# Compte-rendu Projet Codage et Cryptographie

Perrin Matthieu - Gallay Jules

Avril 2020

# Sommaire

# 1 Présentation

Ce projet consiste à coder et décoder un message à l'aide d'une clef en se basant sur la technique du Solitaire. Nous devons implémenter cette clef. Celle-ci sera différente à chaque fois que l'on veut coder un nouveau message pour assurer une sécurité au système. Notre flux de clefs sera obtenu après la réalisation de 5 opérations à partir d'un jeu de 54 cartes. Une fois la clef obtenue, le codage/décodage par somme est utilisé.

# 2 Description des classes

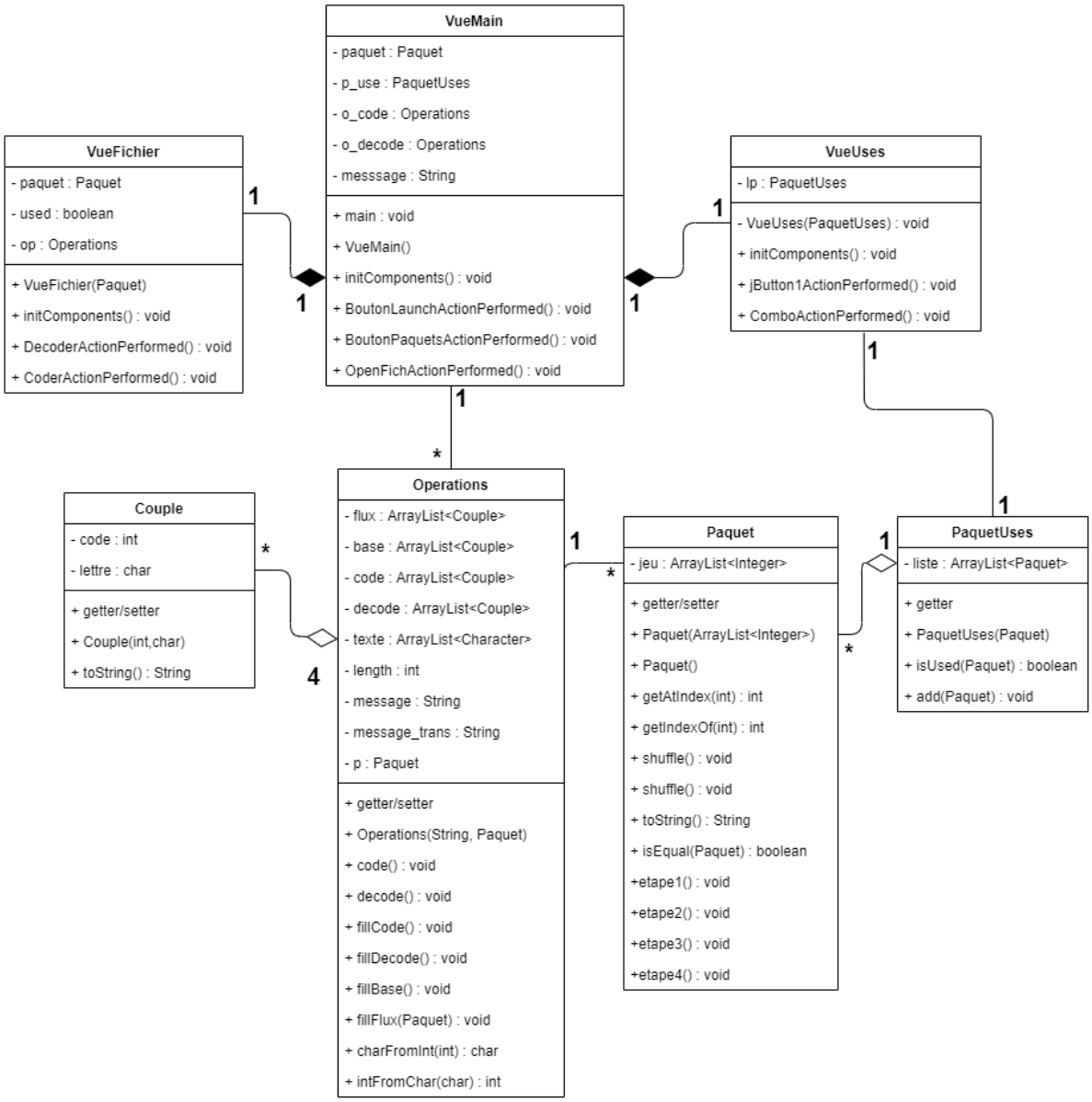
Pour ce faire, nous avons besoin d'une classe, que l'on va nommer **Paquet**, qui va réaliser les opérations pour obtenir le flux de clefs. Il s'agit de 5 opérations distinctes qui vont s'enchainer dans un ordre précis. Une deuxième classe **PaquetUses** va nous permettre d'assurer la sécurité mentionnée ci-dessus afin de ne pas avoir deux fois une même clef.

