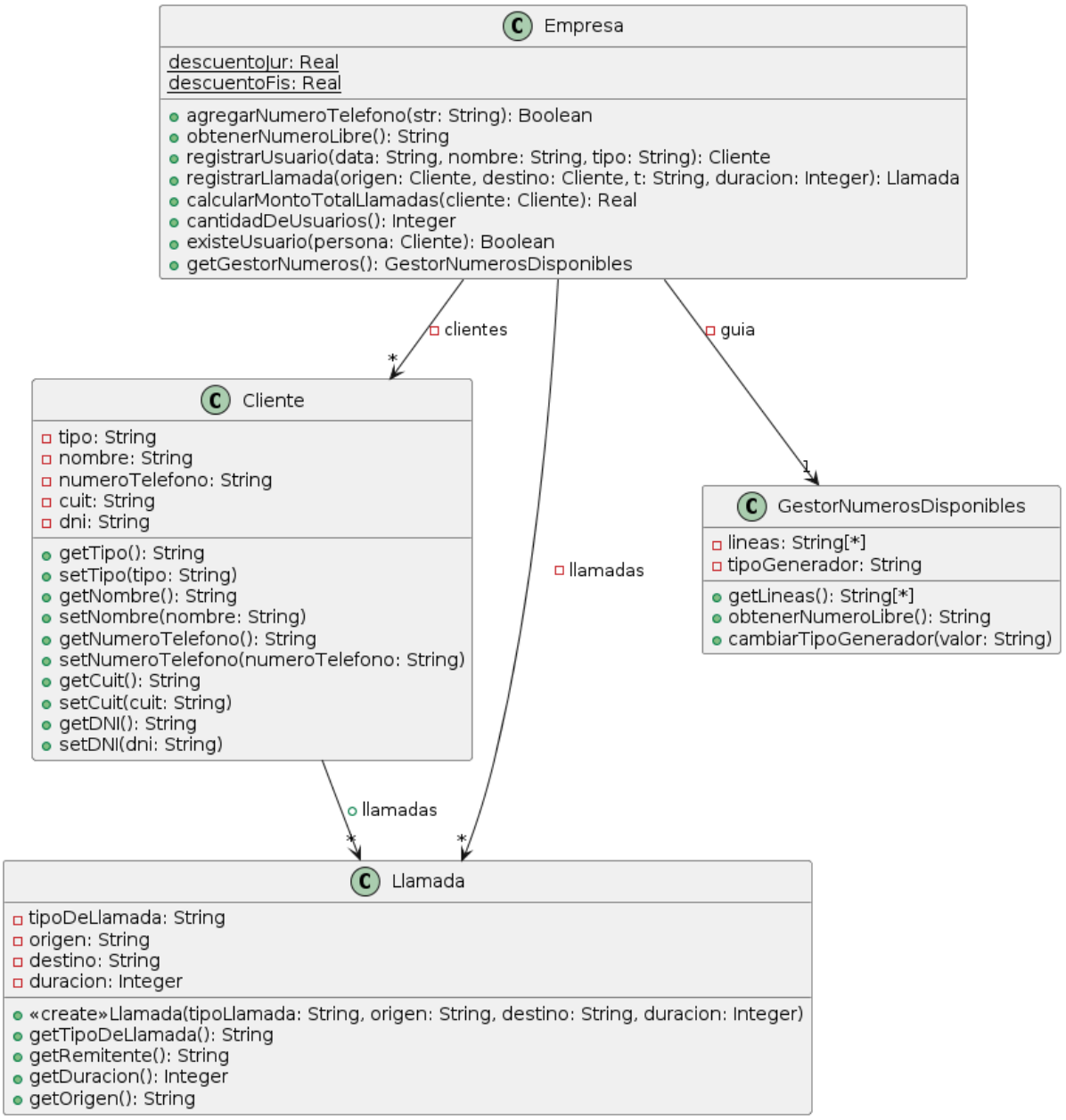
Refactoring

Ejercicio 3 - Facturación de llamadas

Integrantes

Gregorio Ponce - 21361/2
Pedro Spadari - 21586/8

UML del diseño inicial



Secuencia de refactorings

I. Mal olor: Falta un constructor para inicializar los objetos

- No hay un constructor explícito para instanciar objetos de la clase Cliente.
- Esto implica que se tenga que hacer uso de los setters de la clase Cliente para inicializar sus valores.

II. Extracto de código que representa el mal olor:

Clase Empresa

```
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;
}
```

III. Refactoring: Constructor Initialization

- Se crea un constructor en la clase Cliente: Cliente(tipo: String, nombre: String, numeroTelefono: String, cuit: String, dni: String)
- Se reemplaza la inicialización de los objetos de la clase Cliente, pasando de usarse setters a usarse el nuevo constructor.
- El parametro cuit se pasa como un string vacío en el Cliente de tipo “física”, y el dni se pasa como un string vacío en el Cliente de tipo “jurídica”

IV. Código modificado:

Clase Empresa

```
public Cliente registrarUsuario(String data, String nombre, String tipo){
	String tel = this.obtenerNumeroLibre();
	Cliente var;
	if (tipo.equals("fisica")) {
		var = new Cliente(tipo, nombre, tel, "", data);
```

```
	}
	else {
		var = new Cliente(tipo, nombre, tel, data, "");
	}
	clientes.add(var);
	return var;
	}
```

Clase Cliente

```
public Cliente(String tipo, String nombre, String numeroTelefono, String
cuit, String dni) {
	this.tipo = tipo;
	this.nombre = nombre;
	this.numeroTelefono = numeroTelefono;
	this.cuit = cuit;
	this.dni = dni;
}
```

I. Mal olor: Feature Envy y Data Class

- El método Empresa>>calcularMontoTotalLlamadas(cliente: Cliente) accede a las llamadas de un cliente específico para recorrerlas y calcular el monto total de las mismas.
- Este método no debería estar en la clase Empresa, sino en la clase Cliente, ya que es quien tiene la lista de sus llamadas (Feature Envy).
- Además, la clase Cliente es una clase de datos, pero al recibir el comportamiento que le corresponde dejaría de serlo (Data Class).

