TD9

Exceptions

À retenir

- Le bloc **try .. catch** permet de *rattraper une exception*:

```
try
{
     int x = 0;
     int y = 3/x;
     System.out.println(y);
}
catch(ArithmeticException e)
{
     System.out.println("Division par 0");
}
```

- tentative de division par 0 (indéfini) ; dans la bloc *catch* l'exception récupérée est une exception arithmétique (prédéfinie en java). On affiche juste un message d'alerte, le programme continue ensuite en séquence.
- Définir une exception personnalisée :

```
class ListeVideException extends Exception
{
// C'est tout (car Exception possède un constructeur sans paramètre)
}
```

- Le mot clé **throw** permet de *Lever une exception*, attention erreur potentielle :

```
ListeVideException ex = new ListeVideException()
throw ex;
```

- Le mot clé **throws** indique qu'une méthode peut potentiellement lever une exception :

```
public class BibMachin
{

  public static int minimum(List<Integer> liste) throws ListeVideException
  // minimum peut lever une exception (quand la lite est vide)
  {
      if(liste.size() == 0)
          throw new ListeVideException(); // ici c'est le cas
      else
          return Collections.min(liste);
  }
}
```

Il nous est souvent arrivé de devoir écrire des méthodes pour lesquelles sur certaines entrées, aucun résultat valide n'est possible. Par exemple la méthode max qui prend en entrée une liste d'entiers n'a pas de résultat valide sur une liste vide.

1. Pour les méthodes suivantes, proposez des entrées pour lesquelles il ne peut y avoir de résultat valide :
   - *min* calculant le minimum d'une liste,
   - *int parseInt(String)* prenant en entrée une chaine et renvoyant un entier (vous pouvez regarder la documentation si nécessaire),
   - *Integer getValeur(List<Integer> l, int indice)* renvoyant la valeur de la liste à l'indice donné,
   - *int getAge(List<Personne> lp, String nom)*,
   - *List<Integer> creerTableau(int taille)*.

Il n'est *pas toujours possible* de décider d'une valeur par défaut à renvoyer sur une telle entrée. Bien souvent, c'est la méthode qui **fait l'appel** qui va pouvoir décider de ce qu'il convient de faire.

2. Supposons qu'on veuille calculer la somme des maximums d'une liste de liste d'entiers, par exemple `sommeMax([ [1,3,2], [2,10,1], [1,100,3,2,4]]` renverrait `113`. Si on a une fonction max, quel serait le code de sommeMax ? Que se passe-t-il si une des listes est vide ? Que faut-il que max renvoie dans ce cas ?
3. Supposons maintenant qu'on veuille calculer le produit des maximums d'une liste de liste d'entiers, par exemple `produitMax([ [1,3,2], [2,10,1], [1,100,3,2,4]]` renverrait `3000`. Si on a une fonction max, quel serait le code de produitMax ? Que se passe-t-il si une des listes est vide ? Que faut-il que max renvoie dans ce cas ?
4. Pouvez vous imaginer une valeur par défaut que renverrait max pour qu'on puisse écrire produitMax et sommeMax utilisant notre fonction max ?

En java (comme en python et la plupart des langages), on va utiliser des **Exceptions** . Ces dernières permettent à la personne qui écrit une méthode et qui ne peut pas savoir comment cette dernière sera utilisée de faire remonter l'information à la méthode **appelante**.

Par exemple, la méthode min de Collections est spécifiée de la manière suivante :

Documentation de Collections.min

public static T min(Collection<T> coll, Comparator<? super T> comp)

Returns the minimum element of the given collection …

Parameters:

coll - the collection whose minimum element is to be determined. comp - the comparator with which to determine the minimum element. A null value indicates that the elements' natural ordering should be used.

Returns:

the minimum element of the given collection, according to the specified comparator.

Throws:

NoSuchElementException - if the collection is empty.

5. Quelle est l'exception levée dans le cas où la liste passée en argument est vide ?
6. Selon vous, que fera le code suivant :

```
public class Executable{

      public static void afficheMin(List<Integer> liste)
      {
              Integer minimum = Collections.min(liste);
              System.out.println("Ma liste: " + liste);
              System.out.println("Et voici le minimum:");
              System.out.println(minimum);

      }
      public static void main(String[] args)
      {
              List<Integer> liste = new ArrayList<>();
              System.out.println("Liste 1");
              afficheMin(liste);
              System.out.println("Liste 2");
              List<Integer> liste2 = new ArrayList<>();
              liste2.add(1);
              liste2.add(7);
              liste2.add(-2);
              afficheMin(liste2);
      }
}
```

7. En utilisant le bloc try..catch dans la méthode afficheMin , modifiez le code précédent pour que l'affichage soit

```
Liste1
Ma Liste : []
La liste est vide
Liste2
Ma Liste: [1,7,-2]
Et voici le minimum:
-2
```

8. Même question, mais cette fois-ci en mettant le bloc try..catch dans la méthode main.

Lever une exception

Une exception est en fait un objet comme un autre. Pour "lever une exception", on utilisera le mot-clé **throw**.

1. Écrivez votre méthode min qui lève une exception NoSuchElementException si la liste est vide et renvoie la valeur du minimum sinon.
2. Il faut aussi indiquer que cette méthode min peut renvoyer une telle exception. Il faut le faire au niveau de la définition de la méthode. Ajoutez la ligne nécessaire (référez vous à la section *A retenir*).

produitMax et sommeMax

À l'aide des concepts précédents écrivez une méthode max, et les méthodes produitMax et

sommeMax.