Pour facilité le lien lettre/code, nous utilisons une classe **Couple** qui va nous permettre de récupérer une lettre et le code correspondant.

La classe **Operations** va nous permettre de réaliser toutes les opérations que ce soit le codage ou le décodage du texte.On peut y retrouver les fonctions qui vont récupérer le flux, la base, le message codé et décodé mais aussi deux fonctions qui transforment un chiffre en caractère et inversement.

Enfin nous avons les classes **VueMain**, **VueUses** et **VueFichier** qui correspondent à notre interface graphique. On peut tester toutes nos fonctions depuis cette interface que ce soit par le biais d'un fichier texte ou simplement en inscrivant notre message dans une zone de texte.

## 2.1 Diagramme de classe



## 2.2 Classe « Paquet »

La classe **Paquet** possède un seul attribut qui est une ArrayList de type Integer et une fonction get() permet de récupérer cette liste. On retrouve deux constructeurs, un par défaut et un autre avec une ArrayList en paramètre. Les deux constructeurs initialisent la liste d'entier avec des chiffres allant de 1 à 54. Ces chiffres représentent les cartes du paquet selon l'ordre du Bridge.

Des accesseurs permettent d'accéder à la liste mais aussi aux éléments de cette liste ainsi qu'à leurs positions. Une méthode ToString affiche les éléments de la liste et une méthode isEqual détermine si le paquet passé en paramètre et l'objet courant sont dans le même ordre.

Enfin nous retrouvons les fonctions qui permettent d'effectuer les différentes opérations pour obtenir le flux de clefs.

Etape 1 :
*Recul du joker noir d'une position : Vous faites reculer le joker noir d'une place (vous le permutez avec la carte qui est juste derrière lui). Si le joker noir est en dernière position il passe derrière la carte du dessus (donc en deuxième position).*

Nous l'avons codé de cette façon : On récupère la position du joker noir et on vérifie sa position, soit il n'est pas en dernière position et on le permute avec la carte d'avant, soit il l'est et alors on permute une fois avec la première carte et une autre fois avec la deuxième.

Etape 2 :
*Recul du joker rouge de deux positions : Vous faites reculer le joker rouge de deux cartes. S'il était en dernière position, il passe en troisième position ; s'il était en avant dernière position il passe en deuxième.*

Pour le code, nous avons utilisé le même procédé que l'étape 1 : on récupère la position du joker rouge, on vérifie et on permute selon la position.

Etape 3 :
*Vous repérez les deux jokers et vous intervertissez le paquet des cartes situées au-dessus du joker qui est en premier avec le paquet de cartes qui est au-dessous du joker qui est en second. Dans cette opération la couleur des jokers est sans importance.*

Le code consiste à récupérer les positions des 2 jokers puis de remplir une liste de stockage selon un ordre : d'abord les cartes en dessous du deuxième joker, ensuite les cartes entre les deux jokers pour les cartes au dessus du premier joker.

Etape 4 :
*Coupe simple déterminée par la dernière carte : vous regardez la dernière carte et vous évaluez son numéro selon l'ordre du Bridge : trèfle-carreau-cœur-pique et dans chaque couleur as, 2, 3, 4, 5, 6, 7, 8, 9, 10, valet, dame et roi (l'as de trèfle a ainsi le numéro 1, le roi de pique a le numéro 52). Les jokers on par convention le numéro 53. Si le numéro de la dernière carte est n vous prenez les n premières cartes du dessus du paquet et les placez derrière les autres cartes à l'exception de la dernière carte qui reste la dernière.* Ici nous réutilisons une liste de stockage que l'on remplit d'abord avec les cartes au dessus du joker puis avec celle en dessous.

Etape 5 :
*Lecture d'une lettre pseudo-aléatoire : Vous regardez la numéro de la première carte, soit n ce numéro. Vous comptez n cartes à partir du début et vous regardez la carte à laquelle vous êtes arrivé (la n + 1- i'eme), soit m son numero. Si c'est un jokers vous refaites une opération complète de mélange et de lecture (les points 1-2-3-4-5). Si m dépasse 26 vous soustrayez 26. Au nombre entre 1 et 26 ainsi obtenu est associée une lettre qui est la lettre suivante dans du flux de clefs.*

Cette étape est réalisée dans la classe **Opérations** avec la fonction fillFlux qui remplit la liste avec le flux obtenu suite au différentes étape du paquet. Pour cela on utlise un paquet auquel on applique l'étape 1, 2, 3 et 4 le nombre de fois équivalent à la taille du texte.