II. Extracto de código que representa el mal olor:

Clase 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 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;
}
```

III. Refactoring: Move method

- Se mueve el método Empresa>>calcularMontoTotalLlamadas(cliente: Cliente) a Cliente>>calcularMontoTotalLlamadas().
- Se elimina el método original.

IV. Código modificado:

Clase Cliente

```
public double calcularMontoTotalLlamadas() {
	double c = 0;
```

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

Este refactoring nos obliga a modificar los tests:

EmpresaTest>>testcalcularMontoTotalLlamadas (antes de la modificación)

```
assertEquals(11454.64,this.sistema.calcularMontoTotalLlamadas(emisorPersonaFisca), 0.01);
assertEquals(2445.40,this.sistema.calcularMontoTotalLlamadas(emisorPersonaJuridica), 0.01);
assertEquals(0, this.sistema.calcularMontoTotalLlamadas(remitentePersonaFisica));
assertEquals(0,this.sistema.calcularMontoTotalLlamadas(remitentePersonaJuridica));
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
assertEquals(11454.64, emisorPersonaFisca.calcularMontoTotalLlamadas(), 0.01);
assertEquals(2445.40, emisorPersonaJuridica.calcularMontoTotalLlamadas(), 0.01);
assertEquals(0, remitentePersonaFisica.calcularMontoTotalLlamadas());
assertEquals(0, remitentePersonaJuridica.calcularMontoTotalLlamadas());
```

I. Mal olor: Long Method

- El método Cliente>>calcularMontoTotalLlamadas() es un método largo.

II. Extracto de código que representa el mal olor:

Clase Cliente

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

III. Refactoring: Extract Method

- Se extrae la funcionalidad para obtener el monto de una llamada particular, al método Cliente>>getMontoLlamada(l: Llamada)

IV. Código modificado:

Clase Cliente

```
private double getMontoLlamada(Llamada l) {
	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;
        }
}

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

I. Mal olor: Feature Envy y Data Class

- El método Cliente>>getMontoLlamada(l: Llamada) recibe una llamada y dependiendo su tipo, retorna el monto de la misma.
- Este método no debería estar en la clase Cliente, sino en la clase Llamada, ya que es quien tiene el tipo y duración de la misma (Feature Envy).
- Además, la clase Llamada es una clase de datos, pero al recibir el comportamiento que le corresponde dejaría de serlo (Data Class).

II. Extracto de código que representa el mal olor:

Clase Cliente

```
private double getMontoLlamada(Llamada l) {
	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;
	}
	return auxc;
}
```

III. Refactoring: Move Method y Rename Method

- Se mueve el método Cliente>>getMontoLlamada(l: Llamada) a Llamada>>calcularMonto().
- Se elimina el método original.

IV. Código modificado:

Clase Llamada

```
public double getMonto() {
	double auxc = 0;
	if (this.getTipoDeLlamada() == "nacional") {
		// el precio es de 3 pesos por segundo más IVA sin adicional por establecer la llamada
		auxc += this.getDuracion() * 3 + (this.getDuracion() * 3 * 0.21);
	} else if (this.getTipoDeLlamada() == "internacional") {
		// el precio es de 150 pesos por segundo más IVA más 50 pesos por establecer la llamada
		auxc += this.getDuracion() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
	}
	return auxc;
```

```
}
```

Clase Cliente

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

I. Mal olor: Dead Code, Feature Envy y Duplicated Code

- La variable Empresa.llamadas no se usa (solo se agregan las llamadas a la misma) (Dead Code).
- Además, para acceder a las llamadas del sistema se podría solicitar a cada Cliente sus llamadas, ya que ya existe una variable llamadas en la clase Cliente (Duplicated Code), que es la clase que debería tener las llamadas (Feature Envy).

II. Extracto de código que representa el mal olor:

Clase Empresa

```
private List<Llamada> llamadas = new ArrayList<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); // única referencia a la variable llamadas
	origen.llamadas.add(llamada);
	return llamada;
}
```

III. Refactoring: Remove Dead Code

- Se elimina la variable llamadas de la clase Empresa.
- Se elimina la única referencia a la variable llamadas

IV. Código modificado:

Clase Empresa

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

I. Mal olor: Switch Statements

- Se realizan diferentes calculos del monto de una llamada dependiendo de su tipo.
- Esto sucede en el método Llamada>>getMonto().

II. Extracto de código que representa el mal olor:

Clase Llamada

```
public double getMonto() {
	double auxc = 0;
	if (this.getTipoDeLlamada() == "nacional") {
		// el precio es de 3 pesos por segundo más IVA sin adicional por establecer la llamada
		auxc += this.getDuracion() * 3 + (this.getDuracion() * 3 * 0.21);
	} else if (this.getTipoDeLlamada() == "internacional") {
		// el precio es de 150 pesos por segundo más IVA más 50 pesos por establecer la llamada
		auxc += this.getDuracion() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
	}
	return auxc;
}
```

