Alumno: Guillermo Helfer

Refactorizaciones sobre la clase Empresa:

```java
package ar.edu.unlp.info.oo2.facturacion_llamadas;

import java.util.ArrayList;

public class Empresa {
    private List<Cliente> clientes = new ArrayList<Cliente>();
//  private List<Llamada> llamadas = new ArrayList<Llamada>();      //olor clase larga, no hace falta que la empresa lleve registro de las llamadas, delegeado al cliente
    GestorNumerosDisponibles guia = new GestorNumerosDisponibles();
```

```java
//  static double descuentoJur = 0.15; //olor feature envy esto deberia ser propio del correspondiente cliente, move variable
//  static double descuentoFis = 0;
```

```java
public class ClienteJuridico extends Cliente {

    private String cuit;

    private static double descuento = 0.15;         //agregado por el move variable desde empresa
```

```java
public class ClienteFisico extends Cliente {

    private String dni;

    private static double descuento = 0;        //agregado por el move variable desde empresa
```

```java
//  public boolean agregarNumeroTelefono(String str) {              //olor metodo largo?, decompose conditional
//      boolean encontre = guia.getLineas().contains(str);          //olor nombre parametro, rename variable
//      if (!encontre) {
//          guia.getLineas().add(str);
//          encontre= true;
//          return encontre;
//      }
//      else {
//          encontre= false;
//          return encontre;
//      }
//  }

    public boolean agregarNumeroTelefono(String numeroAgregar) {    //solucion refactorizada
        if (!this.guia.isRegistrado(numeroAgregar)) {
            guia.getLineas().add(numeroAgregar);
            return true;
        }
        return false;
    }
```

```
36 //  public boolean isRegistrado(String num) {              //olor esto no le corresponde a esta clase (feature envy), move method a guia.
37 //      return this.guia.getLineas().contains(num);
38 //  }
```

(Clase GestorNumerosDisponibles:)

```
54
55      public boolean isRegistrado(String num) {  //refactorizado (move method empresa linea 36)
56          return getLineas().contains(num);
57      }
58  }
59
```

- - -

```
44 //  public Cliente registrarUsuario(String data, String nombre, String tipo) {
45 //      Cliente var = new Cliente();
46 //      if (tipo.equals("fisica")) {               //olor esto deberia ser polimorfico, replace conditional with polymorphism
47 //          var.setNombre(nombre);
48 //          String tel = this.obtenerNumeroLibre(); //olor nombre de variable cliente, rename variable
49 //          var.setTipo(tipo);
50 //          var.setNumeroTelefono(tel);
51 //          var.setDNI(data);
52 //      }
53 //      else if (tipo.equals("juridica")) {
54 //          String tel = this.obtenerNumeroLibre();
55 //          var.setNombre(nombre);
56 //          var.setTipo(tipo);
57 //          var.setNumeroTelefono(tel);
58 //          var.setCuit(data);
59 //      }
60 //      clientes.add(var);
61 //      return var;
62 //  }
63
64      public ClienteJuridico registrarUsuarioJuridico(String data, String nombre) {
65          String tel = this.obtenerNumeroLibre();
66          ClienteJuridico cli = new ClienteJuridico(nombre,tel,data);
67          clientes.add(cli);
68          return cli;
69      }
70
71      public ClienteFisico registrarUsuarioFisico(String data, String nombre) {
72          String tel = this.obtenerNumeroLibre();
73          ClienteFisico cli = new ClienteFisico(nombre,tel,data);
74          clientes.add(cli);
75          return cli;
76      }
77
```

- - -

```
78  public Llamada registrarLlamada(Cliente origen, Cliente destino, String t, int duracion) {
79      Llamada llamada = new Llamada(t, origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);    //este metodo se deberia resolver polimorficamente debido al replace conditional with polym.
80      llamadas.add(llamada);                                                                                   //realizado en el metodo calcularMontoLlamada de esta clase.
81      origen.llamadas.add(llamada);                                                                            //ademas ya no se necesita agregar la llamada debido al move method
82      return llamada;
83  }
84
85  public LlamadaNacional registrarLlamadaNacional(Cliente origen, Cliente destino, int duracion) {            //ya no se necesita la variable del tipo por la solucion polimorfica
86      LlamadaNacional llamada = new LlamadaNacional(origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);
87      origen.llamadas.add(llamada);
88      return llamada;
89  }
90
91  public LlamadaInternacional registrarLlamadaInternacional(Cliente origen, Cliente destino, int duracion) {
92      LlamadaInternacional llamada = new LlamadaInternacional(origen.getNumeroTelefono(), destino.getNumeroTelefono(), duracion);
93      origen.llamadas.add(llamada);
94      return llamada;
95  }
96
```