## 2.3 Classe « PaquetUses »

Elle comporte un attribut ArrayList de Paquet qui nous sert à ajouter les paquets utilisés afin de ne pas les réutiliser après. Un accesseur permet de récupérer cette liste et une méthode nous permet d'ajouter un paquet à la liste.

Le constructeur crée une liste dans laquelle on ajoute le jeu du paquet en paramètre.

La fonction boolean nous sert à comparer un paquet avec ceux de la liste afin de savoir si il est déjà utilisé.

## 2.4 Classe « Couple »

Cette classe possède deux attributs(code et lettre) que l'on peut récupérer avec deux fonctions get et définir dans un constructeur. Une méthode ToString permet d'afficher la lettre et le code correspondant, fonction que l'on utilisera plus tard.

## 2.5 Classe « Opérations »

La classe opérations possèdent toutes les fonctions qui vont coder le message inscrit, décoder le message, récupérer la clef, convertir un int en caractère et inversement, ... On a plusieurs attributs : 4 ArrayList de Couple pour le flux (qui correspond à la clef), la base (le texte que l'on veut coder), le code (le message codé) et le decode (le message décodé). Une ArrayList de character va nous permettre d'isoler chaque caractère du texte pour ensuite les traiter avec la clef. Enfin on utilise un attribut String pour le message de base, un autre pour le message transformé, un attribut Paquet et un int pour la taille du texte.

Le constructeur possède 2 paramètres : un String qui correspond au message et un Paquet pour la clef. On va récupérer les informations nécessaire puis traiter le message. Pour se faire, le message va être normalisé, les accents sont supprimés, les majuscules vont être remplacées par des minuscules et les exceptions seront remplacées par des combinaisons particulières : par exemple le chiffre 1 est remplacé par 'uuuu'. Les autres caractères spéciaux ne sont pas prix en compte et seront supprimés au décodage. Enfin on récupère chaque caractère du message traité que l'on place dans la liste de Character.

Ensuite on retrouve 2 fonctions importantes, intFromChar et charFromInt. Ces deux fonctions permettent de convertir un int en caractère et inversement, par exemple le chiffre 1 correspond à la lettre a, la lettre e correspond au chiffre 5.

charFromInt :

Nous avons aussi les méthodes de remplissage des listes avec les différents messages. La fonction fillBase() remplit la liste base avec le texte que l'on veut coder. On récupère chaque caractère avec son code correspondant grâce à la fonction intFromChar et on les met dans un couple. Le couple est ensuite ajouté à la liste.

La méthode fillCode() remplit la liste avec le message codé, c'est donc ici que le message se code avec le flux. On récupère chaque nouvelle valeur grâce à l'addition du code du texte de base avec le code du flux correspondant. On convertit ensuite cette valeur afin d'obtenir le caractère codé. Le couple ensuite complété est ajouté à la liste.

