Travail pratique #2

Ce travail doit être fait individuellement.

Notions mises en pratique : le respect d'un ensemble de spécifications, la conception et l'utilisation de méthodes (séparation fonctionnelle), et l'utilisation de la classe `String`.

1. Spécifications

Il s'agit d'écrire un programme qui permet d'encoder et de décoder des messages à partir d'une clé de cryptage donnée, comportant diverses opérations de transformation d'une chaine de caractères (le message). La section suivante présente la liste des opérations valides qui peuvent se trouver dans la clé de cryptage.

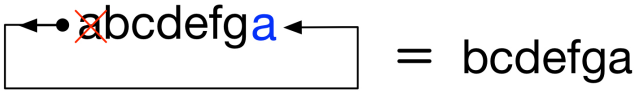
NOTE : Prenez le temps de lire attentivement la section 2 (Précisions) AVANT de commencer votre TP.

1.1 OPÉRATIONS DE TRANSFORMATION D'UNE CHAINE DE CARACTÈRES

Chaque opération est représentée par deux lettres (peu importe la casse) que l'on appellera le **code** de l'opération.

1) Rotation gauche (code = RG)

Cette opération consiste à prendre le premier caractère de la chaine (le plus à gauche), et de le placer à la fin de la chaine, comme ceci :

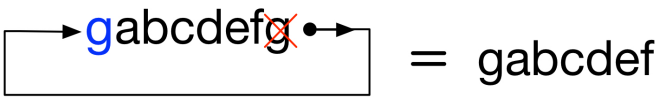


Cette opération peut être répétée entre 0 et 99 fois inclusivement, le nombre de fois étant indiqué par EXACTEMENT 2 chiffres placés après le code de l'opération. Par exemple :

Opération	Description de l'opération	chaine	Résultat de l'opération sur chaine
`RG00`	0 rotation gauche	`"abcdefg"`	`"abcdefg"`
`RG01`	1 rotation gauche	`"avertissement"`	`"vertissementa"`
`RG02`	2 rotations gauches	`"brocoli"`	`"ocolibr"`
`RG21`	21 rotations gauches	`"noir"`	`"oirn"`
`RG99`	99 rotations gauches	`"visage"`	`"agevis"`

2) Rotation droite (code = RD)

Cette opération consiste à prendre le dernier caractère de la chaine (le plus à droite), et de le placer au début de la chaine, comme ceci :



Cette opération peut être répétée entre 0 et 99 fois inclusivement, le nombre de fois étant indiqué par EXACTEMENT 2 chiffres placés après le code de l'opération. Par exemple :

Opération	Description de l'opération	chaine	Résultat de l'opération sur chaine
RD00	0 rotation droite	"abcdefg"	"abcdefg"
RD01	1 rotation droite	"avertissement"	"tavertissemen"
RD02	2 rotations droites	"brocoli"	"libroco"
RD21	21 rotations droites	"noir"	"rnoi"
RD99	99 rotations droites	"visage"	"agevis"

3) Permutation extérieure (code = PE)

Cette opération consiste à permuter (échanger) les caractères de la chaine, de l'extérieur vers l'intérieur. On peut permuter de 0 à 99 caractères, le nombre de caractères à permuter étant indiqué par EXACTEMENT 2 chiffres placés après le code de l'opération. Par exemple :

Opération	Description de l'opération	chaine	Résultat de l'opération sur chaine
PE00	0 permutation extérieure	"abcdefg"	"abcdefg"
PE01	1 permutation extérieure	"avertissement"	"tvertissemena" (échanger a et t)
PE02	2 permutations extérieures	"brocoli"	perm. #1 => "irocolb" (échanger b et i) perm. #2 => "ilocorb" (échanger r et l)
PE03	3 permutations extérieures	"brocoli"	perm. #1 => "irocolb" (échanger b et i) perm. #2 => "ilocorb" (échanger r et l) perm. #3 => "ilocorb" (échanger o et o)
PE05	5 permutations extérieures *Lorsqu'il ne reste qu'un caractère au centre de la chaine (pour une chaine de longueur impaire) la permutation recommence à l'extérieur.*	"brocoli"	perm. #1 => "irocolb" (échanger b et i) perm. #2 => "ilocorb" (échanger r et l) perm. #3 => "ilocorb" (échanger o et o) perm. #4 => "blocori" (échanger i et b) perm. #5 => "brocoli" (échanger l et r)
PE21	21 permutations extérieures *Lorsqu'il ne reste aucun caractère à permuter au centre de la chaine (pour une chaine de longueur paire) la permutation recommence à l'extérieur.*	"noir"	perm. #1 => "roin" (échanger n et r) perm. #2 => "rion" (échanger o et i) perm. #3 => "nior" (échanger r et n) perm. #4 => "noir" (échanger i et o) perm. #5 => "roin" (échanger n et r) perm. #6 => "rion" (échanger o et i) etc. perm. #21 => "roin" (résult. final)
PE99	99 permutations extérieures	"visage"	"egasiv" (résult. final)
PE07	7 permutations extérieures Ici, rien n'est effectué, car la longueur de la chaine < 2.	"o"	"o" (résult. final)