- - -

```java
public double calcularMontoTotalLlamadas(Cliente cliente) {  //olor esto lo deberia calcular cada tipo polimorficamente
    double c = 0;                                            //olor nombre de variable confuso, rename varibale c -> count
    for (Llamada l : cliente.llamadas) {
        double auxc = 0;
        if (l.getTipoDeLlamada() == "nacional") {           //olor metodo largo, reemplazar condicional por polimorfismo
            // 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") {   //olor metodo largo reemplazar condicional por polimorfismo
            auxc -= auxc*descuentoFis;
        } else if(cliente.getTipo() == "juridica") {
            auxc -= auxc*descuentoJur;
        }
        c += auxc;
    }
    return c;
}
                                                             //el metodo de abajo es la refactorizacion del metodo de arriba
public double calcularMontoTotalLlamadas(Cliente cliente) { //olor le envidia los atributos al cliente, move method hacia cliente
    double count = 0;
    for (Llamada l : cliente.llamadas) {
        double auxCount = 0;
        auxCount = l.calcularMontoLlamada();
        auxCount -= auxCount * cliente.getDescuento();
        count += auxCount;
    }
    return count;
}
```

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

(En la clase cliente:)

```java
    public double calcularMontoTotalLlamadas() { //resultado del move method desde la clase empresa, no se necesita el parametro Empresa
        double count = 0;
        for (Llamada l : llamadas) {
            double auxCount = 0;
            auxCount = l.calcularMontoLlamada();
            auxCount -= auxCount * getDescuento();
            count += auxCount;
//  private String tipo;        //la variable de tipo ya no se utiliza por el cambio a polimorfismo
//  private String dni          //move variable en consecuencia del replace conditional with polymorphism en empresa (linea 44)
//  private String cuit;        //sobre la registracion del usuario

//  public String getTipo() {   //ya no se necesitan estos metodos en consecuencia de la conversion polimorfica
//      return tipo;
//  }
//  public void setTipo(String tipo) {
//      this.tipo = tipo;
//  }
```

(En la clase Llamada:)

```java
//  private String tipo;                                    //no se utiliza por el cambio a polimorfismo
```

```java
  public abstract double calcularMontoLlamada();

  protected double calcularMontoLlamadaAux() {                //template method
      return this.getDuracion() * this.getCostoSegundo()
              + calcularPrecioConIva();                       //olor metodo largo, extract method
  }

/ private double calcularPrecioConIva () {                          //consecuencia del anterior extract method
/     return this.getDuracion() * this.getCostoSegundo() * 0.21;    //olor numero magico, reemplazar por constante
/ }
  private double calcularPrecioConIva () {
      return this.getDuracion() * this.getCostoSegundo() * IVA;
  }
```

```
private double costoSegundo;
private static final double IVA = 0.21;
private String tipo;
```

- - -

```
public GestorNumerosDisponibles getGestorNumeros() {    //olor no me parece muy fiable que se pueda obtener la lista de los
    return this.guia;                                    //numeros disponibles de manera publica pero debe ser para el test
}
```

Al final decidí no modificarlo pero considero que es un error de mi parte

- - -

Refactorizaciones sobre la clase GestorNumerosDisponibles

```
//  public String obtenerNumeroLibre() {    //olor esto deberia resolverse polimorficamente (SWITCH ST/conditional logic)
//      String linea;                       //replace conditional logic with polymorphism (strategy)
//      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 String obtenerNumeroLibre() {
    String linea = this.tipoGenerador.obtenerNumeroLibre(lineas);
    this.getLineas().remove(linea);
    return linea;
}
```

Dependiendo de la estrategia resuelve distinto.

```
private SortedSet<String> lineas = new TreeSet<String>();
//  private String tipoGenerador = "ultimo"; //olor (ver linea 15); borrado, ya no se necesita
```

```
//  public void cambiarTipoGenerador(String valor) { //deberia resolverse con un strategy en general
//      this.tipoGenerador = valor;
//  }

    public void cambiarTipoGeneradorUltimo() {              //un cambio para cada tipo de estrategia
        this.tipoGenerador = new UltimoStrategy();
    }
    public void cambiarTipoGeneradorPrimero() {
        this.tipoGenerador = new PrimeroStrategy();
    }
    public void cambiarTipoGeneradorRandom() {
        this.tipoGenerador = new RandomStrategy();
    }
```

```java
public interface Strategy {

    public abstract String obtenerNumeroLibre(SortedSet<String> lineas);

}
```

… luego en su respectiva sección entra en detalle cada estrategia (para no pegar varias veces el mismo código)

- - -

Refactorizaciones sobre la clase Cliente:

```java
import java.util.ArrayList;

public abstract class Cliente {
    public List<Llamada> llamadas = new ArrayList<Llamada>();   //olor el cliente no usa su lista de llamadas (data class)
                                                                 //deberia haber envidia de atributo en otra clase
    private String nombre;
    private String numeroTelefono;
```

Efectivamente sucedía, luego se corrigió.

- - -

```java
//  public void setNombre(String nombre) {   //olor innapropiate intimacy, cambio de public a protected
//      this.nombre = nombre;
//  }
    protected void setNombre(String nombre) {    //olor innapropiate intimacy, cambio de public a protected
        this.nombre = nombre;
    }
    public String getNumeroTelefono() {
        return numeroTelefono;
    }
//  public void setNumeroTelefono(String numeroTelefono) {  //olor innapropiate intimacy, cambio de public a protected
//      this.numeroTelefono = numeroTelefono;
//  }
```

- - -

```java
    }
//  public String getCuit() {               //move method o push down del get y set hacia las clases ClienteFisico y ClienteJuridico
//      return cuit;                        //respectivamente en consecuencia del replace conditional with polymorphism en empresa
//  }                                       //linea 44 sobre el metodo de registrarUsuaruio.
//
//  public void setCuit(String cuit) {
//      this.cuit = cuit;
//  }
//  public String getDNI() {
//      return dni;
//  }
//  public void setDNI(String dni) {
//      this.dni = dni;
//  }
```

(En la Clase ClienteFisico:)

```java
public class ClienteFisico extends Cliente {

    private String dni;

    private static double descuento = 0;        //agregado por el move variable desde empresa

    public ClienteFisico(String nombre, String tel, String data) {
        this.setNombre(nombre);
        this.setNumeroTelefono(tel);
        this.setDNI(data);
        this.setDescuento(descuento);
    }

    public String getDNI() {
        return dni;
    }

    private void setDNI(String dni) {
        this.dni = dni;
    }

}
```

(En la Clase ClienteJuridico:)

```java
public class ClienteJuridico extends Cliente {

    private String cuit;

    private static double descuento = 0.15;         //agregado por el move variable desde empresa


    public ClienteJuridico(String nombre, String tel, String data) {
        this.setNombre(nombre);
        this.setNumeroTelefono(tel);
        this.setCuit(data);
        this.setDescuento(descuento);
    }

    public String getCuit() {
        return cuit;
    }

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


}
```

- - -

Refactorizaciones en la clase Llamada:

```java
public abstract class Llamada {                         //olor data class, puede haber envidia de atributo en otra clase (resuelto)
    private String origen;
    private String destino;
```

Luego resuelto por los pasos previos…

- - -

La mayoria de los cambios fueron consecuencia de las previas refactorizaciones…

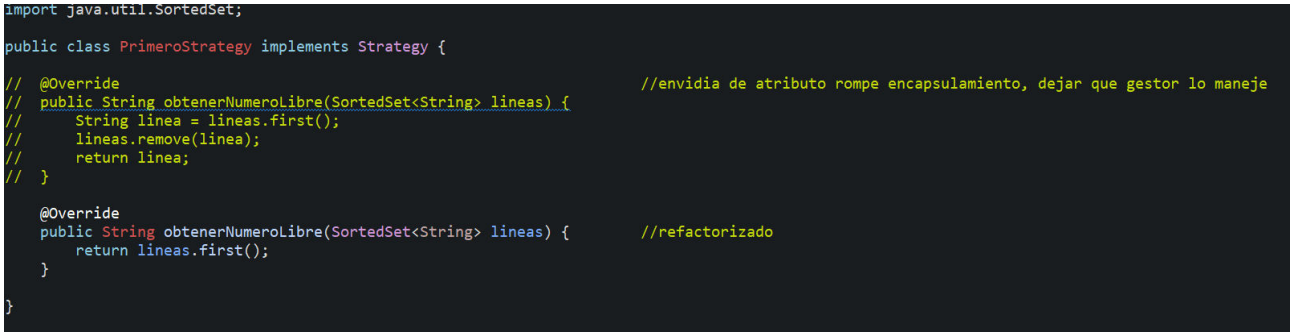
- - -

Refactorizaciones en la clase UltimoStrategy:

```
import java.util.SortedSet;

public class UltimoStrategy implements Strategy {

//  @Override
//  public String obtenerNumeroLibre(SortedSet<String> lineas) {        //olor envidia de atributo, rompe encapsulamiento. dejo que la clase
//      String linea = lineas.last();                                   //gestorNumerosDisponibles maneje la eliminacion de la linea usada.
//      lineas.remove(linea);
//      return linea;
//  }

    @Override
    public String obtenerNumeroLibre(SortedSet<String> lineas) {        //refactorizado
        return lineas.last();
    }

}
```

- - -

Refactorizaciones en la clase PrimeroStrategy:



```
import java.util.SortedSet;

public class PrimeroStrategy implements Strategy {

//  @Override                                                           //envidia de atributo rompe encapsulamiento, dejar que gestor lo maneje
//  public String obtenerNumeroLibre(SortedSet<String> lineas) {
//      String linea = lineas.first();
//      lineas.remove(linea);
//      return linea;
//  }

    @Override
    public String obtenerNumeroLibre(SortedSet<String> lineas) {        //refactorizado
        return lineas.first();
    }

}
```

- - -

Refactorizaciones en la clase RandomStrategy:

```java
public class RandomStrategy implements Strategy{

//  @Override
//  public String obtenerNumeroLibre(SortedSet<String> lineas) {        //olor envidia atributo, rompe encapsulamiento, dejo que gestor maneje
//      String linea = new ArrayList<String>(lineas)                    //la eliminacion de la linea usada
//              .get(new Random().nextInt(lineas.size()));
//      lineas.remove(linea);
//      return linea;
//  }

    @Override
    public String obtenerNumeroLibre(SortedSet<String> lineas) {        //refactorizado
        return new ArrayList<String>(lineas)
                .get(new Random().nextInt(lineas.size()));
    }

}
```