Trabajo Práctico - Refactoring

Ejercicio 3

Rubbini, Agostina, 21867/5
Santa Cruz, Joaquín Emanuel, 22432/4

1. Bad smell: Nombre de variable poco descriptivo

Las siguientes variables tienen nombres poco descriptivos:

- clase Empresa
  - variable de instancia descuentoJur → descuentoClienteJuridico
  - variable de instancia descuentoFis → descuentoClienteFisico
  - método agregarNumeroTelefono
    - parámetro str → numero
  - método registrarUsuario
    - parámetro data → DNIoCUIT
    - parámetro tipo → tipoDeCliente
    - variable local var → cliente
    - variable local tel → telefono
  - método registrarLlamada
    - parámetro t → tipoDeLlamada
  - método calcularMontoTotalLlamadas
    - variable local c →
    - variable local l →
    - variable local auxc →
- clase GestorNumerosDisponibles
  - variable de instancia lineas → numerosDisponibles
  - método obtenerNumeroLibre
    - variable local linea → numeroLibre
  - método cambiarTipoGenerador
    - parámetro valor → tipoGenerador

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

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;
		}
	}

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

public class 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(String valor) {
                this.tipoGenerador = valor;
        }
}
```

Refactoring aplicado: Rename Field, Rename Variable

```
public class Empresa {
        ...
        static double descuentoClienteJuridico = 0.15;
        static double descuentoClienteFisico = 0;

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

        public Cliente registrarUsuario(String DNIoCUIT, String nombre, String tipoDeCliente) {
                Cliente cliente = new Cliente();
                if (tipoDeCliente.equals("fisica")) {
                        cliente.setNombre(nombre);
                        String telefono = this.obtenerNumeroLibre();
                        cliente.setTipo(tipoDeCliente);
                        cliente.setNumeroTelefono(telefono);
                        cliente.setDNI(DNIoCUIT);
                }
                else if (tipoDeCliente.equals("juridica")) {
                        String telefono = this.obtenerNumeroLibre();
                        cliente.setNombre(nombre);
                        cliente.setTipo(tipoDeCliente);
                        cliente.setNumeroTelefono(telefono);
                        cliente.setCuit(DNIoCUIT);
                }
                clientes.add(cliente);
                return cliente;
        }
        public Llamada registrarLlamada(Cliente origen, Cliente destino, String tipoDeLlamada,
int duracion) {
                Llamada llamada = new Llamada(tipoDeLlamada, origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
                llamadas.add(llamada);
                origen.llamadas.add(llamada);
                return llamada;
```

```
	}
	public double calcularMontoTotalLlamadas(Cliente cliente) {
		double montoTotal = 0;
		for (Llamada llamada : cliente.llamadas) {
			double montoAuxiliar = 0;
			if (llamada.getTipoDeLlamada() == "nacional") {
				// el precio es de 3 pesos por segundo más IVA sin adicional por
establecer la llamada
				montoAuxiliar += llamada.getDuracion() * 3 + (llamada.getDuracion() *
3 * 0.21);
			} else if (llamada.getTipoDeLlamada() == "internacional") {
				// el precio es de 150 pesos por segundo más IVA más 50 pesos por
establecer la llamada
				montoAuxiliar += llamada.getDuracion() * 150 + (llamada.getDuracion()
* 150 * 0.21) + 50;
			}
			if (cliente.getTipo() == "fisica") {
				montoAuxiliar -= montoAuxiliar*descuentoClienteFisico;
			} else if(cliente.getTipo() == "juridica") {
				montoAuxiliar -= montoAuxiliar*descuentoClienteJuridico;
			}
			montoTotal += montoAuxiliar ;
		}
		return montoTotal;
	}
	...
}


public class GestorNumerosDisponibles {
	private SortedSet<String> numerosDisponibles = new TreeSet<String>();
	private String tipoGenerador = "ultimo";

	public SortedSet<String> getLineas() {
		return numerosDisponibles;
	}
	public String obtenerNumeroLibre() {
		String numeroLibre;
		switch (tipoGenerador) {
			case "ultimo":
				numeroLibre = numerosDisponibles.last();
				numerosDisponibles.remove(numeroLibre);
				return numeroLibre;
			case "primero":
				numeroLibre = numerosDisponibles.first();
				numerosDisponibles.remove(numeroLibre);
				return numeroLibre;
			case "random":
				numeroLibre = new ArrayList<String>(numerosDisponibles)
					.get(new Random().nextInt(numerosDisponibles.size()));
				numerosDisponibles.remove(numeroLibre);
				return numeroLibre;
		}
		return null;
	}
	public void cambiarTipoGenerador(String tipoGenerador) {
		this.tipoGenerador = tipoGenerador;
	}
}
```

2. Bad smell: Nombre de método poco descriptivo

Los siguientes métodos tienen nombres poco descriptivos:

- clase Llamada
  - método getRemitente() → getDestino()
- clase GestorNumerosDisponibles
  - metodo getLineas() → getNumerosDisponibles()

