Refactoring

Carlos Aguirre
Baltazar Ochagavía
Pedro Saratscheff

Qué es?

El **refactoring** es el proceso de reestructurar código existente sin cambiar el comportamiento externo de este. El fin de estos cambios es mejorar los atributos no funcionales del software.

Ventajas

Las principales ventajas son aumentar la **entendibilidad** y disminuir la **complejidad**. Con esto se mejora la **mantenibilidad** y permite una arquitectura para mejorar **extensibilidad**

Consejos

- Hacer solo un paso a la Vez
- Mantener test automáticos y correrlos antes de hacer cualquier refactoring, sin importar lo chico del cambio. Estos pueden ser test unitarios, de integración o funcionales.
- Usar herramientas en el ambiente de desarrollo si existiesen
- Programar de la mejor manera
- Intentar que el refactoring sea hecho por la misma persona quien escribió el código o alguien que lo acompañó.

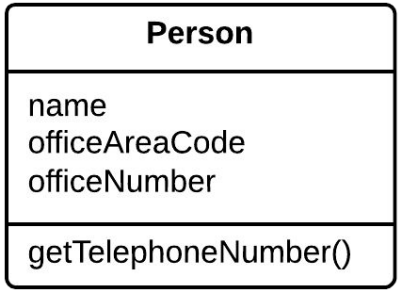
Técnicas de Refactoring

Mover Funcionalidades Entre Componentes

“

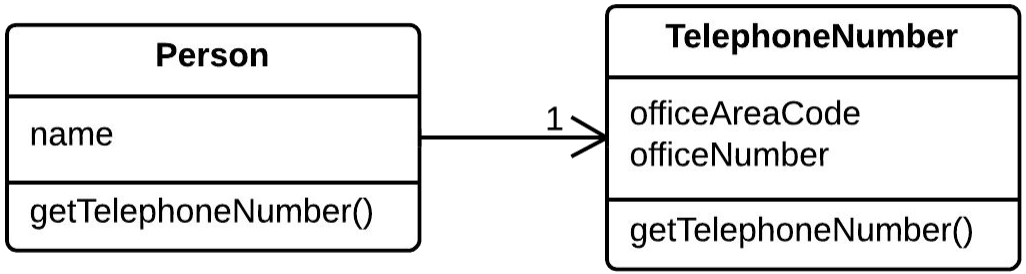
Muestran cómo mover y manejar clases e implementaciones sin alterar comportamiento

Extract Class



P: Una clase hace el trabajo de dos

S: Crear nueva clase y agregarle sus campos propios



Otros

- Mover Campos
- Inline Class
- Hide Delegate
- Remove Middle Man

Técnicas de Refactoring

Organizar Datos

Muestra el beneficio de usar clases y objetos en vez de tipos primitivos. Agrega **portabilidad**

Reemplazar Arreglo Por Objeto

```java
String[] row = new String[2];
row[0] = "Liverpool";
row[1] = "15";
```

```java
Performance row = new Performance();
row.setName("Liverpool");
row.setWins("15");
```

P: Un arreglo contiene varios tipos de datos

S: Reemplazar el arreglo por un objeto con esos atributos

B: Asociar comportamiento a clases y documentación

Otros

- Cambiar Valor Por Referencia
- Cambiar Asociación Unidireccional por Bidireccional
- Reemplazar Subclase por Campos en la clase Base

Técnicas de Refactoring

Simplificación de expresiones condicionales

“

A lo largo del tiempo las expresiones lógicas tienden a ponerse cada vez más complejas y las técnicas son para poder combatir esto.

Decompose Conditional

```
if (date.before(SUMMER_START) || date.after(SUMMER_END)) {
  charge = quantity * winterRate + winterServiceCharge;
}
else {
  charge = quantity * summerRate;
}
```

```
if (notSummer(date)) {
  charge = winterCharge(quantity);
}
else {
  charge = summerCharge(quantity);
}
```

P: La condición es compleja y resulta fácil olvidar lo relevante

S: Extraer la condición a un método separado con un nombre descriptivo

B: Facilita la mantención y evita confusiones

Otros

- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
- Introduce Null Object
- Introduce Assertion

Técnicas de Refactoring

Simplificación de llamadas a métodos

“

Estas técnicas hacen las llamadas más simples y fáciles de entender. Esto a su vez, simplifica las interfaces entre clases.

