R1.01 – TP10
Exceptions / Listes chaînées

L'objectif de ce TP est :
- *de découvrir comment réellement contrôler la saisie d'une variable de type numérique, en traitant l'exception levée par la JVM en cas de saisie invalide*
- *d'apprendre à déclarer, initialiser et utiliser des listes chaînées.*

Avant de commencer...

- Ouvrez un terminal et placez-vous dans votre répertoire `R1.01`
  - ✓ Exécutez la commande : `cp -r /users/info/pub/1a/R1.01/TP10_Files .`
  - ✓ Lancez **IJ** et créez un projet `TP10`
- Ouvrez le dossier `TP10_Files`
- Sélectionnez chaque fichier de ce dossier et déplacez-le par *glisser/déposer* dans le répertoire `src` de votre projet `TP10`
- Vérifiez que le répertoire `src` du projet `TP10` contient les classes : `Cellule`, `ExceptionMauvaisIndice`, `ListeChainee`, `ListeInterface`

Partie A : Exceptions - Contrôle de saisie

Lors de la lecture d'une variable de type numérique, l'exception `InputMismatchException`[1] de la classe `java.util` est levée, si l'utilisateur entre une valeur qui n'appartient pas à l'ensemble des valeurs du type attendu.
*Exemple : le type attendu est `int` et l'utilisateur tape **abc***

Jusqu'à présent, nous avons contourné ce problème en partant du principe que l'utilisateur suit les consignes.
Exemple : l'utilisateur saisit vraiment une valeur du type `int` quand on lui demande de saisir un entier...

Cette solution n'est bien sûr pas recevable dans un contexte professionnel...

A1. Contrôle de saisie d'une variable de type numérique par gestion de l'exception `InputMismatchException`

`typeNum` étant un type simple numérique (`byte` | `short` | `int` | `long` | `float` | `double`), une bonne pratique pour contrôler la saisie d'une variable de ce type, consiste à écrire une fonction selon le modèle suivant :

```
public static typeNum getTypeNum(Scanner lecteur) {
  typeNum val;
  try {
      ... // message invitant l'utilisateur à entrer une valeur de type typeNum
      ... // instruction de saisie de val
      return val; // retour du résultat (exécuté si et seulement l'utilisateur entre une valeur de type typeNum)
  } catch (InputMismatchException e) { // traitement de l'exception e en cas de mauvaise saisie
      lecteur.nextLine(); // pour vider le buffer suite à la mauvaise saisie
      ... // message invitant l'utilisateur à une nouvelle saisie
      return getTypeNum(lecteur) // rappel de la fonction (pour nouvelle saisie)
  }
}
```

1.1. Dans le projet `TP10`, créez une classe `Utilitaire` où vous coderez les fonctions suivantes :

a) Saisie d'une variable de type `int` :

```
public static int getInt(Scanner lecteur) {
    // {} => {résultat = un entier saisi par l'utilisateur, SAISIE CONTRÔLÉE }
```

b) Saisie d'une variable de type `float` :

```
public static int getFloat(Scanner lecteur) {
    // {} => {résultat = un réel saisi par l'utilisateur, SAISIE CONTRÔLÉE }
```

[1] https://docs.oracle.com/javase/7/docs/api/java/util/InputMismatchException.html

1.2. Créez une classe `ControleSaisie` dans laquelle vous ajoutez une procédure `main`, où vous écrirez les déclarations et les instructions nécessaires au test de ces fonctions

1.3. **Testez**…

A2. Contrôle de saisie d'une variable de type numérique par gestion de l'exception `NumberFormatException`

Les classes enveloppes `Byte`, `Short`, `Integer`, `Long`, `Float` et `Double` disposent respectivement des méthodes `parseByte(String s)`, `parseShort(String s)`, `parseInt(String s)`, `parseLong(String s)`, `parseFloat(string s)` et `parseDouble(String s)`, qui renvoient (*quand c'est possible*) la valeur dans le type primitif encapsulé, représentée par la chaîne `s` passée en paramètre.

Exemple :

La méthode `parseInt(String s)` de la classe `Integer` :
- renvoie la valeur de type `int` représentée par `s`, dans le cas où `s` représente bien une valeur de type `int`
- lève l'exception `NumberFormatException`[2] si `s` ne représente pas une valeur de type `int`

Une autre bonne pratique pour saisir une variable du type numérique typeNum est d'écrire une fonction selon le modèle suivant :

```
public static typeNum getTypeNumV2(Scanner lecteur) {
  try {
      ... // message invitant l'utilisateur à entrer une valeur de type typeNum
      ... // retour de la conversion dans le type typeNum d'une chaîne résultant de l'instruction lecteur.getLine();
          // non exécuté si cette chaîne n'est pas convertible dans le type typeNum
  } catch (NumberFormatException nfe) { // traitement de l'exception nfe en cas de mauvaise saisie
      ... // message invitant l'utilisateur à une nouvelle saisie
      ... // rappel de la fonction (pour nouvelle saisie)
  }
}
```

2.1. Dans la classe `ControleSaisie` ajoutez, puis codez les fonctions suivantes :

a) Saisie d'une variable de type `int` par conversion chaîne -> entier

```
public static int getIntV2(Scanner lecteur) {
    // {} => {résultat = un entier saisi par l'utilisateur, SAISIE CONTRÔLÉE }
```

b) Saisie d'une variable de type `float` par conversion chaîne -> réel

```
public static int getFloatV2(Scanner lecteur) {
    // {} => {résultat = un réel saisi par l'utilisateur, SAISIE CONTRÔLÉE }
```

2.2. Dans la procédure `main` de la classe `ControleSaisie`, ajoutez les instructions nécessaires au test de ces fonctions.

2.3. **Testez**…

ATTENTION :

Contrairement à la méthode `nextFloat()` de la classe `Scanner`, pour laquelle le séparateur décimal doit être une **virgule**, la méthode `parseFloat()` de la classe `Float` attend un **point**, comme séparateur décimal…

[2] https://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html

Partie B : Listes chaînées

Nous vous avons présenté en cours :
- une classe interface `ListeInterface` décrivant le TAD liste
- une classe `Cellule` pour représenter un élément d'une liste chaînée
- une classe `ListeChainee` pour implanter le TAD liste au moyen d'une liste chaînée de cellules

Ces trois classes sont présentes dans votre projet TP10

COMMENCEZ PAR EN (RE)LIRE LE CONTENU AVANT DE RÉPONDRE AUX QUESTIONS DE CETTE PARTIE.

B1. Classe ListeInt_Main : manipulation de listes chaînées de Integer

Dans le projet `TP10`, créez une classe `ListeInt_Main` et ajoutez-y une procédure `main`

1.1. Dans la procédure main, ajoutez les instructions suivantes :
- ✓ déclaration d'une liste chaînée de nom `listeInt` pour le type `Integer`
- ✓ boucle d'insertion de 15 cellules dans cette liste
  NOTE : vous pouvez générer la valeur de l'attribut `info` en utilisant `Math.random()` – cf. TP9
- ✓ affichage du nombre de cellules de `listeInt`
- ✓ affichage de gauche à droite, puis de droite à gauche de `listeInt`

1.2. Testez…

B2. Premiers algorithmes sur listes chaînées de Integer

2.1. Somme des infos portées par les éléments de `listeInt`

- Dans la classe `Utilitaire`, ajoutez les fonctions suivantes :

a) Somme des valeurs d'une liste d'entiers, non vide – FORME ITÉRATIVE

```
public static int sommeListeInt(ListeChainee<Integer> liste) {
    //{liste non vide} => {résultat = somme des éléments de liste
    //                     ALGORITHME ITÉRATIF}
```

b) Somme des valeurs d'une liste d'entiers, non vide – FORME RÉCURSIVE

b1 – le "modèle"

```
public static int sommeListeIntRec(ListeChainee<Integer> liste) {
    //{liste non vide} => {résultat = somme des éléments de liste }
```

b2 – le "worker"

```
private static int sommeListeIntRecWorker(Cellule<Integer> cellCour) {
    //{ } => {résultat = somme des éléments de la liste de tête cellCour
    //        ALGORITHME RÉCURSIF}
```

- Testez ces fonctions dans la procédure main de la classe `ListeIntMain`

2.2. Recherche dans `listeInt` d'une cellule portant une information saisie par l'utilisateur

- Dans la classe `Utilitaire`, ajoutez les fonctions suivantes :

a) Recherche d'une valeur dans une liste d'entiers non triée – FORME RÉCURSIVE

b1 – le "modèle"

```
public static boolean rechValListe(ListeChainee<Integer> liste, int val) {
    //{} => {résultat = vrai si au moins un élément de liste porte l'info val}
```

b2 – le "worker"

```
private static boolean rechValListeWorker(Cellule<Integer> cellCour, int val) {
    //{} => {résultat = vrai si au moins une valeur de la liste de tête cellCour
    //                  porte l'info val
    //       ALGORITHME RÉCURSIF}
```

b) Recherche de la position de la 1ère cellule portant une info donnée dans une liste d'entiers non triée et non vide – FORME ITÉRATIVE

```
public static int premPosVal(ListeChainee<Integer> liste, int val) {
    //{liste non vide} => {résultat = position de la première cellule de liste
    //                               portant l'info val, 0 si non trouvée
    //                     ALGORITHME ITÉRATIF}
```

- Dans la classe `ListeInt_Main` :
  - ✓ Déclarez et faites saisir par l'utilisateur un entier `unEnt` (PENSEZ À CONTRÔLER LA SAISIE - cf. Partie **A**)
  - ✓ Testez par appel de `rechValListe` s'il existe dans `listeInt` une cellule portant l'info `unEnt`
  - ✓ Déclarez un entier `posVal` initialisé par appel de la fonction `premPosVal`
  - ✓ Affichez (si possible) par appel de la méthode `getInfoAtPosit` de la classe `ListeChainee` la valeur portée par la cellule de position `posVal` dans `listeInt`

    ATTENTION :
    > La méthode `getInfoAtPosit` peut lever l'exception une exception !
    > Faites le nécessaire pour empêcher que le programme s'interrompe brutalement…

B3. Classe Utilitaire : algorithmes sur listes chaînées triées de Integer

3.1. Dans la classe `Utilitaire`, ajoutez et codez la procédure et la fonction suivantes :

a) insertion dans une liste d'entiers, triée par ordre croissant au sens large – FORME ITÉRATIVE

```
public static void insereDansListeTriee(ListeChainee<Integer> liste, int val) {
    //{liste triée} => {une cellule d'info = val a été insérée dans liste,
    //                  liste reste triée après insertion - FORME ITÉRATIVE}
```

b) vérification du tri – FORME ITÉRATIVE

```
public static boolean verifTri(ListeChainee<Integer> liste) {
    //{} => {résultat = vrai si liste est triée
    //       ALGORITHME ITÉRATIF}
```

3.2. Dans la procédure `main` de `ListeInt_Main`, ajoutez les instructions suivantes :

- ✓ déclaration d'une nouvelle liste chaînée de `Integer` sous le nom `listeTriee`
- ✓ insertion des éléments de listeInt dans `listeTriee` par appel de `insereDansListeTriee`
- ✓ affichage d'un message indiquant si `listeTriee` est effectivement triée par ordre croissant au sens large
- ✓ affichage *pour vérification* du nombre d'éléments de `listeTriee` et des éléments de `listeTriee`

3.3. **Testez**…