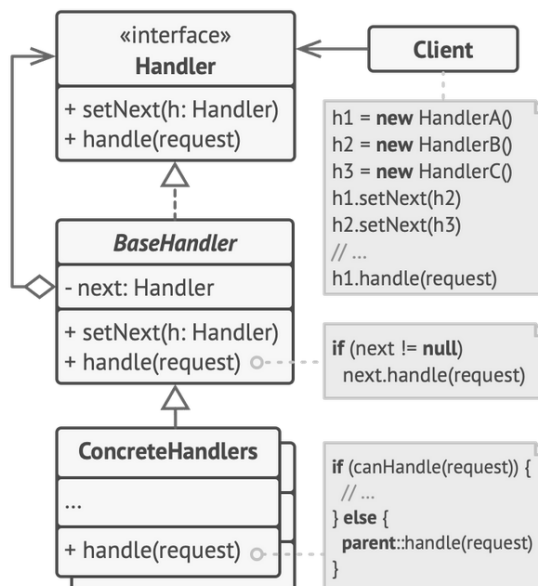


# Les chaines de responsabilités

La chaine de responsabilité est un moyen de faire beaucoup de traitement sur un même objet. Cela peut être un très bon moyen de gérer une cascade de conditions `if` dans un code. Ainsi chaque bloc de `if` est séparé et sont liés entre eux. Cela permet aussi d'isoler ces différentes vérifications dans des fichiers séparés.

- 1 Le **Handler** déclare une interface commune pour tous les handlers concrets. En général, il ne comporte qu'une seule méthode pour gérer les demandes, mais il peut parfois en contenir une autre pour désigner le prochain handler de la chaîne.
- 2 Le **Handler de Base** est une classe facultative dans laquelle le code commun à tous les handlers peut être écrit.  
  
En général, cette classe définit un attribut qui pointe vers le prochain handler. Les clients peuvent assembler une chaîne en passant un handler au constructeur ou au setter du handler précédent. La classe peut également implémenter le comportement par défaut d'un handler : il s'assure de l'existence du prochain handler, puis lui délègue le travail.



- 4 Le **Client** peut créer les chaînes juste une fois ou les assembler dynamiquement en fonction de la logique métier. Notez bien que la demande initiale n'est pas obligatoirement envoyée au premier handler de la chaîne.
- 3 Les **Handlers Concrets** contiennent le code qui traite les demandes. Lors de la réception d'une demande, chaque handler décide s'il doit la traiter et s'il doit l'envoyer plus loin dans la chaîne.  
  
Les handlers sont généralement autonomes et non modifiables, et n'accepteront qu'une seule fois les données nécessaires par le biais du constructeur.

A noter que ce schéma utilise l'héritage mais qu'il faut toujours préférer la composition à l'héritage, il vaut donc mieux simplement avoir des `ConcreteHandlers` qui implémentent tous `Handler`.

Aussi pour simplifier l'écriture on peut simplement mettre le `setNext` dans le constructeur du `Handler` (ce qui permet de rendre la chaîne plus propre par après)

## Exemple

## Code avant

Voici l'horrible code à réorganiser :

```
package com.gildedrose;

class GildedRose {
    Item[] items;

    public GildedRose(Item[] items) {
        this.items = items;
    }

    public void updateQuality() {
        for (int i = 0; i < items.length; i++) {
            if (!items[i].name.equals("Aged Brie")
                && !items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                if (items[i].quality > 0) {
                    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                        items[i].quality = items[i].quality - 1;
                    }
                }
            } else {
                if (items[i].quality < 50) {
                    items[i].quality = items[i].quality + 1;

                    if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                        if (items[i].sellIn < 11) {
                            if (items[i].quality < 50) {
                                items[i].quality = items[i].quality + 1;
                            }
                        }

                        if (items[i].sellIn < 6) {
                            if (items[i].quality < 50) {
                                items[i].quality = items[i].quality + 1;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
            items[i].sellIn = items[i].sellIn - 1;
        }

        if (items[i].sellIn < 0) {
            if (!items[i].name.equals("Aged Brie")) {
                if (!items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (items[i].quality > 0) {
                        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                            items[i].quality = items[i].quality - 1;
                        }
                    }
                } else {
                    items[i].quality = items[i].quality - items[i].quality;
                }
            } else {
                if (items[i].quality < 50) {
                    items[i].quality = items[i].quality + 1;
                }
            }
        }
    }
}
}

```

## Code après

Et voici ce que l'on obtient en utilisant le patron de la chaîne de responsabilités.

- Classe principale GildedRose :

```

package com.gildedrose;

import com.gildedrose.handlers.*;

class GildedRose {
    Item[] items;

    public GildedRose(Item[] items) {
        this.items = items;
    }
}

```

```

    }

    public void updateQuality() {
        // On enchaîne les handlers
        // Ainsi Sulfuras >> AgedBrie >> BackStage >> Default
        Handler handlers = new Sulfuras(new AgedBrie(new Backstage(new Default())));

        for (int i = 0; i < items.length; i++)
            handlers.update(items[i]);
    }
}

```

- L'interface Handler

```

package com.gildedrose.handlers;

import com.gildedrose.Item;

public interface Handler {
    void update(Item item);
}

```

- Les différentes classes implémentant Handler (notez qu'ici toutes les classes sont mises au même endroit, mais dans le projet elles sont dans des fichiers séparés)

```

package com.gildedrose.handlers;

import com.gildedrose.Item;

public class AgedBrie implements Handler {
    private Handler next = null;

    public AgedBrie(Handler next) {
        this.next = next;
    }

    @Override
    public void update(Item item) {
        if (item.name.equals("Aged Brie")) {
            item.sellIn--;
            if (item.quality < 50)

```

```

        item.quality++;
        if (item.sellIn < 0 && item.quality < 50)
            item.quality++;
    }

    else if (next != null)
        next.update(item);
}

}

public class Backstage implements Handler {
    private Handler next = null;

    public Backstage(Handler next) {
        this.next = next;
    }

    @Override
    public void update(Item item) {
        if (item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
            if (item.quality < 50)
                item.quality++;
            if (item.sellIn < 11 && item.quality < 50)
                item.quality++;
            if (item.sellIn < 6 && item.quality < 50)
                item.quality++;

            // I have no idea what this shit does
            if (item.sellIn < 0)
                item.quality = item.quality - item.quality;
        } else if (next != null) {
            next.update(item);
        }
    }
}

public class Default implements Handler {
    @Override
    public void update(Item item) {

```

```
        if (item.quality > 0) {
            item.quality--;
            item.sellIn--;
            if (item.sellIn < 0)
                item.quality--;
        }
    }
}

public class Sulfuras implements Handler {
    private Handler next = null;

    public Sulfuras(Handler next) {
        this.next = next;
    }

    @Override
    public void update(Item item) {
        if (item.name.equals("Sulfuras, Hand of Ragnaros"))
            return;
        if (next != null)
            next.update(item);
    }
}
```

---

Revision #1

Created 3 October 2023 19:00:02 by SnowCode

Updated 3 October 2023 21:07:06 by SnowCode