Notez que cette opération **ne peut s'effectuer que sur une chaine contenant au moins 2 caractères**. Si ce n'est pas le cas, l'opération n'est tout simplement pas appliquée, et ne modifie pas la chaine vide ou de longueur égale à 1.

4) Permutation intérieure (code = `PI`)

Cette opération consiste à permuter (échanger) les caractères de la chaine, de l'intérieur (centre) vers l'extérieur. On peut permuter de 0 à 99 caractères, le nombre de caractères à permuter étant indiqué par EXACTEMENT 2 chiffres placés après le code de l'opération. Lorsque la longueur de la chaine est impaire, la permutation intérieure commence avec les 2 caractères se trouvant de chaque côté du caractère central. Lorsque la longueur de la chaine est paire, la permutation intérieure commence avec les 2 caractères centraux. Par exemple :

Opération	Description de l'opération	chaine	Résultat de l'opération sur chaine
`PI00`	0 permutation intérieure	`"abcdefg"`	`"abcdefg"`
`PI01`	1 permutation intérieure	`"avertissement"`	`"avertsiement"` (échanger i et s)
`PI02`	2 permutations intérieures	`"brocoli"`	perm. #1 => `"brocoli"` (échanger o et o) perm. #2 => `"blocori"` (échanger r et l)
`PI03`	3 permutations intérieures	`"brocoli"`	perm. #1 => `"brocoli"` (échanger o et o) perm. #2 => `"blocori"` (échanger r et l) perm. #3 => `"ilocorb"` (échanger b et i)
`PI05`	5 permutations intérieures *Lorsqu'on a permuté le premier et le dernier caractère de la chaine, la permutation recommence au centre.*	`"brocoli"`	perm. #1 => `"brocoli"` (échanger o et o) perm. #2 => `"blocori"` (échanger r et l) perm. #3 => `"ilocorb"` (échanger b et i) perm. #4 => `"ilocorb"` (échanger o et o) perm. #5 => `"irocolb"` (échanger l et r)
`PI21`	21 permutations intérieures	`"noir"`	perm. #1 => `"nior"` (échanger o et i) perm. #2 => `"rion"` (échanger n et r) perm. #3 => `"roin "` (échanger i et o) perm. #4 => `"noir"` (échanger r et n) perm. #5 => `"nior"` (échanger o et i) perm. #6 => `"rion"` (échanger n et r) etc. perm. #21 => `"nior"` (résult. final)
`PI99`	99 permutations intérieures	`"visage"`	`"egasiv"` (résult. final)
`PI07`	7 permutations intérieures Ici, rien n'est effectué, car la longueur de la chaine < 2.	`"o"`	`"o"` (résult. final)

5) Inversion (`IV`)

Cette opération consiste à inverser un nombre donné `n` de caractères en début de chaine, et en fin de chaine. L'opération commence toujours par inverser les `n` caractères en début de chaine, avant d'inverser les `n` caractères en fin de chaine. Le nombre `n` doit être entre 0 et 99 inclusivement, et il est indiqué par EXACTEMENT 2 chiffres placés après le code de l'opération. Notez que si `n` est plus petit ou égal à 1 ou s'il est supérieur ou égal à la longueur de la chaine, la chaine demeure inchangée. Par exemple :

Opération	Description de l'opération	chaine	Résultat de l'opération sur chaine
`IV00`	Inversion de 0 caractère	`"abcdefg"`	`"abcdefg"`

