Travail pratique 2 – Agence spatiale boulonnaise (ASB)

Établissement d'éducation reconnu pour son innovation, le Collège de Bois-de-Boulogne élabore un nouveau programme des sciences de la nature en immersion spatiale. En effet, les prochaines cohortes résideront et étudieront dans l'espace. Oui, oui, vous avez bien lu… Le cégep transportera les étudiants à l'aide d'une navette fabriquée dans le cadre d'une activité parascolaire à laquelle vous participez. Votre rôle dans cette aventure, c'est d'écrire le code d'autorisation du lancement, c'est-à-dire le programme qui s'assure que tous les systèmes sont fonctionnels et sécuritaires juste avant la mise à feu. C'est exact : la décision ultime de procéder au décollage repose sur votre logiciel. Quoi? Vous n'étiez pas informé? Et bien, et bien… Dites-vous qu'il n'est pas encore trop tard pour commencer à programmer!

DIRECTIVES ET INFORMATIONS RELATIVES AU TRAVAIL

- Ce travail compte pour 20% de la note finale.
- Ce travail se fait individuellement.
- La remise du travail pratique complet (format zip) se fait sur Léa avant la date butoir (1er mai 23h59).
- Tout plagiat ou partage de code sera sanctionné par l'attribution de la note zéro.

SPÉCIFICATIONS DU SYSTÈME À RÉALISER

Pour la version 1.0 du logiciel, vous avez la responsabilité de vérifier cinq systèmes de la navette avant le décollage :


Système de poussée


Système de pilotage


Système de communication


Système d'oxygénation


Système de recyclage des déchets organiques

Chacun de ces systèmes produira une chaîne de contrôle qui consiste en 1024 caractères hexadécimaux. Voici à quoi ressemble une de ces chaînes : `D89B01C5FFC`*…plusieurs autres caractères…*`0EB12`.

Ces codes doivent être vérifiés à l'aide de deux types de composants :


Convertisseurs de données


Analyseurs de données

LES CONVERTISSEURS

Votre logiciel doit implémenter cinq types de convertisseurs :

- Convertisseur INV

  Le convertisseur INV inverse tous les caractères dans la chaîne selon les règles suivantes :
  0→F, 1→E, 2→D, 3→C, 4→B, 5→A, 6→9, 7→8, 8→7, 9→6, A→5, B→4, C→3, D→2, E→1, F→0.

  Par exemple, le convertisseur transforme « 73C900AB » en « 8C36FF54 ».

- Convertisseur ROT

  Le convertisseur ROT effectue la rotation de tous les caractères en fonction de sa configuration (*delta*). Un convertisseur est créé avec un *delta* spécifique qui ne peut pas être changé.

  La règle est la suivante : « nouveau caractère = ancien caractère + delta ». Ainsi, avec un *delta* de 3, le caractère « A » devient « D » (« A » + 3). Le dépassement est également géré. Par exemple : « E » devient « 4 » si le *delta* est 6.

  Les valeurs négatives de *delta* sont permises. Par exemple, avec un *delta* de -2, « 7 » devient « 5 ».

  Il n'y a pas de valeur minimale ou maximale pour le *delta* autre que les limites du type (selon les normes de l'Association mondiale des constructeurs d'aéronefs artisanaux, seuls les *deltas* de type `int` sont autorisés). Par exemple, un delta de 20 fait en sorte que « A » devient « E ».

- Convertisseur HPF

  Le convertisseur HPF est un filtre passe-haut inclusif. Il utilise une valeur *seuil* et ne garde que les valeurs égales ou supérieures. Un *seuil* par défaut de « 8 » est spécifié au moment de la fabrication du convertisseur.

  Le *seuil* doit pouvoir être ajusté au besoin à l'aide d'un caractère compris entre [« 0 », « 9 »] ou [« A », « F »].

  Lorsqu'on applique un convertisseur HPF avec un *seuil* de « 3 » à la chaîne « 921A7230 », on obtient la chaîne « 900A7030 ». Tout ce qui est plus petit que le *seuil* est atténué à « 0 ».

- Convertisseur LPF

  Le convertisseur LPF est un filtre passe-bas inclusif. Il utilise une valeur *seuil* et ne garde que les valeurs égales ou inférieures. Un *seuil* par défaut de « 7 » est spécifié au moment de la fabrication du convertisseur.

