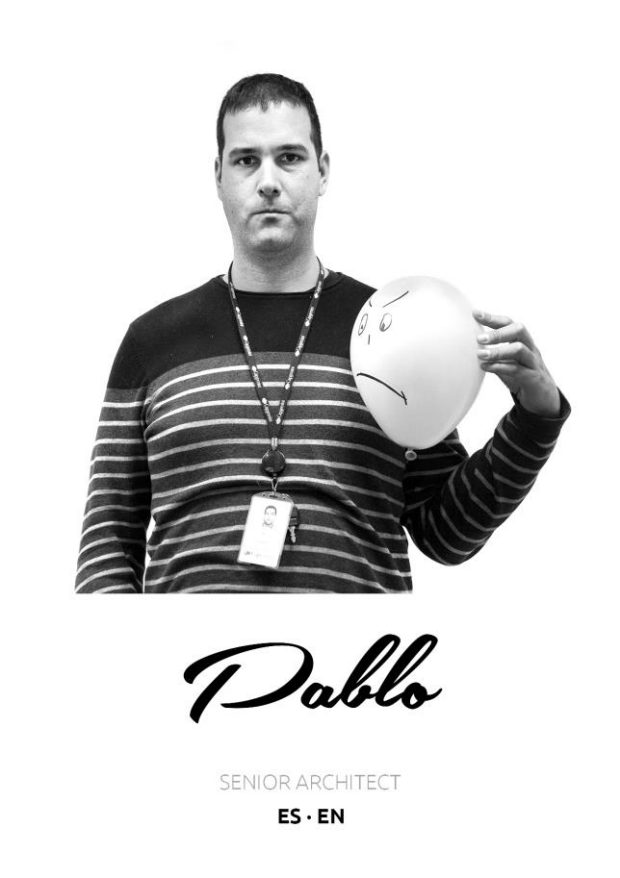
Charla Refactorización

Capgemini

¿Quién soy yo?



pablo.jimenez-martinez@capgemini.com

https://github.com/pajimene

Technical Lead Senior de Solution Delivery Office (SDO) en Capgemini.

- Ayudamos en el diseño / arquitectura de las aplicaciones y a solucionar las piezas más complejas
- Damos soporte en el arranque de los proyectos y, sobre todo, cuando saltan riesgos y problemas.
- Iniciativas de formación, mentoring y reskilling para los compañeros del área de Capgemini Spain.

CODING DOJO

¿Qué es Refactorización?

Smells más comunes

A refactorizar!

¿Qué es refactorización?

¿Qué es?

¿Qué es refactorización?

¿Qué es?

“Cambios en el código para hacerlo más fácil de entender y más barato de modificar, sin alterar su comportamiento observable”

- Martin Fowler -

¿Qué es refactorización?

¿Qué es?

“Cambios en el código para hacerlo **más fácil de entender** y más barato de modificar, **sin alterar su comportamiento observable**”

- Martin Fowler -

¿Qué es refactorización?

¿Por qué quiero el código más fácil de entender?

¿Qué es refactorización?

¿Por qué quiero el código más fácil de entender?

Tiempo → Es muchísimo más sencillo dedicarle un tiempo a ordenar y limpiar tu código (refactoring), que dedicar tiempo a entender lo que hace un “código mal escrito” por otra persona.

- Ojo, que esa otra persona puedes ser tu, pero hace 6 meses.
- El código se tiene que poder leer como si fuera una novela.

¿Qué es refactorización?

¿Por qué quiero el código más fácil de entender?

Tiempo → Es muchísimo más sencillo dedicarle un tiempo a ordenar y limpiar tu código (refactoring), que dedicar tiempo a entender lo que hace un “código mal escrito” por otra persona.

Testeabilidad → Es mucho más fácil detectar errores con un código que se entiende y es más fácil generar test unitarios que prueben ese código.

- Un “código mal escrito”, a menudo, tiene zonas oscuras que no sabes realmente lo que hace y es más propenso a tener errores.
- Generalmente el código limpio tiene menos errores, menos duplicidades, menos trozos de código, y crear test para él es sencillo porque comprendes lo que debe hacer y lo que no.

¿Qué es refactorización?

¿Por qué quiero el código más fácil de entender?

Tiempo → Es muchísimo más sencillo dedicarle un tiempo a ordenar y limpiar tu código (refactoring), que dedicar tiempo a entender lo que hace un "código mal escrito" por otra persona.

Testeabilidad → Es mucho más fácil detectar errores con un código que se entiende y es más fácil generar test unitarios que prueben ese código.

Civismo / Respeto → Somos seres sociales, trabajamos en equipo, el código siempre termina volviendo o algún compañero lo lee.

¿Qué es refactorización?

¿Por qué quiero el código más fácil de entender?

Tiempo → Es muchísimo más sencillo dedicarle un tiempo a ordenar y limpiar tu código (refactoring), que dedicar tiempo a entender lo que hace un "código mal escrito" por otra persona.

Testeabilidad → Es mucho más fácil detectar errores con un código que se entiende y es más fácil generar test unitarios que prueben ese código.

Civismo / Respeto → Somos seres sociales, trabajamos en equipo, el código siempre termina volviendo o algún compañero lo lee.

Ejemplo proyecto real

¿Qué es refactorización?

¿Cómo refactorizamos?

¿Qué es refactorización?

¿Cómo refactorizamos?

Clean Code / Testing → Lo primero siempre que hagamos Refactor estamos obligados a testear el código resultante. No podemos saltar al vacío sin una **red de seguridad**.

¿Qué es refactorización?

¿Cómo refactorizamos?

Clean Code / Testing → Lo primero siempre que hagamos Refactor estamos obligados a testear el código resultante. No podemos saltar al vacío sin una **red de seguridad**.

Smells / Dirty Code → Después debemos detectar el código sucio. Para eso tenemos los "smells" (catálogo de malas prácticas).

Ayudarnos de herramientas de análisis de código (linter, SonarQube o similares).

https://refactoring.guru/refactoring/smells

¿Qué es refactorización?

¿Cómo refactorizamos?

Clean Code / Testing → Lo primero siempre que hagamos Refactor estamos obligados a testear el código resultante. No podemos saltar al vacío sin una **red de seguridad**.

Smells / Dirty Code → Después debemos detectar el código sucio. Para eso tenemos los "smells" (catálogo de malas prácticas).