IV01	Inversion de 1 caractère	"**a**bcdef**g**"	"**a**bcdef**g**"
IV02	Inversion de 2 caractères	"**ab**cdef**g**"	Inv. au début => **ba**cdefg Inv. à la fin => bacde**gf** (résultat final)
IV03	Inversion de 3 caractères	"**abc**def**g**"	Inv. au début => **cba**defg Inv. à la fin => cbad**gfe** (résultat final)
IV04	Inversion de 4 caractères *Ici, les caractères à inverser au début et à la fin de la chaine se chevauchent => toujours commencer par l'inversion au début.*	"**abcd**efg"	Inv. au début => **dcba**efg Inv. à la fin => dcb**gfea** (résultat final)
IV05	Inversion de 5 caractères	"abcdefg"	Inv. au début => **edcba**fg Inv. à la fin => ed**gfabc** (résultat final)
IV06	Inversion de 6 caractères	"abcdefg"	Inv. au début => **fedcba**g Inv. à la fin => f**gabcde** (résultat final)
IV07	Inversion de 7 caractères	"abcdefg"	Inv. au début => **gfedcba** Inv. à la fin => **abcdefg** (résultat final)
IV08	Inversion de 8 caractères *Lorsque le nombre de caractères à inverser est > que la longueur de la chaine, l'opération ne modifie pas la chaine.*	"abcdefg"	"abcdefg"

Notes générales sur les opérations

Notez que si, par exemple, `NN` est égal à la longueur de la `chaine`, le résultat de l'application de `RGNN` sur la `chaine` est égal à `chaine`. Par exemple, l'application de `RG06` ou `RG12` ou `RG24` ou `RG72` sur la chaine `"visage"` donne la même chaine. Il s'ensuite que l'application de `RG07` ou `RG13` ou `RG25` ou `RG73` sur la chaine `"visage"` est équivalent à l'application de `RG01` sur cette même chaine, etc. Dans ce dernier cas, par exemple, il suffirait d'appliquer l'opération une seule fois plutôt que 73 fois pour obtenir le même résultat. Un principe similaire s'applique aussi pour certaines des autres opérations. **Tenez-en compte pour optimiser votre code**.

1.2 CLÉ DE CRYPTAGE ET ENCODAGE D'UN MESSAGE

Clé de cryptage

Une clé de cryptage valide est une chaine de caractères composée d'une suite d'opérations où chacune d'entre elles est représentée par exactement 4 caractères : deux caractères alphabétiques (lettres) suivis de deux caractères numériques (chiffres). De plus, les 2 lettres doivent correspondre à l'un des codes d'opérations présentés à la section précédente : `RG`, `RD`, `PE`, `PI`, ou `IV`. Une clé de cryptage valide comporte toujours au moins 4 caractères, et sa longueur est toujours un multiple de 4. **Notez qu'on ne tient pas compte de la casse** pour valider une clé de cryptage : la clé valide `"IV44RG23IV05PI10RD03PE15"` est équivalente à la clé `"Iv44rg23iV05PI10rd03pe15"`. La clé de cryptage définit donc une ou plusieurs opérations de transformation du message pour l'encoder, chaque opération étant définie sur 4 caractères.

Encodage d'un message avec la clé de cryptage

Pour encoder un message, il suffit d'appliquer les opérations indiquées dans la clé de cryptage, à partir du début de la clé. Par exemple, pour crypter un message avec la clé `"IV44RG23IV05PI10RD03PE15"`, il faut appliquer les opérations suivantes, une à la suite de l'autre, sur le message à encoder (dans cet ordre) :

1) Inversion de 44 caractères
2) 23 rotations gauches
3) Inversion de 5 caractères
4) 10 permutations intérieures
5) 3 rotations droites
6) 15 permutations extérieures

Exemples d'encodage avec la clé de cryptage `"IV06RG04RD07"` :

Message en clair	**Message encodé**
`Feu vert !`	`rFeev u! t`
`RV 14h00, place habituelle.`	`leuh41 VR00, place habit.el`

1.3 DÉCODAGE D'UN MESSAGE CRYPTÉ

Pour pouvoir décoder un message crypté, il faut posséder la clé qui a servi à encoder le message. Il suffit alors d'exécuter les opérations inverses, dans l'ordre inverse, pour récupérer le message en clair.

