Programmation orientée objet
TP2 Exceptions

L3 MIAGE

14 et 21 septembre 2023

1 Quelques rappels

Note : les exercices commencent à la section 2. Nous vous recommandons chaudement de lire cette section avant de commencer les exercices.

Manipuler les entrées

En java, il est possible de récupérer les entrées saisies au clavier par l'intermédiaire de la classe *Scanner*. Voici quelques exemples :

```
import java.util.Scanner;

class Main {
   public static void main(String[] args) {
      Scanner entree = new Scanner(System.in);

      System.out.println("Saisir une chaine de caractere:");
      String s = entree.nextLine();
      System.out.println("Saisir un entier:");
      int a = entree.nextInt();
   }
}
```

Exception

Définir une exception :

```
class MonException extends Exception {

   // Le constructeur par defaut
   MonException() {
      super();
   }

   // Constructeur pour creer une exception avec un message
      personnalise
   MonException(String msg) {
      super(msg);
   }
```

```
    // D'autres constructeurs possibles, exemple:
    MonException(int id, String nom) {
        super("L'utilisateur " + nom " avec l'identifiant " + id +
            " n'est pas enregistre dans la BDD");
    }
}
```

Lancer une exception :

```
class Jukebox {
    /* ... */
    void lireMorceau(String titre) {
        if (rechercheMorceau(titre) == false) {
            throw new JukeboxException(titre); // L'exception
                JukeboxException est levee. Si celle-ci n'est pas
                attrapee, alors le programme s'arretera.
        }
    }
    /* ... */
}
```

Attraper une exception :

```
class Main {
    /* ... */
    public static void main(String[] args) {
        Jukebox j = new Jukebox();
        try {
            j.lireMorceau("Ce titre n'existe pas !");
            System.out.println("Ce message ne s'affichera que si
                l'exception JukeboxException n'est pas levee !");
        } catch (JukeboxException e) {
            System.out.println("Les instructions de ce bloc ne seront
                executees que si l'exception JukeboxException est levee
                !");
            e.printStackTrace();
            System.out.println(e.getMessage());
        } finally {
            // Le bloc finally est optionnel
            System.out.print("Les instructions de ce bloc seront toutes
                executees apres les instructions du try ou du catch.");
        }
    }
    /* ... */
}
```

Lire dans un fichier

En java, il existe de nombreux outils pour lire des fichiers. Une des solutions les plus simples consiste à utiliser la classe *BufferedReader* qui fait partie de la bibliothèque de base de java. Voici un lien vers sa documentation : docs.oracle.com.

Voici un exemple d'ouverture d'un tampon pour la lecture d'un fichier nommé *readme.txt*.

```
// Attention: il faut attraper l'exception !
BufferedReader fichierTampon = new BufferedReader(new
    FileReader("readme.txt"));
```

Attention : le constructeur de *BufferedReader* prend en paramètre un objet *FileReader* qui peut lever une exception de type *FileNotFoundException* si le fichier n'est pas présent dans le répertoire courant.

Une fois le tampon pour la lecture du fichier initialisé, vous pouvez lire celui-ci ligne à ligne avec la méthode *readLine*. Cette méthode retourne un objet *String* si la fin du fichier n'a pas été atteinte, et *null* sinon.

```
// Attention: il faut attraper l'exception !
String test = fichierTampon.readLine();
```

Attention : la méthode *readLine* peut lever une exception de type *IOException*. Une telle exception peut être levée si, par exemple, le tampon a déjà été fermé.

Afin d'éviter la corruption d'un fichier ou la fuite de mémoire, il est préférable de fermer le tampon quand la lecture du fichier est terminée. Pour cela, on utilise la méthode *close*.

```
// Attention: il faut attraper l'exception !
fichierTampon.close();
```

Attention : la méthode *close* peut lever une exception de type *IOException*. Il faut aussi vérifier que le fichier tampon n'est pas *null*. Cela peut être le cas si celui-ci est présent dans le bloc d'instruction *finally* après un *try-catch* de la création du *BufferedReader*.

