# Ejercicio 3

Tareas:

1. Enumere los code smell y que refactorings utilizará para solucionarlos.
2. Aplique los refactorings encontrados, mostrando el código refactorizado luego de aplicar cada uno.
3. Analice el código original y detecte si existe un problema al calcular las estadísticas. Explique cuál es el error y en qué casos se da ¿El error identificado sigue presente luego de realizar los refactorings? En caso de que no esté presente, ¿en qué momento se resolvió? De acuerdo a lo visto en la teoría, ¿podemos considerar esto un refactoring?

```
public class Document {
    List<String> words;

    public long characterCount() {
    long count = this.words
.stream()
.mapToLong(w → w.length())
.sum();
        return count;
    }
    public long calculateAvg() {
        long avgLength = this.words
.stream()
.mapToLong(w → w.length())
.sum() / this.words.size();
    return avgLength;
    }
// Resto del código que no importa
}
```

Solucion 1

Code smell: codigo duplicado.

Refactoring a aplicar:  usar el metodo characterCount() ya creado.

```
public class Document {
    List<String> words;

    public long characterCount() {
    return this.words.stream()
  .mapToLong(w → w.length())
  .sum();
    }
    public long calculateAvg() {
        return this.characterCount() / this.words.size();
    }
// Resto del código que no importa
}
```

Code smell 2: reinventa la rueda.

Refactoring a aplicar: agregar una funcion de mas alto nivel.

```
public class Document {
    List<String> words;

    public long characterCount() {
    return this.words.stream()
  .mapToLong(w → w.length())
  .sum();
    }
    public long calculateAvg() {
        return this.words.stream()
        .mapToLong(w → w.length())
        .average().orElse(0);
    }
// Resto del código que no importa
}
```

El problema del codigo original al calcular las estadisticas, es que si el numero de palabras de la lista era 0, se iba a dar el caso que tenia que dividir por 0 palabras, y esto generaria un error.

Luego de hacer refactoring, usamos una funcion de mas alto nivel, para que nos permita manejar estos casos, sin producir errores. Usando la funcion de stream .average().orElse(0);

Tomando la definicion de refactoring estrictamente, quiza no seria refactoring ya que modifica cierto comportamiento del codigo, ya que ahora no tirar error en ese caso.

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(), Loca

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    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;
  }
}
```

Replace Loop with Pipeline (líneas 16 a 19)

```
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 = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    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(), Loca

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    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;
  }
}
```

Replace Conditional with Polymorphism (líneas 21 a 27)

Creamos la jerarquia

```
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 = this.productos.stream()
        .mapToInt(p -> p.getPrecio())
        .sum();

    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(), Loca

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    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;
  }
}

public Interface FormaPago {
    public Double calcularExtra();
}

public class Efectivo implements FormaPago {

}
```

```
public class SeisCuotas implements FormaPago {

}

public class DoceCuotas implements FormaPago {

}
```

Modificamos el constructor que tenia formapago como String.

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

  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    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(), Loca
```

```
    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    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;
  }
}

public Interface FormaPago {
    public Double calcularExtra();
}

public class Efectivo implements FormaPago {

}

public class SeisCuotas implements FormaPago {

}

public class DoceCuotas implements FormaPago {
```

```
}
```

Modificamos el metodo getCostoTotal() ahora haciendo uso del polimorfismo.

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

  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    double extraFormaPago = this.formaPago.calcularExtra(costoProductos);)

    int añosDesdeFechaAlta = Period.between(this.cliente.getFechaAlta(), Loca

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    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;
  }
}

public Interface FormaPago {
    public Double calcularExtra(double costoProductos);
}

public class Efectivo implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return 0;
    }
}

public class SeisCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.2;
    }
}

public class DoceCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.5;
    }
}
```

Extract method y move method (línea 28)

Creamos el metodo en la clase Cliente.

