

Tous supports autorisés.

L'objectif du TP est de développer un micro-système de cryptage de message.

Nous allons considérer trois types de code de cryptage :

- un code à clef secrète,
- un code à clef aléatoire,
- un code à crans (appelé également code de César).

Éléments fournis. Un ensemble de classes de test vous sont fournies, elles sont disponibles :

- dans le répertoire `/home/depot/1A/POO/TP_NOTE`

Éléments à rendre. Avant la fin de la séance, vous devez rendre votre travail et recopiant les classes dans le répertoire : `/home/depot/TP-NOTES/POO-<horaire>/<login>`. L'accès à ce répertoire sera automatiquement interdit à la fin de la séance.

Tests et Compilation. Les classes de test fournies vous permettront de tester les différentes classes que vous écrierez. Ces classes vous afficheront quels sont les tests qui ne s'exécutent pas correctement et vous donneront à titre indicatif un score pouvant s'apparenter à votre note de TP. **À la fin de la séance de TP, il est impératif que toutes vos classes compilent ! Un manquement à cette règle sera sévèrement sanctionné.**

Rappel de dernière minute. Il est bien sûr interdit de s'inspirer (plus ou moins) du travail d'un autre élève (voisin ou d'un autre groupe). Nous disposons d'un outil capable de détecter le plagiat. Il sera utiliser pour s'assurer que votre travail est une œuvre originale.

► **Question 1.** `crypto/Code.java`.

Écrivez une classes abstraite `crypto.Code` dont les profils des méthodes sont les suivants :

- une méthode `getName()` sans paramètre et dont le type de retour est `String`. Cette méthode retournera le nom du code de cryptage utilisé.
- une méthode abstraite `encode()` prenant un paramètre de type `String` et dont le type de retour est `String`. L'implémentation concrète de cette méthode dans les sous-classes retournera en résultat la version cryptée du message passé en paramètre.
- une méthode abstraite `decode()` prenant un paramètre de type `String` et dont le type de retour est `String`. L'implémentation concrète de cette méthode dans les sous-classes retournera en résultat la version décryptée du message passé en paramètre.

Cette classe devra conserver le nom du code de cryptage utilisé. Vous ajouterez donc le/les attributs nécessaire(s) ainsi qu'un constructeur.

Réponse

```

1 package crypto;
2
3 public abstract class Code {
4     protected String name;
5
6     public Code(String name) {
7         this.name = name;
8     }
9
10    public String getName() {
11        return this.name;
12    }
13
14    public abstract String encode(String s);
15
16    public abstract String decode(String s);
17
18 }
19

```

Fin réponse

✓ **Validation 1.** Testez votre implémentation.

Compilez et exécutez les classes de test fournies nommées `test.TestCode1` et `test.TestCode2`. Si nécessaire, corriger les différentes erreurs dans votre code.

Nous nous intéressons à la conception et la réalisation d'un code à clef secrète. Un tel code utilise une autre chaîne de caractères, que l'on appelle clef, pour crypter/décrypter les messages. La même clef étant utilisée le cryptage et le décryptage, ce type de code est appelé cryptage symétrique. Il fonctionne de la manière suivante :

- À chaque caractère du message, on associe une valeur entière ;
- À chaque caractère de la clef, on associe également une valeur entière ;
- On aligne l'un en dessous de l'autre les caractères du message et ceux de la clef (on répète si nécessaire la clef plusieurs fois) ;
- On calcule pour chaque couple de caractères la somme des valeurs entières en retirant 27 lorsque celle-ci est égale ou dépasse 27 ;
- On convertit la valeur obtenue pour chaque couple de caractères, en un caractère.
- La concaténation des caractères obtenus donne la version cryptée du message.

Message	T	R	O	P		D	E		S	E	C	R	E	T
Valeur entière : m	19	17	14	15	26	3	4	26	18	4	2	17	4	19
Clef	E	S	I	A	L	E	S	I	A	L	E	S	I	A
Valeur entière : k	4	18	8	0	11	4	18	8	0	11	4	18	8	0
Valeur : $(m + k) \bmod 27$	23	8	22	15	10	7	22	7	18	15	6	8	12	19
Message crypté	X	I	W	P	K	H	W	H	S	P	G	I	M	T

Le décryptage se déroule de la même manière sauf que l'on effectue la différence entre la valeur du caractère du message m , à laquelle on a ajouté 27, et la valeur de la clef k . Si la valeur obtenue est égale ou dépasse 27, on retire 27.

