encodeCesar

```
public static String encodeCesar(String s, int k)
```

Applique le code César sur la chaîne grâce à la clef.

Pour chaque lettre de la chaîne, on ramène son code ASCII à 0 on ajoute la clef, modulo 26 pour rester entre 0 et 25, les lettres de l'alphabet correspondant à un chiffre ; puis on ajoute de nouveau le code ASCII pour revenir aux lettres, on concatène le résultat dans une nouvelle chaîne.

Paramètres :
s – Chaîne à chiffrer grâce au code César
k – Clef désignée par un entier et utilisée dans le code César

decodeCesar

```
public static String decodeCesar(String s, int k)
```

Permet de retrouver la chaîne d'origine avant l'application du code César grâce à la clef utilisée pour chiffrer.

L'algorithme est le même que celui du chiffrage, la différence étant qu'au lieu d'ajouter la clef, on la soustrait.

Paramètres :
s – Chaîne à déchiffrer grâce au code César
k – Clef désignée par un entier et utilisée dans le code César

Sortie console :

```
Algo Cesar :
BONJOUR TOUT LE MONDE // avant
ERQMRXU=WRXW=OH=PRQGH // chiffré
BONJOUR:TOUT:LE:MONDE // déchiffré
```

encodeVigenere

```
public static String encodeVigenere(String s, String k)
```

Applique le code Vigenère sur la chaîne grâce à la clef.

L'algorithme est sensiblement le même que celui du code César, à la différence que le clef change pour chaque lettre. Pour rendre l'opération plus simple, on peut créer une clef de la même taille que notre chaîne à chiffrer.

Paramètres :
s – Chaîne à chiffrer grâce au code Vigenère
k – Clef désignée par un entier et utilisée dans le code Vigenère

decodeVigenere

```
public static String decodeVigenere(String s, String k)
```

Permet de retrouver la chaîne d'origine avant l'application du code Vigenère grâce à la clef utilisée pour chiffrer.

L'algorithme est le même que celui du chiffrage, la différence étant qu'au lieu d'ajouter la clef, on la soustrait.

Paramètres :
s – Chaîne à déchiffrer grâce au code Vigenère
k – Clef désignée par un entier et utilisée dans le code Vigenère

Sortie console :

```
Algo Vigenere :
J adore ecouter la radio toute la journee // avant
V=UVWHY=IOIMBUL=PM=LSLYI=XAOLM=BU=NAOJVUY // chiffré
J:ADORE:ECOUTER:LA:RADIO:TOUTE:LA:JOURNEE // déchiffré
```

encodeHill

```
public static String encodeHill(String s, int[][] k)
```

Applique le code Hill sur la chaîne grâce à la clef.

La clef doit être une matrice carrée inversible ayant un déterminant compris entre 1 et 26 (taille de l'alphabet).
Le principe est de tout d'abord transformer notre chaîne en une longueur divisible par le taille de la matrice clef, on peut par exemple pour ça ajouter des X en fin de chaîne. Pour tous les N lettres de la chaîne, N étant la taille de la matrice clef, multiplier la matrice par le vecteur de taille N représentant les N lettres de la chaîne en entier, appliquer le modulo 26 pour rester dans l'alphabet, transformer en ASCII et concaténer dans la chaîne finale.

Paramètres :
s – Chaîne à chiffrer grâce au code Hill
k – Clef désignée par une matrice d'entier et utilisée dans le code Hill

decodeHill

```
public static String decodeHill(String s, int[][] k)
```

Permet de retrouver la chaîne d'origine avant l'application du code Hill grâce à la clef utilisée pour chiffrer.

Pour déchiffrer nous pouvons simplement appliquer de nouveau le chiffrement Hill sur la chaîne chiffrée mais en utilisant une nouvelle clef, cette nouvelle clef étant la matrice inverse modulaire 26 de la clef d'origine. Pour plus de simplicité d'implémentation, cela a été uniquement fait pour les

matrices 2x2, car le code pour une matrice NxN étant assez complexe, cela n'apportais pas vraiment d'intérêt supplémentaire ; car ici le but est plus de comprendre le principe et l'implémentation de la technique cryptographique de Hill plutôt que les mathématiques derrières.