  Le *seuil* doit pouvoir être ajusté au besoin à l'aide d'un caractère compris entre [« 0 », « 9 »] ou [« A », « F »].

  Lorsqu'on applique un convertisseur LPF avec un *seuil* de « 3 » à la chaîne « 921A7230 », on obtient la chaîne « 0210230 ». Tout ce qui est plus grand que le *seuil* est atténué à 0.

- Convertisseur POI

  Le convertisseur POI (pair ou impair) produit deux transformations : il change tous caractères impairs (« 1 », « 3 », « 5 », « 7 », « 9 », « B », « D », « F ») en « 0 » ou il change tous les caractères pairs (« 0 », « 2 », « 4 », « 6 », « 8 », « A », « C », « E ») en « 0 ». Pour déterminer le comportement, on actionne un *interrupteur*. Si l'*interrupteur* est en position ouverte, on obtiendra des caractères pairs seulement. Dans le cas contraire, on obtiendra des caractères impairs seulement.

  Il est possible de changer la position de l'*interrupteur* entre les conversions de chaînes de contrôle (entre deux appels de la méthode `convertir()`).

Tous les convertisseurs doivent avoir une méthode `convertir()` qui accepte une chaîne (`String`) à convertir en paramètre et qui retourne la chaîne convertie (`String`).

Si la chaîne de contrôle n'a pas 1024 caractères, la méthode `convertir()` doit retourner `null`.

LES ANALYSEURS

Vous devez également créer trois analyseurs de données :

- Analyseur 1D

  L'analyseur 1D, comme son nom l'indique, traite les données en une dimension. De fait, la chaîne de 1024 sera analysée sous la forme d'un tableau à une dimension.

  L'analyseur 1D recherche des séquences, de gauche à droite et de droite à gauche, dans la chaîne à analyser de différentes méthodes (façons) :

- **Méthode 1** – On peut lui fournir *une séquence* à rechercher. Il retourne `true` si la séquence est présente dans *la chaîne à analyser*, `false` sinon.

- **Méthode 2** – On peut lui fournir *une liste de séquences* à rechercher. Il retourne `true` si au moins une des séquences est présente, `false` si aucune des séquences ne se trouve dans *la chaîne à analyser*.

- **Méthode 3** – On peut lui fournir *une séquence* à rechercher et *un seuil*. Le seuil est le nombre minimal de fois que la séquence doit être trouvée. Par exemple, si on recherche la séquence « ABC » dans la chaîne « 5552ABABC1123CBD9 » avec un seuil de 2, l'analyseur doit retourner `false` parce que *la chaîne à analyser* ne contient qu'une seule occurrence de la chaîne « ABC ». Il retournerait toutefois `true` si la séquence était « AB », « BC » ou « 55 ».

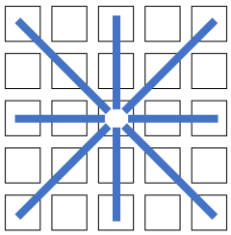
- **Méthode 4** – On peut lui fournir *une liste de séquences* et *un seuil*. Dans un tel cas, l'analyseur vérifie si le nombre d'occurrences de toutes les séquences est égal ou supérieur au seuil demandé. Par exemple, soit 2 séquences (« ABC » et « DEF »), le seuil 2 et *la chaîne à analyser* « ABCDEF7ABCDEFF3ABC8 ». Dans un tel cas, l'analyseur doit retourner `true` puisqu'il y a trois occurrences de la séquence « ABC » et deux occurrences de la séquence « DEF ». Le seuil est respecté pour chaque séquence.

- Analyseur 2D

L'analyseur 2D, comme son nom l'indique, traite les données en deux dimensions. De fait, la chaîne de 1024 sera analysée sous la forme d'un tableau à deux dimensions où les 32 premiers caractères seront la première ligne, les 32 suivants la deuxième ligne, les 32 suivants la troisième et ainsi de suite jusqu'à la 32e ligne. Le tableau comptera 32 lignes et 32 colonnes (32 x 32 = 1024).

À noter : le tableau est une représentation conceptuelle. Le code ne doit pas obligatoirement créer de tableau pour procéder à l'analyse. Vous êtes libre de créer l'algorithme et les structures de données qui vous conviennent.

L'analyseur 2D recherche des séquences dans la chaîne à analyser des mêmes quatre façons que l'analyseur 1D, mais dans plusieurs directions qu'on peut imaginer en forme d'étoile plane :



- Analyseur 3D

  L'analyseur 3D, comme son nom l'indique, traite les données en trois dimensions.

  De fait, la chaîne de 1024 est divisée en 2 blocs de 512 caractères (bloc 1 = 512 premiers caractères, bloc 2 = 512 derniers caractères) chacun. Chaque bloc est ensuite analysé sous la forme d'un tableau à trois dimensions où les 8 premiers caractères seront la première ligne (de [0][0][0] à [7][0][0]), les 8 suivants la deuxième ligne (de [0][1][0] à [7][1][0]), les 8 suivants la troisième (de [0][2][0] à [7][2][0]) et ainsi de suite jusqu'à la 8e ligne de la première couche/profondeur (de [0][7][0] à [7][7][0]). Le 9e groupe de 8 caractères sera la première ligne de la deuxième couche/profondeur (de [0][0][1] à [7][0][1]), le 10e groupe de 8 caractères sera la deuxième ligne de la deuxième couche/profondeur (de [0][1][1] à [7][1][1]), etc. Chacun des deux tableaux comptera 8 lignes, 8 colonnes et 8 couches (8 x 8 x 8 = 512).

  > *À noter : le tableau est une représentation conceptuelle. Le code ne doit pas obligatoirement créer de tableau pour procéder à l'analyse. Vous êtes libre de créer l'algorithme et les structures de données qui vous conviennent.*

  L'analyseur 3D recherche des séquences dans la chaîne à analyser des mêmes quatre façons que les analyseurs 1D et 3D, mais dans toutes les directions (ou un déplacement entier est possible) qu'on pourrait imaginer en forme d'étoile en trois dimensions.

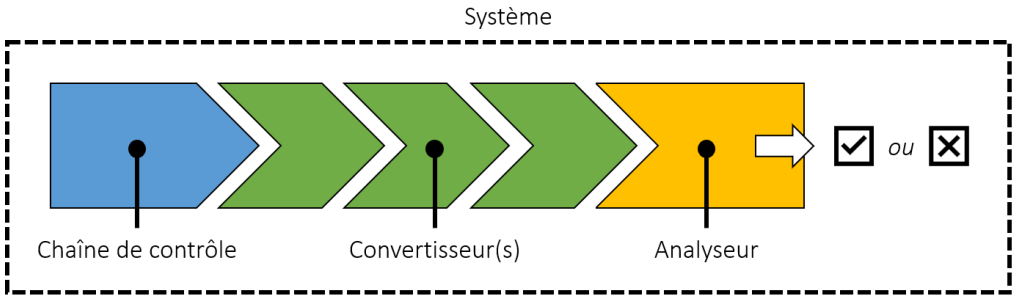
Tous les analyseurs doivent avoir quatre méthodes `analyser()` qui accepte les paramètres d'analyse et qui retourne un booléen.

Note : dans tous les cas, la chaîne à analyser doit compter 1024 caractères hexadécimaux. Si ce n'est pas le cas, l'analyseur doit afficher une erreur et retourner `false`.

Une même séquence ne peut être comptée deux fois. Par exemple, avec l'analyseur 1D : « 77 » dans « 7777 » ne sera comptée que 3 fois. Dans la direction de gauche à droite : « 7777 » (1 fois), « 7777 » (2 fois) et « 7777 » (3 fois). Ensuite, de droite à gauche, on trouve « 7777 » (déjà comptée), « 7777 » (déjà comptée) et « 7777 » (déjà comptée). Le même comportement est attendu des analyseurs 2D et 3D.

LES SYSTÈMES

Sur chaque système, on installe un analyseur et un ou plusieurs convertisseurs :



Le système produit la chaîne de contrôle de 1024 caractères. Cette chaîne devient l'entrée du premier convertisseur qui la traite (méthode `convertir()`). La chaîne traitée devient ensuite l'entrée d'un autre convertisseur (et ainsi de suite jusqu'à ce que tous les convertisseurs soient appliqués) ou d'un analyseur. L'analyseur effectue son traitement (la méthode `analyser()` utilisée dans ce système) et produit un résultat : le système est vérifié ou non.

Si tous les systèmes sont vérifiés, le décollage est autorisé. C'est le résultat final de l'exécution de votre logiciel.

Si vous avez bien effectué votre travail, bravo : les étudiants se rendront probablement sains et saufs à destination. Si vous avez mal effectué votre travail, vous vivrez avec des morts sur la conscience.

Toutefois, si un seul des systèmes n'est pas vérifié, la procédure de lancement sera interrompue parce qu'un des systèmes est défectueux (ou votre code fait de mauvaises vérifications et coûtera des millions de dollars au cégep).

Pour produire une chaîne de contrôle, chacun de vos systèmes devra faire appel au **générateur de chaînes de contrôle** en appelant une méthode statique comme suit :

```
String chaineControle = GrandGenerateur.obtenirChaineControle(germe);
```

Ce sont les enseignants du collège qui travaillent sur le générateur et qui vous fourniront cette classe (voir dernière page du TP) ainsi que la valeur du germe à utiliser pour chacun des cinq systèmes. Ils vous informeront également des convertisseurs et analyseurs à installer sur les systèmes au moment d'effectuer les tests de mise en production de la navette.

BARÈME DE CORRECTION

LE PROGRAMME PRINCIPAL	**5**
Classe **`Navette`** avec une structure de données contenant 5 objets **`Systeme`**s	*2*
Identification claire des 5 **`Systeme`**s	*1*
Autorisation ou non du lancement de la navette (fonctionnement du code)	*2*
LES SYSTÈMES	**10**
Obtention des chaînes de contrôle	*2*
Chaînage des **`Composant`**s (**`Convertisseur`**(s) et **`Analyseur`**)	*2*
Validation du chaînage (un ou plusieurs convertisseurs suivi(s) d'un analyseur)	*3*
Vérification du **`Systeme`** (fonctionnement du code)	*3*
LES CONVERTISSEURS	**30**
`ConvertisseurINV`	*5*
`ConvertisseurROT`	*5*
`ConvertisseurHPF`	*6*
`ConvertisseurLPF`	*6*
`ConvertisseurPOI`	*8*
LES ANALYSEURS	**35**
`Analyseur1D` – méthode 1	*7*
`Analyseur1D` – méthode 2	*3*
`Analyseur1D` – méthode 3	*7*
`Analyseur1D` – méthode 4	*3*
`Analyseur2D` – méthode 1	*4*
`Analyseur2D` – méthode 2	*3*
`Analyseur2D` – méthode 3	*5*
`Analyseur2D` – méthode 4	*3*
AUTRES	**20**
Utilisation adéquate de l'encapsulation	*5*
Utilisation adéquate de l'héritage	*5*
Utilisation adéquate de l'abstraction	*5*
Utilisation adéquate du polymorphisme	*5*
BONUS	**5**
`Analyseur3D` – méthode 1	*1*
`Analyseur3D` – méthode 2	*1*
`Analyseur3D` – méthode 3	*2*
`Analyseur3D` – méthode 4	*1*

** Le respect du guide de style et la lisibilité du code (commentaires, indentation, javadoc, etc.) seront pris en considération lors de l'évaluation des éléments ci-dessous.*

GÉNÉRATEUR DE CHAÎNES DE CONTRÔLE

À utiliser tel quel (aucune modification permise) :

```
import java.util.Random;

/**
 * GrandGenerateur
 * Génère aléatoirement une chaîne de contrôle de 1024 caractères.
 *
 * @author  Gilles-Philippe Grégoire et Eric Drouin
 * @version 0.1
 * @since   2022-03-28
 */
public class GrandGenerateur {

    public static final int LONGUEUR_CHAINE_CONTROLE = 1024;

    /**
     * Génère une chaîne de contrôle en utilisant des structures de données et
     * des objets de niveau 2CP.
     *
     * @param germe pour la génération aléatoire
     * @return la chaîne de contrôle de LONGUEUR_CHAINE_CONTROLE caractères
     */
    public static String obtenirChaineControle(int germe) {

        final char[] CARACTERES = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B',
'C', 'D', 'E', 'F'};

        char[] chaine = new char[LONGUEUR_CHAINE_CONTROLE];

        Random aleatoire = new Random(germe);

        for (int i = 0; i < chaine.length; i++) {
            chaine[i] = CARACTERES[aleatoire.nextInt(16)];
        }

        return new String(chaine);
    }
}
```