Par exemple, pour décoder un message qui a été encodé avec la clé de cryptage `"IV06RG04RD07"`, il faut exécuter les opérations suivantes, dans cet ordre :

1) 7 rotations gauches (inverse de l'opération `RD07`)
2) 4 rotations droites (inverse de l'opération `RG04`)
3) inversion de 6 caractères, mais en commençant par l'inversion en fin de chaine plutôt qu'en début de chaine (inverse de l'opération `IV06`)

1.4 LANCEMENT DU PROGRAMME

Le programme démarre en affichant un message de présentation, et en demandant d'appuyer sur ENTRÉE pour continuer. Lorsque l'utilisateur tape ENTRÉE, le programme affiche un menu de 3 options qui sont décrites dans les sections suivantes. Exemple :

```
Ce logiciel permet de crypter et de decrypter
des messages secrets.

Tapez <ENTREE> pour continuer...

----
MENU
----
1. Crypter un message
2. Decrypter un message
3. Quitter

Entrez votre choix :
```

Vous devez valider le choix au menu. ATTENTION, les SEULS 3 choix valides sont 1, 2 ou 3. Par exemple, si l'utilisateur ne fait que taper ENTRÉE, ou s'il saisit "11", ce n'est pas valide. Lorsqu'une valeur entrée n'est pas valide, le programme doit afficher un message d'erreur, réafficher le menu, et solliciter de nouveau le choix au menu. Notez aussi que la saisie du choix au menu ne doit jamais faire planter le programme.

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier **`lancementProgEtValidationMenu.txt`**).

1.5 MENU CRYPTER UN MESSAGE

Au choix de cette option, le programme vous permet d'encoder un message avec la clé de cryptage de votre choix.

Le programme demande d'abord de saisir la clé de cryptage qui servira à encoder le message. Vous devez valider la clé de cryptage (voir la section 1.2 qui explique les critères de validité d'une clé de cryptage). Ensuite, le programme demande d'entrer le message à encoder. Vous devez valider ce message. Un message valide est une chaine de caractères pouvant contenir des lettres non accentuées (majuscules ou minuscules), des chiffres, des espaces, et les 9 caractères suivants : `.!?,;:'-"` (point, point d'exclamation, point d'interrogation, virgule, point-virgule, deux-points, apostrophe, tiret, et guillemets). **Notez que le message ne peut pas contenir de sauts de ligne (\n)**. De plus, lors de la validation du message, si l'utilisateur tape seulement ENTRÉE, l'opération est annulée, et le programme affiche de nouveau le menu dans l'attente d'un autre choix de l'utilisateur.

Lorsque la clé et le message ont été validés, le programme affiche le message encodé, **entre crochets**, et demande de taper ENTRÉE pour continuer (et revenir au menu).

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier **`menu1.txt`**).

1.6 MENU DÉCRYPTER UN MESSAGE

Au choix de cette option, le programme vous permet de décoder un message avec la clé de cryptage qui a servi à l'encoder.

Le programme demande d'abord de saisir la clé de cryptage qui a servi à encoder le message. Vous devez valider la clé de cryptage (voir la section 1.2 qui explique les critères de validité d'une clé de cryptage). Ensuite, le programme demande d'entrer le message à décrypter. Vous devez valider ce message (voir la section 1.5 qui explique les critères de validité d'un message). Lors de la validation du message, si l'utilisateur tape seulement ENTRÉE, l'opération est annulée, et le programme affiche de nouveau le menu dans l'attente d'un autre choix de l'utilisateur.

Lorsque la clé et le message ont été validés, le programme affiche le message décodé, **entre crochets**, et demande de taper ENTRÉE pour continuer (et revenir au menu).

Spécifications complémentaires :
Voir les exemples d'exécution fournis avec l'énoncé du TP (fichier **`menu2.txt`**).

1.7 MENU QUITTER

Au choix de cette option, le programme se termine et affiche `AUREVOIR !`

Spécifications complémentaires :
Voir tous les fichiers d'exemples d'exécution fournis avec l'énoncé du TP.

1.8 MÉTHODE POUR CONVERTIR UNE CHAINE DE CARACTÈRES NUMÉRIQUES EN ENTIER

La méthode de classe `parseInt` de la classe `Integer` permet de convertir une chaine de caractères en nombre entier (`int`) lorsque cela est possible. Elle prend en paramètre la chaine à convertir (`String`), et retourne l'entier (`int`) correspondant. ATTENTION : si la chaine à transformer ne contient pas que des caractères numériques (et ne peut donc être transformée en `int`), l'appel de cette méthode fera planter votre programme (`NumberFormatException`). Il faut donc vous assurer que la chaine à transformer ne contient que des chiffres avant d'utiliser cette méthode.

