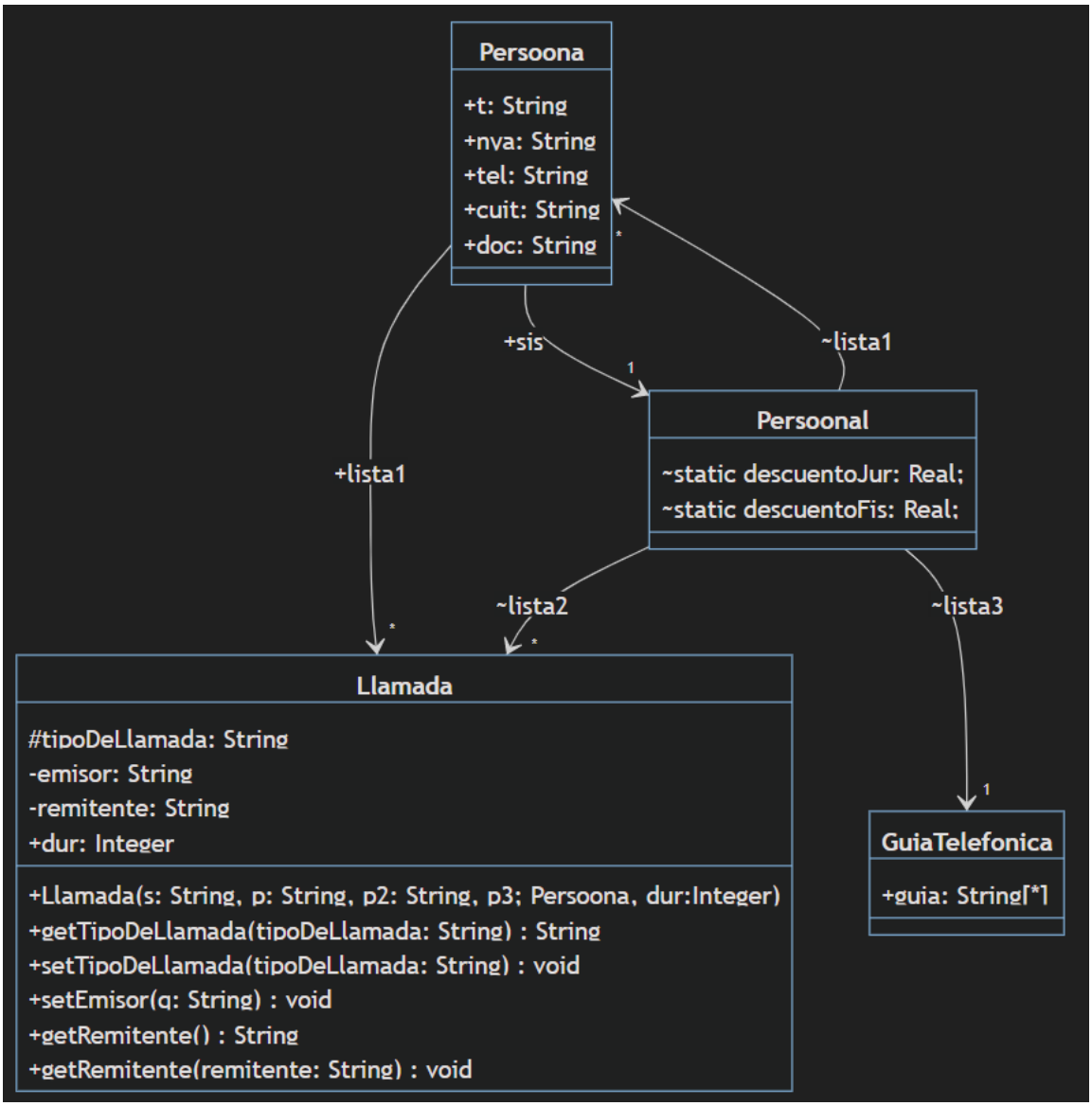
Ejercicio 3 Refactoring: Nicolás Bonoris

UML Inicial



Refactorings Aplicados

▼ 1° Paso

Bad Smell

Nombre: Mysterious Name

Extracto:

```
public class Persoona {
  public List<Llamada> lista1 = new ArrayList<Llamada>();
  public String t;
  public String nya;
  public String tel;
  public String cuit;
```

```
  public String doc;
  public Persoonal sis;
  public List<Llamada> getLista1() {
    return lista1;
  }
  public void setLista1(List<Llamada> lista1) {
    this.lista1 = lista1;
  }
  public String getT() {
    return t;
  }
  public void setT(String t) {
    this.t = t;
  }
  public String getNya() {
    return nya;
  }
  public void setNya(String nya) {
    this.nya = nya;
  }
  public String getTel() {
    return tel;
  }
  public void setTel(String tel) {
    this.tel = tel;
  }
  public String getCuit() {
    return cuit;
  }
  public void setCuit(String cuit) {
    this.cuit = cuit;
  }
  public String getDoc() {
    return doc;
  }
  public void setDoc(String doc) {
    this.doc = doc;
  }
}
```

```
public class Llamada {
  protected String tipoDeLlamada;
  private String emisor;
  private String remitente;
  public int dur;
  public Llamada() {
  }
  public Llamada(String s, String p, String p2, Persoona p3, int dur) {
    this.tipoDeLlamada = s;
    this.emisor= p;
    this.remitente= p2;
    this.dur = dur;
  }
  public String getTipoDeLlamada() {
    return tipoDeLlamada;
  }
  public void setTipoDeLlamada(String tipoDeLlamada) {
    this.tipoDeLlamada = tipoDeLlamada;
  }
  public void setEmisor(String q) {
    emisor = q;
  }
  public String getRemitente() {
    return remitente;
  }
  public void setRemitente(String remitente) {
    this.remitente = remitente;
  }
}
```

```
public class GuiaTelefonica {
  public SortedSet<String> guia = new TreeSet<String>();
}
```

```
public class Persoonal {
  List<Persoona> lista1 = new ArrayList<Persoona>();
  List<Llamada> lista2 = new ArrayList<Llamada>();
  GuiaTelefonica lista3 = new GuiaTelefonica();
  static double descuentoJur = 0.15;
```

```
static double descuentoFis = 0;
public boolean agregarTelefono(String str) {
  boolean encontre = lista3.guia.contains(str);
  if (!encontre) {
    lista3.guia.add(str);
    encontre= true;
    return encontre;
  }
  else {
    encontre= false;
    return encontre;
  }
}
public Persoona registrarUsuario(String data, String nombre, String t) {
  Persoona var = new Persoona();
  if (t.equals("fisica")) {
    var.setNya(nombre);
    String tel = lista3.guia.last();
    lista3.guia.remove(tel);
    var.setT(t);
    var.setTel(tel);
    var.setDoc(data);
  }
  else if (t.equals("juridica")) {
    String tel = lista3.guia.last();
    lista3.guia.remove(tel);
    var.nya =nombre;
    var.t =t;
    var.tel = tel;
    var.cuit =data;
  }
  var.sis = this;
  lista1.add(var);
  return var;
}
public boolean eliminarUsuario(Persoona p) {
  List<Persoona> l = (p.sis.lista1.stream()
    .filter(persona -> persona != p).collect(Collectors.toList()));
  boolean borre = false;
  if (l.size() < lista1.size()) {
    this.lista1 = l;
    this.lista3.guia.add(p.getTel());
    borre = true;
  }
  return borre;
}
public Llamada registrarLlamada(Persoona q, Persoona q2, String t, int d) {
  Llamada x = new Llamada();
  x.tipoDeLlamada = t;
  x.setEmisor(q.tel);
  x.setRemitente(q2.getTel());
  x.dur= d;
  lista2.add(x);
  q.lista1.add(x);
  return x;
}
public double calcularMontoTotalLlamadas(Persoona p) {
  double c = 0;
  Persoona aux = null;
  for (Persoona pp : lista1) {
    if (pp.tel == p.getTel()) {
      aux = pp;
      break;
    }
  } if (aux == null) return c;
  if (aux != null) {
    for (Llamada l : aux.lista1) {
      double auxc = 0;
      if (l.tipoDeLlamada == "nacional") {
        auxc += l.dur *3 + (l.dur*3*0.21);
      } else if (l.tipoDeLlamada == "internacional") {
        auxc += l.dur *200 + (l.dur*200*0.21);
      }
      if (aux.t == "fisica") {
        auxc -= auxc*descuentoFis;
      } else if(aux.t == "juridica") {
        auxc -= auxc*descuentoJur;
      }
      c += auxc;
    }
  }
  return c;
}
public int cantidadDeUsuarios() {
  return lista1.size();
}
```

```
  public boolean existeUsuario(Persoona persona) {
    return lista1.contains(persona);
  }
}
```

Refactoring

Nombre: Rename field, rename method y rename variable.