On rappelle que :

- l'expression `a % b` en Java calcule a modulo b (reste de la division entière de a par b).
- que la classe `String` fournit une méthode d'instance `char charAt(int index)` qui retourne le caractère se trouvant à la position `index` dans une chaîne de caractères.
Par exemple, `String s = "ESIAL"; s.charAt(2)` retourne le caractère 'I'.

► **Question 2.** `crypto/KeyBased.java`.

Écrivez une interface `crypto.KeyBased` dont définit les méthodes suivantes :

- une méthode `getSecretKey` sans paramètre dont le type de retour est une chaîne de caractères (type `String`).
- une méthode `getKeyLength` sans paramètre dont le type de retour est une valeur entière (type `int`);

Réponse

```

1 package crypto;
2
3 public interface KeyBased {
4
5     public String getSecretKey();
6
7     public int getKeyLength();
8
9 }

```

Fin réponse

✓ **Validation 2.** Testez votre implémentation.

La classes de test est `test.TestKeyBased`.

► **Question 3.** `crypto/SecretKeyCodeImpl.java`.

Écrivez une classe `crypto.SecretKeyCodeImpl` qui réalise l'interface `crypto.KeyBased` et hérite de la classe `crypto.Code`.

Cette classe doit :

- conserver une chaîne de caractère correspondant à la clef de cryptage/décryptage;
- définir les méthodes de l'interface `crypto.KeyBased`;
- définir les méthodes (qui étaient abstraites) de la classe `crypto.Code`;
- définir un constructeur dont le paramètre est la clef de cryptage/décryptage (une chaîne de caractères).

Ce constructeur affectera la chaîne `Algorithm using Secret Key` comme nom d'algorithme de codage. Afin de réaliser les algorithmes de cryptage/décryptage, vous utiliserez les méthodes définies dans la classe `crypto.Utils` fournie :

- `static int charToInt(char c)` qui convertit un caractère (intervalle de 'A' à 'Z' avec ' ' en plus) en une valeur entière comprise entre 0 et 26;
- `static char intToChar(int i)` qui convertit une valeur entière (comprise en 0 et 26) en un caractère.

Réponse

```

1 package crypto;
2
3 public class SecretKeyCodeImpl extends Code implements KeyBased {
4
5     private String secretKey;
6
7     public SecretKeyCodeImpl(String secretKey) {
8         super("Algorithm using Secret Key");
9         this.secretKey = secretKey;
10    }
11
12    public String getSecretKey() {
13        return this.secretKey;
14    }
15
16    public int getKeyLength() {
17        return this.secretKey.length();
18    }
19
20    public String encode(String message) {
21        String result = "";
22        for (int i=0; i<message.length(); i++) {
23            result += Utils.intToChar((Utils.charToInt(message.charAt(i))
24                + Utils.charToInt(this.secretKey.charAt(i % this.secretKey.length())) % 27);
25        }
26        return result;

```

```

27     }
28
29     public String decode(String message) {
30         String result = "";
31         for (int i=0; i<message.length(); i++) {
32             result += Utlis.intToChar( ( Utlis.charToInt(message.charAt(i)) + 27
33                                     - Utlis.charToInt(this.secretKey.charAt(i % this.secretKey.length())) ) % 27 );
34         }
35         return result;
36     }
37 }

```

Fin réponse

✓ **Validation 3.** Testez votre implémentation.

Les classes de test sont `test.TestSecretKeyCodeImpl1` et `test.TestSecretKeyCodeImpl2`.

Nous nous intéressons maintenant à la conception et la réalisation d'un code à clef aléatoire. Un tel code utilise le **même algorithme de cryptage/décryptage** que l'algorithme à clef secrète. L'unique différence est que la clef n'est pas fournie par l'utilisateur, mais **générée aléatoirement**.