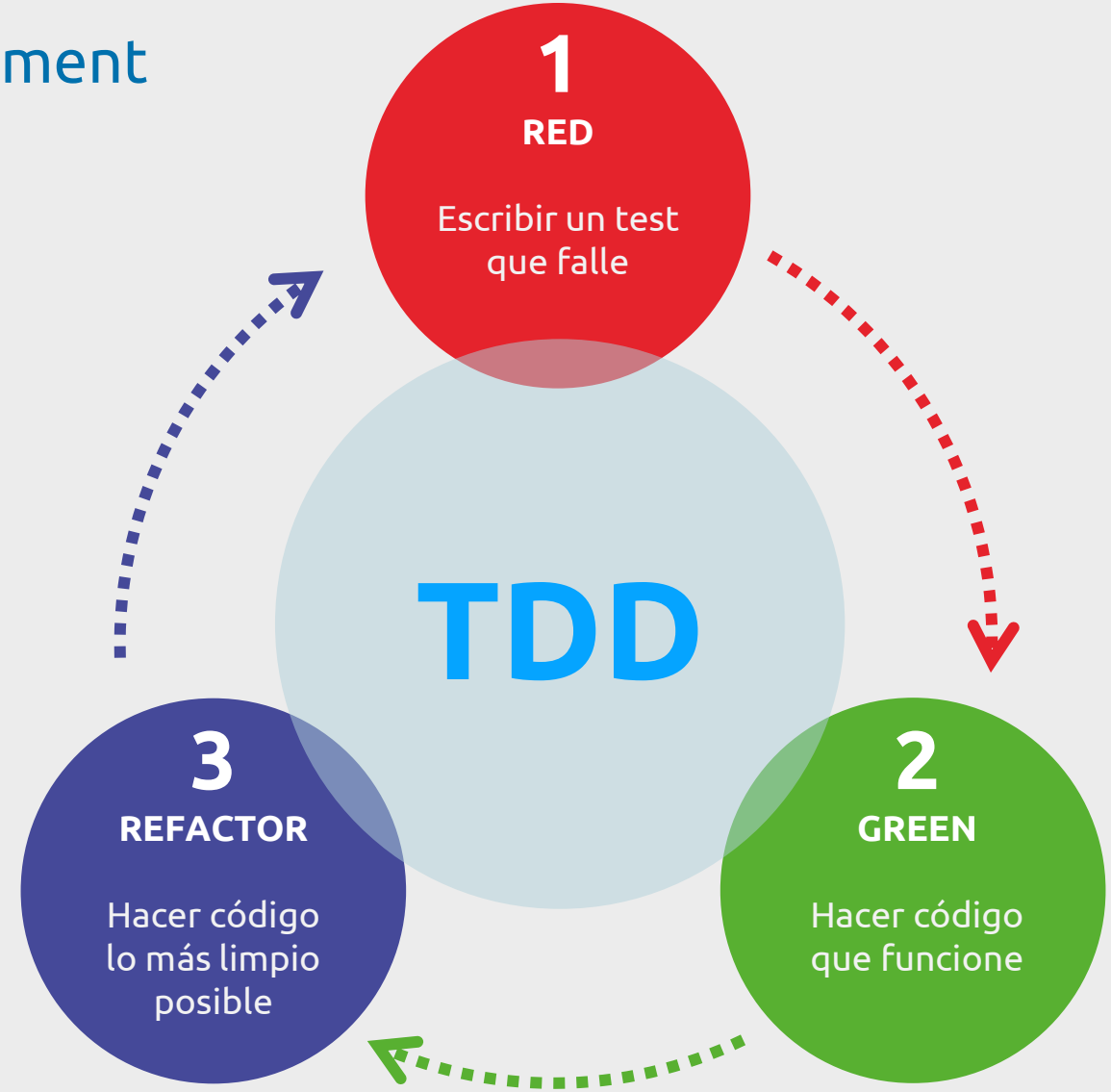
Ayudarnos de herramientas de análisis de código (linter, SonarQube o similares).

Refactor Techniques → Tener claras las acciones que se deben acometer para corregir ciertos "smells". Intentar aplicar pequeños cambios. **Ojo, aplicar el sentido común**.

https://refactoring.guru/refactoring/smells

¿Qué es refactorización?

Test-Driven Development



¿Qué es refactorización?

Basic Smells

COMMENTS

MAGIC NUMBER

LONG METHOD

DUPLICATE METHOD

LARGE CLASS

LONG PARAMETER LIST

https://refactoring.guru/refactoring/smells

¿Qué es refactorización?