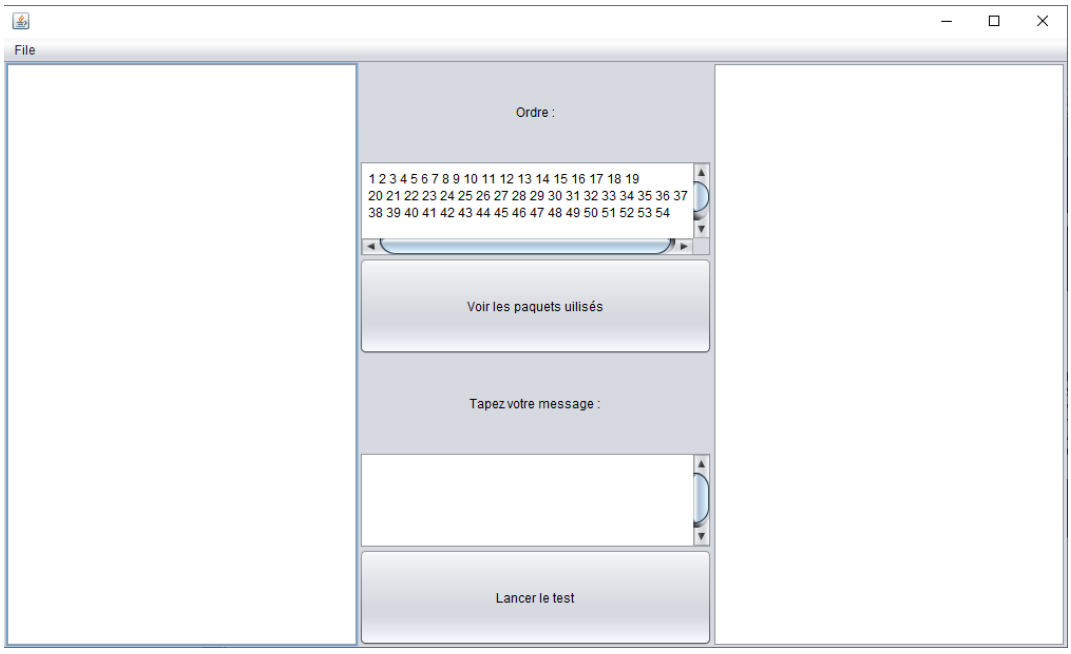
Enfin la fonction fillDecode() remplit la liste avec le message décodé. C'est la méthode inverse à fillCode() avec une soustraction à la place de l'addition.

## 2.6 Classe « VueMain »

La classe VueMain nous sert d'application graphique pour utiliser les fonctions précédentes. Elle possède 5 attributs : un Paquet(pour le flux de clef), un PaquetUses(pour stocker les paquets utilisés), 2 Opérations(un pour le codage du message, un autre pour le décodage) et un String pour le message que l'on veut traiter.

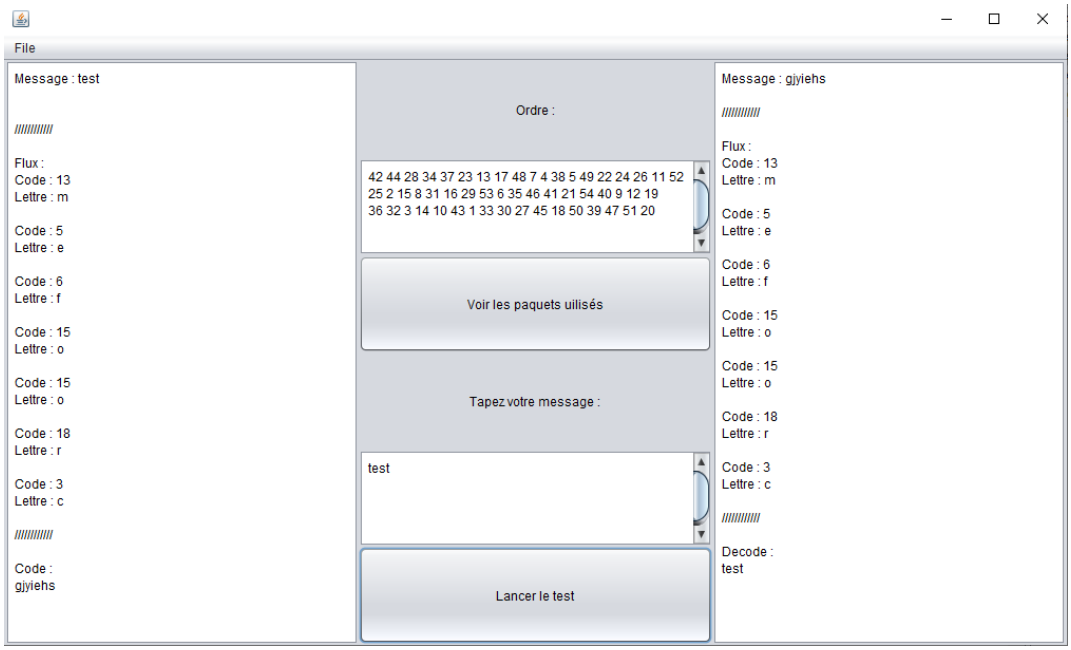
Le constructeur permet d'initialiser le Paquet, le PaquetUses et d'afficher l'application. Il va aussi afficher la clef obtenu au lancement.

Voici ce qu'on obtient au lancement du programme :




Les opérations sont lancées à partir du moment où l'on clique sur le bouton "lancer le test". Le paquet est alors mélangé puis s'il a déjà été utilisé alors le traitement ne s'effectue pas. Sinon le paquet est ajouté à la liste PaquetUses et le codage peut alors se faire.

