09/06/2015

RAPPORT PROJET MCR 2015

La chaine de Responsabilités

Participants :

D’Agostino Eleonore

Meguep Sakam Michelle Vanessa

Moret Jerôme

Akesson Henrik

Ngueukam Djeuda Wilfried Karel

Professeur

Donini Pier

Assistant :

Rosat Sébastien

Table des matières

[Introduction 2](#_Toc422378735)

[I. Présentation de la Chaine de responsabilité 2](#_Toc422378736)

[1. Structure et constituants 2](#_Toc422378737)

[2. Contraintes et Conséquences 3](#_Toc422378738)

[II. Présentation de l’application à réaliser 3](#_Toc422378739)

[1. Description de l’application 3](#_Toc422378740)

[2. Diagramme de classes 4](#_Toc422378741)

[Description 4](#_Toc422378742)

[III. Problèmes irrésolus 5](#_Toc422378743)

[IV. Conclusion 5](#_Toc422378744)

# Introduction

Dans le cadre du cours de Modèles de Conception Réutilisables (**MCR**), nous avons été amenés à mettre en place une application permettant de mettre en avant un modèle de conception réutilisable précis.

Dans le cadre de notre projet, nous avons choisi d’utiliser la ***chaine de responsabilité***.

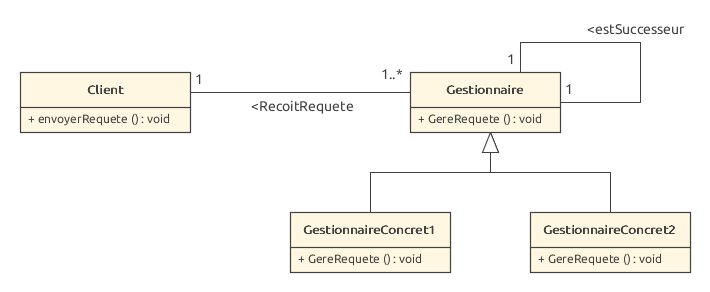
Dans ce document, nous allons présenter les particularités de ce patron de conception, ses avantages et inconvénients, et le cadre dans lequel il est avantageux de l’appliquer.

# Présentation de la Chaine de responsabilité

Dans le catalogue de modèles de conception réutilisables du ***Gof***, la ***chaine de responsabilités*** est classée comme un modèle **Objet-comportemental**. Le but d’une ***chaine de responsabilités*** est d’éviter le couplage entre la classe qui émet une requête et celle qui va la traiter. Ainsi, il existe différents cas où ce modèle est adapté à l’utilisation :

* Plusieurs objets peuvent gérer une requête mais on ne connait pas exactement l’objet qui le fera et ce dernier doit être déterminé automatiquement.
* On souhaite adresser une requête à plusieurs objets sans préciser qui la traitera
* Une requête peut être traitée par plusieurs objets.

## Structure et constituants

Une chaine de responsabilité peut être représentée par le schéma qui suit :

Les principaux constituants de la chaine sont :

* **Client :** Représente la classe qui envoie la requête à gérer à l’objet **Gestionnaire** qui doit être un gestionnaire concret.
* **Gestionnaire :** Définit les méthodes permettant de gérer les requêtes et met en place (éventuellement) la liaison entre les différents objets **GestionnaireConcret**.
* **GestionnaireConcret :** S’occupe de la requête dont il a la charge et la transmet (éventuellement) à son successeur s’il ne parvient pas à la gérer.

## Contraintes et Conséquences

* Le **Client** n’a pas besoin de connaître l’objet qu’il traitera sa requête. Il sait juste que s’il est possible de traiter cette requête alors ce sera fait.
* Ajouter de nouvelles responsabilités est plutôt aisé. On a juste à rajouter une nouvelle classe qui gère la requête.
* Il n’est pas garanti que la requête soit traitée. S’il n’existe pas un objet dans la chaine pouvant traiter une requête, celle-ci traversera toute la chaine sans trouver de gestionnaire adéquat et ne sera donc pas traitée.

# Présentation de l’application à réaliser

## Description de l’application

Notre application est un jeu qui se joue à deux sur une machine.

Le premier joueur prendra le rôle du « Killer », et devra, dans une fenêtre graphique, infliger autant de dégâts à une victime qu’il peut durant un intervalle de 10 secondes.

Le second joueur jouera le rôle du « healer ». Son but sera de trouver la meilleure chaîne de responsabilité afin de soigner la victime. Une liste de médecins lui sera proposée et il devra désigner l’ordre des médecins dans la chaîne.

Ensuite l’application va traiter le patient en respectant la chaîne de responsabilité établie.