Refactor Techniques

EXTRACT CLASS

INTRODUCE EXPLAINING VARIABLE

EXTRACT METHOD

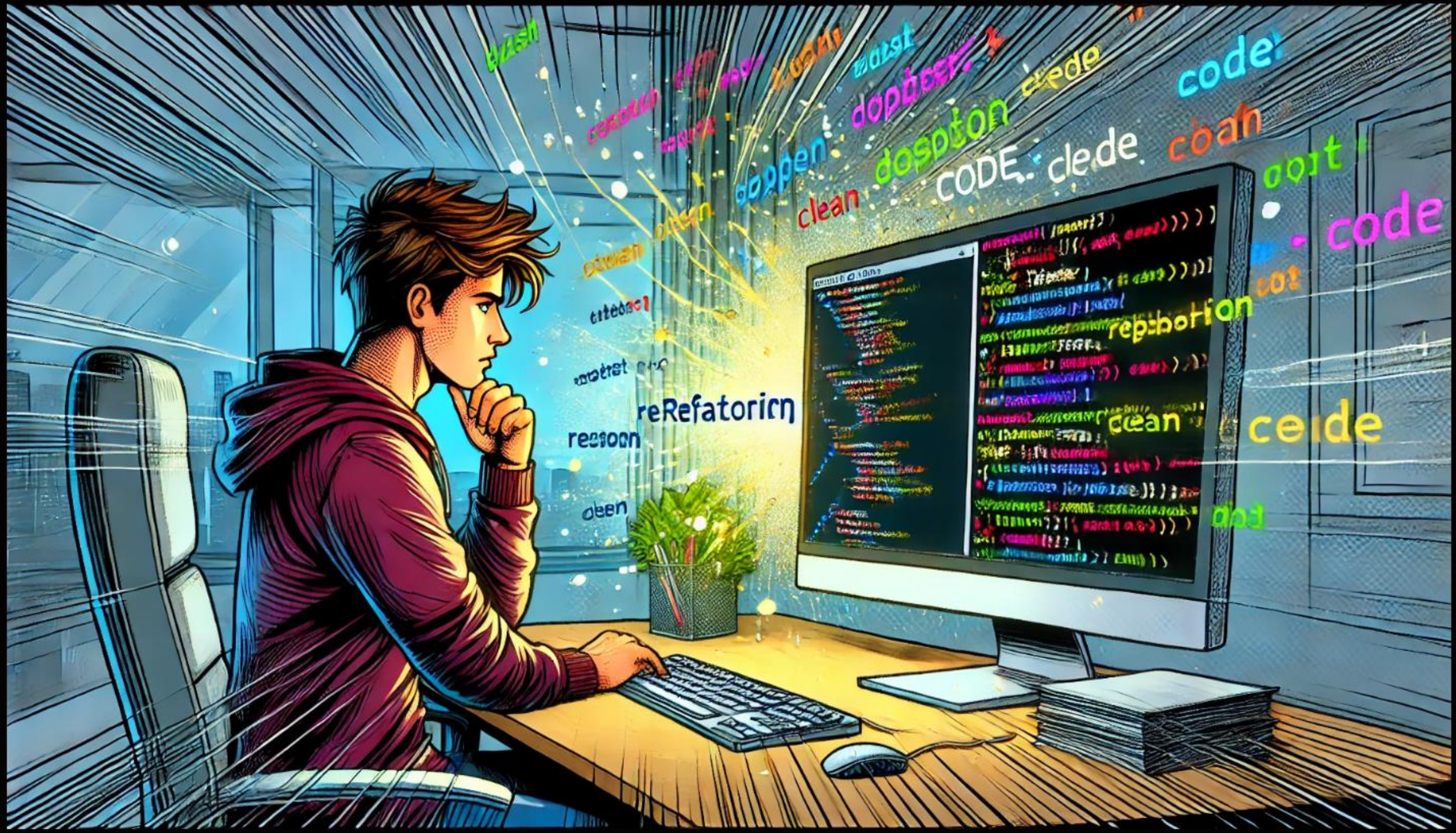
REMOVE ASSIGMENTS TO PARAMETER

SELF ENCAPSULATE FIELDS

INLINE METHODS

ENCAPSULATE COLLECTIONS

https://refactoring.guru/refactoring/smells


clean
CODE
code
reRefatorirn
clean
code

¿Qué es refactorización?

Design Smells

SWITCHS

PRIMITIVE OBSESSION

MESSAGE CHAINS

SPECULATIVE GENERALIZATION

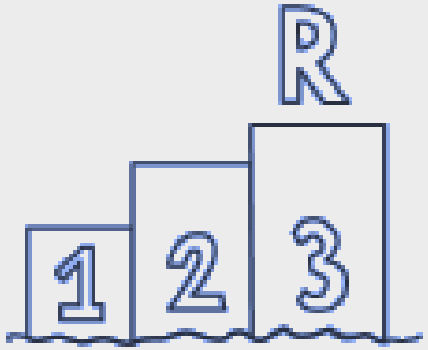
DATA CLUMPS

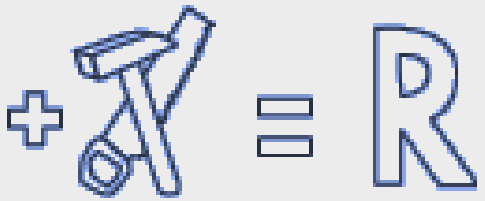
FEATURE ENVY

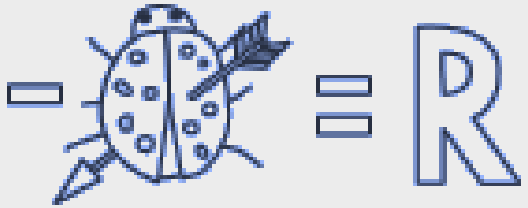
https://refactoring.guru/refactoring/smells

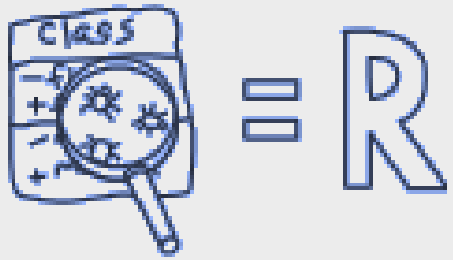
¿Qué es refactorización?

¿Cuándo refactorizamos?









¿Qué es refactorización?

¿Cuándo refactorizamos?

Regla del Boy Scout: Siempre deja el lugar de acampada más limpio que como lo encontraste.

¿Qué es refactorización?

Resumen

¿Qué es? → Hacer cambios en el código para hacerlo **más legible** sin cambiar su comportamiento.

¿Para qué? → Mejor mantenibilidad, código más limpio, más legible, más testable. Por responsabilidad con el resto de los compañeros y con uno mismo (dentro de unos años).

¿Cómo? → Detectar smells (básicos / diseño) y aplicar técnicas. Ayudaos de herramientas (sonarqube, linter, etc.). Se aprende y se mejora con la experiencia. Utilizad **siempre** testing (TDD).

¿Cuándo? → Siempre que **modifiquemos un trozo de código**, dejarlo mejor que como estaba inicialmente (nueva funcionalidad, corrección o revisión de código).

¿Qué es Refactorización?

Smells más comunes

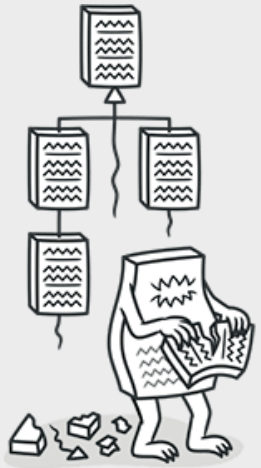
A refactorizar!

Smells más comunes

Clasificación de smells

Object-Orientation Abusers

Dispensables



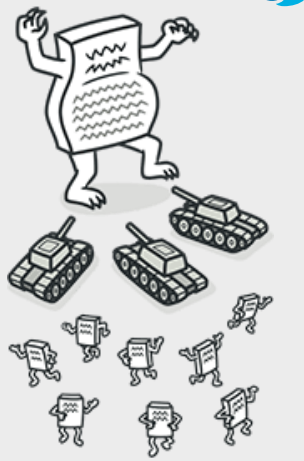
Bloaters

Change Preventers

Couplers

Smells más comunes

Bloaters. **Long Method.**



Método que contiene demasiadas líneas de código.

Cómo detectarlo	Refactoring Techniques
Un método cuyo código no cabe en la pantalla.	• Extraer método • Extraer método en otro objeto • Descomponer condicionales

Cuanto más largo es un método más difícil es entenderlo. Además los métodos largos, a menudo, esconden código duplicado más fácilmente.