► **Question 4.** `crypto/RandomKeyCodeImpl.java`.

Écrivez une classe `crypto.RandomKeyCodeImpl` qui hérite de la classe `crypto.SecretKeyCodeImpl`.

Cette classe doit :

- définir une **méthode de classe** `generateKey` qui prend en paramètre une valeur entière indiquant la taille (en nombre de caractères) de la clef aléatoire à générer. Le résultat de cette méthode sera une chaîne de caractères (type `String`) égale à la clef générée.
- définir un constructeur dont l'unique paramètre est la taille de la clef aléatoire à générer et qui sera donc utilisée pour le cryptage/décryptage. Ce constructeur affectera la chaîne `Algorithm using Random Key` comme nom d'algorithme de codage.

Il faut noter que :

- Par héritage, cette classe implémente le contrat défini dans l'interface `crypt.KeyBased`. Ainsi, les méthodes définies dans cette interface doivent retourner les informations concernant la clef secrète qui a été générée aléatoirement.

Afin de réaliser la méthode de génération de clef aléatoire, vous utiliserez la méthode définie dans la classe `crypto.Utlis` fournie :

- `static char randomChar()` qui retourne un caractère générée de manière aléatoire (dans l'intervalle 'A' à 'Z' avec ' ' en plus).

Réponse

```

1 package crypto;
2
3 public class RandomKeyCodeImpl extends SecretKeyCodeImpl {
4     public RandomKeyCodeImpl(int keyLength) {
5         super(generateKey(keyLength));
6         this.name = "Algorithm using Random Key";
7     }
8
9     public static String generateKey(int keyLength) {
10        String result = "";
11        for (int i=0; i<keyLength; i++) {
12            result += Utlis.randomChar();
13        }
14        return result;
15    }
16 }

```

Fin réponse

✓ **Validation 4.** Testez votre implémentation.

Les classes de test sont `test.TestRandomKeyCodeImpl1` et `test.TestRandomKeyCodeImpl2`.

Nous nous intéressons maintenant à la conception et la réalisation d'un code à cran (dit code de César). Ce code fonctionne par décalage circulaire. Chaque lettre du message est remplacée par la lettre apparaissant n crans plus loin dans l'alphabet, et ce, de façon cyclique : 'Y' décalée de 4 crans devient 'B'). Vous supposerez que le caractère ' ' correspondant à l'espace est située à la fin de l'alphabet.

Message	E	S	I	A	L
Message crypté (1 cran)	F	T	J	B	M
Message crypté (2 crans)	G	U	K	C	N
Message crypté (13 crans)	R	E	V	N	Y

Le décryptage se déroule de la même manière sauf que l'on effectue un décalage de $(27 - nbcrans)$.

► **Question 5.** `crypto/CesarCodeImpl.java`.

Écrivez une classe `crypto.CesarCodeImpl` qui hérite de la classe abstraite `crypto.Code`. Elle se caractérise par le nombre de crans de décalage.

Cette classe doit :

- définir les méthodes (qui étaient abstraites) de la classe `crypto.Code`;
- définir un constructeur dont l'unique paramètre est une valeur entière (type `int`) qui correspond au nombre de crans (décalage) à utiliser par le code. Ce constructeur affectera la chaîne `Cesar Algorithm` comme nom d'algorithme de codage. À noter que le nombre de crans est obligatoirement positif ou nul.
- définir une méthode `getShiftLevel` sans paramètre dont le type de retour est une valeur entière (type `int`) qui correspond au nombre de crans utilisé par l'algorithme de cryptage/décryptage.

Afin de réaliser les algorithmes de cryptage/décryptage, vous utiliserez la méthode définie dans la classe `crypto.Utils` fournie :

- `static int charToInt(char c)` qui convertit un caractère (intervalle de 'A' à 'Z' avec ' ' en plus) en une valeur entière comprise entre 0 et 26;
- `static char intToChar(int i)` qui convertit une valeur entière (comprise en 0 et 26) en un caractère.

