Orientación a Objetos II
2025

Explicación de práctica
Semana del 24 de marzo

Refactoring

Refactoring

- Refactorización (sustantivo): cambio realizado en la estructura interna de un programa informático para que sea más fácil de entender y más “barato” de modificar sin cambiar su comportamiento observable.

- Refactorizar (verbo): reestructurar software aplicando una serie de refactorizaciones sin cambiar su comportamiento observable.

Refactoring v1 - Fowler

Refactoring

- Refactoring es una transformación que preserva el comportamiento, pero mejora el diseño

Bill Opdyke, PhD Thesis "Refactoring Object-Oriented Frameworks". Univ. of Illinois at Urbana-Champaign (UIUC). 1992. Director: Ralph Johnson.

Refactoring

- Refactorizar NO es tirar el código y escribirlo de nuevo
- Se debe preservar el comportamiento
- Refactorizar es seguir un método para mejorarlo:

1. Identificar code smells
2. Determinar cómo mejorarlo
3. Aplicar la mejora

Ejercicio 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) {...
```

Ejercicio 1.2

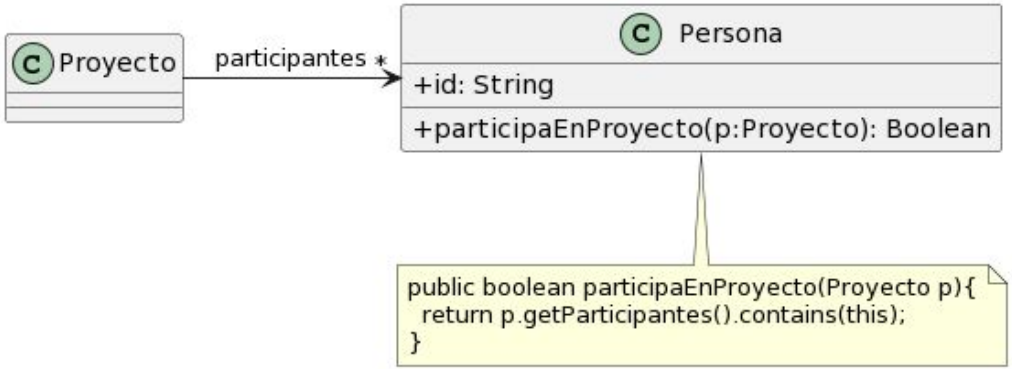
Un diagrama de clases UML con el diseño inicial de la solución provista

La secuencia de refactorings aplicados, documentados cada uno de la siguiente manera:

- Mal olor detectado en el código
- Extracto del código que presenta el mal olor
- Refactoring a aplicar que resuelve el mal olor
- Código con el refactoring aplicado

Un diagrama de clases UML con el diseño final

El código java refactorizado



Ejercicio 1.2

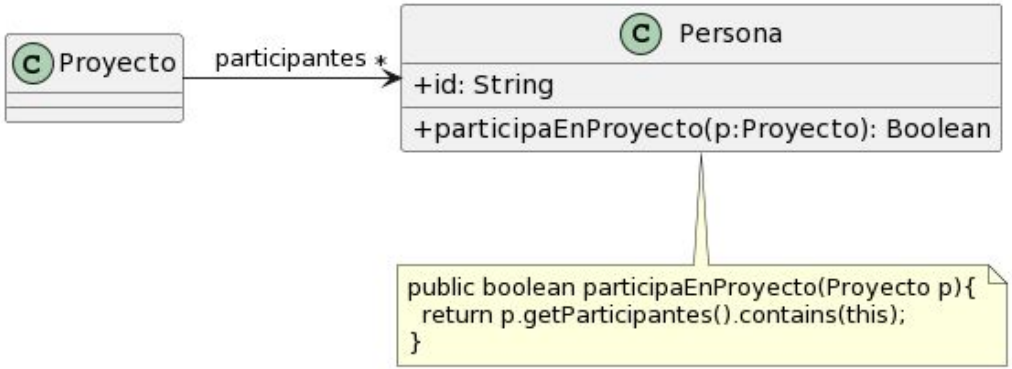
Un diagrama de clases UML con el diseño inicial de la solución provista

La secuencia de refactorings aplicados, documentados cada uno de la siguiente manera:

- **Mal olor detectado en el código**
- Extracto del código que presenta el mal olor
- Refactoring a aplicar que resuelve el mal olor
- Código con el refactoring aplicado

Un diagrama de clases UML con el diseño final

El código java refactorizado



Bad smell: feature envy

Ejercicio 1.2

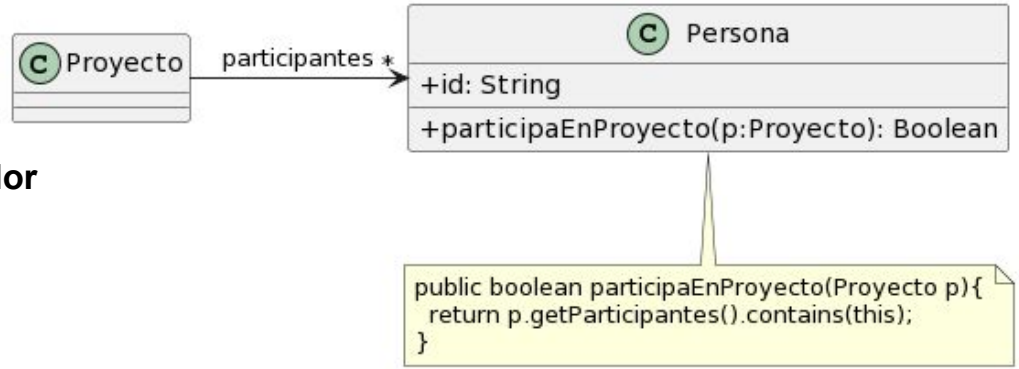
Un diagrama de clases UML con el diseño inicial de la solución provista

La secuencia de refactorings aplicados, documentados cada uno de la siguiente manera:

- Mal olor detectado en el código
- Extracto del código que presenta el mal olor
- **Refactoring a aplicar que resuelve el mal olor**
- Código con el refactoring aplicado

Un diagrama de clases UML con el diseño final

El código java refactorizado



Refactoring a aplicar: move method

Ejercicio 1.2

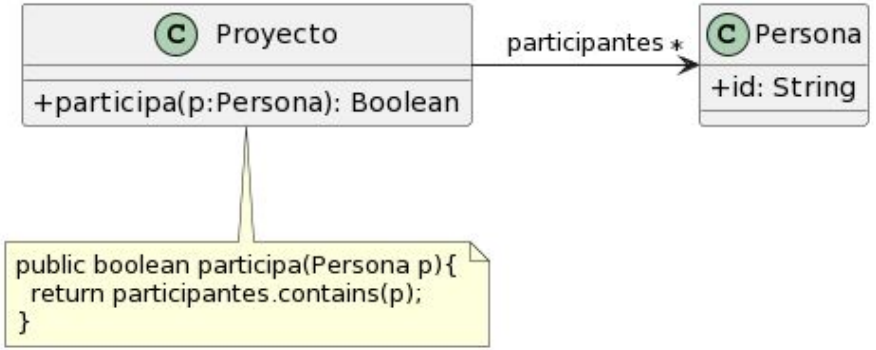
Un diagrama de clases UML con el diseño inicial de la solución provista

La secuencia de refactorings aplicados, documentados cada uno de la siguiente manera:

- Mal olor detectado en el código
- Extracto del código que presenta el mal olor
- Refactoring a aplicar que resuelve el mal olor
- **Código con el refactoring aplicado**

Un diagrama de clases UML con el diseño final

El código java refactorizado



Ejercicio 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 detectado en el código

-Extracto del código que presenta el mal olor

-Refactoring a aplicar que resuelve el mal olor

-Código con el refactoring aplicado

Ejercicio 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 edade
totalSalarios);

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

-Mal olor detectado en el código

-Extracto del código que presenta el mal olor

-Refactoring a aplicar que resuelve el mal olor

-Código con el refactoring aplicado

Bad smell: long method
reinventa la rueda
temporary field

Comenzamos por uno: long method
Refactoring a aplicar:
Replace temp with query
Extract method

Ejercicio 1.3

```
public void imprimirValores() {
        double promedioEdades = this.calcularPromedioEdades();
        double totalSalarios = 0;

        for (Empleado empleado : personal) {
                totalSalarios = totalSalarios + empleado.getSalario();
        }
totalS

}
```

```
public double calcularPromedioEdades() {
        int totalEdades = 0;
        double promedioEdades = 0;

        for (Empleado empleado : personal) {
                totalSalarios = totalSalarios + empleado.getEdad();
        }

        promedioEdades = totalEdades / personal.size();

        return promedioEdades;
}
```

-Mal olor detectado en el código

-Extracto del código que presenta el mal olor

-Refactoring a aplicar que resuelve el mal olor

-Código con el refactoring aplicado

Bad smell: long method
reinventa la rueda
temporary field

menzamos por uno: long method
factoring a aplicar:
Replace temp with query
Extract method

Ejercicio 1.3

```
public double calcularPromedioEdades() {
        return personal.stream()
                .mapToInt(e -> e.getEdad())
                .average();
}
```

-Mal olor detectado en el código

-Extracto del código que presenta el mal olor

-Refactoring a aplicar que resuelve el mal olor

-Código con el refactoring aplicado

Bad smell: reinventa la rueda

Comenzamos por uno:

Refactoring a aplicar:

Replace loop with pipeline

Ejercicio 1.3

Replace Loop with Pipeline

```
const names = [];
for (const i of input) {
  if (i.job === “programmer”)
    names.push(i.name);
}
```

```
const names = input
  .filter(i => i.job === “programmer”)
  .map(i => i.name)
;
```

Ejercicio 1.3

```
public void imprimirValores() {
        double promedioEdades = this.calcularPromedioEdades();
        double totalSalarios = 0;

        for (Empleado empleado : personal) {
                totalSalarios = totalSalarios + empleado.getSalario();
        }

        String message = String.format("El promedio de las edade
totalSalarios);

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

-Mal olor detectado en el código

-Extracto del código que presenta el mal olor

-Refactoring a aplicar que resuelve el mal olor

-Código con el refactoring aplicado

Bad smell: long method
reinventa la rueda
temporary field

Comenzamos por uno: long method

Refactoring a aplicar:
Replace temp with query
Extract method