Smells más comunes

Bloaters. **Long Method.**

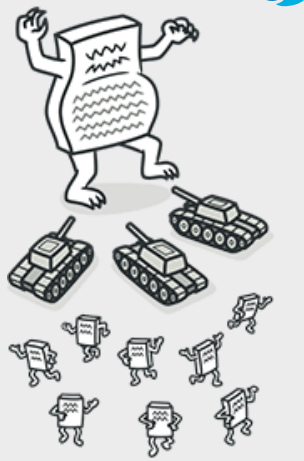
```java
public File generateReport(DataReportDto data, PDFGenerator generator) {

    if (data.isHeaderVisible()) {
        generator.addHeader(data.getHeader());
    }

    for (int i = 0; i < data.getPages().size(); i++) {
        generator.addPage(data.getPages().get(i));

        if (data.isNumberPageVisible() && data.getNumberPage() > 0
        && data.getTitle() != null && data.getTitle().length > 0) {
            generator.addPageNumber(data.getNumberPage());
        }
    }

    generator.addFooter(data.getFooter());

    return generator.renderize();
}
```

```java
public File generateReport(DataReportDto data, PDFGenerator generator) {

    generateHeader(data, generator);
    generateBody(data, generator);
    generateFooter(data, generator);

    return generator.renderize();
}
```

menudo, esconden código duplicado más fácilmente.

Smells más comunes

Bloaters. **Long Parameter List.**



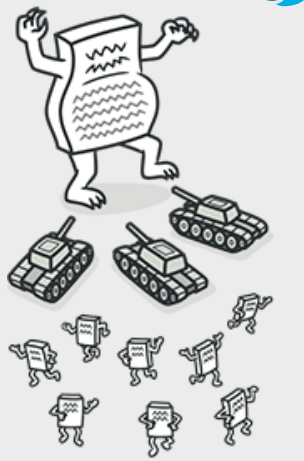
Un método que contiene demasiados parámetros.

Cómo detectarlo	Refactoring Techniques
Más de tres o cuatro parámetros para un método.	• Reemplazar por una llamada • Introducir un objeto de parámetros • Preservar el objeto original

Una lista larga de parámetros suele ser difícil de entender. Además, por lo general, la lista de parámetros tiende a crecer y eso nos obligará a ir modificando constantemente la llamada.

Smells más comunes

Bloaters. **Data Clumps.**



Diferentes métodos que contiene grupos idénticos de parámetros.

Cómo detectarlo	Refactoring Techniques
Deja-vu en la lista de parámetros entre dos o más métodos.	• Introducir un objeto de parámetros • Preservar el objeto original • Extraer a clase

Es una forma de código duplicado y hace complicado la lectura del código. Si modificas el grupo de parámetros lo más probable es que tengas que modificarlo en todos los métodos.

Smells más comunes

Bloaters. **Data Clumps.**

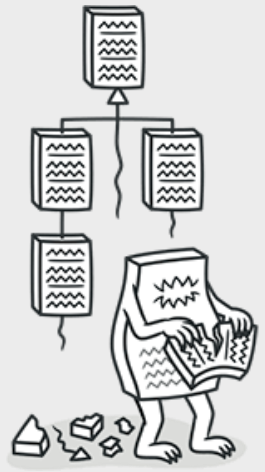
```
public void sendEmail(String from,
                      String to[],
                      String cc[],
                      String subject,
                      String body) {

    ...

    registerEmailSendSuccessfull(from, to, cc, subject, body);

}
```

```
public class EmailData {
    private String from;
    private String to[];
    private String cc[];
    private String subject;
    private String body;
}


public void sendEmail(EmailData data) {

    ...

    registerEmailSendSuccessfull(data);
}
```

métodos.

Smells más comunes

OO Abusers. **Switch Statements (or if-else).**



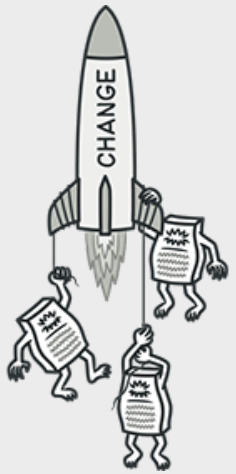
Un método que contiene demasiadas secuencias if-else o un switch complejo.

Cómo detectarlo	Refactoring Techniques
Una sentencia switch compleja Estructuras encadenadas de if-else	• Reemplazar condición por métodos • Usar cláusulas de guarda • Reemplazar con patrón State / Strategy • Reemplazar con polimorfismo

Dificulta la lectura del código, obliga al cerebro a releer muchas veces el código por cada una de las condiciones del switch.
A menudo el switch tiende a replicarse por varias partes del código.

Smells más comunes

Change Preventers. **Magic Number.**



Son valores que aparecen en el código y sin un contexto previo no sabemos el significado de ese valor.

Cómo detectarlo	Refactoring Techniques
Valores literales en el código	• Reemplazar por Constante

Dificulta la lectura del código, al no tener contexto previo se puede malinterpretar el valor.
A menudo estos valores están muy repetidos en el código y un cambio en uno significa buscarlo en todo el código.

Smells más comunes

Change Preventers. **Magic Number.**

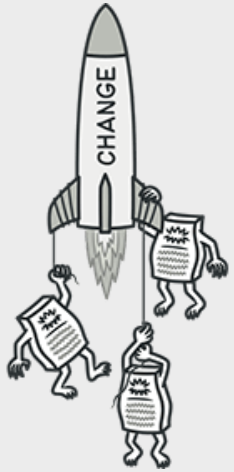

```
if (strDatos.countTokens() == 26)

...

listaGrupos.add("00000000000001");

...

if (Misc.esVacio(regVacuna.getMotivosNovac())
    && (edad < 18)
    && !Misc.esVacio(regVacuna.getProducto())
    && regVacuna.getProducto().equals("69911000122102"))
```

```
if (strDatos.countTokens() == MAX_TOKENS)

...

listaGrupos.add(GRUPO_POBLACIONAL_GENERICO);

...

if (Misc.esVacio(regVacuna.getMotivosNovac())
    && (edad < MENOR_EDAD)
    && !Misc.esVacio(regVacuna.getProducto())
    && regVacuna.getProducto().equals(VACUNA_PFIZER))
```

A menudo estos valores están muy repetidos en el código y un cambio en uno significa buscarlo en todo el código.

Smells más comunes

Change Preventers. **Uncommunicative Name.**



El nombre de un método, atributo o clase, no describe correctamente la intención del código.

Cómo detectarlo	Refactoring Techniques
Nombres cortos (de una, dos o tres letras) Abreviaciones técnicas de nombres	• Renombrar para dar semántica y contexto

Dificulta la lectura del código, al no tener un nombre correcto y descriptivo puede hacer que se malinterprete la intención del algoritmo.