```
public class Llamada {
	...
	public String getRemitente() {
		return destino;
	}
	...
}

public class GestorNumerosDisponibles {
	...
	public SortedSet<String> getLineas() {
		return numerosDisponibles;
	}
	...
}
```

Refactoring aplicado: Rename Method

```
public class Llamada {
	...
	public String getDestino() {
		return destino;
	}
	...
}

public class GestorNumerosDisponibles {
	...
	public SortedSet<String> getNumerosDisponibles() {
		return numerosDisponibles;
	}
	...
}
```

3. Bad smell: Declaración de atributo público

El problema es que hay variables de instancia declaración de visibilidad en ‘public’.

```
public class Cliente{
	...
	public List<Llamada> llamadas = new ArrayList<Llamada>();
	...
}
```

Refactoring aplicado: Encapsulate Field

```
public class Cliente{
	...
	private List<Llamada> llamadas = new ArrayList<Llamada>();
	...
	public List<Llamada> getLlamadas() {
		return llamadas;
	}
	public void setLlamadas(List<Llamada> llamadas) {
		this.llamadas = llamadas;
	}
	...
}
```

4. Bad smell: Declaración de atributo sin visibilidad

El problema es que hay variables de instancia sin declaración de visibilidad

```
public class Empresa {
	...
	static double descuentoClienteJuridico = 0.15;
	static double descuentoClienteFisico = 0;
	...
}
```

Refactoring aplicado: Encapsulate Field

```
public class Empresa {
	...
	private static double descuentoClienteJuridico = 0.15;
	private static double descuentoClienteFisico = 0;

public static double getDescuentoClienteJuridico() {
		return descuentoClienteJuridico;
	}
	public static double getDescuentoClienteFisico() {
		return descuentoClienteFisico;
	}
	...
}
```

5. Bad smell: Método largo

El problema es el método de la clase Empresa calcularMontoTotalLlamadas(), ya que es largo, esto genera que sea más difícil de entenderlo, cambiarlo y reusarlo.

```
public double calcularMontoTotalLlamadas(Cliente cliente) {
	double montoTotal = 0;
	for (Llamada llamada : cliente.getLlamadas()) {
		double montoAuxiliar = 0;
		if (llamada.getTipoDeLlamada() == "nacional") {
			// el precio es de 3 pesos por segundo más IVA sin adicional por establecer
la llamada
			montoAuxiliar += llamada.getDuracion() * 3 + (llamada.getDuracion() * 3 *
0.21);
		} else if (llamada.getTipoDeLlamada() == "internacional") {
			// el precio es de 150 pesos por segundo más IVA más 50 pesos por
establecer la llamada
			montoAuxiliar += llamada.getDuracion() * 150 + (llamada.getDuracion() * 150
* 0.21) + 50;
		}
		if (cliente.getTipo() == "fisica") {
			montoAuxiliar -= montoAuxiliar*descuentoClienteFisico;
		} else if(cliente.getTipo() == "juridica") {
			montoAuxiliar -= montoAuxiliar*descuentoClienteJuridico;
		}
		montoTotal += montoAuxiliar ;
	}
	return montoTotal;
}
```

Refactoring aplicado: Extract Method

```
public 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() * 150 + (llamada.getDuracion() * 150 * 0.21) + 50;
	}
	return 0;
}
public double calcularDescuentoCliente(Cliente cliente, double montoAuxiliar) {
	if (cliente.getTipo() == "fisica") {
		return montoAuxiliar*descuentoClienteFisico;
	} else if(cliente.getTipo() == "juridica") {
		return montoAuxiliar*descuentoClienteJuridico;
	}
	return 0;
}
public double calcularMontoTotalLlamadas(Cliente cliente) {
	double montoTotal = 0;
	for (Llamada llamada : cliente.getLlamadas()) {
		double montoAuxiliar = 0;
		montoAuxiliar += calcularCostoLlamada(llamada);
		montoAuxiliar -= calcularDescuentoCliente(cliente, montoAuxiliar);
		montoTotal += montoAuxiliar;
	}
	return montoTotal;
}
```

6. Bad smell: Data class y Feature Envy

El problema es que las clases Cliente y Llamada son clases que solo tienen variables y getters/setters para esas variables, actuando únicamente como contenedoras de datos. Además, la clase Empresa hace usos de esos getters/setters para realizar su trabajo, mostrandose envidiosa de las capacidades de ambas clases. Por otro lado, también usa un getter de la clase GestorNumerosDisponibles.