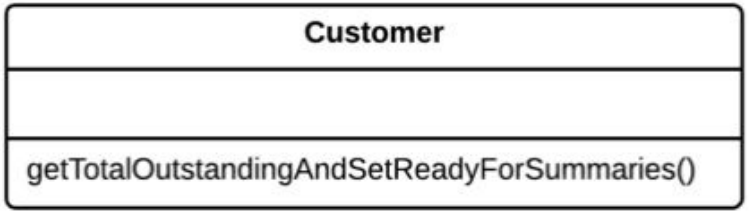
Rename Method

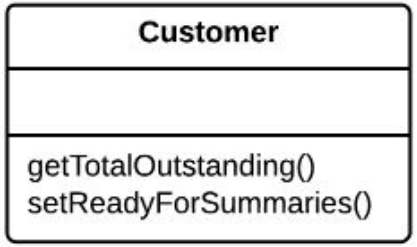
P: El nombre del método no describe lo que realiza

S: Cambiar nombre por uno descriptivo

B: Facilita la mantención y evita confusiones

Separate Query from Modifier





P: Método retorna un valor y además modifica el objeto

S: Separar funcionalidad en dos métodos

B: El contar con un método que solo retorna el valor, se puede llamar cuantas veces se desee sin preocuparse por modificar el objeto

Otros

- Add Parameter
- Remove Parameter
- Parameterize Method
- Replace Parameter with Explicit Methods
- Preserve Whole Object
- Replace Parameter with Method Call
- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test

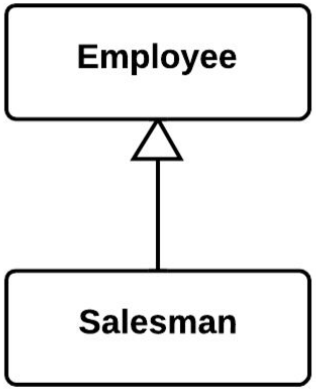
Técnicas de Refactoring

Tratando la generalización

“

Asociado principalmente con mover funcionalidades a lo largo de la jerarquía de herencia de clase, la creación de nuevas clases e interfaces, y la sustitución de la herencia con la delegación y viceversa.

Collapse Hierarchy

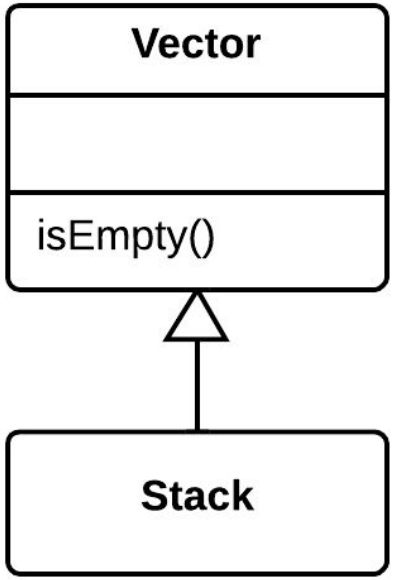


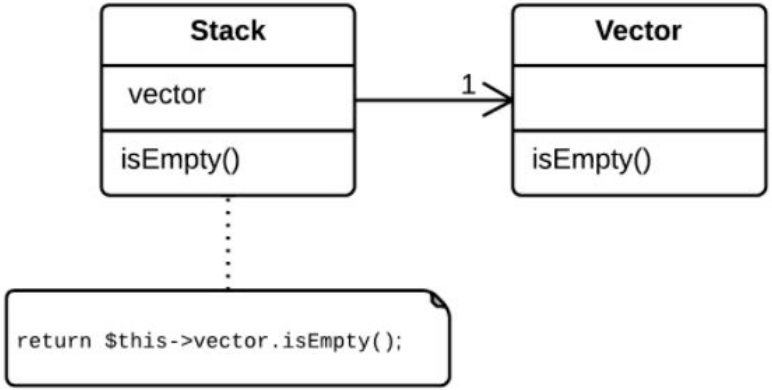
P: Una clase que es parecida a otra

S: Juntar ambas clases en una

B: La complejidad del código se ve disminuida

Replace Inheritance with Delegation





P: Subclase que usa una porción de métodos de su superclase

S: Crear un atributo en vez de heredar

B: Una clase no contiene métodos innecesarios.

Otros

- Pull Up Field
- Pull Up Method
- Pull Up Constructor Body
- Push Down Method
- Push Down Field
- Extract Subclass
- Extract Superclass
- Extract Interface
- Collapse Hierarchy
- Form Template Method
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance

Tools for Refactoring

1. Code Climate
2. Team City
3. Clone Doctor
4. Checkstyle
5. Designite
6. PMD

Referencias utilizadas

1. https://sourcemaking.com/refactoring/refactorings
2. https://en.wikipedia.org/wiki/Code_refactoring#List_of_refactoring_techniques
3. http://www.ceiba.com.co/es/conociendo-el-termino-refactoring/
4. https://blog.inf.ed.ac.uk/sapm/2014/03/14/refactoring-good-practices/