Smells más comunes

Change Preventers. **Uncommunicative Name.**


```
private boolean intervaloValido(Date fechaV1, Date fechaV2, int inter) throws Exception{
    try {
        Calendar cal = Calendar.getInstance();
        cal.setTime(fechaV2);
        cal.add(Calendar.DAY_OF_YEAR, inter);
        Date fechaComp = cal.getTime();

        //Si la fecha
        if (fechaComp.before(fechaV1)){
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        return false;
    }
}
```

que se malinterprete la intención del algoritmo.

Smells más comunes

Change Preventers. **Uncommunicative Name.**


```
private boolean intervaloValido(Date fechaV1, Date fechaV2, int inter) throws Exception{
    try {
        Calendar cal = Calendar.getInstance();
        cal.setTime(fechaV2);
        cal.add(Calendar.DAY_OF_YEAR, inter);
        Date fechaComp = cal.getTime();

        //Si la fecha de primera vacuna (+ intervalo) es anterior a la fecha de segunda vacuna
        if (fechaComp.before(fechaV1)){
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        return false;
    }
}
```

que se malinterprete la intención del algoritmo.

Smells más comunes

Dispensables. **Comments.**

Usar comentarios para clarificar partes del código evidencia que el código está mal implementado.

Cómo detectarlo	Refactoring Techniques
Comentarios explicando variables, métodos o pasos dentro de un método. **OJO,** no javadoc.	• Extraer método • Renombrar método o campo • Introducir una aserción

Dificulta la lectura del código y a veces confunde al lector. Además al formar parte del código se debe mantener cosa que no se hace muy a menudo y se quedan comentarios obsoletos.

Smells más comunes

Dispensables. **Comments.**

```
private boolean intervaloValido(Date fechaV1, Date fechaV2, int inter) throws Exception{
    try {
        Calendar cal = Calendar.getInstance();
        cal.setTime(fechaV2);
        cal.add(Calendar.DAY_OF_YEAR, inter);
        Date fechaComp = cal.getTime();

        //Si la fecha de primera vacuna (+ intervalo) es anterior a la fecha de segunda vacuna
        if (fechaComp.before(fechaV1)){
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        return false;
    }
}
```

obsoletos.

Smells más comunes

Dispensables. **Duplicate Code.**

Cuando hay duplicidad de código en diferentes sitios o también cuando dos métodos hacen lo mismo pero tienen diferente código.

Cómo detectarlo	Refactoring Techniques
Deja-vu en código	• Extraer método / clase • Ascender a método padre • Extraer método a template
Dificulta la lectura del código, al tener código duplicado existe más código, es más largo y es más complejo.	

¿Qué es Refactorización?

Smells más comunes

A refactorizar!

A refactorizar!

Antes de empezar

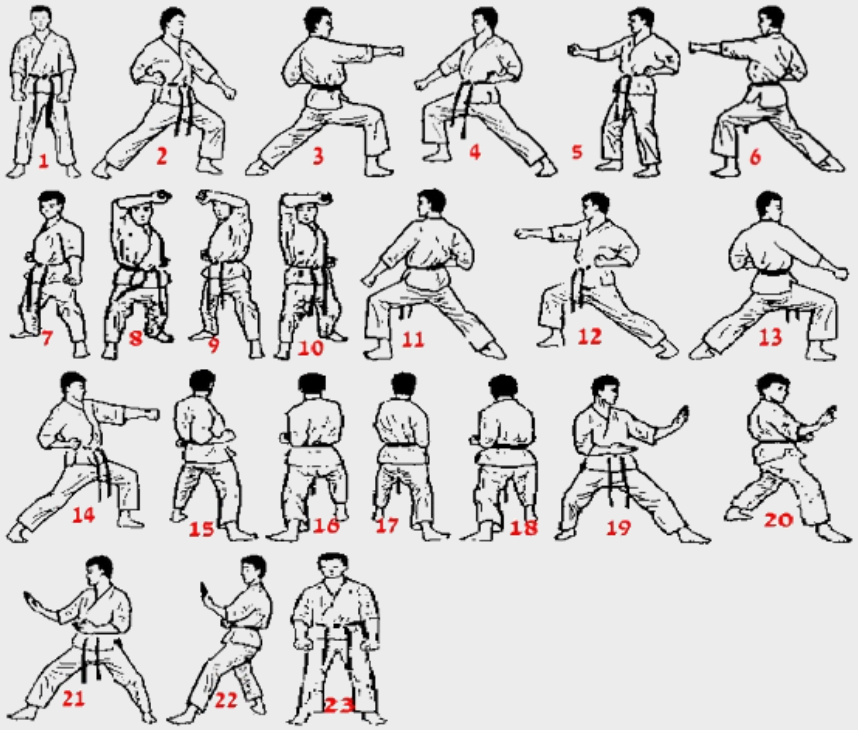
Lugar donde se reúne la gente para practicar y entrenar artes marciales.

CODING DOJO

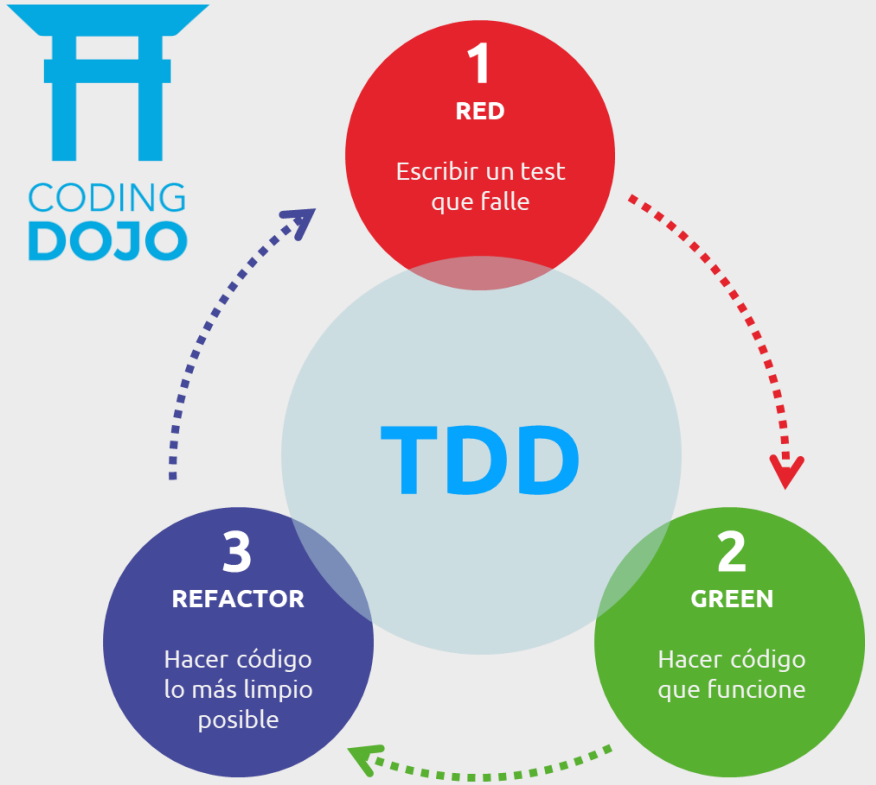
Lugar donde nos reunimos para practicar y entrenar buenas formas de programación.