Réponse

```

1 package crypto;
2
3 public class CesarCodeImpl extends Code {
4     private int shiftLevel;
5
6     public CesarCodeImpl(int shiftLevel) {
7         super("Cesar Algorithm");
8         this.shiftLevel = shiftLevel;
9     }
10
11     public String encode(String message) {
12         String result = "";
13         for (int i=0; i<message.length(); i++) {
14             char c = Utils.intToChar((Utils.charToInt(message.charAt(i)) + shiftLevel) % 27);
15             result += c;
16         }
17         return result;
18     }
19
20     public String decode(String message) {
21         this.shiftLevel = 27-this.shiftLevel;
22         String res = encode(message);
23         this.shiftLevel = 27+this.shiftLevel;
24         return res;
25     }
26
27     public int getShiftLevel() {
28         return this.shiftLevel;
29     }
30 }
31
32

```

Fin réponse

✓ **Validation 5.** Testez votre implémentation.

Les classes de test sont `test.TestCesarCodeImpl1` et `test.TestCesarCodeImpl2`.

Nous allons écrire maintenant une classe nous permettant de manipuler facilement nos primitives de cryptographie.

► **Question 6.** `crypto/CryptoMachine.java`.

Écrivez une classe `crypto.CryptoMachine`.

Cette classe doit :

- conserver une instance d'un des trois algorithmes de codage que vous venez de définir.

- fournir un constructeur sans paramètre. Par défaut l'algorithme de codage utilisé sera le code de César avec un décalage de 0 cran.
- définir une méthode `setCryptoStrategy` sans valeur de retour qui prend un algorithme de codage en paramètre (type `crypto.Code`) et le conserve.
- définir une méthode `encode` qui prend en paramètre un message (type `String`) retourne le message crypté (type `String`) en utilisant l'algorithme choisi par la méthode précédente.
- définir une méthode `decode` qui prend en paramètre un message (type `String`) retourne le message décrypté (type `String`) en utilisant l'algorithme choisi par la méthode précédente.
- définir une méthode `test` qui prend en paramètre un message (type `String`) retourne une chaîne de caractères (type `String`) constituée du message original, suivi de la chaîne ' -> ', suivi du message crypté, suivi de la chaîne ' -> ', suivi du message décrypté.
- définir une méthode `getInformation` sans paramètre dont le type de retour est une chaîne de caractères (type `String`) égale à :
 - dans le cadre d'un code à base de clef (secrète ou aléatoire) :

```
1 Algorithm using Secret Key
2 key length: 5
3 key: ESIAL
```

- dans le cadre d'un code de César :

```
1 Cesar Algorithm
2 shift level: 13
```

Réponse

```
1 package crypto;
2
3 public class CryptoMachine {
4     private Code code;
5
6     public CryptoMachine() {
7         this.code = new CesarCodeImpl(0);
8     }
9
10    public void setCryptoStrategy(Code code) {
11        this.code = code;
12    }
13
14    public String encode(String message) {
15        return this.code.encode(message);
16    }
17
18    public String decode(String message) {
19        return this.code.decode(message);
20    }
21
22    public String test(String message) {
23        String ciphered = this.encode(message);
24        return message+" -> "+ciphered+" -> "+this.decode(ciphered);
25    }
26
27    public String getInformation() {
28        String res = "";
29        res += code.getName() + "\n";
30        if (code instanceof KeyBased) {
31            KeyBased kbCode = (KeyBased) code;
32            res += "key length: " + kbCode.getKeyLength() + "\n";
33            res += "key: " + kbCode.getSecretKey();
34        } else if (code instanceof CesarCodeImpl) {
35            res += "shift level: " + ((CesarCodeImpl) code).getShiftLevel();
36        }
37        return res;
38    }
39 }
40 }
```

Fin réponse

✓ **Validation 6.** Testez votre implémentation.

Les classes de test sont `test.TestCryptoMachine1` et `test.TestCryptoMachine2`.

✓ **Validation 7.** Testez votre implémentation complète.

Afin de relancer l'ensemble des tests fournis, vous pouvez compiler et exécuter la classe de test nommée `crypto.TestAll`.

Bonne chance ! ;-)