```
public class Cliente {
	private List<Llamada> llamadas = new ArrayList<Llamada>();
	private String tipo;
	private String nombre;
	private String numeroTelefono;
	private String cuit;
	private String dni;
	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 getNumeroTelefono() {
		return numeroTelefono;
	}
	public void setNumeroTelefono(String numeroTelefono) {
		this.numeroTelefono = numeroTelefono;
	}
	public String getCuit() {
		return cuit;
	}
	public void setCuit(String cuit) {
		this.cuit = cuit;
	}
	public String getDNI() {
		return dni;
	}
	public void setDNI(String dni) {
		this.dni = dni;
	}
	public List<Llamada> getLlamadas() {
		return llamadas;
	}
	public void setLlamadas(List<Llamada> llamadas) {
		this.llamadas = llamadas;
	}
}


public class Llamada {
	private String tipoDeLlamada;
	private String origen;
	private String destino;
	private int duracion;
```

```
	public Llamada(String tipoLlamada, String origen, String destino, int duracion) {
		this.tipoDeLlamada = tipoLlamada;
		this.origen= origen;
		this.destino= destino;
		this.duracion = duracion;
	}
	public String getTipoDeLlamada() {
		return tipoDeLlamada;
	}
	public String getDestino() {
		return destino;
	}
	public int getDuracion() {
		return this.duracion;
	}
	public String getOrigen() {
		return origen;
	}
}

public class GestorNumerosDisponibles {
	private SortedSet<String> numerosDisponibles = new TreeSet<String>();
	...
	public SortedSet<String> getNumerosDisponibles() {
		return numerosDisponibles;
	}
	...
}

public class Empresa {
	...
	private static double descuentoClienteJuridico = 0.15;
	private static double descuentoClienteFisico = 0;
	...
	public boolean agregarNumeroTelefono(String numero) {
		boolean encontre = guia.getNumerosDisponibles().contains(numero);
		if (!encontre) {
			guia.getNumerosDisponibles().add(numero);
			encontre= true;
			return encontre;
		}
		else {
			encontre= false;
			return encontre;
		}
	}
	public Cliente registrarUsuario(String DNIoCUIT, String nombre, String tipoDeCliente) {
		Cliente cliente = new Cliente();
		if (tipoDeCliente.equals("fisica")) {
			cliente.setNombre(nombre);
			String telefono = this.obtenerNumeroLibre();
			cliente.setTipo(tipoDeCliente);
			cliente.setNumeroTelefono(telefono);
			cliente.setDNI(DNIoCUIT);
		}
		else if (tipoDeCliente.equals("juridica")) {
			String telefono = this.obtenerNumeroLibre();
			cliente.setNombre(nombre);
			cliente.setTipo(tipoDeCliente);
```

```
                cliente.setNumeroTelefono(telefono);
                cliente.setCuit(DNIoCUIT);
            }
            clientes.add(cliente);
            return cliente;
      }
      ...
      public Llamada registrarLlamada(Cliente origen, Cliente destino, String tipoDeLlamada,
int duracion) {
            Llamada llamada = new Llamada(tipoDeLlamada, origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
            llamadas.add(llamada);
            origen.getLlamadas().add(llamada);
            return llamada;
      }

public 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() * 150 + (llamada.getDuracion() * 150 * 0.21) +
50;
            }
            return 0;
      }

      public double calcularDescuentoCliente(Cliente cliente, double montoAuxiliar) {
            if (cliente.getTipo() == "fisica") {
                  return montoAuxiliar*descuentoClienteFisico;
            } else if(cliente.getTipo() == "juridica") {
                  return montoAuxiliar*descuentoClienteJuridico;
            }
            return 0;
      }

      public double calcularMontoTotalLlamadas(Cliente cliente) {
            double montoTotal = 0;
            for (Llamada llamada : cliente.getLlamadas()) {
                  double montoAuxiliar = 0;
                  montoAuxiliar += calcularCostoLlamada(llamada);
                  montoAuxiliar -= calcularDescuentoCliente(cliente, montoAuxiliar);
                  montoTotal += montoAuxiliar;
            }
            return montoTotal;
      }
```

Refactoring aplicado: Move Method, Move Field

```
public class Cliente {
      ...
      public void agregarLlamada(Llamada llamada) {
            this.llamadas.add(llamada);
```

```
	}
	public double calcularDescuentoCliente(double montoAuxiliar) {
		if (this.getTipo() == "fisica") {
			return montoAuxiliar*descuentoClienteFisico;
		} else if(this.getTipo() == "juridica") {
			return montoAuxiliar*descuentoClienteJuridico;
		}
		return 0;
	}
	public double calcularMontoTotalLlamadas() {
		double montoTotal = 0;
		for (Llamada llamada : llamadas) {
			double montoAuxiliar = 0;
			montoAuxiliar += llamada.calcularCostoLlamada();
			montoAuxiliar -= this.calcularDescuentoCliente(montoAuxiliar);
			montoTotal += montoAuxiliar;
		}
		return montoTotal;
	}
}