- Los llevamos a cabo en las cuatro clases para dar nombres descriptivos a las variables y métodos.
- También aplicamos change method signature en el constructor *Llamada()*, eliminamos el parámetro *p3:Persoona* no utilizado.

Código Modificado

```
public class Persoonal {
  List<Persoona> personas = new ArrayList<Persoona>();
  List<Llamada> llamadas = new ArrayList<Llamada>();
  GuiaTelefonica guia = new GuiaTelefonica();
  static double descuentoJur = 0.15;
  static double descuentoFis = 0;

  public boolean agregarTelefono(String telefono) {
    boolean encontre = guia.telefonos.contains(telefono);
    if (!encontre) {
      guia.telefonos.add(telefono);
      encontre= true;
      return encontre;
    }
    else {
      encontre= false;
      return encontre;
    }
  }

  public Persoona registrarUsuario(String identificacion, String nombreApellido,
    String tipo) {
    Persoona persona = new Persoona();
    if (tipo.equals("fisica")) {
      persona.setNombreApellido(nombreApellido);
      String telefono = guia.telefonos.last();
      guia.telefonos.remove(telefono);
      persona.setTipo(tipo);
      persona.setTelefono(telefono);
      persona.setDocumento(identificacion);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.telefonos.last();
      guia.telefonos.remove(telefono);
      persona.nombreApellido =nombreApellido;
      persona.tipo =tipo;
      persona.telefono = telefono;
      persona.cuit =identificacion;
    }
    persona.sistema = this;
    personas.add(persona);
    return persona;

  }

  public boolean eliminarUsuario(Persoona persona) {
    List<Persoona> personasNueva = (persona.sistema.personas.stream()
      .filter(p -> p != persona).collect(Collectors.toList()));
    boolean borre = false;
    if (personasNueva.size() < personas.size()) {
      this.personas = personasNueva;
      this.guia.telefonos.add(persona.getTelefono());
      borre = true;
    }
    return borre;
  }

  public Llamada registrarLlamada(Persoona emisor, Persoona remitente,
    String tipoLlamada, int duracion) {
    Llamada llamada = new Llamada();
    llamada.tipoDeLlamada = tipoLlamada;
```

```
    llamada.setEmisor(emisor.telefono);
    llamada.setRemitente(remitente.getTelefono());
    llamada.duracion= duracion;
    llamadas.add(llamada);
    emisor.llamadas.add(llamada);
    return llamada;
  }

  public double calcularMontoTotalLlamadas(Persoona persona) {
    double montoTotal = 0;
    Persoona personaAux = null;
    for (Persoona p : personas) {
      if (p.telefono == persona.getTelefono()) {
        personaAux = p;
        break;
      }
    } if (personaAux == null) return montoTotal;
    if (personaAux != null) {
      for (Llamada l : personaAux.llamadas) {
        double costoAux = 0;
        if (l.tipoDeLlamada == "nacional") {
          costoAux += l.duracion *3 + (l.duracion*3*0.21);
        } else if (l.tipoDeLlamada == "internacional") {
          costoAux += l.duracion *200 + (l.duracion*200*0.21);
        }

        if (personaAux.tipo == "fisica") {
          costoAux -= costoAux*descuentoFis;
        } else if(personaAux.tipo == "juridica") {
          costoAux -= costoAux*descuentoJur;
        }
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }

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

  public boolean existeUsuario(Persoona persona) {
    return personas.contains(persona);
  }

}
```

```
public class Persoona {
  public List<Llamada> llamadas = new ArrayList<Llamada>();
  public String tipo;
  public String nombreApellido;
  public String telefono;
  public String cuit;
  public String documento;
  public Persoonal sistema;

  public List<Llamada> getLlamadas() {
    return llamadas;
  }
  public void setLlamadas(List<Llamada> llamadas) {
    this.llamadas = llamadas;
  }
  public String getTipo() {
    return tipo;
  }
  public void setTipo(String tipo) {
    this.tipo = tipo;
  }
  public String getNombreApellido() {
    return nombreApellido;
  }
  public void setNombreApellido(String nombreApellido) {
    this.nombreApellido = nombreApellido;
  }
  public String getTelefono() {
    return telefono;
  }
  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }
  public String getCuit() {
    return cuit;
  }
  public void setCuit(String cuit) {
```

```
    this.cuit = cuit;
  }
  public String getDocumento() {
    return documento;
  }
  public void setDocumento(String documento) {
    this.documento = documento;
  }
}
```

```
public class Llamada {
  protected String tipoDeLlamada;
  private String emisor;
  private String remitente;
  public int duracion;

  public Llamada() {

  }
  public Llamada(String tipoDeLlamada, String emisor,
    String remitente, int duracion) {
    this.tipoDeLlamada = tipoDeLlamada;
    this.emisor = emisor;
    this.remitente = remitente;
    this.duracion = duracion;
  }

  public String getTipoDeLlamada() {
    return tipoDeLlamada;
  }

  public void setTipoDeLlamada(String tipoDeLlamada) {
    this.tipoDeLlamada = tipoDeLlamada;
  }
  public void setEmisor(String emisor) {
    this.emisor = emisor;
  }
  public String getRemitente() {
    return remitente;
  }
  public void setRemitente(String remitente) {
    this.remitente = remitente;
  }
}
```

```
public class GuiaTelefonica {
  public SortedSet<String> telefonos = new TreeSet<String>();
}
```

▼ **2° Paso**

Bad Smell

Nombre: Global Data

Extracto:

```
public class Persoona {
  public List<Llamada> llamadas = new ArrayList<Llamada>();
  public String tipo;
  public String nombreApellido;
  public String telefono;
  public String cuit;
  public String documento;
  public Persoonal sis;
  (...)
}

public class GuiaTelefonica {
  public SortedSet<String> telefonos = new TreeSet<String>();
  (...)
}

public class Llamada {
  public int duracion;
  (...)
}
```

Refactoring

Nombre: Encapsulate Field

- Vamos a encapsular las variables de instancia publicas de las tres clases que las poseen y crear los accesors necesarios para que puedan ser accedidas.

