Ejercicio 1

1.1 Protocolo de Cliente

Malor olor: Nombre poco descriptivo.
Refactoring que lo corrige: Rename method. El nombre de los métodos debe ser representativo a su comportamiento.

```
/**
* Retorna el li´mite de cre´dito del cliente
*/
protected double lmtCrdt() {...
```

Aplicando refactoring:

```
protected double limiteDeCredito() {...
```

Malor olor: Nombre poco descriptivo.
Refactoring que lo corrige: Rename.

```
/**
* Retorna el monto facturado al cliente desde la fecha f1 a la fecha f2
*/
protected double mtFcE(LocalDate f1, LocalDate f2) {...
```

Aplicando refactoring:

```
protected double montoFacturado(LocalDate f1, LocalDate f2) {...
```

Malor olor: Nombre poco descriptivo.
Refactoring que lo corrige: Rename.

```
/**
* Retorna el monto cobrado al cliente desde la fecha f1 a la fecha f2
*/
protected double mtCbE(LocalDate f1, LocalDate f2) {...
```

Aplicando refactoring:

```
protected double montoCobrado(LocalDate f1, LocalDate f2) {...
```

1.2 Participación en proyectos

Malor olor: Envidia de atributos.
Refactoring que lo corrige: Move method. Es apropiado porque ese método se le tiene que delegar a Proyecto porque tiene las variables de instancia para resolverlo y si no se sobrecarga la clase Cliente con comportamiento que no le corresponde.

Diseño inicial:

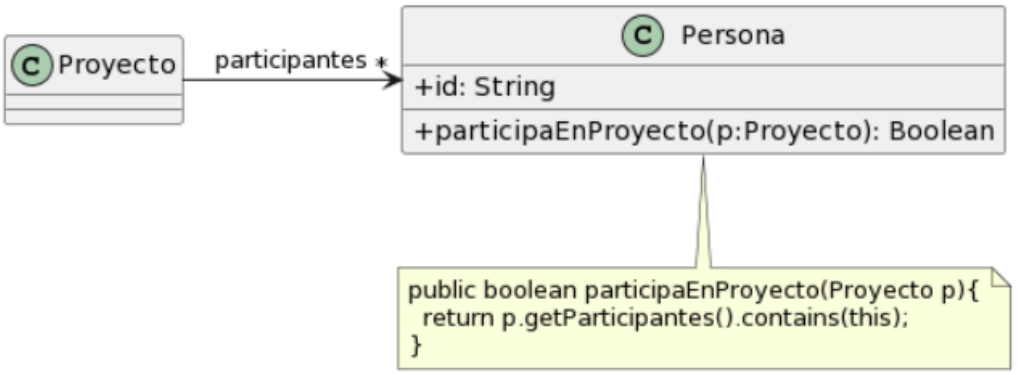


Figura 1: Diagrama de clases del diseño inicial.

Diseño revisado:

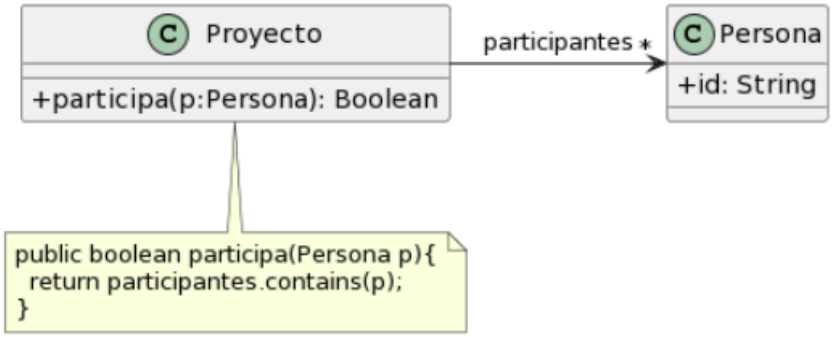


Figura 2: Diagrama de clases modificado.