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() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
		}
		return 0;
	}
}

public class GestorNumerosDisponibles {
	...
	public boolean existeNumero(String numero) {
		return this.numerosDisponibles.contains(numero);
	}
	public void agregarNumero(String numero) {
		this.numerosDisponibles.add(numero);
	}
	...
}


public class Empresa {
	private List<Cliente> clientes = new ArrayList<Cliente>();
	private List<Llamada> llamadas = new ArrayList<Llamada>();
	private GestorNumerosDisponibles guia = new GestorNumerosDisponibles();
...
	public boolean agregarNumeroTelefono(String numero) {
		boolean encontre = guia.existeNumero(numero);
		if (!encontre) {
			guia.agregarNumero(numero);
			encontre= true;
			return encontre;
		}
		else {
			encontre= false;
```

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

7. Bad smell: No existe constructor para inicializar los objetos

El problema es que en la clase Cliente no se implemento un constructor para que en el momento de instanciación de un objeto de tipo Cliente, sus variables internas tengan valores útiles, trabajo que hace luego la clase Empresa a través de setters

```
public class Empresa {
...
public Cliente registrarUsuario(String DNIoCUIT, String nombre, String tipoDeCliente) {
            Cliente cliente = new Cliente();
            if (tipoDeCliente.equals("fisica")) {
                  cliente.setNombre(nombre);
                  String telefono = this.obtenerNumeroLibre();
                  cliente.setTipo(tipoDeCliente);
                  cliente.setNumeroTelefono(telefono);
                  cliente.setDNI(DNIoCUIT);
            }
            else if (tipoDeCliente.equals("juridica")) {
                  String telefono = this.obtenerNumeroLibre();
                  cliente.setNombre(nombre);
                  cliente.setTipo(tipoDeCliente);
                  cliente.setNumeroTelefono(telefono);
                  cliente.setCuit(DNIoCUIT);
            }
            clientes.add(cliente);
            return cliente;
      }
...
```

Refactoring aplicado: Declare Constructor

No existe un refactoring llamado Declare Constructor, pero es un paso necesario para después poder aplicar el método Remove Setting Method.

```
public class Cliente {
      private List<Llamada> llamadas = new ArrayList<Llamada>();
      private String tipo;
      private String nombre;
      private String numeroTelefono;
      private String cuit;
      private String dni;
      private static double descuentoClienteJuridico = 0.15;
      private static double descuentoClienteFisico = 0;

      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;
      }
}

public class Empresa {
…
```

```
public Cliente registrarUsuario(String DNIoCUIT, String nombre, String tipoDeCliente) {
            Cliente cliente = null;
            if (tipoDeCliente.equals("fisica")) {
                  String telefono = this.obtenerNumeroLibre();
                  cliente = new Cliente(tipoDeCliente,nombre,telefono,"",DNIoCUIT);
            }
            else if (tipoDeCliente.equals("juridica")) {
                  String telefono = this.obtenerNumeroLibre();
                  cliente = new Cliente(tipoDeCliente,nombre,telefono,DNIoCUIT,"");
            }
            clientes.add(cliente);
            return cliente;
      }
...
}
```

8. Bad smell: Setters

Los valores de una variable deberían ser asignados cuando se construye el objeto, no en cualquier momento

```
public class Cliente {
	...
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public void setNumeroTelefono(String numeroTelefono) {
		this.numeroTelefono = numeroTelefono;
	}
	public void setCuit(String cuit) {
		this.cuit = cuit;
	}
	public void setDNI(String dni) {
		this.dni = dni;
	}
	public void setLlamadas(List<Llamada> llamadas) {
		this.llamadas = llamadas;
	}
}
```

Refactoring aplicado: Remove Setting Method

```
public class Cliente {
	private List<Llamada> llamadas = new ArrayList<Llamada>();
	private String tipo;
	private String nombre;
	private String numeroTelefono;
	private String cuit;
	private String dni;
	private static double descuentoClienteJuridico = 0.15;
	private static double descuentoClienteFisico = 0;

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;
	}
	public String getTipo() {
		return tipo;
	}
	public String getNombre() {
		return nombre;
	}
	public String getNumeroTelefono() {
		return numeroTelefono;
	}
	public String getCuit() {
		return cuit;
	}
	public String getDNI() {
```

```
            return dni;
      }
      public List<Llamada> getLlamadas() {
            return llamadas;
      }
      public static double getDescuentoClienteJuridico() {
            return descuentoClienteJuridico;
      }
      public static double getDescuentoClienteFisico() {
            return descuentoClienteFisico;
      }
      ...
}
```

9. Bad smell: Condicionales

El problema son los métodos en la clase Cliente y la clase Llamada que presentan sentencias condicionales que contienen lógica para diferentes tipos de objetos, es decir que se está usando una única clase para dos tipos distintos que comparten parte de su estructura. Genera uso de variables innecesarias, repetición de código y uso de condicionales. La solución es la herencia, para implementarla creamos subclases para cada tipo específico, heredando de la clase base la estructura en común 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, sino quedarian Data clases. Genera cambio en el metodo registrarUsuario y en el metodo registrarLlamada de la clase empresa. Aplico Encapsulate Field a las variables de instancia de la clase Llamada