Clase Empresa

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

III. Refactoring: Replace Conditional with Polymorphism y Replace Type Code with Subclasses

- La clase Llamada pasa a ser abstracta.
- Se crean las subclases de Llamada: LlamadaNacional y LlamadaInternacional.
- Ambas subclases implementan el método getMonto() realizando el cálculo correspondiente en él.
- A su vez, el método Llamada>>getMonto() pasa a ser un método abstracto.
- La variable de instancia Llamada.tipoLlamada ya no es necesaria y se elimina.
- Como consecuencia de la modificación de la jerarquía de clases, se debe eliminar el método Empresa>>registrarLlamada(origen: Cliente, destino: Cliente, t: String, duracion: Integer), siendo reemplazado por:

- Empresa>>registrarLlamadaNacional(origen: Cliente, destino: Cliente, duracion: Integer)
- Empresa>>registrarLlamadaInternacional(origen: Cliente, destino: Cliente, duracion: Integer)

IV. Código modificado:

Clase Llamada

```
public abstract class Llamada {
	private String origen;
	private String destino;
	private int duracion;
	public Llamada(String origen, String destino, int duracion) {
		this.origen= origen;
		this.destino= destino;
		this.duracion = duracion;
	}

	// ............

	public abstract double getMonto();
}
```

Clase LlamadaNacional

```
public class LlamadaNacional extends Llamada {
	public LlamadaNacional(String origen, String destino, int duracion)
	{
		super(origen, destino, duracion);
	}

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

Clase LlamadaInternacional

```
public class LlamadaInternacional extends Llamada {
	public LlamadaInternacional(String origen, String destino, int
	duracion) {
		super(origen, destino, duracion);
	}

	@Override
	public double getMonto() {
```

```
            return this.getDuracion() * 150 + (this.getDuracion() * 150
      * 0.21) + 50;
      }
}
```

Clase Empresa

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

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

Este refactoring nos obliga a modificar los tests:

EmpresaTest>>testcalcularMontoTotalLlamadas (antes de la modificación)

```
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);
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
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);
```

I. Mal olor: Long Method

- El método Cliente>>calcularMontoTotalLlamadas() es un método largo.

II. Extracto de código que representa el mal olor:

Clase Cliente

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

III. Refactoring: Extract Method

- Se extrae la funcionalidad para obtener el monto del descuento, en base al monto de una llamada, del método Cliente>>calcularMontoTotalLlamadas() al método Cliente>>getMontoDescuento(monto: Real).

IV. Código modificado:

Clase Cliente

```
private double getMontoDescuento(double monto) {
	double aux = 0;
	if (this.getTipo() == "fisica") {
		aux = monto * Empresa.descuentoFis;
	} else if (this.getTipo() == "juridica") {
		aux = monto * Empresa.descuentoJur;
	}
	return aux;
}

public double calcularMontoTotalLlamadas() {
	double c = 0;
	for (Llamada l : this.llamadas) {
		double auxc = l.getMonto();
		auxc -= this.getMontoDescuento(auxc);
		c += auxc;
	}
	return c;
}
```

I. Mal olor: Switch Statements

- Se realizan diferentes calculos del descuento a aplicar al monto de una llamada dependiendo de su tipo.
- Esto sucede en el método Cliente>>getMontoDescuento(monto: Real).

II. Extracto de código que representa el mal olor:

Clase Cliente

```
private double getMontoDescuento(double monto) {
	double aux = 0;
	if (this.getTipo() == "fisica") {
		aux = monto * Empresa.descuentoFis;
	} else if (this.getTipo() == "juridica") {
		aux = monto * Empresa.descuentoJur;
	}
	return aux;
}
```

Clase Empresa

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

III. Refactoring: Replace Conditional with Polymorphism y Replace Type Code with Subclasses

- La clase Cliente pasa a ser abstracta.
- A su vez, el método Cliente>>getMontoDescuento() también pasa a ser abstracto.
- Se crean las subclases de Cliente: PersonaFisica y PersonaJuridica.
- Ambas subclases implementan el método getMontoDescuento() realizando el cálculo correspondiente en él.
- La variable de instancia cuit solo debe pertenecer a la clase PersonaJuridica por lo que se realiza un Push Down Field para pasarla a dicha clase. También se realiza un Push Down Method para pasar el método getCuit() y setCuit(cuit: String)

- La variable de instancia dni solo debe pertenecer a la clase PersonaFisica por lo que se realiza un Push Down Field para pasarla a dicha clase. También se realiza un Push Down Method para pasar el método getDni() y setDni(dni: String)
- La variable de instancia Cliente.tipo ya no es necesaria y se elimina.
- Como consecuencia de la modificación de la jerarquía de clases, se debe eliminar el método Empresa>>registrarUsuario(data: String, nombre: String, tipo: String), siendo reemplazado por:
    - Empresa>>registrarPersonaFisica(data: String, nombre: String)
    - Empresa>>registrarPersonaJuridica(data: String, nombre: String)

IV. Código modificado:

Clase Cliente

```
public abstract class Cliente {
	public List<Llamada> llamadas = new ArrayList<Llamada>();
	private String nombre;
	private String numeroTelefono;
	public Cliente(String nombre, String numeroTelefono) {
		this.nombre = nombre;
		this.numeroTelefono = numeroTelefono;
		this.llamadas = new ArrayList<>();
	}

	// ........

	public abstract double getMontoDescuento(double monto);
}
```

Clase PersonaFisica

```
public class PersonaFisica extends Cliente {
	private String dni;

	public PersonaFisica(String nombre, String numeroTelefono, String
	dni) {
		super(nombre, numeroTelefono);
		this.dni = dni;
	}

	@Override
	public double getMontoDescuento(double monto) {
		return monto * Empresa.descuentoFis;
	}
}
```

Clase PersonaJuridica

```
public class PersonaJuridica extends Cliente {
	private String cuit;

	public PersonaJuridica(String nombre, String numeroTelefono, String
cuit) {
		super(nombre, numeroTelefono);
		this.cuit = cuit;
	}

