NFA019 : Série 3.1- Morane Gruenpeter

a. La série 3.1 consiste à créer une classe ES qui gère les entrées et les sorties en utilisant les exceptions. La zone de capture qui peut soulever une exception-- try{} catch—et le soulèvement des exceptions, sont les concepts étudiés.

b. la seule classe que l'on avait créé pour cette série est la classe ES dans le package iConsole, j'ai changé mes classes de gestion et la classe client, en utilisant la nouvelle classe ES au lieu de ma classe Tools (qui avait la méthode saisie() que pour les entiers).

iConsole

Class ES

Method Summary	
static void	**affiche**(java.lang.String msg) méthode d'affichage
static String	**saisie**(java.lang.String msg) méthode de saisie d'une chaîne de caractères après un message qui le demande, détermine si l'entier choisi est bien dans les bornes possible si n'est pas numérique l'exception InputMismatch est levée
static float	**saisie**(java.lang.String msg, float inf, float sup) méthode de saisie d'un float après un message qui le demande, détermine si l'entier choisi est bien dans les bornes possible si n'est pas numérique l'exception InputMismatch est levée
static int	**saisie**(java.lang.String msg, int inf, int sup) méthode de saisie d'un entier après un message qui le demande, détermine si l'entier choisi est bien dans les bornes possible si n'est pas numérique l'exception InputMismatch est levée

c. **Le scénario de la classe Client :**

Le scénario du client est le même que dans la série 2.3, sauf que pour les affichages et les saisie les gestionnaires passe par la classe ES.
La classe Tools (boite à outils) est chargé maintenant que de la validation qui est appelées par les trois gestionnaires.

d**. Les apports du cours :**

En cours, nous avons traité les entrées/sorties et les exceptions. On a créé une classe qui contient un Scanner et qui lit à chaque fois l'élément que l'on veut avoir comme entrée.

On a vu l'utilisation du même scanner pour toutes les méthodes de la classe ES et l'importance de vider ce scanner pendant la saisie et entre les saisies. Personnellement, j'ai ajouté un reset() du scanner au début de chaque méthode.

Par ailleurs, on peut, en traitant les exceptions, traiter une exception et la relever pour la signaler à la méthode qui l'appelle. J'ai laissé le traitement de l'exception dans la classe ES, car je trouvais inutile à ce moment-là de la relevée dans les classes de gestion. Peut-être ce serait plus adéquat de créer un gestionnaire d'exceptions.

e. **Conclusion :**

La classe ES est une classe fondamentale, qui prend le relais de la classe Tools que j'avais créé, en lui ajoutant différent type d'entrée. Elle me rappelait la classe TERMINAL que j'utilisais en NFA031, donc je suis allé vérifier la différence entre les deux classes. La classe TERMINAL lit toutes les entrées en String et elle les parse() en double, en boolean ou en int si elle a besoin. Je n'aime pas trop cette approche, je préfère levée l'exception et lire directement les entrées dans leurs type demandé. Donc je préfère notre classe ES. Ceci-dit si on ne connait pas le type de l'entrée voulu, utiliser un parse() sur l'entrée est plus à-propos. Je trouve cette série trop courte même si elle est une base importante est inévitable avant d'abordée la suite. J'aurais bien aimé plus de conseil concernant l'application des exceptions, par exemple ; si créer une nouvelle exception qui hérite de la classe exception ou utiliser les exceptions existantes est plus optimiser et les endroits les plus adaptés au traitement des exceptions.

NFA019 : Série 3.2- Morane Gruenpeter

a. Après avoir créé la classe ES dans un package iConsole, qui prend en charge les entrées/sorties en console, La série 3.2 consiste à reprendre la classe ES dans un package iPane, qui prend en charge les entrées/sorties en fenêtre graphique, pour une application plus communicative avec les utilisateurs. Cela est le début de l'interface graphique, avec la classe JOptionPane qui crée une fenêtre de dialogue facilement.

b. La seule classe que l'on avait créé pour cette série est la classe ES dans le package iPane, j'ai changé mes classes de gestion et la classe client, car j'ai changé ma méthode saisie(String titre, String msg, int inf, int sup), pour donner à chaque étape du programme un titre à la fenêtre de dialogue qui demande une saisie. Ceci me permet de savoir exactement où je me trouve et quel est le contexte de la demande faite à l'utilisateur. Aussi j'ai ajouté des closes try-catch dans les méthodes de gestion qui appelle la saisie d'une entrée avec la classe ES.

iPane

Class ES

Method Detail

affiche

public static void **affiche**(java.lang.String titre,
java.lang.String msg)

Méthode d'affichage d'un message dans une fenêtre de dialogue

Parameters:

msg – le message affiché à l'utilisateur dans la fenêtre de dialogue.

titre- - le title de la fenêtre

---

saisie

public static java.lang.String **saisie**(java.lang.String titre,
java.lang.String msg)

méthode de saisie d'une chaîne de caractères dans une fenêtre de dialogue qui le demande sur l'entrée on applique la méthode trim() qui enlève les espaces pour vérifier que l'entrée saisit n'est pas que des espaces. Si cette entrée est vide une fenêtre d'alerte est affichée et la saisie est redemandée. Sinon l'entrée est renvoyée.