2 Attraper les exceptions communes

Nous vous conseillons de faire les exercices de cette section dans une classe *Main* avec des méthodes statiques.

2.1 Tableau et indices

Déclarer une méthode statique *genererTab*. Celle-ci initialise un tableau de 10 entiers avec des valeurs quelconques (soit des constantes de votre choix ou générées aléatoirement avec *Random*). Ensuite, elle demande à l'utilisateur de saisir un entier i, puis elle affiche le contenu du tableau à la position i.

Question : si l'utilisateur saisit un nombre supérieur ou égale à 10, quelle exception est levée ?

Modifier votre code pour attraper l'exception avec un bloc *try-catch* si elle survient. Dans ce cas, on affichera le message suivant :

```
/* Msg: mauvaise pioche !
```

2.2 Référence null

Nous donnons le code suivant d'une classe *Message* (donc à copier dans un fichier *Message.java*) :

```
// Fichier Message.java
public class Message {
   private String msg;

   public void genererMsg(int n) {
      char[] tab = new char[n];
      for (int i = 0; i < n; i++) {
         tab[i] = (char) i;
      }
      this.msg = new String(tab);
   }

   /*
   * Retourne vrai si msg contient la chaine de caracteres s
   */
   public boolean contient(String s) {
      return msg.contains(s);
   }
}
```

Voici le code d'une méthode *main* qui instancie un objet de type *Message* (donc à copier dans un fichier *Main.java*) :

```
// Fichier Main.java
public class Main {
   public static void main(String[] args) {
      Message m = new Message();
      m.contient("toto");
   }
}
```

Question : que se passe-t-il à l'exécution du programme *main* ?
Proposer une solution afin de corriger ce problème sans modifier la méthode *main*.

3 Dates et exceptions

Voici la classe *Date* suivante :

```
public class Date {
   int jour;
   int mois;
   int annee;

   public Date(int jour, int mois, int annee) {
      this.jour = jour;
      this.mois = mois;
      this.annee = annee;
   }
}
```

3.1 Création d'une date valide

Modifier le constructeur de la classe *Date* afin que celui-ci lève une exception de type *DateException* si le jour ou le mois ne correspond pas à une date valide. On supposera pour cet exercice que tous les mois de l'année ont trente jours.

Voici le message attendu :

```
/* Msg: la date jj/mm/aaaa n'est pas valide
```

3.2 Attraper une exception

Ajouter les instructions suivantes à une fonction *main* puis apporter les corrections nécessaires afin que les exceptions levées soient correctement traitées.

```
Date d1 = new Date(33, 10, 1885);
Date d2 = new Date(9, 15, 2005);
Date d3 = new Date(66, 30, 1966);
```

3.3 Plusieurs exceptions

On souhaite à présent différencier ces deux types d'exceptions : celles liées à un numéro de jour incorrect et celles liées à un numéro de mois incorrect. Pour cela, on déclarera deux sous-classes de la classe *DateException* qu'on appellera *JourException* et *MoisException*. Il est donc nécessaire de mettre à jour les constructeurs de *DateException* et de modifier la réception des exceptions dans la fonction main.

Note : il est possible de faire des réceptions multiples avec plusieurs blocs de *catch* à la suite après un seul bloc *try*

3.4 31 jours dans le mois, c'est possible

On souhaite prendre en compte le nombre exact de jours dans chaque mois. Pour cela, on déclarera une sous-classe de la classe *JourException* que l'on appellera *JourDansMoisException*. On adaptera ensuite le code afin de gérer les exceptions suivantes :

- Si ni le jour ni le mois n'est correct, alors une exception *DateException* est levée ;
- Autrement, si le mois n'est pas valide, alors une exception *MoisException* est levée ;
- Autrement, si le jour n'est pas valide car il n'existe pas de 31 pour le mois saisie, alors une exception *JourDansMoisException* est levée ;
- Autrement, si le jour n'est pas valide, alors une exception *JourException* est levée.

Note : on ne prend pas en compte les années bissextiles. On supposera donc que le mois de février à seulement 28 jours.