	@Override
	public double getMontoDescuento(double monto) {
		return monto * Empresa.descuentoJur;
	}
}
```

Clase Empresa

```
public Cliente registrarPersonaFisica(String dni, String nombre) {
	String tel = this.obtenerNumeroLibre();
	Cliente var = new PersonaFisica(nombre, tel, dni);
	clientes.add(var);
	return var;
}

public Cliente registrarPersonaJuridica(String cuit, String nombre) {
	String tel = this.obtenerNumeroLibre();
	Cliente var = new PersonaJuridica(nombre, tel, cuit);
	clientes.add(var);
	return var;
}
```

Este refactoring nos obliga a modificar los tests:

EmpresaTest>>testcalcularMontoTotalLlamadas (antes de la modificación)

```
Cliente emisorPersonaFisca = sistema.registrarUsuario("11555666", "Brendan Eich" , "fisica");
Cliente remitentePersonaFisica = sistema.registrarUsuario("00000001", "Doug Lea" , "fisica");
Cliente emisorPersonaJuridica = sistema.registrarUsuario("17555222", "Nvidia Corp" , "juridica");
Cliente remitentePersonaJuridica = sistema.registrarUsuario("25765432", "Sun Microsystems",
"juridica");
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
Cliente emisorPersonaFisca = sistema.registrarPersonaFisica("11555666", "Brendan Eich");
Cliente remitentePersonaFisica = sistema.registrarPersonaFisica("00000001", "Doug Lea");
Cliente emisorPersonaJuridica = sistema.registrarPersonaJuridica("17555222", "Nvidia Corp");
Cliente remitentePersonaJuridica = sistema.registrarPersonaJuridica("25765432", "Sun
Microsystems");
```

EmpresaTest>>testAgregarUsuario (antes de la modificación)

```
Cliente nuevaPersona = this.sistema.registrarUsuario("2444555","Alan Turing", "fisica");
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
Cliente nuevaPersona = this.sistema.registrarPersonaFisica("2444555", "Alan Turing");
```

I. Mal olor: Feature Envy

- Las variables estáticas Empresa.descuentoJur y Empresa.descuentoFis solo son usadas dentro de la clase PersonaJuridica y PersonaFisica respectivamente.

II. Extracto de código que representa el mal olor:

Clase Empresa

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

Clase PersonaFisica

```
public double getMontoDescuento(double monto) {
	return monto * Empresa.descuentoFis;
}
```

Clase PersonaJuridica

```
public double getMontoDescuento(double monto) {
	return monto * Empresa.descuentoJur;
}
```

III. Refactoring: Move Field

- Se mueve el campo descuentoJur de la clase Empresa a la clase PersonaJuridica.
- Se mueve el campo descuentoFis de la clase Empresa a la clase PersonaFisica.

IV. Código modificado:

Clase PersonaFisica

```
static double descuentoFis = 0;

public PersonaFisica(String nombre, String numeroTelefono, String dni) {
	super(nombre, numeroTelefono);
	this.dni = dni;
}

@Override
public double getMontoDescuento(double monto) {
	return monto - monto * PersonaFisica.descuentoFis;
}
```

Clase PersonaJuridica

```
static double descuentoJur = 0.15;

public PersonaJuridica(String nombre, String numeroTelefono, String cuit)
{
	super(nombre, numeroTelefono);
	this.cuit = cuit;
}

@Override
public double getMontoDescuento(double monto) {
		return monto - monto * PersonaJuridica.descuentoJur;
}
```

I. Mal olor: Campos sin visibilidad declarada

- Los campos PersonaFisica.descuentoFis y PersonaJuridica.descuentoJur no tienen ninguna visibilidad declarada.

II. Extracto de código que representa el mal olor:

Clase PersonaFisica
`static double descuentoFis = 0;`

Clase PersonaJuridica
`static double descuentoJur = 0.15;`

III. Refactoring: Encapsulate Field

- Se les añade la visibilidad private a los campos PersonaFisica.descuentoFis y PersonaJuridica.descuentoJur.

IV. Código modificado:

Clase PersonaFisica
`private static double descuentoFis = 0;`

Clase PersonaJuridica
`private static double descuentoJur = 0.15;`

I. Mal olor: Nombres poco descriptivos

- Los campos PersonaFisica.descuentoFis y PersonaJuridica.descuentoJur tienen nombres poco descriptivos

II. Extracto de código que representa el mal olor:

Clase PersonaFisica

```
private static double descuentoFis = 0;

// .........

@Override
public double getMontoDescuento(double monto) {
	return monto - monto * PersonaFisica.descuentoFis;
}
```

Clase PersonaJuridica

```
private static double descuentoJur = 0.15;

// .........

@Override
public double getMontoDescuento(double monto) {
	return monto - monto * PersonaJuridica.descuentoJur;
}
```

III. Refactoring: Rename Field

- Se cambia el nombre del campo PersonaFisica.descuentoFis a PersonaFisica.descuentoParaPersonaFisica.
- Se cambia el nombre del campo PersonaJuridica.descuentoJur a PersonaJuridica.descuentoParaPersonaJuridica.

IV. Código modificado:

Clase PersonaFisica

```
private static double descuentoParaPersonaFisica = 0;

// .........

@Override
public double getMontoDescuento(double monto) {
	return monto - monto * PersonaFisica.descuentoParaPersonaFisica;
}
```

Clase PersonaJuridica

```
private static double descuentoParaPersonaJuridica = 0.15;

// .........

