Mal olor: switch statements.

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

Refactoring: replace conditional logic with strategy.

Creamos la clase strategy y las respectivas subclases que son las concreteStrategy

```
public abstract class GestorNumerosDisponibles {
      private SortedSet<String> lineas = new TreeSet<String>();
      private GestorNumerosDisponibles tipoGenerador = new
GestorNumerosDisponiblesUltimo;

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

      public abstract String obtenerNumeroLibre();

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

public class GestorNumerosDisponiblesUltimo extends
GestorNumerosDisponibles{
```

```
    public String obtenerNumeroLibre() {
           String linea = getLineas().last();
           getLineas().remove(linea);
           return linea;
     }
}

public class GestorNumerosDisponiblesPrimero extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           String linea = getLineas().first();
           getLineas().remove(linea);
           return linea;
     }
}

public class GestorNumerosDisponiblesRandom extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           String linea = new ArrayList<String>(getLineas())
                                  .get(new
Random().nextInt(getLineas().size()));
           lineas.remove(getLineas());
           return linea;
    }
}
```

Mal olor: Mala delegacion de las tareas

```
public abstract class GestorNumerosDisponibles {
     private SortedSet<String> lineas = new TreeSet<String>();
     private GestorNumerosDisponibles tipoGenerador = new
GestorNumerosDisponiblesUltimo;

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

Refactoring: Move method a empresa

```
public class Empresa {
     private List<Cliente> clientes = new ArrayList<Cliente>();
     private List<Llamada> llamadas = new ArrayList<Llamada>();
     private GestorNumerosDisponibles guia = new
GestorNumerosDisponiblesUltimo();

     static double descuentoJur = 0.15;
```

```
	static double descuentoFis = 0;

	public void cambiarTipoGenerador(GestorNumerosDisponibles valor)
{
		SortedSet<String> lineas = guia.getLineas();
		this.guia = valor;
		guia.setLineas(lineas);
	}

public abstract class GestorNumerosDisponibles {
	private SortedSet<String> lineas = new TreeSet<String>();

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

    public void setLineas(SortedSet<String> lineas) {
		this.lineas = lineas;
	}

	public abstract String obtenerNumeroLibre();
}
```

Test modificado: al modificar el case con subclases ahora el metodo cambiarTipoGenerador recibe por parametro un tipo de clase GestorNumerosDisponibles.

```
@Test
	void obtenerNumeroLibre() {
		// por defecto es el ultimo
		assertEquals("2214444559",
this.sistema.obtenerNumeroLibre());

		this.sistema.cambiarTipoGenerador(new
GestorNumeroDisponiblePrimero());

		assertEquals("2214444554",
this.sistema.obtenerNumeroLibre());

		this.sistema.cambiarTipoGenerador(new
GestorNumeroDisponibleRandom());
		assertNotNull(this.sistema.obtenerNumeroLibre());
	}
```

Mal olor: Duplicated Code

```
public class GestorNumerosDisponiblesUltimo extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           String linea = getLineas().last();
           getLineas().remove(linea);
           return linea;
     }
}

public class GestorNumerosDisponiblesPrimero extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           String linea = getLineas().first();
           getLineas().remove(linea);
           return linea;
     }
}

public class GestorNumerosDisponiblesRandom extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           String linea = new ArrayList<String>(getLineas())
                                    .get(new
Random().nextInt(getLineas().size()));
           getLineas.remove(linea);
           return linea;
    }
}
```

Refactoring: From template method

Refactoring 1: Compose method

Como buena práctica eliminamos las variables temporales que no eran necesarias en los métodos ya que el resultado que almacenaban ahora es retornado directamente.