```
public class Cliente {
...
public double calcularDescuentoCliente(double montoAuxiliar) {
            if (this.getTipo() == "fisica") {
                  return montoAuxiliar*descuentoClienteFisico;
            } else if(this.getTipo() == "juridica") {
                  return montoAuxiliar*descuentoClienteJuridico;
            }
            return 0;
      }
...
}

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() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
            }
            return 0;
      }
...
}
```

Refactoring aplicado: Replace Type code with Subclasses

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

      public Cliente(String nombre, String numeroTelefono) {
            this.nombre = nombre;
            this.numeroTelefono = numeroTelefono;
```

```
	}
	public String getNombre() {
		return nombre;
	}
	public String getNumeroTelefono() {
		return numeroTelefono;
	}
	public List<Llamada> getLlamadas() {
		return llamadas;
	}
	public abstract double calcularDescuentoCliente(double montoAuxiliar);
	public double calcularMontoTotalLlamadas() {
		double montoTotal = 0;
		for (Llamada llamada : llamadas) {
			double montoAuxiliar = 0;
			montoAuxiliar += llamada.calcularCostoLlamada();
			montoAuxiliar -= this.calcularDescuentoCliente(montoAuxiliar);
			montoTotal += montoAuxiliar;
		}
		return montoTotal;
	}
	public void agregarLlamada(Llamada llamada) {
		this.llamadas.add(llamada);
	}
}



public class ClienteJuridico extends Cliente {
	private String cuit;
	private static double descuentoClienteJuridico = 0.15;

public ClienteJuridico(String nombre, String numeroTelefono, String cuit) {
		super(nombre, numeroTelefono);
		this.cuit = cuit;
	}
	public String getCuit() {
		return cuit;
	}
	public static double getDescuentoClienteJuridico() {
		return descuentoClienteJuridico;
	}
	public double calcularDescuentoCliente(double montoAuxiliar) {
		return montoAuxiliar*descuentoClienteJuridico;
	}
}



public class ClienteFisico extends Cliente{

	private String dni;
	private static double descuentoClienteFisico = 0;

	public ClienteFisico(String nombre, String numeroTelefono, String dni) {
		super(nombre, numeroTelefono);
		this.dni = dni;
	}
	public String getDNI() {
```

```
		return dni;
	}
	public static double getDescuentoClienteFisico() {
		return descuentoClienteFisico;
	}
	public double calcularDescuentoCliente(double montoAuxiliar) {
		return montoAuxiliar*descuentoClienteFisico;
	}
}

public class Empresa {
...
public Cliente registrarUsuarioFisico(String dni, String nombre) {
		String telefono = this.obtenerNumeroLibre();
		Cliente cliente = new ClienteFisico(nombre,telefono,dni);
		clientes.add(cliente);
		return cliente;
	}

	public Cliente registrarUsuarioJuridico(String cuit, String nombre) {
		String telefono = this.obtenerNumeroLibre();
		Cliente cliente = new ClienteJuridico(nombre,telefono,cuit);
		clientes.add(cliente);
		return cliente;
	}
...
}

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 String getDestino() {
		return destino;
	}
	public int getDuracion() {
		return this.duracion;
	}
	public String getOrigen() {
		return origen;
	}
	public abstract double calcularCostoLlamada();
}


public class LlamadaNacional extends Llamada {

	public LlamadaNacional(String origen, String destino, int duracion) {
		super (origen,destino,duracion);
	}

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

```
}


public class LlamadaInternacional extends Llamada {

	public LlamadaInternacional(String origen, String destino, int duracion) {
		super (origen,destino,duracion);
	}

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

}


public class Empresa {

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

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

Debido a los cambios realizados en la clase Empresa, el test EmpresaTest debe ser actualizado:

Previo al refactoring:

```
class EmpresaTest {
	Empresa sistema;
	...
	@Test
	void testcalcularMontoTotalLlamadas() {
		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");
		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);
		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));
	}
	@Test
	void testAgregarUsuario() {
		assertEquals(this.sistema.cantidadDeUsuarios(), 0);
		this.sistema.agregarNumeroTelefono("2214444558");
		Cliente nuevaPersona = this.sistema.registrarUsuario("2444555","Alan Turing",
"fisica");
		assertEquals(1, this.sistema.cantidadDeUsuarios());
		assertTrue(this.sistema.existeUsuario(nuevaPersona));
	}
	...
}
```