A refactorizar!

Antes de empezar



Ejecutan Katas para aprender los movimientos y las técnicas.



Desarrollamos Katas usando metodología TDD.

A refactorizar!

Más información

https://github.com/ccsw-csd/codingDojo

https://ccsw-csd.github.io/kata-rock_paper_scissors_lizard_spock/site/

https://ccsw-csd.github.io/kata-string_calculator/site/

https://codingdojo.org/kata/

http://codekata.com/

Otros → Buscar en Google: **Coding Dojo** o **Kata Code**

A refactorizar!

Kata Gilded Rose

A refactorizar!

Gilded Rose

Bienvenido al equipo de Gilded Rose. Somos una pequeña posada ubicada estratégicamente en una prestigiosa ciudad, atendida por la amable Allison. También compramos y vendemos objetos de alta calidad. Por desgracia, nuestros objetos van bajando de calidad a medida que se aproxima la fecha de venta.

Tenemos un sistema instalado que actualiza automáticamente el inventario. Este sistema fue desarrollado por un muchacho con muy poco cuidado, llamado Leeroy, que ahora se dedica a nuevas aventuras. Por suerte, Leeroy implementó un sistema de tests muy meticuloso.

Necesitamos agregar un nuevo objeto al sistema para comerciar con él.

A refactorizar!

Gilded Rose

En el documento del sistema, tenemos las siguientes directrices:

- Todos los artículos (Item) tienen una propiedad sellIn que denota el número de días que tenemos para venderlo
- Todos los artículos tienen una propiedad quality que denota cuan valioso es el artículo
- Al final de cada día, nuestro sistema decrementa ambos valores para cada artículo mediante el método updateQuality

Bastante simple, ¿no? Bueno, ahora es donde se pone interesante.

A refactorizar!

Gilded Rose

- Una vez superado el sellIn de venta, la quality se degrada al doble de velocidad
- La quality de un artículo nunca es negativa, ni tampoco mayor a 50

- El artículo "**Aged brie**" incrementa su quality a medida que envejece
  - Su quality aumenta en 1 unidad cada día
  - Cuando caduca su quality aumenta 2 unidades por día
- El artículo "**Sulfuras, Hand of Ragnarok**", no modifica su sellIn ni quality
- El artículo "**Backstage Passes**", incrementa su quality a medida que envejece
  - si faltan 10 días o menos para el concierto, quality se incrementa en 2 unidades
  - si faltan 5 días o menos, quality se incrementa en 3 unidades
  - Al vencimiento del sellIn su quality cae a 0

A refactorizar!

Gilded Rose

¿Cuál es el requerimiento nuevo?

Hace poco contratamos un proveedor de artículos conjurados mágicamente y por tanto necesitamos una ampliación de nuestra aplicación para poder venderlos.

- Los artículos "**Conjured**" degradan su calidad 2 unidades al día.

Siéntete libre de realizar cualquier cambio al método updateQuality y agregar el código que sea necesario, sin embargo, no alteres el objeto Item ni sus propiedades ya que pertenecen al goblin de la posada, que en un ataque de ira te puede liquidar de un golpe.

A refactorizar!

Kata Gilded Rose

People matter, results count.

About Capgemini

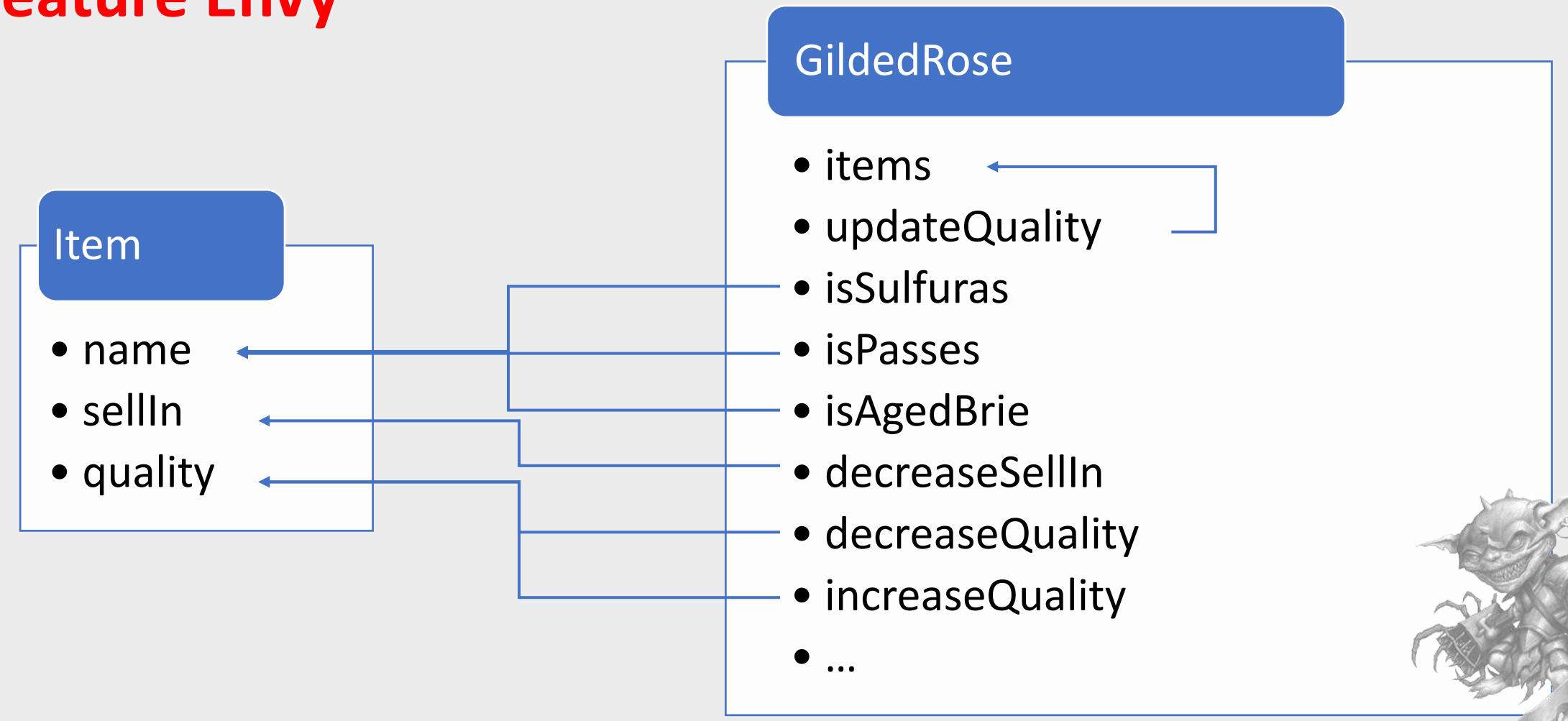
With more than 190,000 people, Capgemini is present in over 40 countries and celebrates its 50th Anniversary year in 2017. A global leader in consulting, technology and outsourcing services, the Group reported 2016 global revenues of EUR 12.5 billion. Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at