Exemple d'utilisation :

```
int nombre = Integer.parseInt("123"); //après l'appel, nombre contient l'entier 123
```

2. Précisions

- Vous devez écrire votre programme dans une classe nommée `Cryptage` qui contiendra votre méthode `main` et toutes les autres méthodes que vous aurez conçues.
- Votre classe `Cryptage` doit se trouver dans le paquetage par défaut (il ne doit pas y avoir d'instruction de paquetage `package`… dans le haut de votre classe).
- Prenez soin de bien découper votre code à l'aide de méthodes. Voici quelques lignes directrices pour vous guider dans la conception de vos méthodes :
  - Chaque fois que vous vous apercevez que différentes parties de votre code se ressemblent énormément, essayer d'en faire une méthode bien paramétrée (si possible).
  - Faites une méthode pour chaque petite tâche spécifique à accomplir dans la résolution du problème.
  - Si une méthode semble longue, posez-vous la question à savoir si elle pourrait être découpée en plusieurs méthodes plus spécialisées.

- Votre programme doit OBLIGATOIREMENT contenir les 2 méthodes suivantes, `crypter` et `decrypter` (en plus de toutes vos autres méthodes). Ces méthodes prennent en paramètres une clé de cryptage ainsi qu'un message à crypter/décrypter, et retourne le message crypté/décrypté.

```
/**
 * Crypte le msg donne avec la cle de cryptage donnee, et retourne
 * le message crypte.
 *
 * ANTECEDENT : la cle et le msg doivent etre valides.
 *
 * @param cle la cle de cryptage
 * @param msg le message a cripter avec la cle donnee
 * @return le message crypte
 */
public static String crypter(String cle, String msg)


/**
 * Decrypte le msg donne avec la cle de cryptage donnee, et retourne
 * le message decrypte.
 *
 * ANTECEDENT : la cle et le msg doivent etre valides.
 *
 * @param cle la cle de cryptage
 * @param msg le message a decripter avec la cle donnee
 * @return le message decrypte
 */
public static String decrypter(String cle, String msg)
```

ATTENTION : respectez à la lettre les 2 entêtes de méthodes donnés ci-dessus (et respectez l'ordre des paramètres), car elles seront testées telles quelles.

Exemples d'appels :

```
crypter("IV06RG04RD07", "Feu vert !")     retourne      "rFeev u! t"
decrypter("IV06RG04RD07", "rFeev u! t")   retourne      "Feu vert !"
```

- Les méthodes `crypter` et `decrypter` devraient, elles aussi, être bien découpées en faisant appel à d'autres méthodes. Comme par exemple, chaque operation (`RG`, `RD`, `PE`, `PI`, `IV`) devrait faire l'objet d'une méthode, etc.
- **Toutes vos méthodes doivent être** `public static`.