@Override
public double getMontoDescuento(double monto) {
       return monto - monto *
PersonaJuridica.descuentoParaPersonaJuridica;
}
```

I. Mal olor: Long Method

- El método Cliente>>calcularMontoTotalLlamadas() es un método largo.

II. Extracto de código que representa el mal olor:

Clase Cliente

```
public double calcularMontoTotalLlamadas() {
	double c = 0;
	for (Llamada l : this.llamadas) {
		double auxc = l.getMonto();
		auxc -= this.getMontoDescuento(auxc);
		c += auxc;
	}
	return c;
}
```

III. Refactoring: Replace Temp with Query

- Se reemplaza la variable temporal usada para el cálculo del monto de la llamada con el descuento aplicado, con el método privado Cliente>>getMontoLlamadaConDescuento(llamada: LLamada)

IV. Código modificado:

Clase Cliente

```
private double getMontoLlamadaConDescuento(Llamada llamada) {
	double auxc = llamada.getMonto();
	return auxc - this.getMontoDescuento(auxc);
}

public double calcularMontoTotalLlamadas() {
	double c = 0;
	for (Llamada l : this.llamadas) {
		c += this.getMontoLlamadaConDescuento(l);
	}
	return c;
}
```

I. Mal olor: Ya está implementada la funcionalidad para sumar los elementos de una colección

- En el método Cliente>>calcularMontoTotalLlamadas() se realiza un loop, iterando sobre los elementos de la variable Cliente.llamadas, para sumar el valor que retorna el método Cliente>>getMontoLlamadaConDescuento(l: Llamada), para cada llamada de la colección.
- Esta funcionalidad está implementada en el protocolo de stream de Java.

II. Extracto de código que representa el mal olor:

Clase Cliente

```
public double calcularMontoTotalLlamadas() {
	double c = 0;
	for (Llamada l : this.llamadas) {
		c += this.getMontoLlamadaConDescuento(l);
	}
	return c;
}
```

III. Refactoring: Replace Loop with Pipeline

- Se reemplaza el loop con un pipeline provisto por Java
- Se retorna el valor de retorno del stream, en vez de hacer uso de la variable temporal c.

IV. Código modificado:

Clase Cliente

```
public double calcularMontoTotalLlamadas() {
	return this.llamadas.stream()
		.mapToDouble(l -> this.getMontoLlamadaConDescuento(l))
		.sum();
}
```

I. Mal olor: Nombres poco descriptivos

- El nombre de la variable auxc es poco descriptivo.
- El nombre de la variable l es poco descriptivo.

II. Extracto de código que representa el mal olor:

Clase Cliente

```
private double getMontoLlamadaConDescuento(Llamada llamada) {
	double auxc = llamada.getMonto();
	return auxc - this.getMontoDescuento(auxc);
}

public double calcularMontoTotalLlamadas() {
	return this.llamadas.stream()
		.mapToDouble(l -> this.getMontoLlamadaConDescuento(l))
		.sum();
}
```

III. Refactoring: Rename Variable

- Se renombra la variable auxc a montoLlamada.
- Se renombra la variable l a llamada.

IV. Código modificado:

Clase Cliente

```
private double getMontoLlamadaConDescuento(Llamada llamada) {
	double montoLlamada = llamada.getMonto();
	return montoLlamada - this.getMontoDescuento(montoLlamada);
}

public double calcularMontoTotalLlamadas() {
   return this.llamadas.stream()
	.mapToDouble(llamada -> this.getMontoLlamadaConDescuento(llamada))
	.sum();
}
```

I. Mal olor: Feature Envy

- El método Empresa>>registrarLlamadaNacional(origen: Cliente, destino: Cliente, duracion: Cliente) crea una llamada nacional y la agrega a la lista de llamadas del cliente de origen.
- El método Empresa>>registrarLlamadaInternacional(origen: Cliente, destino: Cliente, duracion: Cliente) hace lo mismo pero con las llamadas internacionales.
- Ambos métodos deberían estár en la clase Cliente, y que el Cliente de origen se encargue de crear las llamadas y agregarlas a su lista de llamadas.

II. Extracto de código que representa el mal olor:

Clase Empresa

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

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

III. Refactoring: Move Method

- Se mueve el método Empresa>>registrarLlamadaNacional(origen: Cliente, destino: Cliente, duracion: Cliente) y el método Empresa>>registrarLlamadaInternacional(origen: Cliente, destino: Cliente, duracion: Cliente) a la clase Cliente.
- Los nuevos métodos son:
  - Cliente>>registrarLlamadaNacional(clienteDestino: Cliente, duracion: Integer)
  - Cliente>>registrarLlamadaInternacional(clienteDestino: Cliente, duracion: Integer)
- Se eliminan los métodos originales

IV. Código modificado:

Clase Cliente

```
public Llamada registrarLlamadaNacional(Cliente clienteDestino, int duracion) {
```

```
		Llamada llamada = new LlamadaNacional(this.getNumeroTelefono(),
clienteDestino.getNumeroTelefono(), duracion);
		this.llamadas.add(llamada);
		return llamada;
}

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

Este refactoring nos obliga a modificar los tests:

EmpresaTest>>testcalcularMontoTotalLlamadas (antes de la modificación)

```
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);
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
emisorPersonaJuridica.registrarLlamadaNacional(remitentePersonaFisica, 10);
emisorPersonaJuridica.registrarLlamadaInternacional(remitentePersonaFisica, 8);
emisorPersonaJuridica.registrarLlamadaNacional(remitentePersonaJuridica, 5);
emisorPersonaJuridica.registrarLlamadaInternacional(remitentePersonaJuridica, 7);
emisorPersonaFisca.registrarLlamadaNacional(remitentePersonaFisica, 15);
emisorPersonaFisca.registrarLlamadaInternacional(remitentePersonaFisica, 45);
emisorPersonaFisca.registrarLlamadaNacional(remitentePersonaJuridica, 13);
emisorPersonaFisca.registrarLlamadaInternacional(remitentePersonaJuridica, 17);
```

I. Mal olor: Feature Envy

- El método Empresa>>agregarNumeroTelefono(str: String) agrega un numero de telefono a la lista de telefonos contenida en GestorNumerosDisponibles
- Dado que se modifica y accede a la variable lineas de GestorNumerosDisponibles, este método debería estar en dicha clase

II. Extracto de código que representa el mal olor:

Clase Empresa

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

III. Refactoring: Move Method

- Se mueve el método Empresa>>agregarNumeroTelefono(str: String) a la clase GestorNumerosDisponibles>>agregarNumeroTelefono(numeroDeTelefono: String).
- Se convierte al método original en una delegación.