```
public class Pedido {
  private Cliente cliente;
```

```
  private List<Producto> productos;
  private FormaPago formaPago;

  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    double extraFormaPago = this.formaPago.calcularExtra(costoProductos);)

    int añosDesdeFechaAlta = Period.between(this.cliente.getFechaAlta(), Loca

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    if (añosDesdeFechaAlta > 5) {
      return (costoProductos + extraFormaPago) * 0.9;
    }
    return costoProductos + extraFormaPago;
  }
}

public class Cliente {
  private LocalDate fechaAlta;

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

    public int calcularAñosDesdeFechaAlta() {
        return period.between(this.fechaAlta, LocalDate.now()).getYears();
    }

}
```

```
public class Producto {
  private double precio;

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

public Interface FormaPago {
    public Double calcularExtra(double costoProductos);
}

public class Efectivo implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return 0;
    }
}

public class SeisCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.2;
    }
}

public class DoceCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.5;
    }
}
```

Reemplazamos el metodo original llamando al metodo desde cliente.

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

```
  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    double extraFormaPago = this.formaPago.calcularExtra(costoProductos);)

    int añosDesdeFechaAlta = this.cliente.calcularAñosDesdeFechaAlta();

    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    if (añosDesdeFechaAlta > 5) {
      return (costoProductos + extraFormaPago) * 0.9;
    }
    return costoProductos + extraFormaPago;
  }
}

public class Cliente {
  private LocalDate fechaAlta;

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

    public int calcularAñosDesdeFechaAlta() {
        return period.between(this.fechaAlta, LocalDate.now()).getYears();
    }

}

public class Producto {
  private double precio;
```

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

public Interface FormaPago {
    public Double calcularExtra(double costoProductos);
}

public class Efectivo implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return 0;
    }
}

public class SeisCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.2;
    }
}

public class DoceCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.5;
    }
}
```

Extract method y replace temp with query (líneas 28 a 33)

Hacemos extract method.

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

  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
```
```

```
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double calcularDescuento(double costoProductos, double extraForma
      int añosDesdeFechaAlta = this.cliente.calcularAñosDesdeFechaAlta();
    // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüeda
    if (añosDesdeFechaAlta > 5) {
      return (costoProductos + extraFormaPago) * 0.9;
    }
    return costoProductos + extraFormaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    double extraFormaPago = this.formaPago.calcularExtra(costoProductos);)

    return this.calcularDescuento(costoProducto, extraFormaPago);
  }
}

public class Cliente {
  private LocalDate fechaAlta;

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

    public int calcularAñosDesdeFechaAlta() {
        return period.between(this.fechaAlta, LocalDate.now()).getYears();
    }

}
```

```
public class Producto {
  private double precio;

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

public Interface FormaPago {
    public Double calcularExtra(double costoProductos);
}

public class Efectivo implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return 0;
    }
}

public class SeisCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.2;
    }
}

public class DoceCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.5;
    }
}
```

Aplicamos replace temp with query.

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

  public Pedido(Cliente cliente, List<Producto> productos, FormaPago formaP
```

```
    this.cliente = cliente;
    this.productos = productos;
    this.formaPago = formaPago;
  }

  public double calcularDescuento(double costoProductos, double extraForma
      // Aplicar descuento del 10% si el cliente tiene más de 5 años de antigüed
    if (this.cliente.calcularAñosDesdeFechaAlta()> 5) {
      return (costoProductos + extraFormaPago) * 0.9;
    }
    return costoProductos + extraFormaPago;
  }

  public double getCostoTotal() {
    double costoProductos = this.productos.stream()
        .mapToInt(p → p.getPrecio())
        .sum();

    double extraFormaPago = this.formaPago.calcularExtra(costoProductos);)

    return this.calcularDescuento(costoProducto, extraFormaPago);
  }
}

public class Cliente {
  private LocalDate fechaAlta;

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

    public int calcularAñosDesdeFechaAlta() {
        return period.between(this.fechaAlta, LocalDate.now()).getYears();
    }

}

public class Producto {
```

```
  private double precio;

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

public Interface FormaPago {
    public Double calcularExtra(double costoProductos);
}

public class Efectivo implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return 0;
    }
}

public class SeisCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.2;
    }
}

public class DoceCuotas implements FormaPago {
    public Double calcularExtra(double costoProductos){
        return costoProductos * 0.5;
    }
}
```