TOP 10 Refactorings

para maximizar el ROI

1 RENAME

El nombre del método, variable, constante o clase **no explica lo que es/hace.**
Nombrar lo que es/hace cada elemento ayuda a un código más semántico.

```
public Cage(double w, double h) {
    this.width = w;
    this.height = h;
}
```

```
public Cage(double width, double height) {
    this.width = width;
    this.height = height;
}
```

Shortcut JetBrains IDEs

Shift + F6

2 INTRODUCE EXPLAINING VARIABLE

Las condiciones no son del todo claras a simple vista.
Extrayéndolas a una variable se puede dar un mejor nombre indicando lo que se pretende evaluar.

```
//Grupo sin búsqueda
if (params.group == null || params.group.isEmpty()) {
    ...
}
```

```
boolean isGroupWithoutSearch = params.group == null || params.group.isEmpty();
if (isGroupWithoutSearch) {
    ...
}
```

Shortcut JetBrains IDEs

⌘ + ⌥ + V
Ctrl + Alt + V

3 EXTRACT METHOD

El método es muy largo y hace muchas cosas. Gran parte de esta lógica se puede dividir en métodos independientes.

```
private void createUserWithFriend(UserDTO userToCreate,
                                  String friendId) {
    validate(userToCreate);
    User user = create(userToCreate);

    // Associate user with a friend 👀
    User friend = find(friendId);
    if (!friend) {
        throw new FriendAssignException(friendId, user);
    }
    user.makeFriends(friend);
    save(user, friend);
}
```

```
private void createUserWithFriend(UserDTO userToCreate,
                                  String friendId) {
    validate(userToCreate);
    User user = create(userToCreate);
    makeFriends(friendId, user);
}

private void makeFriends(String friendId, User user) {
    User friend = find(friendId);
    if (!friend) {
        throw new FriendAssignException(friendId, user);
    }
    user.makeFriends(friend);
    save(user, friend);
}
```

Shortcut JetBrains IDEs

⌘ + ⌥ + M
Ctrl + Alt + M

4 INLINE METHOD

Para eliminar abstracciones inconsistentes o incoherentes. Se pueden detectar porque los nombres de los métodos resultan extraños, confusos, demasiado genéricos o demasiado concretos.
Sirve también para unificar varios bloques de código y volver a hacer *extract method* con una agrupación

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

public boolean moreThanFiveLateDeliveries(Driver driver) {
  return driver.numberOfLateDeliveries > 5;
}
```

```
public int getRating(Driver driver) {
  return (driver.numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Shortcut JetBrains IDEs

⌘ + ⌥ + N
Ctrl + Alt + N

5 MOVE METHOD

Al extraer en métodos más pequeños para ir reestructurando el código seguramente se detecten algunos **métodos que deberían pertenecer a otra clase**. Para llevarlo al sitio que le corresponde se aplica este refactor.

Shortcut JetBrains IDEs

F6
F6

leanmind.es @yodralopez @leanfulness_es Yodra López Carlos Blé

REPLACE NESTED CONDICIONAL WITH GUARD CLAUSES

6

Un listado de condiciones anidadas dificulta seguir el flujo del código. Aislar los casos extremos y ponerlos al principio del método permitirá tener un listado de condiciones que en caso de cumplirse rompen el flujo de este, dejando de evaluar el resto de condiciones.

```
public double getPayAmount() {
  double result;

  if (!isDead) {
    if (isSeparated) {
      result = separatedAmount();
    } else {
      if (isRetired) {
        result = retiredAmount();
      } else {
        result = normalPayAmount();
      }
        }
  } else {
    result = deadAmount();
  }

  return result;
}
```

```
public double getPayAmount() {
  if (isDead) {
    return deadAmount();
  }

  if (isSeparated) {
    return separatedAmount();
  }

  if (isRetired) {
    return retiredAmount();
  }

  return normalPayAmount();
}
```

Shortcut JetBrains IDEs

🍏⌥ + ⏎
Alt + ⏎

✏️ Ejecutar el shortcut sobre la palabra ***if*** o ***else*** para que el IDE sugiera las diferentes opciones:
Invert 'if' condition
Remove redundant 'else' keyword

DECOMPOSE CONDITIONAL

7

Muchas condiciones anidadas con '&&' y '||' complican la lectura y continuación del flujo. *Descomponer* estas condiciones extrayendo a nuevos métodos aporta encapsulación y añade semántica. Esto también aplica a los ***then.***

```
if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd)) {
  charge = quantity * plan.summerRate;
} else {
  charge = quantity * plan.regularRate + plan.regularServiceCharge;
}
```

```
if (summer()) {
  charge = summerCharge();
} else {
  charge = regularCharge();
}
```

```
public boolean summer() {...}
public int summerCharge() {...}
public int regularCharge() {...}
```

Shortcut JetBrains IDEs

🍏⌥ + ⌘ + M
Ctrl + Alt + M

INTRODUCE PARAMETER OBJECT

8

Este refactor se puede aplicar cuando un método tiene varios parámetros que siempre viajan juntos, lo cual puede indicar que deberían formar un nuevo objeto. Este objeto a menudo es de tipo *Value Object.*

```
Circle makeCircle(double x, double y, double radius);
```

```
Circle makeCircle(Point center, double radius);
```

⚠️ **No dispone de shortcut por defecto**, pero los IDEs de JetBrains disponen de este refactor. Con el cursor sobre los parámetros **Ctrl + T** (refactor this) > **Introduce Parameter Object**

```
class Point {
    private double x;
    private double y;

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

REPLACE PARAMETER WITH QUERY

Para evitar pasar varios parámetros con los diferentes atributos de un objeto, se puede pasar el objeto directamente y que sea en el propio método quien accede a los parámetros necesarios.

9

```
availableVacation(anEmployee, anEmployee.grade);

public Vacation availableVacation(Employee anEmployee, Grade grade) {
  // calculate vacation...
}
```

⚠️ **No dispone de un shortcut o refactor directo.** Se realiza en varios pasos:
1. **Duplicar** el método a modificar, modificando los parámetros para que reciban el objeto deseado.
2. **Sustituir** la implementación del método original por una llamada al nuevo método.
3. Sobre el método original hacer un **Inline method (** 🍏 ⌥ + ⌘ + **N** o **Ctrl + Alt + N)**

```
public Vacation availableVacation(Employee anEmployee) {
    Grade grade = anEmployee.grade;
  // calculate vacation...
}
```
1

```
availableVacation(anEmployee, anEmployee.grade);

public Vacation availableVacation(Employee anEmployee, Grade grade) {
    return availableVacation(anEmployee);
}

public Vacation availableVacation(Employee anEmployee) {
    Grade grade = anEmployee.grade;
  // calculate vacation...
}
```
2

```
availableVacation(anEmployee);

public Vacation availableVacation(Employee anEmployee) {
  Grade grade = anEmployee.grade;
  // calculate vacation...
}
```
3

REPLACE PARAMETER WITH EXPLICIT METHOD

10

Cuando los métodos tienen parámetros con valor por defecto, estos suele provocar que el método tenga condicionales con diferentes caminos a seguir. Reemplazar estos métodos por otros específicos que implementen cada caso de manera aislada ayuda a evitar bifurcaciones y posibles errores al tener estos valores por defecto.

```
public void setVisible(boolean visible = false) {
        this.visible = visible;
}
```

⚠️ **No dispone de un shortcut o refactor directo.**

```
public void show() {
        this.visible = true;
}

public void hide() {
        this.visible = false;
}
```