Fiche méthodologique de projet

I.1 Passer à l'implémentation

Objectif

Écrire du code.

Préambule

Les classes et leurs relations développées durant la phase de conception sont finalement traduites en une implémentation dans un langage de programmation, une base de données ou un matériel spécifique.

La programmation devrait être une partie relativement mineure et mécanique dans le processus de développement car toutes les décisions difficiles ont déjà été prises en conception. Le code devrait être une simple traduction des décisions de conception en prenant en compte les spécificités d'un langage particulier.

Pendant l'implémentation, il est important de suivre de bonnes pratiques de génie logiciel. La manière dont est écrit le code est important pour la maintenabilité et les extensions (évolution).

Méthode

De manière générale et légèrement simplifiée, le codage revient à :

- **déclarer les classes** à partir du diagramme de classes de conception
- **définir les méthodes** à partir des diagrammes de séquence

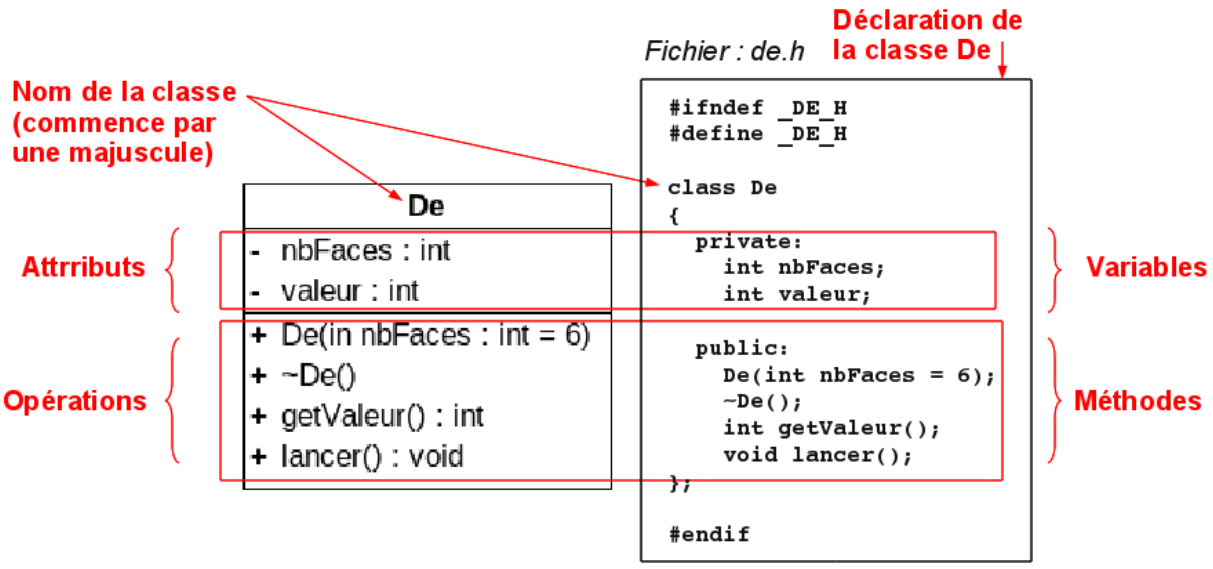
Remarque

Le principe de séparation Commande-Requête
Dans le diagramme de séquence, le message pour lancer le dé et suivi d'un second message getValeur(). La méthode lancer() n'a donc pas de valeur de retour. On utilise ici un principe de conception Orienté Objet classique pour les méthodes. Ce principe énonce que chaque méthode doit appartenir à l'une des deux catégories suivantes:
- une **commande** qui effectue une action a souvent des effets de bords comme une modification de l'état d'un objet et n'a pas de valeur de retour (sauf pour indiquer si l'action a réussi ou a échoué)
- une **requête** qui retourne des données à l'appelant et n'a pas d'effets de bord. Elle ne doit pas modifier de façon permanente l'état d'un objet.

La méthode lancer() est une commande : elle a pour effet de modifier la valeur du Dé. En conséquence, elle ne doit pas également retourner cette nouvelle valeur sinon elle violerait la règle selon laquelle elle ne doit pas appartenir aux deux catégories.

C'est un **pattern** simple : il permet de raisonner plus facilement sur l'état d'un programme et de rendre les conceptions plus faciles à comprendre. Il est donc agréable de pouvoir lui faire confiance.

Normalement, on aurait dû l'utiliser en conception mais il n'est pas interdit de le faire au moment du codage (c'est du **remaniement de code**). Il est par contre important de faire remonter les modifications dans le diagramme de classes et de séquence afin de conserver une **cohérence**. Pour rappel, on applique un processus de développement **intératif et incrémental**.



Déclaration de la classe De

Fichier : de.h

```
#ifndef _DE_H
#define _DE_H

class De
{
  private:
    int nbFaces;
    int valeur;

  public:
    De(int nbFaces = 6);
    ~De();
    int getValeur();
    void lancer();
};

#endif
```

Variables

Méthodes

Définition de la classe De

Fichier : de.cpp

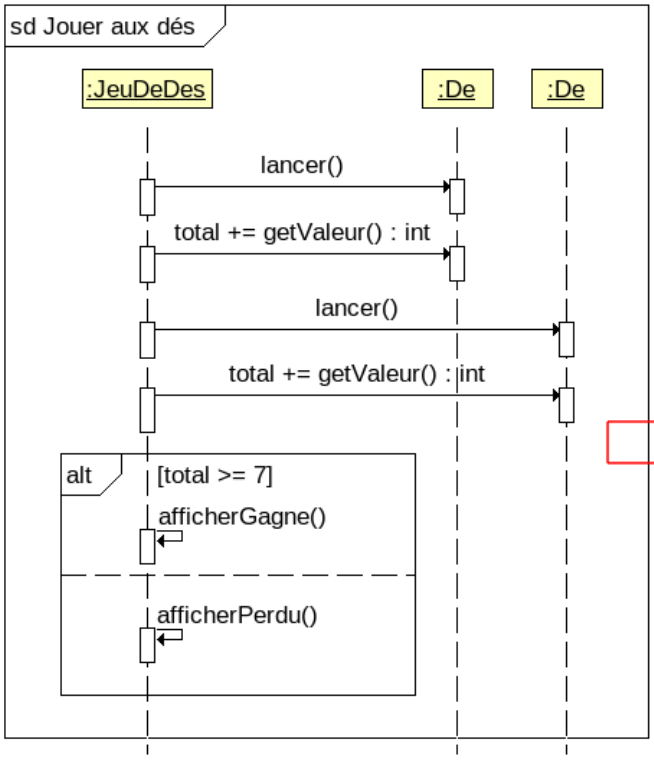
```
#include "De.h"

De::De(int nbFaces)
{
  this->nbFaces = nbFaces;
  srand(time(NULL));
}

De::~De() {}

int De::getValeur()
{
  return valeur;
}

void De::lancer()
{
  valeur = 1 + (int)((float)nbFaces *
rand() / (RAND_MAX + 1.0) );
}
```



Extrait du fichier : JeuDeDes.cpp

Définition de la méthode jouer()

```
#include "JeuDeDes.h"
#include "De.h"

…

void JeuDeDes::jouer()
{
  int total = 0;

  des[0]->lancer();
  total += des[0]->getValeur();
  des[1]->lancer();
  total += des[1]->getValeur();

  if(total >= 7)
        cout << "Gagné !" << endl;
  else  cout << "Perdu !" << endl;
}
```