EjerciciosRefactoring

Ejercicios

Refactoring

Ejercicio 5 - Facturación de llamadas

En el material adicional encontrará una aplicación que registra y factura llamadas telefónicas. Para lograr tal objetivo, la aplicación permite administrar números telefónicos, como así también clientes asociados a un número. Los clientes pueden ser personas físicas o jurídicas. Además, el sistema permite registrar las llamadas realizadas, las cuales pueden ser nacionales o internacionales. Luego, a partir de las llamadas, la aplicación realiza la facturación, es decir, calcula el monto que debe abonar cada cliente.

Importe el material adicional provisto por la cátedra y analícelo para identificar y corregir los malos olores que presenta. En forma iterativa, realice los siguientes pasos:

(i) indique el mal olor,

(ii) indique el refactoring que lo corrige,

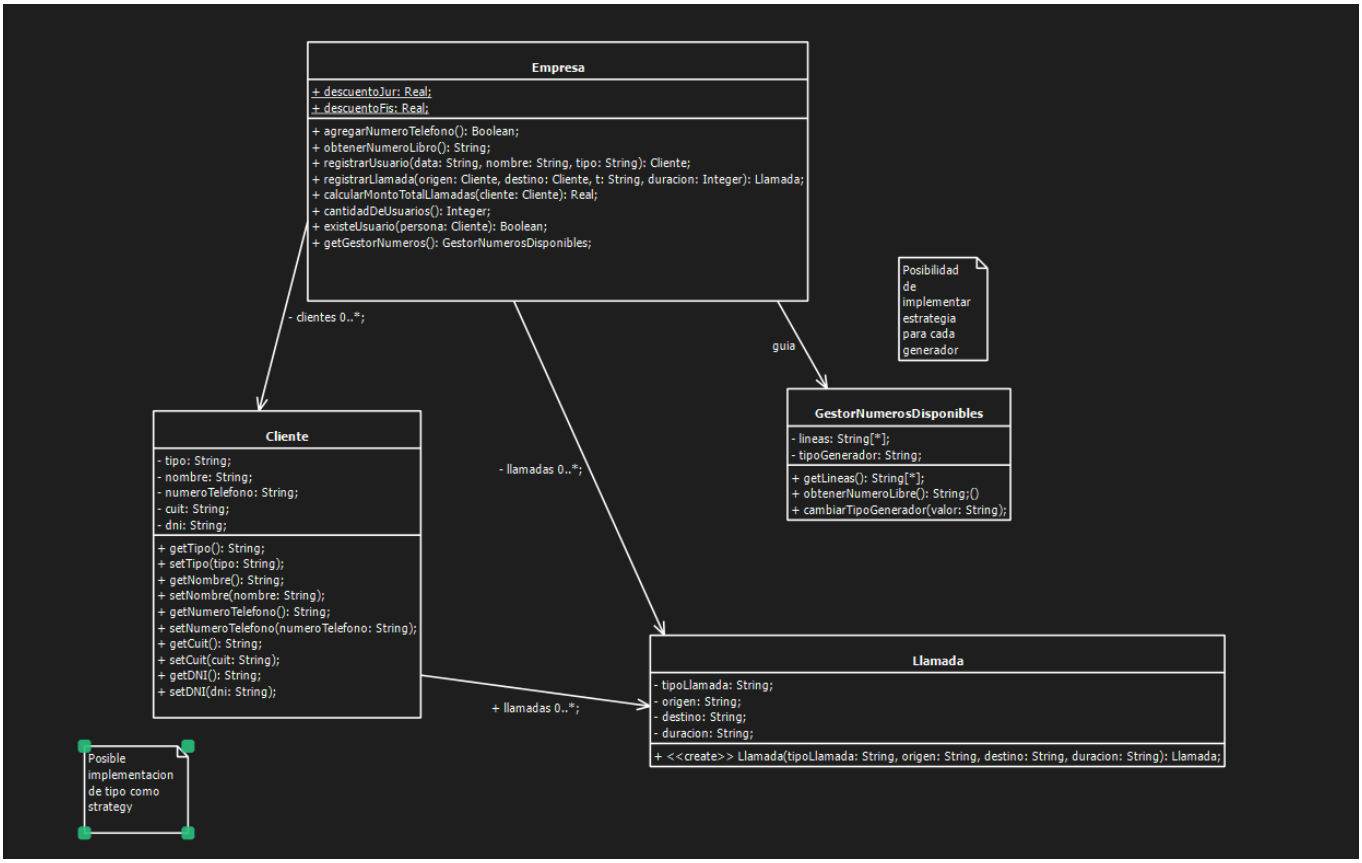
(iii) aplique el refactoring (modifique el código).