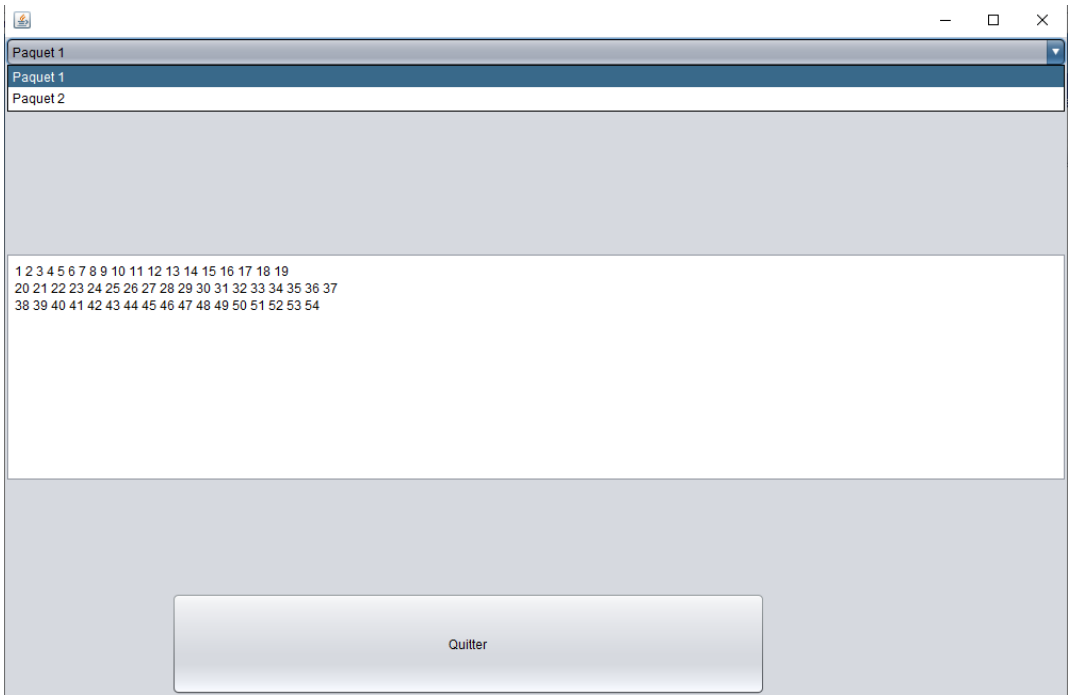
Le texte que l'on veut coder est récupéré puis traité avec l'opération code. On affiche le message codé et on le récupère pour ensuite le décoder. Ce message décodé est aussi affiché.




Depuis ce menu, nous avons accès aux 2 autres vues : VueFichier et VueUses.

## 2.7 Classe « VueUses »

Cette classe permet de voir toutes les combinaisons de clef que l'on a obtenu. On a un attribut qui est la liste PaquetUses que l'on utilise dans VueMain. Le constructeur va afficher la Vue et créer une ComboBox permettant de sélectionner une clef parmi celle déjà utilisée. Lorsque l'on sélectionne une clef, sa combinaison est affichée.




## 2.8 Classe « VueFichier »

Dans cette classe on y retrouve un Paquet, celui qu'on utilise dans l'application principale, et une opération. Le constructeur permet d'afficher la Vue.

On inscrit ensuite le nom du fichier que l'on veut coder et on appuye sur le bouton coder, si il n'existe pas alors un message d'erreur s'affiche. Sinon le traitement se fait avec la récupération du texte et l'opération de codage comme pour la méthode traditionelle. Le message transformé est inscrit dans un fichier que l'on crée pour l'occasion. Des messages d'erreurs surviennent si le fichier ne peut pas se créer ou si le message ne parvient pas à s'inscrire dedans.

***Remarque :*** *Les fichiers textes se trouvent dans le dossier Solitaire, à la racine du dossier src.*

Pour décoder le message, c'est le même traitement sauf que c'est la fonction decode qui est utilisée.

## 2.9 Conclusion

Ce projet et ce cours nous ont permis d'avoir une vision plus claire sur les avantages et les inconvénients de la méthode du Solitaire. Nous avons réussi sans complication à programmer le flux de clef qui était le plus intéressant à réaliser. De plus, nous avons pris une certaine liberté sur la compréhension du sujet sur le traitement des chiffres et des caractères spéciaux. L'interface graphique permet de tester toutes les fonctionnalités programmées et est assez explicite. La fonctionnalité de coder des fichiers texte a été programmée bien après le reste mais son implémentation n'a pas été un problème.