Después del refactoring:

```
class EmpresaTest {
	Empresa sistema;
	...
	@Test
	void testcalcularMontoTotalLlamadas() {
		Cliente emisorPersonaFisca = sistema.registrarUsuarioFisico("11555666", "Brendan
Eich");
		Cliente remitentePersonaFisica = sistema.registrarUsuarioFisico("00000001", "Doug
Lea");
		Cliente emisorPersonaJuridica = sistema.registrarUsuarioJuridico("17555222",
"Nvidia Corp");
		Cliente remitentePersonaJuridica = sistema.registrarUsuarioJuridico("25765432",
"Sun Microsystems");
		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);
		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));
	}
	@Test
	void testAgregarUsuario() {
		assertEquals(this.sistema.cantidadDeUsuarios(), 0);
		this.sistema.agregarNumeroTelefono("2214444558");
		Cliente nuevaPersona = this.sistema.registrarUsuarioFisico("2444555","Alan
Turing");
		assertEquals(1, this.sistema.cantidadDeUsuarios());
		assertTrue(this.sistema.existeUsuario(nuevaPersona));
	}
}
```

10. Bad smell: Duplicated Code

El problema es que el metodo obtenerNumeroLibre() de la clase GestorNumerosDisponibles repite las mismas dos sentencias para cada “case” de la estructura de control switch

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

Refactoring: Consolidate Duplicate Conditional Fragment

```
public class GestorNumerosDisponibles {
	...

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

11. Bad smell: Switch Statements

El problema es el método en la clase GestorNumerosDisponibles que presenta la sentencia Switch la cual contiene lógica para diferentes tipos de objetos, es decir que se esta usando una unica clase para tres tipos distintos que comparten parte de su estructura. Genera uso de variables innecesarias, repetición de código y uso de condicionales. La solucion es la herencia.

```
public class GestorNumerosDisponibles {
	...

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

Refactoring aplicado: Replace Conditional with Polimorfism

```
public class GestorNumerosDisponibles {
	private Generador generador = new GeneradorUltimo();
	...
	public String obtenerNumeroLibre() {
		String numero = generador.obtenerNumeroLibre(numerosDisponibles);
		numerosDisponibles.remove(numero);
		return numero;
	}
	public void cambiarTipoGenerador(Generador tipoGenerador) {
		this.generador = tipoGenerador;
	}
}

public interface Generador {
	public abstract String obtenerNumeroLibre(SortedSet<String> numerosDisponibles);
}
```

```
public class GeneradorPrimero implements Generador {
	@Override
public String obtenerNumeroLibre(SortedSet<String> numerosDisponibles) {
		return numerosDisponibles.first();
	}
}

public class GeneradorRandom implements Generador {
	@Override
```

```
	public String obtenerNumeroLibre(SortedSet<String> numerosDisponibles) {
		return new ArrayList<String>(numerosDisponibles).get(new
Random().nextInt(numerosDisponibles.size()));
	}
}

public class GeneradorUltimo implements Generador {
	@Override
	public String obtenerNumeroLibre(SortedSet<String> numerosDisponibles) {
		return numerosDisponibles.last();
	}
}
```

Al haber realizado este cambio necesitamos cambiar la clase EmpresaTest.
Previo al refactoring:

```
class EmpresaTest {
	@Test
	void obtenerNumeroLibre() {
		// 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());
	}
}
```

Después del refactoring:

```
class EmpresaTest {
@Test
	void obtenerNumeroLibre() {
		// por defecto es el ultimo
		assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
		this.sistema.getGestorNumeros().cambiarTipoGenerador(new GeneradorPrimero());
		assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
		this.sistema.getGestorNumeros().cambiarTipoGenerador(new GeneradorRandom());
		assertNotNull(this.sistema.obtenerNumeroLibre());
	}
}
```

12. Bad smell: Duplicated Code

El problema es que el método agregarNumeroTelefono() de la clase Empresa repite líneas de código dentro de la estructura de control ‘if’.

```
public class Empresa {
public boolean agregarNumeroTelefono(String numero) {
            boolean encontre = guia.existeNumero(numero);
            if (!encontre) {
                  guia.agregarNumero(numero);
                  encontre= true;
                  return encontre;
            }
            else {
                  encontre= false;
                  return encontre;
            }
      }
```

Refactoring: Substitute Algorithm

```
public class Empresa {
public boolean agregarNumeroTelefono(String numero) {
      boolean encontre = guia.existeNumero(numero);
            if (!encontre) {
                  guia.agregarNumero(numero);
                  return true;
            }
            return false;
      }
```

13. Bad smell: Temporary Field

La variable ‘telefono’ obtiene un valor que es utilizado una única vez en la creación de un objeto de tipo Cliente. Tambien se puede eliminar del metodo agregarNumeroTelefono(), la variable boolean ‘encontre’, utilizando su contenido como condición del ‘if’

```
public class Empresa {
	public Cliente registrarUsuarioFisico(String dni, String nombre) {
		String telefono = this.obtenerNumeroLibre();
		Cliente cliente = new ClienteFisico(nombre,telefono,dni);
		clientes.add(cliente);
		return cliente;
	}

	public Cliente registrarUsuarioJuridico(String cuit, String nombre) {
		String telefono = this.obtenerNumeroLibre();
		Cliente cliente = new ClienteJuridico(nombre,telefono,cuit);
		clientes.add(cliente);
		return cliente;
	}

public boolean agregarNumeroTelefono(String numero) {
	boolean encontre = guia.existeNumero(numero);
		if (!encontre) {
			guia.agregarNumero(numero);
			return true;
		}
		return false;
	}
}
```