```
public class GestorNumerosDisponiblesUltimo extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           return getLineas().last();
     }

    public void borrarLinea(String linea) {
           getLineas.remove(linea);
     }
}
```

```
public class GestorNumerosDisponiblesPrimero extends
GestorNumerosDisponibles{
    public String obtenerNumeroLibre() {
           return getLineas().first();
     }

    public void borrarLinea(String linea) {
           getLineas.remove(linea);
     }
}

public class GestorNumerosDisponiblesRandom extends
GestorNumerosDisponibles{
        public String obtenerNumeroLibre() {
           return new ArrayList<String>(getLineas())
                                    .get(new
Random().nextInt(getLineas().size()));
    }

        public void borrarLinea(String linea) {
           getLineas.remove(linea);
     }
}
```

Refactoring 2: Pull up method

```
public abstract class GestorNumerosDisponibles {
     private SortedSet<String> lineas = new TreeSet<String>();

    public abstract String obtenerNumeroLibre();

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

    public void setLineas(SortedSet<String> lineas) {
           this.lineas = lineas;
     }

     public String obtenerNumero(){
        String linea = this.obtenerNumeroLibre()
     borrarLinea(linea);
           return linea;
     }

    public void borrarLinea(String linea) {
           getLineas().remove(linea);
     }
}
```

Mal olor: Los nombres no describen el comportamiento de los metodos/variables

```
public class Llamada{
    public String getRemitente()
}
```

Refactor: Rename method

```
public class Llamada{
    public String getDestino()
}
```

Mal olor: Se rompe en encapsulamiento

```
public class Cliente {
     public 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;
	}
}
```

Refactoring: Crear un constructor

```
    public Cliente(String tipo, String nombre, String cuit, String
dni) {
		this.llamadas = new ArrayList<Llamada>();
		this.tipo = tipo;
		this.nombre = nombre;
		this.cuit = cuit;
		this.dni = dni;
	}
```

Mal olor: Switch statements.

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

Refactoring 1: replace conditional with polymorfism.

```
public class Cliente {
	public List<Llamada> llamadas;
	private String nombre;
	private String numeroTelefono;
```

```
	public Cliente(String nombre) {
		this.llamadas = new ArrayList<Llamada>();
		this.nombre = nombre;
	}

public class ClienteFisica extends Cliente{
	private String dni;

	public ClienteFisica(String nombre, String dni) {
		super(nombre);
		this.dni = dni;
	}
	public String getDNI() {
		return dni;
	}
}

public class ClienteJuridica extends Cliente{
	private String cuit;

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

	public String getCuit() {
		return cuit;
	}
}
```

Refactoring 2: introducir un objeto por parametro.

Este segundo paso lo hacemos porque los clientes pueden ser de 2 tipos diferentes.

```
public class Empresa {
    public Cliente registrarUsuario(Cliente cliente) {
		cliente.setNumeroTelefono(this.obtenerNumeroLibre());
	   clientes.add(cliente);
		return cliente;
	}
}
```

Test: Pasa un objeto usuario por parametro

```
	@Test
	void testAgregarUsuario() {
		assertEquals(this.sistema.cantidadDeUsuarios(), 0);
```

```
            this.sistema.agregarNumeroTelefono("2214444558");
            Cliente nuevaPersona = new ClienteFisica("Alan Turing",
"2444555");
            sistema.registrarUsuario(nuevaPersona);

            assertEquals(1, this.sistema.cantidadDeUsuarios());
            assertTrue(this.sistema.existeUsuario(nuevaPersona));
     }
```

Mal olor: Lista de parametros larga

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

Refactoring: Introuducir como parametro un objeto

```
public class Empresa {
    public Llamada registrarLlamada(Cliente origen, Llamada llamada) {
            llamadas.add(llamada);
            origen.llamadas.add(llamada);
            return llamada;
    }
}
```

Mal olor: Mala delegacion de las tareas

```
public class Empresa {
    public Llamada registrarLlamada(Cliente origen, Llamada llamada) {
            llamadas.add(llamada);
            origen.llamadas.add(llamada);
            return llamada;
     }
}
```

Refactoring: Move method

```
public class Cliente
public void agregarLlamada(Llamada llamada) {
```

```
            this.llamadas.add(llamada);
      }

public class Empresa
      public Llamada registrarLlamada(Cliente origen, Llamada llamada)
{
            origen.agregarLlamada(llamada);
            return llamada;
      }
```

Test: En el registrar llamada pasamos un objeto llamada

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

            this.sistema.registrarLlamada(emisorPersonaJuridica, new
LlamadaNacional (emisorPersonaJuridica.getNumeroTelefono(),
remitentePersonaFisica.getNumeroTelefono(), 10));
            this.sistema.registrarLlamada(emisorPersonaJuridica, new
LlamadaInternacional (emisorPersonaJuridica.getNumeroTelefono(),
remitentePersonaFisica.getNumeroTelefono(), 8));
            this.sistema.registrarLlamada(emisorPersonaJuridica, new
LlamadaNacional(emisorPersonaJuridica.getNumeroTelefono(),
remitentePersonaJuridica.getNumeroTelefono(), 5));
            this.sistema.registrarLlamada(emisorPersonaJuridica, new
LlamadaInternacional (emisorPersonaJuridica.getNumeroTelefono(),
remitentePersonaJuridica.getNumeroTelefono(), 7));
            this.sistema.registrarLlamada(emisorPersonaFisca, new
LlamadaNacional (emisorPersonaFisca.getNumeroTelefono(),
remitentePersonaFisica.getNumeroTelefono(), 15));
            this.sistema.registrarLlamada(emisorPersonaFisca, new
LlamadaInternacional (emisorPersonaFisca.getNumeroTelefono(),
remitentePersonaFisica.getNumeroTelefono(), 45));
            this.sistema.registrarLlamada(emisorPersonaFisca, new
LlamadaNacional (emisorPersonaFisca.getNumeroTelefono(),
remitentePersonaJuridica.getNumeroTelefono(), 13));
```

```
          this.sistema.registrarLlamada(emisorPersonaFisca, new
LlamadaInternacional (emisorPersonaFisca.getNumeroTelefono(),
remitentePersonaJuridica.getNumeroTelefono(), 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));
     }
```

Mal olor: switch statements

```
public class Empresa {
    public double calcularMontoTotalLlamadas(Cliente cliente){
    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;
        }
}
```

Refactoring 1: replace conditional with polymorfism.

```
public 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;
     }

    pubic abstract double getPrecioMasIVA():

public class LlamadaNacional extends Llamada{
```

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

    pubic double getPrecioMasIVA(){
        return this.duracion() * 150 + (this.duracion() * 150 * 0.21);
    }
}
public class LlamadaInternacional extends Llamada{
      public LlamadaInternacional(String origen, String destino, int
duracion, double iva, double precio){
            super(origen, destino, duracion, iva, precio);
      }

    pubic double getPrecioMasIVA()[
        return this.duracion() * 150 + (this.duracion() * 150 * 0.21)
+ 50;
    ]
}

class Empresa {
    public double calcularMontoTotalLlamadas(Cliente cliente) {
        double auxc = l.getPrecioMasIVA();
    }
}
```

Mal olor: se usan directamente valor fijos que deberian ser metodos

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

    pubic double getPrecioMasIVA(){
        return this.duracion() * 3 + (this.duracion() * 3 * 0.21);
    }
}
public class LlamadaInternacional extends Llamada{
      public LlamadaInternacional(String origen, String destino, int
duracion, double iva, double precio){
            super(origen, destino, duracion, iva, precio);
      }

    pubic double getPrecioMasIVA()[
```

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

Refactoring: Replace Temp with Query

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

    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA());
    }

    public double getPrecio() {
           return 3;

     }

    private double getIVA(){
        return 0.21;
    }

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

public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA()+50);
    }

     public double getPrecio() {
           return 150;

    private double getIVA(){
        return 0.21;
    }
}
```

Mal olor: codigo repetido

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

    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA());
    }

    public double getPrecio() {
           return 3;

      }

    private double getIVA(){
        return 0.21;
    }

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

    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA()+50);
    }

      public double getPrecio() {
           return 150;

    private double getIVA(){
        return 0.21;
    }
}
```

