escuela técnica superior
de ingeniería informática

Tema 4
Refactorizaciones

Departamento de
Lenguajes y Sistemas Informáticos

Ingeniería del Software
de Gestión II

Objetivos generales

Capacidades específicas

- Entender la refactorización como actividad fundamental en el desarrollo del software.
- Distinguir los “malos olores” en el software.
- Conocer los mecanismos de refactorización más frecuentes.

Capacidades transversales

- Manejar documentación externa.
- Aprendizaje bajo demanda.

Guión

- Contexto
- Malos olores
- Catálogo
- Resumen
- Caso de estudio

Guión

- **Contexto**
- Malos olores
- Catálogo
- Resumen
- Caso de estudio

Contexto

Definición

Cambios que se realizan a la estructura interna del software para facilitar la comprensión y abaratar el mantenimiento sin cambiar su comportamiento observable

Contexto

¿Por qué refactorizar?

- + legibilidad
- + mantenibilidad
- Ayuda a encontrar errores
- + productividad

Contexto

¿Cuándo refactorizar?

- Añadir funcionalidad

  (prueba si el diseño es bueno o malo)
- Arreglar un error

  (suelen generarse por código ofuscado)
- Durante una revisión de código

  (el código desde otros ojos, XP)

Contexto

¡Cuidado con...!

- Migración de bases de datos

  (los datos existentes pueden ser vitales)
- Interface públicas

  (no lleves el cambio a los usuarios)

Contexto

¿Cuándo NO refactorizar?

- Código no funciona

  (considerar el desarrollo desde cero)
- Cerca de un *deadline*

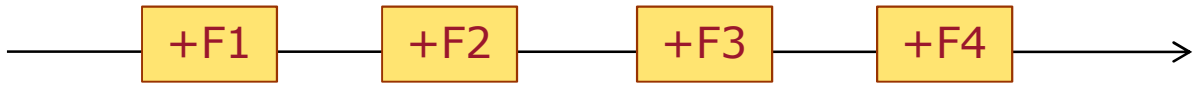
  (los beneficios vendrán tras el deadline)

Contexto

La vida del desarrollador

- Sin refactorizar

- Refactorizando



Guión

- Contexto
- **Malos olores**
- Catálogo
- Resumen
- Caso de estudio

Malos olores

Código duplicado

Mal olor: Mismo código en más de un sitio

Refactorizaciones frecuentes: Extract Method, Pull Up Method, Form Template Method, Substitute Algorithm, Extract Class.

Métodos extensos

Mal olor: métodos muy largos (regla de una página)

Refactorizaciones frecuentes: Extract Method, Replace Temp with Query, Introduce Parameter Object, Preserve Whole Object, Replace Method with Method Object, Decompose Conditional.

Malos olores

Clase grande

Mal olor: clase intenta hacer muchas cosas(poca cohesión)

Refactorizaciones frecuentes: Extract Class, Extract Subclass, Duplicate Observed Data.

Lista de parámetros grande

Mal olor: muchos parámetros en un método

Refactorizaciones frecuentes: Replace Parameter with Method, Preserve Whole Object, Introduce Parameter Object.

Malos olores

Cambios divergentes

Mal olor: varios cambios afectan a una única parte del código

Refactorizaciones frecuentes: Extract Class.

Cirugía de perdigones

Mal olor: 1 cambio afecta a muchas partes

Refactorizaciones frecuentes: Move Method, Move Field, Inline Class.

Malos olores

Envidia

Mal olor: un método se interesa más por los métodos de otras clases

Refactorizaciones frecuentes: Move Method, Extract Method.

Pandillas de datos

Mal olor: conjunto de parámetros que siempre van juntos

Refactorizaciones frecuentes: Extract Class, Introduce Parameter Object, Preserve Whole Object.

Malos olores

Obsesión primitiva

Mal olor: uso de int en lugar de Integer; String en lugar de Phone, etc.

Refactorizaciones frecuentes: Replace Data Value with Object, Replace Type Code with Class, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Introduce Parameter Object, Replace Array with Object.

Sentencias *switch*

Mal olor: uso de switch en objetos

Refactorizaciones frecuentes: Extract Method, Move Method, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Replace Conditional with Polymorphism, Replace Parameter with Explicit Methods, Introduce Null Object.

Malos olores

Jerarquías de herencia paralelas

Mal olor: dos o más jerarquías tienen la misma estructura.

Refactorizaciones frecuentes: Move Method, Move Field.

Clases perezosas

Mal olor: clases que no hacen casi nada (poltergeist)

Refactorizaciones frecuentes: Collapse Hierarchy, Inline Class.

Malos olores

Generalidad especulativa

Mal olor: exceso de previsión

Refactorizaciones frecuentes: Collapse Hierarchy, Inline Class, Remove Parameter, Rename Method

Campos temporales

Mal olor: atributos de clases que se usan para casos excepcionales.

Refactorizaciones frecuentes: Extract Class, Introduce Null Object.

Malos olores

Cadenas de Mensajes

Mal olor: clase que pide objetos a otra clase, y usa este objeto para solicitar otro objeto a otra clase, etc. (acoplamiento a la estructura de navegación)

Refactorizaciones frecuentes: Hide Delegate, Extract Method, Move Method.

El intermediario

Mal olor: una clase intermediario no da valor añadido.

Refactorizaciones frecuentes: Remove Middle Man, Inline Method, Replace Delegation with Inheritance

Malos olores

Acoso a la intimidad

Mal olor: clases que acceden a los entresijos de otras clases.

Refactorizaciones frecuentes: Move Method, Move Field, Change Bidirectional Association to Unidirectional, Extract Class, Hide Delegate, Replace Inheritance with Delegation.

Clases alternativas con distintas interfaces

Mal olor: 2+ clases intercambiables no implementan la misma interfaz

Refactorizaciones frecuentes: Rename Method, Move Method, Extract Superclass.

Malos olores

Bibliotecas de clases incompletas

Mal olor: una biblioteca no hace todo lo que queremos.

Refactorizaciones frecuentes: Move Method, Introduce Foreign Method, Introduce Local Extension.

Clases Dato

Mal olor: getters/setters y nada más.

Refactorizaciones frecuentes: Encapsulate Field, Encapsulate Collection, Remove Setting Method, Move Method, Extract Method, Hide Method.

Malos olores

Legados rechazados

Mal olor: una subclase no quiere parte de la herencia.

Refactorizaciones frecuentes: Push Down Method, Push Down Field, Replace Inheritance with Delegation.

Comentarios desodorantes

Mal olor: los comentarios huelen bien, el código de abajo no.

Refactorizaciones frecuentes: Extract Method, Rename Method, Introduce Assertion.

Guión

- Contexto
- Malos olores
- **Catálogo**
- Resumen
- Caso de estudio

Catálogo

Tipos de refactorizaciones

- Composición de métodos
- Desplazamiento de funcionalidad entre métodos
- Organización de datos
- Simplificación de expresiones condicionales
- Simplificación de llamadas a métodos
- Generalización

Catálogo: Composición de Métodos

Extract Method

Fragmento de código que puede agruparse

Convertir el fragmento en un método con nombre autoexplicativo

```
void printOwing() {
    printBanner();

    //print details
    System.out.println ("name: " + _name);
    System.out.println ("amount " + getOutstanding());
}
```

```
void printOwing() {
    printBanner();
    printDetails(getOutstanding());
}

void printDetails (double outstanding) {
    System.out.println ("name: " + _name);
    System.out.println ("amount " + outstanding);
}
```

Catálogo: Composición de Métodos

Inline Method

El código es tan claro como el nombre del método

Colocar el cuerpo del método en sus llamadas y eliminar el método

```
int getRating() {
      return (moreThanFiveLateDeliveries()) ? 2 : 1;
}

boolean moreThanFiveLateDeliveries() {
      return _numberOfLateDeliveries > 5;
}
```

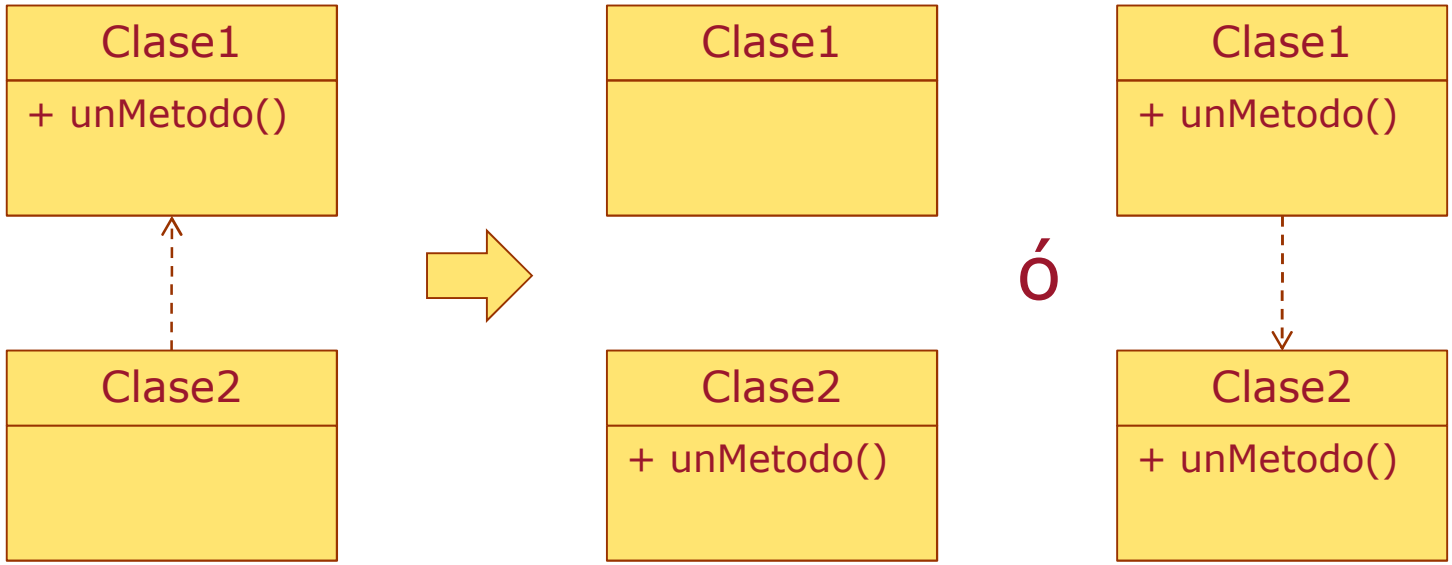
```
int getRating() {
      return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Catálogo: Desplazamiento de funcionalidad entre objetos

Move Method

Un método se usa o usará más por otras clases que por su clase

Crear un método de cuerpo similar en la clase que más lo usa.
Eliminar o convertir en una delegación el antiguo método

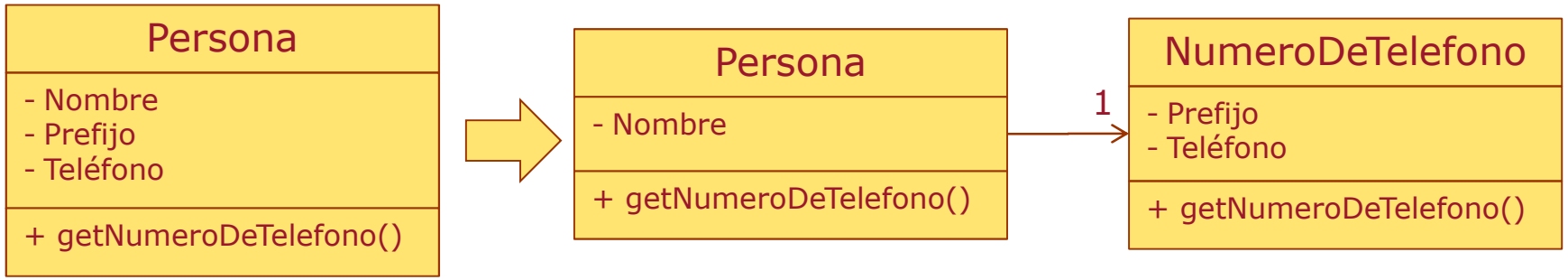


Catálogo: Desplazamiento de funcionalidad entre objetos

Extract Class

Una sóla clase hace el trabajo de 2

Crear una clase y mover los atributos y métodos relevantes

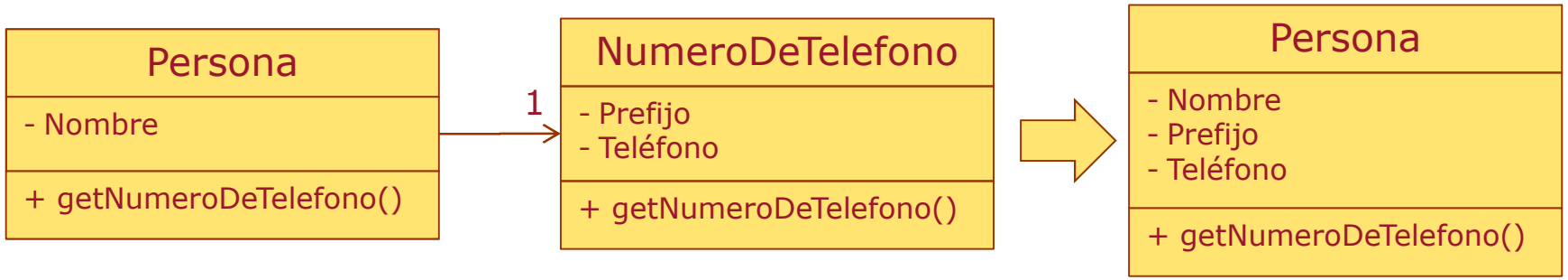


Catálogo: Desplazamiento de funcionalidad entre objetos

Inline Class

Una clase no hace gran cosa

Moverla a otra clase y borrarla



Catálogo: Organización de datos

Encapsulate Field

Existe un atributo público

Hacerlo privado y crear sus accesors

```
public String nombre;
```

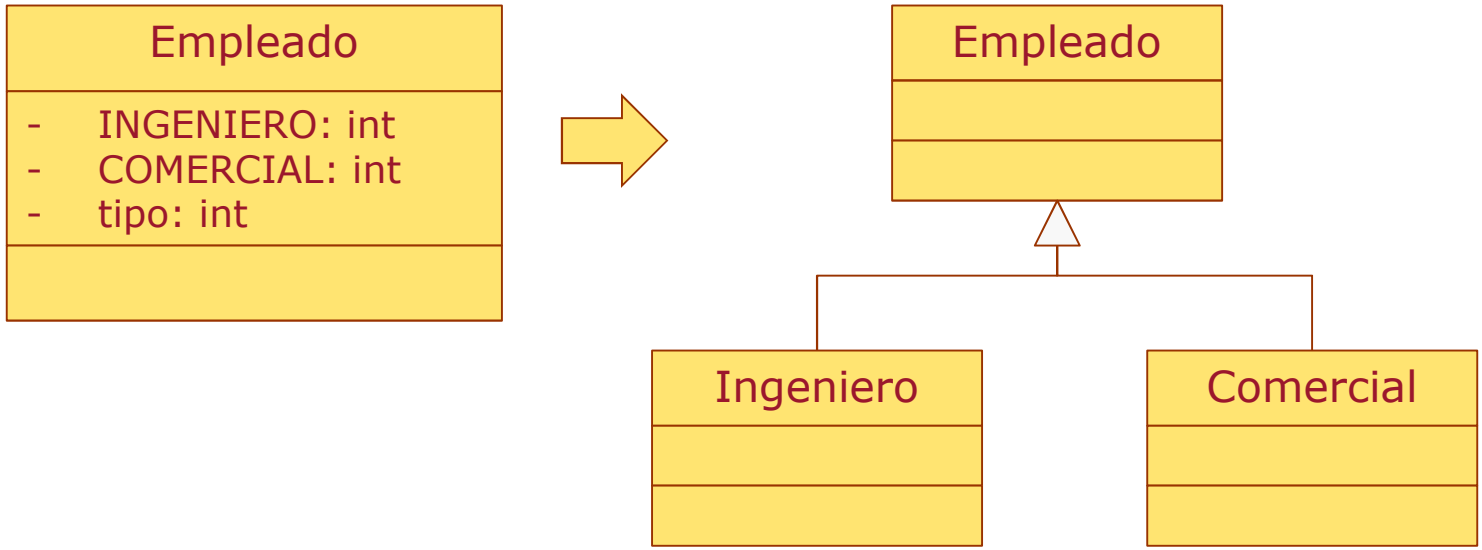
```
private String nombre;

public String getName() {
    return name;
}

public void setName(String arg) {
    name = arg;
}
```

Catálogo: Organizar datos

Replace Type Code with Subclasses

Un atributo "tipo" afecta al comportamiento de una clase

Reemplazar el tipo por subclases



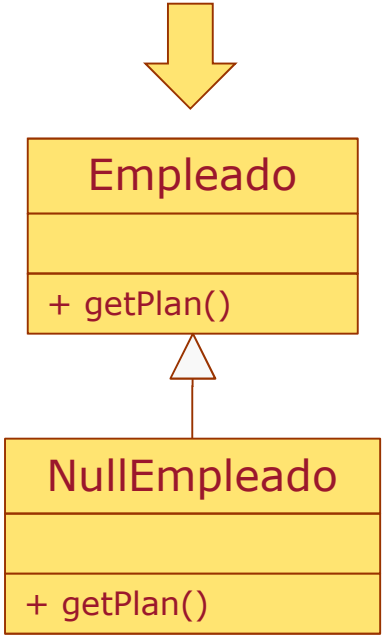
Catálogo: Simplificación de expresiones condicionales

Introduce Null Object

Comprobaciones reiteradas de valor *null*

Reemplazar el valor null *por un objeto* null

```
if (customer == null)
    plan = BillingPlan.basic();
else
    plan = customer.getPlan();
```

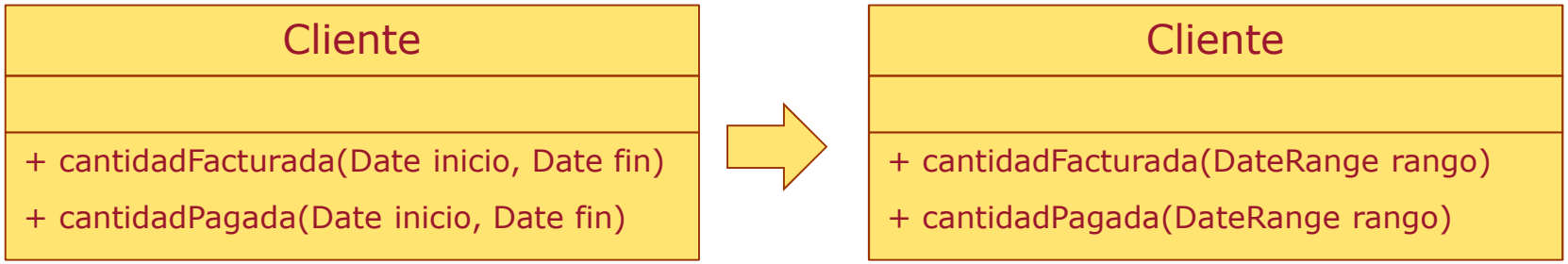


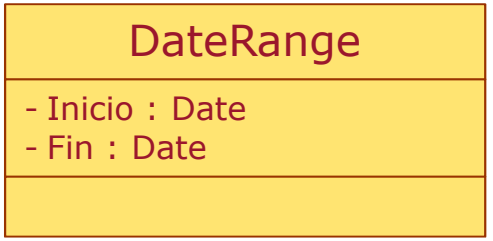
Catálogo: Simplificación de llamadas a métodos

Introduce Parameter Object

Un conjunto de parámetros suele ir junto de forma natural

Reemplazar al conjunto de parámetros por un objeto.



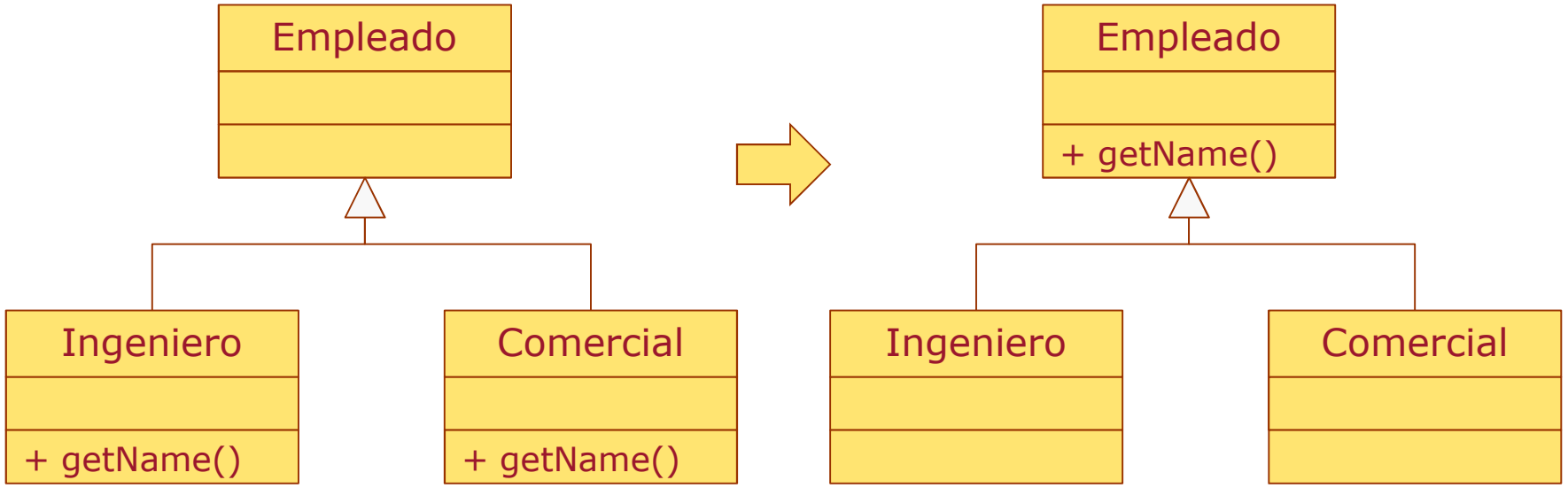


Catálogo: Generalización

Pull up Method

Métodos con idénticos resultados en subclases

Mover los métodos a la superclase

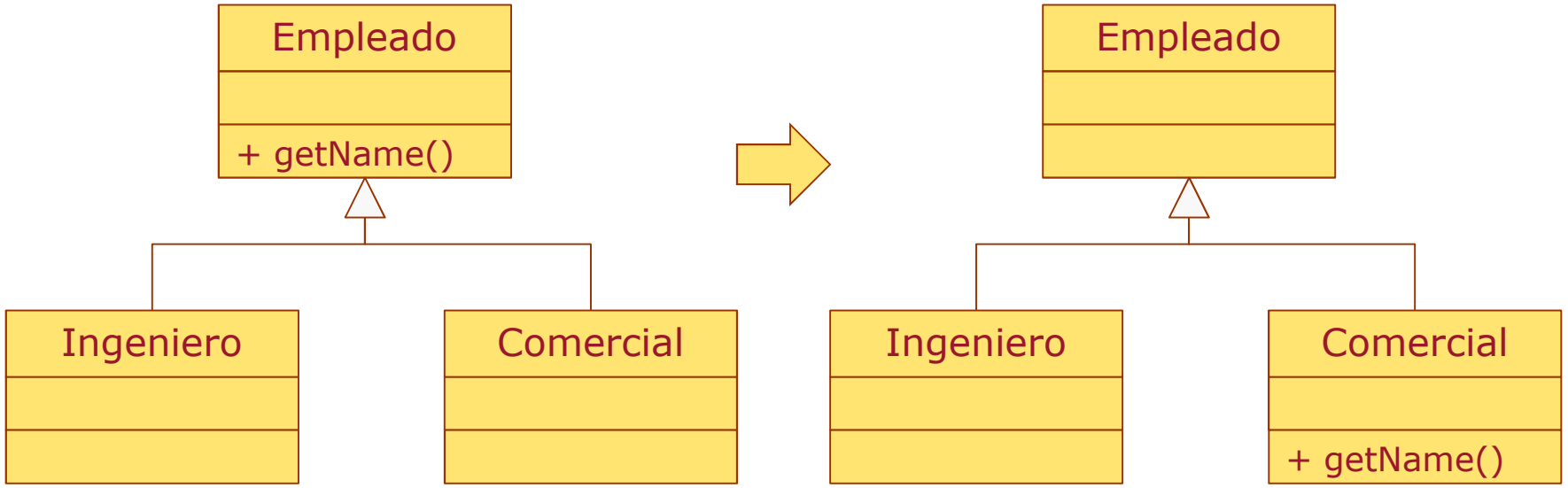


Catálogo: Generalización

Push down Method

El comportamiento en una superclase es irrelevante para algunas subclases

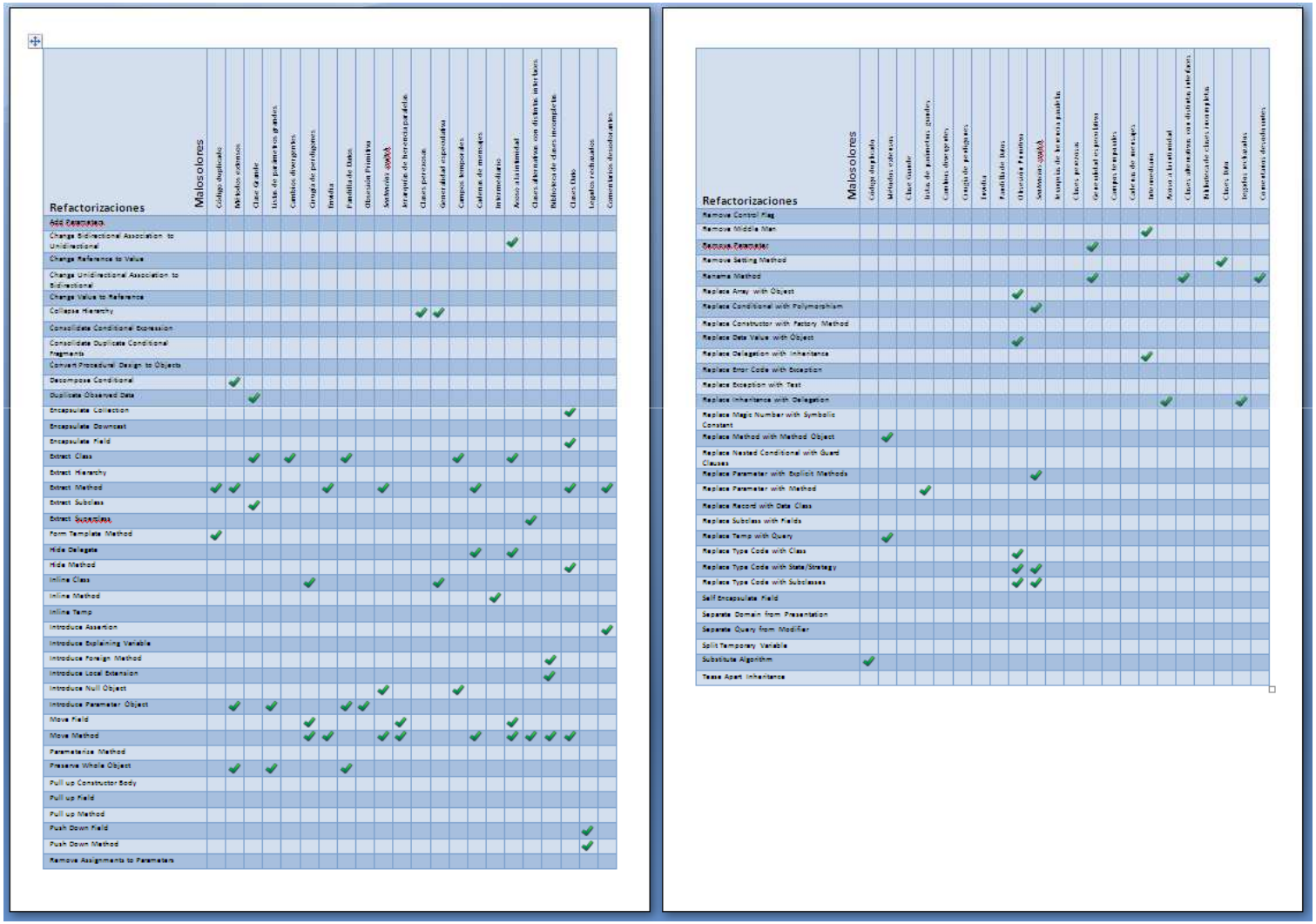
Mover los métodos a las subclases para las que es relevante



Guión

- Contexto
- Malos olores
- Catálogo
- **Resumen**
- Caso de estudio

Resumen


Refactorizaciones
Malosolores
Código duplicado
Métodos extensos
Clase Grande
Lista de parámetros grandes
Cambios divergentes
Cirugía de perdigones
Envidia
Familia de Datos
Obsesión Primitiva
Sentencias switch
Jerarquía de herencia paralela
Clases perezosas
Generalidad especulativa
Campos temporales
Cadena de mensajes
Intermediario
Acoso a la intimidad
Clases Alternativas con distintas interfaces
Biblioteca de clases incompleta
Clases Data
Legados rechazados
Comentarios desodorantes
Add Parameter
Change Bidirectional Association to Unidirectional
Change Reference to Value
Change Unidirectional Association to Bidirectional
Change Value to Reference
Collapse Hierarchy
Consolidate Conditional Expression
Consolidate Duplicate Conditional Fragments
Convert Procedural Design to Objects
Decompose Conditional
Duplicate Observed Data
Encapsulate Collection
Encapsulate Downcast
Encapsulate Field
Extract Class
Extract Hierarchy
Extract Method
Extract Subclass
Extract Superclass
Form Template Method
Hide Delegate
Hide Method
Inline Class
Inline Method
Inline Temp
Introduce Assertion
Introduce Explaining Variable
Introduce Foreign Method
Introduce Local Extension
Introduce Null Object
Introduce Parameter Object
Move Field
Move Method
Parameterize Method
Preserve Whole Object
Pull up Constructor Body
Pull up Field
Pull up Method
Push Down Field
Push Down Method
Remove Assignments to Parameters
Refactorizaciones
Malosolores
Código duplicado
Métodos extensos
Clase Grande
Lista de parámetros grandes
Cambios divergentes
Cirugía de perdigones
Envidia
Familia de Datos
Obsesión Primitiva
Sentencias switch
Jerarquía de herencia paralela
Clases perezosas
Generalidad especulativa
Campos temporales
Cadena de mensajes
Intermediario
Acoso a la intimidad
Clases Alternativas con distintas interfaces
Biblioteca de clases incompleta
Clases Data
Legados rechazados
Comentarios desodorantes
Remove Control Flag
Remove Middle Man
Remove Parameter
Remove Setting Method
Rename Method
Replace Array with Object
Replace Conditional with Polymorphism
Replace Constructor with Factory Method
Replace Data Value with Object
Replace Delegation with Inheritance
Replace Error Code with Exception
Replace Exception with Test
Replace Inheritance with Delegation
Replace Magic Number with Symbolic Constant
Replace Method with Method Object
Replace Nested Conditional with Guard Clauses
Replace Parameter with Explicit Methods
Replace Parameter with Method
Replace Record with Data Class
Replace Subclass with Fields
Replace Temp with Query
Replace Type Code with Class
Replace Type Code with State/Strategy
Replace Type Code with Subclasses
Self Encapsulate Field
Separate Domain from Presentation
Separate Query from Modifier
Split Temporary Variable
Substitute Algorithm
Tease Apart Inheritance

Guión

- Contexto
- Malos olores
- Catálogo
- Resumen
- **Caso de estudio**

Caso de estudio

En prácticas aplicaremos estos conceptos aun caso de estudio real.

Bibliografía

Fowler, Martin

Refactoring, improving the design of existing code

Addison Wesley, 1999.

www.refactoring.com

¡Gracias!

- ¿Podemos mejorar esta lección?
  - Mándanos un email a ptrinidad en us punto es
  - Visite la web de la asignatura en Enseñanza Virtual