Parameters:

titre – le titre de la fenêtre de dialogue.

msg – la demande à l'utilisateur dans la fenêtre de dialogue.

Returns: String **throws:** NullPointerException

---

saisie

public static float **saisie**(java.lang.String titre,
java.lang.String msg,
float inf,
float sup)

méthode de saisie d'un float dans une fenêtre de dialogue qui le demande l'entrée est lu en tant que String et avec la méthode Float.parse(entrée) on récupère l'entrée en type float. si cette entrée n'est pas numérique l'exception NumberFormatException est levée et traité dans la méthode en demandant une nouvelle entrée. si le float choisi est bien dans les bornes possible=> il est retourné sinon une fenêtre d'alerte est affichée et la saisie est redemandée.

Paramètres:

titre – le titre de la fenêtre de dialogue.

msg – la demande à l'utilisateur dans la fenêtre de dialogue.

inf – borne inférieur.

sup – borne supérieur.

Returns: float **throws:** NullPointerException

---

saisie

public static int **saisie**(java.lang.String titre,
java.lang.String msg,
int inf,
int sup)

méthode de saisie d'un entier dans une fenêtre de dialogue qui le demande l'entrée est lu en tant que String et avec la méthode Interger.parse(entrée) on récupère l'entier entrée. si cette entrée n'est pas numérique l'exception NumberFormatException est levée et traité dans la méthode en demandant une nouvelle entrée. Si l'entier choisi est bien dans les bornes possible=> il est retourné sinon une fenêtre d'alerte" est affichée et la saisie est redemandée.

Paramètres:

titre – le titre de la fenêtre de dialogue.

msg – la demande à l'utilisateur dans la fenêtre de dialogue.

inf – borne inférieur.

sup – borne supérieur.

Returns: int **throws:** NullPointerException

---

c. **Le scénario de la classe Client :**
Le scénario du client est le même que dans la série 2.3, sauf qu'elle ce produit dans des fenêtres qui pop sur l'écran.

d**. Les apports du cours :**

Cette semaine je n'ai pas pu venir en cours, donc j'écrirai mes apports du travail sur la série 3.2 sans les notes du cours et après avec ce que mes collègues m'ont raconté.
Premièrement, cette série est ma première rencontre avec la classe JOptionPane, même si j'ai déjà abordé le SWING. Je trouve cette classe efficace et facile à manipuler, or elle donne un résultat moyennement appétissant.

Par ailleurs, J'ai rencontré un vrai défi que je n'ai pas encore résolu, pour la fenêtre qui pop avec la méthode **showInputDialog** un bouton « Cancel » apparait et je n'arrive pas à prendre le contrôle de ce bouton pour sortir de la méthode en l'appuyant. En plus, pour les demandes de saisie simultanées, comme l'ajout d'un article, si j'annule la saisie du prix unitaire par exemple j'aurais un problème en créant l'article.

Après avoir vu avec mes collègues les notion abordées en cours, je comprends maintenant que le bouton « cancel » est accessible quand j'écris la close
____if(entrée==null) throw NullPointerException ____
Avec cela, la classe ES renvoie une exception si la saisie entrée est null, qui désigne le déclenchement du bouton 'annuler'. Je peux attraper cette exception dans les classes de gestion où je demande des saisies à l'utilisateur.
Au départ je me demandais quelle est la meilleure approche pour les demandes complexes à l'utilisateur. En ajoutant des clauses de try-catch pour chaque saisie, j'ai remarqué que je pouvais mettre les saisies complètes dans une clause de try-catch, et seulement si l'utilisateur a effectué toutes les étapes de la saisie complexe, celle-ci est prise en compte. Donc, si j'ai une exception j'affiche un message d'annulation et on retourne au menu précédent.

e. **Conclusion :**
La nouvelle classe ES m'a permis de connaître une manière facile et direct pour créer une interface graphique de base, je pense que c'est un bon début et une approche plus intuitive pour continuer. Je me souviens qu'en NFA035 j'étais bien perplexe devant tous les composants dans un JFrame et les différentes méthodes qui activaient les boutons et les lectures des entrées. En découvrant cette classe j'ai décidé d'appliquer la méthode **showInputDialog(null, msg, titre, int)** pour avoir un titre à chaque fenêtre de dialogue, je trouvais que sans le titre la navigation est plus difficile et parfois on se retrouve avec une question dont on ne sait pas comment on est arrivée là.