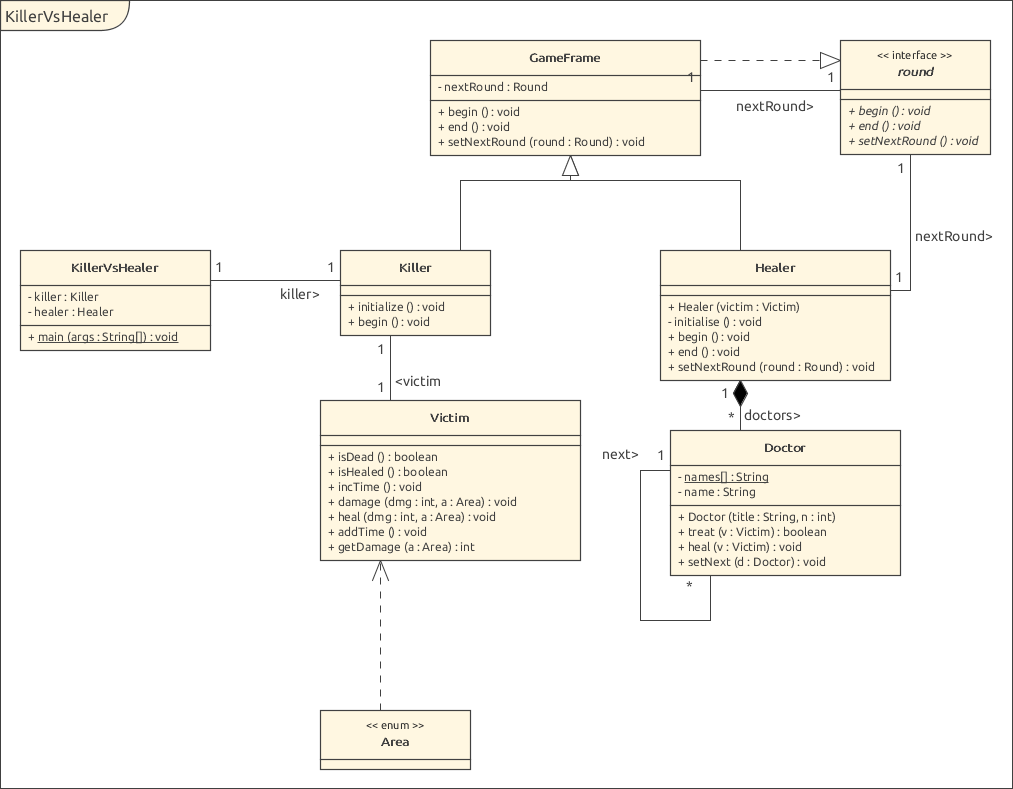
Si au bout d’un temps imparti la victime perd tous ses points de vie, « Killer » aura gagné. Si les traitements de la chaîne de responsabilité permettent de soigner le patient, le « healer » aura gagné.

Enfin, si les dégâts ne sont pas soignés intégralement et que la santé de la victime ne s’est pas dégradée au point qu’elle meurt, la partie sera comptée comme une égalité.

Il y a quatre types de docteurs :

* Les médecins, qui soignent de 2 à 3 points de vie par traitement.
* Les chirurgiens, qui divisent les dégâts par 2 ou 3.
* Les Sauveteurs, qui soignent de 3% à 5% des dégâts.
* Les réanimateurs, qui ajoutent de 1 à 5 points de vie.

## Diagramme de classes



### Description

#### KillerVsHealer

Cette classe contient la méthode main, qui va instancier un Killer « killer », un Healer « healer » et va lancer le jeu en précisant que le « next round » de killer est healer et vice-versa.

#### Killer

Killer a une victime, à laquelle l’utilisateur pourra infliger des dégâts dans la méthode « initialize ». Killer hérite de GameFrame, qui est une classe permettant de mettre en commun avec Healer des méthodes et attributs.

#### Healer

Healer, comme Killer, hérite de GameFrame. Une instance de Healer est composée d’une liste de Doctors, qui pourront appliquer des traitements à la victime qui sera passée en paramètre au constructeur de cette classe.

La chaîne de responsabilité est instanciée et utilisée dans cette classe. Les Doctors de Healer vont avoir un traitement unique, défini à leurs instanciations.

#### Doctor

Un Doctor peut soigner une victime ou la guérir. Dans le cas où un Doctor ne peut traiter la victime, il passe la requête au médecin suivant. Un docteur a un et un seul « next », mais peut théoriquement être le médecin suivant de plusieurs docteurs. Notons que dans « KillerVsHealer », ce cas n’est pas utilisé ; un docteur n’a qu’un suivant dans la chaîne et qu’un seul précédent.

#### Area

Area est un type énuméré défini dans la classe Victime. Elle peut être les jambes, les bras, la tête et le corps.

# Problèmes irrésolus

# Conclusion