4 Fichiers et exceptions

On s'intéresse dans cet exercice à la lecture des informations contenues dans un fichier et aux exceptions qui peuvent en résulter.
L'objectif est de réaliser un programme dont le fonctionnement est le suivant :

Étape 1 l'utilisateur saisit au clavier un nom de fichier. Tant que l'utilisateur ne saisit pas un nom de fichier valide, c.-à-d. présent dans le répertoire courant, alors le programme redemande la saisie d'un nom de fichier à l'utilisateur.

Étape 2 le programme tire un nombre au hasard entre 1 et 10. Ce nombre correspond à une ligne interdite.

Étape 3 le programme demande à l'utilisateur de saisir un entier qui correspond à la ligne que l'utilisateur souhaite lire. Trois configurations sont alors possibles :

- le programme affiche la ligne si elle existe et que ce n'est pas une ligne interdite ;
- le programme lève une exception de type *LigneInterditeException* si la ligne est interdite ;
- le programme lève une exception de type *LigneManquanteException* si la ligne n'existe pas dans le fichier.

Voici les différentes étapes à réaliser :

4.1 Exception ligne interdite

Définir une classe *LigneInterditeException* avec un constructeur recevant en paramètre un entier correspondant à la ligne interdite. Le message de l'exception devra être le suivant :

```
/* Msg: la ligne x est interdite en lecture !
```

4.2 Exception numéro de ligne

Définir une classe *LigneManquanteException* avec un constructeur recevant en paramètre un entier correspondant à la ligne qui n'existe pas dans le fichier. Le message de l'exception devra être le suivant :

```
/* Msg: la ligne x n'existe pas dans le fichier !
```

4.3 Classe gestion de fichier

Définir une classe *GestionFichier* qui sert à initialiser des objets *GestionFichier* avec comme attribut un nom de fichier présent dans le répertoire courant. Pour cela, définir un constructeur qui demande à l'utilisateur de saisir au clavier un nom de fichier. Tant que l'utilisateur ne saisit pas un nom de fichier valide, c.-à-d. présent dans le répertoire courant, alors le programme redemande la saisie d'un nom de fichier à l'utilisateur.

Note : l'instruction suivante lève une exception de type *FileNotFoundException* si le fichier *nomFichier.txt* n'est pas trouvé dans le répertoire courant :

```
FileReader f = new FileReader("nomFichier.txt");
```

Bonus : si l'utilisateur saisit le message *STOP*, alors le programme se termine. Pour cela, utiliser l'instruction suivante :

```
System.exit(0);
```

4.4 Générer un nombre aléatoirement

Une fois que l'utilisateur a saisi un nom de fichier correct, le programme tire un entier entre 1 et 10 aléatoirement. Celui-ci est enregistré dans un attribut de l'objet. Pour tirer un entier aléatoirement, utiliser les instructions suivantes :

```
import java.util.Random;
/* ... */
Random random = new Random();
int alea = random.nextInt(10)+1;
```

4.5 Lecture de la ligne

Définir une méthode *lire* dans la classe *GestionFichier*. Celle-ci demande à l'utilisateur de saisir un entier correspondant à la ligne que l'utilisateur souhaite lire. Ensuite, trois événements peuvent survenir :

- la méthode affiche la ligne si elle existe et que ce n'est pas une ligne interdite. Elle retourne aussi le contenu de cette ligne ;
- la méthode lève une exception de type *LigneInterditeException* si la ligne est interdite ;
- la méthode lève une exception de type *LigneManquanteException* si la ligne n'existe pas dans le fichier.

4.6 Tester avec une méthode principale

Définir une méthode *main* et instancier un objet de type *GestionFichier*. Appeler ensuite la méthode *lire* sur cet objet et gérer les exceptions qui pourraient survenir avec un bloc *try-catch*. Nous rappelons qu'il est possible d'enchaîner plusieurs *catch* après un *try*.

4.7 Bonus : fermer le fichier

Fermer le fichier après sa lecture par la méthode *lire*. Pour cela, il est possible d'utiliser le bloc *finally* après un *try-catch*. L'idéal serait de fermer le fichier en dehors de la méthode *main*.