Paramètres :
s – Chaîne à déchiffrer grâce au code Hill
k – Clef désignée par une matrice d'entier et utilisée dans le code Hill

Sortie console :

```
Algo Hill :
Bonjour // avant
HBXQSQNG // chiffré
BONJOURX // déchiffré
```

encodeAES

```
public static String encodeAES(String s, SecretKey k) throws Exception
```

Applique l'algorithme de chiffrement AES sur la chaîne grâce à la clef.

En utilisant la classe Cipher on peut initialiser l'objet pour AES en mode chiffrement, puis l'appliquer sur notre chaîne grâce à notre clef. La clef est générée de manière aléatoire grâce à la classe KeyGenerator sur 128 bits. Il faut donc la conserver pour pouvoir déchiffrer le message plus tard.

Paramètres :
s – Chaîne à chiffrer grâce à l'algorithme AES
k – Clef secrète utilisée dans l'algorithme AES

decodeAES

```
public static String decodeAES(String s, SecretKey k) throws Exception
```

Applique l'algorithme de déchiffrement AES sur la chaîne grâce à la clef.

En utilisant la classe Cipher on peut initialiser l'objet pour AES en mode déchiffrement, puis l'appliquer sur notre chaîne grâce à notre clef. La clef doit donc être la même que celle utilisée pour le chiffrement.

Paramètres :
s – Chaîne à déchiffrer grâce à l'algorithme AES
k – Clef secrète utilisée dans l'algorithme AES

Sortie console :

```
AES :
BONJOUR TOUT LE MONDE // avant
I4Axs4KlnoAA+bf1yuhT+MwRGUllFLiLt0M1kWR0C9A= // chiffré
BONJOUR TOUT LE MONDE // déchiffré
```

encodeRSA

```
public static String encodeRSA(String s, PublicKey k) throws
Exception
```

Applique l'algorithme de chiffrement RSA sur la chaîne grâce à la clef publique.

En utilisant la classe Cipher on peut initialiser l'objet pour RSA en mode chiffrement, puis l'appliquer sur notre chaîne grâce à notre clef public. La clef publique est générée de manière aléatoire grâce à la classe KeyPairGenerator sur 2048 bits. Il faut ensuite utiliser la classe KeyPair pour obtenir notre paire de clefs, et obtenir notre clef publique.

Paramètres :
s – Chaîne à chiffrer grâce à l'algorithme RSA
k – Clef publique utilisée dans l'algorithme RSA

decodeRSA

```
public static String decodeRSA(String s, PrivateKey k) throws
Exception
```

Applique l'algorithme de déchiffrement RSA sur la chaîne grâce à la clef publique.

En utilisant la classe Cipher on peut initialiser l'objet pour RSA en mode déchiffrement, puis l'appliquer sur notre chaîne grâce à notre clef privée. La clef privée est générée de manière aléatoire grâce à la classe KeyPairGenerator sur 2048 bits. Il faut ensuite utiliser la classe KeyPair pour obtenir notre paire de clefs, et obtenir notre clef privée. On l'obtient en même temps que la publique, il ne faut pas régénérer de nouvelles paires.

Paramètres :
s – Chaîne à déchiffrer grâce à l'algorithme RSA
k – Clef privée utilisée dans l'algorithme RSA

Sortie console :

```
RSA :
BONJOUR TOUT LE MONDE // avant
R8l+/1YVO9khCo35nihf30x4uG5zggPRuKlw6/81EAriALSedDbV+vvHTsVQ8GVXCFw3nEMvUPdnPps/
gi1xv89pDs0FEWLwlzfCNPgQ+PND8Asrfu6gU2Lfho0w21mDTuW/
P7Y+e3WJ2pAlknIfLbKI92nK8aRxM1ztStiQNO4d9x7DIx0xqGKoZWtXXbk4CG79/
CQ31V5PDFQhbwYmMWighLcPH2PVJ8lpSaCLbuY+hub9+N4B/HikQPfanQFTLBsGJjL+DSoPJEg+efSfX/
YjU5ddeQade+Oi0Z6W5azN33Aeb7PYHtJNj7DyuNfV24Ov12XMZqh1aCWt1SaglQ== // chiffré
BONJOUR TOUT LE MONDE // déchiffré
```