1.3 Cálculos

```
public void imprimirValores() {
   int totalEdades = 0;
   double promedioEdades = 0;
   double totalSalarios = 0;

   for (Empleado empleado : personal) {
         totalEdades = totalEdades + empleado.getEdad();
         totalSalarios = totalSalarios + empleado.getSalario();
   }
   promedioEdades = totalEdades / personal.size();

   String message = String.format("El promedio de las edades es %s y el total de
salarios es %s", promedioEdades, totalSalarios);

   System.out.println(message);

}
```

Malor olor: Método2 muy largo.
Refactoring que lo corrige: Compose method.

Aplicando el refactoring:

public double calcularPromedioDeEdades(){

```
    int totalEdades = 0;
    double promedioEdades = 0;

    for (Empleado empleado : personal)
          totalEdades = totalEdades + empleado.getEdad();

    promedioEdades = totalEdades / personal.size();
return primedioEdades;
}
```

public double calcularTotalDeSalarios(){

```
    double totalSalarios = 0;
    for (Empleado empleado : personal)
          totalSalarios = totalSalarios + empleado.getSalario();
    return totalSalarios;
}

public void imprimirValores() {
    String message = String.format("El promedio de las edades es %s y el total de
salarios es %s",this.calcularPromedioDeEdades(), this.calcularTotalDeSalarios());

    System.out.println(message);

}
```

Malor olor: Variables de instancia que deberían ser temporales y reinventar la rueda.

Refactoring que lo corrige: aplicamos una libreria/biblioteca ya implementada.

Aplicando el refactoring:

public double calcularPromedioDeEdades(){

```
return personal.stream()
                    .mapToInt(Empleado::getEdad)
                    .average()
                    .orElse(0.0);
}
```

public double calcularTotalDeSalarios(){

```
    return personal.stream()
            .mapToDouble(Empleado::getSalario)
            .sum();
}

public void imprimirValores() {
    String message = String.format("El promedio de las edades es %s y el total de
salarios es %s",this.calcularPromedioDeEdades(), this.calcularTotalDeSalarios());

    System.out.println(message);

}
```

2.1 Empleados

```
public class EmpleadoTemporario {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;
    public double horasTrabajadas = 0;
    public int cantidadHijos = 0;
    // ......

    public double sueldo() {
          return this.sueldoBasico
                  +  (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000)
                  -  (this.sueldoBasico * 0.13);
              }
          }

public class EmpleadoPlanta {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;
    public int cantidadHijos = 0;
    // ......

    public double sueldo() {
        return this.sueldoBasico
                + (this.cantidadHijos * 2000)
                - (this.sueldoBasico * 0.13);
    }
}

public class EmpleadoPasante {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;
    // ......

    public double sueldo() {
        return this.sueldoBasico - (this.sueldoBasico * 0.13);
    }
}
```

Malor olor: Es un. Todas las clases comparten variables y son un empleado.
Refactoring que lo corrige: Creamos una clase padre.

1 Creo la clase padre

```
public class Empleado {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;
```

```
    public double sueldo() {
          return this.sueldoBasico
                  +  (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000)
                  -  (this.sueldoBasico * 0.13);
              }
          }
public class EmpleadoTemporario extends Empleado {
   public double horasTrabajadas = 0;
    public double sueldo() {
          return this.sueldoBasico
                  +  (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000)
                  -  (this.sueldoBasico * 0.13);
              }
          }

public class EmpleadoPlanta extends Empleado{
    public int cantidadHijos = 0;

    public double sueldo() {
        return this.sueldoBasico
                + (this.cantidadHijos * 2000)
                - (this.sueldoBasico * 0.13);
    }
}

public class EmpleadoPasante extends Empleado{
    public double sueldo() {
        return this.sueldoBasico - (this.sueldoBasico * 0.13);
    }
}
```