Refactoring aplicado: Replace Temp With Query

```
public class Empresa {
	public Cliente registrarUsuarioFisico(String dni, String nombre) {
		Cliente cliente = new ClienteFisico(nombre,this.obtenerNumeroLibre(),dni);
		clientes.add(cliente);
		return cliente;
	}
	public Cliente registrarUsuarioJuridico(String cuit, String nombre) {
		Cliente cliente = new ClienteJuridico(nombre,this.obtenerNumeroLibre(),cuit);
		clientes.add(cliente);
		return cliente;
	}

	public boolean agregarNumeroTelefono(String numero) {
		if (!guia.existeNumero(numero)) {
			guia.agregarNumero(numero);
			return true;
		}
		return false;
	}

}
```

14. Bad smell: Dead Code

El problema es la variable de instancia llamadas de la clase Empresa, ya que esta no se usa más que para guardar las llamadas, las cuales podrían ser obtenidas desde la lista de todos sus clientes.

```
public class Empresa {
	…
	private List<Llamada> llamadas = new ArrayList<Llamada>();
…
public Llamada registrarLlamadaNacional(Cliente origen, Cliente destino, int duracion) {
		Llamada llamada = new LlamadaNacional(origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
		llamadas.add(llamada);
		origen.agregarLlamada(llamada);
		return llamada;
	}

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

}
```

Refactoring aplicado: Remove Dead Code

Se eliminó la variable de instancia y su uso en los métodos registrarLlamadaInternacional() y registrarLlamadaNacional()

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

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

15. Bad smell: Feature Envy

El problema es que la clase Empresa al agregar una llamada está manipulando una colección que no le pertenece (llamadas del cliente). Por ende, una llamada debe ser creada y agregada por la clase Cliente. Además, la clase Empresa, al agregar un número de teléfono disponible, utiliza mucha lógica de la clase GestorNumerosDisponibles.

```
public class Empresa {
	...
	public boolean agregarNumeroTelefono(String numero) {
		if (!guia.existeNumero(numero)) {
			guia.agregarNumero(numero);
			return true;
		}
		return false;
	}
	public String obtenerNumeroLibre() {
		return guia.obtenerNumeroLibre();
	}
	public Cliente registrarUsuarioFisico(String dni, String nombre) {
		Cliente cliente = new ClienteFisico(nombre,this.obtenerNumeroLibre(),dni);
		clientes.add(cliente);
		return cliente;
	}
	public Cliente registrarUsuarioJuridico(String cuit, String nombre) {
		Cliente cliente = new ClienteJuridico(nombre,
	this.obtenerNumeroLibre(),cuit);
		clientes.add(cliente);
		return cliente;
	}
	public Llamada registrarLlamadaNacional(Cliente origen, Cliente destino, int
duracion) {
		Llamada llamada = new LlamadaNacional(origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
		origen.agregarLlamada(llamada);
		return llamada;
	}

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

Refactoring: Move Method

```
public class Empresa {
	public Cliente registrarUsuarioFisico(String dni, String nombre) {
		Cliente cliente = new ClienteFisico(nombre,
	this.guia.obtenerNumeroLibre(),dni);
		clientes.add(cliente);
		return cliente;
	}
```

```
	public Cliente registrarUsuarioJuridico(String cuit, String nombre) {
		Cliente cliente = new ClienteJuridico(nombre,
	this.guia.obtenerNumeroLibre(),cuit);
		clientes.add(cliente);
		return cliente;
	}
}
public abstract class Cliente {
	public Llamada registrarLlamadaNacional(Cliente destino, int duracion) {
		Llamada llamada = new LlamadaNacional(this.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
		this.agregarLlamada(llamada);
		return llamada;
	}
	public Llamada registrarLlamadaInternacional(Cliente destino, int duracion) {
		Llamada llamada = new LlamadaInternacional(this.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
		this.agregarLlamada(llamada);
		return llamada;
	}
}
public class GestorNumerosDisponibles {
	...
	public boolean agregarNumeroTelefono(String numero) {
		if (!this.existeNumero(numero)) {
			this.agregarNumero(numero);
			return true;
		}
		return false;
	}
}
```

Debido a los cambios realizados el test debe ser modificado

Previo al refactoring:

```
class EmpresaTest {
	Empresa sistema;
	@BeforeEach
	public void setUp() {
		this.sistema = new Empresa();
		this.sistema.agregarNumeroTelefono("2214444554");
		this.sistema.agregarNumeroTelefono("2214444555");
		this.sistema.agregarNumeroTelefono("2214444556");
		this.sistema.agregarNumeroTelefono("2214444557");
		this.sistema.agregarNumeroTelefono("2214444558");
		this.sistema.agregarNumeroTelefono("2214444559");
	}

	@Test
	void testcalcularMontoTotalLlamadas() {
		Cliente emisorPersonaFisca = sistema.registrarUsuarioFisico("11555666",
	"Brendan Eich");
		Cliente remitentePersonaFisica = sistema.registrarUsuarioFisico("00000001",
	"Doug Lea");
		Cliente emisorPersonaJuridica = sistema.registrarUsuarioJuridico("17555222",
	"Nvidia Corp");
		Cliente remitentePersonaJuridica =
	sistema.registrarUsuarioJuridico("25765432", "Sun Microsystems");
```

```
            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);
            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));
      }

      @Test
      void testAgregarUsuario() {
            assertEquals(this.sistema.cantidadDeUsuarios(), 0);
            this.sistema.agregarNumeroTelefono("2214444558");
            Cliente nuevaPersona = this.sistema.registrarUsuarioFisico("2444555","Alan
      Turing");
            assertEquals(1, this.sistema.cantidadDeUsuarios());
            assertTrue(this.sistema.existeUsuario(nuevaPersona));
      }
      @Test
      void obtenerNumeroLibre() {
            // por defecto es el ultimo
            assertEquals("2214444559", this.sistema.obtenerNumeroLibre());
            this.sistema.getGestorNumeros().cambiarTipoGenerador(new
      GeneradorPrimero());
            assertEquals("2214444554", this.sistema.obtenerNumeroLibre());
            this.sistema.getGestorNumeros().cambiarTipoGenerador(new GeneradorRandom());
            assertNotNull(this.sistema.obtenerNumeroLibre());
      }
}
```

Después del refactoring:

```
class EmpresaTest {
      Empresa sistema;
      @BeforeEach
      public void setUp() {
            this.sistema = new Empresa();
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444554");
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444555");
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444556");
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444557");
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444558");
            this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444559");
```

```
	}
	@Test
	void testcalcularMontoTotalLlamadas() {
		Cliente emisorPersonaFisca = sistema.registrarUsuarioFisico("11555666",
"Brendan Eich");
		Cliente remitentePersonaFisica = sistema.registrarUsuarioFisico("00000001",
"Doug Lea");
		Cliente emisorPersonaJuridica = sistema.registrarUsuarioJuridico("17555222",
"Nvidia Corp");
		Cliente remitentePersonaJuridica =
sistema.registrarUsuarioJuridico("25765432", "Sun Microsystems");
		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);
		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));
	}
	@Test
	void testAgregarUsuario() {
		assertEquals(this.sistema.cantidadDeUsuarios(), 0);
		this.sistema.getGestorNumeros().agregarNumeroTelefono("2214444558");
		Cliente nuevaPersona = this.sistema.registrarUsuarioFisico("2444555","Alan
Turing");
		assertEquals(1, this.sistema.cantidadDeUsuarios());
		assertTrue(this.sistema.existeUsuario(nuevaPersona));
	}
	@Test
	void obtenerNumeroLibre() {
		// por defecto es el ultimo
		assertEquals("2214444559",this.sistema.getGestorNumeros().obtenerNumeroLibre
		());
		this.sistema.getGestorNumeros().cambiarTipoGenerador(new
	GeneradorPrimero());
		assertEquals("2214444554",
	this.sistema.getGestorNumeros().obtenerNumeroLibre());
		this.sistema.getGestorNumeros().cambiarTipoGenerador(new GeneradorRandom());
		assertNotNull(this.sistema.getGestorNumeros().obtenerNumeroLibre());
	}
}
```

16. Bad smell: Reinventa la rueda

El problema es que en la clase Empresa se recorre la lista llamadas con la estructura de control for, cuando es mucho más práctico y fácil utilizar streams y expresiones lambda que provee java.

```
public abstract class Cliente {
…
public double calcularMontoTotalLlamadas() {
		double montoTotal = 0;
		for (Llamada llamada : llamadas) {
			double montoAuxiliar = 0;
			montoAuxiliar += llamada.calcularCostoLlamada();
			montoAuxiliar -= this.calcularDescuentoCliente(montoAuxiliar);
			montoTotal += montoAuxiliar;
		}
		return montoTotal;
	}
…
```

Refactoring aplicado: Replace Loop with Pipeline

```
public abstract class Cliente {
	...
	public double calcularMontoTotalLlamadas() {
		return llamadas.stream()
				.mapToDouble(llamada -> llamada.calcularCostoLlamada() -
this.calcularDescuentoCliente(llamada.calcularCostoLlamada()))
				.sum();
	}
	...
}
```