Refactoring: From temple method

Paso 1: Compose method

```
public abstarct 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 getPrecioMasIVA();


public class LlamadaNacional extends Llamada{
     public LlamadaNacional(String origen, String destino, int
duracion, double precio) {
          super(origen, destino, duracion, precio);
     }
    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA() + this.getAdicional());
    }
     public double getPrecio() {
          return 3;
     }
    private double getIVA(){
        return 0.21;
    }
    public double getAdional(){
        return 0;
    }


}
public class LlamadaInternacional extends Llamada{
     public LlamadaInternacional(String origen, String destino, int
duracion, double precio){
          super(origen, destino, duracion,  precio);
     }
    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA() + this.getAdicional());
    }
    private double getIVA(){
        return 0.21;
    }
     public double getPrecio() {
          return 150;
     }
    public double getAdional(){
        return 50;
    }
```

Paso 2: Pull up

```
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 getPrecio();
    public abstract double getAdional();

    public double getPrecioMasIVA(){
        return this.duracion * getPrecio() + (this.duracion *
getPrecio() * this.getIVA() + this.getAdicional());
    }

    private double getIVA(){
        return 0.21;
    }


]

public class LlamadaNacional extends Llamada{
      public LlamadaNacional(String origen, String destino, int
duracion, double precio) {
            super(origen, destino, duracion, precio);
      }
      public double getPrecio() {
            return 3;
      }
    public double getAdional(){
        return 0;
    }


}
public class LlamadaInternacional extends Llamada{
      public LlamadaInternacional(String origen, String destino, int
duracion, double precio){
            super(origen, destino, duracion,  precio);
      }
      public double getPrecio() {
            return 150;
      }
```

```
    public double getAdional(){
        return 50;
    }
```

Mal olor: variables mal delegadas

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

Refactoring: move variable.

```
public class ClienteJuridica extends Cliente{
     private String cuit;
     static double descuentoJur = 0.15;

public class ClienteFisica extends Cliente{
     private String dni;
     static double descuentoFis = 0;
```

Mal olor: variables de instancia que deberian ser temporales

```
public class ClienteJuridica extends Cliente{
     private String cuit;
     static double descuentoJur = 0.15;

public class ClienteFisica extends Cliente{
     private String dni;
     static double descuentoFis = 0;
```

Refactoring: Replace Temp with Query

```
public abstract class Cliente {
     public abstract double getDescuento();

public class ClienteJuridica extends Cliente{
     private String cuit;

     public double getDescuento() {
          return 0.15;
}

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

```
      public double getDescuento() {
            return 0;
}
```

Mal olor: Switch staments

```
public class Empresa{
public double calcularMontoTotalLlamadas(Cliente cliente){
    if (cliente.getTipo() == "fisica") {
                        auxc -= auxc*descuentoFis;
                  } else if(cliente.getTipo() == "juridica") {
                        auxc -= auxc*descuentoJur;
                  }
}
```

Refactoring: Repalce conditional with polymorfism (hicimos la jerarquia de clientes en pasos anteriores)

```
public class Empresa{
public double calcularMontoTotalLlamadas(Cliente cliente){
auxc -= auxc* cliente.getDescuento();
}
```

Mal olor: Envidia de atributos y mala delegacion de tareas

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

Refactoring: Move method

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

```
     }

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

Mal olor: Los nombres de las variables no indican su rol

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

Refactoring: Rename variable

```
public class Cliente{
    public double calcularMontoTotalLlamadas() {
           double montoTotal = 0;
           for (Llamada l : llamadas) {
                double precioLlamada = l.getPrecioMasIVA();
                precioLlamada -= precioLlamada*this.getDescuento();
                montoTotal += precioLlamada;
           }
           return montoTotal;
     }
}
```

Mal olor: Reinventa la rueda

```
public class Cliente{
    public double calcularMontoTotalLlamadas() {
           double montoTotal = 0;
           for (Llamada l : llamadas) {
                double precioLlamada = l.getPrecioMasIVA();
                precioLlamada -= precioLlamada*this.getDescuento();
                montoTotal += precioLlamada;
           }
           return montoTotal;
     }
}
```

Refactoring: Replace Loop with Pipeline

```
public class Cliente{
    public double calcularMontoTotalLlamadas() {
        return llamadas.stream()
            .mapToDouble(l -> l.getPrecioMasIVA() * (1 -
this.getDescuento()))
            .sum();
    }
}
```

Mal olor: Método largo

```
public class 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;
           }
     }
}
```

Refactor: Extract method

Acomodamos el if para que sea mas legible y eliminamos variables necesarios que no eran necesarias en este metodo corto.

```
public class Empresa{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else {
                 guia.getLineas().add(str);
                 return true;
           }
     }

    private boolean encontreNumero(String str){
        return guia.getLineas().contains(str);
    }
}
```

Mal olor: Envidia de atributos, romper encapsulamiento y mala delegacion

```
public class Empresa{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else {
                 guia.getLineas().add(str);
                 return true;
           }
     }
```

Refactoring: Move method

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

public class GestorNumeroDisponible{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else {
                 this.getLineas().add(str);
                 return true;
           }
     }

      private boolean encontreNumero(String str){
```

```
          return this.getLineas().contains(str);
     }
```

Mal olor: Metodo largo

```
public class GestorNumeroDisponible{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else {
                 this.getLineas().add(str);
                 return true;
           }
     }
```

Refactoring: Extract method

```
public class GestorNumeroDisponible{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else
                 return agregarNumero(str);
    }

    private boolean agregarNumero(String str){
        this.getLineas().add(str);
           return true;
    }
     }
```

Mal olor: Nombres de las variables no indican su rol

```
public class GestorNumeroDisponible{
    public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else
                 return agregarNumero(str);
    }

    private boolean agregarNumero(String str){
        this.getLineas().add(str);
           return true;
```

```
    }
     private boolean encontreNumero(String str){
         return this.getLineas().contains(str);
     }
```

Refactoring: Rename variable

```
public class GestorNumeroDisponible{
    public boolean agregarNumeroTelefono(String numeroNuevo) {
           if (encontreNumero(numeroNuevo))
                 return false;
           else
                 return agregarNumero(numeroNuevo);
    }

    private boolean agregarNumero(String numeroNuevo){
        this.getLineas().add(numeroNuevo);
           return true;
    }

     private boolean encontreNumero(String numeroNuevo){
         return this.getLineas().contains(numeroNuevo);
     }
```

Mal olor: Nombres de las variables no indican su rol

```
public class Empresa {
     private List<Cliente> clientes = new ArrayList<Cliente>();
     private List<Llamada> llamadas = new ArrayList<Llamada>();
     private GestorNumerosDisponibles guia = new
GestorNumeroDisponibleUltimo();


     public boolean agregarNumeroTelefono(String str) {
           if (encontreNumero(str))
                 return false;
           else {
                 guia.getLineas().add(str);
                 return true;
           }
     }

     private boolean encontreNumero(String str){
         return guia.getLineas().contains(str);
     }
```

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

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

	public void cambiarTipoGenerador(GestorNumerosDisponibles valor)
{
		SortedSet<String> lineas = guia.getLineas();
		this.guia = valor;
		guia.setLineas(lineas);
	}
}
```

Refactoring: Rename variable

```
public class Empresa
public boolean agregarNumeroTelefono(String numeroNuevo) {
		if (encontreNumero(numeroNuevo))
			return false;
		else {
			guiaNumeros.getLineas().add(numeroNuevo);
			return true;
		}
	}

	private boolean encontreNumero(String numero){
	    return guiaNumeros.getLineas().contains(numero);
	}

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

	public Cliente registrarUsuario(Cliente cliente) {
		cliente.setNumeroTelefono(this.obtenerNumeroLibre());
		clientes.add(cliente);
		return cliente;
	}

	public Llamada registrarLlamada(Cliente origen, Llamada llamada)
{
		origen.agregarLlamada(llamada);
		return llamada;
	}

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

```
	}

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

	public boolean existeUsuario(Cliente persona) {
		return clientes.contains(persona);
	}

	public GestorNumerosDisponibles getGestorNumeros() {
		return this.guiaNumeros;
	}

	public void cambiarTipoGenerador(GestorNumerosDisponibles
guiaNumerosNueva) {
		SortedSet<String> lineas = guiaNumeros.getLineas();
		this.guiaNumeros = guiaNumerosNueva;
		guiaNumeros.setLineas(lineas);
	}
}
```