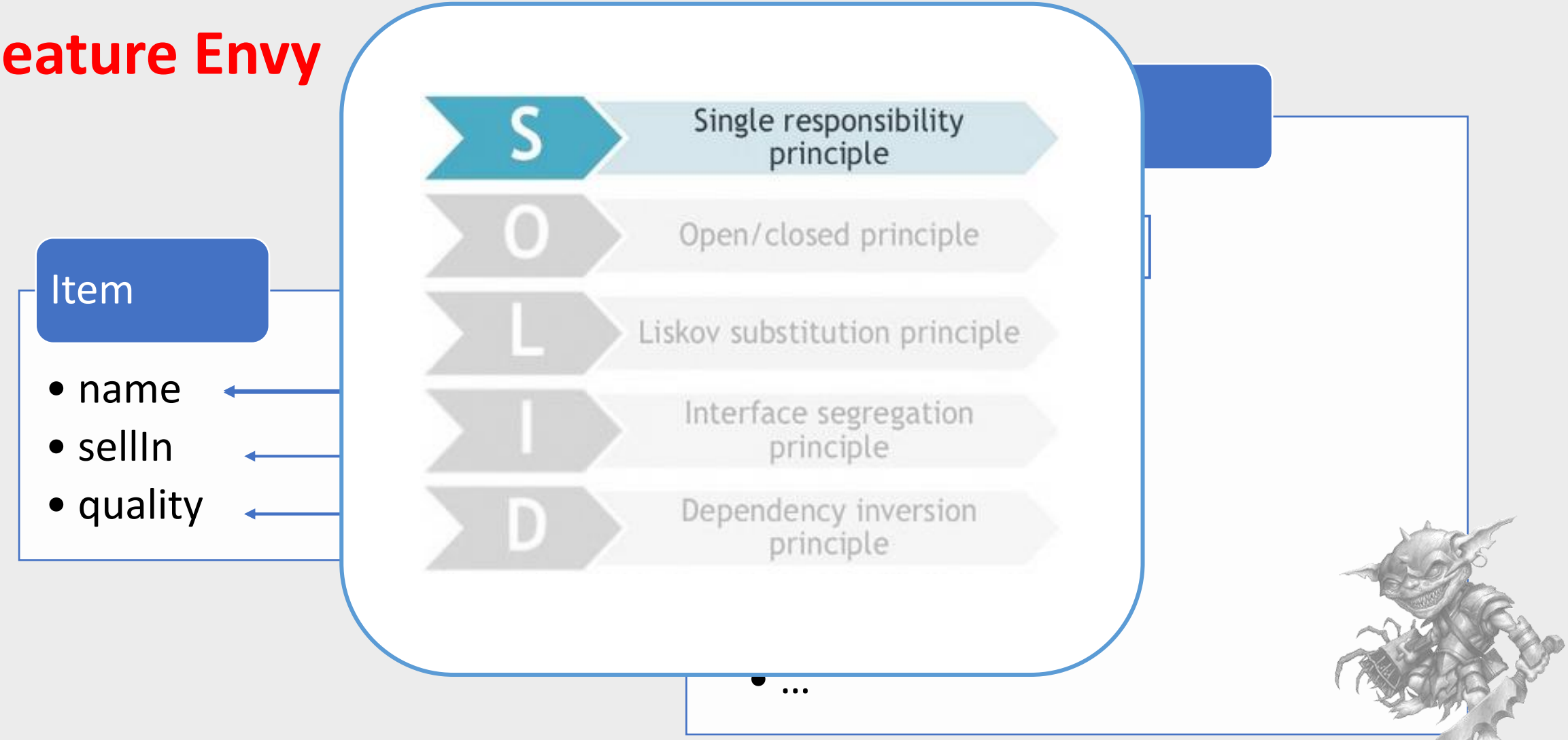
www.capgemini.com

Advanced Refactoring (Design smells)

Feature Envy



Feature Envy



Advanced Refactoring (Design smells)

Item

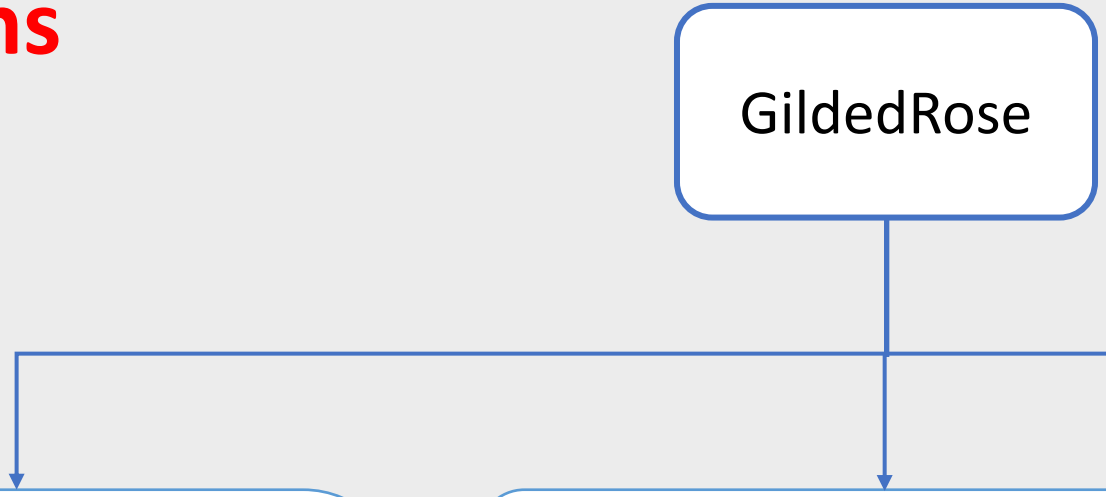
- name
- sellIn
- quality
- updateQuality
- isSulfuras
- isPasses
- isAgedBrie
- decreaseSellIn
- decreaseQuality
- increaseQuality
- …

