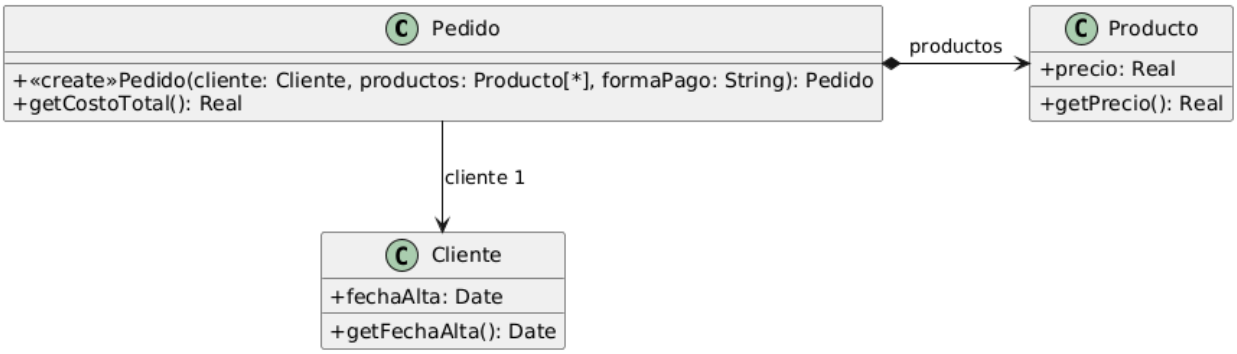
Ejercicio 4




```
public class Pedido {
    private Cliente cliente;
    private List<Producto> productos;
    private String formaPago;

    public Pedido(Cliente cliente, List<Producto> productos,
        String formaPago) {
        if (!"efectivo".equals(formaPago)
            && !"6 cuotas".equals(formaPago)
            && !"12 cuotas".equals(formaPago)) {
                throw new Error("Forma de pago incorrecta");
        }
        this.cliente = cliente;
        this.productos = productos;
        this.formaPago = formaPago;
    }

    public double getCostoTotal() {
        double costoProductos = 0;
        for (Producto producto : this.productos) {
            costoProductos += producto.getPrecio();
        }
```

```
        double extraFormaPago = 0;
        if ("efectivo".equals(this.formaPago)) {
             extraFormaPago = 0;
        } else if ("6 cuotas".equals(this.formaPago)) {
            extraFormaPago = costoProductos * 0.2;
        } else if ("12 cuotas".equals(this.formaPago)) {
             extraFormaPago = costoProductos * 0.5;
        }
        int añosDesdeFechaAlta = Period.between(this.cliente.
           getFechaAlta(), LocalDate.now()).getYears();
        // Aplicar descuento del 10% si el cliente tiene más de 5 años de antiguedad
        if (añosDesdeFechaAlta > 5) {
            return (costoProductos + extraFormaPago) * 0.9;
        }
            return costoProductos + extraFormaPago;
        }
}

public class Cliente {
    private LocalDate fechaAlta;
    public LocalDate getFechaAlta() {
        return this.fechaAlta;
    }
}

public class Producto {
    private double precio;
    public double getPrecio() {
        return this.precio;
    }
}
```

Refactoring

1. **Code Smell:** Loops → Línea 16 a 19

a. **Refactoring a aplicar:** Extract Function → Extraje el fragmento y creé el método *getCostoProductos()*

b. **Refactoring a aplicar:** Replace Loop with Pipeline

2. **Code Smell:** Switch Statement → Línea 21 a 27

   a. **Refactoring a aplicar:** Replace Conditional with Polymorphism / Replace Type Code with State/Strategy → Cree la interfaz *IFormaPago* que declara el método abstracto *calcularValor(double costo)*

      i. Crear las clases *FormaPagoEfectivo*, *FormaPagoSeisCuotas*, *FormaPagoDoceCuotas*, y que cada una implemente la interfaz

      ii. Extraer de la sentencia Switch el código que necesita cada método de las clases

      iii. Reemplazar la variable String **formaPago** por una de tipo *IFormaPago*

      iv. Borrar la sentencia Switch

      v. Asignarle a **extraFormaPago** el valor de lo que retorne llamar a *calcularValor()*

      vi. Cambiar las referencias que hagan falta

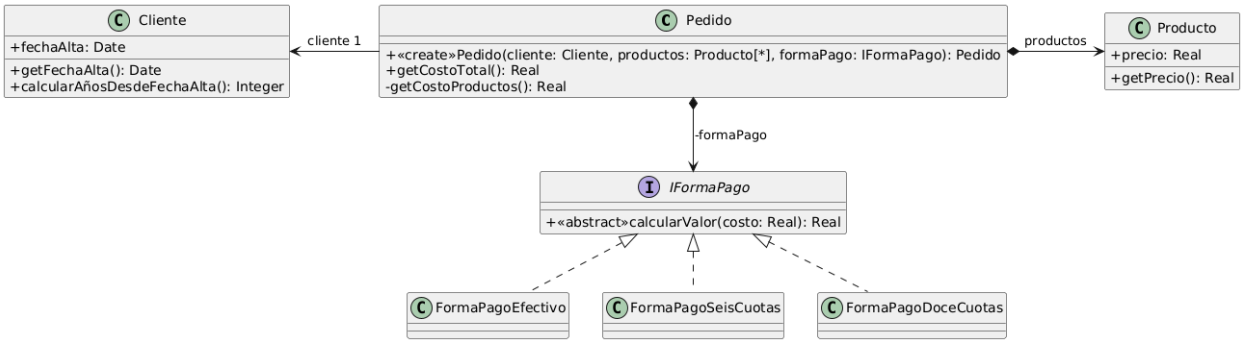
3. **Code Smell:** Envidia de Atributos → Línea 28

   a. **Refactoring a aplicar:** Extract Function → Crear el método *calcularAñosDesdeFechaAlta()*

   b. **Refactoring a aplicar:** Move Function → Moverlo a la clase **Cliente**

4. **Code Smell:** Código Repetido → Línea 28 a 33

Resultado final




```
public class Pedido {
    private Cliente cliente;
    private List<Producto> productos;
    private IFormaPago formaPago;

    public Pedido(Cliente cliente, List<Producto> productos,
        IFormaPago formaPago) {
        if (!"efectivo".equals(formaPago)
            && !"6 cuotas".equals(formaPago)
            && !"12 cuotas".equals(formaPago)) {
                throw new Error("Forma de pago incorrecta");
        }
        this.cliente = cliente;
        this.productos = productos;
        this.formaPago = formaPago;
    }

    public double getCostoTotal() {
        double costoProductos = this.getCostoProductos();
        double extraFormaPago = this.formaPago
            .calcularValor(costoProductos);
        int clienteAñosDesdeFechaAlta = this
            .cliente.calcularAñosDesdeFechaAlta();
        double costoTotal = (costoProductos + extraFormaPago)
            * (clienteAñosDesdeFechaAlta > 5 ? 0.9 : 1);

        return costoTotal;
```

```
    }

    private double getCostoProductos() {
            double suma = this.productos.stream()
                .mapToDouble(p -> p.getPrecio()).sum();
            return suma;
        }
    }

public class Cliente {
    private LocalDate fechaAlta;
    public LocalDate getFechaAlta() {
        return this.fechaAlta;
    }

    int calcularAñosDesdeFechaAlta() {
        return Period.between(this.getFechaAlta(),
            LocalDate.now()).getYears();
    }
}

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

public interface IFormaPago {
    public double calcularValor(double costo);
}

public class FormaPagoEfectivo implements IFormaPago {
    public double calcularValor(double costo) {
        return costo;
    }
```

```
}

public class FormaPagoSeisCuotas implements IFormaPago {
    public double calcularValor(double costo) {
        return costo * 0.2;
    }
}

public class FormaPagoDoceCuotas implements IFormaPago {
    public double calcularValor(double costo) {
        return costo * 0.5;
    }
}
```