(iv) asegúrese de que los tests provistos corran exitosamente.

Si vuelve a encontrar un mal olor, retorne al paso (i).

Tareas:

- Describa la solución inicial con un diagrama de clases UML.
- Documente la secuencia de refactorings aplicados, como se indica previamente.
- Describa la solución final con un diagrama de clases UML.



Clase GestorNumerosDisponibles:

Método obtenerNúmeroLibre:

Malos olores:

- Identifiqué un **bad smell del tipo “método largo”** en `obtenerNumeroLibre()`, ya que contenía varias ramas de lógica condicional con acceso repetido a los datos internos del objeto (`lineas`). Para simplificar su responsabilidad y mejorar la legibilidad y mantenibilidad del código, apliqué el refactoring **“Move Method”**, extrayendo esa lógica a una clase separada llamada `Generador`.

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

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

    public String obtenerNumeroLibre() {
      return this.tipoGenerador.obtenerLibre(this);
    }
```

```
    public void cambiarTipoGenerador(String valor) {
       this.tipoGenerador.setValor(valor);
    }


}
public class Generador {
private String tipoGenerador = "ultimo";
public String obtenerLibre(GestorNumerosDisponibles gestor){
     String linea;
     SortedSet<String> lineas = gestor.getLineas();
     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;
}
}
```

- Identifique los bad smell Switch statements y codigo duplicado en cada una de los tipo de generador y lo soluciono aplicando “Replace Conditional with Polymorphism" y "Move method" convirtiendo el generador en una interfaz, y 3 clases que sean por cada metodo, y cambiaria el metodo cambiarTipo, y que reciba uno de la interfaz Generador para setearlo y dejando la logica del remove en el Gestor

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

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

```
    public String obtenerNumeroLibre() {
      String linea = tipoGenerador.obtenerLibre(this.lineas);
      lineas.remove(linea);
      return linea;
    }

    public void cambiarTipoGenerador(Generador valor) {
       this.tipoGenerador = valor;
    }
}
public interface Generador {
    public String obtenerLibre(GestorNumerosDisponibles gestor);
}
public UltimoGenerador implements Generador {
    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return gestor.getLineas().last();
    }
}
public randomGenerador implements Generador {

    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return new ArrayList<String>(gestor.getLineas())
                        .get(new Random().nextInt(gestor.getLineas().size()));
    }
public PrimeroGenerador implements Generador {

    public String obtenerLibre(GestorNumerosDisponibles gestor){
        return gestor.getLineas().first();
    }
}
```

Clase Empresa:

Método agregarNumeroTelefono:

- Envidia de atributos a la clase GestorNumerosDisponibles ya que en 3 lugares de este método, a principio, a principio del if tambien, esta haciendo algo que no le corresponde a la clase empresa. Para solucionarlo aplico "Move Method" moviendo ambos métodos a la clase GestorNumerosDisponibles

```
public class Empresa{
public boolean agregarNumeroTelefono(String str) {
    boolean encontre = this.guia.contiene(str);
    if (!encontre) {
```

```
        guia.add(str);
        encontre= true;
        return encontre;
    }
    else {
        encontre= false;
        return encontre;
    }
}
}
public class GestorNumerosDisponibles {

    public boolean contiene(String str){
        return this.lineas.contains(str);  }
    public boolean add(String number){
         return this.lineas.add(number);
    }
}
```

- El siguiente bad Smell identificado en el mismo método de Empresa es Temporary Field, el cual es innecesario, ya que hace el código menos legible. Aplico Refactoring Inline Temp 2 veces, una para la condición del if, y otra para dentro del if else. Y ya que tiene 2 returns, quito el segundo return del else

```
public class Empresa{

    public boolean agregarNumeroTelefono(String str) {

        if (!this.guia.contiene(str)) {
           guia.add(str);
           return true;
        }
        return false;

    }
}
```

Método registrarLlamada:

- Envidia de atributos hacia la clase Cliente ya que en lugar de decirle al cliente que lo añada, lo esta haciendo directamente. Aplico "Move Method"

```
Antes:
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;
}
Despues:
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.add(llamada);
        return llamada;
    }
}
public class Cliente {
    public void add(Llamada llamada){
        this.llamadas.add(llamada);
    }
}
```

Método registrar usuario:

- Identifico los siguientes bad smells
  - **Duplicated Code** (Código Duplicado)
    Las instrucciones comunes para ambos tipos (`setNombre` , `setTipo` , `setNumeroTelefono`) se repiten.
  - **Long Method** (Método Largo)
    No es tan largo, pero al haber `if/else` con bloques casi idénticos, afecta la legibilidad y la cohesión.
    Aplicaría extract method para extraer parte de la lógica que esta repetida fuera del método y que el método solo verifique la parte que únicamente varía en ambos.

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

Método calcularMontoTotalLlamadas:
El primer bad Smell identificado es Envidia de atributos ya que es algo que le corresponde a la al cliente calcularlo. Aplico move method hacia la clase cliente.

```
Antes:
public class Empresa{
public double calcularMontoTotalLlamadas(Cliente cliente) {
   double c = 0;
   for (Llamada l : cliente.llamadas) {
      double auxc = 0;
```

```
      if (l.getTipoDeLlamada() == "nacional") {
         // el precio es de 3 pesos por segundo más 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;
}
}
Despues:
public class Empresa{
     public double calcularMontoTotalLlamadas(Cliente cliente) {
        return cliente.calcularMontoTotalLlamadas(descuentoFis,descuentoJur);
    }
}
public class Cliente{
    public double calcularMontoTotalLlamadas(double descuentoFis, double
descuentoJur) {
        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*descuentoFis;
           } else if(this.getTipo() == "juridica") {
              auxc -= auxc*descuentoJur;
```

```
            }
            c += auxc;
        }
        return c;
    }
}
```

Clase Cliente:

- El bad smell identificado es Envidia de atributos ya que es algo que le corresponde a la llamada. El refactoring ideal sería move method ya que es una responsabilida de la llamada

```
Antes:
public class Cliente{
    public double calcularMontoTotalLlamadas(double descuentoFis, double
descuentoJur) {
        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*descuentoFis;
            } else if(this.getTipo() == "juridica") {
                auxc -= auxc*descuentoJur;
            }
            c += auxc;
        }
        return c;
    }
}
Despues:
public class Cliente{

public double calcularMontoTotalLlamadas(double descuentoFis, double descuentoJur)
{
    double c = 0;
```

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

    public double calcularPrecio(){
        if (tipoDeLlamada == "nacional") {
          // el precio es de 3 pesos por segundo más IVA sin adicional por
establecer la llamada
          return duracion * 3 + (duracion * 3 * 0.21);
       } else if (tipoDeLlamada == "internacional") {
          // el precio es de 150 pesos por segundo más IVA más 50 pesos por
establecer la llamada
          return duracion * 150 + (duracion * 150 * 0.21) + 50;
       }
       return 0;
    }

}
```

- Como siguiente medida ya que la empresa ya no necesita los descuentos porque no es su responsabilidad calcularlo aplico move field para mover los descuentos a la clase cliente y a su vez Change Method Signature porque ya no necesita recibir los descuentos

```
Antes:
public class Cliente{

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

```
        }
        c += auxc;
    }
    return c;
}
}

Despues:
public class Cliente{
static double descuentoJur = 0.15;
static double descuentoFis = 0;
public double calcularMontoTotalLlamadas() {
    double c = 0;
    for (Llamada l : this.llamadas) {
        double auxc = l.calcularPrecio();
        if (this.getTipo() == "fisica") {
            auxc -= auxc*descuentoFis;
        } else if(this.getTipo() == "juridica") {
            auxc -= auxc*descuentoJur;
        }
        c += auxc;
    }
    return c;
}
}
```

- El siguiente bad smell identificado es la obsesión por los primitivos con el tipo y podría convertirse en un switch statement en caso de tener mas tipos de empleado. Para solucionarlo aplico Replace Conditional with Polymorphism y a su vez creo un constructor que reciba los datos comunes a ambos en la superClase cliente. Y como consecuencia debo modificar en la clase empresa a la hora de crear el cliente junto con la eliminación de los descuentos como variables de instancia aplicando Replace Temp with Query ya que va a ser parte del método descuento

```
Antes:
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;
    static double descuentoJur = 0.15;
```

```
    static double descuentoFis = 0;
    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 void add(Llamada llamada){
       this.llamadas.add(llamada);
    }

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

```
        return c;
    }
}
Despues:
public class Empresa {

public Cliente registrarUsuario(String data, String nombre, String tipo) {
    Cliente var;
    if (tipo.equals("fisica")) {
        var = new ClienteFisico(nombre,this.obtenerNumeroLibre(), data);
    }
    else {
        var = new ClienteJuridico(nombre,this.obtenerNumeroLibre(), data);
    }
    clientes.add(var);
    return var;
}
}
public abstract class Cliente{
    public List<Llamada> llamadas = new ArrayList<Llamada>();
    private String nombre;
    private String numeroTelefono;
    public Cliente(String nombre, String numero){
        this.nombre = nombre;
        this.numeroTelefono = numero;
    }
    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 void add(Llamada llamada){
        this.llamadas.add(llamada);
    }

    public double calcularMontoTotalLlamadas(){
        double c = 0;
        for (Llamada l : this.llamadas) {
            double auxc = l.calcularPrecio();
```

```
            auxc -= auxc*this.descuento();
            c += auxc;
        }
        return c;
    }
    public abstract double descuento();
}
public class ClienteJuridico extends Cliente{
    private String cuit;

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

    public double descuento(){
        return 0.15;
    }

}
public class ClienteFisico extends Cliente{
    private String dni;

    public ClienteFisico(String nombre, String numero, String dni){
        super(nombre, numero);
        this.dni = dni;
    }

    public double descuento(){
        return 0;
    }

}
```

- El siguiente bad smell es reinventando la rueda ya que utiliza un método for y el bad smell tempory variable. Aplico Replace loop with pipeline e inline temp

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

```
    }
    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 void add(Llamada llamada){
        this.llamadas.add(llamada);
    }

    public double calcularMontoTotalLlamadas(){
        double c = 0;
        for (Llamada l : this.llamadas) {
           double auxc = l.calcularPrecio();
            auxc -= auxc*this.descuento();
            c += auxc;
        }
        return c;
    }
    public abstract double descuento();
}
Despues:
public abstract class Cliente{
    public List<Llamada> llamadas = new ArrayList<Llamada>();
    private String nombre;
    private String numeroTelefono;
    public Cliente(String nombre, String numero){
        this.nombre = nombre;
        this.numeroTelefono = numero;
    }
    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 void add(Llamada llamada){
        this.llamadas.add(llamada);
    }

    public double calcularMontoTotalLlamadas(){
    return this.llamadas.stream().mapToDouble(l -> (l.calcularPrecio() -
(l.calcularPrecio() * this.descuento()))).sum();
}
    public abstract double descuento();
}
```

- El siguiente identificado es que se rompe el encapsulamiento con la variable llamada publica. Utilizo encapsulate field

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

Clase Llamada:

Metodo calcularPrecio():

- Se identifica el bad smell obsesion con los primitivos y switch statement. Para ello se aplica Replace conditional with polimorpishm y como consecuencia hay que cambiar la instanciacion en la clase empresa de una llamada

```
Antes:
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 getRemitente() {
       return destino;
    }

    public int getDuracion() {
       return this.duracion;
    }

    public String getOrigen() {
       return origen;
    }

       public double calcularPrecio(){
          if (tipoDeLlamada == "nacional") {
             // el precio es de 3 pesos por segundo más IVA sin adicional por
establecer la llamada
             return duracion * 3 + (duracion * 3 * 0.21);
          } else if (tipoDeLlamada == "internacional") {
             // el precio es de 150 pesos por segundo más IVA más 50 pesos por
establecer la llamada
             return duracion * 150 + (duracion * 150 * 0.21) + 50;
          }
          return 0;
       }



}
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.add(llamada);
        return llamada;
    }
}
Despues:
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 getRemitente() {
       return destino;
    }

    public int getDuracion() {
       return this.duracion;
    }

    public String getOrigen() {
       return origen;
    }

    public abstract double calcularPrecio();
}
public class LlamadaNacional extends Llamada{

public LlamadaNacional(String origen, String destino, int duracion) {
       super(origen,destino, duracion);
}
public double calcularPrecio(){
    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 calcularPrecio(){
    return this.getDuracion() * 150 + (this.getDuracion() * 150 * 0.21) + 50;
}
}
public class Empresa{

    public Llamada registrarLlamada(Cliente origen, Cliente destino, String t, int
duracion) {
        Llamada llamada;
```

```
        if(t.equals("nacional")){
             llamada = new LlamadaNacional(origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
        }
        else{
             llamada = new LlamadaInternacional(origen.getNumeroTelefono(),
destino.getNumeroTelefono(), duracion);
        }
        llamadas.add(llamada);
        origen.add(llamada);
        return llamada;
    }
}
```