GildedRose

- items
- updateQuality

Advanced Refactoring (Design smells)

Switchs




```
private void updateQualityItem(Item item) {
   if (isSulfuras(item))
     return;

   else if (isBackstage(item))
     updateQualityBackstage(item);

   else if (isAgedBrie(item))
     increaseQuality(item);

   else if (isConjuredItem(item))
     decreaseQuality(item, QUALITY_STEP * 2);

   else
     decreaseQuality(item);
 }
```

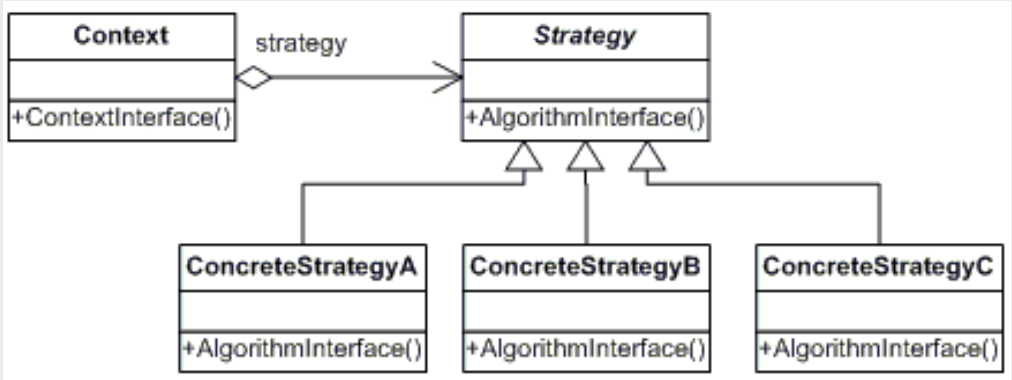
```
private void updateSellIn(Item item) {

    if (isSulfuras(item))
      return;

    item.setSellIn(item.getSellIn() - SELLIN_STEP);
  }
```

```
private void checkSellInExpired(Item item) {

   if (sellInNotExpired(item))
     return;

   if (isAgedBrie(item))
     increaseQuality(item);

   else if (isBackstage(item))
     dropQuality(item);

   else
     decreaseQuality(item);
 }
```


Advanced Refactoring (Design smells)

Strategy Pattern

El patrón ‘Strategy’ tiene sentido cuando nos encontramos un escenario en el que para conseguir un objetivo, existen diferentes estrategias para lograrlo.

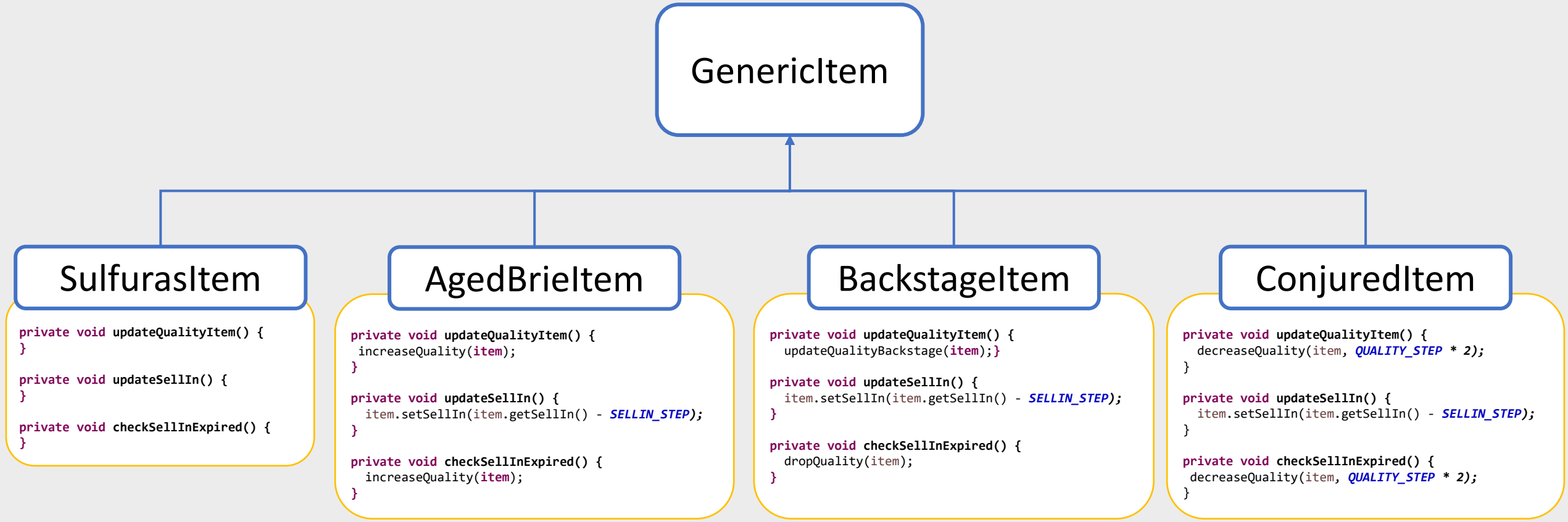


Como se implementa:

- El contexto utiliza una clase donde se define el método o métodos generales y la implementación más generalista
- Cada una de las diferentes estrategias, extienden esa clase padre e implementan su propia estrategia concreta

Si aparece una nueva estrategia, tan solo hay que añadir una nueva clase que herede de la genérica e implementar esa estrategia concreta.

Advanced Refactoring (Design smells)




```
private void updateQualityItem() {
}

private void updateSellIn() {
}

private void checkSellInExpired() {
}
```

```
private void updateQualityItem() {
 increaseQuality(item);
}

private void updateSellIn() {
  item.setSellIn(item.getSellIn() - SELLIN_STEP);
}

private void checkSellInExpired() {
  increaseQuality(item);
}
```

```
private void updateQualityItem() {
  updateQualityBackstage(item);}

private void updateSellIn() {
  item.setSellIn(item.getSellIn() - SELLIN_STEP);
}

private void checkSellInExpired() {
  dropQuality(item);
}
```

```
private void updateQualityItem() {
  decreaseQuality(item, QUALITY_STEP * 2);
}

private void updateSellIn() {
  item.setSellIn(item.getSellIn() - SELLIN_STEP);
}

private void checkSellInExpired() {
 decreaseQuality(item, QUALITY_STEP * 2);
}
```