2 Mal olor: código repetido
Refactoring para corregir: From templated method

2.1 Compose method

```
public class Empleado {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;

    public double sueldo() {
          return this.sueldoBasico
                  +  (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000)
                  -  (this.sueldoBasico * 0.13);
              }
          }

public class EmpleadoTemporario extends Empleado {
   public double horasTrabajadas = 0;
```

```
   public int cantidadHijos = 0;


   public double adicional(){
         return (this.horasTrabajadas * 500)
                  +  (this.cantidadHijos * 1000);
   }
   public double resta(){
         return (this.sueldoBasico * 0.13);
   }


public class EmpleadoPlanta extends Empleado{
    public int cantidadHijos = 0;

   public double adicional(){
         return (this.cantidadHijos * 2000);
   }
   public double resta(){
         return (this.sueldoBasico * 0.13);
   }
}

public class EmpleadoPasante extends Empleado{
   public double adicional(){
         return 0;
   }
   public double resta(){
         return (this.sueldoBasico * 0.13);
   }

}
```

2.2 Pull up

```
public class Empleado {
    public String nombre;
    public String apellido;
    public double sueldoBasico = 0;

   public abstract double adicional();
   public double resta(){
         return (this.sueldoBasico * 0.13);
   }
    public double sueldo() {
          return this.sueldoBasico
                  +  this.adicional()
                  -  this.resta()
               }
          }

public class EmpleadoTemporario extends Empleado {
   public double horasTrabajadas = 0;
   public int cantidadHijos = 0;
```

```
    public double adicional(){
          return (this.horasTrabajadas * 500)
                   +  (this.cantidadHijos * 1000);
    }
```

```
public class EmpleadoPlanta extends Empleado{
    public int cantidadHijos = 0;

    public double adicional(){
          return (this.cantidadHijos * 2000);
    }
}

public class EmpleadoPasante extends Empleado{
    public double adicional(){
          return 0;
    }
}
```

2.2 Juego

```
public class Juego {
    public void incrementar(Jugador j) {
        j.puntuacion = j.puntuacion + 100;
    }
    public void decrementar(Jugador j) {
        j.puntuacion = j.puntuacion - 50;
    }
}
```

```
public class Jugador {
    public String nombre;
    public String apellido;
    public int puntuacion = 0;
}
```

Mal olor: Envidia de atributos
Refactoring para corregir: Move method. Ya que la clase Juego está operando con un atributo de la clase Jugador y no es una tarea bien delegada.

```
public class Juego {
    public int puntuacion = 0;
    public void incrementar() {
        puntuacion = puntuacion + 100;
    }
    public void decrementar() {
        puntuacion = puntuacion - 50;
    }
```

```
    public String nombre;
    public String apellido;
    public Juego juego;
}
```

```
   public class Jugador {
```

2.3 Publicaciones


```java
/**
* Retorna los últimos N posts que no pertenecen al usuario user
*/
public List<Post> ultimosPosts(Usuario user, int cantidad) {

    List<Post> postsOtrosUsuarios = new ArrayList<Post>();
    for (Post post : this.posts) {
        if (!post.getUsuario().equals(user)) {
            postsOtrosUsuarios.add(post);
        }
    }

   // ordena los posts por fecha
   for (int i = 0; i < postsOtrosUsuarios.size(); i++) {
       int masNuevo = i;
       for(int j= i +1; j < postsOtrosUsuarios.size(); j++) {
           if (postsOtrosUsuarios.get(j).getFecha().isAfter(
     postsOtrosUsuarios.get(masNuevo).getFecha())) {
               masNuevo = j;
           }
       }
      Post unPost = postsOtrosUsuarios.set(i,postsOtrosUsuarios.get(masNuevo));
      postsOtrosUsuarios.set(masNuevo, unPost);
   }

    List<Post> ultimosPosts = new ArrayList<Post>();
    int index = 0;
    Iterator<Post> postIterator = postsOtrosUsuarios.iterator();
    while (postIterator.hasNext() &&  index < cantidad) {
        ultimosPosts.add(postIterator.next());
    }
    return ultimosPosts;
}
```

Mal olor: Método largo
Refactoring para corregir: Compose method.

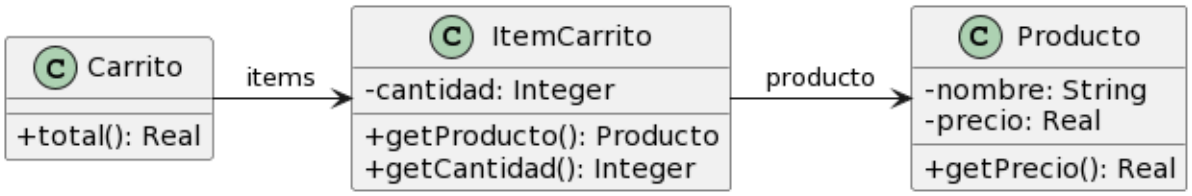
```
private List<Post> postsOtrosUsuarios(Usuario user){
        List<Post> postsOtrosUsuarios = new ArrayList<Post>();
            for (Post post : this.posts) {
                if (!post.getUsuario().equals(user)) {
                    postsOtrosUsuarios.add(post);
                }
            }
        return postOtrosUsuarios;
    }
    private List<Post> ordenarPostsPorFecha (Usuario user){
        // ordena los posts por fecha
           List<Post> postsOtrosUsuarios = this.postsOtrosUsuarios(user)
           for (int i = 0; i < postsOtrosUsuarios.size(); i++) {
               int masNuevo = i;
               for(int j= i +1; j < postsOtrosUsuarios.size(); j++) {
                   if (postsOtrosUsuarios.get(j).getFecha().isAfter(
             postsOtrosUsuarios.get(masNuevo).getFecha())) {
                       masNuevo = j;
                   }
               }
              Post unPost =
        postsOtrosUsuarios.set(i,postsOtrosUsuarios.get(masNuevo));
              postsOtrosUsuarios.set(masNuevo, unPost);
           }
    }
    private List<Post> ultimosCantidadPosts(Usuario user, int cantidad){
           List<Post> ultimosPosts = new ArrayList<Post>();
            int index = 0;
            Iterator<Post> postIterator = ordenarPostsPorFecha(user).iterator();
            while (postIterator.hasNext() &&  index < cantidad) {
                ultimosPosts.add(postIterator.next());
            }
            return ultimosPosts;
           }

public List<Post> ultimosPosts(Usuario user, int cantidad) {
    return this.ultimosCandidadPosts(user,cantidad)
}
```

Malor olor: reinventar la rueda.
Refactoring que lo corrige: aplicamos una libreria/biblioteca ya implementada.

```
private List<Post> postsOtrosUsuarios(Usuario user) {
   return this.posts.stream()
               .filter(post -> !post.getUsuario().equals(user))
               .collect(Collectors.toList());
}

private List<Post> ordenarPostsPorFecha(Usuario user) {
   return postsOtrosUsuarios(user).stream()
                        .sorted((p1, p2) -> p2.getFecha().compareTo(p1.getFecha()))
                        .collect(Collectors.toList());
```

```
}
private List<Post> ultimosCantidadPosts(Usuario user, int cantidad) {
    return ordenarPostsPorFecha(user).stream()
                                .limit(cantidad)
                                .collect(Collectors.toList());
}

public List<Post> ultimosPosts(Usuario user, int cantidad) {
    return this.ultimosCantidadPosts(user, cantidad);
```

2.4 Carrito de compras

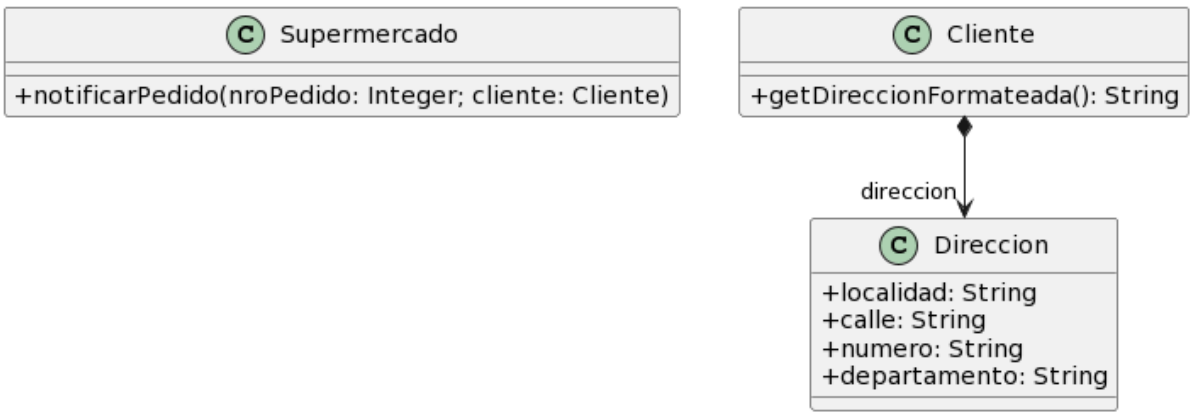



```
public class Producto {
    private String nombre;
    private double precio;

    public double getPrecio() {
        return this.precio;
    }
}

public class ItemCarrito {
    private Producto producto;
    private int cantidad;

    public Producto getProducto() {
        return this.producto;
    }

    public int getCantidad() {
        return this.cantidad;
    }

}

public class Carrito {
    private List<ItemCarrito> items;

    public double total() {
return this.items.stream().mapToDouble(item -> item.getProducto().getPrecio() *
item.getCantidad()).sum();
    }
}
```

Mal olor: Envidia de atributos y Romper el encapsulamiento.
Refactoring para corregir: Move method. Ya que la clase ItemCarrito está operando con atributos de las clases Carrito y Producto y no es una tarea bien delegada

```
public class Producto {
    private String nombre;
    private double precio;

    public double getPrecio() {
        return this.precio;
    }
}

public class ItemCarrito {
    private Producto producto;
    private int cantidad;

   public double total() {
         return this.cantidad * this.producto.getPrecio;
    }

}

public class Carrito {
    private List<ItemCarrito> items;

    public double total() {
         return this.items.stream().mapToDouble(item -> item.total()).sum();
    }
}
```

2.5 Envío de pedidos




```
public class Supermercado {
   public void notificarPedido(long nroPedido, Cliente cliente) {
     String notificacion = MessageFormat.format("Estimado cliente, se le informa que
hemos recibido su pedido con número {0}, el cual será enviado a la dirección {1}",
new Object[] { nroPedido, cliente.getDireccionFormateada() });

     // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
```

```
    System.out.println(notificacion);
  }
}

public class Cliente {
   public String getDireccionFormateada() {
   return
         this.direccion.getLocalidad() + ", " +
         this.direccion.getCalle() + ", " +
         this.direccion.getNumero() + ", " +
         this.direccion.getDepartamento()
     ;
}
```

Mal olor: Envidia de atributos, Romper el encapsulamiento y clase anémica.
Refactoring para corregir: Move method. Ya que la clase Cliente está operando con un atributo de la clase Dirección y no es una tarea bien delegada. La clase Dirección no tiene ningún comportamiento.

```
public class Supermercado {}
   public void notificarPedido(long nroPedido, Cliente cliente) {
     String notificacion = MessageFormat.format("Estimado cliente, se le informa que
hemos recibido su pedido con número {0}, el cual será enviado a la dirección {1}",
new Object[] { nroPedido, cliente.getDireccionFormateada() });

     // lo imprimimos en pantalla, podría ser un mail, SMS, etc..
    System.out.println(notificacion);
  }
}

public class Direccion {
   private String localidad;
   private String calle;
   private int numero;
   private String departamento;

   public String toString() {
   return
         this.localidad() + ", " +
         this.calle() + ", " +
         this.numero() + ", " +
         this.departamento()
     ;
}

public class Cliente {
   private Direccion direccion;
   public String getDireccionFormateada() {
   return
         this.direccion.toString();
     ;
}
```