicar este refactoring estamos obligados a cambiar el set up del test.

Antes PersoonalTest:

```
@BeforeEach
public void setUp() {
        //...
        this.emisorPersonaFisca = sistema.registrarUsuario("11555666", "Marcelo
Tinelli" , "fisica");
        this.remitentePersonaFisica = sistema.registrarUsuario("00000001", "Mirtha
Legrand" , "fisica");          this.emisorPersonaJuridica =
sistema.registrarUsuario("17555222", "Felfort" , "juridica");
        this.remitentePersonaJuridica = sistema.registrarUsuario("25765432",
"Moovistar" , "juridica");
        //...
}
```

Despues PersoonalTest:

```
@BeforeEach
public void setUp() {
        //...
        this.emisorPersonaFisca = sistema.registrarUsuarioFisica("11555666", "Marcelo
Tinelli");             this.remitentePersonaFisica =
sistema.registrarUsuarioFisica("00000001", "Mirtha Legrand");
        this.emisorPersonaJuridica = sistema.registrarUsuarioJuridica("17555222",
"Felfort");             this.remitentePersonaJuridica =
sistema.registrarUsuarioJuridica("25765432", "Moovistar");
        //...
}
```

Mal olor: **Switch Statements**

Utilizar una sola clase de llamada para dos tipos diferentes que comparten parte de su estructura conlleva a que se usen variables innecesarias en tal clase, repetición de código y uso de if solucionable con herencia.

Mal olor en Clase Llamada:

```
public double calcularCostoLlamada() {
        if (this.getTipoDeLlamada() == "nacional") {
                return this.getDuracion()*3 + (this.getDuracion()*3*0.21);
        } else if (this.getTipoDeLlamada() == "internacional") {
                return this.getDuracion() *200 + (this.getDuracion()*200*0.21);
        }
        return 0;
}
```

Mal olor en Clase Llamada:

```
public Llamada registrarLlamada(Persoona emisor, Persoona remitente, String tipo, int
duracion) {
        Llamada llamada = new Llamada(tipo, emisor.getTelefono(),
remitente.getTelefono(), duracion);
        llamadas.add(llamada);
        emisor.getLlamadas().add(llamada);
        return llamada;
}
```

Refactoring: **Replace Type code with Subclasses y Replace Conditional with Polimorfism**

Creamos subclases para cada tipo específico, heredando de la clase base la estructura en comun y agregando el comportamiento específico de cada tipo. Para ello es necesario hacer un push down method del cálculo de costo de llamada y remover el atributo tipoDeLlamada.

Refactoring en Clase Llamada:

```
public abstract class Llamada {
	private String numeroDeEmisor;
	private String numeroDeRemitente;
	private int duracion;

	public Llamada() {

	}

	public Llamada(String numeroDeEmisor, String numeroDeRemitente, int duracion) {
		this.numeroDeEmisor= numeroDeEmisor;
		this.numeroDeRemitente = numeroDeRemitente;
		this.duracion = duracion;
	}

	public abstract double calcularCostoLlamada() {};

	//...
}
```

Refactoring en nueva Clase LlamadaNacional:

```
public class LlamadaNacional extends Llamada {

	public LlamadaNacional(String numeroDeEmisor, String numeroDeRemitente, int
duracion) {
		super(numeroDeEmisor, numeroDeRemitente, duracion);
	}

	public double calcularCostoLlamada() {
		return this.getDuracion()*3 + (this.getDuracion()*3*0.21);
	}

}
```

Refactoring en nueva Clase LlamadaInternacional:

```
public class LlamadaInternacional extends Llamada {

	public LlamadaInternacional(String numeroDeEmisor, String numeroDeRemitente,
int duracion) {
		super(numeroDeEmisor, numeroDeRemitente, duracion);
	}

	public double calcularCostoLlamada() {
		return this.getDuracion() *200 + (this.getDuracion()*200*0.21);
	}
}
```

Refactoring en Clase Persoonal:

```
public Llamada registrarLlamadaNacional(Persoona emisor, Persoona remitente, int
duracion) {
        Llamada llamada = new LlamadaNacional(emisor.getTelefono(),
remitente.getTelefono(), duracion);
        llamadas.add(llamada);
        emisor.getLlamadas().add(llamada);
        return llamada;
}

public Llamada registrarLlamadaInternacional(Persoona emisor, Persoona remitente, int
duracion) {
        Llamada llamada = new LlamadaInternacional(emisor.getTelefono(),
remitente.getTelefono(), duracion);
        llamadas.add(llamada);
        emisor.getLlamadas().add(llamada);
        return llamada;
}
```

Al aplicar este refactoring estamos obligados a cambiar el set up del test.

Antes PersoonalTest:

```
@BeforeEach
public void setUp() {
        //...
        this.sistema.registrarLlamada(emisorPersonaJuridica, remitentePersonaFisica,
"nacional", 10);
        this.sistema.registrarLlamada(emisorPersonaJuridica, remitentePersonaFisica,
"internacional", 8);
        this.sistema.registrarLlamada(emisorPersonaJuridica, remitentePersonaJuridica,
"nacional", 5);
        this.sistema.registrarLlamada(emisorPersonaJuridica, remitentePersonaJuridica,
"internacional", 7);
        this.sistema.registrarLlamada(emisorPersonaFisca, remitentePersonaFisica,
"nacional", 15);
        this.sistema.registrarLlamada(emisorPersonaFisca, remitentePersonaFisica,
"internacional", 45);
        this.sistema.registrarLlamada(emisorPersonaFisca, remitentePersonaJuridica,
"nacional", 13);
        this.sistema.registrarLlamada(emisorPersonaFisca, remitentePersonaJuridica,
"internacional", 17);
}
```

Despues PersoonalTest:

```
@BeforeEach
public void setUp() {
        //...
        this.sistema.registrarLlamadaNacional(emisorPersonaJuridica,
remitentePersonaFisica, 10);
        this.sistema.registrarLlamadaInternacional(emisorPersonaJuridica,
remitentePersonaFisica, 8);
        this.sistema.registrarLlamadaNacional(emisorPersonaJuridica,
remitentePersonaJuridica, 5);
        this.sistema.registrarLlamadaInternacional(emisorPersonaJuridica,
remitentePersonaJuridica, 7);
        this.sistema.registrarLlamadaNacional(emisorPersonaFisca,
remitentePersonaFisica, 15);
        this.sistema.registrarLlamadaInternacional(emisorPersonaFisca,
remitentePersonaFisica, 45);
        this.sistema.registrarLlamadaNacional(emisorPersonaFisca,
remitentePersonaJuridica, 13);
```

```
        this.sistema.registrarLlamadaInternacional(emisorPersonaFisca,
remitentePersonaJuridica, 17);
}
```

Mal olor: **Método extenso y complejo**

Mal olor en Clase Persoonal:

```
public class Persoonal {
        //...
        public boolean agregarTelefono(String telefono) {
                boolean encontreTelefono = guiaTelefonica.contains(telefono);
                if (!encontreTelefono) {
                        guiaTelefonica.add(telefono);
                        encontreTelefono= true;
                        return encontreTelefono;
                }
                else {
                        encontreTelefono= false;
                        return encontreTelefono;
                }
        }

        //...
        public boolean eliminarUsuario(Persoona usuario) {
                List<Persoona> listaSinUsuario = personas.stream().filter(persona ->
persona != usuario).collect(Collectors.toList());
                boolean borreUsuario = false;
                if (listaSinUsuario.size() < personas.size()) {
                        this.personas = listaSinUsuario;
                        this.guiaTelefonica.add(usuario.getTelefono());
                        borreUsuario = true;
                }
                return borreUsuario;

        }
        //...
}
```

Refactoring: **Substitute Algorithm**

Refactoring en Clase Persoonal:

```
public class Persoonal {
        //...
        public boolean agregarTelefono(String telefono) {
                if (!guiaTelefonica.contains(telefono)) {
                        guiaTelefonica.add(telefono);
                        return true;
                }
                return false;
        }

        //...

        public boolean eliminarUsuario(Persoona usuario) {
                boolean borreUsuario = personas.remove(usuario);
                if (borreUsuario) this.guiaTelefonica.add(usuario.getTelefono());
                return borreUsuario;
        }

        //...
```

```
}
```

Mal olor: **Dead Code**

La relación entre Persoonal y múltiples llamadas no es necesaria, ya que las conocemos a través del usuario y no se hace uso de la misma en Persoonal.

Mal olor en Clase Persoonal:

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

Refactoring: **Remove Dead Code**

Refactoring en Clase Persoonal:

Se elimino el campo y sus getters y setters.

Mal olor: **Feature Envy**

La clase Persoonal al agregar una llamada está manipulando una colección que no le pertenece (llamadas del usuario). En realidad, una llamada debe ser creada y agregada por la clase Persoona.

Mal olor en Clase Persoonal:

```
public Llamada registrarLlamadaNacional(Persoona emisor, Persoona remitente, int
duracion) {
        Llamada llamada = new LlamadaNacional(emisor.getTelefono(),
remitente.getTelefono(), duracion);
        emisor.getLlamadas().add(llamada);
        return llamada;
}

public Llamada registrarLlamadaInternacional(Persoona emisor, Persoona remitente, int
duracion) {
        Llamada llamada = new LlamadaInternacional(emisor.getTelefono(),
remitente.getTelefono(), duracion);
        emisor.getLlamadas().add(llamada);
        return llamada;
}
```

Refactoring: **Move Method**

Refactoring en Clase Persoona:

```
public Llamada registrarLlamadaInternacional(Persoona remitente, int duracion) {
        Llamada llamada = new LlamadaInternacional(this.getTelefono(),
remitente.getTelefono(), duracion);
        this.llamadas.add(llamada);
        return llamada;
}
```

Refactoring en Clase Persoonal:

```
public Llamada registrarLlamadaNacional(Persoona emisor, Persoona remitente, int
duracion) {
        return emisor.registrarLlamadaNacional(remitente, duracion);
}

public Llamada registrarLlamadaInternacional(Persoona emisor, Persoona remitente, int
duracion) {
        return emisor.registrarLlamadaInternacional(remitente, duracion);
}
```

Mal olor: **Duplicate code**

Obtener el ultimo número de la guía telefónica es código duplicado

```
public Persoona registrarUsuarioFisica(String documento, String nombre) {
        String telefono = guiaTelefonica.last();
        guiaTelefonica.remove(telefono);
        PersoonaFisica persona = new PersoonaFisica(nombre, telefono, documento);
        personas.add(persona);
        return persona;
}

public Persoona registrarUsuarioJuridica(String cuit, String nombre) {
        String telefono = guiaTelefonica.last();
        guiaTelefonica.remove(telefono);
        PersoonaJuridica persona = new PersoonaJuridica(nombre, telefono, cuit);
        personas.add(persona);
        return persona;
}
```

Refactoring: **Extract Method y Replace Temp with Query**

```
private String extraerUltimoTelefono() {
        String telefono = guiaTelefonica.last();
        guiaTelefonica.remove(telefono);
        return telefono;
}

public Persoona registrarUsuarioFisica(String documento, String nombre) {
        PersoonaFisica persona = new PersoonaFisica(nombre, extraerUltimoTelefono(),
documento);
        personas.add(persona);
        return persona;
}

public Persoona registrarUsuarioJuridica(String cuit, String nombre) {
        PersoonaJuridica persona = new PersoonaJuridica(nombre,
extraerUltimoTelefono(), cuit);
        personas.add(persona);
        return persona;
}
```

Mal olor: **Dead Code**

La línea “if (personaEnPersoonal == null) return costo” no es necesaria ya que si no se al segundo if, el costo retornado va a ser siempre 0.

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        double costo = 0;
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal == null) return costo;
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        double costoAuxiliar= llamada.calcularCostoLlamada();
                        costo += costoAuxiliar;
                        costo -= persona.calcularDescuentoPersoona(costoAuxiliar);
                }
        }
        return costo;
}
```

Refactoring: **Remove Dead Code**

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        double costo = 0;
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        double costoAuxiliar= llamada.calcularCostoLlamada();
                        costo += costoAuxiliar;
                        costo -= persona.calcularDescuentoPersoona(costoAuxiliar);
                }
        }
        return costo;
}
```

Mal olor: **Feature Envy**

La clase Persoonal utiliza las llamadas de cada usuario para calcular el costo de estas (propias del usuario) pero tal funcionalidad debería ser propia de la interfaz de la clase Persoona.

Mal olor en Clase Persoonal:

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        double costo = 0;
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal != null) {
                for (Llamada llamada : personaEnPersoonal.getLlamadas()) {
                        double costoAuxiliar= llamada.calcularCostoLlamada();
                        costo += costoAuxiliar;
                        costo -= persona.calcularDescuentoPersoona(costoAuxiliar);
                }
        }
        return costo;
}
```

Refactoring: **Move method**

Refactoring en Clase Persoona:

```
public double calcularMontoTotalLlamadas() {
        double costo = 0;
        for (Llamada llamada : this.getLlamadas()) {
                double costoAuxiliar= llamada.calcularCostoLlamada();
                costo += costoAuxiliar;
                costo -= this.calcularDescuentoPersoona(costoAuxiliar);
                }
        return costo;
}
```

Refactoring en Clase Persoonal:

```
public double calcularMontoTotalLlamadas(Persoona persona) {
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal != null)
                return personaEnPersoonal.calcularMontoTotalLlamadas();
        return 0;
}
```

Mal olor: **Comprobación de Null**

Mal olor en Clase Persoonal:

```
private Persoona buscarPersoonaEnPersoonal(Persoona persona) {
        for (Persoona usuario : personas) {
                if (usuario.getTelefono() == persona.getTelefono()) {
                        return usuario;
                }
        }
        return null;
}


public double calcularMontoTotalLlamadas(Persoona persona) {
        Persoona personaEnPersoonal = buscarPersoonaEnPersoonal(persona);
        if (personaEnPersoonal != null)
                return personaEnPersoonal.calcularMontoTotalLlamadas();
        return 0;
}
```

Refactoring: **Introduce Null Object**

Refactoring en nueva Clase PersoonaNull:

```
public class PersoonaNull extends Persoona {
        public PersoonaNull() {
                super();
        }

        public double calcularDescuentoPersoona(double costo) {
                return 0;
        }

        @Override
        public double calcularMontoTotalLlamadas() {
                return 0;
        }
}
```

Refactoring en Clase Persoonal:

```
private Persoona buscarPersoonaEnPersoonal(Persoona persona) {
        for (Persoona usuario : personas) {
                if (usuario.getTelefono() == persona.getTelefono()) {
                        return usuario;
                }
        }
        return new PersoonaNull();
}

public double calcularMontoTotalLlamadas(Persoona persona) {
                return buscarPersoonaEnPersoonal(persona).calcularMontoTotalLlamadas();
}
```

Mal olor: **Reinventa la rueda**

Se recorren listas con for cuando es mucho más practico utilizar pipelines que provee java.

Mal olor en Clase Persoonal:

```
private Persoona buscarPersoonaEnPersoonal(Persoona persona) {
        for (Persoona usuario : personas) {
                if (usuario.getTelefono() == persona.getTelefono()) {
                        return usuario;
```

```
                }
        }
        return new PersoonaNull();
}
```

Mal olor en Clase Persoona:

```
public double calcularMontoTotalLlamadas() {
        double costo = 0;
        for (Llamada llamada : this.getLlamadas()) {
                double costoAuxiliar= llamada.calcularCostoLlamada();
                costo += costoAuxiliar;
                costo -= this.calcularDescuentoPersoona(costoAuxiliar);
        }
        return costo;
}
```

Refactoring: **Replace Loop with Pipeline**

Refactoring en Clase Persoonal:

```
private Persoona buscarPersoonaEnPersoonal(Persoona persona) {
        return personas.stream().filter(usuario -> usuario.getTelefono() ==
persona.getTelefono()).findFirst().orElse(new PersoonaNull());
}
```

Refactoring en Clase Persoona:

```
public double calcularMontoTotalLlamadas() {
        return this.llamadas.stream().mapToDouble(llamada -> {
                double costo = llamada.calcularCostoLlamada();
                return costo - this.calcularDescuentoPersoona(costo);
            }).reduce(0, Double::sum);
}
```