Ejercicio 1 Refactoring

Ejercicio 1

1.1

```
/**
* Retorna el límite de crédito del cliente
*/
public double lmtCrdt() {...

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

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

Mal olor: Nombre de métodos poco descriptivos. Refactor a aplicar: **Rename method**

Mal olor: Nombre de parámetros poco descriptivos. Refactor a aplicar: **Rename variable**

```
/**
* Retorna el límite de crédito del cliente
*/
public double limiteCredito() {...

/**
* Retorna el monto facturado al cliente desde la fecha f1 a la fecha f2
*/
protected double montoFacturadoEntreFechas(LocalDate fechaInicio, LocalDa
```

```
/**
* Retorna el monto cobrado al cliente desde la fecha f1 a la fecha f2
*/
private double montoCobradoEntreFechas(LocalDate fechaInicio, LocalDate fe
```

Mal olor: Comentarios, ya son innecesarios porque se aplicó anteriormente el Rename Metod. El siguiente refactor a aplicar es el de eliminar comentarios (**Remove Redundant Comments**).

```
public double limiteCredito() {...

protected double montoFacturadoEntreFechas(LocalDate fechaInicio, LocalDa

private double montoCobradoEntreFechas(LocalDate fechaInicio, LocalDate fe
```

1.2

Lo considero correcto al refactoring aplicado en la Figura 2. Inicialmente se tenía el mal olor **Feature Envy**. Y se aplicó el refactoring **Move Method**. Para mi es apropiado ya que el proyecto es el objeto encargado de manejar sus participantes y de realizar las verificaciones sobre los mismos.

1.3

```
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);
}
```

Mal olor: **Temporary Field**, **Long Method**. Refactor necesario: **Replace Temp With Query**. Para totalEddes y totalSalarios.

```
private double promedioEdades() {
    int totalEdades = 0;
    for (Empleado empleado: personal)
        totalEdades = totalEdades + empleado.getEdad();
    return totalEdades / personal.size();
}

private double totalSalarios() {
    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.promedioEdades(), this.totalSalarios());

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

Mal olor: **Reinventando la rueda**. Refactor necesario: **Replace Loop With Pipeline**. Se utiliza for cuando se debería usar Stream().

```
private double promedioEdades() {
    return this.personal.stream
        .mapToInt(e → e.getEdad())
        .sum() / personal.size();
}
```

```
private double totalSalarios() {
    return this.personal.stream
        .mapToDouble(e → e.getSalario())
        .sum();
}

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

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