Código Modificado

```
public class Persoonal {
  (...)
  public boolean agregarTelefono(String telefono) {
    boolean encontre = guia.getTelefonos().contains(telefono);
    if (!encontre) {
      guia.getTelefonos().add(telefono);
      encontre= true;
      return encontre;
    }
    else {
      encontre= false;
      return encontre;
    }
  }
  public Persoona registrarUsuario(String identificacion, String nombreApellido, String tipo) {
    Persoona persona = new Persoona();
    if (tipo.equals("fisica")) {
      persona.setNombreApellido(nombreApellido);
      String telefono = guia.getTelefonos().last();
      guia.getTelefonos().remove(telefono);
      persona.setTipo(tipo);
      persona.setTelefono(telefono);
      persona.setDocumento(identificacion);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.getTelefonos().last();
      guia.getTelefonos().remove(telefono);
      persona.setNombreApellido(nombreApellido);
      persona.setTipo(tipo);
      persona.setTelefono(telefono);
      persona.setCuit(identificacion);
    }
    persona.setSistema(this);
    personas.add(persona);
    return persona;
  }
  public boolean eliminarUsuario(Persoona persona) {
    List<Persoona> personasNueva = (persona.getSistema()
      .personas.stream().filter(p -> p != persona).collect(Collectors.toList()));
    boolean borre = false;
    if (personasNueva.size() < personas.size()) {
      this.personas = personasNueva;
      this.guia.getTelefonos().add(persona.getTelefono());
      borre = true;
    }
    return borre;
  }
  public Llamada registrarLlamada(Persoona emisor, Persoona remitente, String tipoLlamada, int duracion) {
    Llamada llamada = new Llamada();
    llamada.setTipoDeLlamada(tipoLlamada);
    llamada.setEmisor(emisor.getTelefono());
    llamada.setRemitente(remitente.getTelefono());
    llamada.setDuracion(duracion);
    llamadas.add(llamada);
    emisor.getLlamadas().add(llamada);
    return llamada;
  }
  public double calcularMontoTotalLlamadas(Persoona persona) {
    double montoTotal = 0;
    Persoona personaAux = null;
    for (Persoona p : personas) {
      if (p.getTelefono() == persona.getTelefono()) {
        personaAux = p;
        break;
      }
    } if (personaAux == null) return montoTotal;
    if (personaAux != null) {
      for (Llamada l : personaAux.getLlamadas()) {
        double costoAux = 0;
```

```
        if (l.tipoDeLlamada == "nacional") {
          costoAux += l.getDuracion() *3 + (l.getDuracion()*3*0.21);
        } else if (l.tipoDeLlamada == "internacional") {
          costoAux += l.getDuracion() *200 + (l.getDuracion()*200*0.21);
        }

        if (personaAux.getTipo() == "fisica") {
          costoAux -= costoAux*descuentoFis;
        } else if(personaAux.getTipo() == "juridica") {
          costoAux -= costoAux*descuentoJur;
        }
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }
  public int cantidadDeUsuarios() {
    return personas.size();
  }
  public boolean existeUsuario(Persoona persona) {
    return personas.contains(persona);
  }
}
```

```
public class Persoona {
  private List<Llamada> llamadas = new ArrayList<Llamada>();
  private String tipo;
  private String nombreApellido;
  private String telefono;
  private String cuit;
  private String documento;
  private Persoonal sistema;
  public List<Llamada> getLlamadas() {
    return llamadas;
  }
  public void setLlamadas(List<Llamada> llamadas) {
    this.llamadas = llamadas;
  }
  public String getTipo() {
    return tipo;
  }
  public void setTipo(String tipo) {
    this.tipo = tipo;
  }
  public String getNombreApellido() {
    return nombreApellido;
  }
  public void setNombreApellido(String nombreApellido) {
    this.nombreApellido = nombreApellido;
  }
  public String getTelefono() {
    return telefono;
  }
  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }
  public String getCuit() {
    return cuit;
  }
  public void setCuit(String cuit) {
    this.cuit = cuit;
  }
  public String getDocumento() {
    return documento;
  }
  public void setDocumento(String documento) {
    this.documento = documento;
  }
  public Persoonal getSistema() {
    return sistema;
  }
  public void setSistema(Persoonal sistema) {
    this.sistema = sistema;
  }
}
```

```
// Con el uso de "(...)" indico que la clase sigue, pero se omiten lineas irrelevantes
public class Llamada {
  private int duracion;
  public int getDuracion() {
    return duracion;
  }
```

```
  public void setDuracion(int duracion) {
    this.duracion = duracion;
  }
  (...)
}
```

▼ **3° Paso**

Bad Smell

Nombre: Feature Envy, Data Class, God Class

Extracto:

```
public Persoona registrarUsuario(String identificacion, String nombreApellido,
  String tipo) {
  Persoona persona = new Persoona();
  if (tipo.equals("fisica")) {
    persona.setNombreApellido(nombreApellido);
    String telefono = guia.telefonos.last();
    guia.telefonos.remove(telefono);
    persona.setTipo(tipo);
    persona.setTelefono(telefono);
    persona.setDocumento(identificacion);
  }
  else if (tipo.equals("juridica")) {
    String telefono = guia.telefonos.last();
    guia.telefonos.remove(telefono);
    persona.nombreApellido =nombreApellido;
    persona.tipo =tipo;
    persona.telefono = telefono;
    persona.cuit =identificacion;
  }
  persona.sistema = this;
  personas.add(persona);
  return persona;

}

public boolean eliminarUsuario(Persoona persona) {
  List<Persoona> personasNueva = persona.getSistema().personas.stream().filter(p -> p != persona).collect(Collectors.toList());
  boolean borre = false;
  if (personasNueva.size() < personas.size()) {
    this.personas = personasNueva;
    this.guia.getTelefonos().add(persona.getTelefono());
    borre = true;
  }
  return borre;

}

public Llamada registrarLlamada(Persoona emisor, Persoona remitente, String tipoLlamada, int duracion) {
    Llamada llamada = new Llamada();
    llamada.setTipoDeLlamada(tipoLlamada);
    llamada.setEmisor(emisor.getTelefono());
    llamada.setRemitente(remitente.getTelefono());
    llamada.setDuracion(duracion);
    llamadas.add(llamada);
    emisor.getLlamadas().add(llamada);
    return llamada;
}

public double calcularMontoTotalLlamadas(Persoona persona) {
  double montoTotal = 0;
  Persoona personaAux = null;
  for (Persoona p : personas) {
    if (p.getTelefono() == persona.getTelefono()) {
      personaAux = p;
      break;
    }
  }
  if (personaAux == null) return montoTotal;
  if (personaAux != null) {
    for (Llamada l : personaAux.getLlamadas()) {
      double costoAux = 0;
      if (l.getTipoDeLlamada() == "nacional") {
        costoAux += l.getDuracion() *3 + (l.getDuracion()*3*0.21);
      } else if (l.getTipoDeLlamada() == "internacional") {
        costoAux += l.getDuracion() *200 + (l.getDuracion()*200*0.21);
      }
```

```
        if (personaAux.getTipo() == "fisica") {
          costoAux -= costoAux*descuentoFis;
        } else if(personaAux.getTipo() == "juridica") {
          costoAux -= costoAux*descuentoJur;
        }
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }
```

Refactoring

Nombre: Extract method, move method, move field.

- La clase *Persoonal* utiliza directamente las colecciones de *Persoona* y *GuiaTelefonica*, presentando claramente **envidia de atributos**. Relizamos **extract method** en las lineas que presentan dicho uso directo y luego podemos realizar **move method** para delegar dichas acciones a las clases que poseen cada colección.
- Pasa lo mismo con el uso de constructores vacíos y luego setters, por lo que es muy sugerente hacer un **extract method** y luego un **move method** para crear constructores no vacíos en *Persoona* y *Llamada*. En el caso de llamada el constructor ya existía, por lo que sólo apreovechamos el uso del mismo.
- La responsabilidad del cálculo de los precios por cada llamada es de *Llamada*, y la responsabilidad del cálculo de descuentos es de *Persoona*, por lo que hacemos **extract method** de ambos cálculos y luego un **move method** de ambos cálculos. Esto implica mover *descuentoJur* y *descuentoFis* a *Persoona* usando **move field**.
- Las clases *GuiaTelefonica*, *Llamada* y *Persoona* son **Data Classes** (mientras que *Persoonal* puede ser considerada **God Class**), por lo que también por eso es muy sugerente llegar al uso de dichos **move method**.

Código Modificado

```
public class Persoona {
  (...) // Obviamos las v.i. y accessors que no cambian, para simplificar la lectura
        // Y entendimiento de los cambios
  private double descuentoJur = 0.15;
  private double descuentoFis = 0;

  public Persoona(String tipo, String nombreApellido,
    String telefono, String cuit, String documento, Persoonal sistema) {
    this.tipo = tipo;
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
    this.cuit = cuit;
    this.documento = documento;
    this.sistema = sistema;
  }
  public void agregarLlamada(Llamada llamada) {
    this.llamadas.add(llamada);
  }
  public double aplicarDescuento(double monto) {
    if (this.tipo == "fisica") {
      return monto - monto*descuentoFis;
    }
    return monto - monto*descuentoJur;
  }
}
```

```
public class Llamada {
  (...) // Obviamos las v.i. y métodos que no cambian para facilitar la lectura
  public double costoLlamada() {
    if (this.tipoDeLlamada == "nacional") {
      return this.duracion *3 + (this.duracion*3*0.21);
    }
    return this.duracion *200 + (this.duracion*200*0.21);
  }
}
```

```
public class GuiaTelefonica {

  private SortedSet<String> telefonos = new TreeSet<String>();

  public SortedSet<String> getTelefonos() {
    return telefonos;
  }

  public void setTelefonos(SortedSet<String> telefonos) {
    this.telefonos = telefonos;
  }

  public boolean existeTelefono(String telefono) {
    return this.telefonos.contains(telefono);
  }

  public void agregarTelefono(String telefono) {
    this.telefonos.add(telefono);
  }

  public void eliminarTelefono(String telefono) {
    this.telefonos.remove(telefono);
  }

  public String ultimoTelefono() {
    return this.telefonos.last();
  }
}
```

```
public class Persoonal {
  List<Persoona> personas = new ArrayList<Persoona>();
  List<Llamada> llamadas = new ArrayList<Llamada>();
  GuiaTelefonica guia = new GuiaTelefonica();

  public boolean agregarTelefono(String telefono) {
    boolean encontre = guia.existeTelefono(telefono);
    if (!encontre) {
      guia.agregarTelefono(telefono);
      encontre= true;
      return encontre;
    }
    else {
      encontre = false;
      return encontre;
    }
  }

  public Persoona registrarUsuario(String identificacion, String nombreApellido,
      String tipo) {
    Persoona persona = null;
    if (tipo.equals("fisica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new Persoona(tipo, nombreApellido, telefono,
          null, identificacion, this);
      personas.add(persona);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new Persoona(tipo, nombreApellido, telefono, identificacion,
          null, this);
      personas.add(persona);
    }
    return persona;
  }

  public boolean eliminarUsuario(Persoona persona) {
    List<Persoona> personasNueva = (personas.stream()
        .filter(p -> p != persona).collect(Collectors.toList()));
    boolean borre = false;
    if (personasNueva.size() < personas.size()) {
      this.personas = personasNueva;
      this.guia.agregarTelefono(persona.getTelefono());
      borre = true;
    }
    return borre;

  }

  public Llamada registrarLlamada(Persoona emisor, Persoona remitente,
      String tipoLlamada, int duracion) {
    Llamada llamada = new Llamada(tipoLlamada, emisor.getTelefono(),
```

```
        remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }

  public double calcularMontoTotalLlamadas(Persoona persona) {
    double montoTotal = 0;
    Persoona personaAux = null;
    for (Persoona p : personas) {
      if (p.getTelefono() == persona.getTelefono()) {
        personaAux = p;
        break;
      }
    }
    if (personaAux == null) return montoTotal;
    if (personaAux != null) {
      for (Llamada l : personaAux.getLlamadas()) {
        double costoAux = 0;
        costoAux = persona.aplicarDescuento(l.costoLlamada());
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }

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

  public boolean existeUsuario(Persoona persona) {
    return personas.contains(persona);
  }
}
```

▼ **4° Paso**

Bad Smell

Nombre: Constructor Innecesario

Extracto:

```
public class Llamada {
  public Llamada() {}
  public Llamada(String tipoDeLlamada, String emisor, String remitente, int duracion) {
    this.tipoDeLlamada = tipoDeLlamada;
    this.emisor = emisor;
    this.remitente = remitente;
    this.setDuracion(duracion);
  }
  (...)
}
```

Refactoring

Nombre: Remove Method

- Además de no ser utilizado, no es para nada útil tener un constructor vacío para llamada
- Quitamos el constructor vacío del código

Código Modificado

```
public class Llamada {
  public Llamada(String tipoDeLlamada, String emisor, String remitente, int duracion) {
    this.tipoDeLlamada = tipoDeLlamada;
    this.emisor = emisor;
    this.remitente = remitente;
    this.setDuracion(duracion);
  }
  (...)
}
```

▼ **5° Paso**

Bad Smell

Nombre: Switch Statement

Extracto:

```
public class Persoona {
  (...)
  public double aplicarDescuento(double monto) {
    if (this.tipo == "fisica") {
      return monto - monto*descuentoFis;
    }
    return monto - monto*descuentoJur;
  }
}
```

```
public class Persoonal {
  (...)
  public Persoona registrarUsuario(String identificacion, String nombreApellido,
      String tipo) {
    Persoona persona = null;
    if (tipo.equals("fisica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new Persoona(tipo, nombreApellido, telefono,
          null, identificacion, this);
      personas.add(persona);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new Persoona(tipo, nombreApellido, telefono, identificacion,
          null, this);
      personas.add(persona);
    }
    return persona;
  }
}
```

Refactoring

Nombre: Form Template Method

- Persoona define dos tipos claros “fisica” y “juridica”, los cuales se diferencian sólo en el nombre de su identificación y en el monto a descontar en el método aplicarDescuento
- Primero usamos replace conditional with polymorphism para hacer una jerarquía en Persoona
- Usando extract subclass, creamos dos subclases PersoonaFisica y PersoonaJuridica, ambas heredarán de Persoona
  - PersoonaFisica va a tener campos documento y descuentoFis = 0
  - PersoonaJuridica va a tener campos cuit y descuentoJur = 0.15
  - Persoona va a declarar el método aplicarDescuento como abstracto, y cada subclase va a aplicar el descuento con la forma indicada.
    Deja de tener atributos tipo, cuit, documento, descuentoFis y descuentoJur.
    Deja de tener los accessors para dichos campos.
- El switch statement en *Persoonal* todavía no se resuelve, eso se hará en pasos posteriores.

Código Modificado

```
public abstract class Persoona {
  private List<Llamada> llamadas = new ArrayList<Llamada>();
  private String nombreApellido;
  private String telefono;
  private Persoonal sistema;
```

```
  public Persoona(String nombreApellido, String telefono, Persoonal sistema) {
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
    this.sistema = sistema;
  }
  public List<Llamada> getLlamadas() {
    return llamadas;
  }
  public void setLlamadas(List<Llamada> llamadas) {
    this.llamadas = llamadas;
  }

  public String getNombreApellido() {
    return nombreApellido;
  }

  public void setNombreApellido(String nombreApellido) {
    this.nombreApellido = nombreApellido;
  }

  public String getTelefono() {
    return telefono;
  }

  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }

  public Persoonal getSistema() {
    return sistema;
  }

  public void setSistema(Persoonal sistema) {
    this.sistema = sistema;
  }

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

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

```
public class PersoonaFisica extends Persoona{
  private String documento;
  private static double descuentoFis = 0;

  public PersoonaFisica(String nombreApellido, String telefono, String documento, Persoonal sistema) {
    super(nombreApellido, telefono, sistema);
    this.documento = documento;
  }
  public String getDocumento() {
    return documento;
  }

  public void setDocumento(String documento) {
    this.documento = documento;
  }

  public double aplicarDescuento(double monto) {
    return monto - monto * PersoonaFisica.descuentoFis;
  }

}
```

```
public class PersoonaJuridica extends Persoona {
  private String cuit;
  private static double descuentoJur = 0.15;

  public PersoonaJuridica(String nombreApellido, String telefono, String cuit, Persoonal sistema) {
    super(nombreApellido, telefono, sistema);
    this.cuit = cuit;
  }

  public String getCuit() {
    return cuit;
  }

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

```
  public double aplicarDescuento(double monto) {
    return monto - monto*PersoonaJuridica.descuentoJur;
  }

}
```

```
public class Persoonal {
  List<Persoona> personas = new ArrayList<Persoona>();
  List<Llamada> llamadas = new ArrayList<Llamada>();
  GuiaTelefonica guia = new GuiaTelefonica();

  public boolean agregarTelefono(String telefono) {
    boolean encontre = guia.existeTelefono(telefono);
    if (!encontre) {
      guia.agregarTelefono(telefono);
      encontre= true;
      return encontre;
    }
    else {
      encontre = false;
      return encontre;
    }
  }

  public Persoona registrarUsuario(String identificacion, String nombreApellido,
      String tipo) {
    Persoona persona = null;
    if (tipo.equals("fisica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new PersoonaFisica(nombreApellido, telefono, identificacion, this);
      personas.add(persona);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new PersoonaJuridica(nombreApellido, telefono, identificacion, this);
      personas.add(persona);
    }
    return persona;
  }

  public boolean eliminarUsuario(Persoona persona) {
    List<Persoona> personasNueva = (personas.stream()
        .filter(p -> p != persona).collect(Collectors.toList()));
    boolean borre = false;
    if (personasNueva.size() < personas.size()) {
      this.personas = personasNueva;
      this.guia.agregarTelefono(persona.getTelefono());
      borre = true;
    }
    return borre;

  }

  public Llamada registrarLlamada(Persoona emisor, Persoona remitente,
      String tipoLlamada, int duracion) {
    Llamada llamada = new Llamada(tipoLlamada, emisor.getTelefono(),
        remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }

  public double calcularMontoTotalLlamadas(Persoona persona) {
    double montoTotal = 0;
    Persoona personaAux = null;
    for (Persoona p : personas) {
      if (p.getTelefono() == persona.getTelefono()) {
        personaAux = p;
        break;
      }
    }
    if (personaAux == null) return montoTotal;
    if (personaAux != null) {
      for (Llamada l : personaAux.getLlamadas()) {
        double costoAux = 0;
        costoAux = persona.aplicarDescuento(l.costoLlamada());
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }
```

```
  public int cantidadDeUsuarios() {
    return personas.size();
  }

  public boolean existeUsuario(Persoona persona) {
    return personas.contains(persona);
  }
}
```

▼ **6° Paso**

Bad Smell

Nombre: Duplicated Code

Extracto:

```
public class PersoonaJuridica extends Persoona {
  (...)
  public double aplicarDescuento(double monto) {
    return monto - monto*descuentoJur;
  }
}

public class PersoonaFisica extends Persoona{
  (...)
  public double aplicarDescuento(double monto) {
    return monto - monto*descuentoFis;
  }
}
```

Refactoring

Nombre: Form Template Method

- El descuento se aplica de igual manera a ambas clases, por lo que podemos pensar en un template method definido en *Persoona.*
- Llevamos a cabo remove method en ambas subclases y redefinimos *aplicarDescuento(double monto)*.
- Ahora necesitamos definir como abstracto el método *getDescuento()*, para que sea efectuado en ambas subclases.

Código Modificado

```
public abstract class Persoona {
  private List<Llamada> llamadas = new ArrayList<Llamada>();
  private String nombreApellido;
  private String telefono;
  private Persoonal sistema;

  public Persoona(String nombreApellido, String telefono, Persoonal sistema) {
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
    this.sistema = sistema;
  }

  public List<Llamada> getLlamadas() {
    return llamadas;
  }

  public void setLlamadas(List<Llamada> llamadas) {
    this.llamadas = llamadas;
  }

  public String getNombreApellido() {
    return nombreApellido;
  }

  public void setNombreApellido(String nombreApellido) {
    this.nombreApellido = nombreApellido;
  }

  public String getTelefono() {
    return telefono;
```

```
  }

  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }

  public Persoonal getSistema() {
    return sistema;
  }

  public void setSistema(Persoonal sistema) {
    this.sistema = sistema;
  }

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

  protected abstract double getDescuento();

  public double aplicarDescuento(double monto) {
    return monto - monto * this.getDescuento();
  }
}
```

```
public class PersoonaFisica extends Persoona{
  private String documento;
  private static double descuentoFis = 0;

  public PersoonaFisica(String nombreApellido, String telefono, String documento, Persoonal sistema) {
    super(nombreApellido, telefono, sistema);
    this.documento = documento;
  }
  public String getDocumento() {
    return documento;
  }
  public void setDocumento(String documento) {
    this.documento = documento;
  }
  protected double getDescuento() {
    return PersoonaFisica.descuentoFis;
  }
}
```

```
public class PersoonaJuridica extends Persoona {
  private String cuit;
  private static double descuentoJur = 0.15;

  public PersoonaJuridica(String nombreApellido, String telefono, String cuit, Persoonal sistema) {
    super(nombreApellido, telefono, sistema);
    this.cuit = cuit;
  }
  public String getCuit() {
    return cuit;
  }
  public void setCuit(String cuit) {
    this.cuit = cuit;
  }
  protected double getDescuento() {
    return PersoonaJuridica.descuentoJur;
  }
}
```

▼ **7° Paso**

Bad Smell

Nombre: Switch Statement

Extracto:

```
public class Llamada {
  (...)
  public double costoLlamada() {
    if (this.tipoDeLlamada == "nacional") {
      return this.duracion *3 + (this.duracion*3*0.21);
    }
    return this.duracion *200 + (this.duracion*200*0.21);
```

```
  }
}
```

Refactoring

Nombre: Replace Conditional With Polymorphism

- Utilizamos **extract class** para generar dos subclases de *Llamada: LlamadaNacional* y *LlamadaInternacional.*
- *Llamada* pasará a ser abstracta, donde el único método abstracto será *costoLlamada()*.
- Efectuamos **remove variable** para deshacernos del campo *tipoDeLlamada* y sus accessors.
- Esto nos genera un nuevo condicional en *Persoonal*, el cual solucionaremos en el siguiente paso.

Código Modificado

```
public class Persoonal {
  public Llamada registrarLlamada(Persoona emisor,
      Persoona remitente, String tipoLlamada, int duracion) {
    Llamada llamada;
    if(tipoLlamada == "nacional") {
      llamada = new LlamadaInternacional(
          emisor.getTelefono(), remitente.getTelefono(), duracion);
    }
    else {
      llamada = new LlamadaNacional(
          emisor.getTelefono(), remitente.getTelefono(), duracion);
    }
    emisor.agregarLlamada(llamada);
    return llamada;
  }
}
```

```
public class LlamadaNacional extends Llamada{
  public LlamadaNacional(String emisor, String remitente, int duracion) {
    super(emisor, remitente, duracion);
  }
  public double costoLlamada() {
    return this.getDuracion() *3 + (this.getDuracion()*3*0.21);
  }
}
```

```
public class LlamadaInternacional extends Llamada {
  public LlamadaInternacional(String emisor,
      String remitente, int duracion) {
    super(emisor, remitente, duracion);
  }
  public double costoLlamada() {
    return this.getDuracion() *200 + (this.getDuracion()*200*0.21);
  }
}
```

```
public abstract class Llamada {
  private String emisor;
  private String remitente;
  private int duracion;

  public Llamada(String emisor, String remitente, int duracion) {
    this.emisor = emisor;
    this.remitente = remitente;
    this.setDuracion(duracion);
  }

  public void setEmisor(String emisor) {
    this.emisor = emisor;
  }
  public String getEmisor() {
    return this.emisor;
  }
  public String getRemitente() {
    return remitente;
```

```
  }
  public void setRemitente(String remitente) {
    this.remitente = remitente;
  }

  public int getDuracion() {
    return duracion;
  }

  public void setDuracion(int duracion) {
    this.duracion = duracion;
  }

  public abstract double costoLlamada();
}
```

▼ **8° Paso**

Bad Smell

Nombre: Switch Statement

Extracto:

```
public class Persoonal {
  (...)
  public Llamada registrarLlamada(Persoona emisor, Persoona remitente,
      String tipoLlamada, int duracion) {
    Llamada llamada;
    if(tipoLlamada == "nacional") {
      llamada = new LlamadaInternacional(
          emisor.getTelefono(), remitente.getTelefono(), duracion);
    }
    else {
      llamada = new LlamadaNacional( emisor.getTelefono(), remitente.getTelefono(), duracion);
    }
    emisor.agregarLlamada(llamada);
    return llamada;
  }
}
```

Refactoring

Nombre: Extract method y remove dead code

- Vamos a hacer **extract method** para la construcción de ambos tipos de *Llamada*, creando dos métodos: *registrarLlamadaNacional()* y *registrarLlamadaInternacional()*.
- Luego aplicamos **remove dead code** para eliminar el método *registrarLlamada*()
- Vamos a tener que modificar los tests, para que llamen al método que corresponda, pero al ser una modificación mínima, puede considerarse como un cambio valioso.

Código Modificado

```
class PersoonalTest {
    (...)
    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);
}
```

```
public class Persoonal {
  (...)
  public LlamadaInternacional registrarLlamadaInternacional(
      Persoona emisor, Persoona remitente, String tipoLlamada, int duracion) {
```

```
    LlamadaInternacional llamada = new LlamadaInternacional(
          emisor.getTelefono(), remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }
  public LlamadaNacional registrarLlamadaNacional(
      Persoona emisor, Persoona remitente, String tipoLlamada, int duracion) {
    LlamadaNacional llamada = new LlamadaNacional(
          emisor.getTelefono(), remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }
}
```

▼ **9° Paso**

Bad Smell

Nombre: Switch Statement, Duplicated Code

Extracto:

```
public Persoona registrarUsuario(String identificacion, String nombreApellido,
      String tipo) {
    Persoona persona = null;
    if (tipo.equals("fisica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new PersoonaFisica(nombreApellido, telefono, identificacion, this);
      personas.add(persona);
    }
    else if (tipo.equals("juridica")) {
      String telefono = guia.ultimoTelefono();
      guia.eliminarTelefono(telefono);
      persona = new PersoonaJuridica(nombreApellido, telefono, identificacion, this);
      personas.add(persona);
    }
    return persona;
  }
```

Refactoring

Nombre: Extract method y remove dead code

- Vamos a hacer **extract method** para la construcción de ambos tipos de *Persoona*, creando dos métodos: registrarUsuarioFisico*()* y *registrarUsuarioJuridico().*
- Luego aplicamos **remove dead code** para eliminar el método *registrarUsuario().*
- Vamos a tener que modificar los tests, para que llamen al método que corresponda, pero al ser una modificación mínima, puede considerarse como un cambio valioso.
- Utilizando **replace temp with query** y luego **inline variable** podemos eliminar código duplicado, llegando a una función un poco más mínima.

Código Modificado

```
class PersoonalTest {
    (...)
    this.emisorPersonaFisca = sistema.registrarUsuarioFisico("11555666", "Marcelo Tinelli");
    this.remitentePersonaFisica = sistema.registrarUsuarioFisico("00000001", "Mirtha Legrand");
    this.emisorPersonaJuridica = sistema.registrarUsuarioJuridico("17555222", "Felfort");
    this.remitentePersonaJuridica = sistema.registrarUsuarioJuridico("25765432", "Moovistar");
    (...)
    Persoona nuevaPersona = this.sistema.registrarUsuarioFisico("2444555","Chiche Gelblung");
    (...)
}
```

```
public class Persoonal {
  (...)
```

```
  private String nuevoNumero() {
    String telefono = guia.ultimoTelefono();
    guia.eliminarTelefono(telefono);
    return telefono;
  }
  public PersoonaFisica registrarUsuarioFisico(String identificacion, String nombreApellido) {
    PersoonaFisica persona = new PersoonaFisica(nombreApellido, nuevoNumero(), identificacion, this);
    personas.add(persona);
    return persona;
  }
  public PersoonaJuridica registrarUsuarioJuridico(String identificacion, String nombreApellido) {
    PersoonaJuridica persona = new PersoonaJuridica(nombreApellido, nuevoNumero(), identificacion, this);
    personas.add(persona);
    return persona;
  }
}
```

▼ **10° Paso**

Bad Smell

Nombre: Long Method

Extracto:

```
public class Persoonal {
  (...)
  public boolean agregarTelefono(String telefono) {
    boolean encontre = guia.getTelefonos().contains(telefono);
    if (!encontre) {
      guia.getTelefonos().add(telefono);
      encontre= true;
      return encontre;
    }
    else {
      encontre = false;
      return false;
    }
  }
}
```

Refactoring

Nombre: Inline Temp, Replace **Nested Conditional with Guard Clauses**

- Primero utilizamos inline temp para deshacernos de *encontre.*
- Luego, utilizamos replace nested conditional with guard clauses, para simplificar los retornos

Código Modificado

```
public class Persoonal {
  (...)
  public boolean agregarTelefono(String telefono) {
    if (guia.existeTelefono(telefono)) return false;
    guia.agregarTelefono(telefono);
    return true;
  }
}
```

▼ **11° Paso**

Bad Smell

Nombre: Long Method

Extracto:

```
public class Persoonal {
  (...)
  public boolean eliminarUsuario(Persoona persona) {
    List<Persoona> personasNueva = (personas.stream()
        .filter(p -> p != persona)
```

```
        .collect(Collectors.toList()));
    boolean borre = false;
    if (personasNueva.size() < personas.size()) {
      this.personas = personasNueva;
      this.guia.agregarTelefono(persona.getTelefono());
      borre = true;
    }
    return borre;
  }
}
```

Refactoring

Nombre: Inline Temp, Replace **Nested Conditional with Guard Clauses**

- Primero utilizamos inline temp para deshacernos de *borre* y *personasNueva*
- Luego, utilizamos replace nested conditional with guard clauses, para simplificar los retornos

Código Modificado

```
public class Persoonal {
  (...)
  public boolean eliminarUsuario(Persoona persona) {
    if (! personas.contains(persona)) return false;
    this.personas = (personas.stream().filter(p -> p != persona).collect(Collectors.toList()));
    this.guia.agregarTelefono(persona.getTelefono());
    return true;
  }
}
```

▼ **12° Paso**

Bad Smell

Nombre: Long Method, Dead Code, loops

Extracto:

```
  public double calcularMontoTotalLlamadas(Persoona persona) {
    double montoTotal = 0;
    Persoona personaAux = null;
    for (Persoona p : personas) {
      if (p.getTelefono() == persona.getTelefono()) {
        personaAux = p;
        break;
      }
    }
    if (personaAux == null) return montoTotal;
    if (personaAux != null) {
      for (Llamada l : personaAux.getLlamadas()) {
        double costoAux = 0;
        costoAux = persona.aplicarDescuento(l.costoLlamada());
        montoTotal += costoAux;
      }
    }
    return montoTotal;
  }
```

Refactoring

Nombre: Rename Variable, Inline Temp, Replace Nested Conditional with Guard Clauses, Replace Loop with Pipeline, Remove Dead Code

- Utilizando inline temp, rename variable y remove dead code podemos eliminar *montoTotal y costoAux*, quedándonos sólo con una variable *costo.*
- Utilizando remove dead code podemos deshacernos de *if (personaAux == null) return(…).*
- Utilizando replace loop with pipeline simplificamos y acortamos mucho el método.

- Con replace nested conditional with guard simplificamos los return.
- Utilizando replace temp with query podemos eliminar personaAux.
- Nuevamente con inline temp, también podemos deshacernos de la variable *costo.*
- Los test se modifican levemente para aumentar el margen de error de los tests de esta función

Código Modificado

```
public class PersoonalTest {
  (...)
  assertEquals(this.sistema.calcularMontoTotalLlamadas(emisorPersonaFisca), 15105.640000000001, 0.000001);
  assertEquals(this.sistema.calcularMontoTotalLlamadas(emisorPersonaJuridica), 3131.7825000000003, 0.000001);
  (...)
}
```

```
public class Persoonal {
  (...)
  public double calcularMontoTotalLlamadas(Persoona persona) {
    if (! personas.contains(persona)) return 0;
    return persona.getLlamadas().stream()
          .mapToDouble(l->l.costoLlamada()-l.costoLlamada()*persona.getDescuento()).sum();
  }
}
```

▼ **13° Paso**

Bad Smell

Nombre: Middle Man, Feature Envy

Extracto:

```
public class Persoonal {
  (...)
  public double calcularMontoTotalLlamadas(Persoona persona) {
    if (! personas.contains(persona)) return 0;
    return persona.getLlamadas().stream()
          .mapToDouble(l->l.costoLlamada()-l.costoLlamada()*persona.getDescuento()).sum();
  }
}
```

Refactoring

Nombre: Move Method

- La responsabilidad del cálculo de esta función en realidad pertenece a Persoona, por lo que delegamos la función a la misma, utilizando move method

Código Modificado

```
public class Persoona {
  (...)
  public double calcularMontoTotalLlamadas() {
    return llamadas.stream().mapToDouble(l -> aplicarDescuento(l.costoLlamada())).sum();
  }
}
```

```
public class Persoonal {
  (...)
  public double calcularMontoTotalLlamadas(Persoona persona) {
    if(! this.personas.contains(persona)) return 0;
    return persona.calcularMontoTotalLlamadas();
```

```
  }
}
```

▼ **14° Paso**

Bad Smell

Nombre: Dead Code

Extracto:

```
public abstract class Persoona {
  private Persoonal sistema;
  public Persoona(String nombreApellido, String telefono, Persoonal sistema) {
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
    this.sistema = sistema;
  }

  public Persoonal getSistema() {
    return sistema;
  }

  public void setSistema(Persoonal sistema) {
    this.sistema = sistema;
  }
  (...)
}
```

Refactoring

Nombre: Remove Dead Code

- El atributo *sistema* de *Persoona* está completamente en desuso y no cumple ninguna función en cuanto a roles
- Utilizamos **remove field** y lo eliminamos

Código Modificado

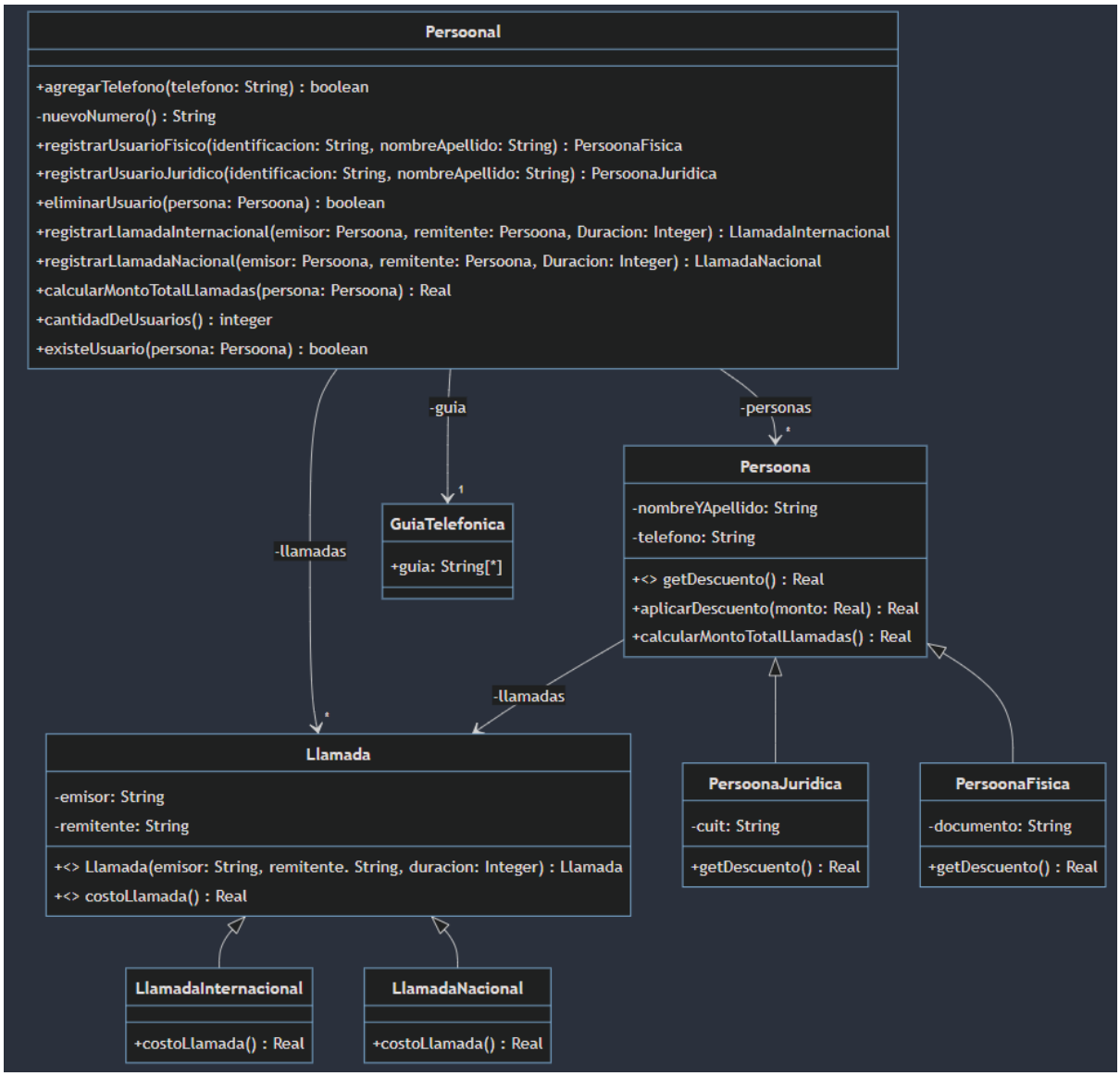
```
public class Persoonal {
  public PersoonaFisica registrarUsuarioFisico(String identificacion, String nombreApellido) {
    PersoonaFisica persona = new PersoonaFisica(nombreApellido, nuevoNumero(), identificacion);
    personas.add(persona);
    return persona;
  }
  public PersoonaJuridica registrarUsuarioJuridico(String identificacion, String nombreApellido) {
    PersoonaJuridica persona = new PersoonaJuridica(nombreApellido, nuevoNumero(), identificacion);
    personas.add(persona);
    return persona;
  }
  (...)
}
```

```
public class PersoonaJuridica extends Persoona {
  public PersoonaJuridica(String nombreApellido, String telefono, String cuit) {
    super(nombreApellido, telefono);
    this.cuit = cuit;
  }
  (...)
}
```

```
public class PersoonaFisica extends Persoona{
  public PersoonaFisica(String nombreApellido, String telefono, String documento) {
    super(nombreApellido, telefono);
    this.documento = documento;
  }
  (...)
}
```

```
public abstract class Persoona {
  public Persoona(String nombreApellido, String telefono) {
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
  }
  (...)
}
```

UML Final



Código Refacrorizado

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

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class PersoonalTest {

  Persoonal sistema;
  Persoona emisorPersonaFisca, remitentePersonaFisica, emisorPersonaJuridica, remitentePersonaJuridica;
```

```
  @BeforeEach
  public void setUp() {
    this.sistema = new Persoonal();
    this.sistema.agregarTelefono("2214444554");
    this.sistema.agregarTelefono("2214444555");
    this.sistema.agregarTelefono("2214444556");
    this.sistema.agregarTelefono("2214444557");

    this.emisorPersonaFisca = sistema.registrarUsuarioFisico("11555666", "Marcelo Tinelli");
    this.remitentePersonaFisica = sistema.registrarUsuarioFisico("00000001", "Mirtha Legrand");
    this.emisorPersonaJuridica = sistema.registrarUsuarioJuridico("17555222", "Felfort");
    this.remitentePersonaJuridica = sistema.registrarUsuarioJuridico("25765432", "Moovistar");

    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);

  }

  @Test
  void testcalcularMontoTotalLlamadas() {
    assertEquals(this.sistema.calcularMontoTotalLlamadas(emisorPersonaFisca), 15105.640000000001, 0.000001);
    assertEquals(this.sistema.calcularMontoTotalLlamadas(emisorPersonaJuridica), 3131.7825000000003, 0.000001);
    assertEquals(this.sistema.calcularMontoTotalLlamadas(remitentePersonaFisica), 0);
    assertEquals(this.sistema.calcularMontoTotalLlamadas(remitentePersonaJuridica), 0);
  }

  @Test
  void testAgregarUsuario() {
    assertEquals(this.sistema.cantidadDeUsuarios(), 4);
    this.sistema.agregarTelefono("2214444558");
    Persoona nuevaPersona = this.sistema.registrarUsuarioFisico("2444555","Chiche Gelblung");

    assertEquals(this.sistema.cantidadDeUsuarios(), 5);
    assertTrue(this.sistema.existeUsuario(nuevaPersona));

  }

  @Test
  void testEliminarUsuario() {
    assertEquals(this.sistema.cantidadDeUsuarios(), 4);
    assertTrue(this.sistema.existeUsuario(emisorPersonaFisca));
    this.sistema.eliminarUsuario(emisorPersonaFisca);
    assertEquals(this.sistema.cantidadDeUsuarios(), 3);
    assertFalse(this.sistema.existeUsuario(emisorPersonaFisca));
  }

}
```

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

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class Persoonal {
  List<Persoona> personas = new ArrayList<Persoona>();
  List<Llamada> llamadas = new ArrayList<Llamada>();
  GuiaTelefonica guia = new GuiaTelefonica();

  public boolean agregarTelefono(String telefono) {
    if (guia.existeTelefono(telefono)) return false;
    guia.agregarTelefono(telefono);
    return true;
  }

  private String nuevoNumero() {
    String telefono = guia.ultimoTelefono();
    guia.eliminarTelefono(telefono);
    return telefono;
  }

  public PersoonaFisica registrarUsuarioFisico(String identificacion, String nombreApellido) {
    PersoonaFisica persona = new PersoonaFisica(nombreApellido, nuevoNumero(), identificacion);
    personas.add(persona);
    return persona;
  }
```

```
  public PersoonaJuridica registrarUsuarioJuridico(String identificacion, String nombreApellido) {
    PersoonaJuridica persona = new PersoonaJuridica(nombreApellido, nuevoNumero(), identificacion);
    personas.add(persona);
    return persona;
  }

  public boolean eliminarUsuario(Persoona persona) {
    if (! personas.contains(persona)) return false;
    this.personas = (personas.stream()
        .filter(p -> p != persona)
        .collect(Collectors.toList()));;
    this.guia.agregarTelefono(persona.getTelefono());
    return true;
  }

  public LlamadaInternacional registrarLlamadaInternacional(Persoona emisor, Persoona remitente, int duracion) {
    LlamadaInternacional llamada = new LlamadaInternacional(emisor.getTelefono(), remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }

  public LlamadaNacional registrarLlamadaNacional(Persoona emisor, Persoona remitente, int duracion) {
    LlamadaNacional llamada = new LlamadaNacional(emisor.getTelefono(), remitente.getTelefono(), duracion);
    emisor.agregarLlamada(llamada);
    return llamada;
  }

  public double calcularMontoTotalLlamadas(Persoona persona) {
    if(! this.personas.contains(persona)) return 0;
    return persona.calcularMontoTotalLlamadas();
  }

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

  public boolean existeUsuario(Persoona persona) {
    return personas.contains(persona);
  }
}
```

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

public abstract class Llamada {
  private String emisor;
  private String remitente;
  private int duracion;

  public Llamada(String emisor, String remitente, int duracion) {
    this.emisor = emisor;
    this.remitente = remitente;
    this.setDuracion(duracion);
  }

  public void setEmisor(String emisor) {
    this.emisor = emisor;
  }
  public String getEmisor() {
    return this.emisor;
  }
  public String getRemitente() {
    return remitente;
  }
  public void setRemitente(String remitente) {
    this.remitente = remitente;
  }

  public int getDuracion() {
    return duracion;
  }

  public void setDuracion(int duracion) {
    this.duracion = duracion;
  }

  public abstract double costoLlamada();
}
```

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

public class LlamadaInternacional extends Llamada {
  public LlamadaInternacional(String emisor, String remitente, int duracion) {
```

```
    super(emisor, remitente, duracion);
  }
  public double costoLlamada() {
    return this.getDuracion() *200 + (this.getDuracion()*200*0.21);
  }
}
```

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

public class LlamadaNacional extends Llamada{
  public LlamadaNacional(String emisor, String remitente, int duracion) {
    super(emisor, remitente, duracion);
  }
  public double costoLlamada() {
    return this.getDuracion() *3 + (this.getDuracion()*3*0.21);
  }
}
```

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

import java.util.ArrayList;
import java.util.List;

public abstract class Persoona {
  private List<Llamada> llamadas = new ArrayList<Llamada>();
  private String nombreApellido;
  private String telefono;

  public Persoona(String nombreApellido, String telefono) {
    this.nombreApellido = nombreApellido;
    this.telefono = telefono;
  }

  public List<Llamada> getLlamadas() {
    return llamadas;
  }

  public void setLlamadas(List<Llamada> llamadas) {
    this.llamadas = llamadas;
  }

  public String getNombreApellido() {
    return nombreApellido;
  }

  public void setNombreApellido(String nombreApellido) {
    this.nombreApellido = nombreApellido;
  }

  public String getTelefono() {
    return telefono;
  }

  public void setTelefono(String telefono) {
    this.telefono = telefono;
  }

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

  protected abstract double getDescuento();

  public double aplicarDescuento(double monto) {
    return monto - monto * this.getDescuento();
  }

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

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

public class PersoonaFisica extends Persoona{
  private String documento;
  private static double descuentoFis = 0;

  public PersoonaFisica(String nombreApellido, String telefono, String documento) {
```

```
    super(nombreApellido, telefono);
    this.documento = documento;
  }
  public String getDocumento() {
    return documento;
  }
  public void setDocumento(String documento) {
    this.documento = documento;
  }
  protected double getDescuento() {
    return PersoonaFisica.descuentoFis;
  }
}
```

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

public class PersoonaJuridica extends Persoona {
  private String cuit;
  private static double descuentoJur = 0.15;

  public PersoonaJuridica(String nombreApellido, String telefono, String cuit) {
    super(nombreApellido, telefono);
    this.cuit = cuit;
  }
  public String getCuit() {
    return cuit;
  }
  public void setCuit(String cuit) {
    this.cuit = cuit;
  }
  protected double getDescuento() {
    return PersoonaJuridica.descuentoJur;
  }
}
```

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

import java.util.TreeSet;
import java.util.SortedSet;

public class GuiaTelefonica {

  private SortedSet<String> telefonos = new TreeSet<String>();

  public SortedSet<String> getTelefonos() {
    return telefonos;
  }

  public void setTelefonos(SortedSet<String> telefonos) {
    this.telefonos = telefonos;
  }

  public boolean existeTelefono(String telefono) {
    return this.telefonos.contains(telefono);
  }

  public void agregarTelefono(String telefono) {
    this.telefonos.add(telefono);
  }

  public void eliminarTelefono(String telefono) {
    this.telefonos.remove(telefono);
  }

  public String ultimoTelefono() {
    return this.telefonos.last();
  }
}
```