IV. Código modificado:

Clase Empresa

```
public boolean agregarNumeroTelefono(String str) {
	return this.guia.agregarNumeroTelefono(str);
}
```

Clase GestorNumerosDisponibles

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

```
}
```

I. Mal olor: La funcionalidad de no tener repeticiones dentro de una colección es la característica de un set

- En la el método GestorNumerosDisponibles>>agregarNumeroTelefono() se lleva a cabo un código para agregar un número de teléfono al set de lineas, pero si el número ya se encuentra en la misma no lo agrega.
- Dado que está es la característica principal de un Set, y la variable líneas de telefono es un Set, se debería hacer uso de dicha funcionalidad.

II. Extracto de código que representa el mal olor:

Clase GestorNumerosDisponibles

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

III. Refactoring: Substitute Algorithm

- Se reemplaza el algoritmo para agregar un numero de telefono a la variable lineas.
- Simplemente se agrega el numero de telefono a la variable lineas. Pueden suceder dos casos:
  - Si el mismo ya se encuentra en el Set, el metodo devuelve falso y no se lo agrega.
  - Si el mismo no se encuentra en el Set, se lo agrega y se devuelve verdadero.

IV. Código modificado:

Clase GestorNumerosDisponibles

```
public boolean agregarNumeroTelefono(String numeroDeTelefono) {
	return this.getLineas().add(numeroDeTelefono);
}
```

I. Mal olor: Nombres poco descriptivos

- El nombre del campo lineas es poco descriptivo.
- El nombre de la variable linea es poco descriptivo.
- El nombre del parametro valor es poco descriptivo.

II. Extracto de código que representa el mal olor:

Clase GestorNumerosDisponibles

```
private SortedSet<String> lineas = new TreeSet<String>();

private String tipoGenerador = "ultimo";

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

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;
}

public void cambiarTipoGenerador(GeneradorDeLineasStrategy valor) {
	this.tipoGenerador = valor;
}
```

III. Refactoring: Rename Field, Rename Variable y Rename Parameter

- Se renombra el campo lineas a lineasTelefonicasDisponibles.
- Se renombra la variable linea a lineaTelefonica.
- Se renombra el parámetro valor a nuevoTipoGenerador.

IV. Código modificado:

Clase GestorNumerosDisponibles

```
private SortedSet<String> lineasTelefonicasDisponibles = new
TreeSet<String>();

private String tipoGenerador = "ultimo";

public SortedSet<String> getLineasTelefonicasDisponibles() {
	return lineasTelefonicasDisponibles;
}

public String obtenerNumeroLibre() {
	String lineaTelefonica;
	switch (tipoGenerador) {
	case "ultimo":
		lineaTelefonica = lineasTelefonicasDisponibles.last();
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	case "primero":
		lineaTelefonica = lineasTelefonicasDisponibles.first();
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	case "random":
		lineaTelefonica = new
ArrayList<String>(lineasTelefonicasDisponibles)
				.get(new
Random().nextInt(lineasTelefonicasDisponibles.size()));
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	}
	return null;
}

public void cambiarTipoGenerador(GeneradorDeLineasStrategy
nuevoTipoGenerador) {
	this.tipoGenerador = nuevoTipoGenerador;
}
```

I. Mal olor: Duplicated Code

- En el método GestorNumerosDisponibles>>obtenerNumeroLibre() se repite el código para eliminar y retornar la lineaTelefonica libre.

II. Extracto de código que representa el mal olor:

Clase GestorNumerosDisponibles

```
public String obtenerNumeroLibre() {
	String lineaTelefonica;
	switch (tipoGenerador) {
	case "ultimo":
		lineaTelefonica = lineasTelefonicasDisponibles.last();
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	case "primero":
		lineaTelefonica = lineasTelefonicasDisponibles.first();
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	case "random":
		lineaTelefonica = new
ArrayList<String>(lineasTelefonicasDisponibles)
				.get(new
Random().nextInt(lineasTelefonicasDisponibles.size()));
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
		return lineaTelefonica;
	}
	return null;
}
```

III. Refactoring: Consolidate Duplicate Conditional Fragments

- Agrupa la lógica realizada en todos los casos del switch, al final del mismo.

IV. Código modificado:

Clase GestorNumerosDisponibles

```
public String obtenerNumeroLibre() {
	String lineaTelefonica = null;

	switch (tipoGenerador) {
	case "ultimo":
		lineaTelefonica = lineasTelefonicasDisponibles.last();
	case "primero":
		lineaTelefonica = lineasTelefonicasDisponibles.first();
	case "random":
		lineaTelefonica = new
ArrayList<String>(lineasTelefonicasDisponibles)
```

```
                                        .get(new
Random().nextInt(lineasTelefonicasDisponibles.size()));
        }

        if(lineaTelefonica != null) {
                lineasTelefonicasDisponibles.remove(lineaTelefonica);
        }
        return lineaTelefonica;
}
```

I. Mal olor: Switch Statements

- Existe lógica condicional en el método GestorNumerosDisponibles>>obtenerNumeroLibre() que controla qué estrategia para retornar un número de teléfono libre se va a tomar.

II. Extracto de código que representa el mal olor:

Clase GestorNumerosDisponibles

```
public String obtenerNumeroLibre() {
	String lineaTelefonica = null;

	switch (tipoGenerador) {
	case "ultimo":
		lineaTelefonica = lineasTelefonicasDisponibles.last();
	case "primero":
		lineaTelefonica = lineasTelefonicasDisponibles.first();
	case "random":
		lineaTelefonica = new
ArrayList<String>(lineasTelefonicasDisponibles)
					.get(new
Random().nextInt(lineasTelefonicasDisponibles.size()));
	}

	if(lineaTelefonica != null) {
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
	}
	return lineaTelefonica;
}
```

III. Refactoring: Replace Conditional Logic with Strategy

- Se crea la clase abstracta GeneradorDeLineasStrategy
- Se aplica Move Method de la clase GestorNumerosDisponibles a la clase abstracta creada, moviendo el switch statement a la nueva clase abstracta.
- El método obtenerNumeroLibre de la clase GeneradorDeLineasStrategy recibe como parametro las lineasTelefonicasDisponibles
- Se referencia a la nueva clase abstracta, modificando el tipo de la variable de instancia tipoGenerador al tipo GeneradorDeLineasStrategy.
- Se modifica el tipo del parametro valor del método GestorNumerosDisponibles>>cambiarTipoGenerador(valor: String) al tipo GeneradorDeLineasStrategy.
- El método GestorNumerosDisponibles>>obtenerNumeroLibre() delega al método de la variable de instancia: tipoGenerador.obtenerNumeroLibre(lineasTelefonicasDisponibles: String[*]), y luego elimina la linea encontrada, en caso que exista.
- Se aplica Replace Conditional with Polymorphism en GeneradorDeLineasStrategy. De esta manera se obtienen las subclases concretas de GeneradorDeLineasStrategy:
  - StrategyPrimero

    - StrategyUltimo
    - StrategyRandom
- Se pasa de inicializar a la variable tipoGenerador con el valor “ultimo” a inicializarla con el valor new StrategyUltimo().
- El método GeneradorDeLineasStrategy>>obtenerNumeroLibre(lineasTelefonicasDisponibles: String[*]) queda como abstracto.
- Las subclases de GeneradorDeLineasStrategy implementan el método obtenerNumeroLibre(lineasTelefonicasDisponibles: String[*]), con la lógica pertinente a cada clase.

IV. Código modificado:

Clase GestorNumerosDisponibles

```
private GeneradorDeLineasStrategy tipoGenerador = new StrategyUltimo();

// ........

public String obtenerNumeroLibre() {
	String lineaTelefonica =
this.tipoGenerador.obtenerNumeroLibre(lineasTelefonicasDisponibles);
	if (lineaTelefonica != null) {
		lineasTelefonicasDisponibles.remove(lineaTelefonica);
	}
	return lineaTelefonica;
}

public void cambiarTipoGenerador(GeneradorDeLineasStrategy valor) {
	this.tipoGenerador = valor;
}
```

Clase GeneradorDeLineasStrategy

```
public abstract class GeneradorDeLineasStrategy {
	public abstract String obtenerNumeroLibre(SortedSet<String>
	lineasTelefonicasDisponibles);
}
```

Clase StrategyPrimero

```
public class StrategyPrimero implements GeneradorDeLineasStrategy {
	@Override
	public String obtenerNumeroLibre(SortedSet<String>
	lineasTelefonicasDisponibles) {
		return lineasTelefonicasDisponibles.first();
	}
}
```

Clase StrategyUltimo

```
public class StrategyUltimo implements GeneradorDeLineasStrategy {
	@Override
	public String obtenerNumeroLibre(SortedSet<String>
	lineasTelefonicasDisponibles) {
		return lineasTelefonicasDisponibles.last();
	}
}
```

Clase StrategyRandom

```
public class StrategyRandom implements GeneradorDeLineasStrategy {
	@Override
	public String obtenerNumeroLibre(SortedSet<String>
	lineasTelefonicasDisponibles) {
		return new ArrayList<String>(lineasTelefonicasDisponibles)
				.get(new
Random().nextInt(lineasTelefonicasDisponibles.size()));
	}
}
```

Este refactoring nos obliga a modificar los tests:

EmpresaTest>>obtenerNumeroLibre (antes de la modificación)

```
// por defecto es el ultimo
assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
this.sistema.getGestorNumeros().cambiarTipoGenerador("primero");
assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
this.sistema.getGestorNumeros().cambiarTipoGenerador("random");
assertNotNull(this.sistema.obtenerNumeroLibre());
```

EmpresaTest>>testcalcularMontoTotalLlamadas (después de la modificación)

```
// por defecto es el ultimo
assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
this.sistema.getGestorNumeros().cambiarTipoGenerador(new StrategyPrimero());
assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
this.sistema.getGestorNumeros().cambiarTipoGenerador(new StrategyRandom());
assertNotNull(this.sistema.obtenerNumeroLibre());
```

I. Mal olor: Nombres poco descriptivos

- El nombre del campo guia es poco descriptivo.
- El nombre de la variable tel es poco descriptivo.
- El nombre de la variable var es poco descriptivo.
- El nombre del parametro str es poco descriptivo.

II. Extracto de código que representa el mal olor:

Clase Empresa

```
private GestorNumerosDisponibles guia = new GestorNumerosDisponibles();

public boolean agregarNumeroTelefono(String str) {
	return this.guia.agregarNumeroTelefono(str);
}

public String obtenerNumeroLibre() {
	return guia.obtenerNumeroLibre();
}

public Cliente registrarPersonaFisica(String dni, String nombre) {
	String tel = this.obtenerNumeroLibre();
	Cliente var = new PersonaFisica(nombre, tel, dni);
	clientes.add(var);
	return var;
}

public Cliente registrarPersonaJuridica(String cuit, String nombre) {
	String tel = this.obtenerNumeroLibre();
	Cliente var = new PersonaJuridica(nombre, tel, cuit);
	clientes.add(var);
	return var;
}

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

III. Refactoring: Rename Variable

- Se renombra el campo guia a gestorNumerosDisponibles.
- Se renombra la variable tel a numeroTelefonoLibre.
- Se renombra la variable var a nuevoCliente.
- Se renombra el parámetro str a numeroDeTelefono.

IV. Código modificado:

Clase Empresa

```
private GestorNumerosDisponibles gestorNumerosDisponibles = new
GestorNumerosDisponibles();
```

```
public boolean agregarNumeroTelefono(String numeroDeTelefono) {
	return
this.gestorNumerosDisponibles.agregarNumeroTelefono(numeroDeTelefono);
}

public String obtenerNumeroLibre() {
	return gestorNumerosDisponibles.obtenerNumeroLibre();
}

public Cliente registrarPersonaFisica(String dni, String nombre) {
	String numeroTelefonoLibre = this.obtenerNumeroLibre();
	Cliente nuevoCliente = new PersonaFisica(nombre,
numeroTelefonoLibre, dni);
	clientes.add(nuevoCliente);
	return nuevoCliente;
}

public Cliente registrarPersonaJuridica(String cuit, String nombre) {
	String numeroTelefonoLibre = this.obtenerNumeroLibre();
	Cliente nuevoCliente = new PersonaJuridica(nombre,
numeroTelefonoLibre, cuit);
	clientes.add(nuevoCliente);
	return nuevoCliente;
}

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

I. Mal olor: Atributos Públicos

- En la clase Cliente, la variable de instancia llamadas está declarada pública por lo que no se respeta el encapsulamiento

II. Extracto de código que representa el mal olor:

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

III. Refactoring: Encapsulate Field

- Se cambia la visibilidad de la variable llamadas de public a private

IV. Código modificado:

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

I. Mal olor: Duplicated Code

- Dos o más métodos en subclases realizan pasos similares en el mismo orden, pero los pasos son distintos:
  - En los métodos LlamadaNacional>>getMonto() y LlamadaInternacional>>getMonto(), el cálculo del monto de la llamada sigue los mismos pasos, pero difiere en valores que representan algo particular (precio por segundo de una llamada o adicional por establecer una llamada)

II. Extracto de código que representa el mal olor:

Clase LlamadaNacional

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

Clase LlamadaInteracional

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

III. Refactoring: Form Template Method

- Encontrar el método que es similar en todas las subclases y extraer sus partes en: métodos idénticos (misma signatura y cuerpo en las subclases) o métodos únicos (distinta signatura y cuerpo):
  - En este caso, el método similar es getMonto()
  - En este caso no hay métodos idénticos
  - Sí hay métodos únicos (que ya quedan con el mismo nombre y signatura por lo que solo varía su cuerpo):

Clase LlamadaNacional

```
public int getPrecioPorSegundo() {
	return 3;
}

public int getCostoEstablecerLlamada() {
	return 0;
}
```

Clase LlamadaInternacional

```
public int getPrecioPorSegundo() {
	return 150;
}
```

```
public int getCostoEstablecerLlamada() {
	return 50;
}
```

- El método similar ahora es idéntico:

Clase LlamadaNacional y Llamada Internacional

```
public double getMonto() {
	return this.getDuracion() * this.getPrecioPorSegundo()
			+ (this.getDuracion() *
this.getPrecioPorSegundo() * 0.21) +
this.getCostoEstablecerLlamada();
}
```

- Se aplica Pull up Method sobre getMonto() el cual deja de ser abstracto en Llamada
- Se definen métodos abstractos para los métodos únicos getPrecioPorSegundo() y getCostoEstablecerLlamada()

IV. Código modificado:

Clase Llamada

```
public abstract int getPrecioPorSegundo();

public abstract int getCostoEstablecerLlamada();

public double getMonto() {
	return this.getDuracion() * this.getPrecioPorSegundo()
			+ (this.getDuracion() * this.getPrecioPorSegundo() *
0.21) + this.getCostoEstablecerLlamada();
}
```

Clase LlamadaNacional

```
public int getPrecioPorSegundo() {
	return 3;
}

public int getCostoEstablecerLlamada() {
	return 0;
}
```

Clase LlamadaInternacional

```
public int getPrecioPorSegundo() {
	return 150;
}

public int getCostoEstablecerLlamada() {
	return 50;
}
```

I. Mal olor: Magic Literal

- En el método Llamada>>getMonto() se usa el magic literal 0.21, que representa el porcentaje de I.V.A..
- Para hacerlo más legible se lo transforma en una constante con el mismo valor.

II. Extracto de código que representa el mal olor:

Clase Llamada

```
public double getMonto() {
	return this.getDuracion() * this.getPrecioPorSegundo()
			+ (this.getDuracion() * this.getPrecioPorSegundo() *
0.21) + this.getCostoEstablecerLlamada();
}
```

III. Refactoring: Replace Magic Literal

IV. Código modificado:

Clase Llamada

```
private static final double IVA = 0.21;

public double getMonto() {
	return this.getDuracion() * this.getPrecioPorSegundo()
			+ (this.getDuracion() * this.getPrecioPorSegundo() *
IVA) + this.getCostoEstablecerLlamada();
}
```

UML del diseño final

Se encuentra como archivo .png en el zip entregado.