- N'oubliez pas d'écrire les commentaires de documentation (Javadoc) pour chacune de vos méthodes (sauf la méthode `main`).
- Lorsqu'on vous demande de valider une valeur saisie par l'utilisateur, cela sous-entend une **BOUCLE** de validation qui sollicite la valeur, lit la valeur, et lorsque la valeur saisie est invalide, affiche un message d'erreur, sollicite et lit de nouveau la valeur, et ce tant que la valeur saisie n'est pas valide.
- **Vous DEVEZ utiliser la classe `Clavier.java` pour effectuer toutes les saisies.**
- **Les seules CLASSES PERMISES sont** `String`, `Math`, `Integer`, `Character`, `Clavier`, et `System`.
- Vous **NE DEVEZ PAS utiliser les expressions régulières**, ni la classe `StringTokenizer` (qu'on ne verra pas dans ce cours).
- **Vous NE DEVEZ PAS utiliser les tableaux (qu'on n'a pas vus encore). ATTENTION, l'utilisation de tableaux peut engendrer une pénalité pouvant aller jusqu'à 30% puisque vous passerez à côté de ce qu'on veut noter dans ce TP sur la manipulation des chaines de caractères.**
- Vous DEVEZ respecter à la lettre les **informations, les messages**, et le **format** de l'affichage qui sont montrés dans les exemples d'exécution fournis avec l'énoncé du TP. Vous devez considérer les exemples d'exécution comme des spécifications complémentaires à celles décrites dans cet énoncé.
- Votre programme NE DOIT JAMAIS PLANTER lors d'une saisie faite par l'utilisateur.
- Toute **VARIABLE GLOBALE** est **INTERDITE : toutes les variables doivent être déclarées à l'intérieur (et au début) d'une méthode** (sauf pour les boucles `for` où vous pouvez déclarer la variable de contrôle dans l'entête de la boucle)**.**
  - Pour passer une information du programme appelant (celui qui appelle la méthode) au programme appelé (la méthode appelée), vous devez passer cette information en paramètre.
  - Pour passer une information du programme appelé (une méthode quelconque) au programme appelant (celui qui appelle la méthode), utilisez la valeur de retour de la méthode appelée.
- Utilisez des constantes (`final`) autant que possible : les messages affichés, les bornes de validation, etc.
- Les constantes doivent être déclarées et initialisées au niveau de la classe (constantes globales) : `public static final` (ou `public final static`)…
- L'affichage des résultats doit se faire à la console.
- Utilisez des variables réelles uniquement lorsque requis. Ex. : un compteur ne doit pas être `float` ou `double`.
- Vous DEVEZ respecter le style Java et les bonnes pratiques de programmation.
- **UTILISATIONS INTERDITES :**
  - L'instruction `break` (sauf comme dernière instruction d'un `case` dans un `switch`).
  - L'instruction `continue`.
  - L'instruction `return` dans une méthode dont l'entête mentionne qu'elle ne retourne rien (`void`).
  - La modification de la variable de contrôle à l'intérieur d'une boucle `for`.
  - L'instruction `System.exit(…)`.

  ATTENTION : Une pénalité de **5 points sur 100** (jusqu'à concurrence de 30 points) sera attribuée pour chaque utilisation interdite. Par exemple, si vous utilisez 3 fois un `break` interdit, vous perdrez 15 points.
- Vous DEVEZ utiliser les notions vues au cours pour faire votre TP.
- Votre code doit pouvoir s'exécuter avec le JDK7. Si vous utilisez les notions vues au cours, il n'y aura pas de problème.
- Votre fichier de code source doit être encodé en UTF-8.
- **Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé du TP (et les exemples d'exécution donnés avec l'énoncé du TP) est susceptible d'engendrer une perte de points.**

Note : *Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.*

3. Détails sur la correction

3.1 LA QUALITÉ DU CODE (40 POINTS)

Concernant les critères de correction du code, **lisez attentivement** le document "CriteresGeneraux DeCorrection.pdf". De plus, votre code sera noté sur le respect des conventions de style Java vues en classe (dont un résumé se trouve dans le document "ConventionsStyleJavaPourINF1120.pdf". Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours.

Note : Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur la qualité de votre code, et le respect des spécifications/consignes mentionnées dans ce document.

3.2 L'EXÉCUTION (60 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Note : Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible. Notez que les exemples d'exécution fournis ne couvrent pas nécessairement tous les cas à tester.

4. Date et modalités de remise

4.1 REMISE

Date de remise : Au plus tard le 26 mars 2023 à 23h59.
Le fichier à remettre : `Cryptage.java` (PAS dans une archive zip, rar, etc.)

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boite de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE) - REMISE DU TP2**.

Vérifiez deux fois plutôt qu'une que vous avez remis le bon fichier, car c'est celui-là qui sera considéré pour la correction.

Assurez-vous de remettre votre travail avant la date limite, sur Moodle, car sinon vous n'aurez plus accès à la boite de remise, et vous ne pourrez plus remettre votre travail.

4.2 POLITIQUE CONCERNANT LES RETARDS

Une pénalité de 10% de la note finale, par jour de retard, sera appliquée aux travaux remis après la date limite. La formule suivante sera utilisée pour calculer la pénalité pour les retards : `Nbr points de pénalité = m / 144,` où `m` est le nombre de minutes de retard par rapport à l'heure de remise. Ceci donne 10 points de pénalité pour 24 heures de retard, 1.25 point de pénalité pour 3 heures, etc.

Aucun travail ne sera accepté après 1 jour (24 heures) de retard et la note attribuée sera 0.

4.3 REMARQUES GÉNÉRALES

- **Aucun programme reçu par courriel ne sera accepté.** Plus précisément, un travail reçu par courriel sera considéré comme un travail non remis.
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire (entre autres) votre nom complet et votre code permanent dans l'entête de la classe à remettre.**

N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus!

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !